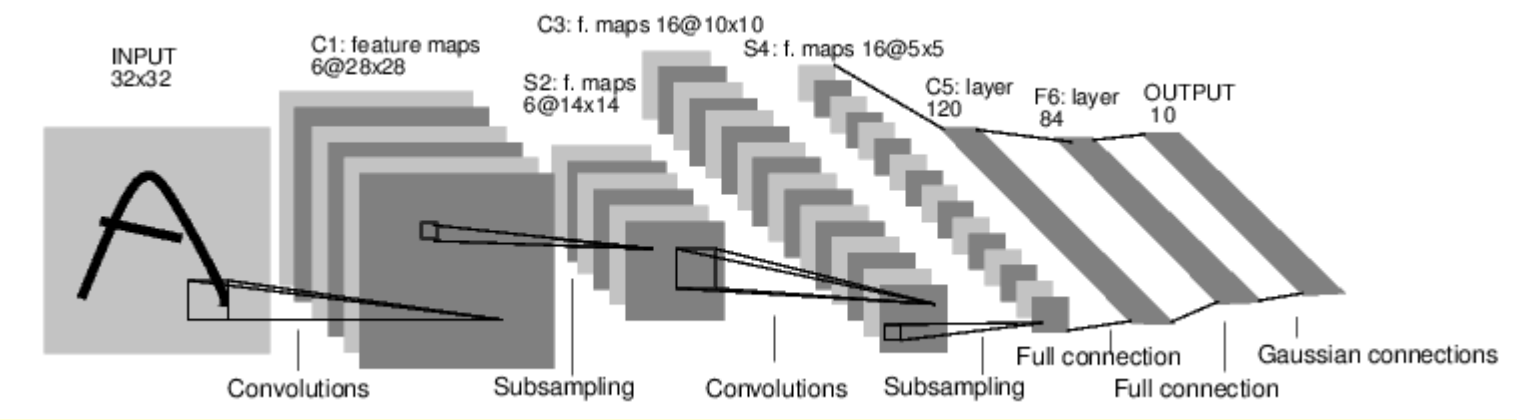
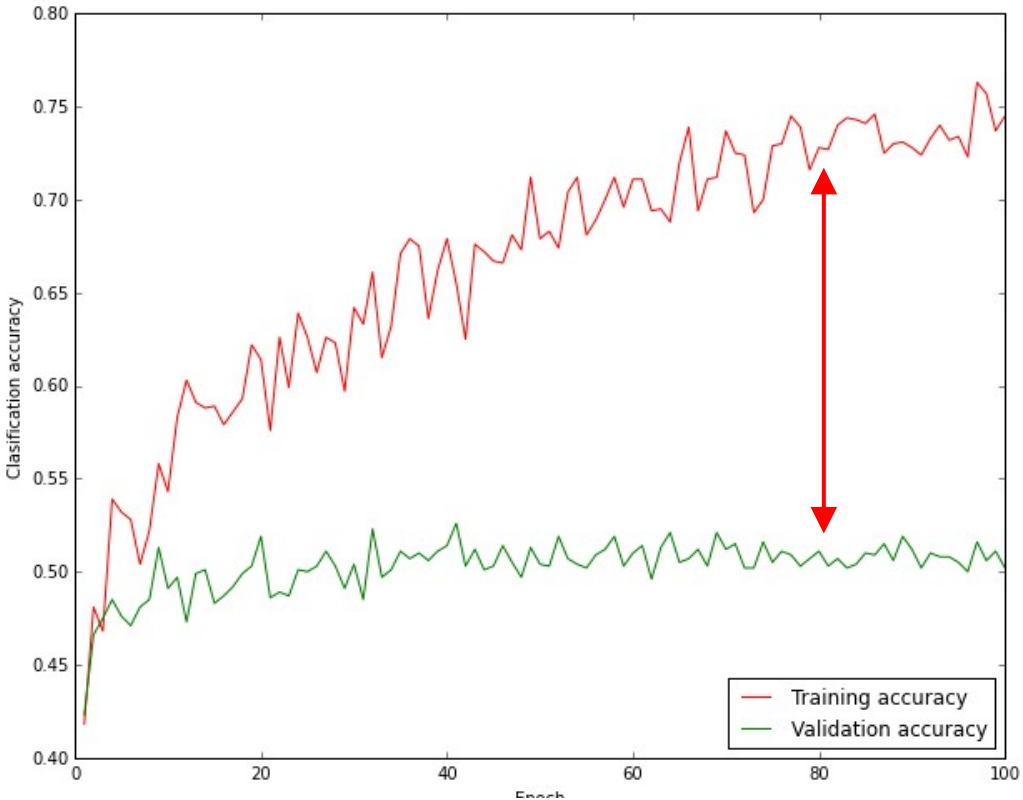
Convolutional Neural Networks (aka ConvNets, CNNs)



[LeNet-5, LeCun 1998]

Baby-sitting your network: monitoring accuracy



big gap = overfitting

=> increase regularization strength

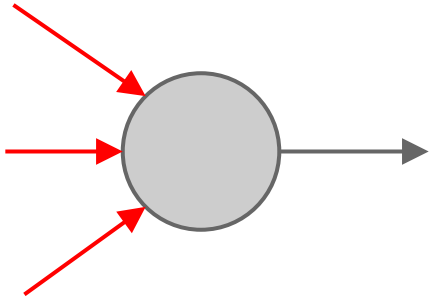
no gap

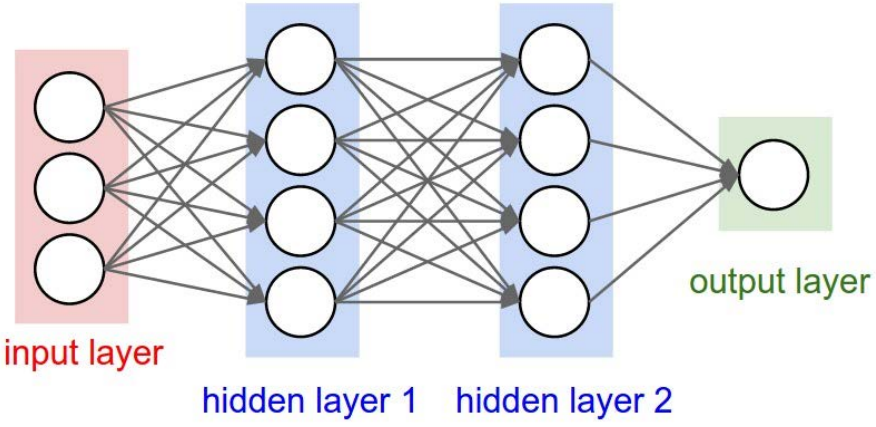
=> increase model capacity

Regularization knobs

- L2 regularization $\frac{1}{2}\lambda w^2$
- L1 regularization $\lambda \mid w \mid$
- L1 + L2 can also be combined

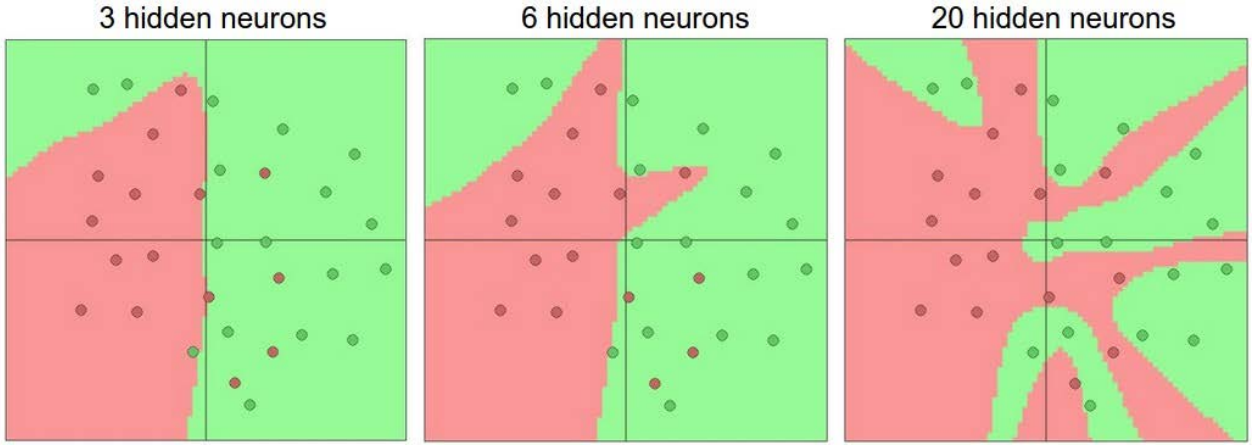
L1 is “sparsity inducing” (many weights become almost exactly zero)





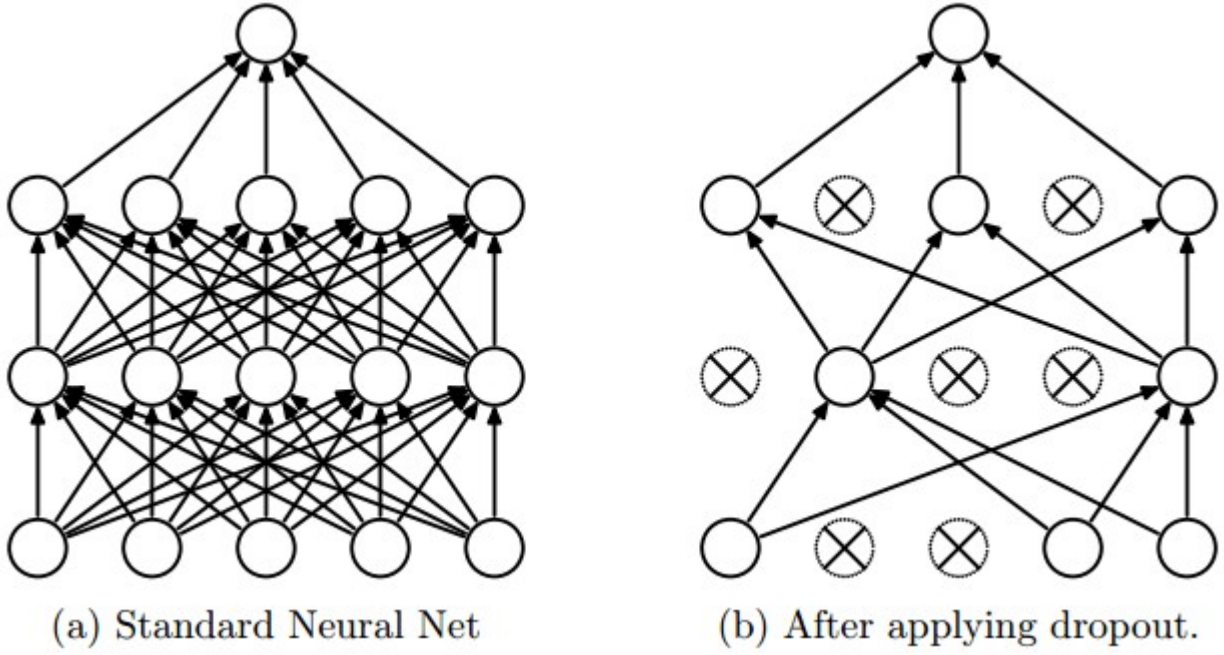
Seemingly unrelated: **Model Ensembles**

- One way to *always* improve final accuracy: take several trained models and average their predictions



Regularization: **Dropout**

"randomly set some neurons to zero"

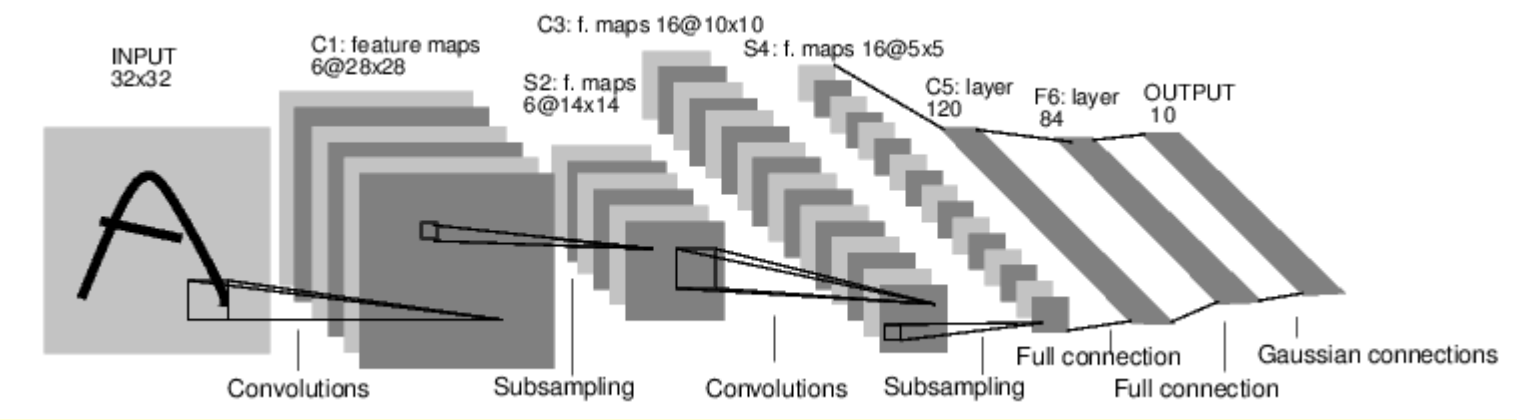


(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al.]

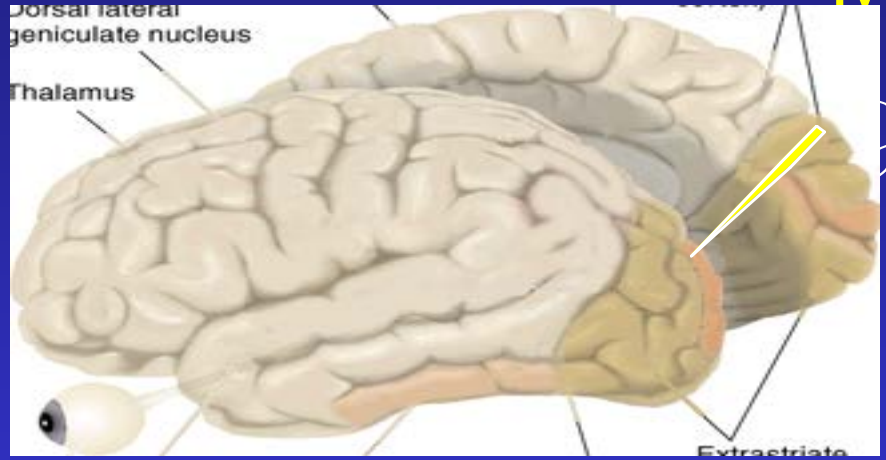
Convolutional Neural Networks (aka ConvNets, CNNs)



[LeNet-5, LeCun 1989]

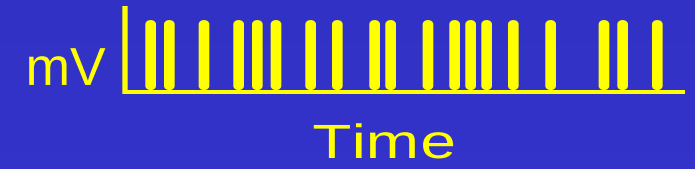
Single Cell Recording

Microelectrode

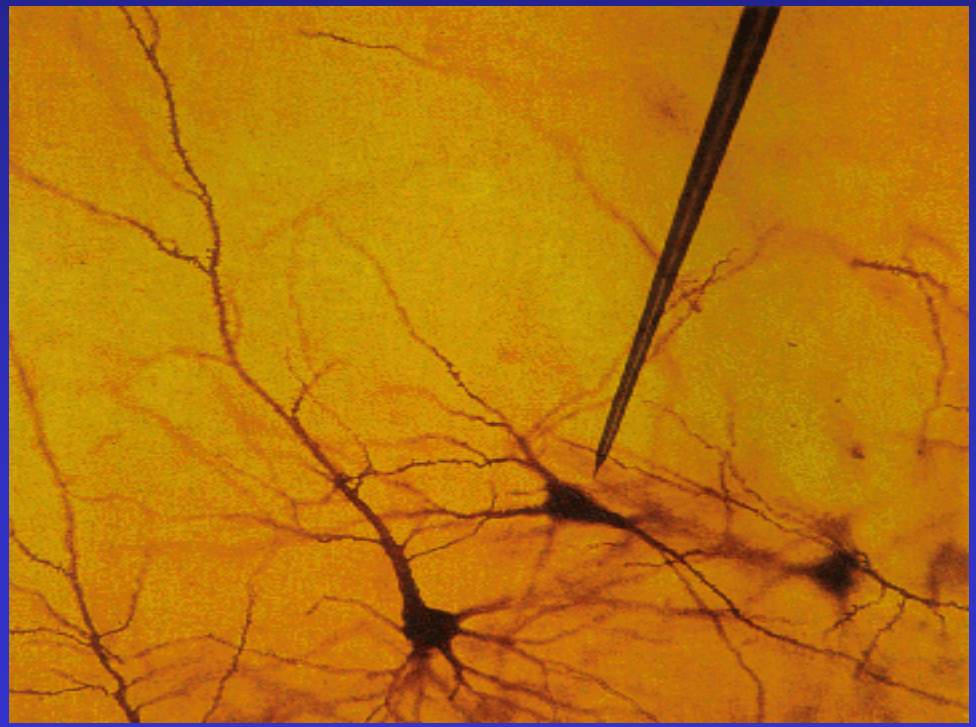


Amplifier

Electrical response
(action potentials)

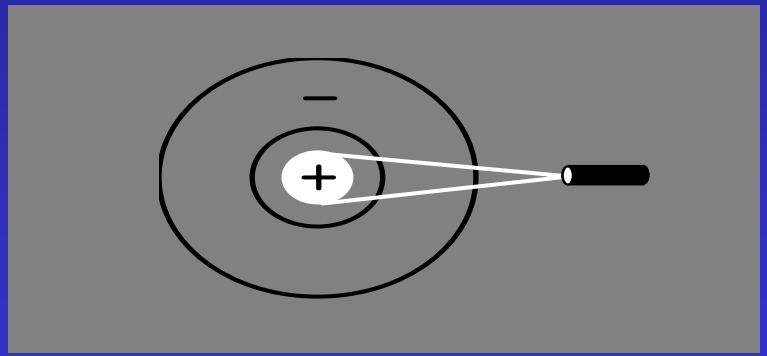


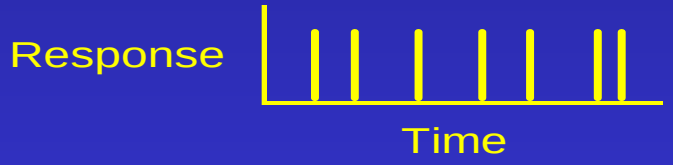
Single Cell Recording



Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



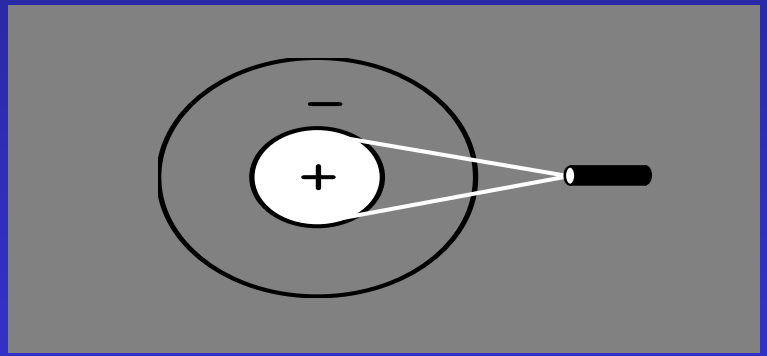


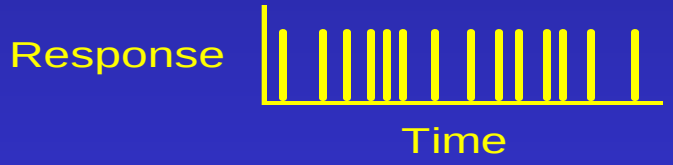
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



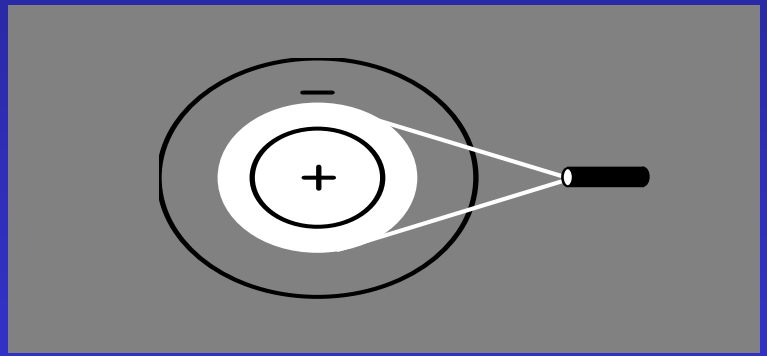


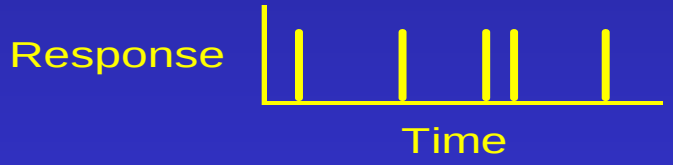
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



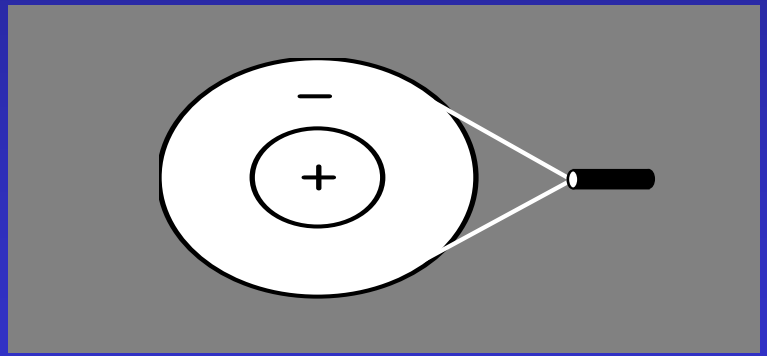


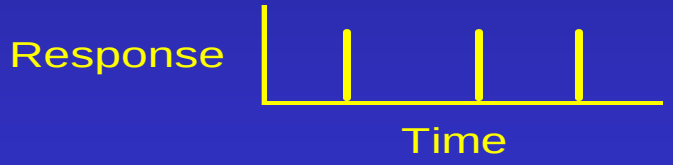
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



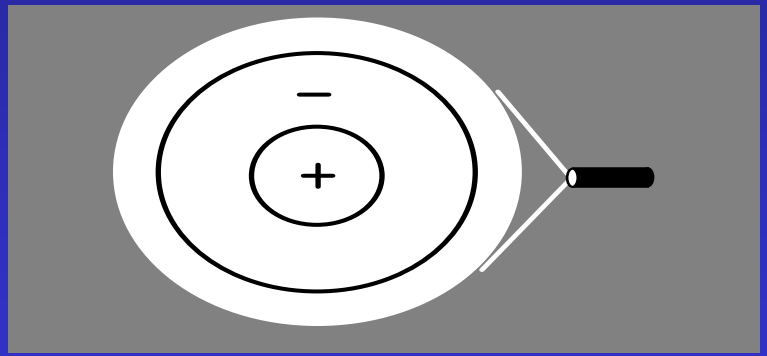


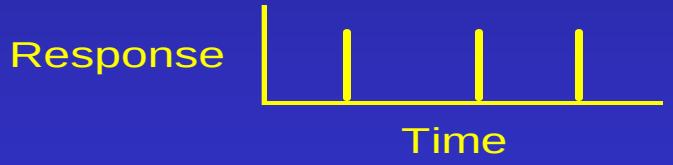
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



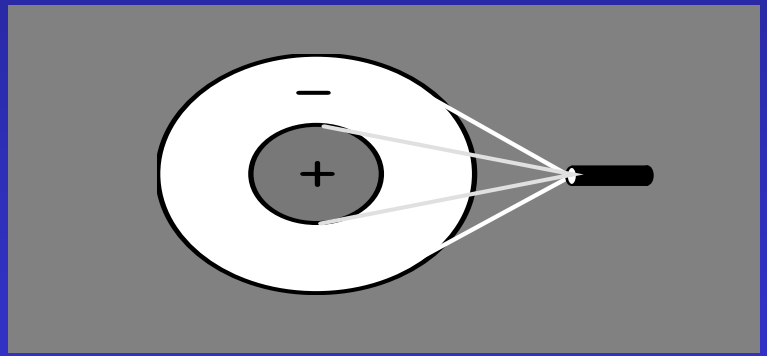


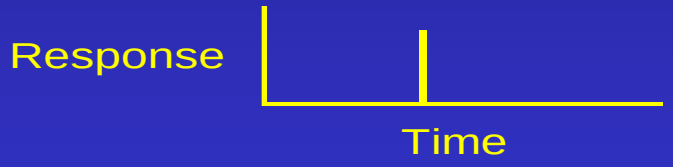
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



