CS6375: Machine Learning

Gautam Kunapuli

Convolutional Neural Networks

Slides by Ian Goodfellow, Fei-Fei Li, Justin Johnson,
Serena Yeung, Marc'Aurelio Ranzato

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

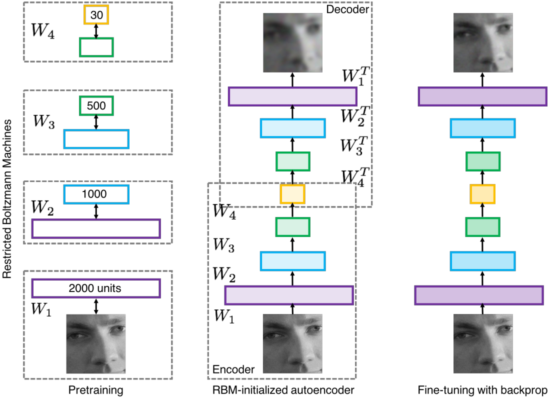


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

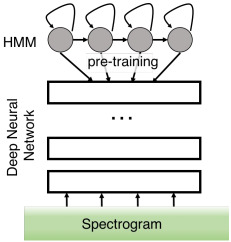
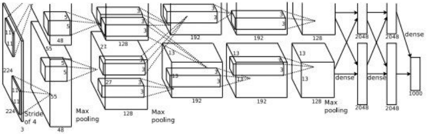
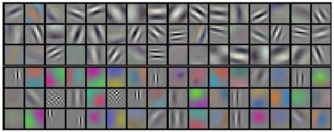


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





A bit of history:

Hubel & Wiesel,

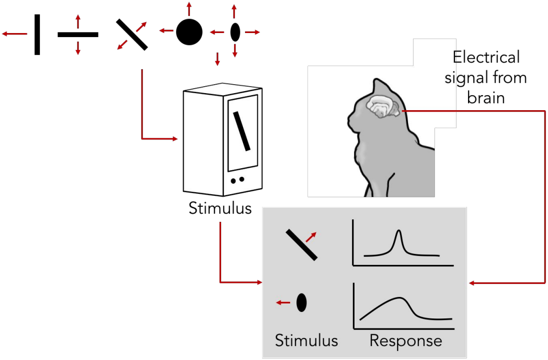
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

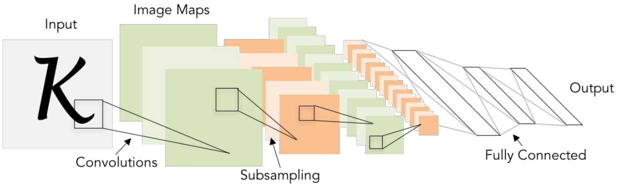
1968...



A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

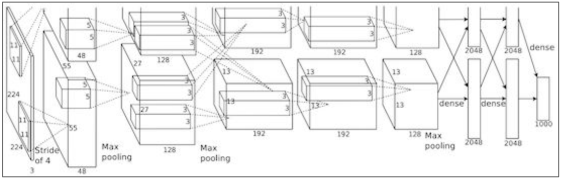


LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

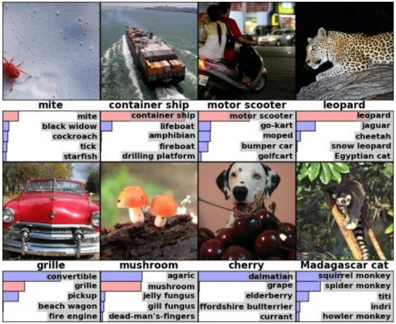
[Krizhevsky, Sutskever, Hinton, 2012]



“AlexNet”

Fast-forward to today: ConvNets are everywhere

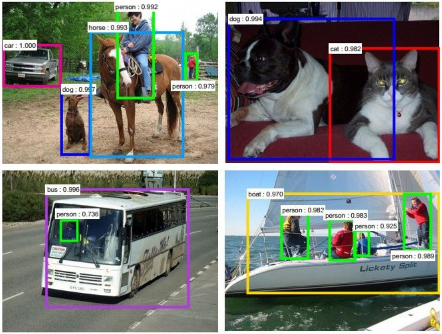
Classification



Retrieval

Fast-forward to today: ConvNets are everywhere

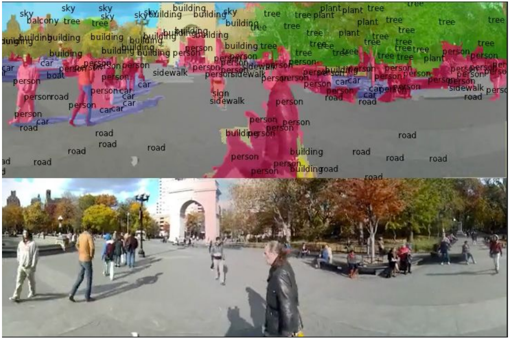
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

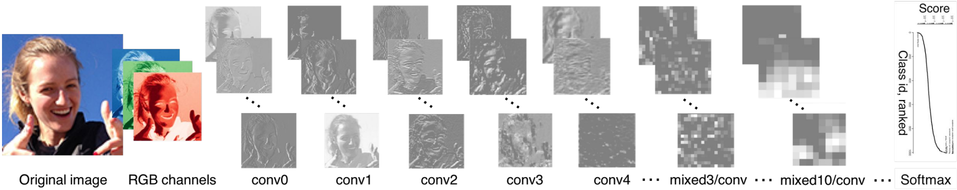
Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

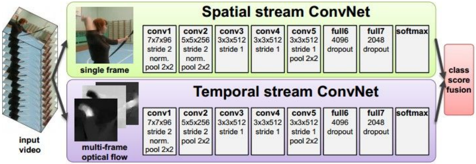
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

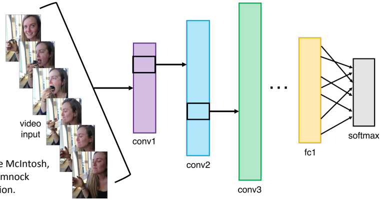
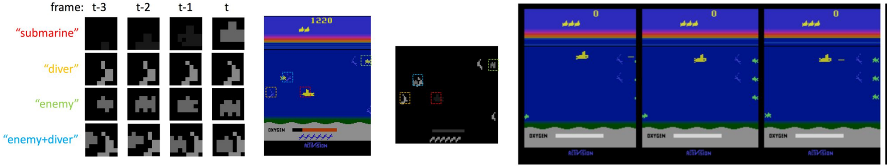


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

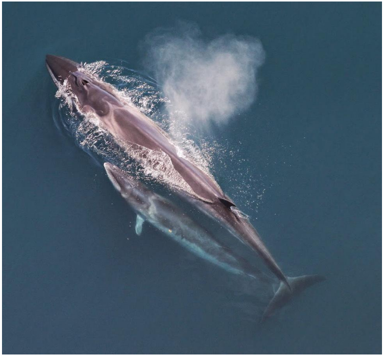
[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.



Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

No errors | Minor errors | Somewhat related

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

Convolutional Networks

- Scale up neural networks to process very large images / video sequences
  - Sparse connections
  - Parameter sharing
- Automatically generalize across spatial translations of inputs
- Applicable to any input that is laid out on a grid (1-D, 2-D, 3-D, ...)

Sparse Connectivity

Sparse connections due to small convolution kernel

Dense connections

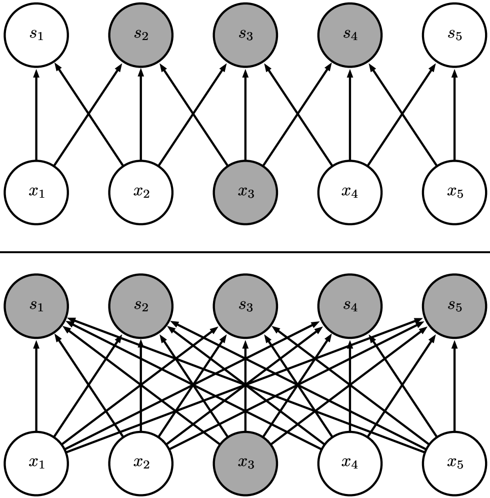


Figure 9.2

Sparse Connectivity

Sparse connections due to small convolution kernel

Dense connections

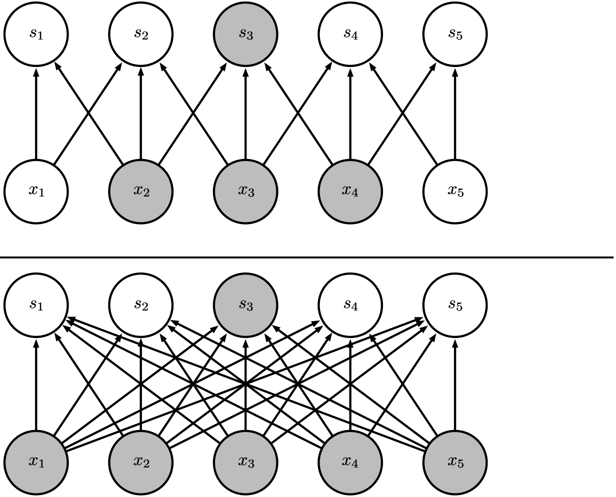


Figure 9.3

(Goodfellow 2016)

Growing Receptive Fields

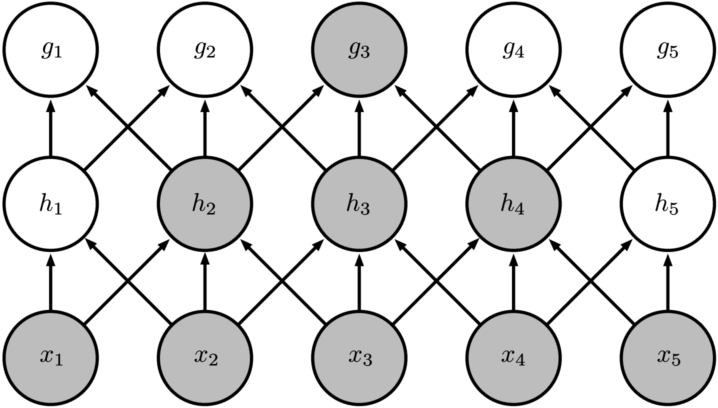
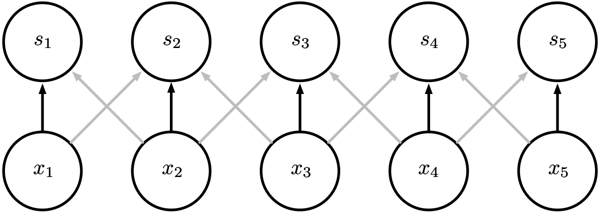


Figure 9.4

Parameter Sharing

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters

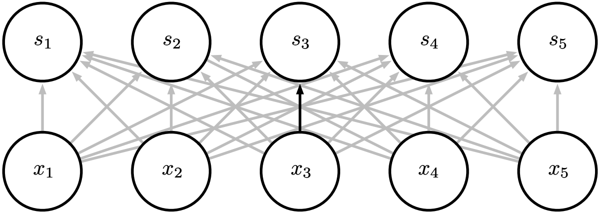


Figure 9.5

(Goodfellow 2016)

Edge Detection by Convolution

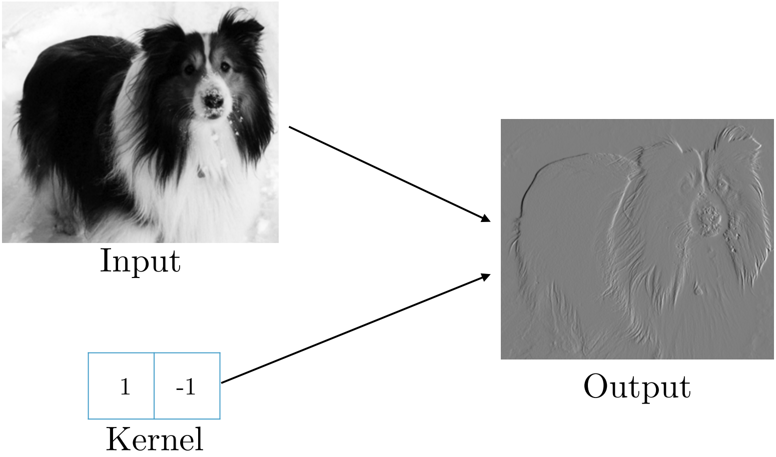


Figure 9.6

(Goodfellow 2016)

