Convolutional Nueral Network

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

February 2, 2015

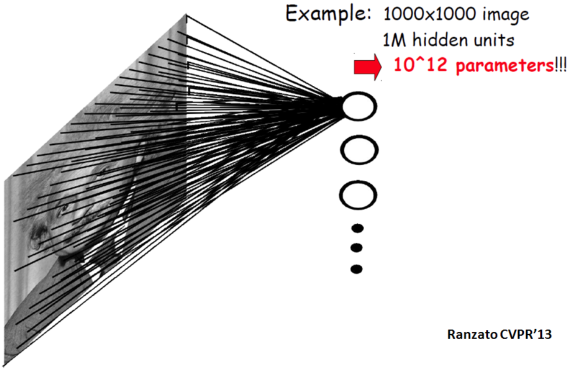
Outline

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

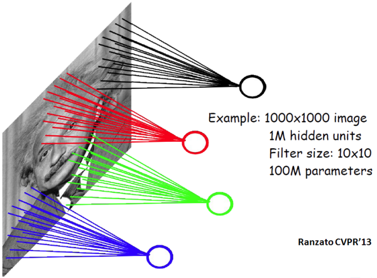
- Every output unit interacts with every input unit
- The number of weights grows largely with the size of the input image
- Pixels in distance are less correlated



Ranzato CVPR'13

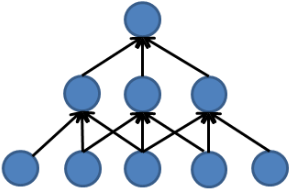
Locally connected neural networks

- 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?)



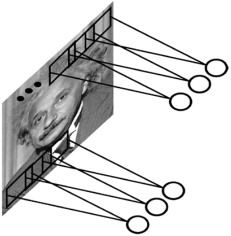
Locally connected neural networks

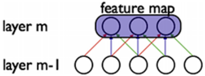
- The learned filter is a spatially local pattern
- A hidden node at a higher layer has a larger receptive field in the input
- Stacking many such layers leads to“filters”(not anymore linear) which become increasingly “global”



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



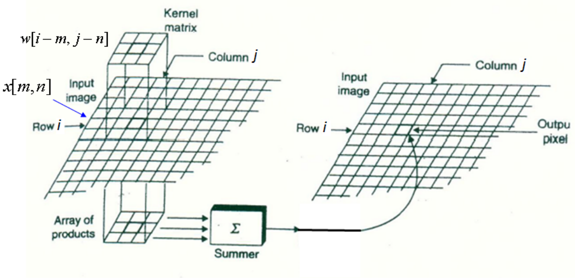


Weights with the same color have identical values

Convolution

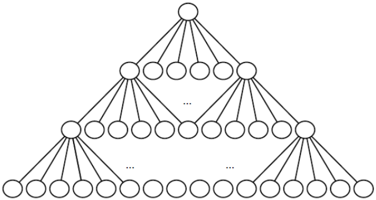
- Computing the responses at hidden nodes is equivalent to convoluting the input image **x** with a learned filter **w**
- After convolution, a filter map *net* is generated at the hidden layer
- Parameter sharing causes the layer to have *equivariance* to translation. A function $f(x)$ is equivalent 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 (optional)

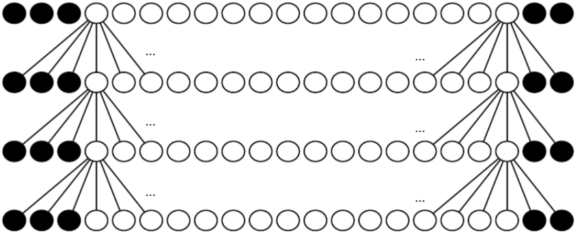
- The valid feature map is smaller than the input after convolution
- Implementation of neural networks needs to zero-pad the input **x** to make it wider
- Without zero-padding, the width of the representation shrinks by the filter width - 1 at each layer
- To avoid shrinking the spatial extent of the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



(Bengio et al. Deep Learning 2014)

Zero-padding in convolutional neural network (optional)

- 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 (optional)

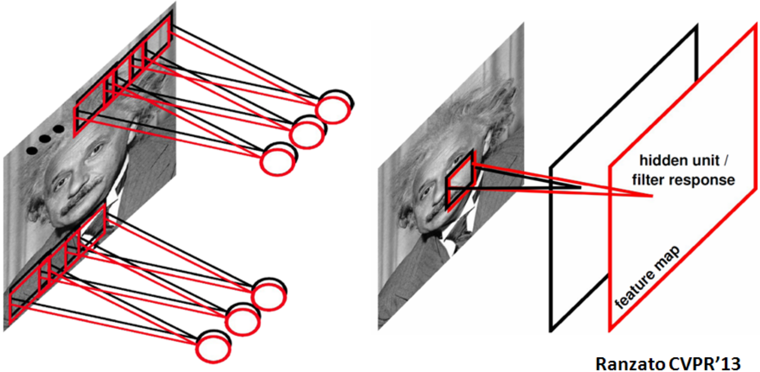
- To reduce computational cost, we may want to skip some positions of the filter and sample only every s pixels in each direction. A downsampled convolution function is defined as

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

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

Multiple filters

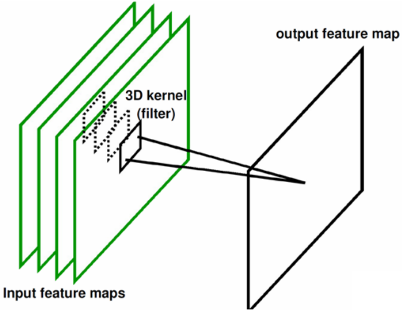
- Multiple filters generate multiple feature maps
- Detect the spatial distributions of multiple visual patterns

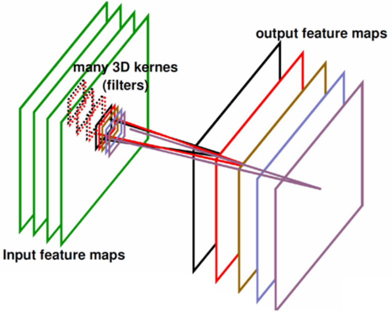


Ranzato CVPR'13

3D filtering when input has multiple feature maps

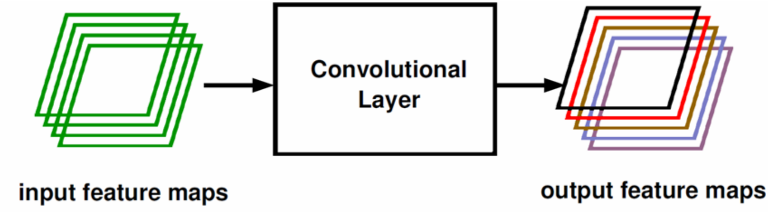
$$net = \sum_{k=1}^{K} \mathbf{x}^k * \mathbf{w}^k$$





