Convolutional Nueral Network

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

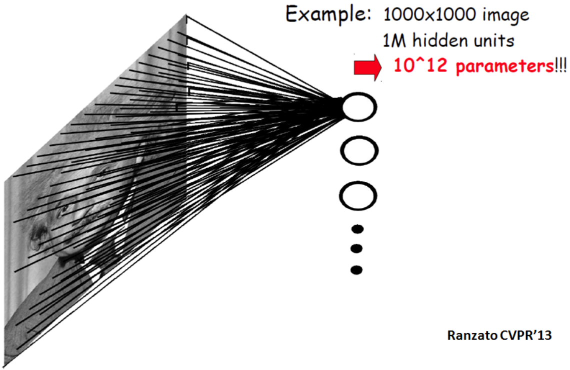
January 28, 2019

Convolutional neural network

- Specially designed for data with grid-like structures (LeCun et al. 98)
  - 1D grid: sequential data
  - 2D grid: image
  - 3D grid: video, 3D image volume
- Beat all the existing computer vision technologies on object recognition on ImageNet challenge with a large margin in 2012

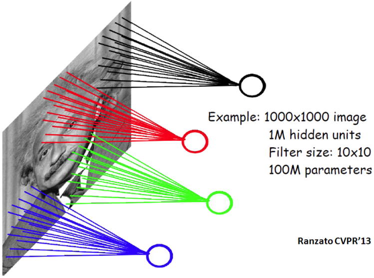
Problems of fully connected neural networks

- Every output unit interacts with every input unit
- The number of weights grows largely with the size of the input image
- Pixels in distance are less correlated



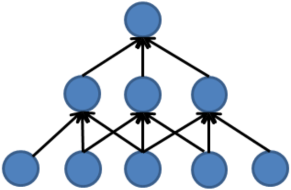
Locally connected neural networks

- Sparse connectivity: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
- It is inspired by biological systems, where a cell is sensitive to a small sub-region of the input space, called a receptive field. Many cells are tiled to cover the entire visual field.
- The design of such sparse connectivity is based on domain knowledge. (Can we apply CNN in frequency domain?)



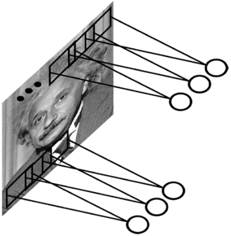
Locally connected neural networks

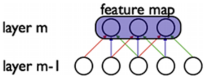
- The learned filter is a spatially local pattern
- A hidden node at a higher layer has a larger receptive field in the input
- Stacking many such layers leads to“filters”(not anymore linear) which become increasingly “global”



Shared weights

- Translation invariance: capture statistics in local patches and they are independent of locations
  - Similar edges may appear at different locations
- Hidden nodes at different locations share the same weights. It greatly reduces the number of parameters to learn
- In some applications (especially images with regular structures), we may only locally share weights or not share weights at top layers

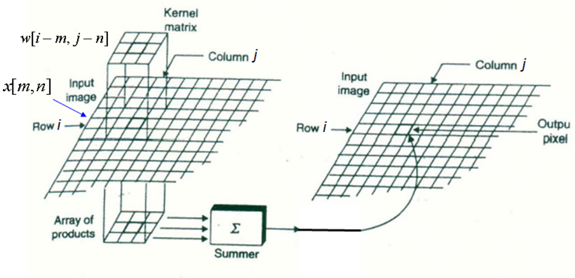




Convolution

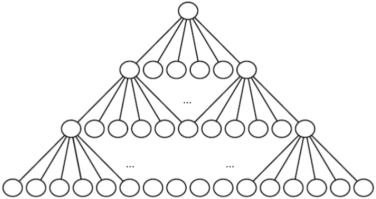
- Computing the responses at hidden nodes is equivalent to convoluting the input image **x** with a learned filter **w**
- After convolution, a filter map *net* is generated at the hidden layer
- Parameter sharing causes the layer to have *equivariance* to translation. A function $f(x)$ is equivalent to a function g if $f(g(x)) = g(f(x))$
- Is convolution equivariant to changes in the scale or rotation?