Efficiency of Convolution

Input size: 320 by 280

Kernel size: 2 by 1

Output size: 319 by 280

	Convolution	Dense matrix	Sparse matrix
Stored floats	2	319*280*320*280 > 8e9	2*319*280 = 178,640
Float muls or adds	319*280*3 = 267,960	> 16e9	Same as convolution (267,960)

Convolutional Network Components

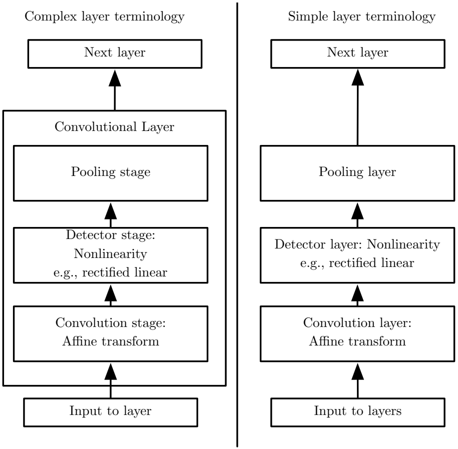
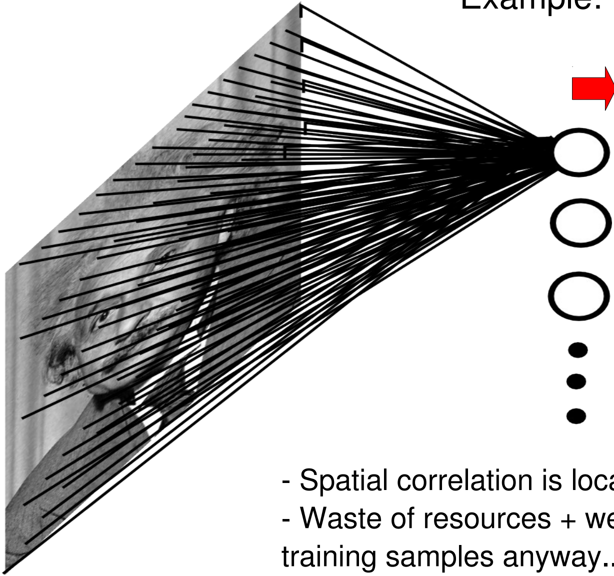


Figure 9.7

(Goodfellow 2016)

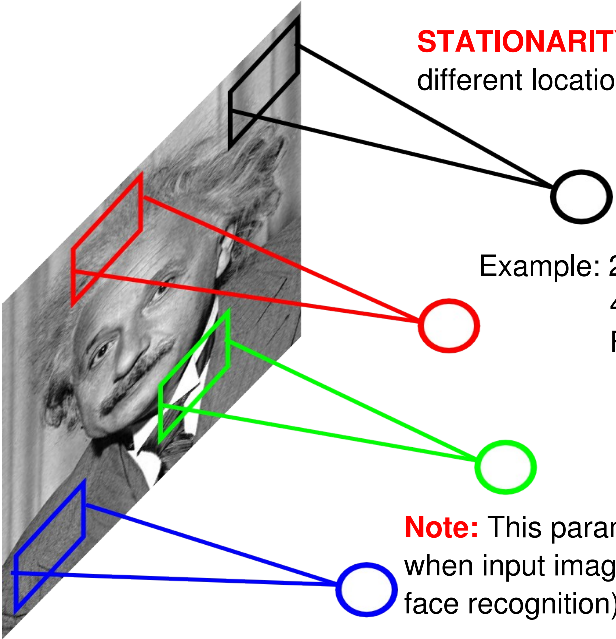
Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Locally Connected Layer



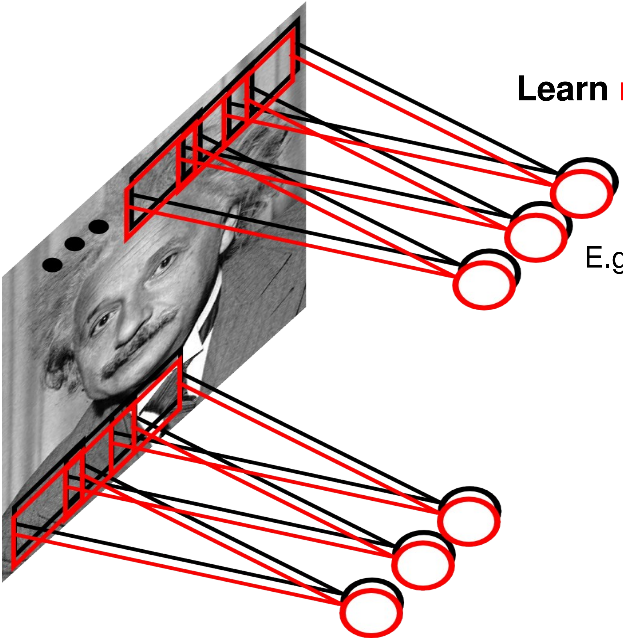
STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Ranzato

Convolutional Layer



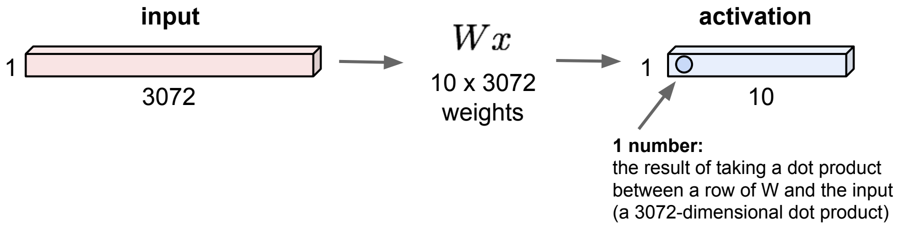
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

Fully Connected Layer

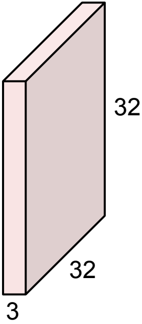
32x32x3 image -> stretch to 3072 x 1



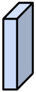
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