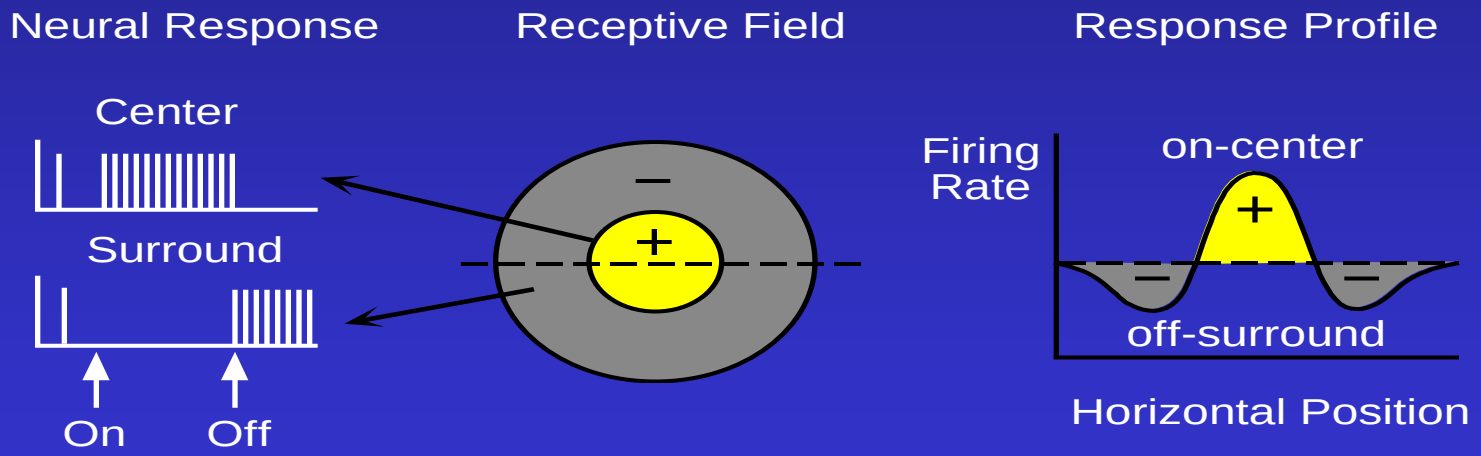


Stimulus condition

Electrical response

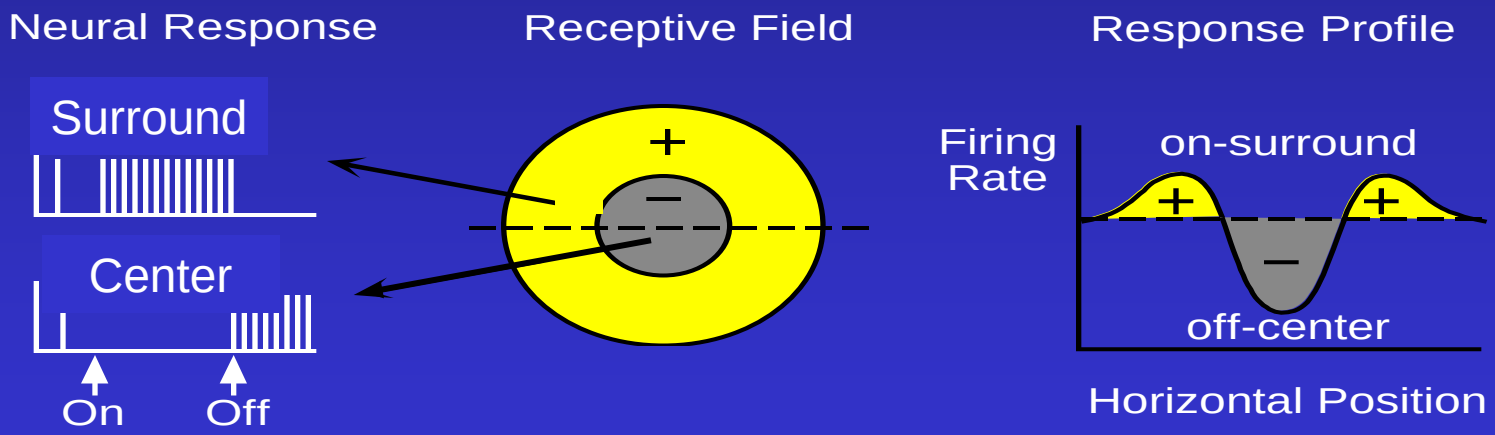
Retinal Receptive Fields

RF of On-center Off-surround cells

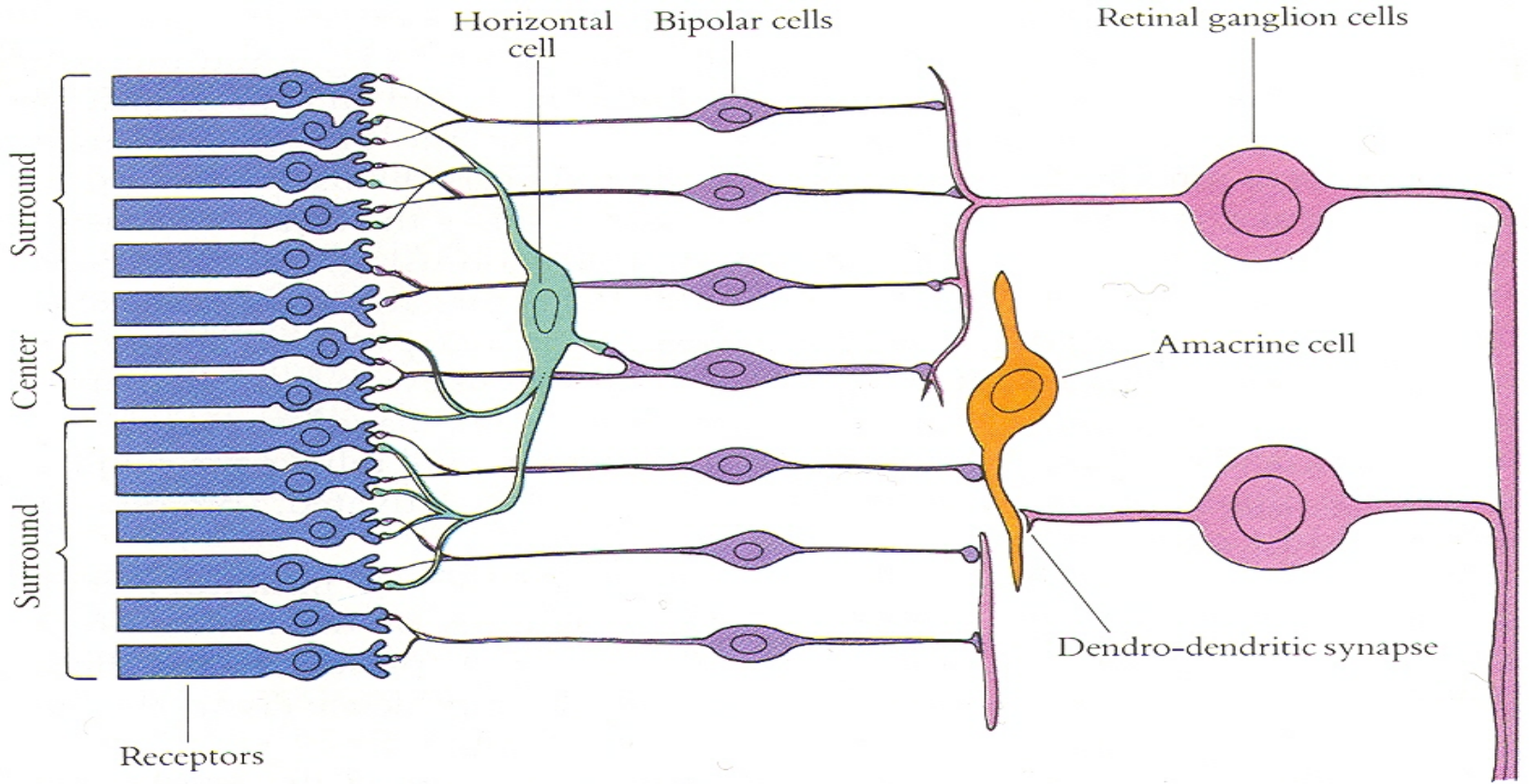


Retinal Receptive Fields

RF of Off-center On-surround cells



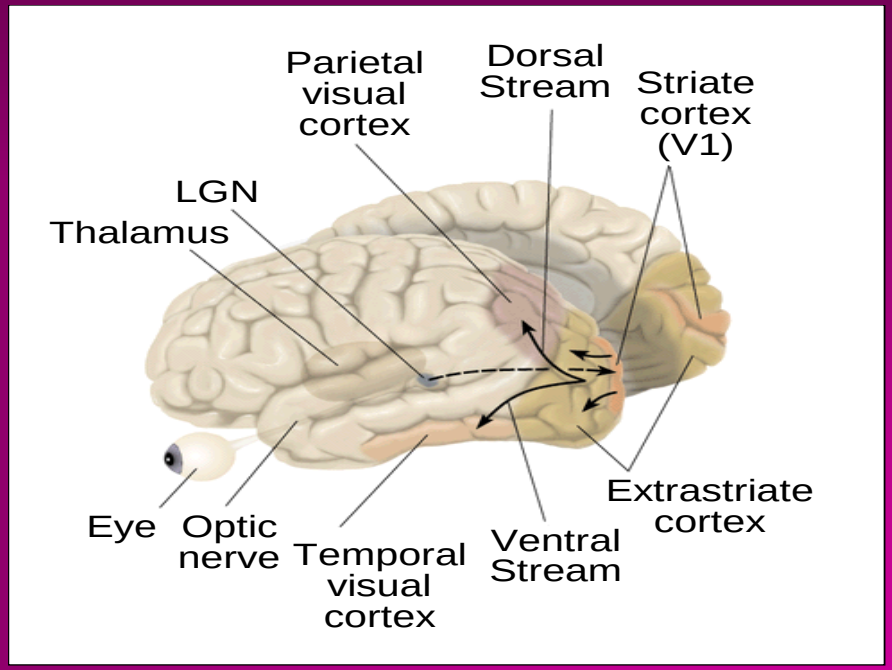
Retinal Receptive Fields



Visual Cortex

Cortical Area V1

aka:
Primary visual cortex
Striate cortex
Brodman's area 17



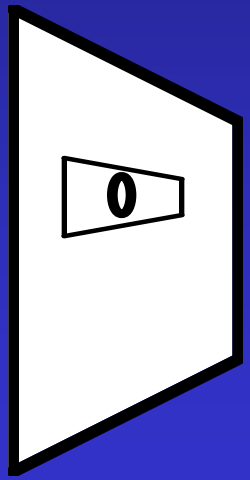
Cortical Receptive Fields

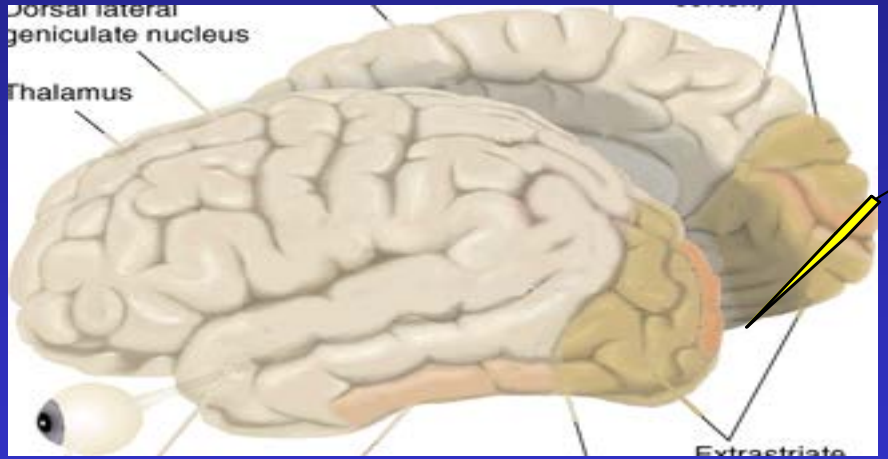
Single-cell recording from visual cortex

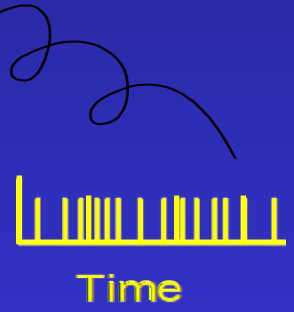
David Hubel & Thorston Wiesel

Cortical Receptive Fields

Single-cell recording from visual cortex







https://www.youtube.com/watch?v=IOHayh06LJ4

Cortical Receptive Fields

Three classes of cells in V1

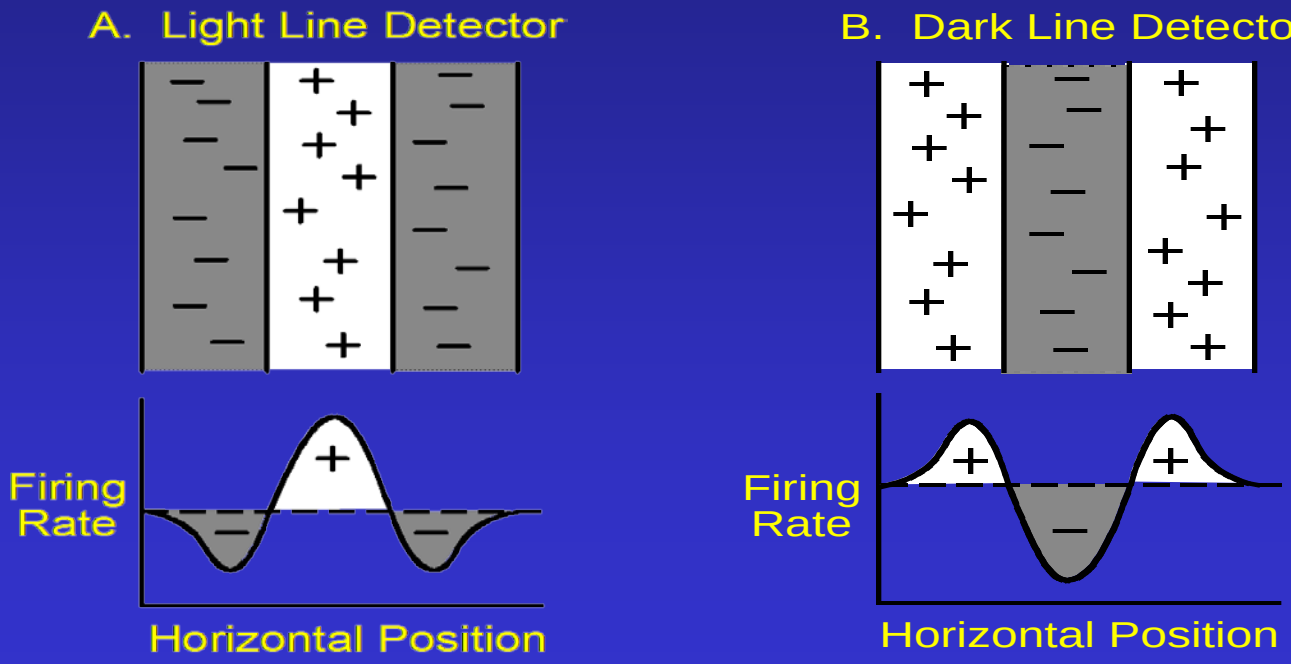
Simple cells

Complex cells

Hypercomplex cells

Cortical Receptive Fields

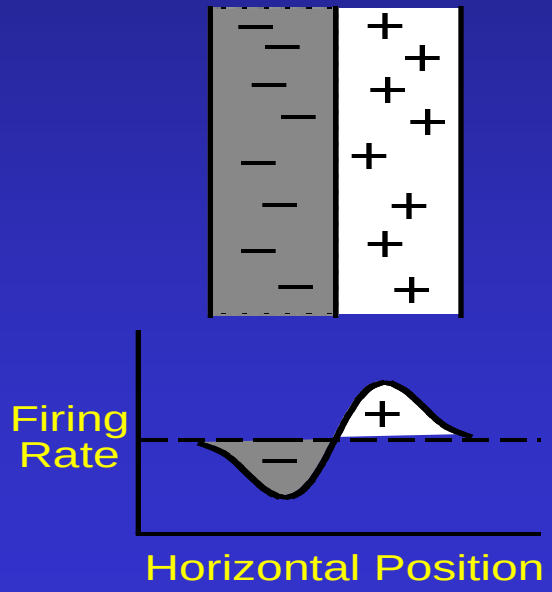
Simple Cells: “Line Detectors”



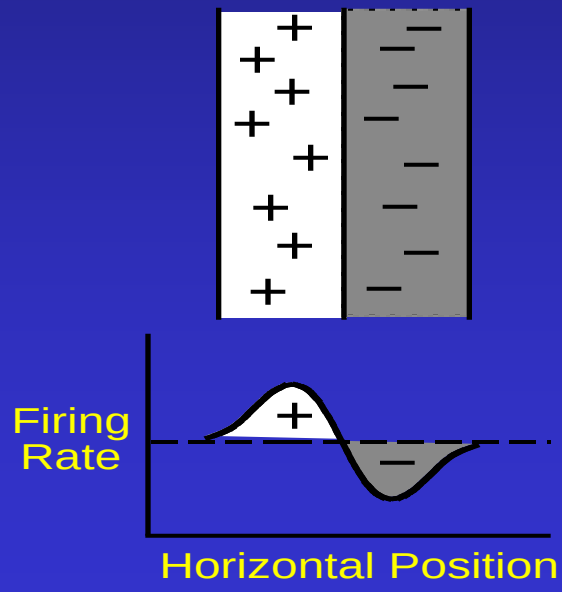
Cortical Receptive Fields

Simple Cells: “Edge Detectors”