$$net[i,j] = (x*w)[i,j] = \sum_m \sum_n x[m,n]w[i-m,j-n]$$



Zero-padding in convolutional neural network

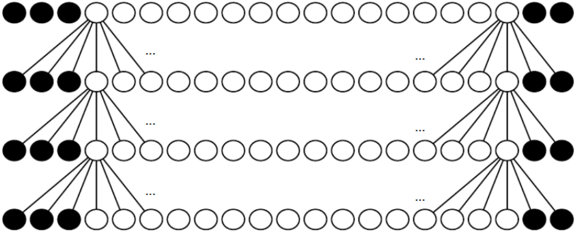
- The valid feature map is smaller than the input after convolution
- Implementation of neural networks needs to zero-pad the input $\mathbf{x}$ to make it wider
- Without zero-padding, the width of the representation shrinks by the filter width - 1 at each layer
- To avoid shrinking the spatial extent of the network rapidly, small filters have to be used



(Bengio et al. Deep Learning 2014)

Zero-padding in convolutional neural network

- By zero-padding in each layer, we prevent the representation from shrinking with depth. It allows us to make an arbitrarily deep convolutional network



(Bengio et al. Deep Learning 2014)

Downsampled convolutional layer

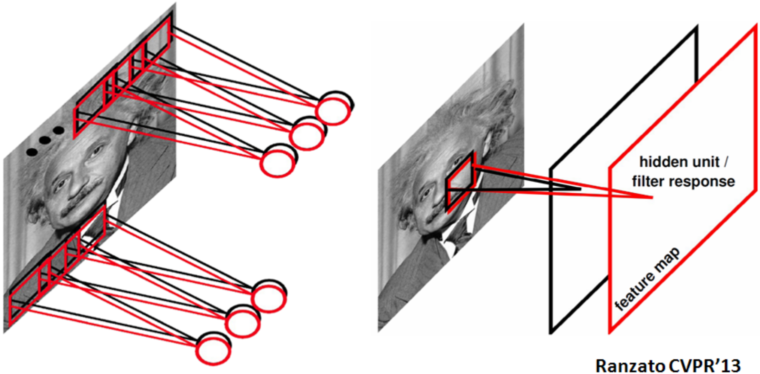
- To reduce computational cost, we may want to skip some positions of the filter and sample only every s pixels in each direction. A downsampled convolution function is defined as

$$net[i,j] = (\mathbf{x} * \mathbf{w})[i \times s, j \times s]$$

- s is referred as the *stride* of this downsampled convolution

Multiple filters

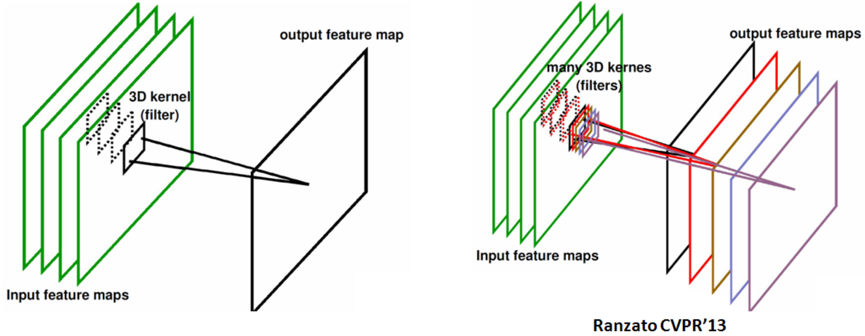
- Multiple filters generate multiple feature maps
- Detect the spatial distributions of multiple visual patterns