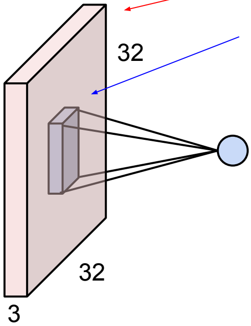


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

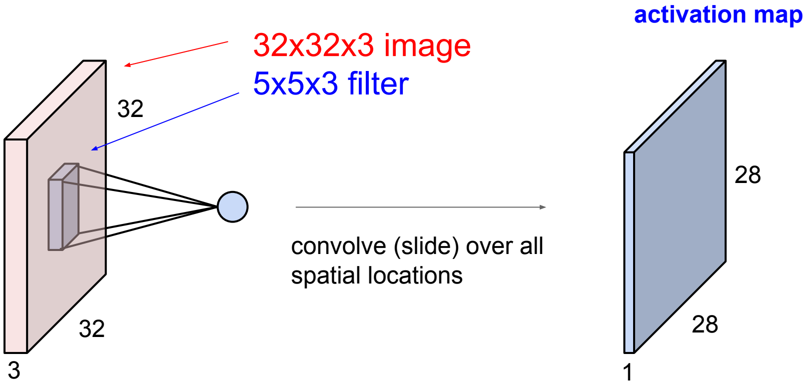
5x5x3 filter w



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

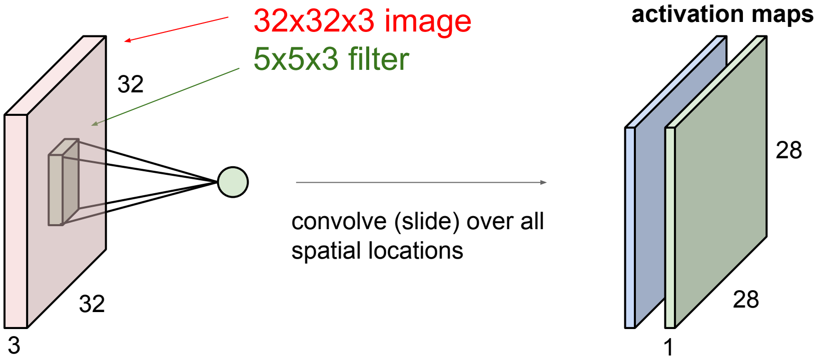
$$w^T x + b$$

Convolution Layer

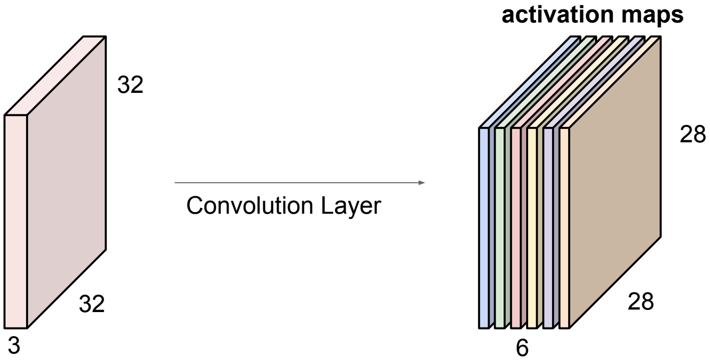


Convolution Layer

consider a second, green filter

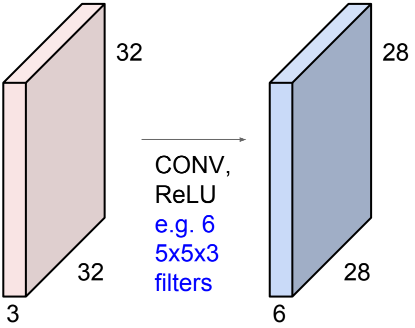


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

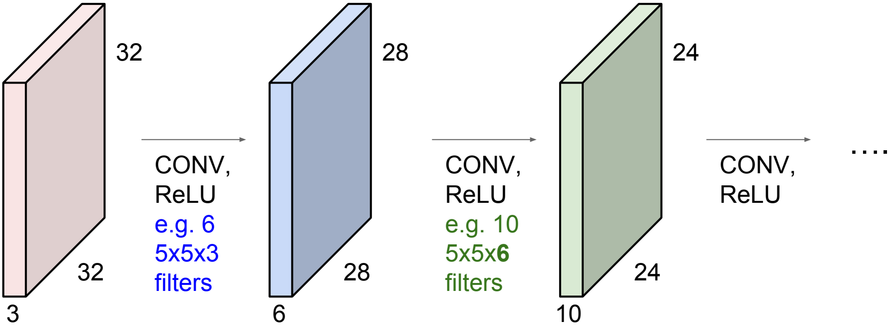


We stack these up to get a "new image" of size 28x28x6!

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



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



Convolution with Stride

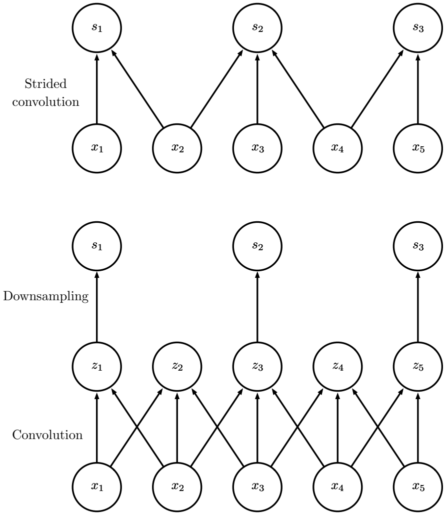
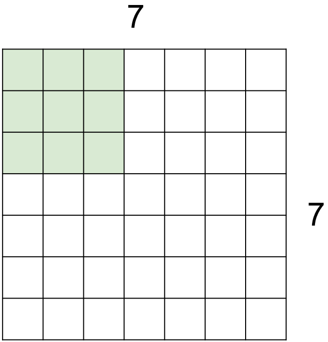


Figure 9.12

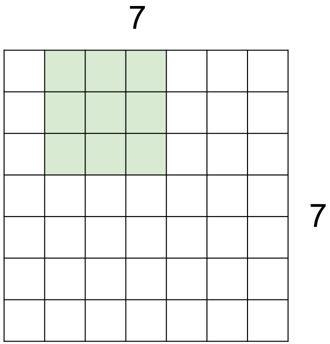
(Goodfellow 2016)

A closer look at spatial dimensions:



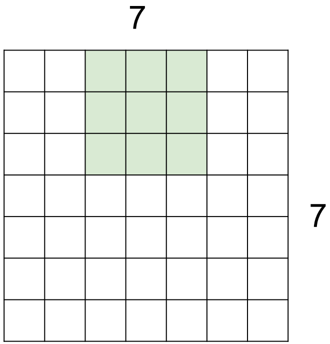
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



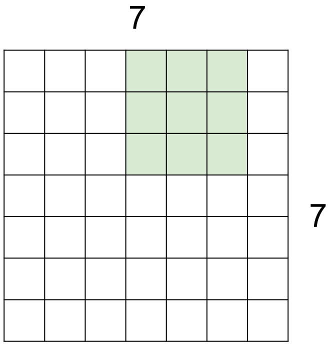
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



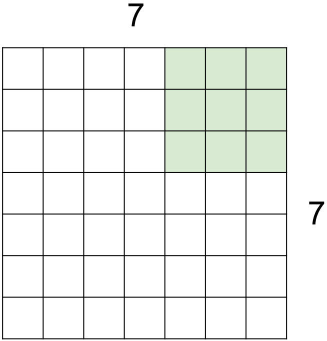
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

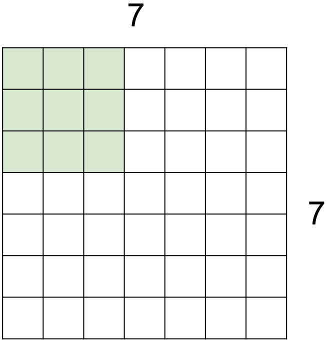
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

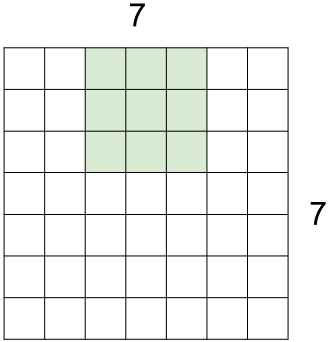
=> 5x5 output

A closer look at spatial dimensions:



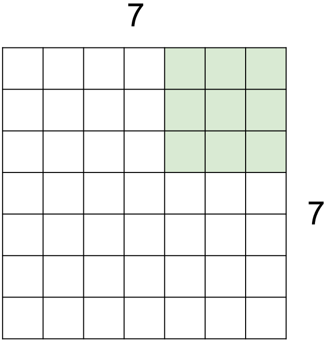
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



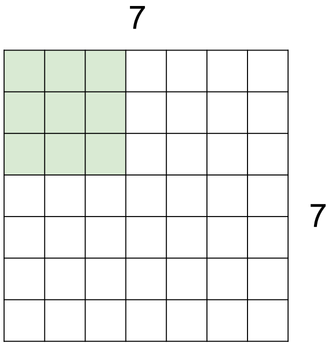
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



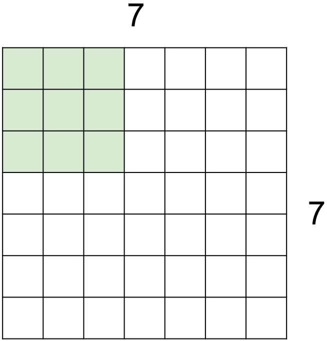
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Zero Padding Controls Size

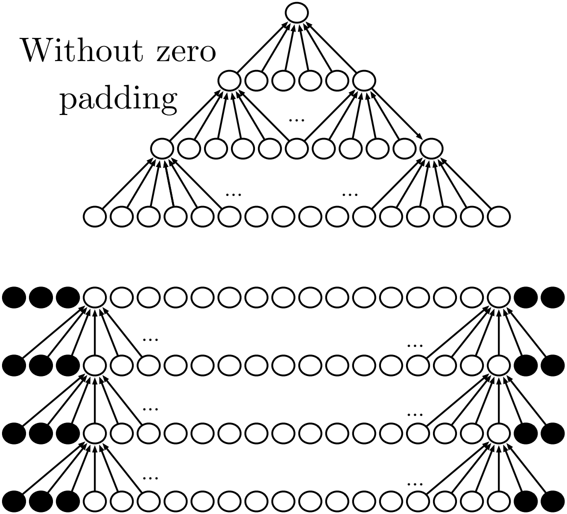
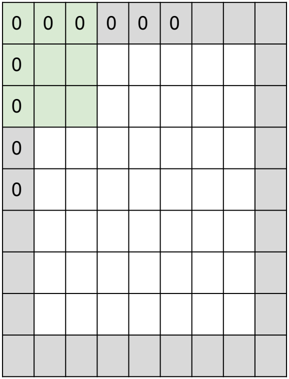


Figure 9.13

(Goodfellow 2016)

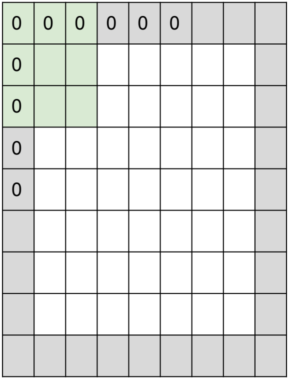
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

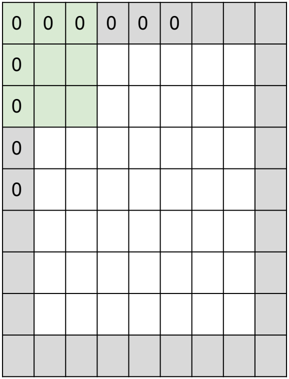
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border