Ranzato CVPR'13

Convolutional layer



Ranzato CVPR'13

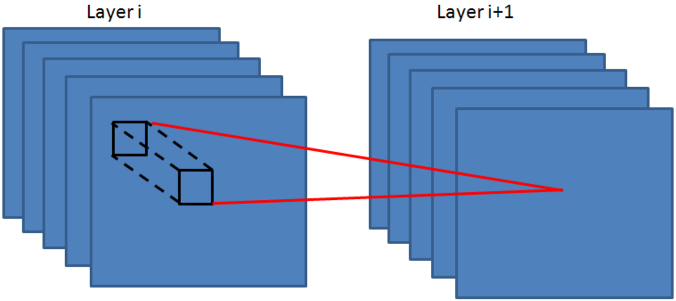
Nonlinear activation function

- tanh()
- Rectified linear unit

Local contrast normalization

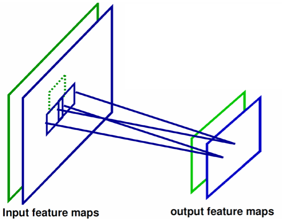
- Normalization can be done within a neighborhood along both spatial and feature dimensions

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



Pooling

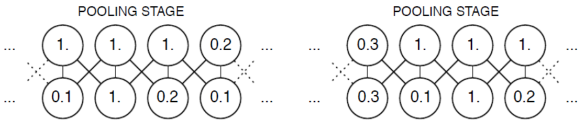
- Max-pooling partitions the input image into a set of rectangles, and for each sub-region, outputs the maximum value
- Non-linear down-sampling
- The number of output maps is the same as the number of input maps, but the resolution is reduced
- Reduce the computational complexity for upper layers and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



Ranzato CVPR'13

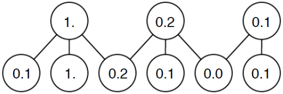
Pooling

- 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)

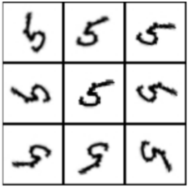
- Pooling with downsampling (commonly used)
- Improve computation efficiency



(Bengio et al. Deep Learning 2014)

Possible extension of pooling

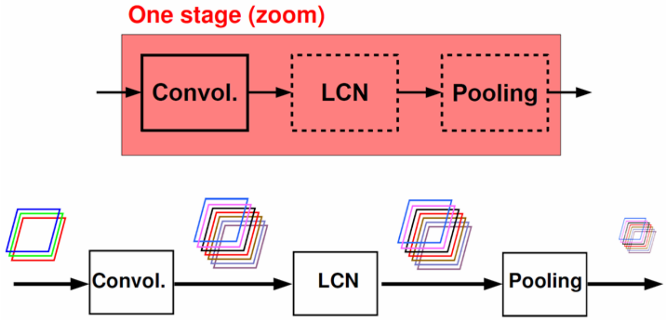
- If we pool over the outputs of separately parameterized convolutions, the features can learn which transformations to become invariant to
- How to achieve scaling invariance?



(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.

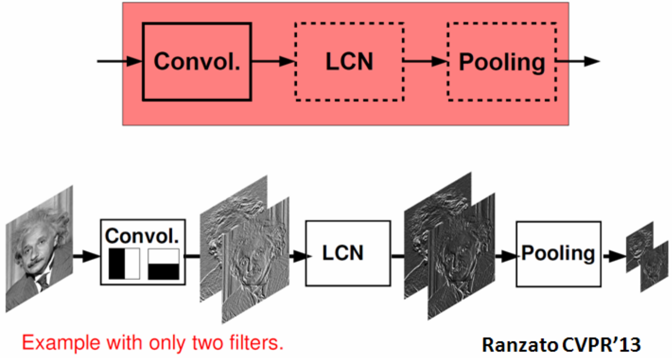
Typical architecture of CNN

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

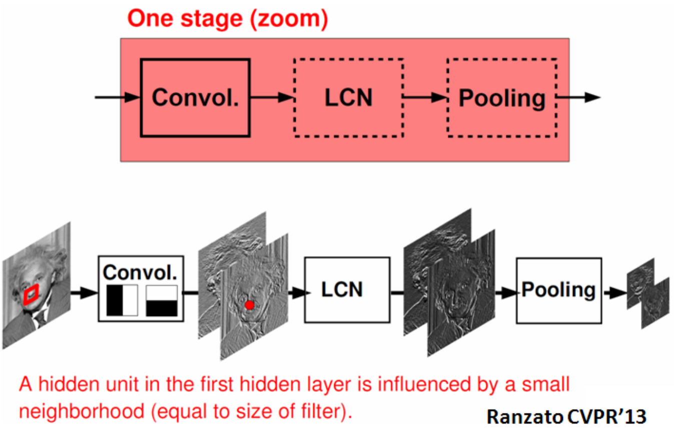


Ranzato CVPR'13

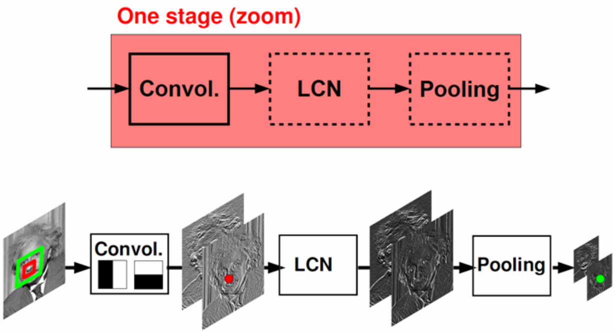
Typical architecture of CNN



Typical architecture of CNN



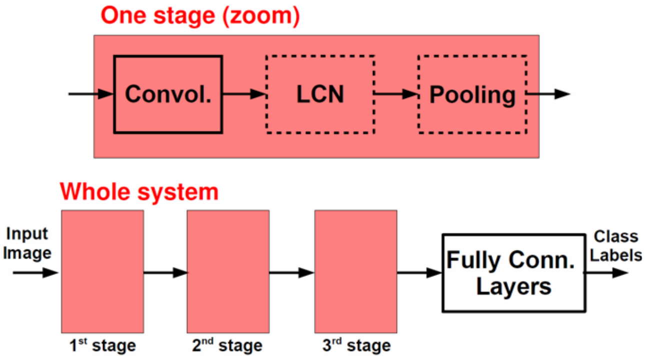
Typical architecture of CNN



A hidden unit after the pooling layer is influenced by a larger neighborhood (it depends on filter sizes and the sizes of pooling regions)