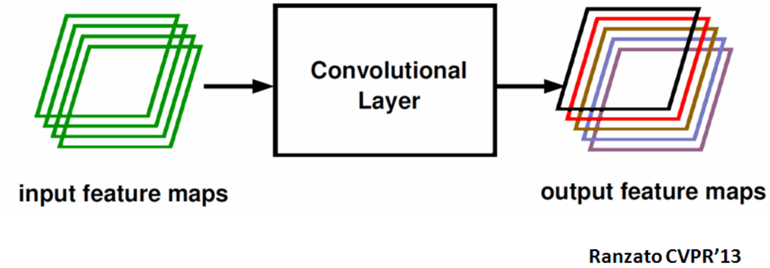
Ranzato CVPR'13

3D filtering when input has multiple feature maps

$$net = \sum_{k=1}^{K} \mathbf{x}^k * \mathbf{w}^k$$



Convolutional layer



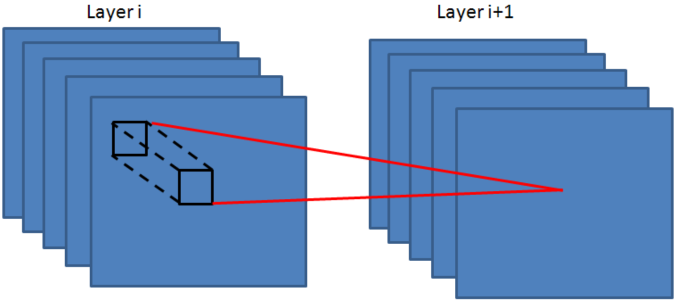
Nonlinear activation function

- tanh()
- Rectified linear unit

Local contrast normalization

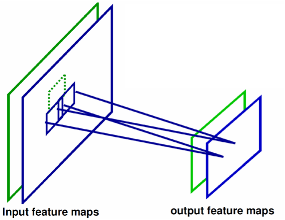
- Normalization can be done within a neighborhood along both spatial and feature dimensions

$$h_{i+1,x,y,k} = \frac{h_{i,x,y,k} - m_{i,N(x,y,k)}}{\sigma_{i,N(x,y,k)}}$$



Pooling

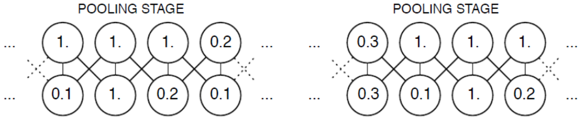
- Max-pooling partitions the input image into a set of rectangles, and for each sub-region, outputs the maximum value
- Non-linear down-sampling
- The number of output maps is the same as the number of input maps, but the resolution is reduced
- Reduce the computational complexity for upper layers and provide a form of translation invariance
- Average pooling can also be used



Ranzato CVPR'13

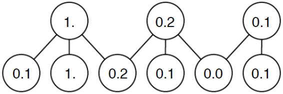
Pooling

- Pooling without downsampling (stride $s = 1$)
- Invariance vs. information loss (even if the resolution is not reduced)
- Pooling is useful if we care more about whether some feature is present than exactly there it is. It depends on applications.



(Bengio et al. Deep Learning 2014)

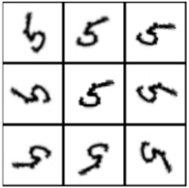
- Pooling with downsampling (commonly used)
- Improve computation efficiency



(Bengio et al. Deep Learning 2014)

Possible extension of pooling

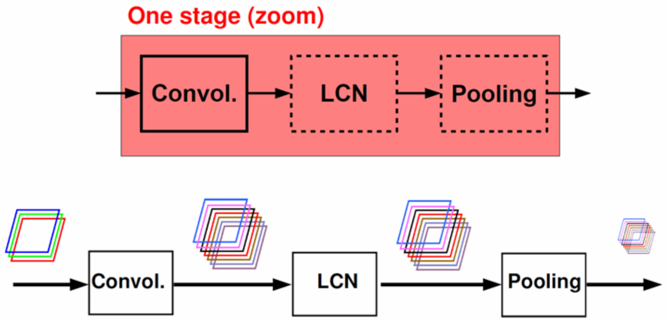
- If we pool over the outputs of separately parameterized convolutions, the features can learn which transformations to become invariant to
- How to achieve scaling invariance?



