# Neural networks

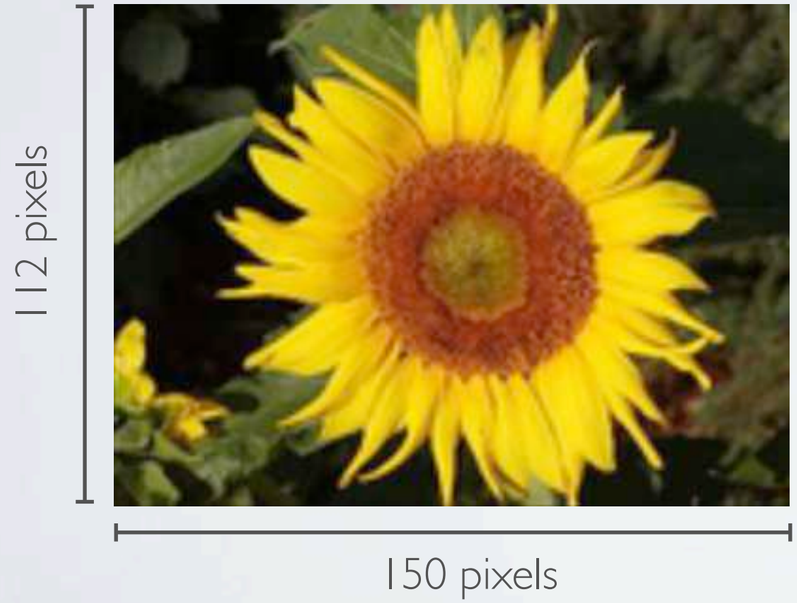
## Computer vision - motivation

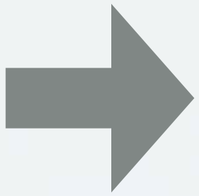
# COMPUTER VISION

**Topics:** computer vision, object recognition

- Computer vision is the design of computers that can process visual data and accomplish some given task
  - we will focus on object recognition: given some input image, identify which object it contains

Caltech 101 dataset



"sun flower"

# COMPUTER VISION

**Topics:** computer vision

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units

# Neural networks

## Computer vision - local connectivity

# COMPUTER VISION

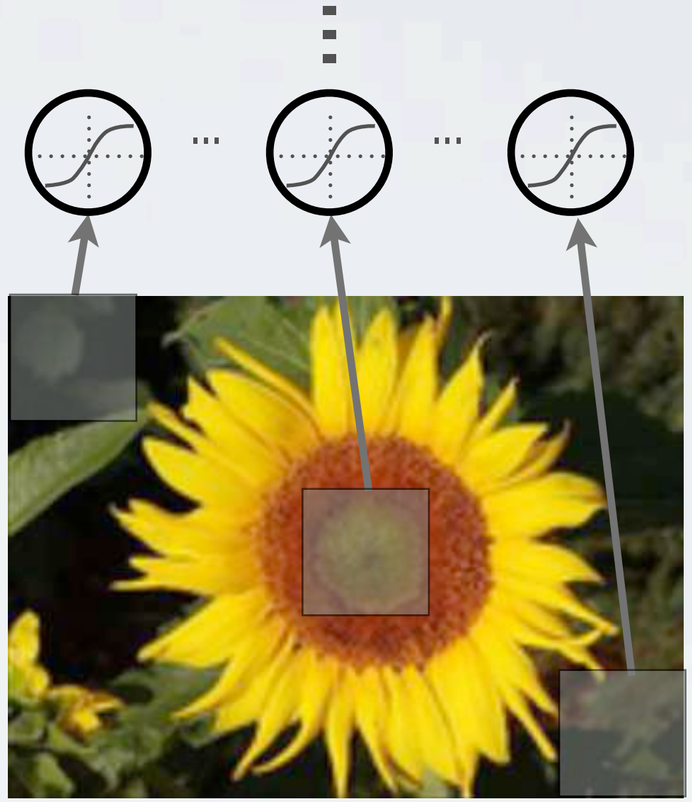
**Topics:** computer vision

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - **local connectivity**
  - parameter sharing
  - pooling / subsampling hidden units