Ranzato CVPR'13

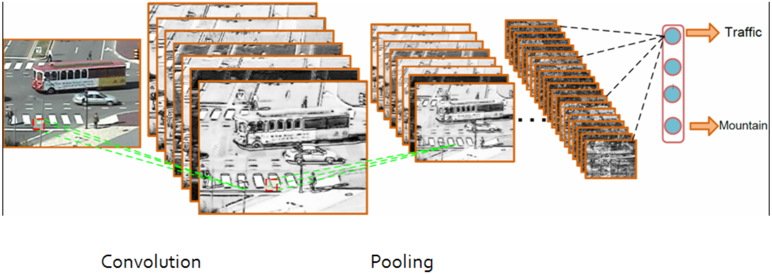
Typical architecture of CNN



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



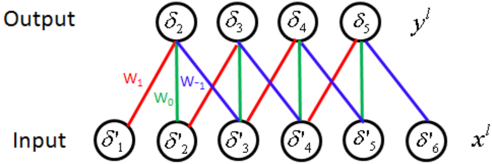
BP on CNN

- 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

Convolutional layer

$$net_i = \sum_{i=0}^{n_x-1} x_i w_{i-m} = \sum_{m=-d}^{d} w_m x_{i-m}$$

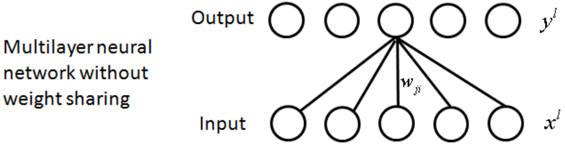
$$y_i = f(net_i)$$



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 convolutional 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



$$net_j = \sum_i w_{ji} x_i \qquad \frac{\partial net_j}{\partial w_{ji}} = x_i$$

- 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

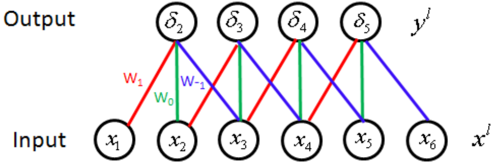
$$\frac{\partial J}{\partial w_m} = \sum_j \frac{\partial J}{\partial net_j}\frac{\partial net_j}{\partial w_m} = -\sum \delta_j x_{j-m}$$

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

Convolutional layer

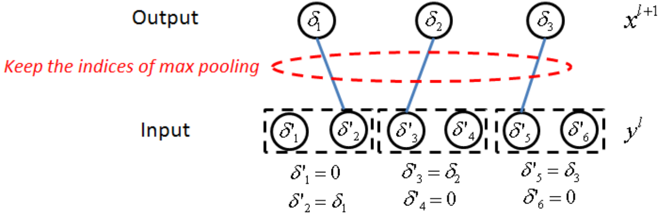
$$net_i = \sum_{i=0}^{n_x-1} x_i w_{i-m} = \sum_{m=-d}^{d} w_m x_{i-m}$$

$$y_i = f(net_i)$$



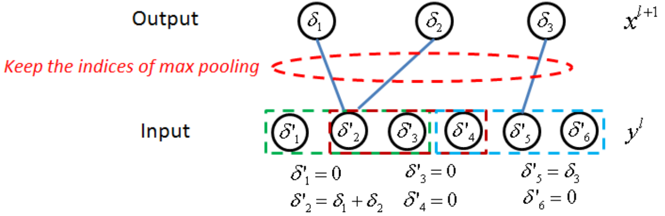
Calculate sensitivities in the pooling layer

- 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,



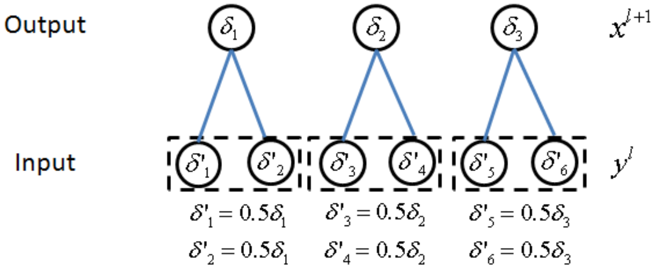
Calculate sensitivities in the pooling layer

- 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

Different CNN structures for image classification

- AlexNet
- Clarifai
- Overfeat
- VGG
- DeepImage of Baidu
- Network-in-network
- GoogLeNet

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

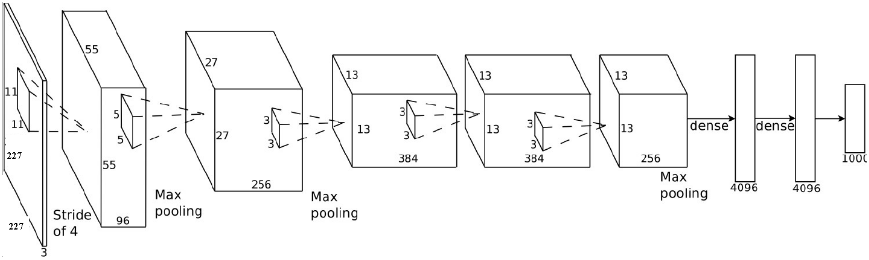
Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

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

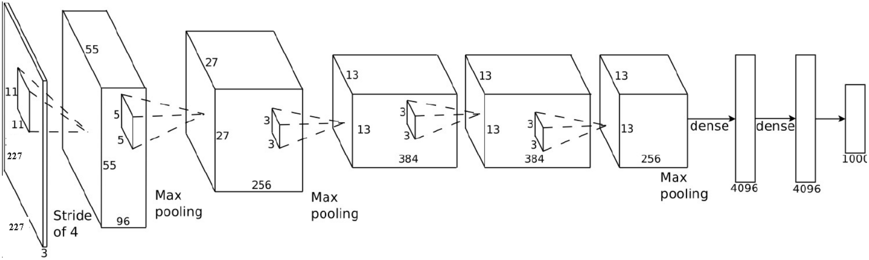
- 5 convolutional layers and 2 fully connected layers for learning features.
- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000
- 650000 neurons, 60000000 parameters, and 630000000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



(Krizhevsky NIPS 2014)

Model architecture-AlexNet Krizhevsky 2012

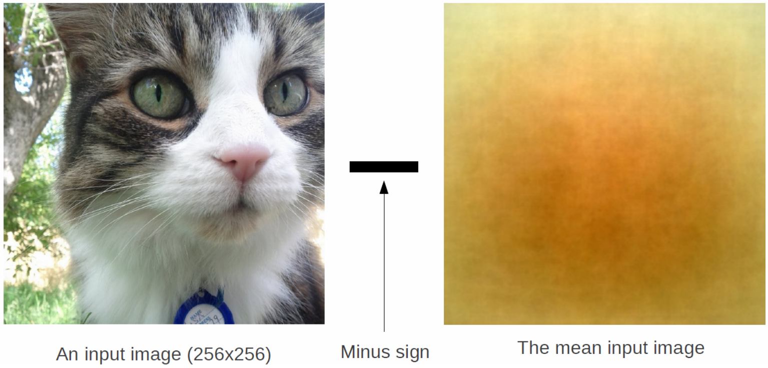
- The first time deep model is shown to be effective on large scale computer vision task.
- The first time a very large scale deep model is adopted.
- GPU is shown to be very effective on this large deep model.