(Bengio et al. Deep Learning 2014)
Example of learned invariances: If each of these filters drive units that appear in the same max-pooling region, then the pooling unit will detect "5"s in any rotation. By learning to have each filter be a different rotation of the "5" template, this pooling unit has learned to be invariant to rotation. This is in contrast to translation invariance, which is usually achieved by hard-coding the net to pool over shifted versions of a single learned filter.

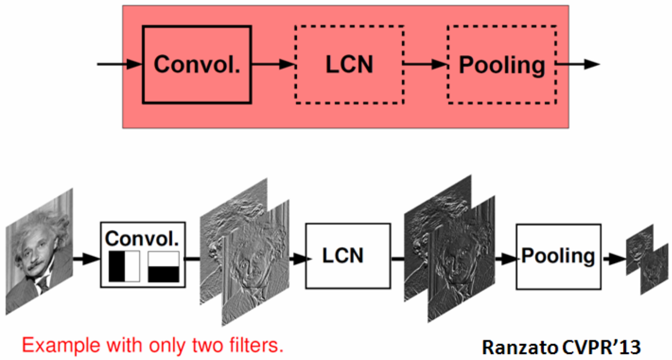
Typical architecture of CNN

- Convolutional layer increases the number of feature maps
- Pooling layer decreases spatial resolution
- LCN and pooling are optional at each stage