C. Dark-to-light Edge Detector

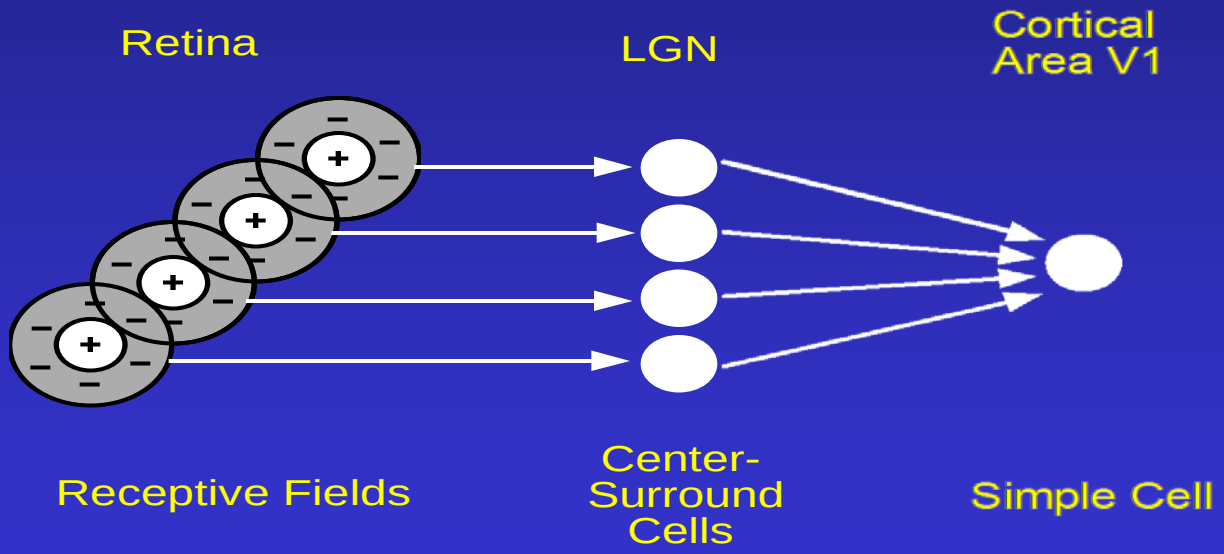


D. Light-to-dark Edge Detector



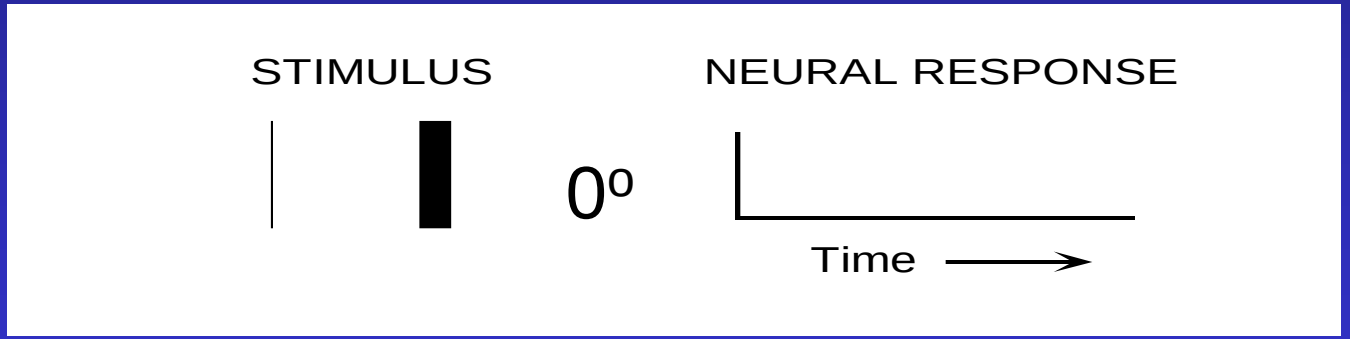
Cortical Receptive Fields

Constructing a line detector



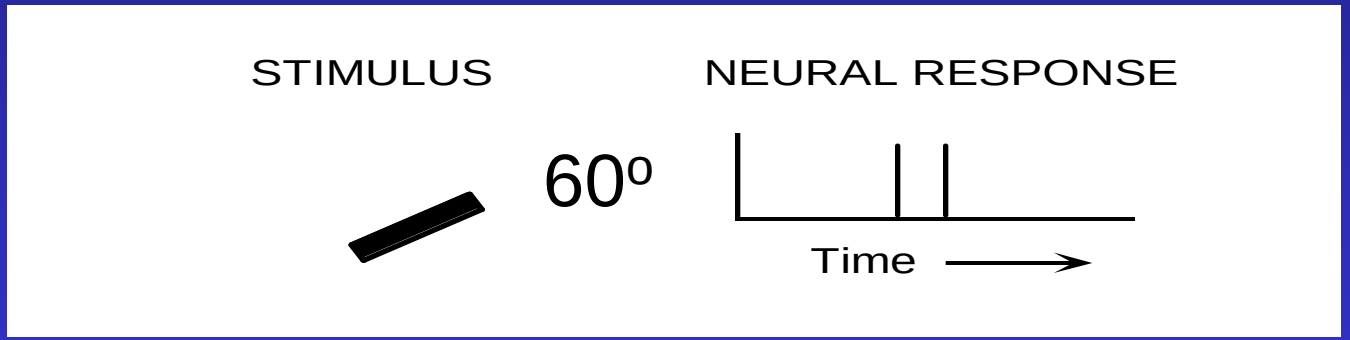
Cortical Receptive Fields

Complex Cells



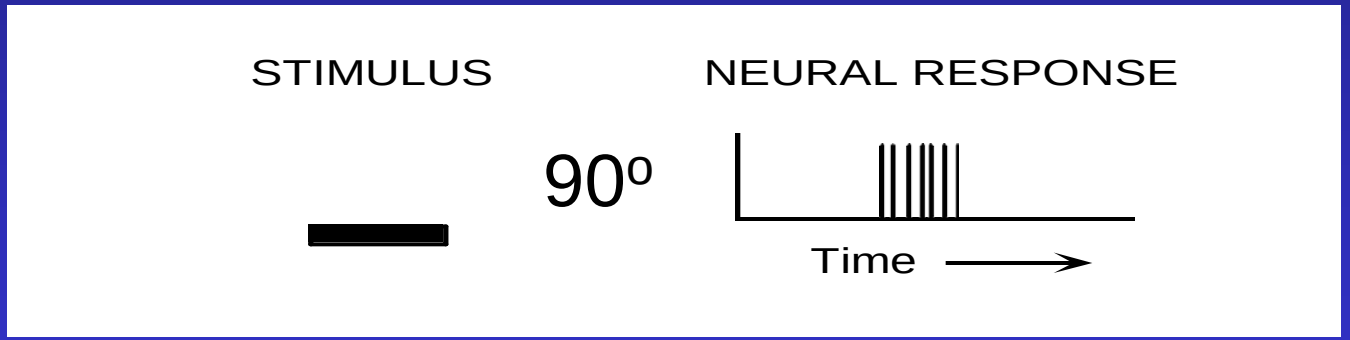
Cortical Receptive Fields

Complex Cells



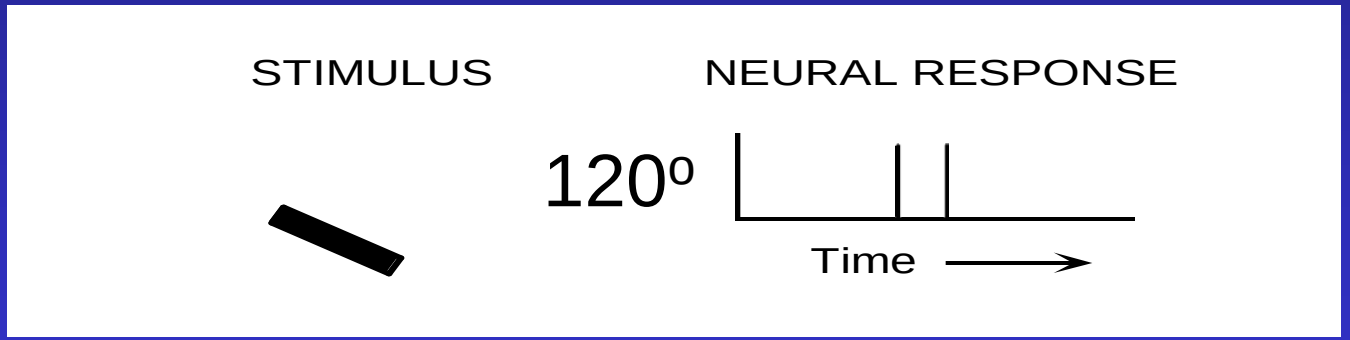
Cortical Receptive Fields

Complex Cells



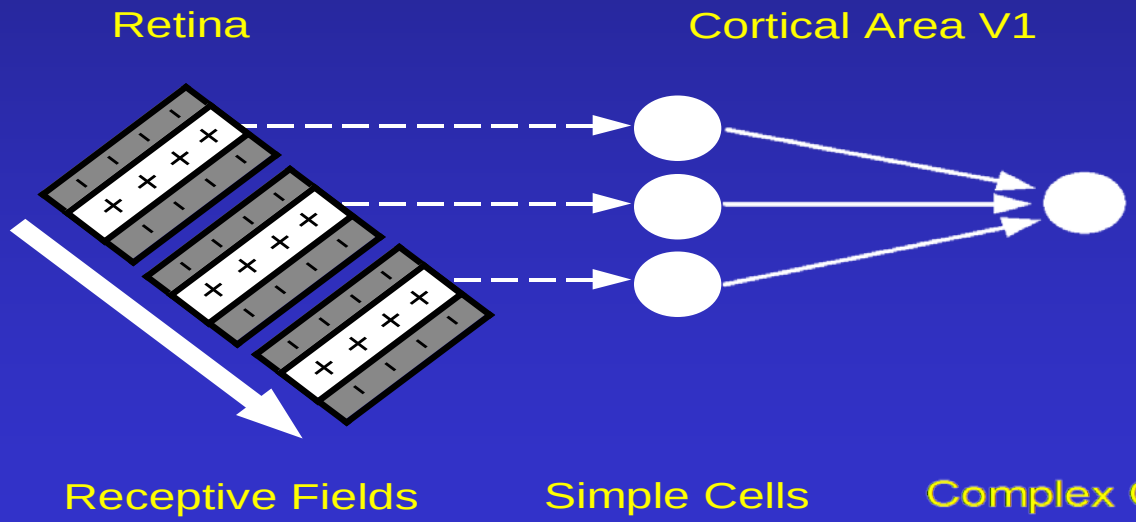
Cortical Receptive Fields

Complex Cells

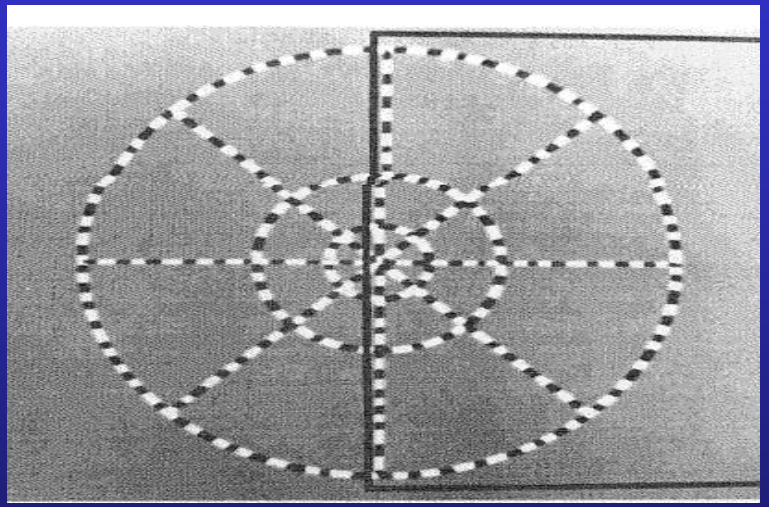


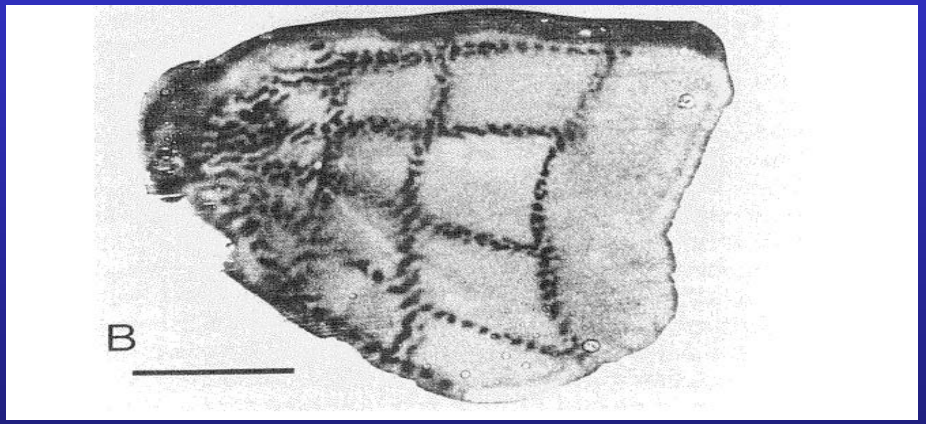
Cortical Receptive Fields

Constructing a Complex Cell



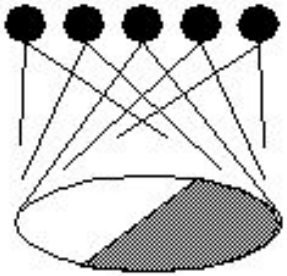
Mapping from Retina to V1



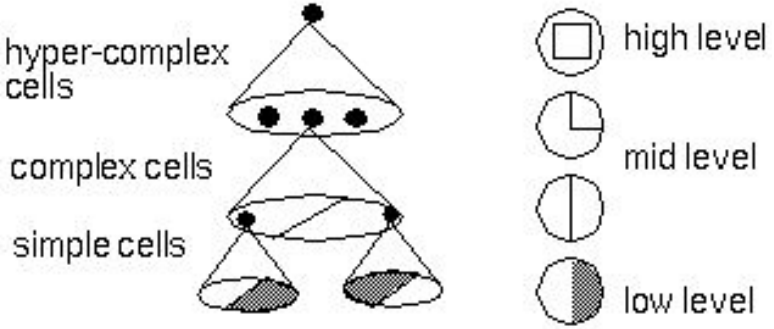


Hubel & Weisel

topographical mapping

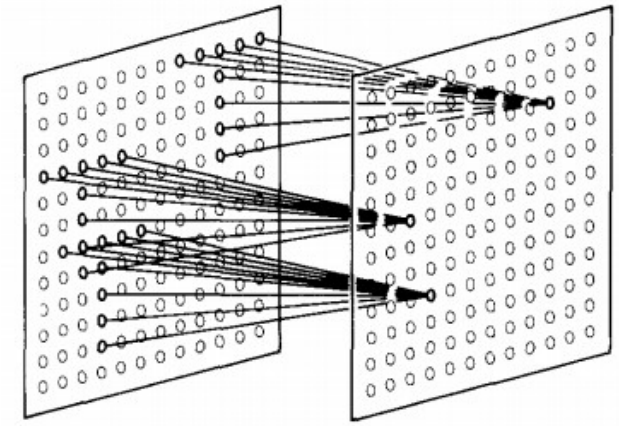


featural hierarchy

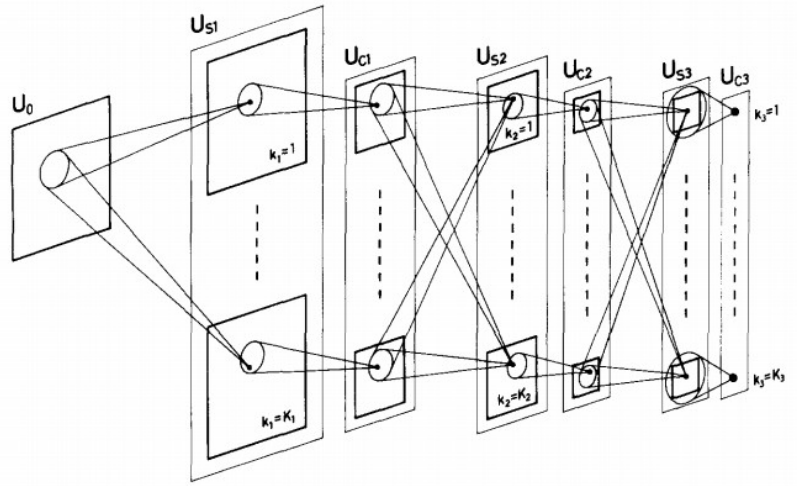


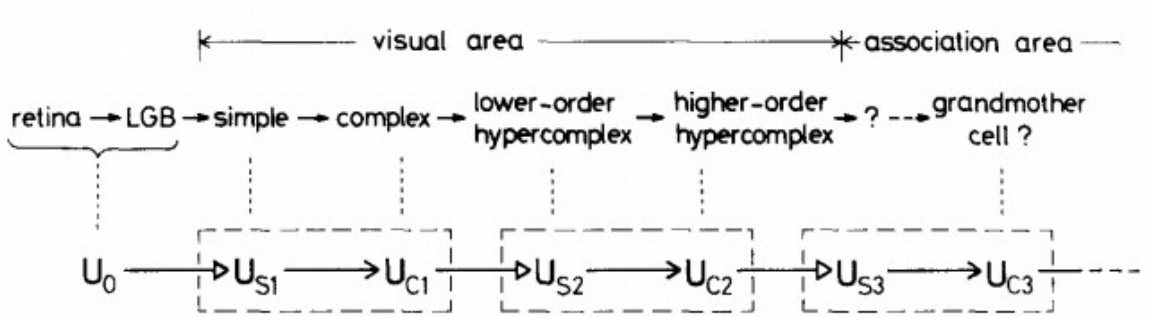
A bit of history:

Neurocognitron
[Fukushima 1980]



"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

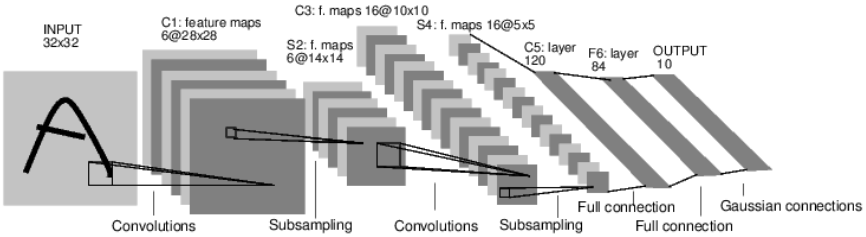




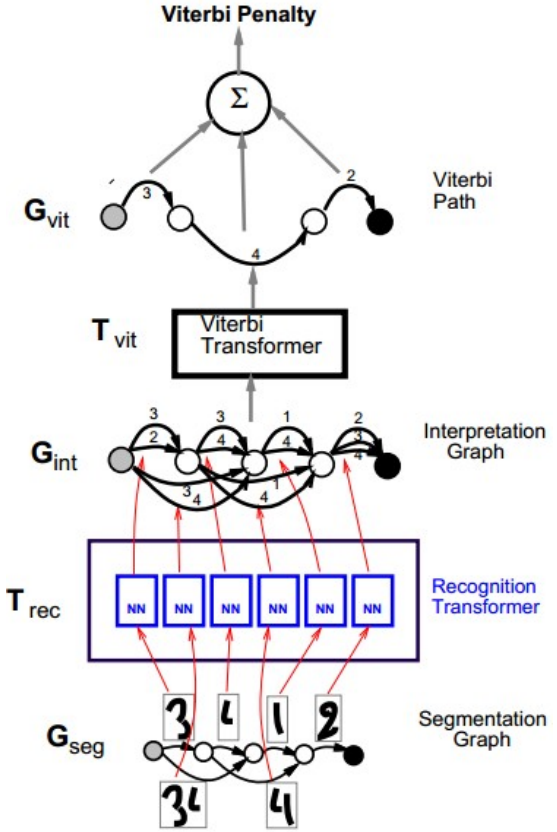
A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

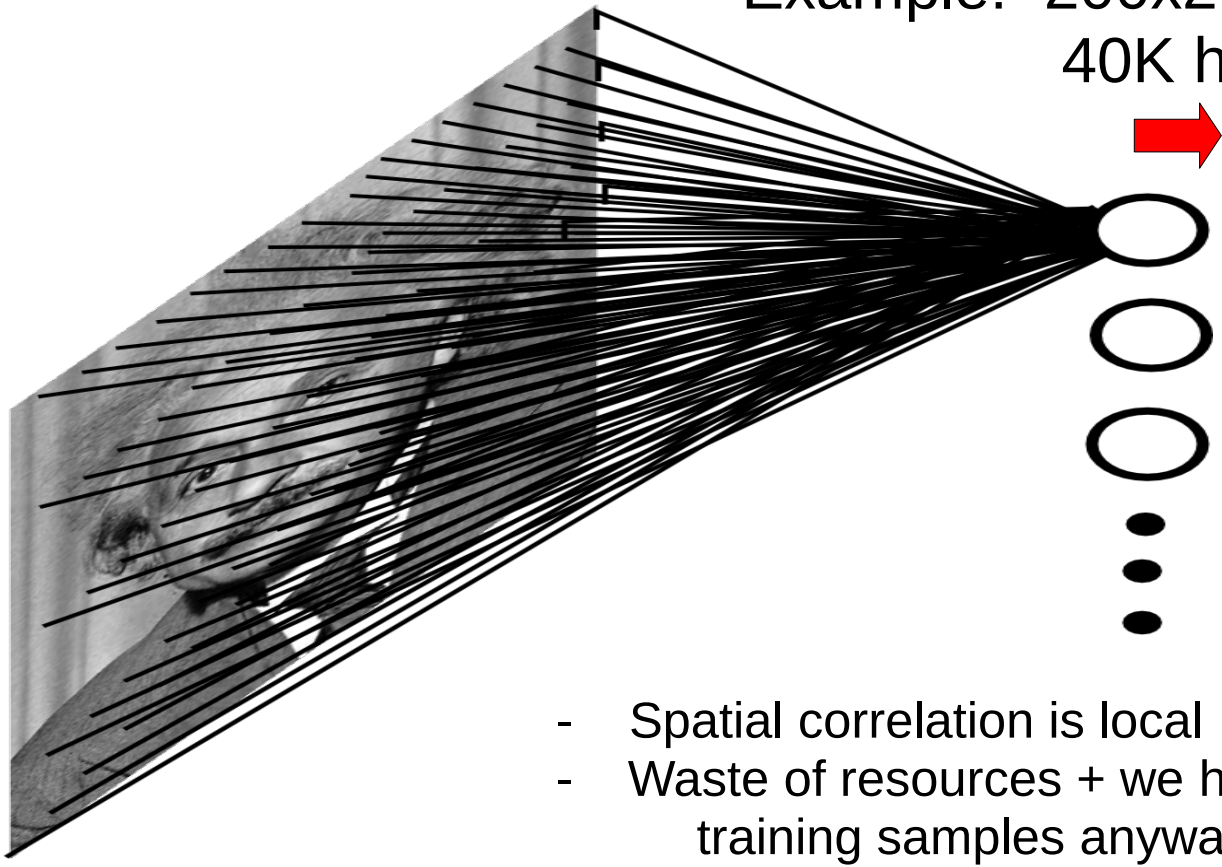


LeNet-5



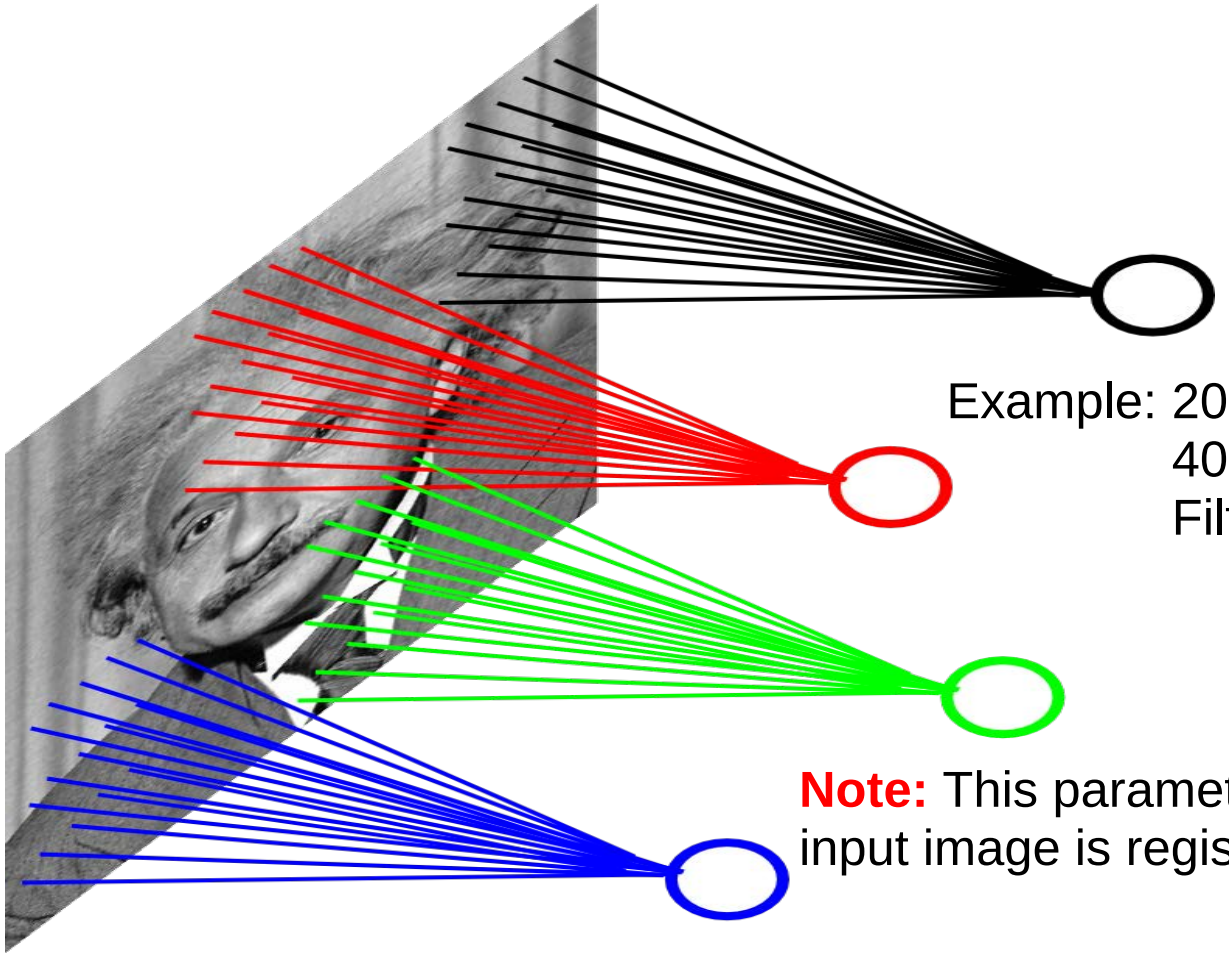
Fully Connected Layer

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



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

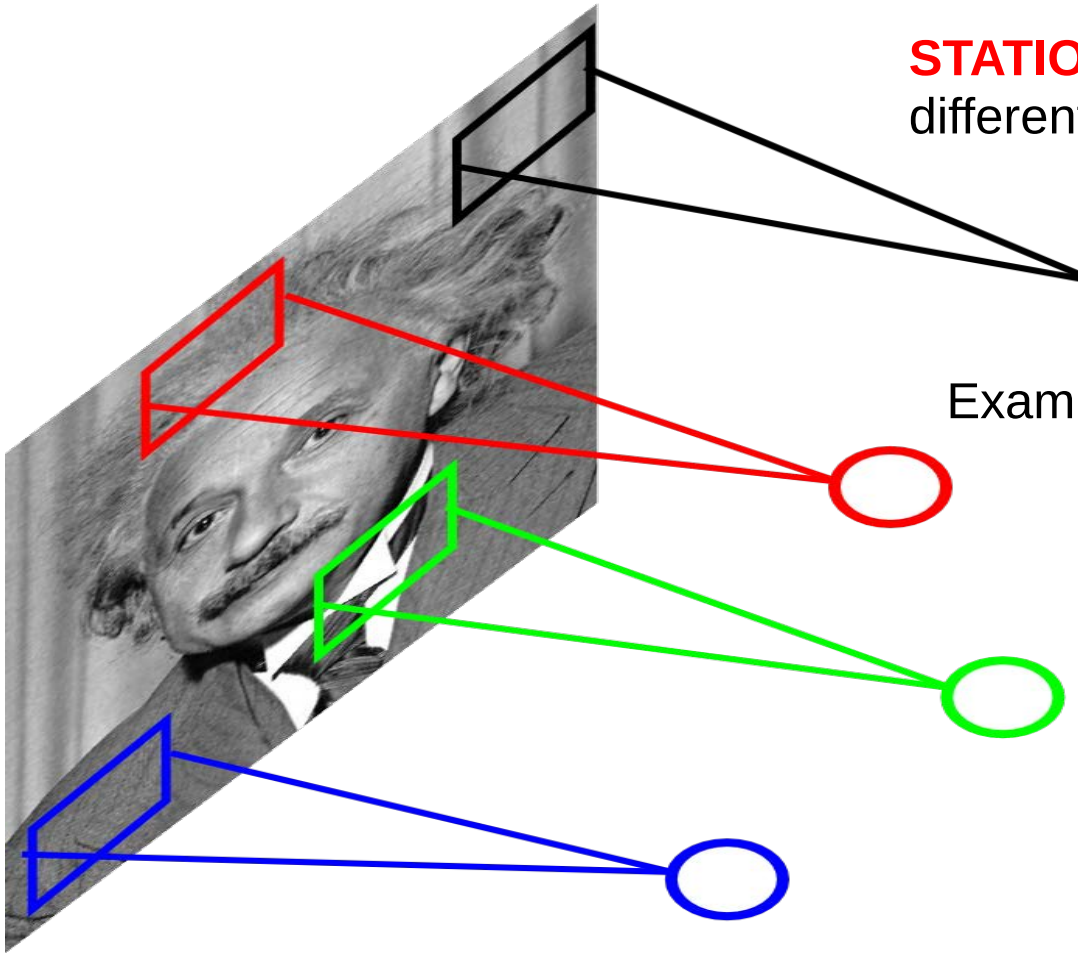
Locally Connected Layer



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).