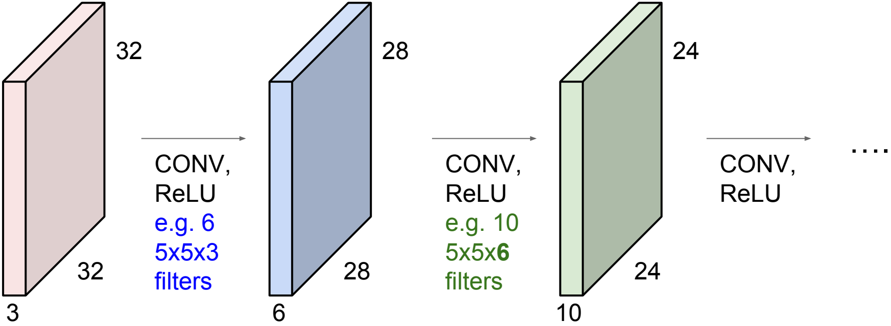


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

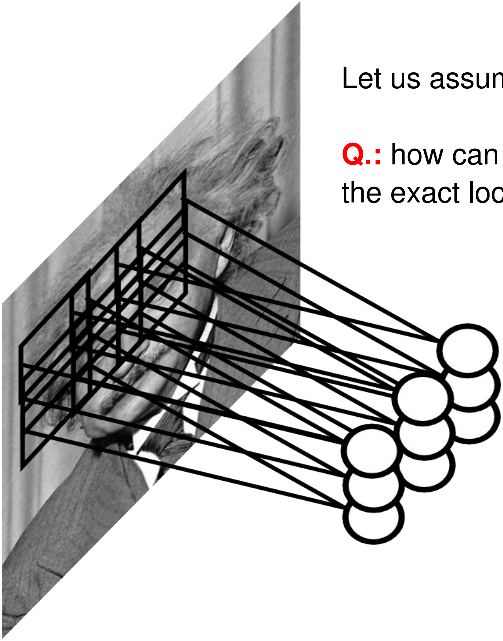
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



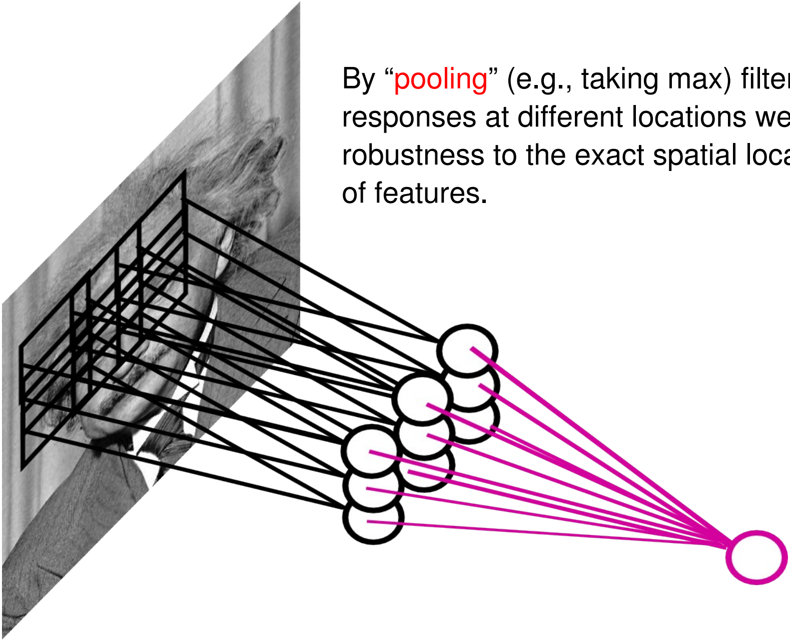
Pooling Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Ranzato

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Max Pooling and Invariance to Translation

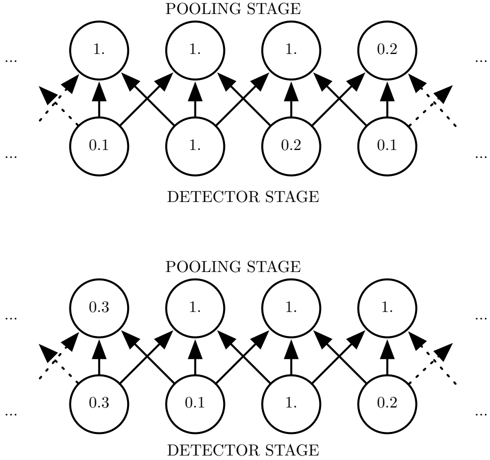


Figure 9.8

(Goodfellow 2016)

Cross-Channel Pooling and Invariance to Learned Transformations

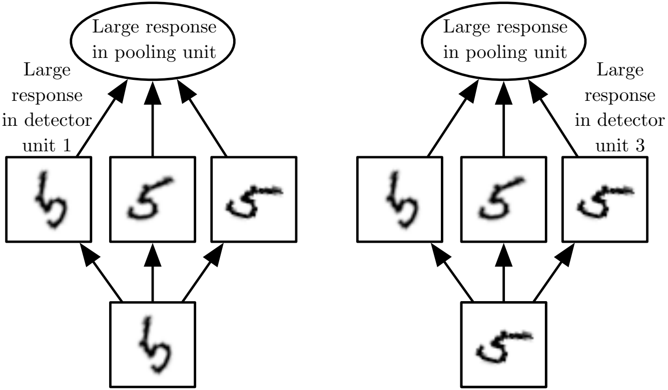
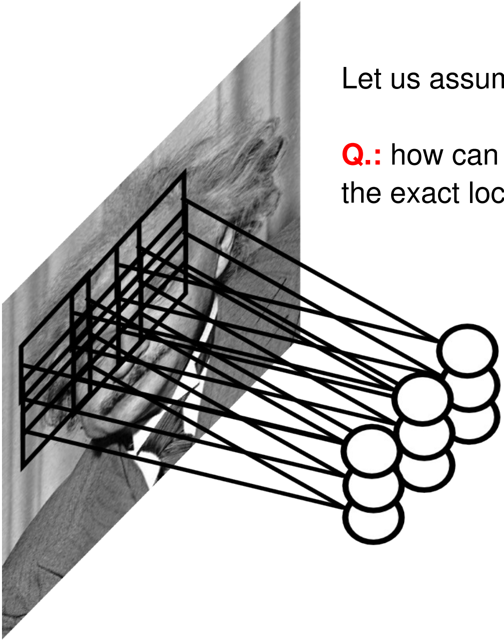