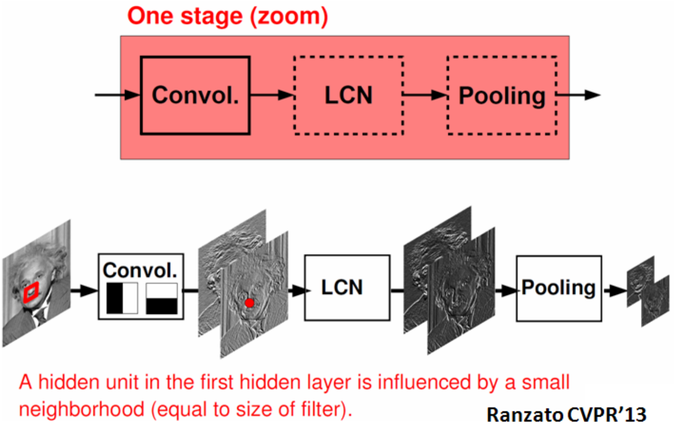


Ranzato CVPR'13

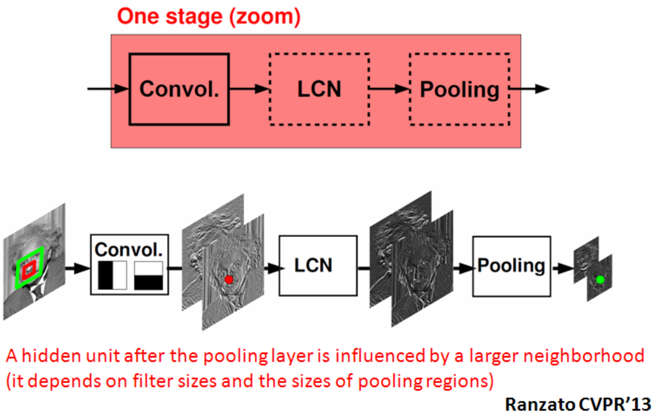
Typical architecture of CNN



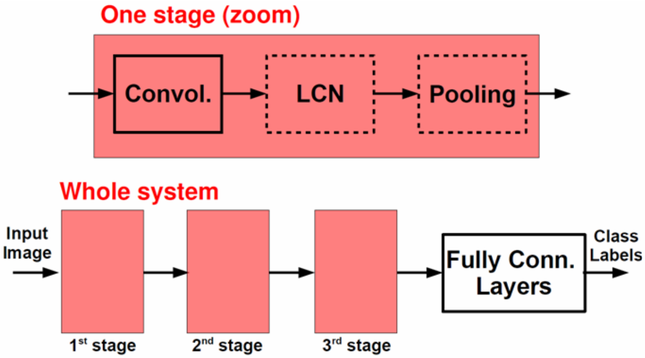
Typical architecture of CNN



Typical architecture of CNN



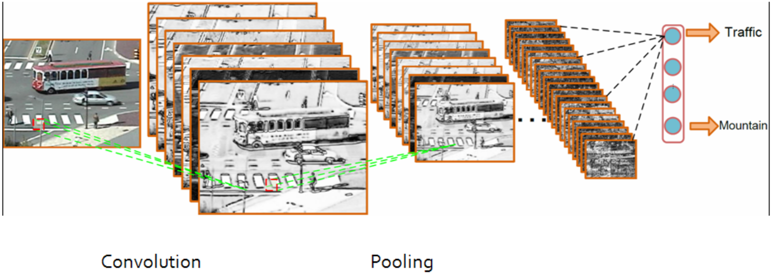
Typical architecture of CNN



After a few stages, residual spatial resolution is very small.
We have learned a descriptor for the whole image.

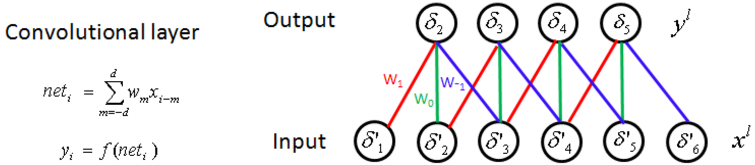
Ranzato CVPR'13

Typical architecture of CNN



BP on CNN

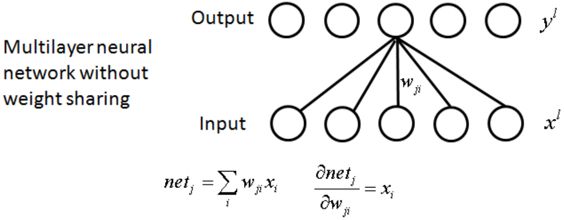
- Calculate sensitivity (back propagate errors) $\delta = -\frac{\partial J}{\partial net}$ and update weights in the convolutional layer and pooling layer
- Calculating sensitivity in the convolutional layer is the same as multilayer neural network



CNN has multiple convolutional layers. Each convolutional layer l has an input feature map (or image) $\mathbf{x}^l$ and also an output feature map $\mathbf{y}^l$. The sizes (n_x^l and n_y^l) of the input and output feature maps, and the filter size d^l are different for different convolutional layers. Each convultional layers has multiple filters, input feature maps and output feature maps. To simplify the notation, we skip the index (l) of the convolutional layer, and assume only one filter, one input feature map and one output feature map.

Calculate $\frac{\partial net}{\partial w}$ in the convulutional layer

- It is different from neural networks without weight sharing, where each weight W_{ij} is only related to one input node and one output node



- Taking 1D data as example, in CNN, assume the input layer $\mathbf{x} = [x_0, \ldots, x_{n_x-1}]$ is of size $n_x$and the filter $\mathbf{w} = [w_{-d}, \ldots, w_d]$ is of size $2 \times d + 1$. With weight sharing, each weight in the related with multiple input and output nodes

$$net_j = \sum_{m=-d}^{d} w_m x_{j-m}$$

Update filters in the convolutional layer

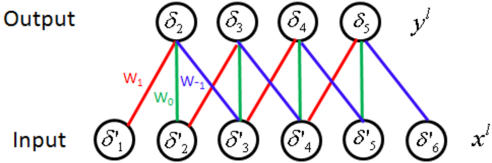
$$\frac{\partial J}{\partial w_m} = \sum_j \frac{\partial J}{\partial net_j} \frac{\partial net_j}{\partial w_m} = -\sum \delta_j x_{j-m}$$

- The gradient can be calculated from the correlation between the sensitivity map and the input feature map