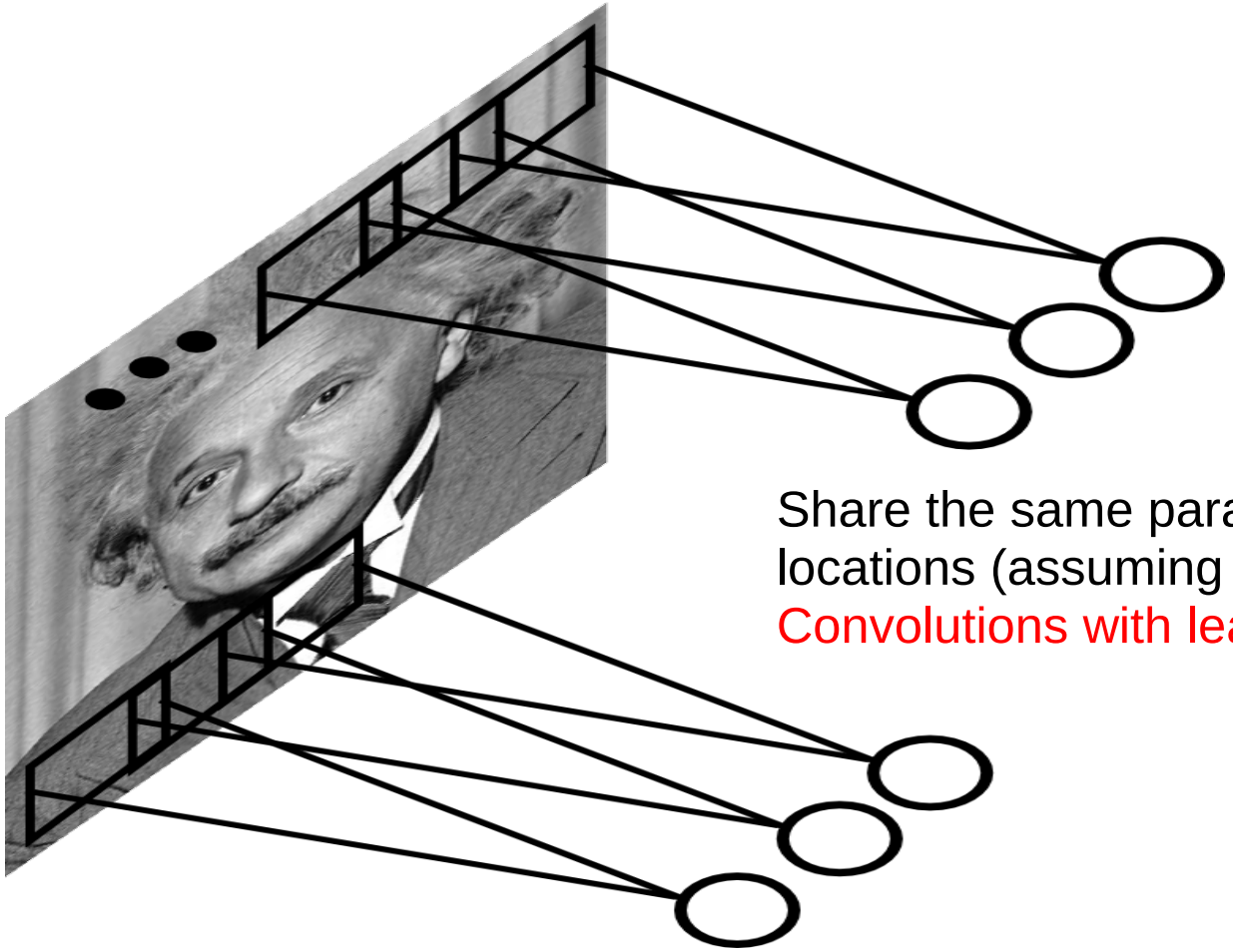
Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

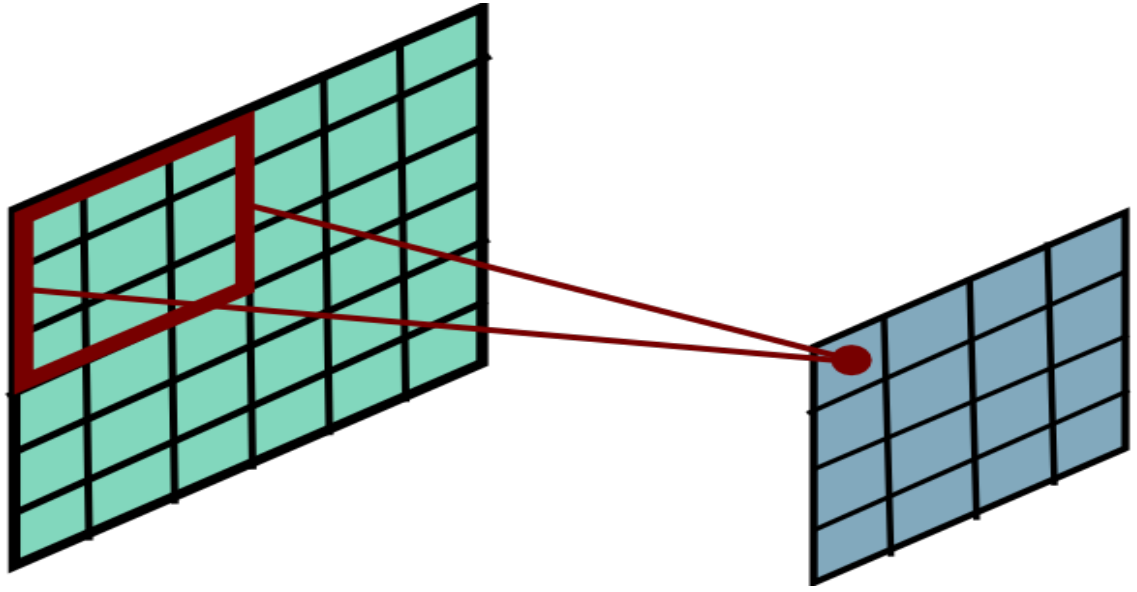
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Convolutional Layer