# COMPUTER VISION

**Topics:** local connectivity

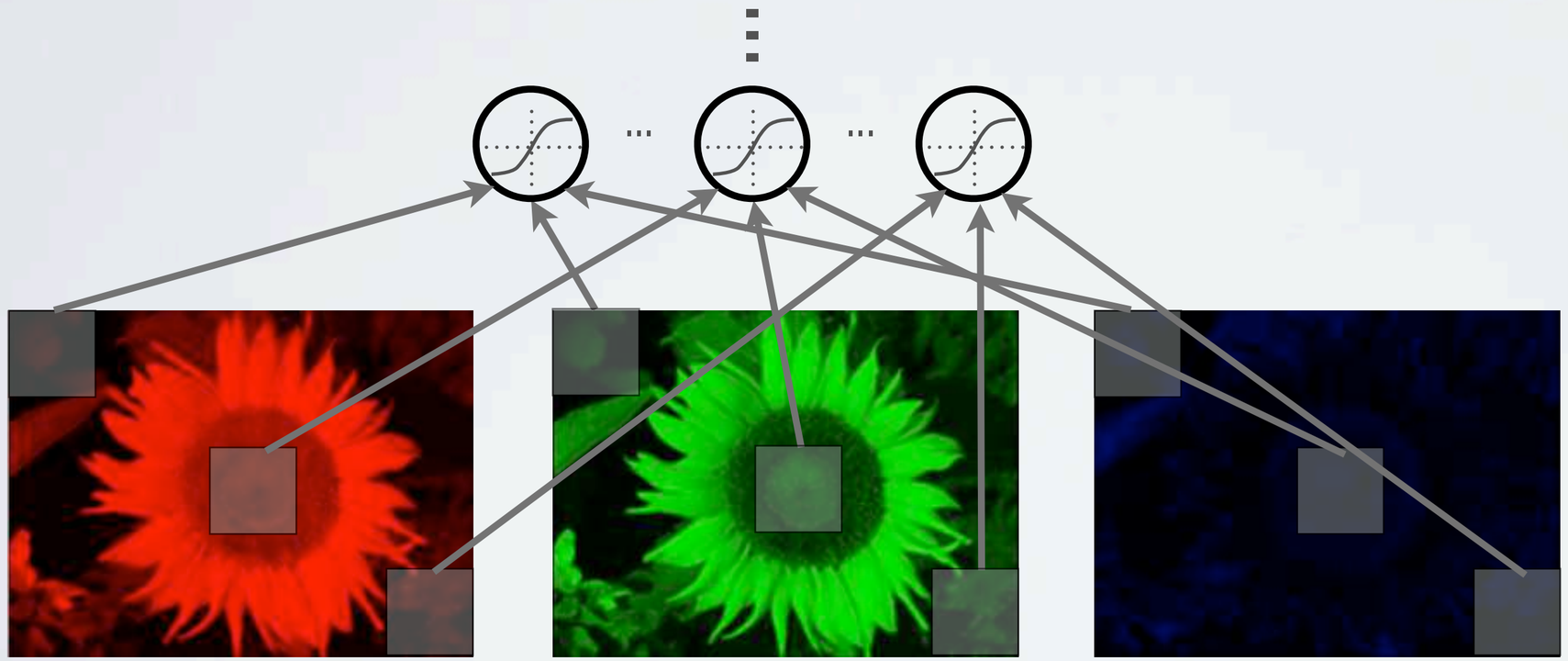
- First idea: use a local connectivity of hidden units
  - each hidden unit is connected only to a subregion (patch) of the input image
  - it is connected to all channels
    - 1 if greyscale image
    - 3 (R, G, B) for color image
- Solves the following problems:
  - fully connected hidden layer would have an unmanageable number of parameters
  - computing the linear activations of the hidden units would be very expensive



# COMPUTER VISION

**Topics:** local connectivity

- Units are connected to all channels:
  - 1 channel if grayscale image, 3 channels (R, G, B) if color image



# Neural networks

## Computer vision - parameter sharing

# COMPUTER VISION

**Topics:** computer vision

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - local connectivity
  - **parameter sharing**
  - pooling / subsampling hidden units