(Krizhevsky NIPS 2014)

Technical details

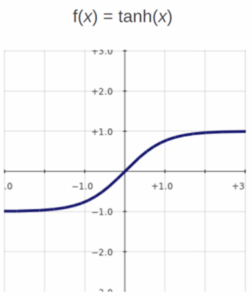
- Normalize the input by subtracting the mean image on the training set



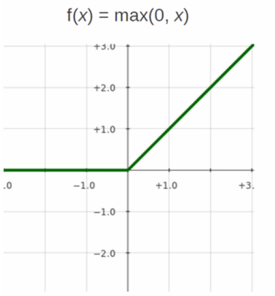
(Krizhevsky NIPS 2014)

Technical details

- Choice of activation function



Very bad (slow to train)



Very good (quick to train)

(Krizhevsky NIPS 2014)

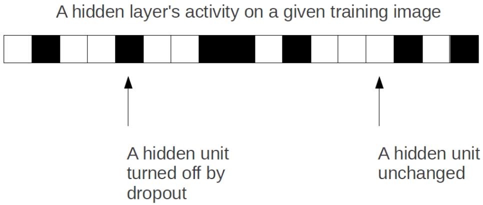
Technical details

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

(Krizhevsky NIPS 2014)

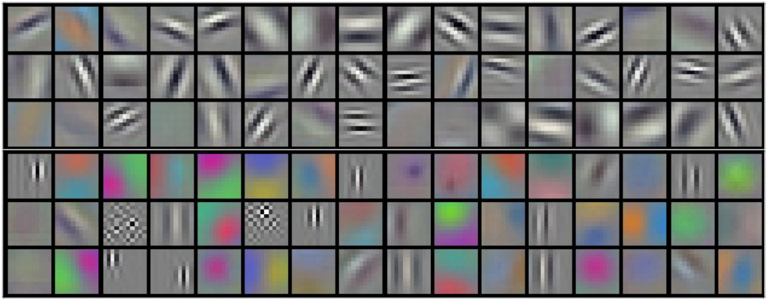
Technical details

- 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



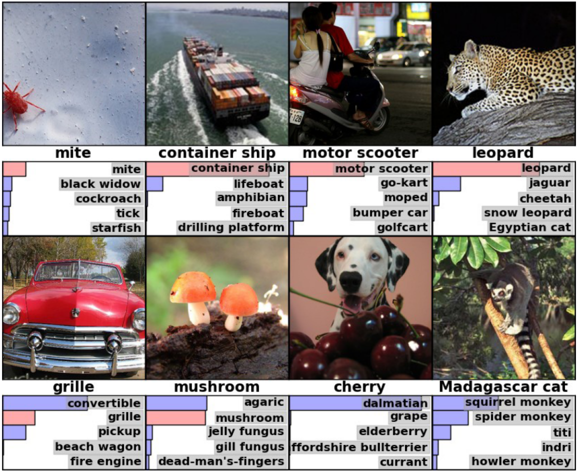
(Krizhevsky NIPS 2014)

96 learned low-level filters



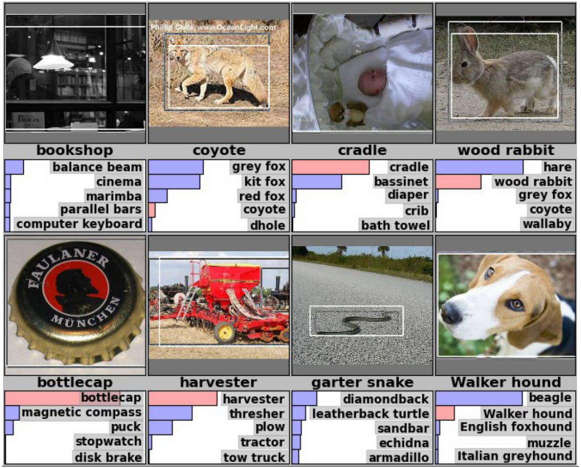
(Krizhevsky NIPS 2014)

Classification result



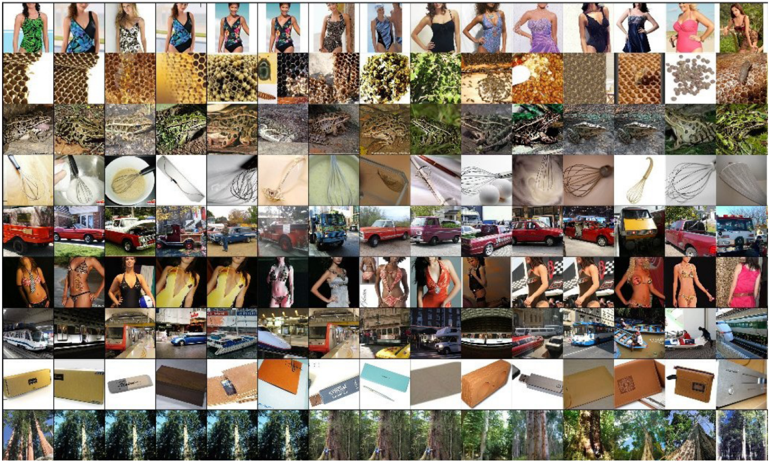
(Krizhevsky NIPS 2014)

Detection result



(Krizhevsky NIPS 2014)

Top hidden layer can be used as feature for retrieval



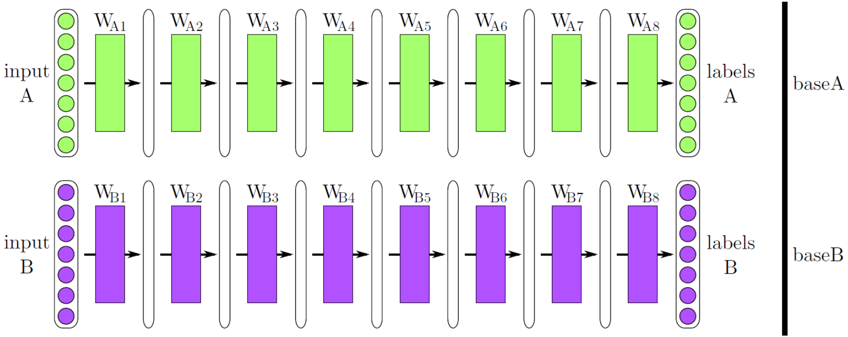
(Krizhevsky NIPS 2014)

How transferable are features in CNN networks?