Convolutional layer

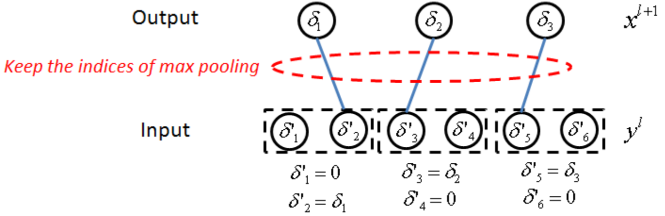
$$net_i = \sum_{m=-d}^{d} w_m x_{i-m}$$

$$y_i = f(net_i)$$



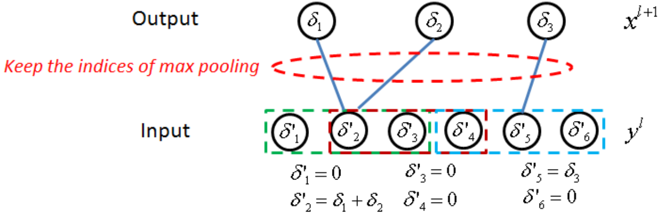
Calculate sensitivities in the pooling layer

- The input of a pooling layer l is the output feature map y^l of the previous convolutional layer. The output x^{l+1} of the pooling layer is the input of the next convolutional layer $l+1$
- For max pooling, the sensitivity is propagated according to the corresponding indices built during max operation. If max pooling regions are nonoverlapped,



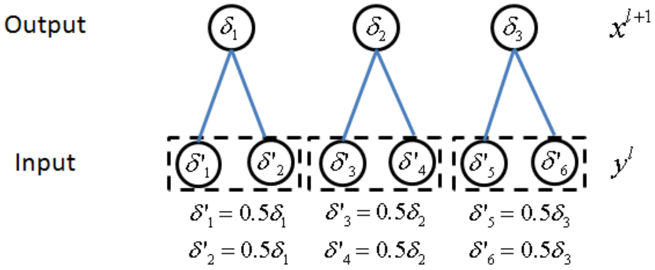
Calculate sensitivities in the pooling layer

- If pooling regions are overlapped and one node in the input layer corresponds to multiple nodes in the output layer, the sensitivities are added



Calculate sensitivities in the pooling layer

- Average pooling



- What if average pooling and pooling regions are overlapped?
- There is no weight to be updated in the pooling layer

CNN for object recognition on ImageNet challenge

- Krizhevsky, Sutskever, and Hinton, NIPS 2012
- Trained on one million images of 1000 categories collected from the web with two GPU. 2GB RAM on each GPU. 5GB of system memory
- Training lasts for one week
- Google and Baidu announced their new visual search engines with the same technology six months after that
- Google observed that the accuracy of their visual search engine was doubled

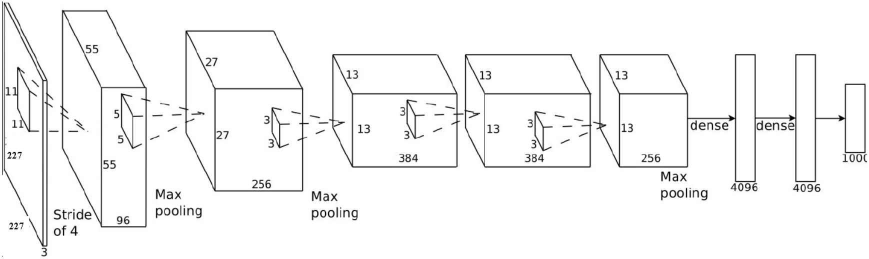
Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

ImageNet


1000 object classes that we recognize
poster created by Fengjun Lv using VIPBase
images courtesy of ImageNet (http://www.image-net.org/challenges/LSVRC/2010/index)