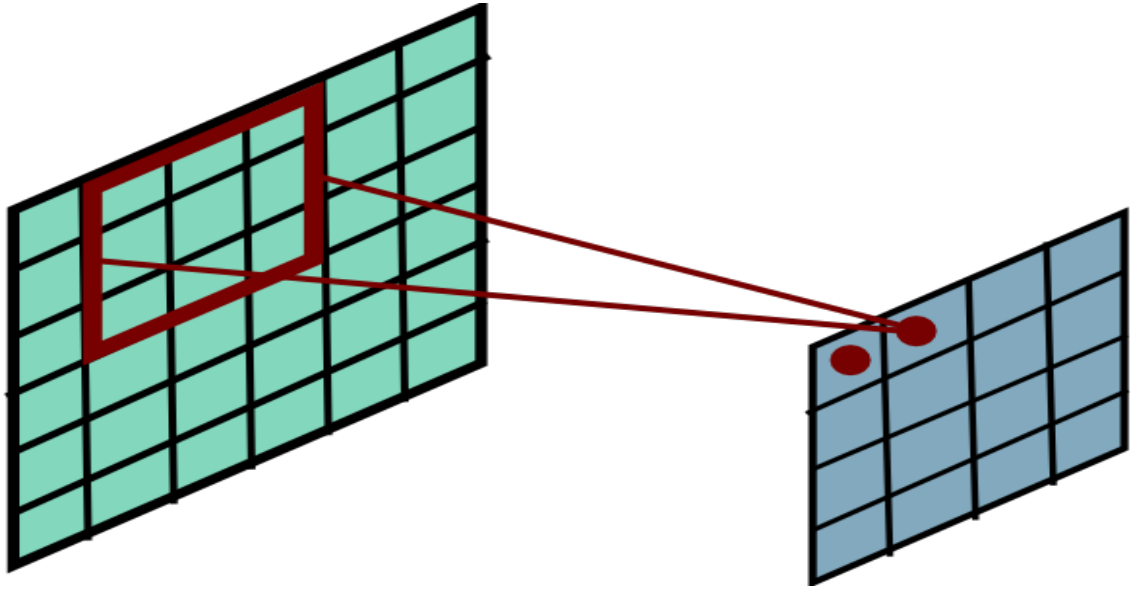


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

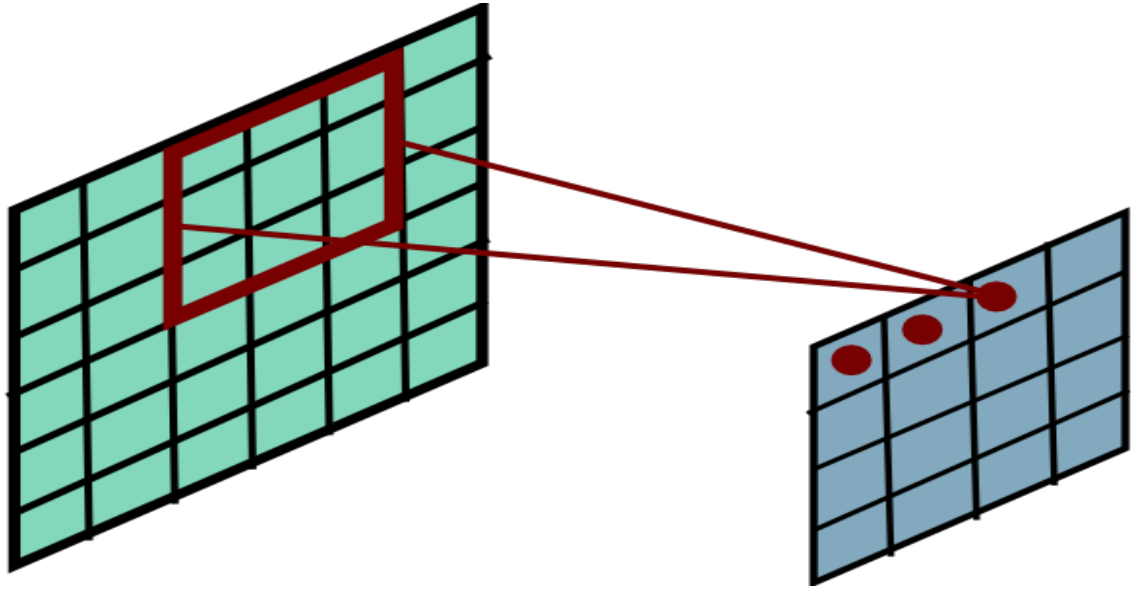
Convolutional Layer



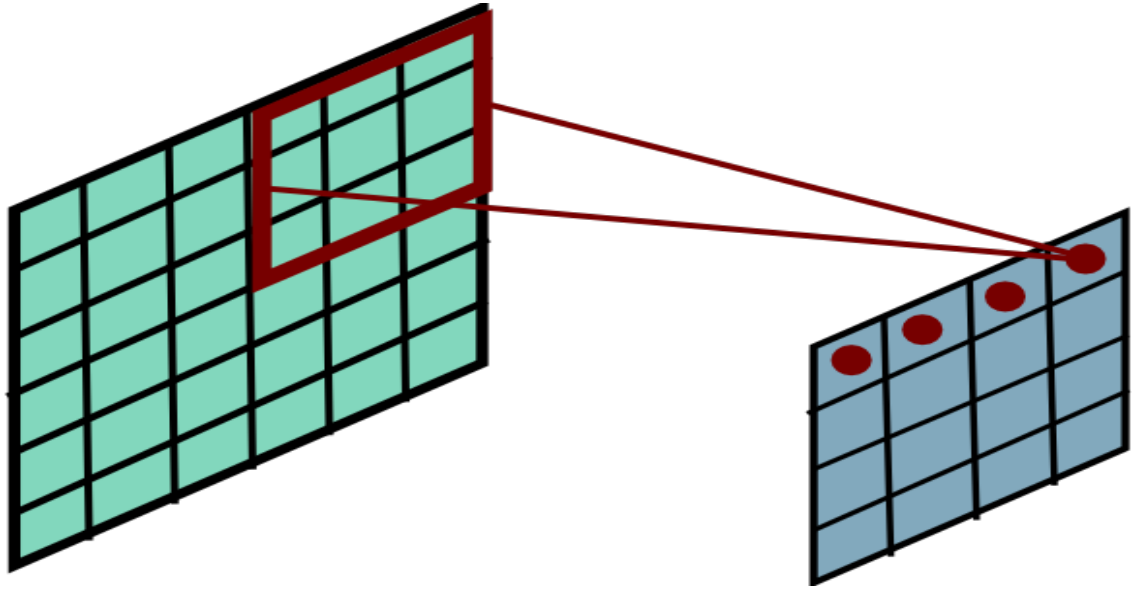
Convolutional Layer



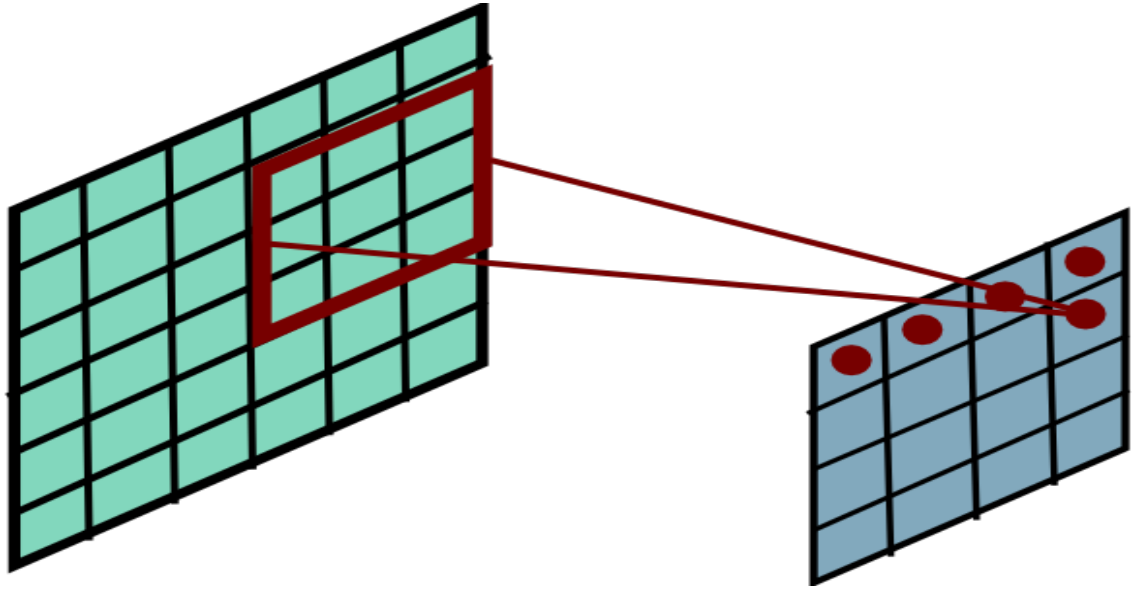
Convolutional Layer



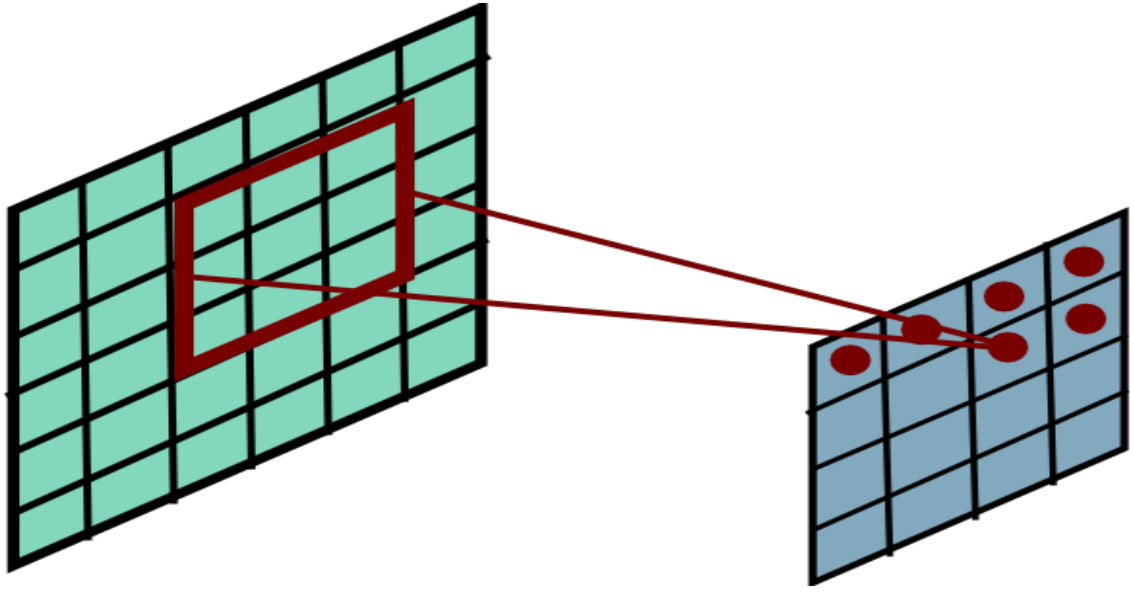
Convolutional Layer



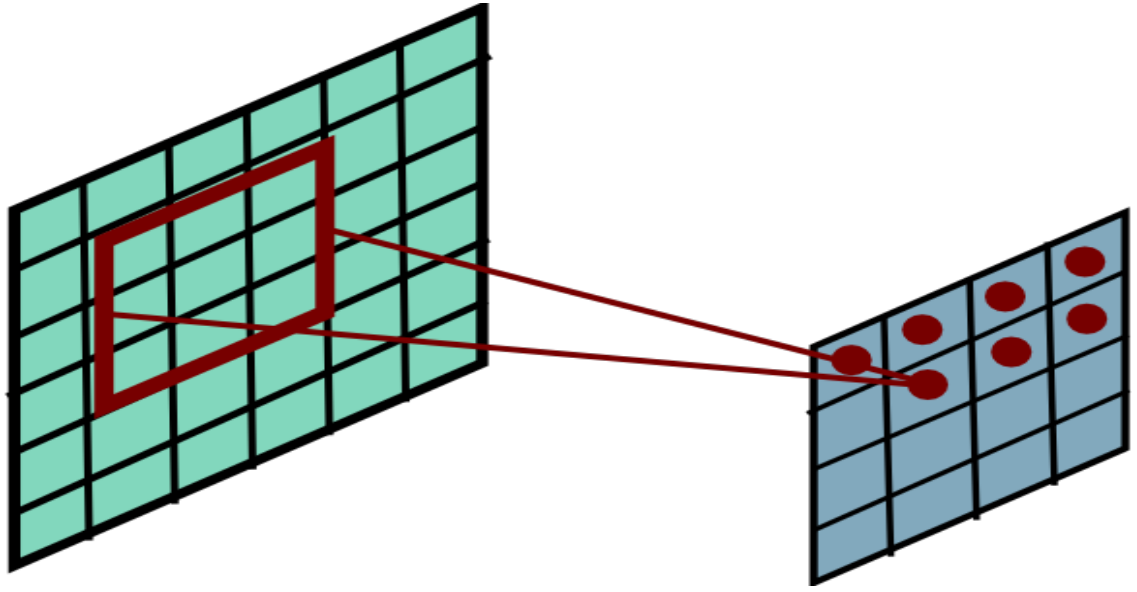
Convolutional Layer



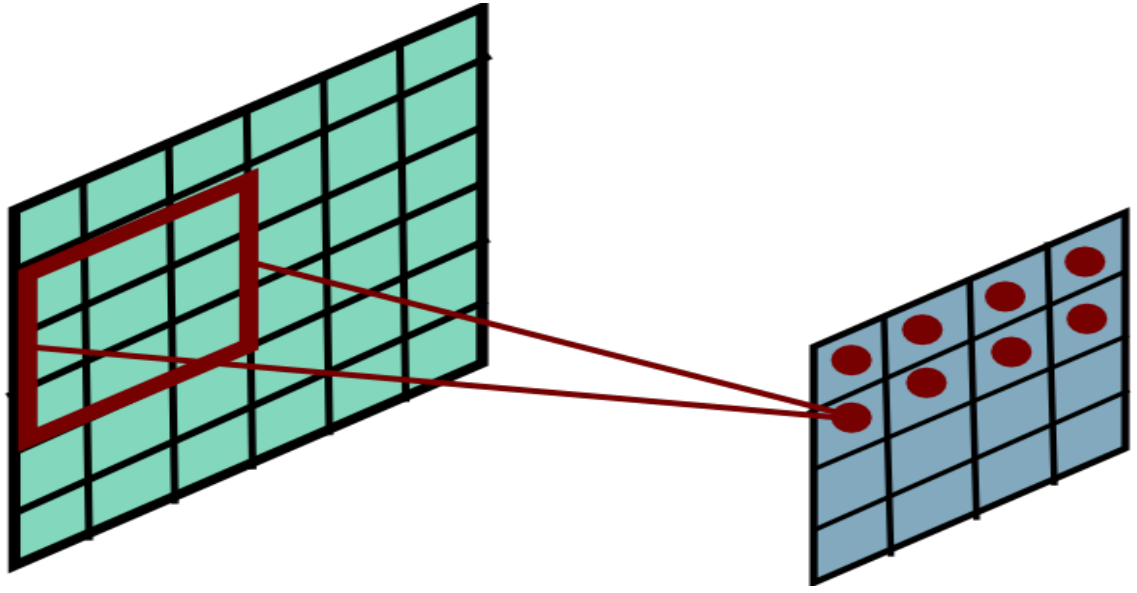
Convolutional Layer



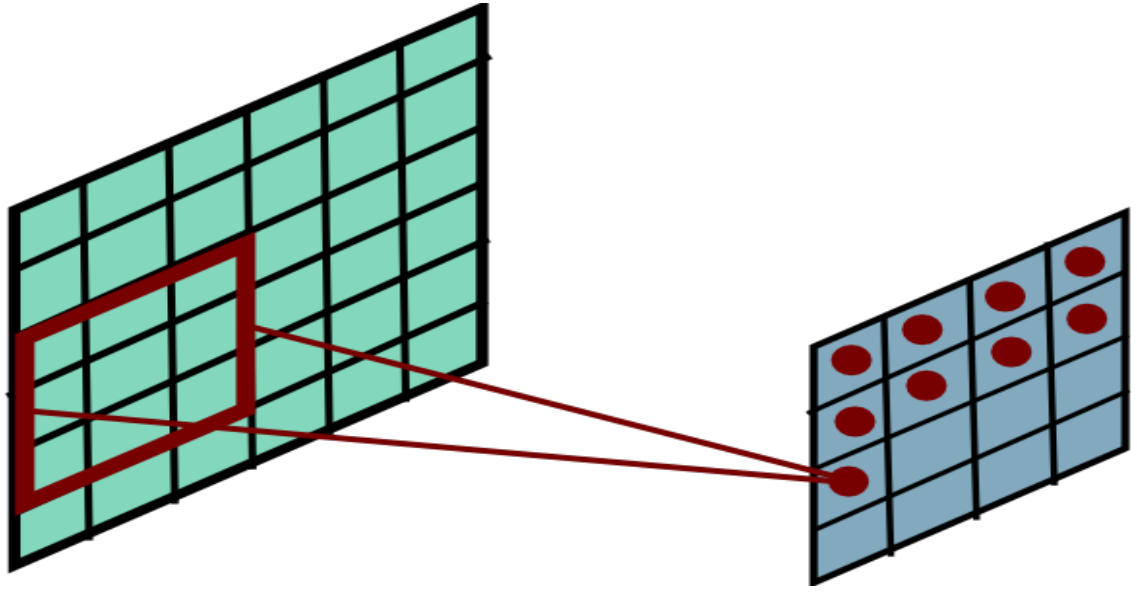
Convolutional Layer



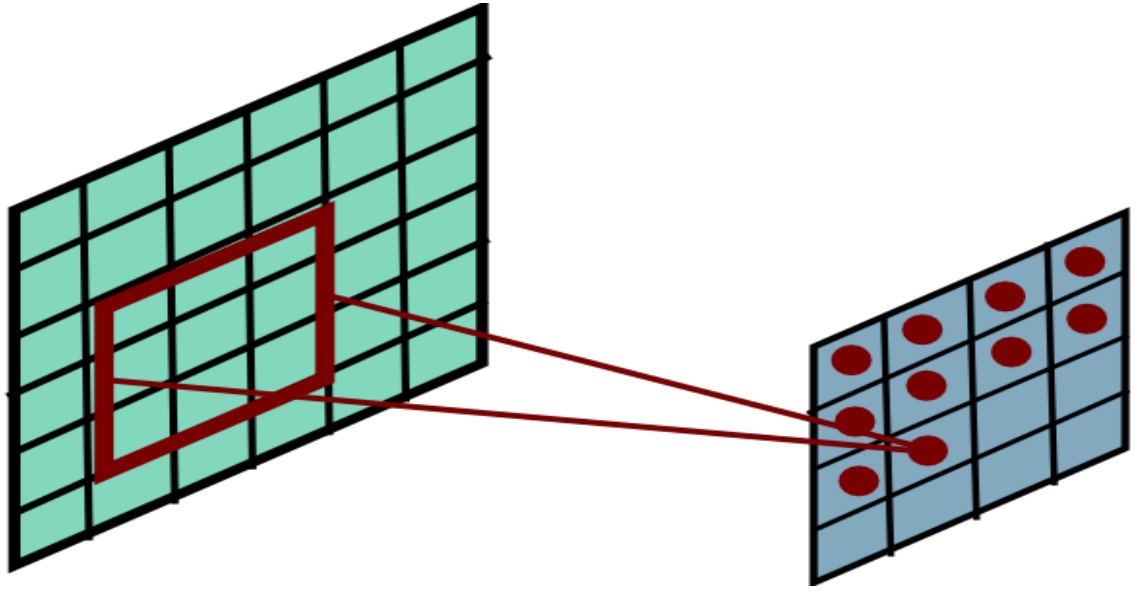
Convolutional Layer



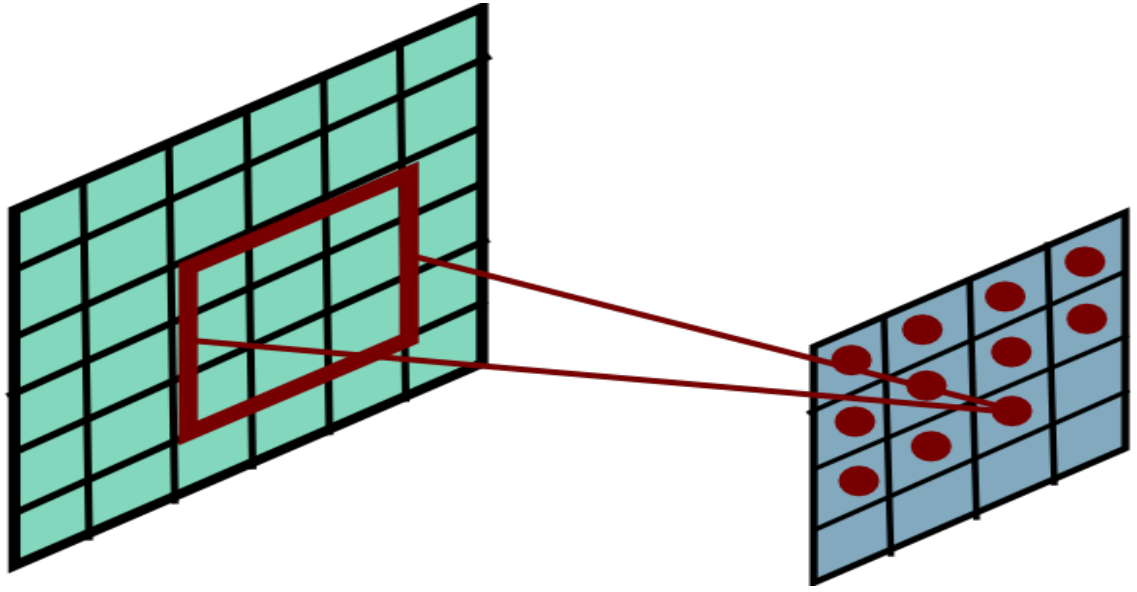
Convolutional Layer



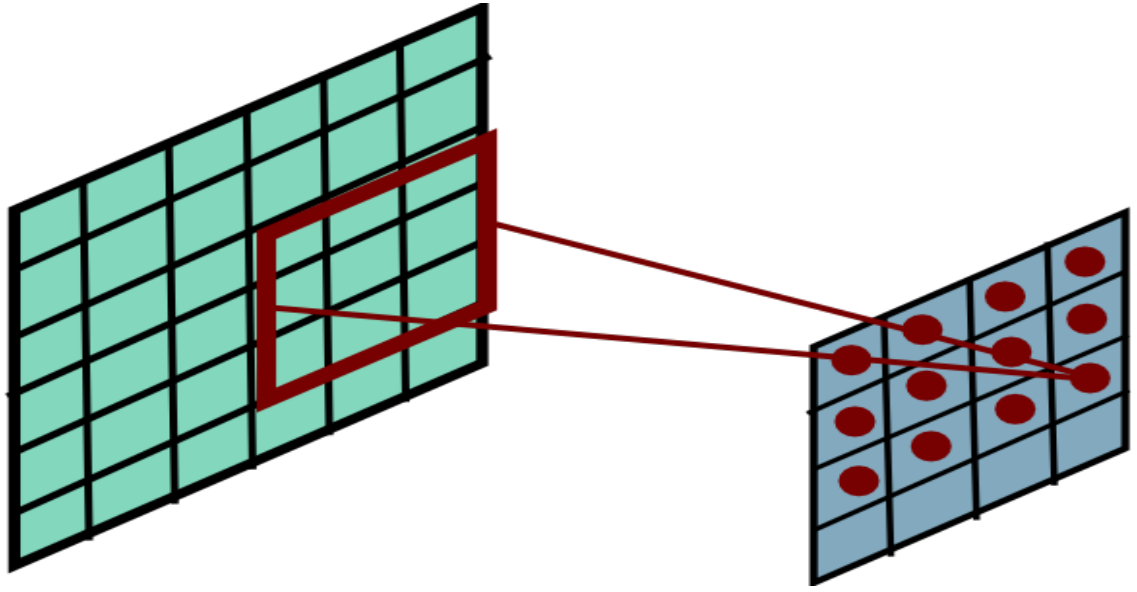
Convolutional Layer



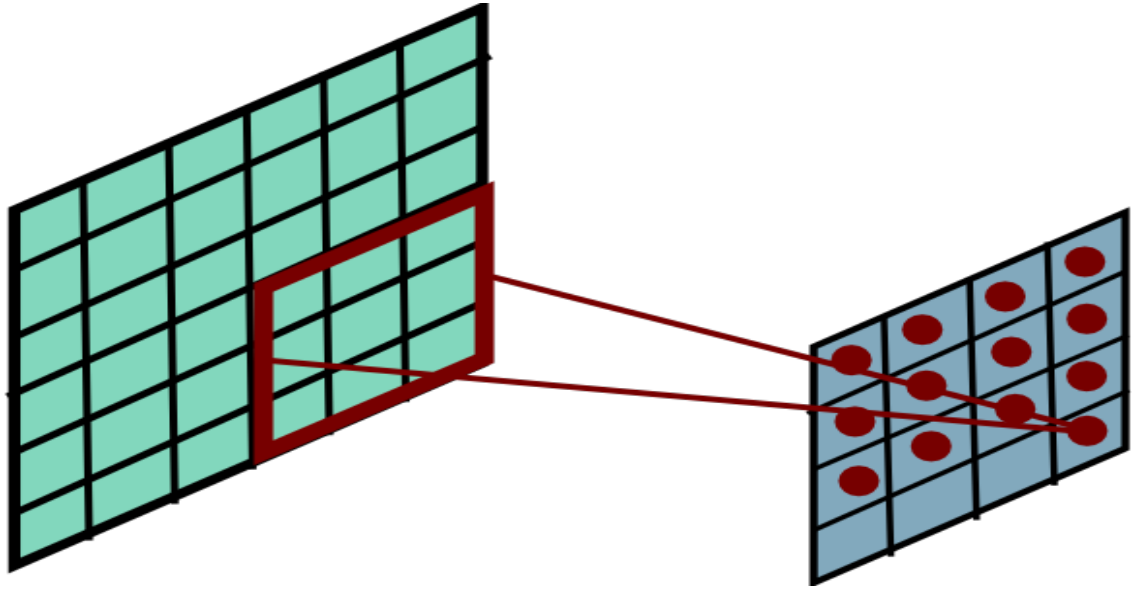
Convolutional Layer



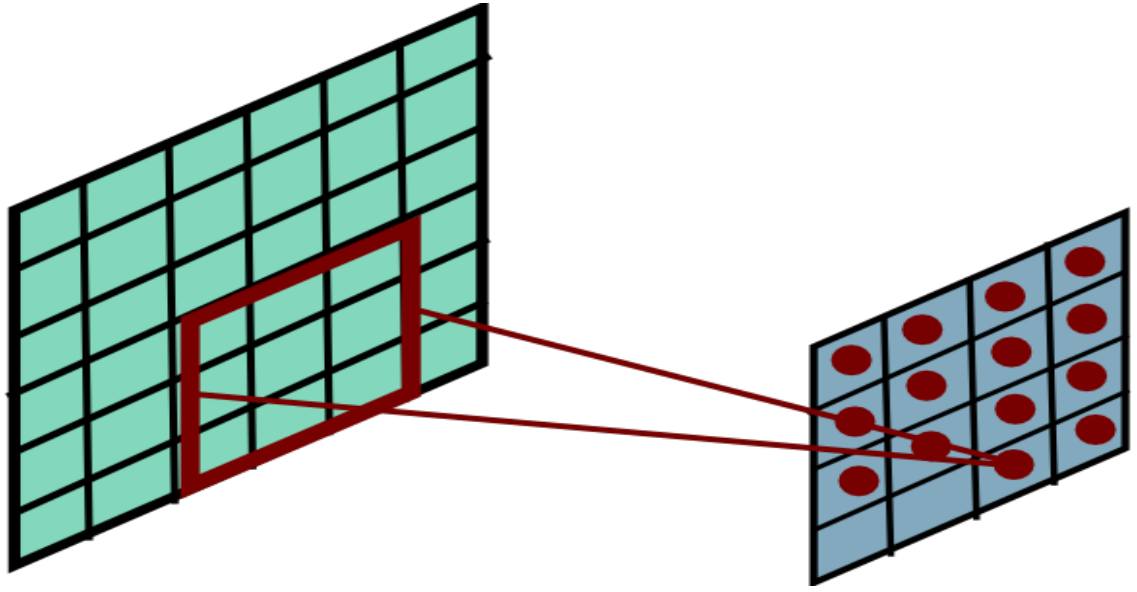
Convolutional Layer



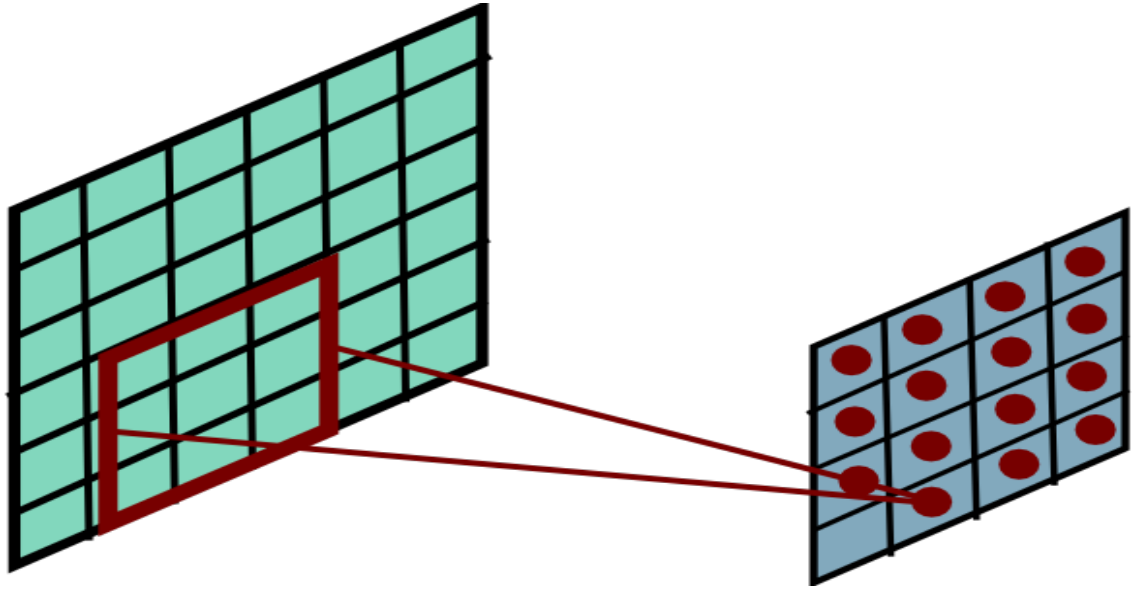
Convolutional Layer



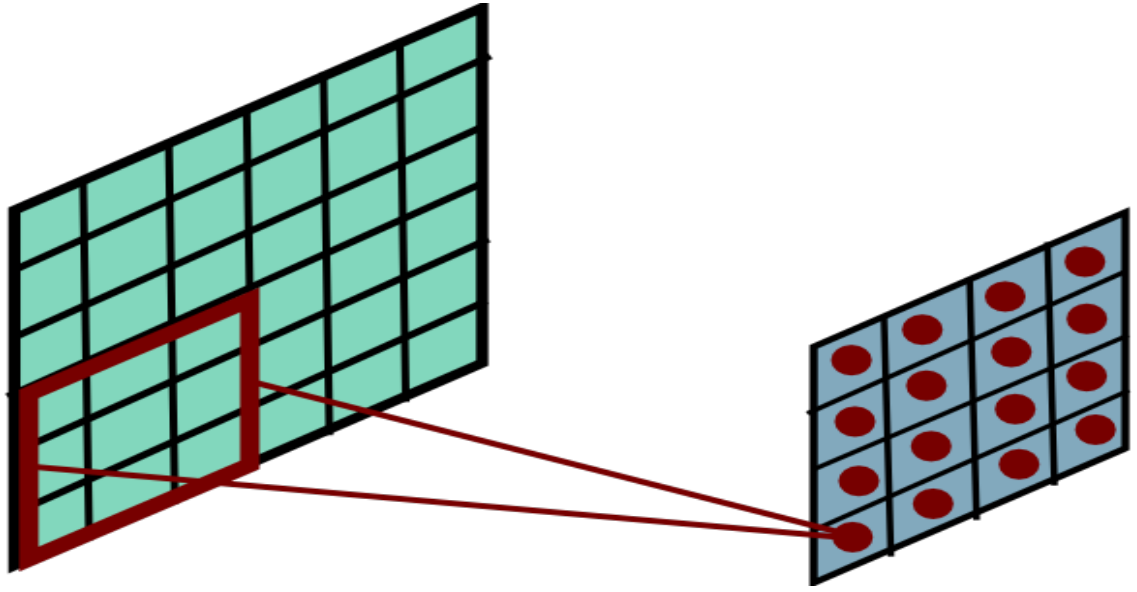
Convolutional Layer



Convolutional Layer

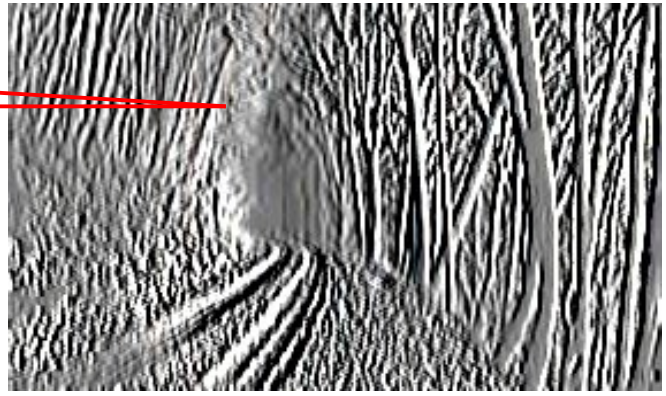


Convolutional Layer



Convolutional of Two Signals

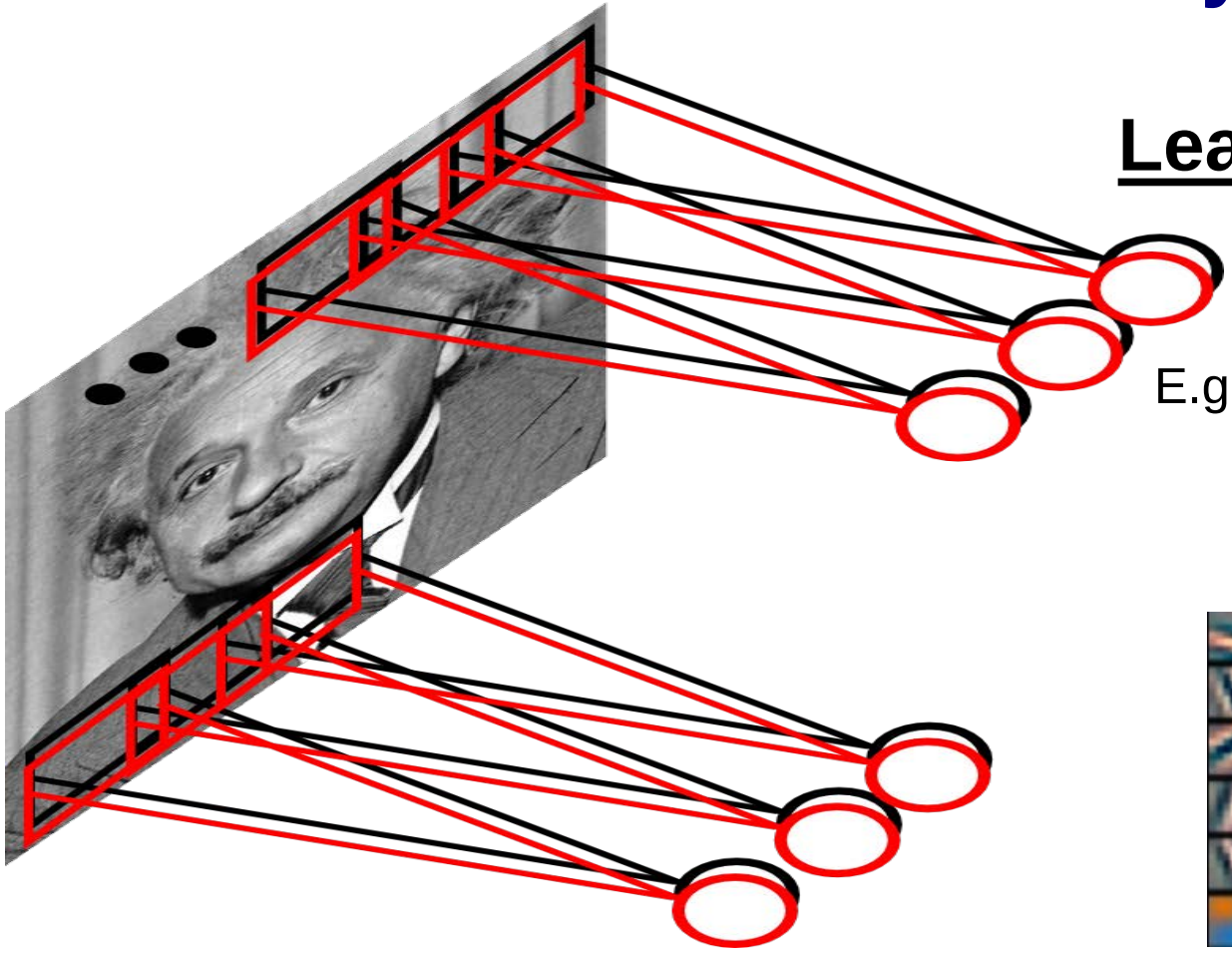
$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Convolutional Layer



Learn multiple filters.

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

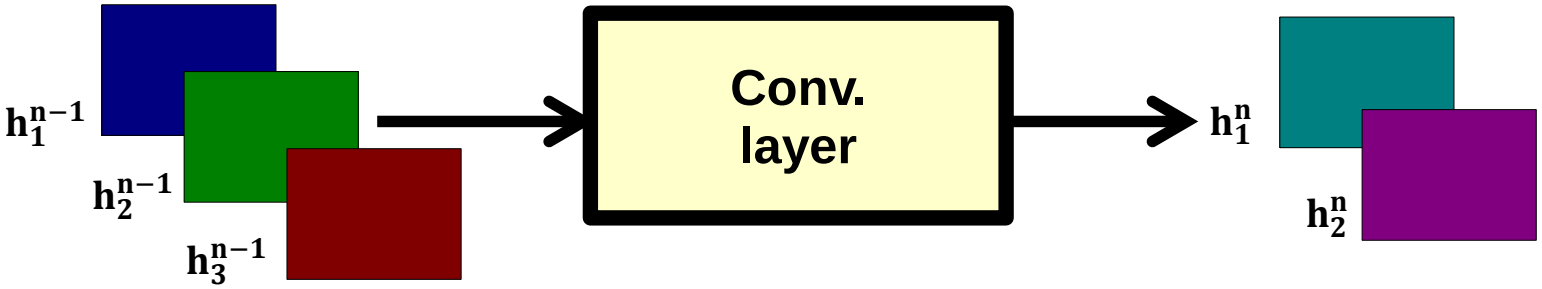
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



Ranzato

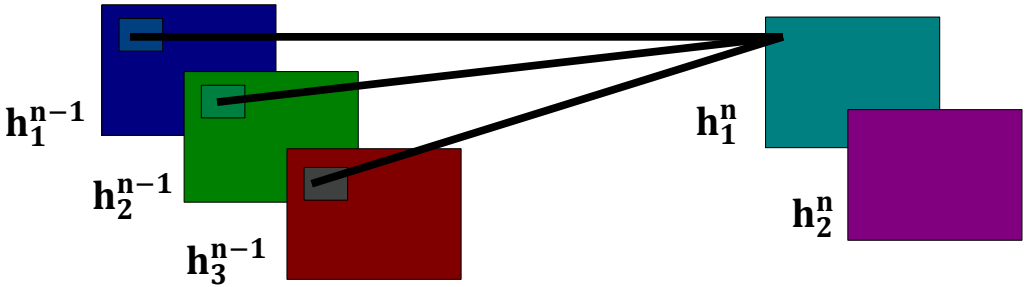
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



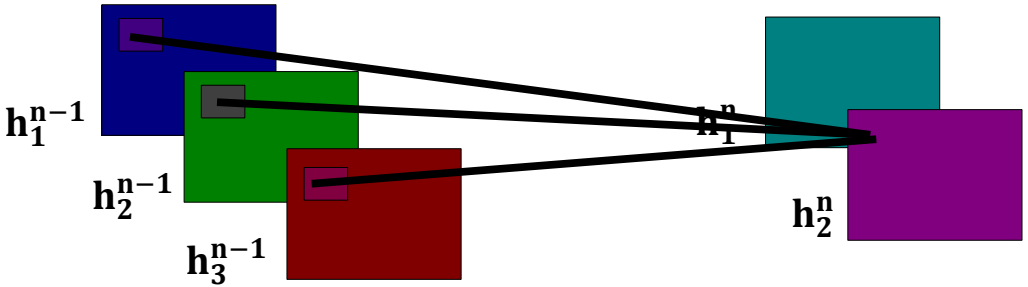
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

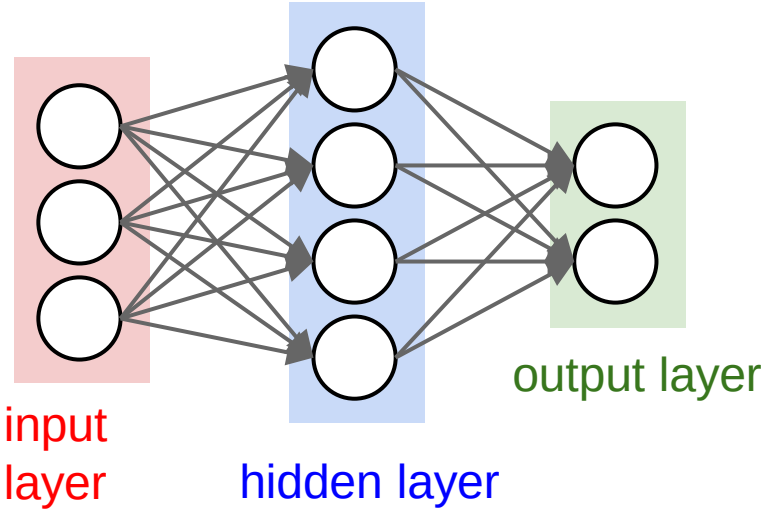
output feature map

input feature map

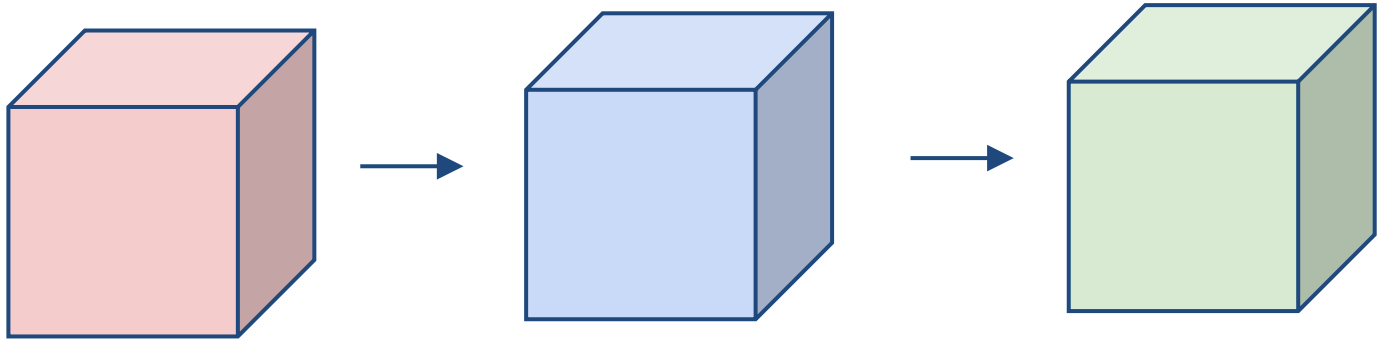
kernel



before:

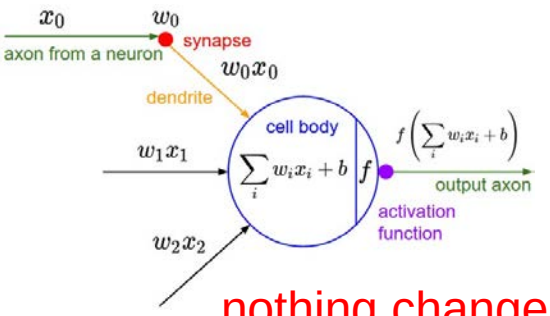


now:

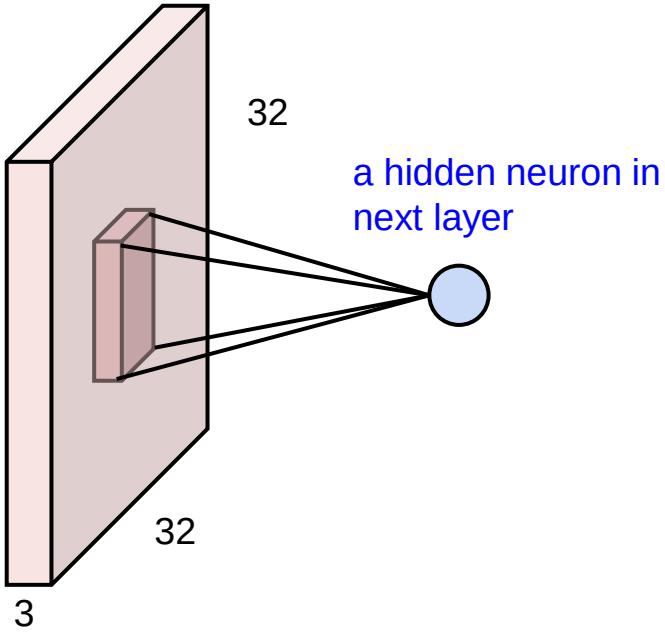


Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

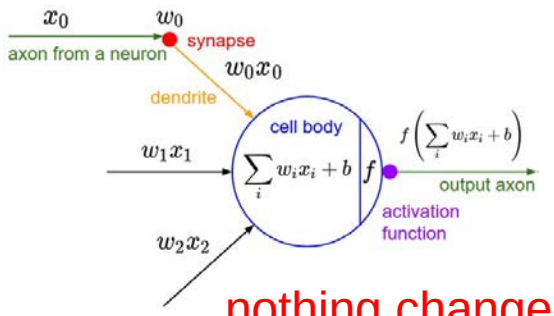


nothing changes



Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

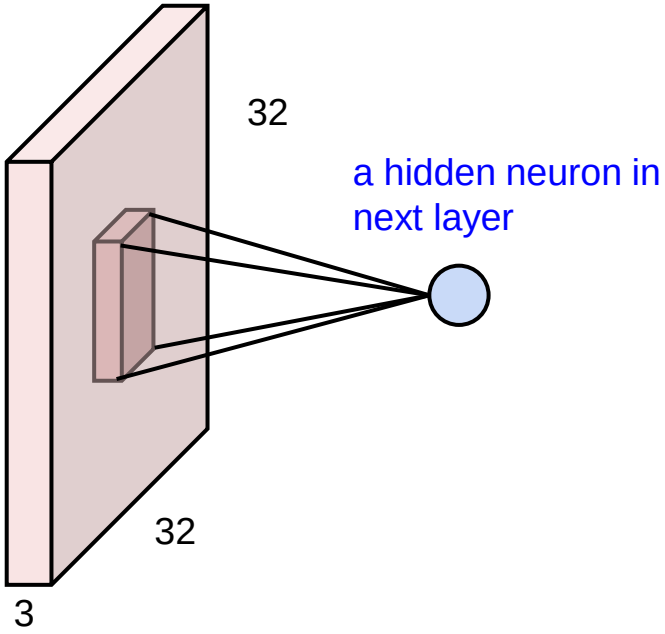


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

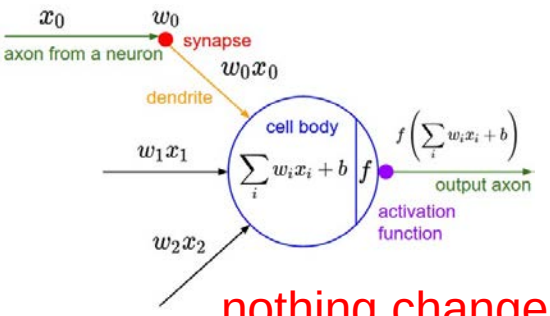
now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

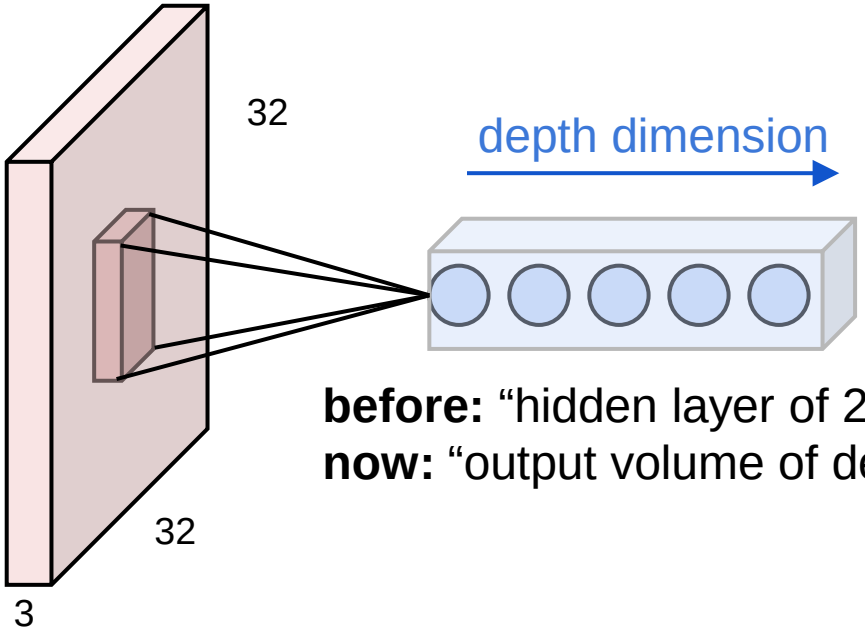
- local in space (5x5 inside 32x32)
- but full in depth (all 3 depth channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



nothing changes

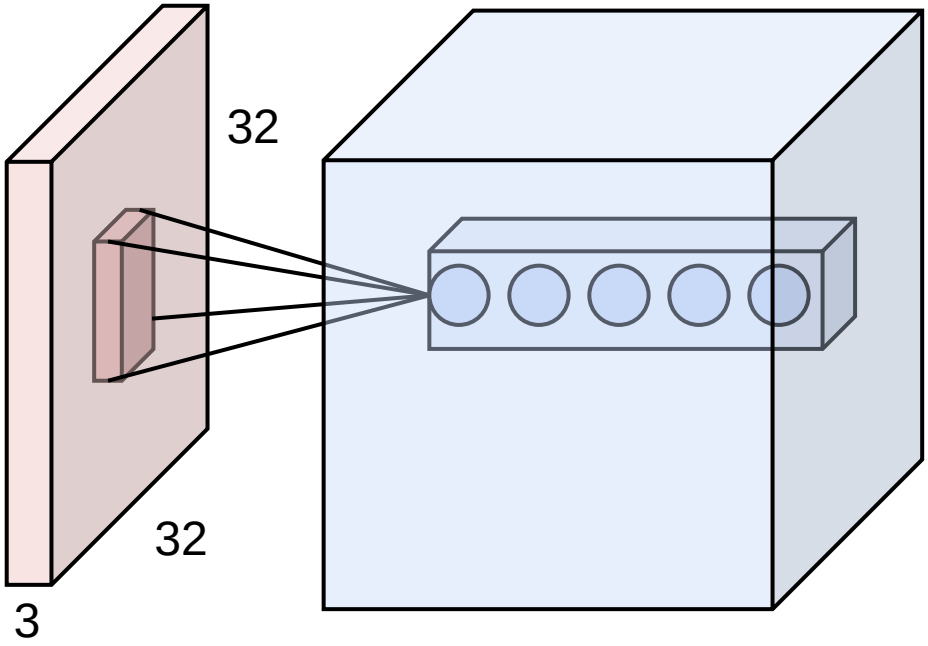


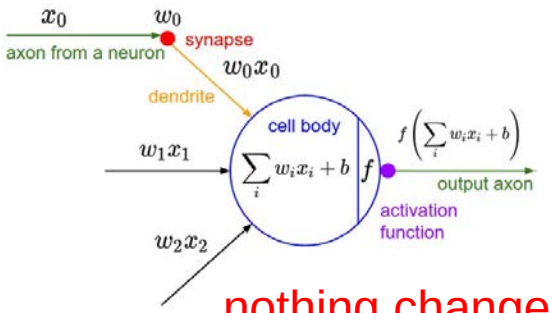
before: “hidden layer of 200 neurons”
now: “output volume of depth 200”

Multiple neurons all looking at the same region of the input volume, stacked along depth.

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

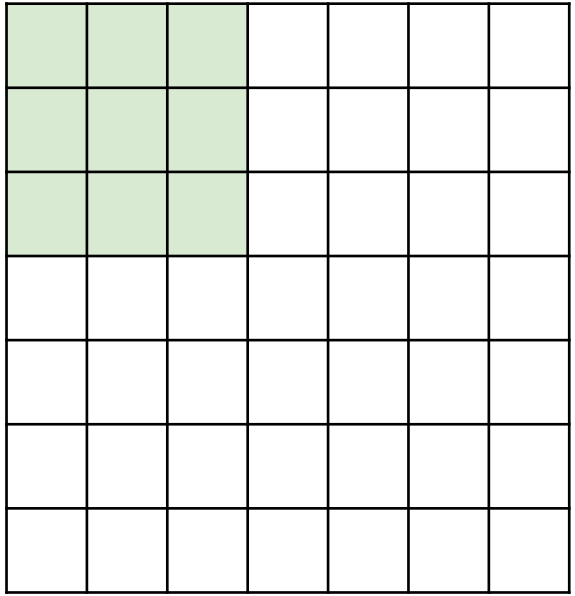




nothing changes

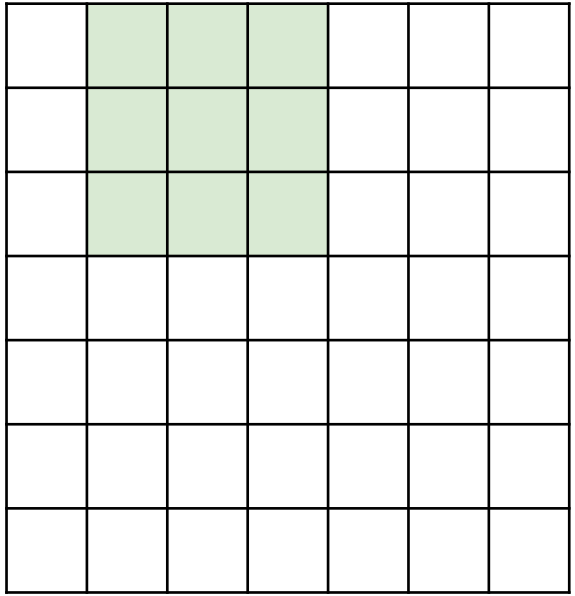
These form a single [1 x 1 x depth] “depth column” in the output volume

Replicate this column of hidden neurons across space, with some **stride.**



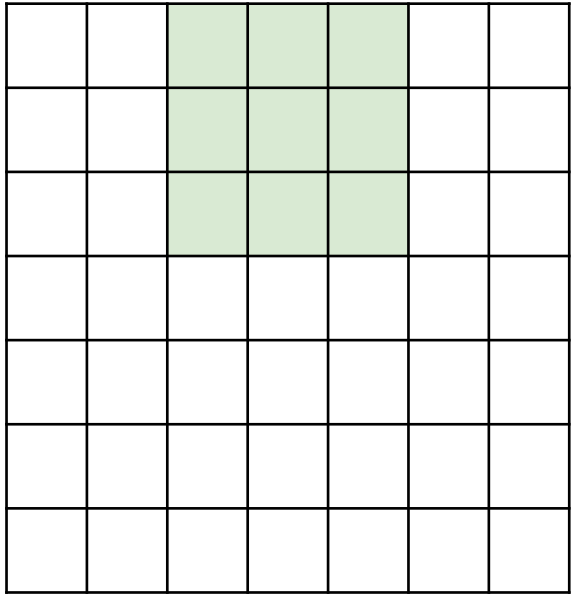
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



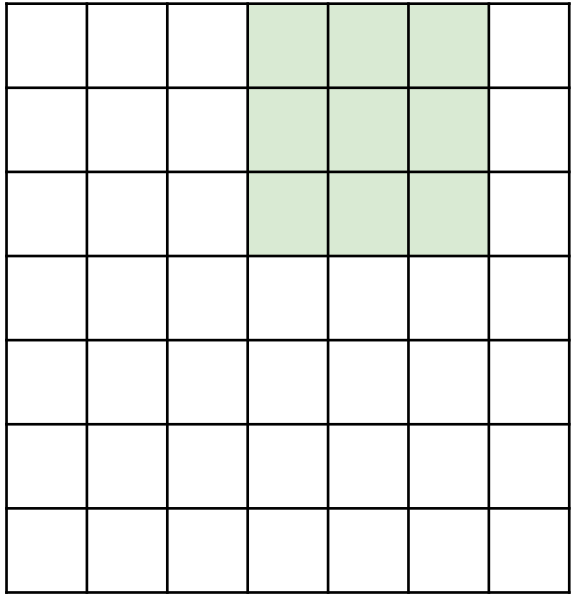
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



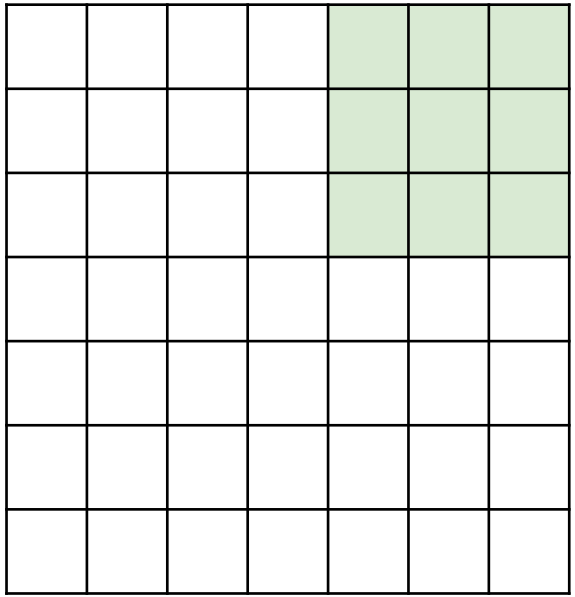
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



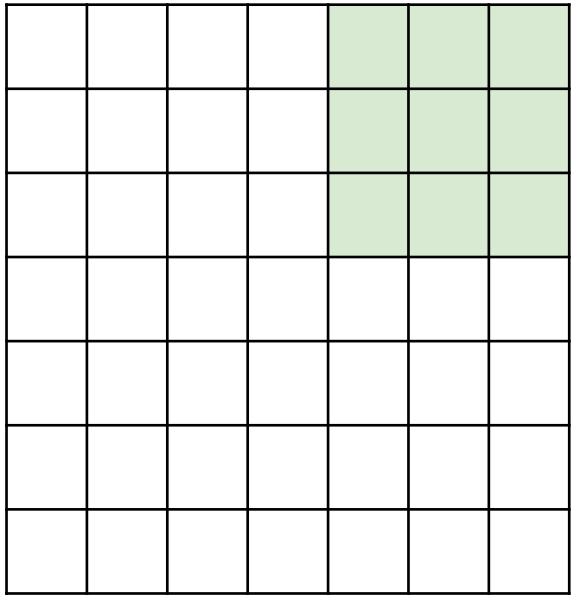
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



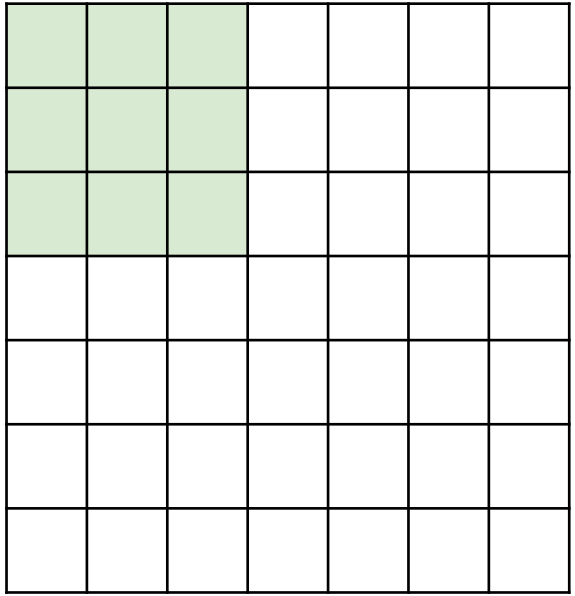
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

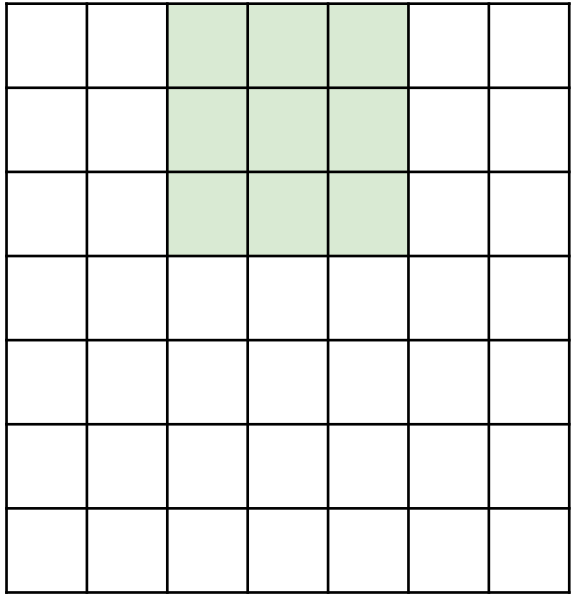
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

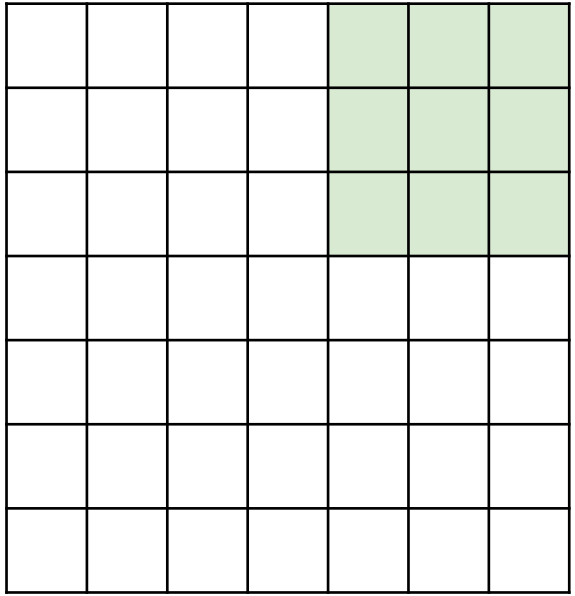
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

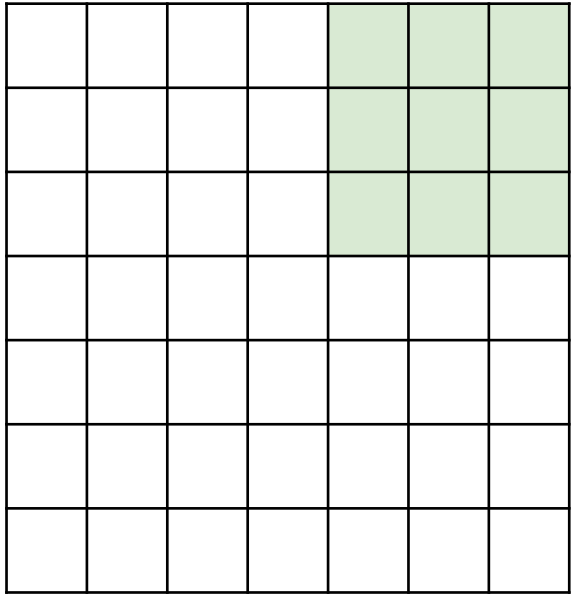
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

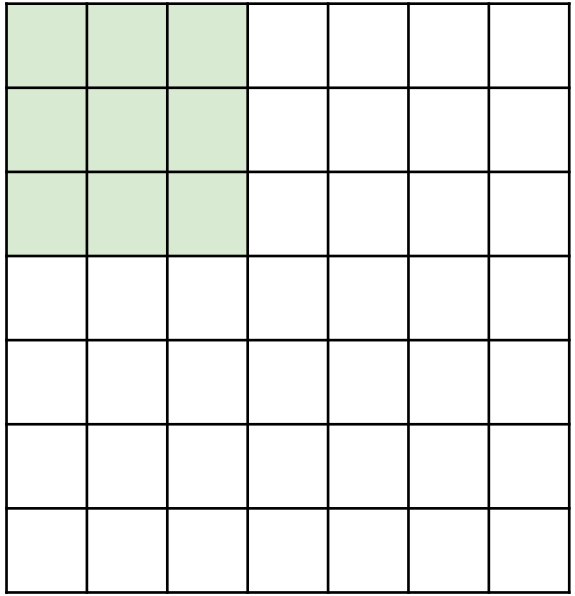
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

Replicate this column of hidden neurons across space, with some **stride.**

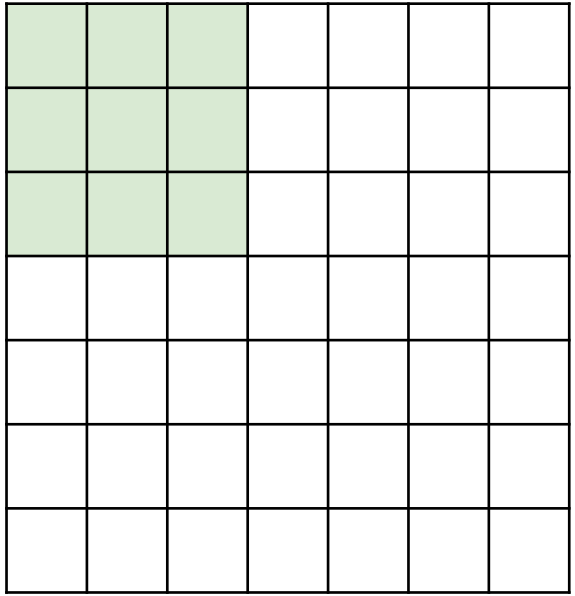


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3?

Replicate this column of hidden neurons across space, with some **stride.**



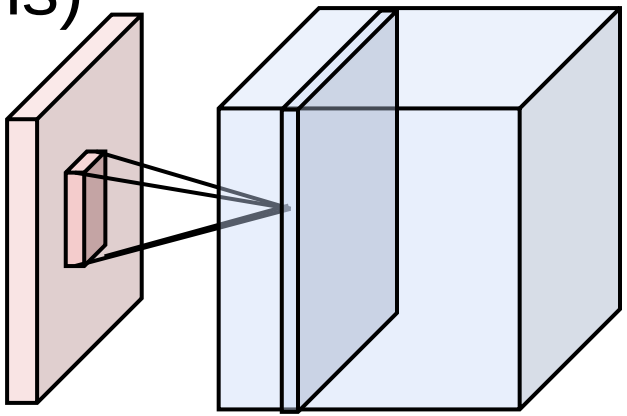
7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3? Cannot.