- (Yosinski et al. NIPS'14) investigate transferability of features by CNNs
- The transferability of features by CNN is affected by
  - Higher layer neurons are more specific to original tasks
  - Layers within a CNN network might be fragilely co-adapted
- Initializing with transferred features can improve generalization after substantial fine-tuning on a new task

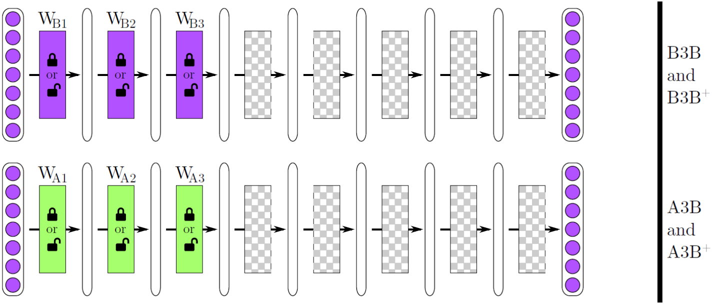
Base tasks

- ImageNet are divied into two groups of 500 classes, A and B
- Two 8-layer AlexNets, baseA and baseB, are trained on the two groups, respectively



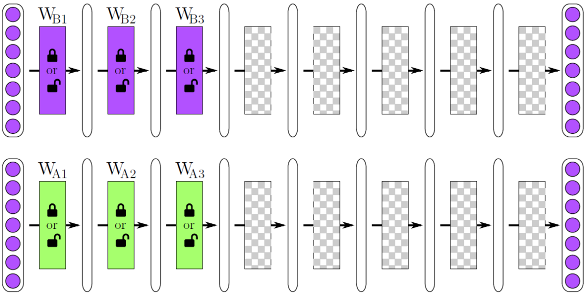
Transfer and selffer networks

- A *selffer* network BnB: the first n layers are copied from baseB and frozen. The other higher layers are initialized randomly and trained on dataset B. This is the control for transfer network
- A *transfer* network AnB: the first n layers are copied from baseA and frozen. The other higher layers are initialized randomly and trained toward dataset B



Transfer and selffer networks (cont'd)

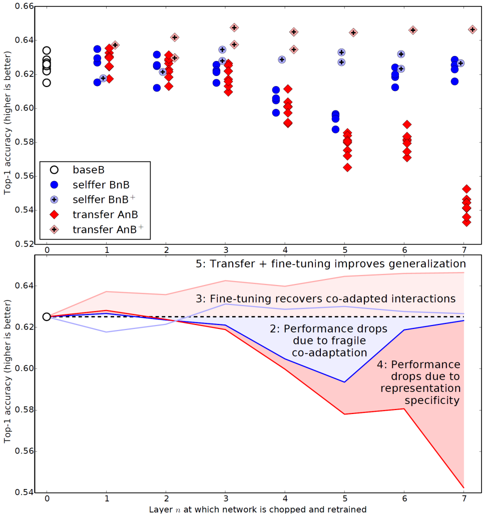
- A *selffer* network BnB+: just like BnB, but where all layers learn
- A *transfer* network AnB+: just like AnB, but where all layers learn



B3B and $B3B^+$

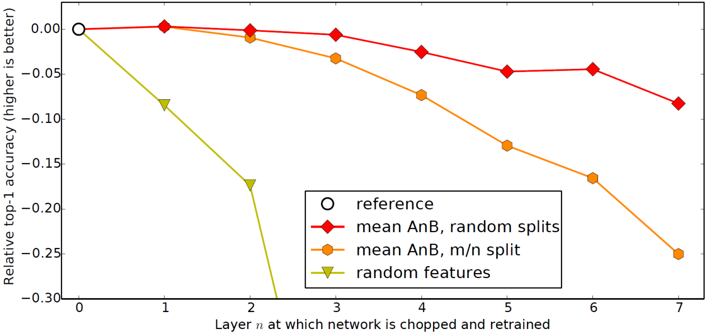
A3B and $A3B^+$

Results


0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
Top-1 accuracy (higher is better)
baseB
selffer BnB
selffer BnB+
transfer AnB
transfer AnB+
0 1 2 3 4 5 6 7
5: Transfer + fine-tuning improves generalization
3: Fine-tuning recovers co-adapted interactions
2: Performance drops due to fragile co-adaptation
4: Performance drops due to representation specificity
Layer n at which network is chopped and retrained

Dissimilar datasets

- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases



Investigate components of CNNs

- Kernel size
- Kernel (channel) number
- Stride
- Dimensionality of fully connected layers
- Data augmentation
- Model averaging

Investigate components of CNNs (cont'd)

- (Chatfield et al. BMVC'14) pre-train on ImageNet and fine-tune on PASCAL VOC 2007
- Different architectures
  - mAP: CNN-S > (marginally) CNN-M > (∼%2.5) CNN-F
- Different data augmentation
  - No augmentation
  - Flipping (almost no improvement)
  - Smaller dimension downsized to 256, cropping 224 × 224 patches from the center and 4 corners, flipping (∼ 3% improvement)

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8	
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max	Fast similar to AlexNet
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max	Medium similar to Clarifai model
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x3 pool	4096 drop- out	4096 drop- out	1000 soft- max	Slow similar to OverFeat Accurate model

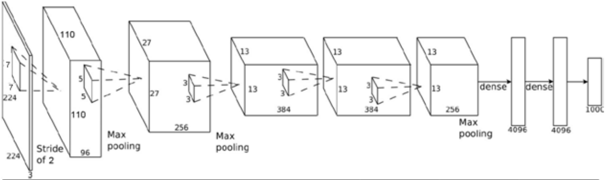
(Chatfield et al. BMVC 2014)

Investigate components of CNNs (cont'd)

- Gray-scale vs. color ($\sim$ 3% drop)
- Decrease the number of nodes in FC7
  - to 2048 (surprisingly, marginally better)
  - to 1024 (marginally better)
  - to 128 ($\sim$ 2% drop but 32x smaller feature)
- Change the softmax regression loss to ranking hinge loss
  - $w_c \phi(I_{pos}) > w_c \phi(I_{neg}) + 1 - \xi$ (ξ is a slack variable)
  - $\sim$ 2.7% improvement
  - Note, $\mathcal{L}_2$ normalising features account for $\sim$ 5% of accuracy for VOC 2007
- On ILSVRC-2012, the CNN-S achieved a top-5 error rate of 13.1%
  - CNN-F: 16.7%
  - CNN-M: 13.7%
  - AlexNet: 17%

Model architecture-Clarifai

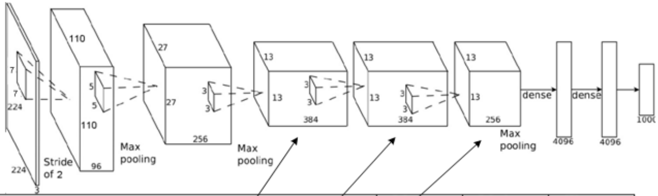
- Winner of ILSVRC 2013
- Max-pooling layers follow first, second, and fifth convolutional layers
- 11×11 to 7×7, stride 4 to 2 in 1st layer (increasing resolution of feature maps)
- Other settings are the same as AlexNet
- reduce the error by 2%.



Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	−−
1 convnet for Clarifai	38.4	16.5	−−

Model architecture-Clarifai further investigation

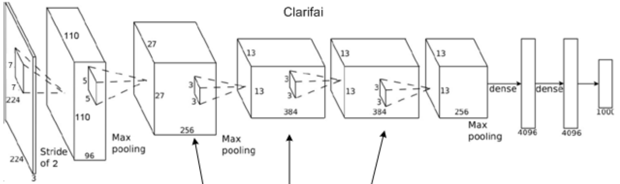
- More maps in the convolutional layers leads to small improvement.
- Model averaging leads to improvement (random initialization).



Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 AlexNet	40.7	18.2	– –
1 convnet for Clarifai	38.4	16.5	– –
5 convnets for Clarifai (a)	36.7	15.3	15.3
1 convnet for Clarifai but with layers 3,4,5: 512,1024,512 maps – (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	**36.0**	**14.7**	**14.8**

Model architecture-Overfeat

- Less pooling and more filters (384 => 512 for conv3 and 384=>1024 for conv4/5).



Layer	1	2	3	4	5	6	7	Output 8
Stage	conv + max	conv + max	conv	conv	conv + max	full	full	full
# channels	96	256	512	1024	1024	3072	4096	1000
Filter size	11x11	5x5	3x3	3x3	3x3	-	-	-
Conv. stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling size	2x2	2x2	-	-	2x2	-	-	-
Pooling stride	2x2	2x2	-	-	2x2	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1

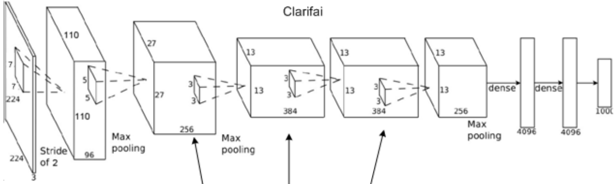
Overfeat

	top-5 error (%)		
	Clarifai	Overfeat-5	Overfeat-7
Without data augmentation	16.5	16.97	14.18

Model architecture-Overfeat

- With data augmentation, more complex model has better performance.

Clarifai



Layer	1	2	3	4	5	6	7	Output 8
Stage	conv + max	conv + max	conv	conv	conv + max	full	full	full
# channels	96	256	512	1024	1024	3072	4096	1000
Filter size	11x11	5x5	3x3	3x3	3x3	-	-	-
Conv. stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling size	2x2	2x2	-	-	2x2	-	-	-
Pooling stride	2x2	2x2	-	-	2x2	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1

Overfeat

	top-5 error (%)		
	Clarifai	Overfeat-5	Overfeat-7
With data augmentation	14.76	13.52	11.97
Without data augmentation	16.5	16.97	14.18

Model architecture-the devil of details

- CNN-F: similar to AlexNet, but less channels in conv3-5.
- CNN-S: the most complex one.
- CNN-M 2048: replace the 4096 features in fc7 by 2048 features. Makes little difference.
- Data augmentation. The input image is downsized so that the smallest dimension is equal to 256 pixels. Then 224 $\times$ 224 crops are extracted from the four corners and the centre of the image.

ILSVRC-2012	(top-5 error)
(a) Clarifai 1 ConvNet	16.0
(b) CNN F	16.7
(c) CNN M	13.7
(d) CNN M 2048	13.5
(e) CNN S	**13.1**

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x3 pool	4096 drop- out	4096 drop- out	1000 soft- max
Clarifai	96x7x7 st. 2, LRN,x2 pool	256x5x5 st. 2, pad1 LRN,x2 pool	384x3x3 st. 1,pad1	384x3x3 st. 1,pad1	256x3x3 st. 1,pad1	4096 drop	4096 drop	4096 drop

Model architecture-very deep CNN

- The deep model VGG in 2014.
- Apply 3×3 filter for all layers.
- 11 layers (A) to 19 layers (E).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Model architecture- very deep CNN

- The deep model VGG in 2014.
- Better to have deeper layers. 11 layers (A) => 16 layers (D).
- From 16 layers (D) to 19 layers (E), accuracy does not improve.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Model architecture- very deep CNN

- Scale jittering at the training time.
- The crop size is fixed to 224×224.
- S: the smallest side of an isotropically-rescaled training image.
- Scale jittering at the training time: [256; 512]: randomly select S to be within [256 512].
- LRN: local response normalisation. A-LRN does not improve on A.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Model architecture- very deep CNN

- Multi-scale averaging at the testing time.
- The crop size is fixed to 224×224.
- Q: the smallest side of an isotropically-rescaled testing image.
- Running a model over several rescaled versions of a test image (corresponding to different Q), followed by averaging the resulting class posteriors. Improves accuracy (25.5 => 24.8).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Model architecture- DeepImage of Baidu

- The deep model of Baidu in 2015.
- More hidden nodes at the fully connected layer (FC1-2), upto 8192.
- 16 layers.

Table 4: Single model comparison.

Team	Top-1 val. error	Top-5 val. error
GoogLeNet [21]	-	7.89%
VGG [20]	25.9%	8.0%
Deep Image	**24.88%**	**7.42%**

Layers	Conv 1-2	Max pool	Conv 3-4	Max pool	Conv 5-6-7	Max pool
# filters	64		128		256	

Conv 8-9-10	Max pool	Conv 11-12-13	Max pool	FC 1-2	FC 3	Softmax
512		512		6144	1000	

Model architecture- DeepImage of Baidu

- The deep model of Baidu in 2015.
- More hidden nodes at the fully connected layer (FC1-2), upto 8192.
- 16 layers.
- Data augmentation.

Table 4: Single model comparison.

Team	Top-1 val. error	Top-5 val. error
GoogLeNet [21]	-	7.89%
VGG [20]	25.9%	8.0%
Deep Image	**24.88%**	**7.42%**

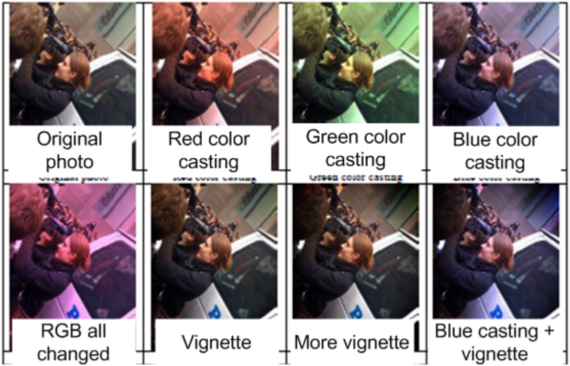
Layers	Conv 1-2	Max pool	Conv 3-4	Max pool	Conv 5-6-7	Max pool
# filters	64		128		256	

Conv 8-9-10	Max pool	Conv 11-12-13	Max pool	FC 1-2	FC 3	Softmax
512		512		6144	1000	

Augmentation	**The number of possible changes**
Color casting	68920
Vignetting	1960
Lens distortion	260
Rotation	20
Flipping	2
Cropping	82944(crop size is 224x224, input image size is 512x512)

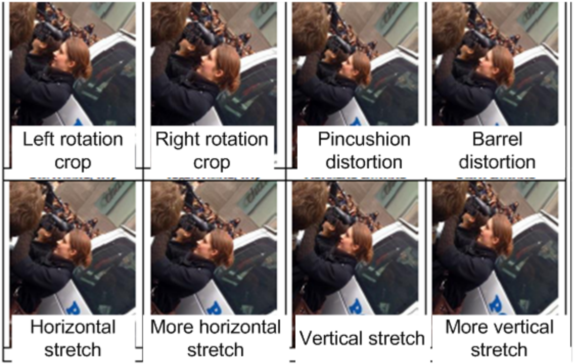
Model architecture- DeepImage of Baidu

- Data augmentation.