Model architecture-AlexNet Krizhevsky 2012

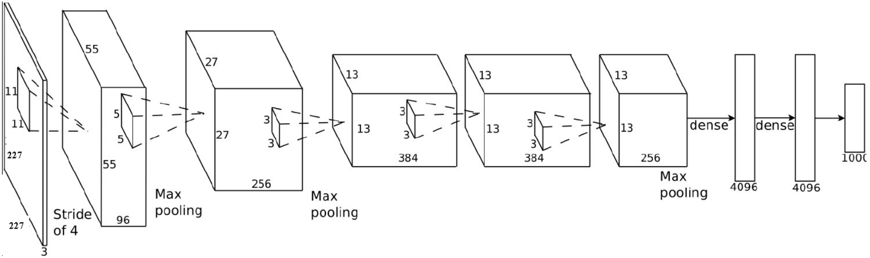
- 5 convolutional layers and 2 fully connected layers for learning features.
- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000
- 650000 neurons, 60000000 parameters, and 630000000 connections



(Krizhevsky NIPS 2014)

Model architecture-AlexNet Krizhevsky 2012

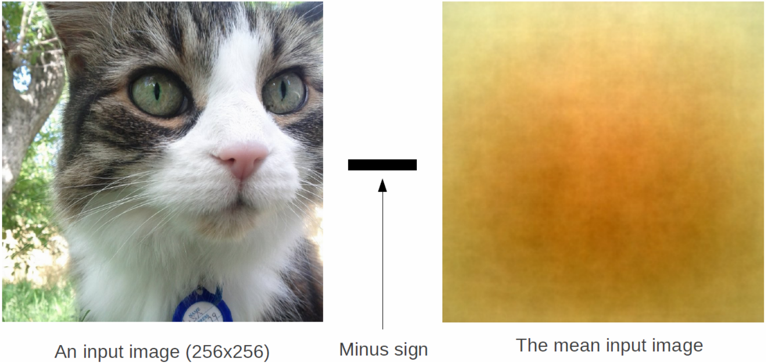
- The first time deep model is shown to be effective on large scale computer vision task.
- The first time a very large scale deep model is adopted.
- GPU is shown to be very effective on this large deep model.



(Krizhevsky NIPS 2014)

Technical details

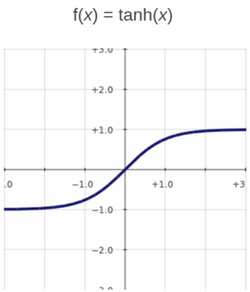
- Normalize the input by subtracting the mean image on the training set



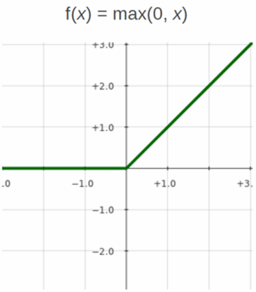
(Krizhevsky NIPS 2014)

Technical details

- Choice of activation function



Very bad (slow to train)



Very good (quick to train)

(Krizhevsky NIPS 2014)

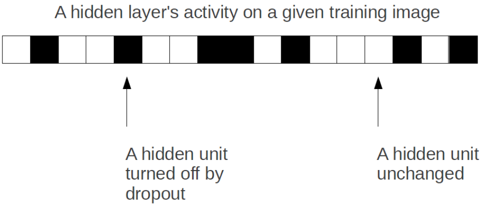
Technical details

- Data augmentation
  - The neural net has 60M real-valued parameters and 650,000 neurons
  - It overfits a lot. 224×224 image regions are randomly extracted from 256 images, and also their horizontal reflections

(Krizhevsky NIPS 2014)

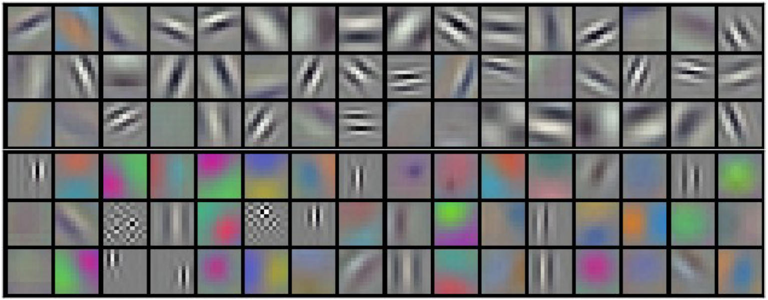
Technical details

- Dropout
  - Independently set each hidden unit activity to zero with 0.5 probability
  - Do this in the two globally-connected hidden layers at the net's output