Convolutional Layers

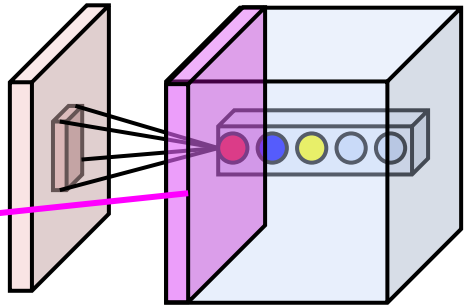
1. Connect neurons only to local receptive fields
2. Share weights: use the *same neuron weight parameters* for neurons in each “depth slice” (i.e. across spatial positions)



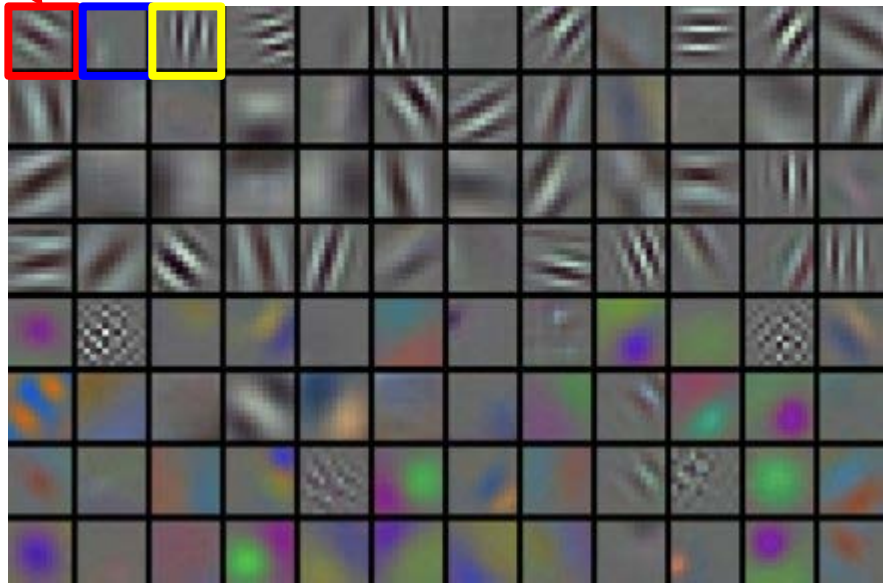
one activation map (a depth slice), computed with one set of weights

Activations:

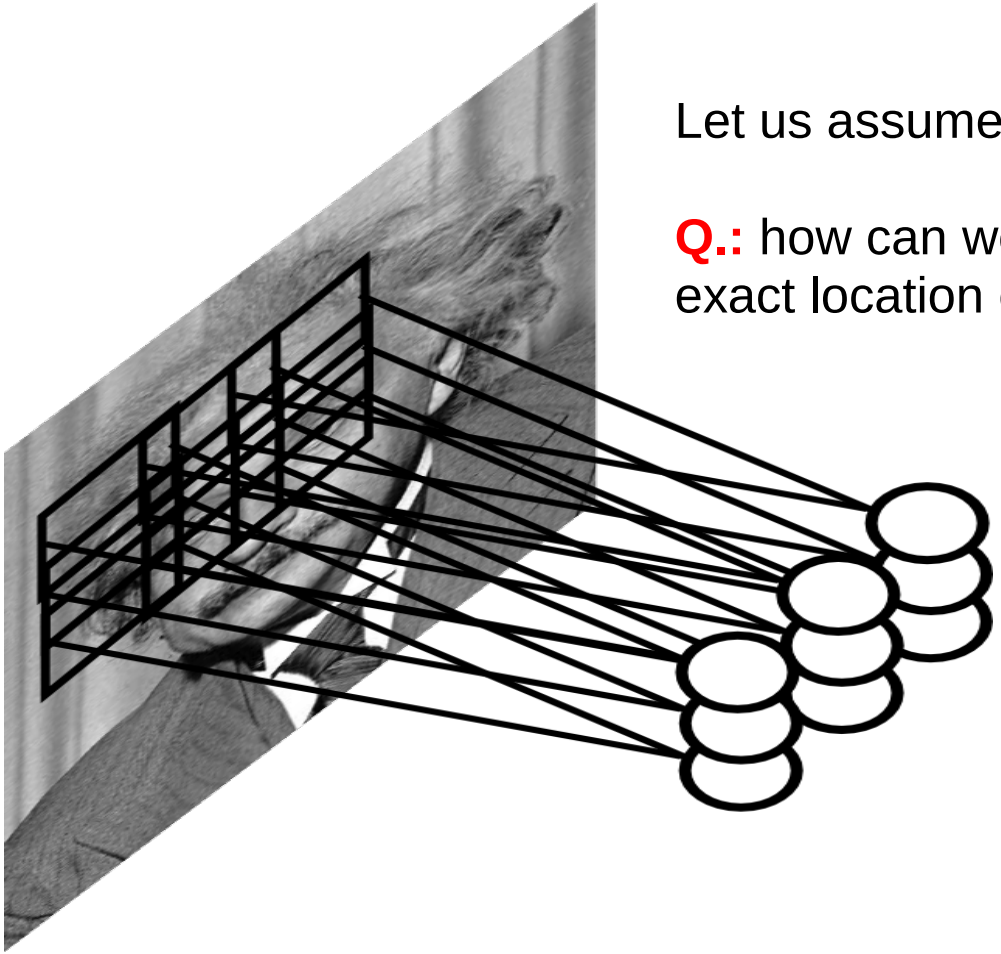
convolving the first filter in the input gives the first slice of depth in output volume



Activations:



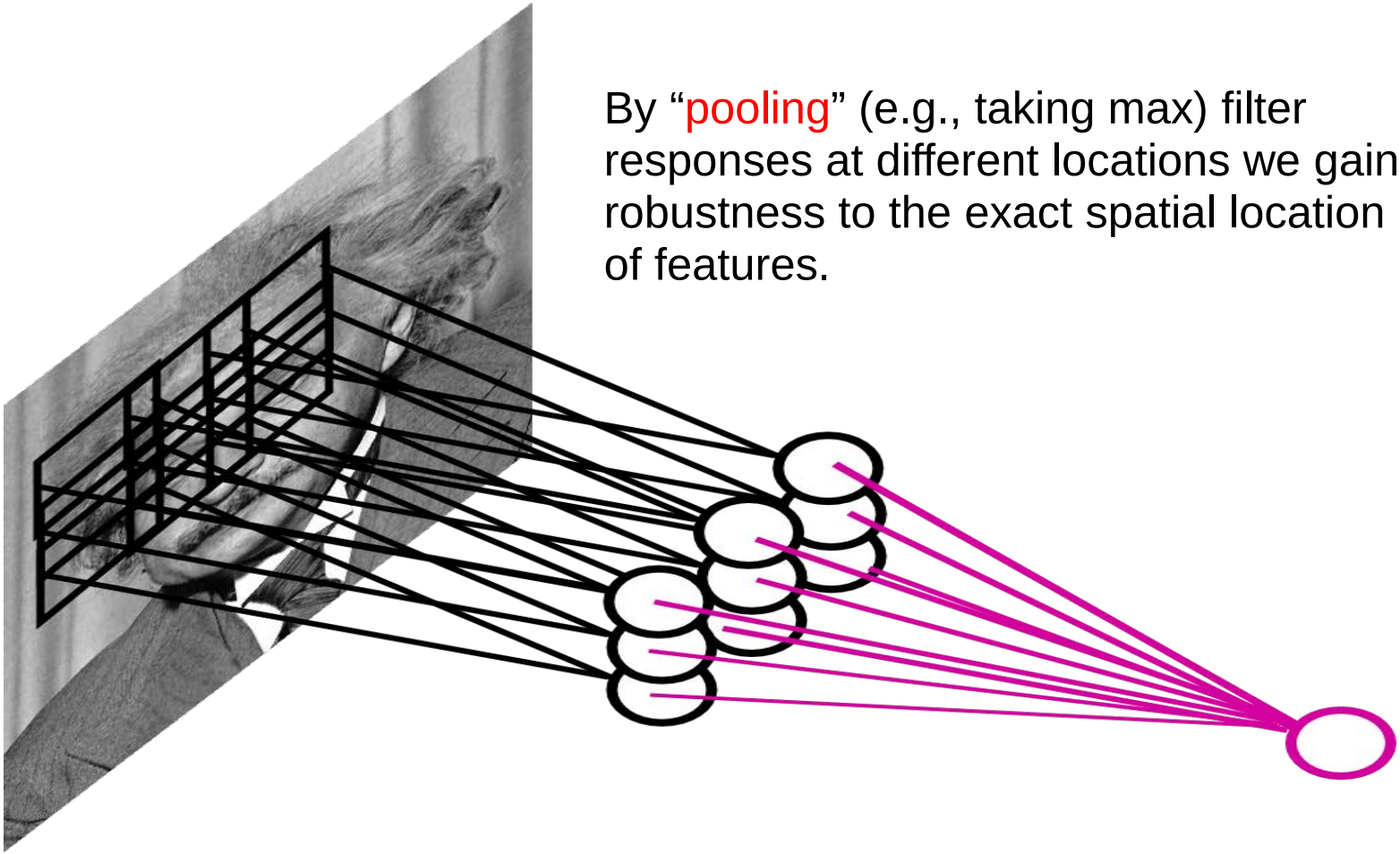
Pooling Layer



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.

Pooling Layer: Examples

Max-pooling:

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

Average-pooling:

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

L2-pooling:

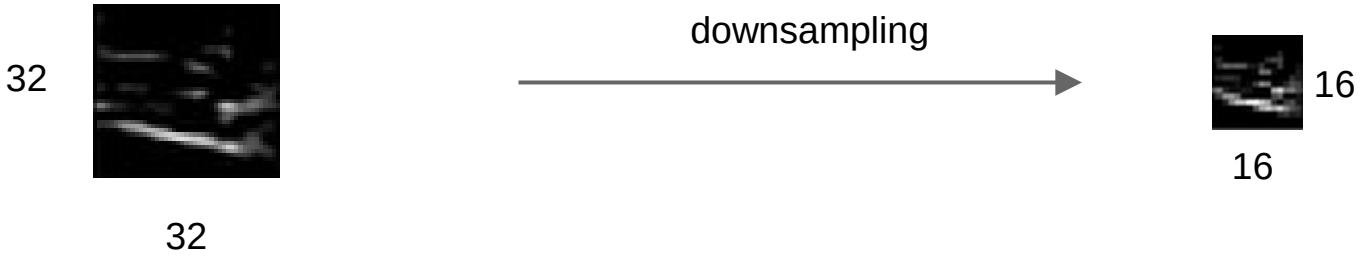
$$h_j^n(x,y) = \sqrt{\sum_{\tilde{x}\in N(x),\tilde{y}\in N(y)} h_j^{n-1}\left(\tilde{x},\tilde{y}\right)^2}$$

L2-pooling over features:

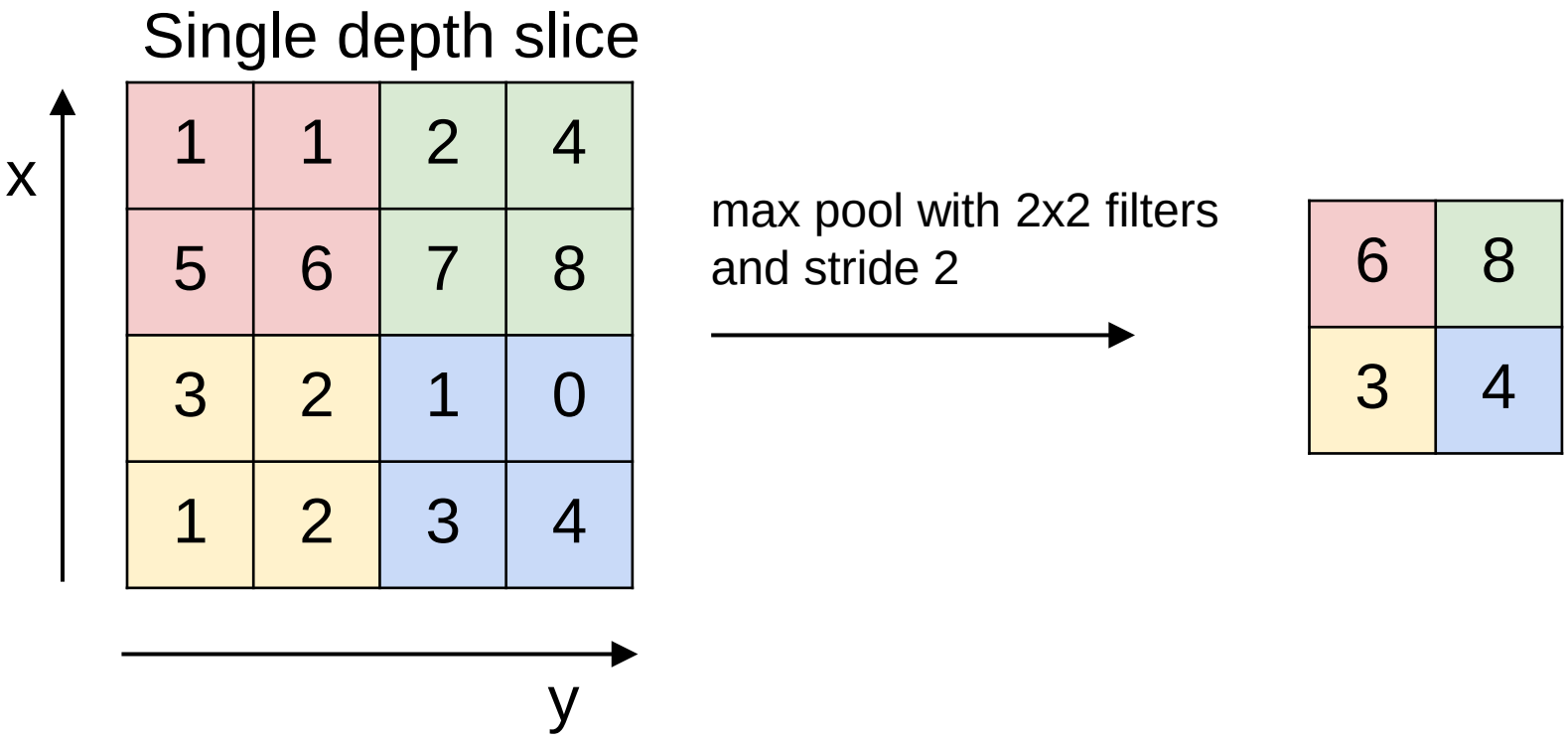
$$h_j^n(x,y) = \sqrt{\sum_{k\in N(j)} h_k^{n-1}(x,y)^2}$$

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)

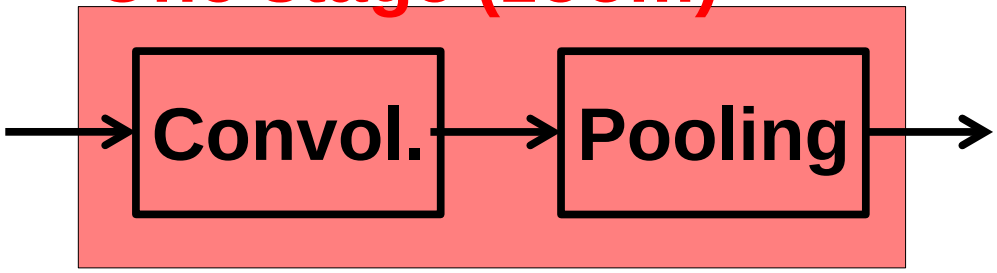


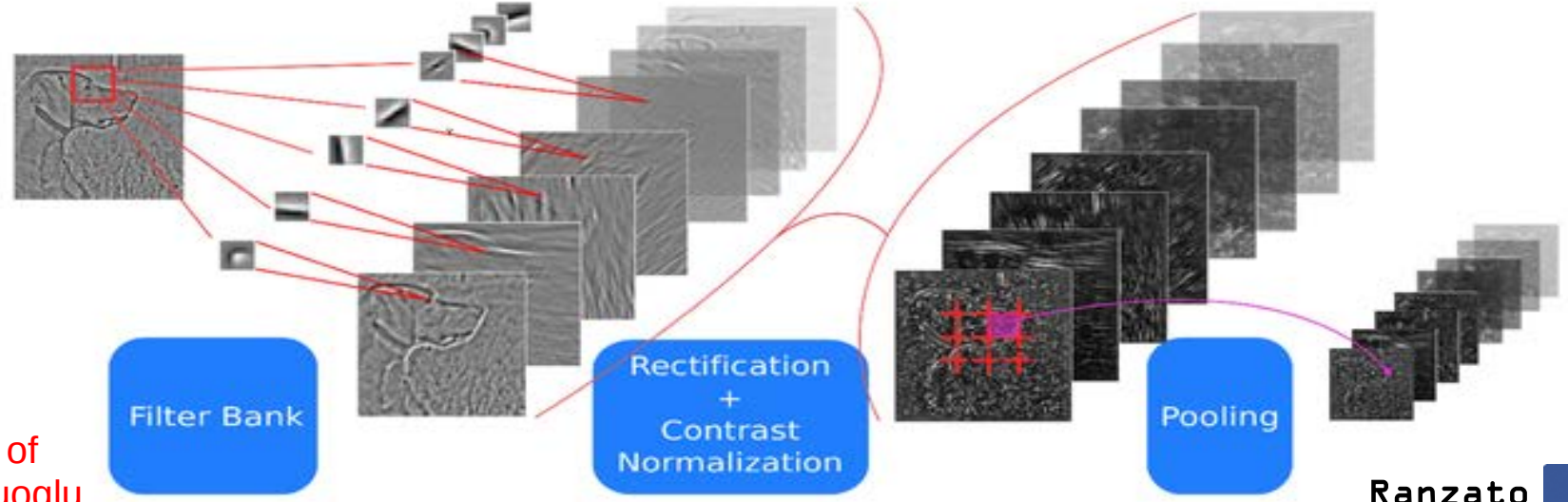
MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

ConvNets: Typical Stage

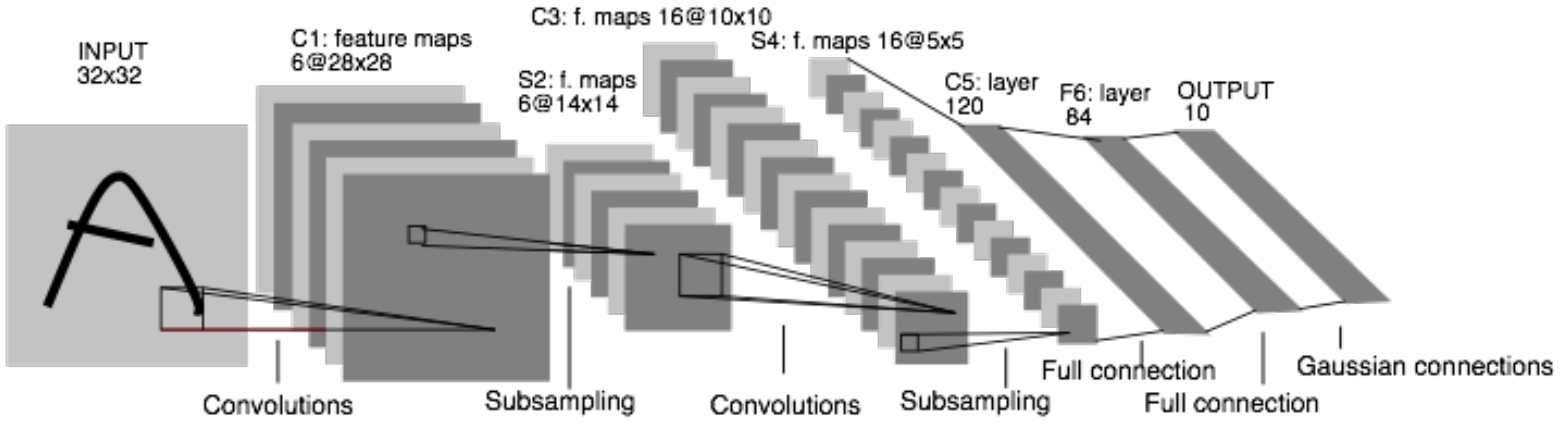
One stage (zoom)





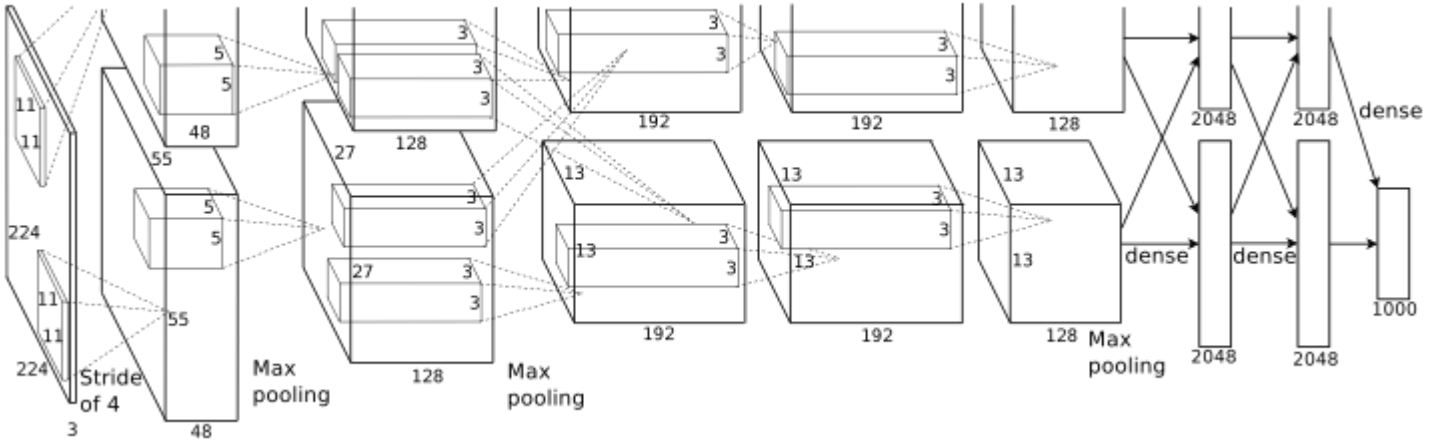
courtesy of
K. Kavukcuoglu

Convolutional Neural Nets (CNNs): 1989



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

Convolutional Nets: (Krizevsky et al, 2012)



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

+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

IMAGENET [Deng et al. 2009]

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
  - Food
  - Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

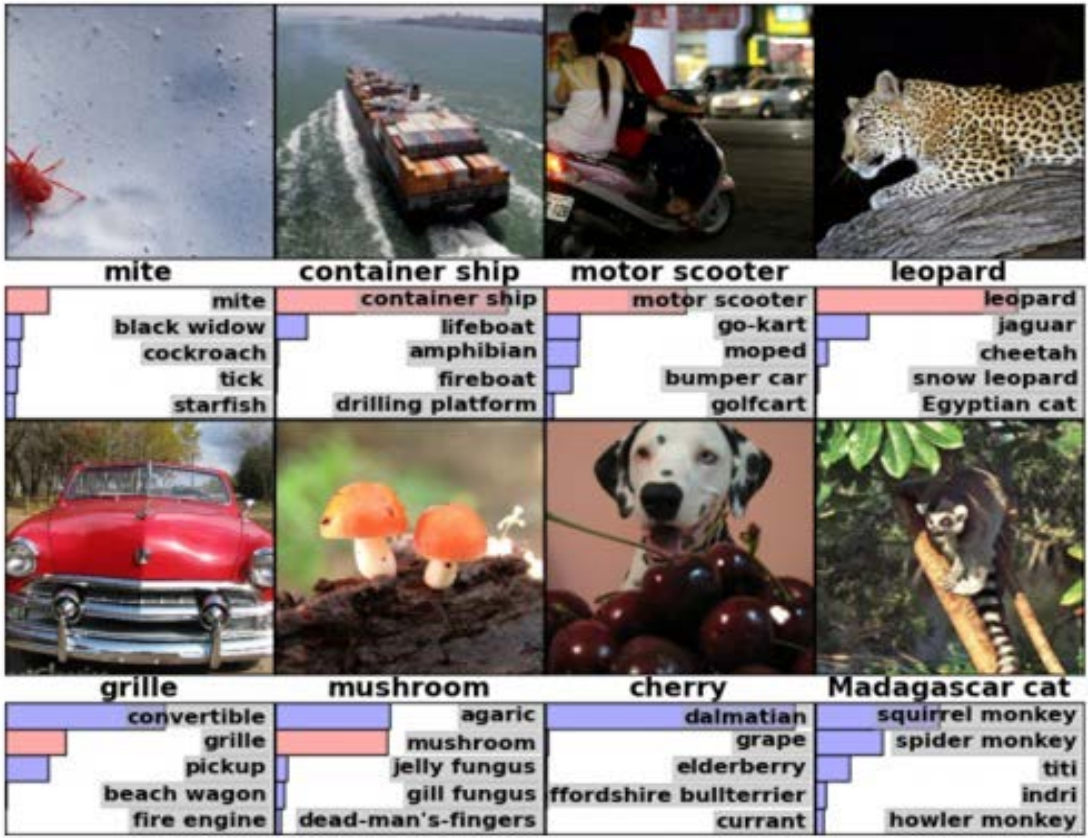
✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

Russakovsky et al. arXiv, 2014

ImageNet Challenge



Clarifai: Amplifying Intellig

Alyosha

www.clarifai.com

Apps | NYC Stuff | Research | Paris Stuff | Кино онлайн беспл... | teaching | Bay Area Stuff | Other bookmarks

clarifai

ABOUT TECHNOLOGY API NEWS JOBS CONTACT

coffee croissant beverage morning breakfast food

night bridge city suspension bridge river

winter snow cold mammal dog arctic

concert performance stage musical performance band singer

train subway railroad railway station transportation

canal river bridge water transportation

clarifai

ABOUT PRICING DEVELOPER CAREERS CONTACT US SIGN UP

Predicted Tags

people fame wear

brunette star sweater

entertainment pop jacket

adult

Similar Images

clarifai

ABOUT PRICING DEVELOPER CAREERS CONTACT US SIGN UP

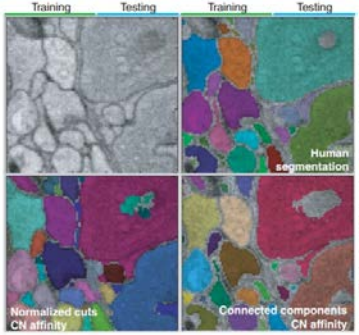
Predicted Tags

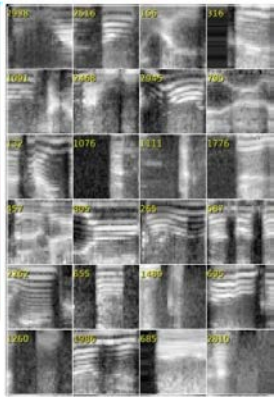
people one indoors adult
woman wear man furniture
room child