# COMPUTER VISION

**Topics:** parameter sharing

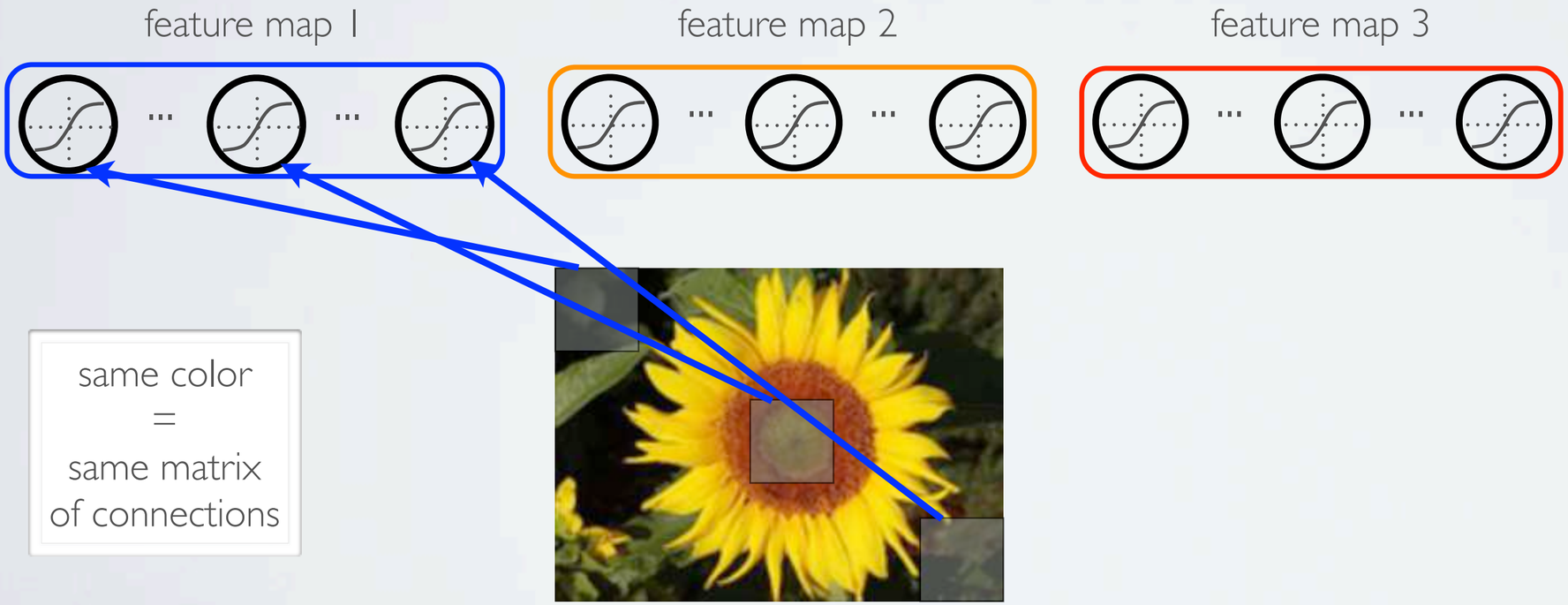
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image