Figure 9.9

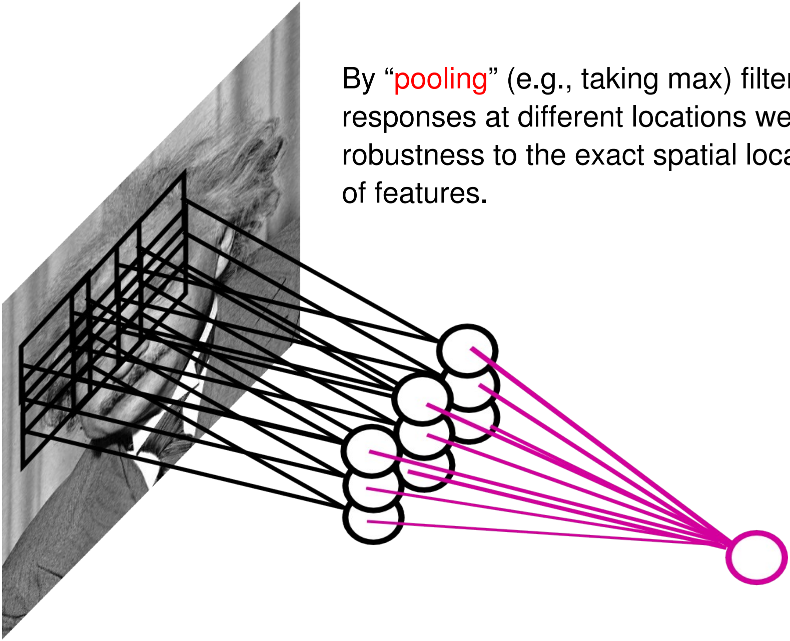
Pooling Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Ranzato

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

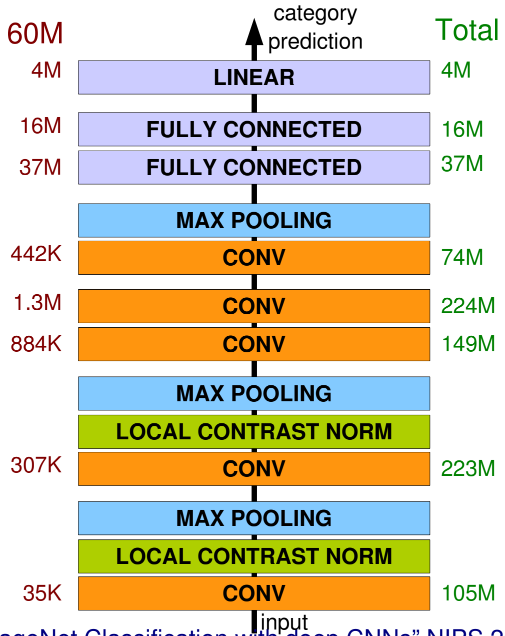
$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Ranzato

Architecture for Classification

Total nr. params: 60M

Total nr. flops: 832M



Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012

Choosing The Architecture

- Task dependent
- Cross-validation
- [Convolution $\rightarrow$ LCN $\rightarrow$ pooling]* + fully connected layer
- The more data: the more layers and the more kernels
  - Look at the number of parameters at each layer
  - Look at the number of flops at each layer
- Computational resources
- Be creative :)

How To Optimize

- SGD (with momentum) usually works very well

- Pick learning rate by running on a subset of the data
  Bottou "Stochastic Gradient Tricks" Neural Networks 2012
  - Start with large learning rate and divide by 2 until loss does not diverge
  - Decay learning rate by a factor of ~1000 or more by the end of training

- Use  non-linearity

- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

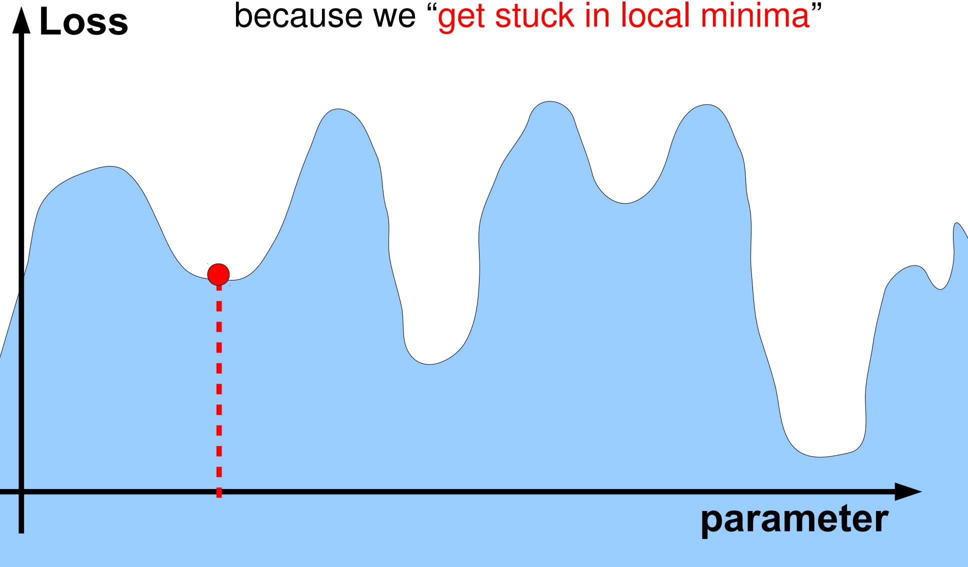
Improving Generalization

- Weight sharing (greatly reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, etc.)
- Dropout

  Hinton et al. “Improving Nns by preventing co-adaptation of feature detectors” arxiv 2012
- Weight decay (L2, L1)
- Sparsity in the hidden units
- Multi-task (unsupervised learning)

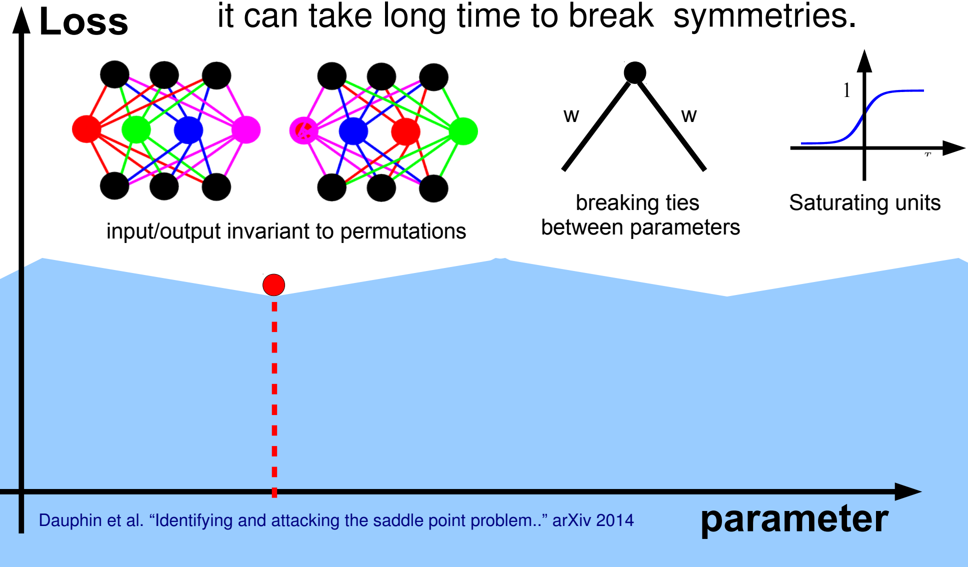
ConvNets: till 2012

Common wisdom: training does not work because we "get stuck in local minima"



ConvNets: today

Local minima are all similar, there are long plateaus,
it can take long time to break symmetries.