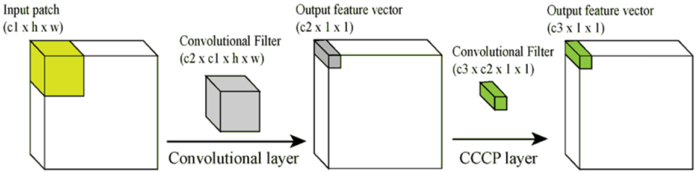
Model architecture- DeepImage of Baidu

- Data augmentation.



Model architecture- Network in Network

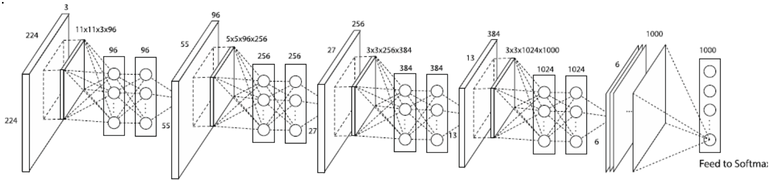
- Use 1$\times$1 filters after each convolutional layer.



Efficient implementation of CCCP

Model architecture- Network in Network

- Remove the two fully connected layers (fc6, fc7) of the AlexNet but add NIN into the AlexNet.



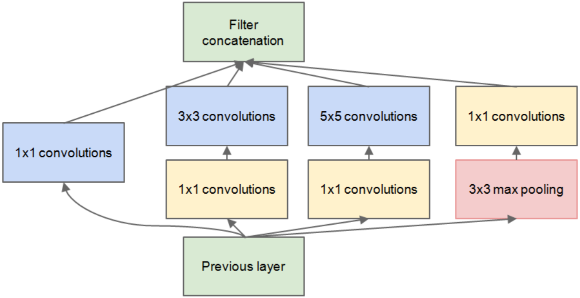
	Parameter Number	Performance	Time to train (GTX Titan)
AlexNet	60 Million (230 Megabytes)	40.7% (Top 1)	8 days
NIN	7.5 Million (29 Megabytes)	39.2% (Top 1)	4 days

Model architecture- GoogleNet

- Inspired by the good performance of NIN.

Model architecture- GoogleNet

- Inception model.
- Variable filter sizes to capture different visual patterns of different sizes. Enforce sparse connection between previous layer and output.
- The 1×1 convolutions are used for reducing the number of maps from the previous layer.



Model architecture- GoogleNet

- Based on inception model.
- Cascade of inception models.
- Widths of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.

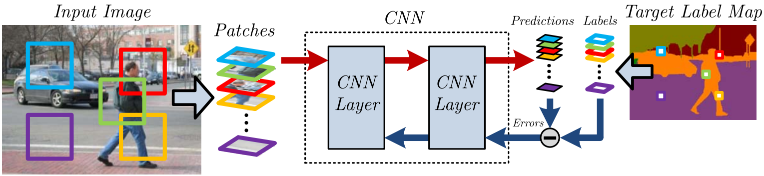
Model architecture- GoogleNet

- Parameters.

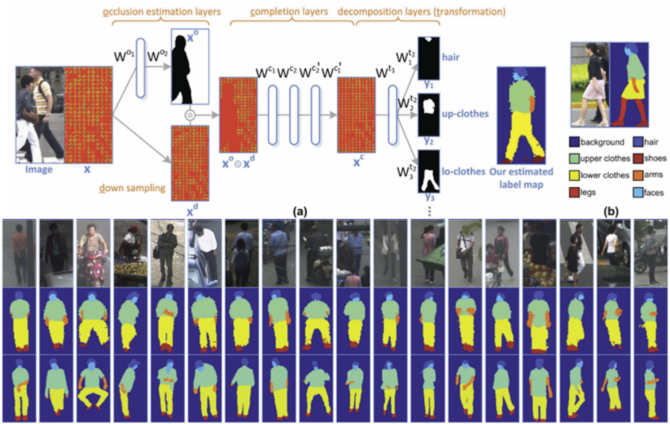
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

CNN for pixelwise classification

- Forward and backward propagation algorithms were proposed for whole-image classification: predicting a single label for a whole image
- Pixelwise classification: predicting a label at every pixel (e.g. segmentation, detection, and tracking)
- For pixelwise classification problems, it is generally trained and tested in a patch-by-patch manner, i.e. cropping a large patch around every pixel and inputting the patch to CNN for prediction (larger patches leading to better performance)
- It involves much redundant computation and is extremely inefficient



Directly Predict Segmentation Maps



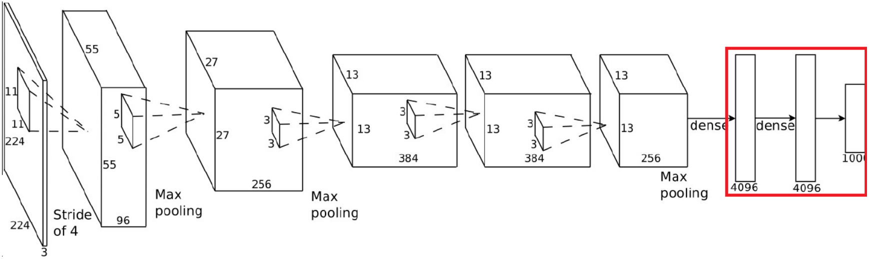
P. Luo, X. Wang, and X. Tang, "Pedestrian Parsing via Deep Decompositional Network," ICCV 2013.

Directly Predict Segmentation Maps

- Classifier is location sensitive has no translation invariance
  - Prediction not only depends on the neighborhood of the pixel, but also its location
- Only suitable for images with regular structures, such as faces and humans

Fully convolutional network

- One solution is to use fully convolutional network (Kang and Wang, arXiv:1411.4464)
- The convolution and pooling kernels can be directly applied to a full input image
- For fully connected layers, they can be converted into convolution layers



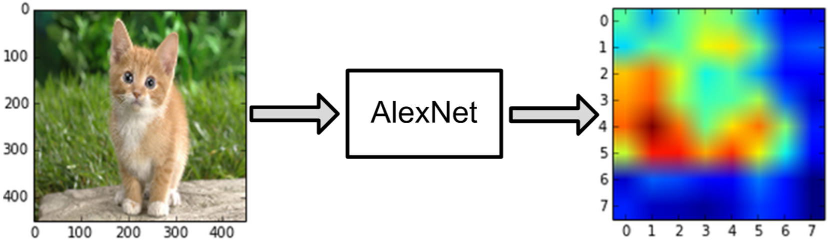
(Krizhevsky NIPS 2014)