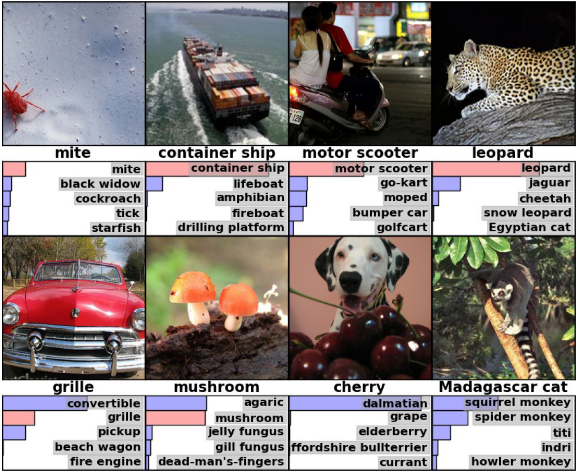
(Krizhevsky NIPS 2014)

96 learned low-level filters



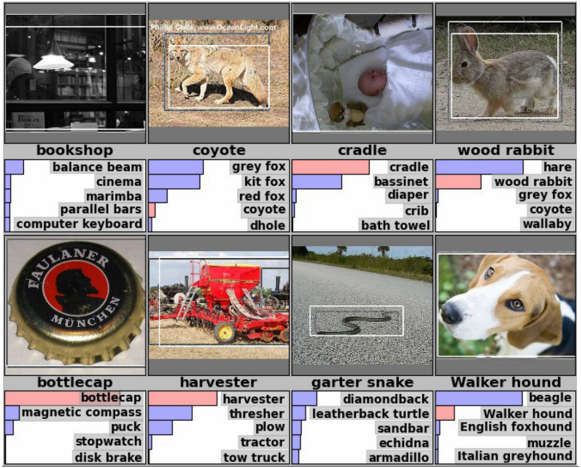
(Krizhevsky NIPS 2014)

Classification result



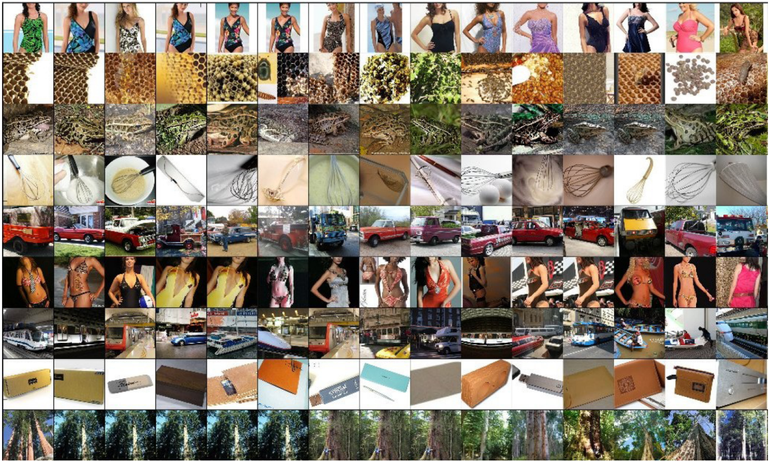
(Krizhevsky NIPS 2014)

Detection result



(Krizhevsky NIPS 2014)

Top hidden layer can be used as feature for retrieval



(Krizhevsky NIPS 2014)

Reading materials

- Y. Bengio, I. J. GoodFellow, and A. Courville, “Convolutional Networks,” Chapter 11 in “Deep Learning”, Book in preparation for MIT Press, 2014.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner,“Gradient-based learning applied to document recognition,” Proc. of the IEEE, 1998.
- J. Bouvrie, “Notes on Convolutional Neural Networks,” 2006.
- A. Krizhevsky, L. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Proc. NIPS, 2012.
- M. Ranzato, “Neural Networks,” tutorial at CVPR 2013.