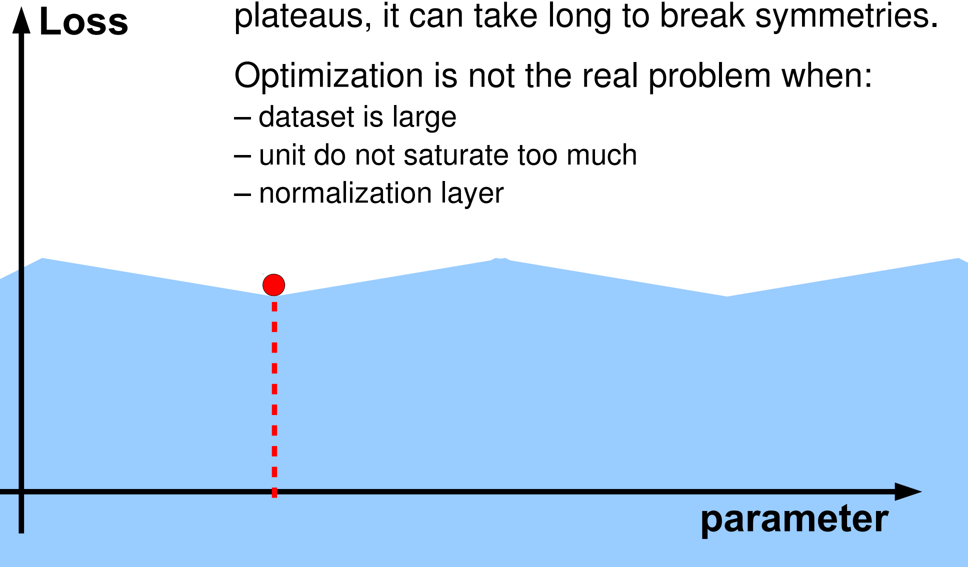


Dauphin et al. “Identifying and attacking the saddle point problem..” arXiv 2014

ConvNets: today

Local minima are all similar, there are long plateaus, it can take long to break symmetries.

Optimization is not the real problem when:
– dataset is large
– unit do not saturate too much
– normalization layer



ConvNets: today

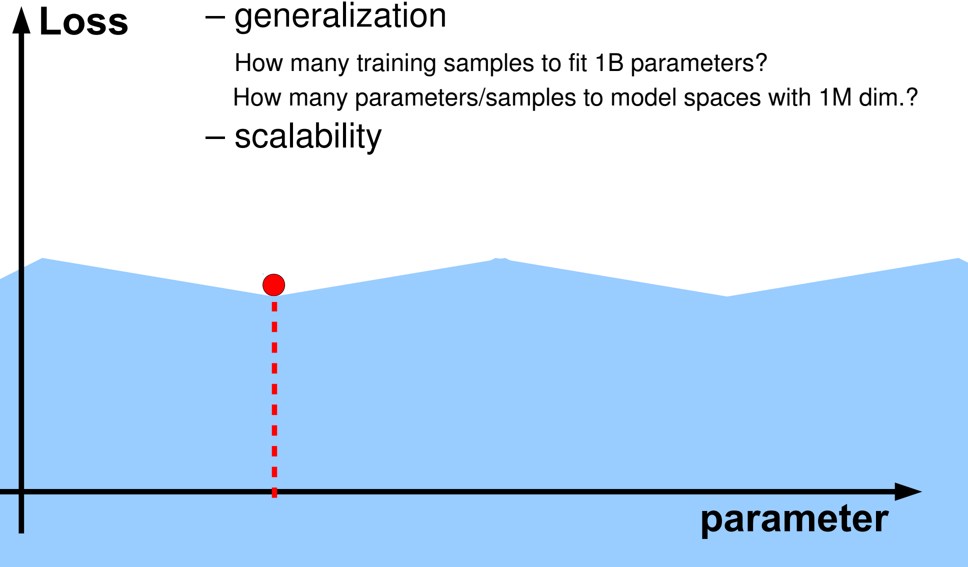
Today's belief is that the challenge is about:

– generalization

How many training samples to fit 1B parameters?

How many parameters/samples to model spaces with 1M dim.?

– scalability



Good To Know

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters
- Measure error on both training and validation set.
- Test on a small subset of the data and check the error $\rightarrow 0$.

What If It Does Not Work?

- Training diverges:
  - Learning rate may be too large → decrease learning rate
  - BPROP is buggy → numerical gradient checking
- Parameters collapse / loss is minimized but accuracy is low
  - Check loss function:
    - Is it appropriate for the task you want to solve?
    - Does it have degenerate solutions? Check “pull-up” term.
- Network is underperforming
  - Compute flops and nr. params. → if too small, make net larger
  - Visualize hidden units/params → fix optmization
- Network is too slow
  - Compute flops and nr. params. → GPU,distrib. framework, make net smaller

SOFTWARE

Torch7: learning library that supports neural net training

http://www.torch.ch
http://code.cogbits.com/wiki/doku.php (tutorial with demos by C. Farabet)
https://github.com/sermanet/OverFeat

Python-based learning library (U. Montreal)

- http://deeplearning.net/software/theano/ (does automatic differentiation)

Efficient CUDA kernels for ConvNets (Krizhevsky)

– code.google.com/p/cuda-convnet

Caffe (Yangqing Jia)

– http://caffe.berkeleyvision.org