same color
=
same matrix
of connections

# COMPUTER VISION

**Topics:** parameter sharing

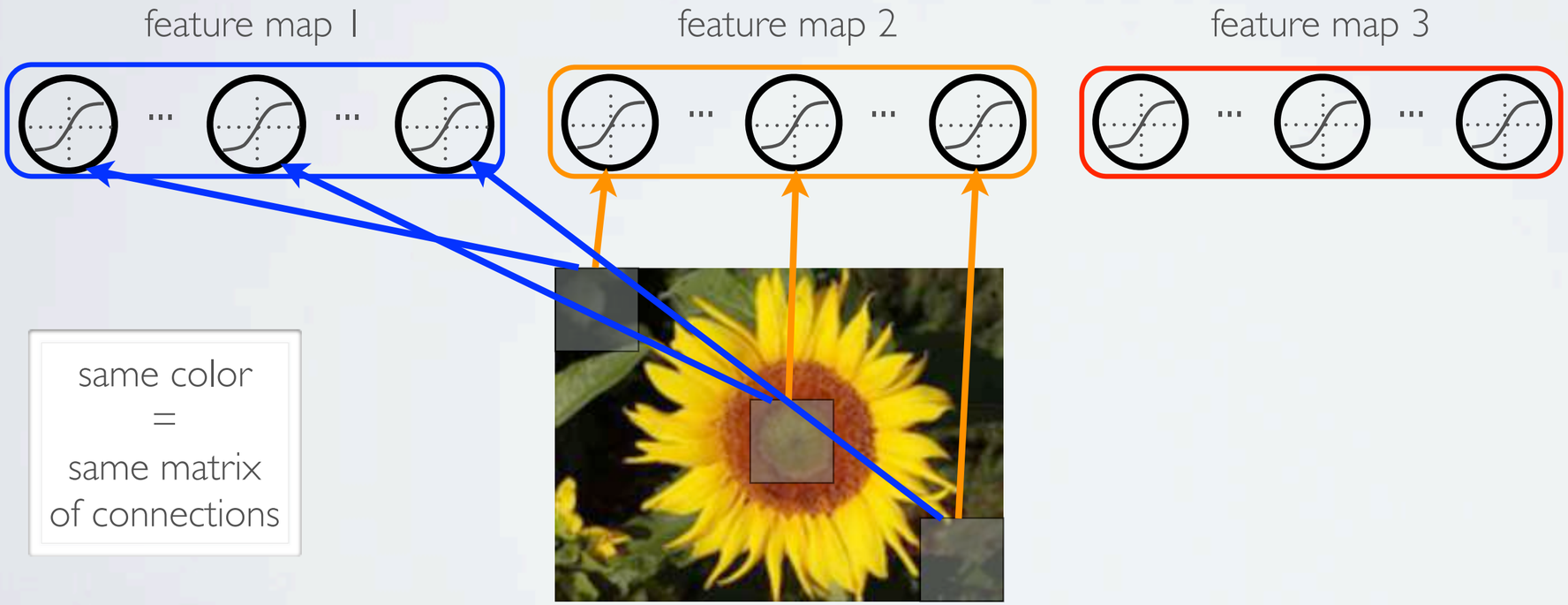
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



# COMPUTER VISION

**Topics:** parameter sharing

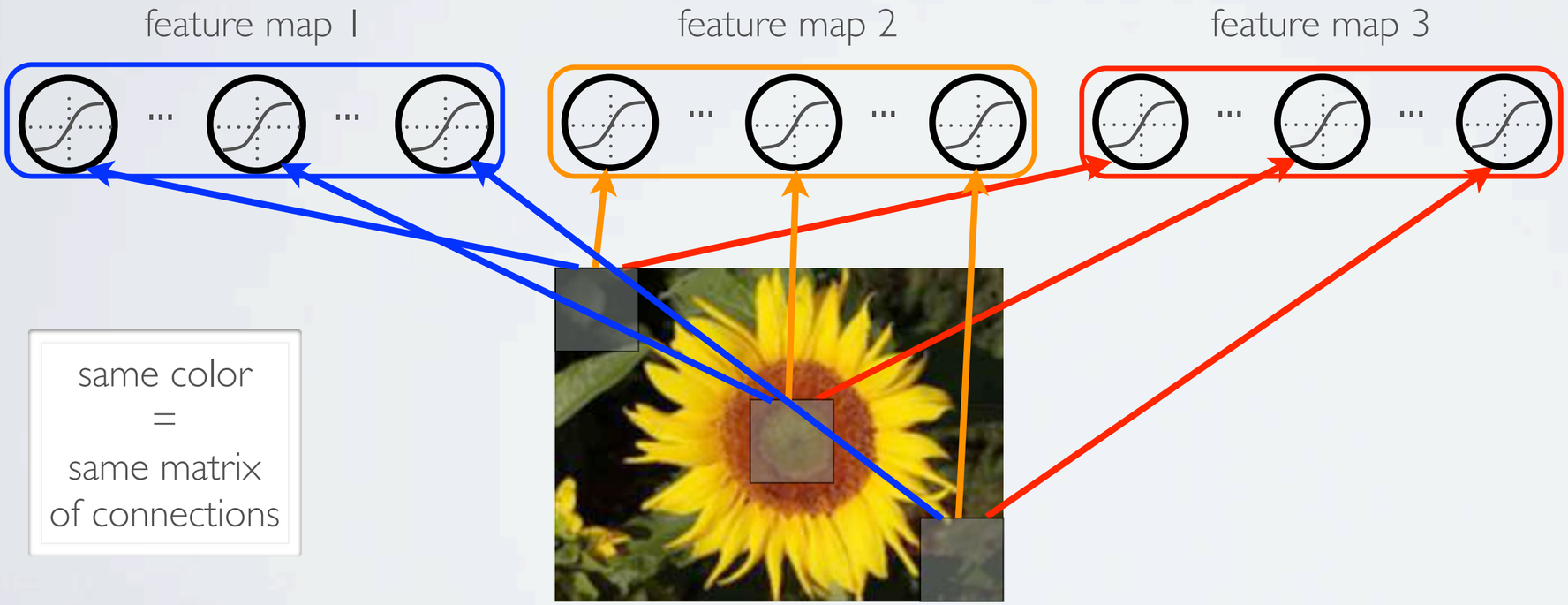
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