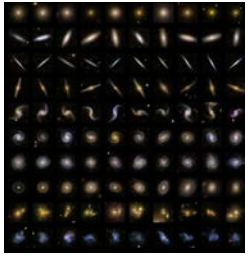
Similar Images

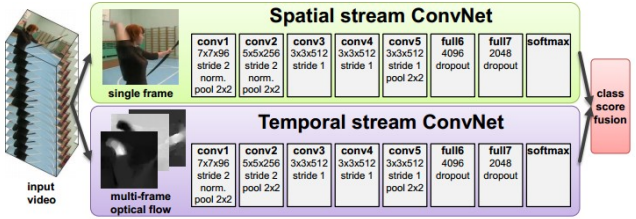
(show convnetjs example of training of CIFAR-10)

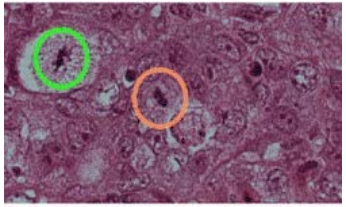
demo

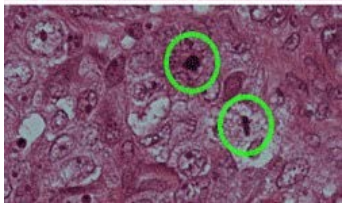

Training
Testing
Training
Testing
Human segmentation
Normalized cuts CN affinity
Connected components CN affinity

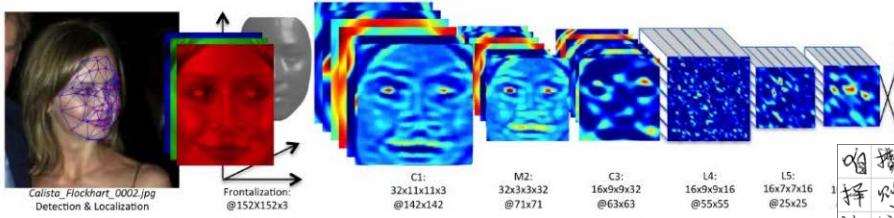


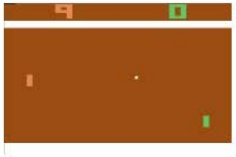



Spatial stream ConvNet
single frame
conv1 7x7x96 stride 2 norm. pool 2x2
conv2 5x5x256 stride 2 norm. pool 2x2
conv3 3x3x512 stride 1
conv4 3x3x512 stride 1
conv5 3x3x512 stride 1 pool 2x2
full6 4096 dropout
full7 2048 dropout
softmax
Temporal stream ConvNet
multi-frame optical flow
input video
class score fusion





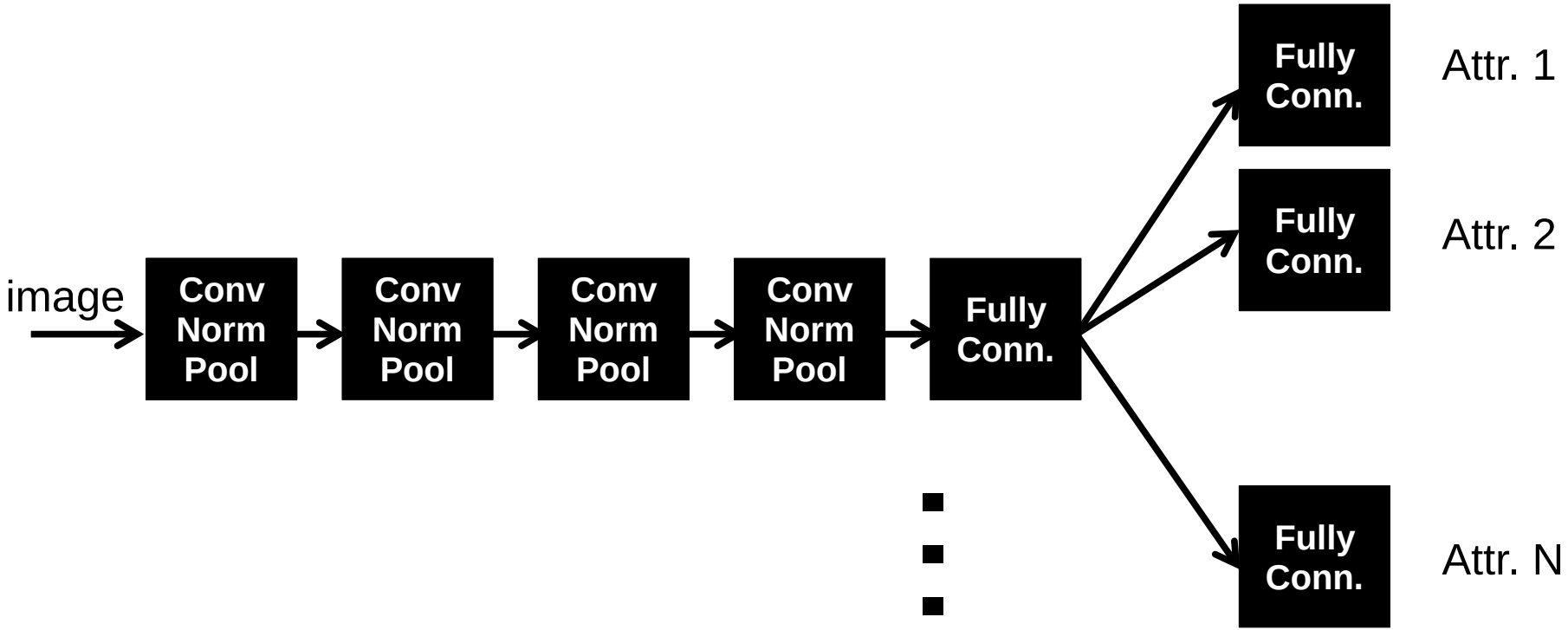

Calista_Flockhart_0002.jpg
Detection & Localization
Frontalization: @152x152x3
C1: 32x11x11x3 @142x142
M2: 32x3x3x32 @71x71
C3: 16x9x9x32 @63x63
L4: 16x9x9x16 @55x55
L5: 16x7x7x16 @25x25
REPRESENTATION
SFC labels




OXYGEN


000044
SECTOR 01

Fancier Architectures: Multi-Task



Zhang et al. “PANDA..” CVPR 2014

Fancier Architectures: Multi-Task

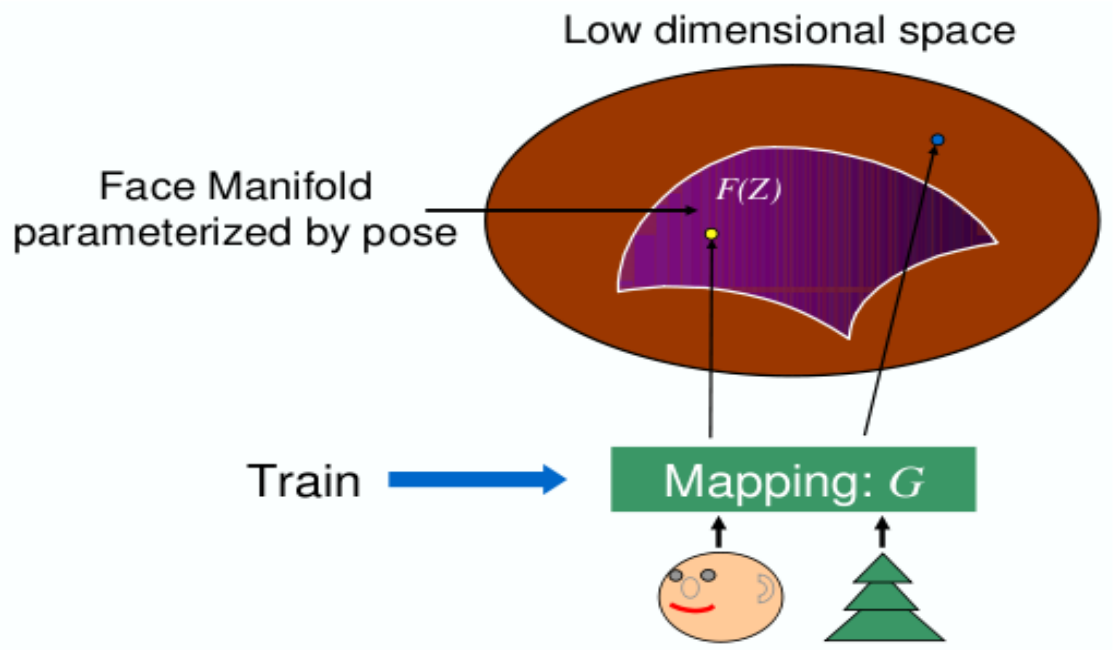
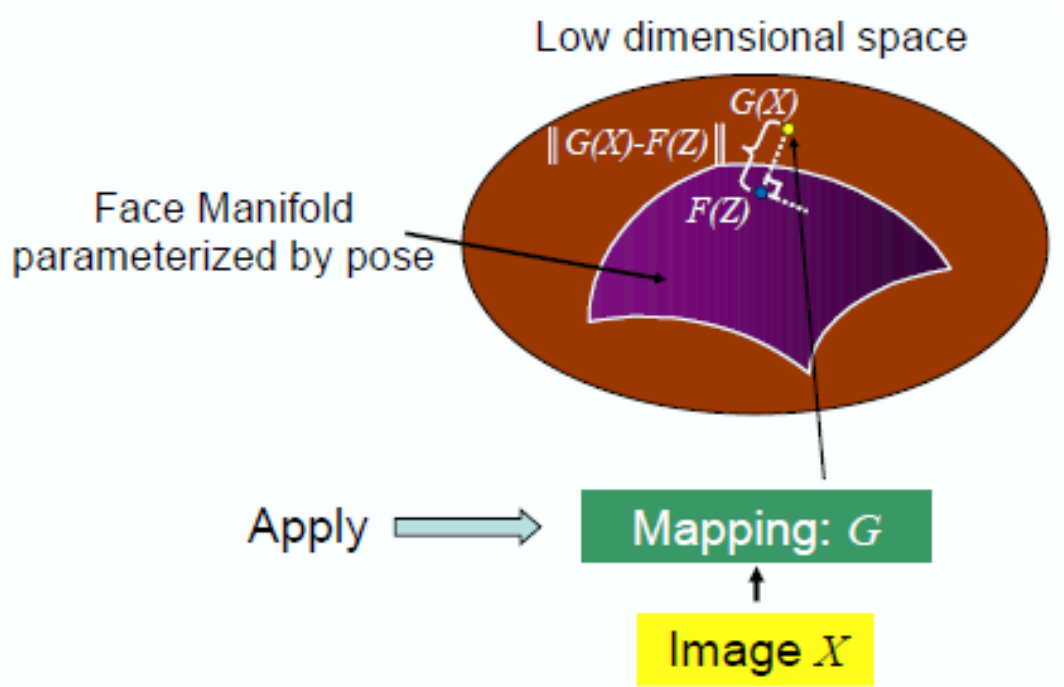


Figure 1: Manifold Mapping—Training

Jame Bromley, Jim W. Bentz, Léon Bottou, Isabelle Guyon, Yann Le Cun, C. Moore, Eduard Säckinger, Roopak Shah, Signature verification using a "Siamese" time delay neural network, IJPRAI 1993

Osadchy et al. "Synergistic face detection and pose estimation.." JMLR 2007

Fancier Architectures: Multi-Task



Osadchy et al. “Synergistic face detection and pose estimation..” JMLR 2007

Fancier Architectures: Multi-Task

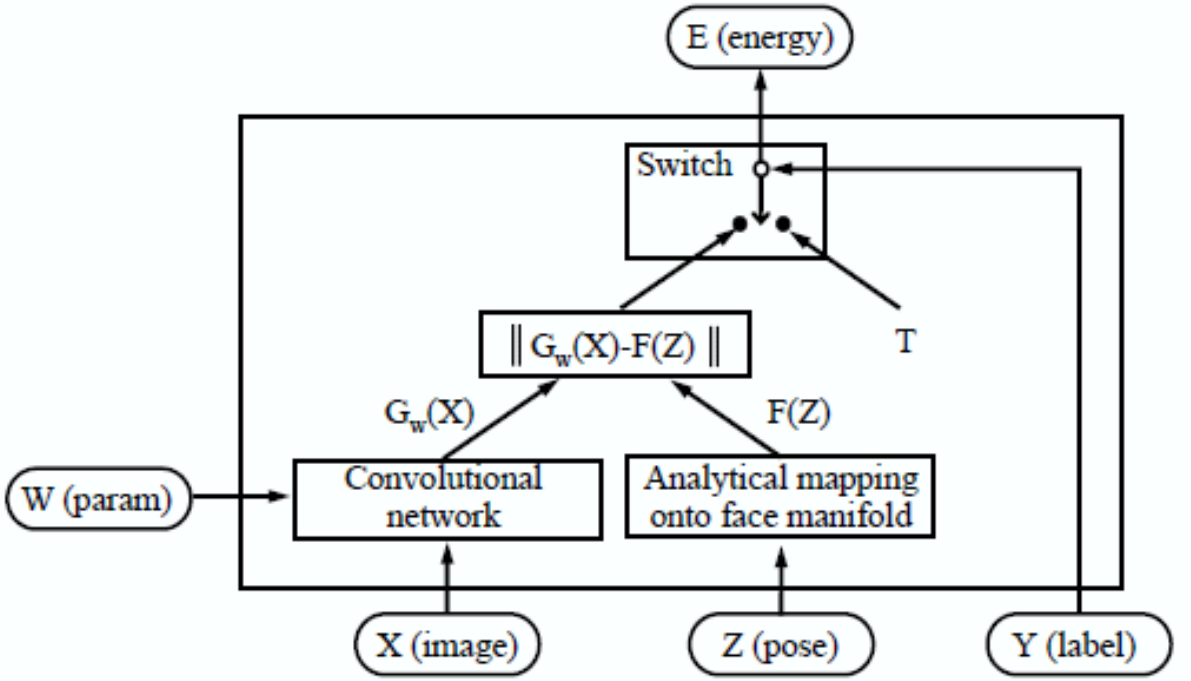
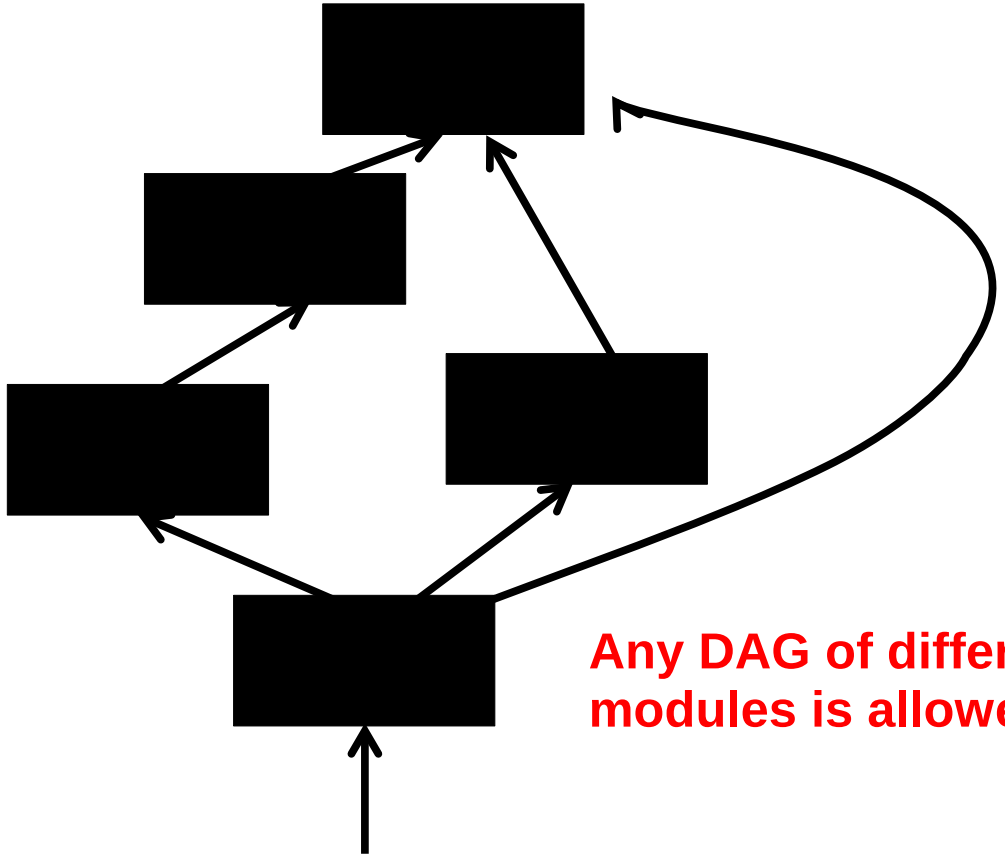


Figure 4: Architecture of the Minimum Energy Machine.

Osadchy et al. “Synergistic face detection and pose estimation..” JMLR 2007

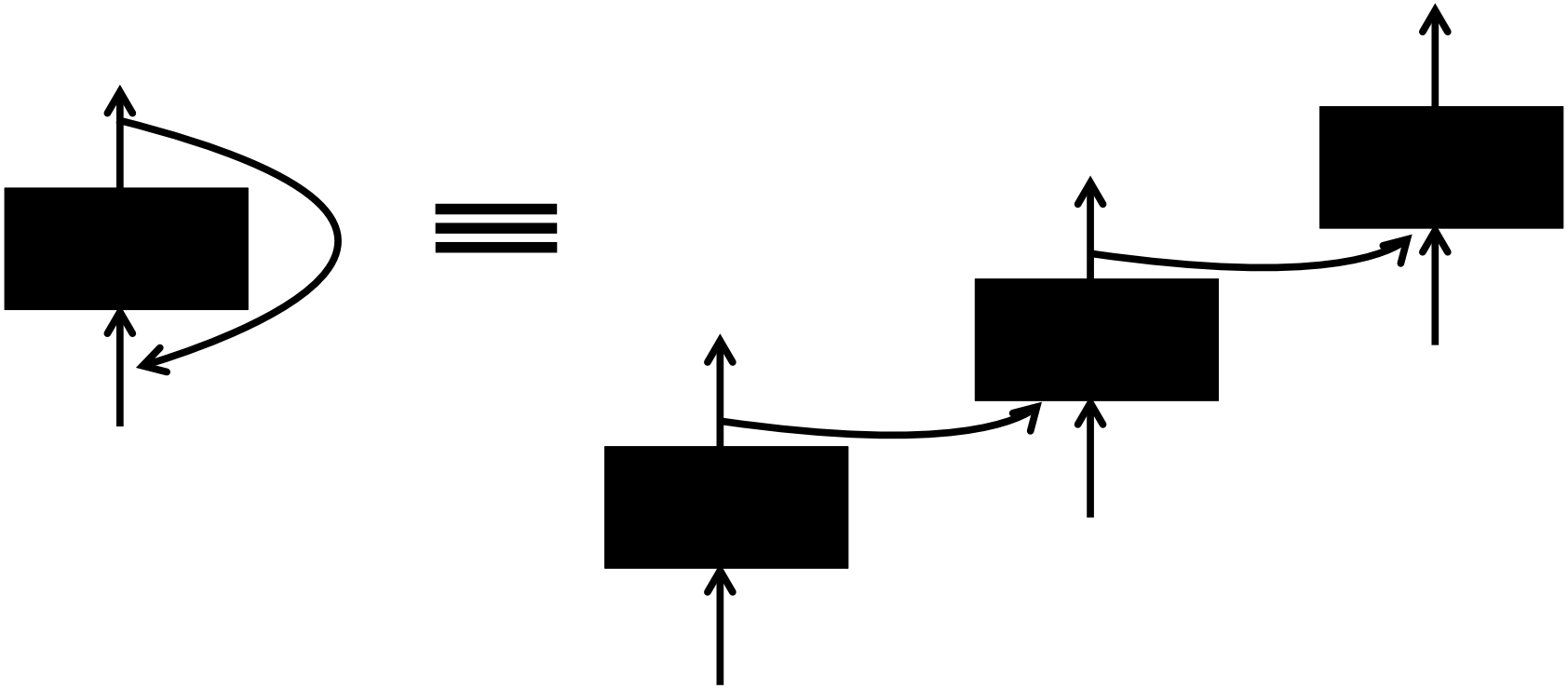
Fancier Architectures: Generic DAG



Any DAG of differentialble modules is allowed!

Fancier Architectures: Generic DAG

If there are cycles (RNN), one needs to un-roll it.



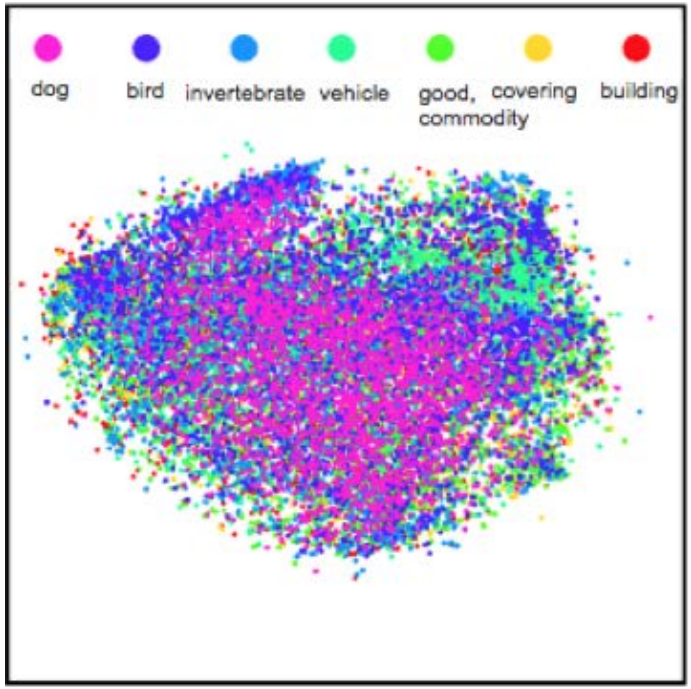
Pinheiro, Collobert “Recurrent CNN for scene labeling” ICML 2014
Graves “Offline Arabic handwriting recognition..” Springer 2012

“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

The Unreasonable Effectiveness of Deep Features



Low-level: $Pool_1$

High-level: FC_6

Classes separate in the deep representations and transfer to many tasks.
[DeCAF] [Zeiler-Fergus]

Can be used as a generic feature

(“CNN code” = 4096-D vector before classifier)

query image

nearest neighbors in the “code” space

Transfer Learning with CNNs

1. Train on Imagenet

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

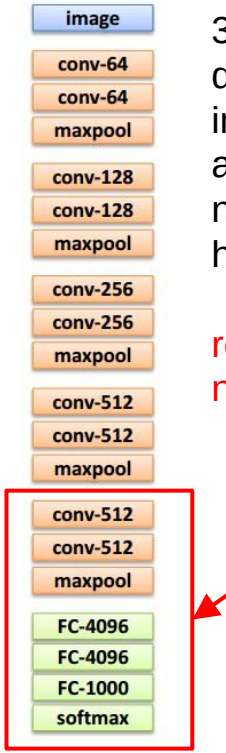
i.e. swap the Softmax layer at the end

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

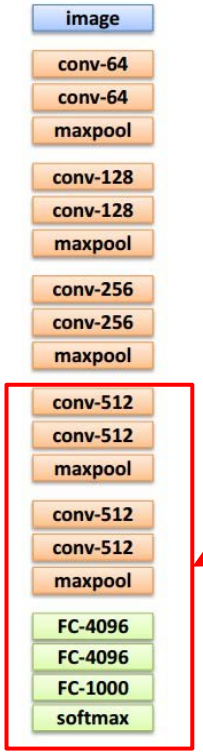
retrain bigger portion of the network, or even all of it.

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

tip: use only ~1/10th of the original learning rate in finetuning to player, and ~1/100th on intermediate layers

SOFTWARE

Torch7: learning library that supports neural net training (NYU/Facebook)

torch.ch
http://code.cogbits.com/wiki/doku.php (tutorial with demos by C. Farabet)
https://github.com/jhjin/overfeat-torch
https://github.com/facebook/fbcunn/tree/master/examples/imagenet

Theano: 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 / UC Berkeley)

– http://caffe.berkeleyvision.org

TensorFlow (Google)

– http://www.tensorflow.org/