Convert FC layers to convolution layers

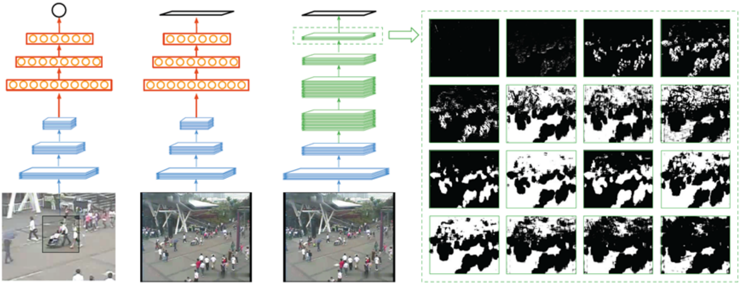
- For FC layers following a convolution or a pooling layer
  - Size of the input for the FC layer: $C \times M \times M$
  - Size of the output for the FC layer: N
  - Size of the converted convolution kernels: $C \times M \times M$
  - Number of the converted convolution kernels: N
- For FC layers following another FC layer
  - Size of the input for the FC layer: M
  - Size of the output for the FC layer: N
  - Size of the converted convolution kernels: $M \times 1 \times 1$
  - Number of the converted convolution kernels: N

Down-sampling due to greater-than-1 strides

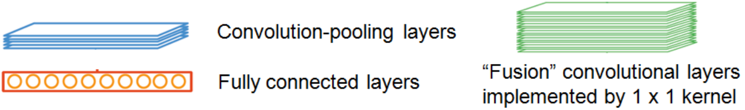
- The output of fully convolutional network is down-sampled due to the greater-than-1 strides in convolution and pooling layers



Fully convolutional neural networks with 1×1 kernels

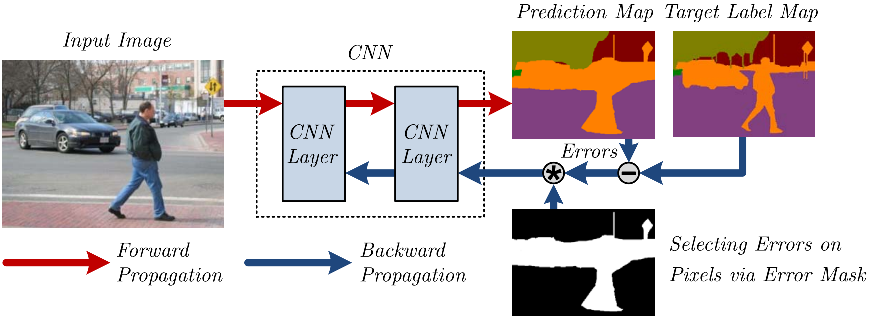


(a) CNN Patch-scanning (b) CNN Regression (c) FCNN Segmentation (d) FCNN Feature Maps



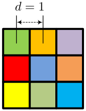
Fully convolutional networks with no down-sampling

- (Li, Zhao and Wang, arXiv:1412.4526) introduce *d*-regularly sparse kernels to avoid down-sampling
- The algorithm generates exactly the same results as patch-by-patch training and testing while speeds up the computation more than 1,500X

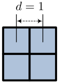


d-regularly convolution and pooling kernels

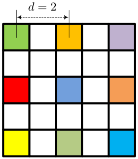
- d of original dense kernels is 1
- Insert all-zero rows and columns to create d-regularly sparse kernels



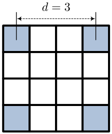
3×3 convolution kernel



2×2 pooling kernel



Converted convolution kernel



Converted pooling kernel

The algorithm

- Set stride to 1 for all convolution and pooling layers

Algorithm 1: Efficient Forward Propagation of CNN

Input: Input image I, convolution parameters W_k, b_k,
pooling kernels P_k, strides of each layer d_k

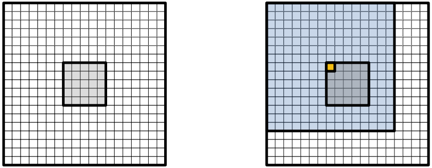
begin
    $d \leftarrow 1$
    $x_1 \leftarrow I$
    for $k = \{1, 2, \cdots, K\}$ **do**
        if *Layer* k == *convolution layer* **then**
            Convert W_k to $W_{k,d}$
            $y_k \leftarrow W_{k,d} *^1 x_k + b_k$,
        else if *Layer* k == *pooling layer* **then**
            Convert P_k to $P_{k,d}$
            $y_k \leftarrow P_{k,d} \circledcirc^1 x_k$
        end
        $d \leftarrow d \times d_k$
        $x_{k+1} \leftarrow y_k$
    end
    return *output feature map* y_k
end

A toy example

- Net architecture

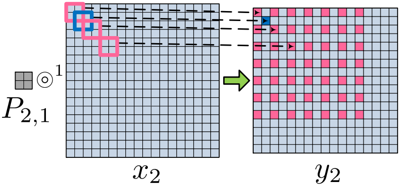
Layer	input patch	conv1	pool1
size / stride	15×15	2×2 / 1	2×2 / 2
Layer	conv2	pool2	conv3
size / stride	2×2 / 1	$3 \times 3/3$	2×2 / 1

- A 5×5 input is properly padded to 19×19
- The 15×15 topleft patch is used for comparison

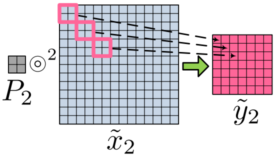


15×15 padded image $\quad$ 15×15 topleft patch

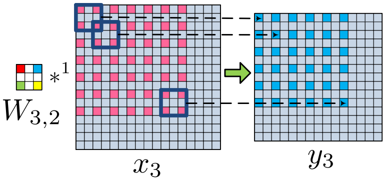
pool1 & conv2



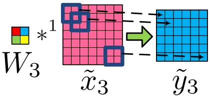
Proposed pool1



Original pool1

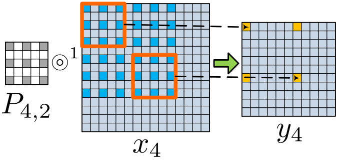


Proposed conv2

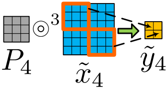


Original conv2

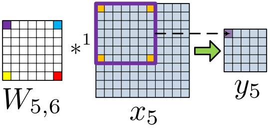
pool2 & conv3



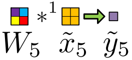
Proposed pool2



Original pool2



Proposed conv3



Original conv3

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.
- K. Chatfield, K. Simonyan, A. Vadaldi, and A. Zisserman, “Return of the Devil in the Details: Delving Deep into Convolutional Networks,” BMVC 2014.
- P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” In Proc. Int’l Conf. Learning Representations, 2014.

Reading materials

- K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556, 2014.
- R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, “Deep Image: Scaling up Image Recognition,” arXiv:1501.02876, 2015.
- M. Lin, Q.. Chen, and S. Yan, “Network in network,” arXiv:1312.4400v3, 2013.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” arXiv:1409.4842, 2014.
- K. Kang and X. Wang, “Fully Convolutional Neural Networks for Crowd Segmentation,” arXiv:1411.4464, 2014.
- H. Li, R. Zhao, and X. Wang, “Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification,” arXiv:1412.4526, 2014.