# COMPUTER VISION

**Topics:** parameter sharing

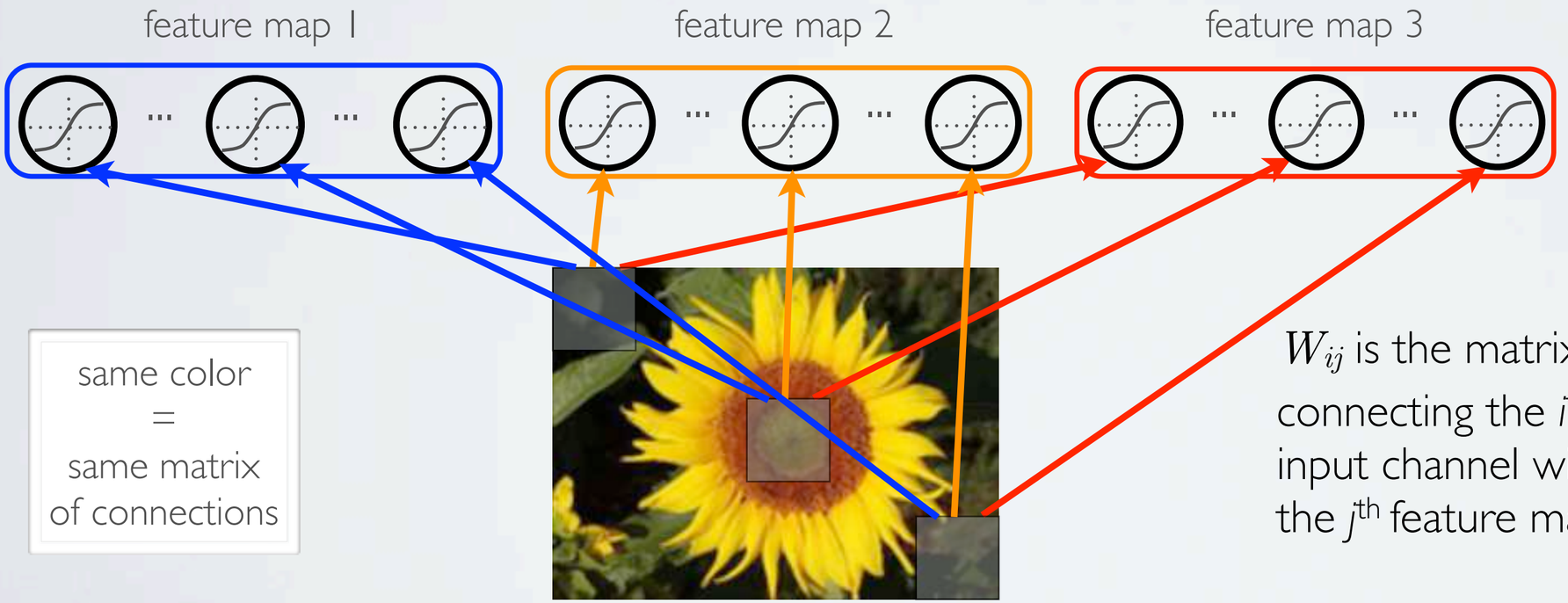
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



# COMPUTER VISION

**Topics:** parameter sharing

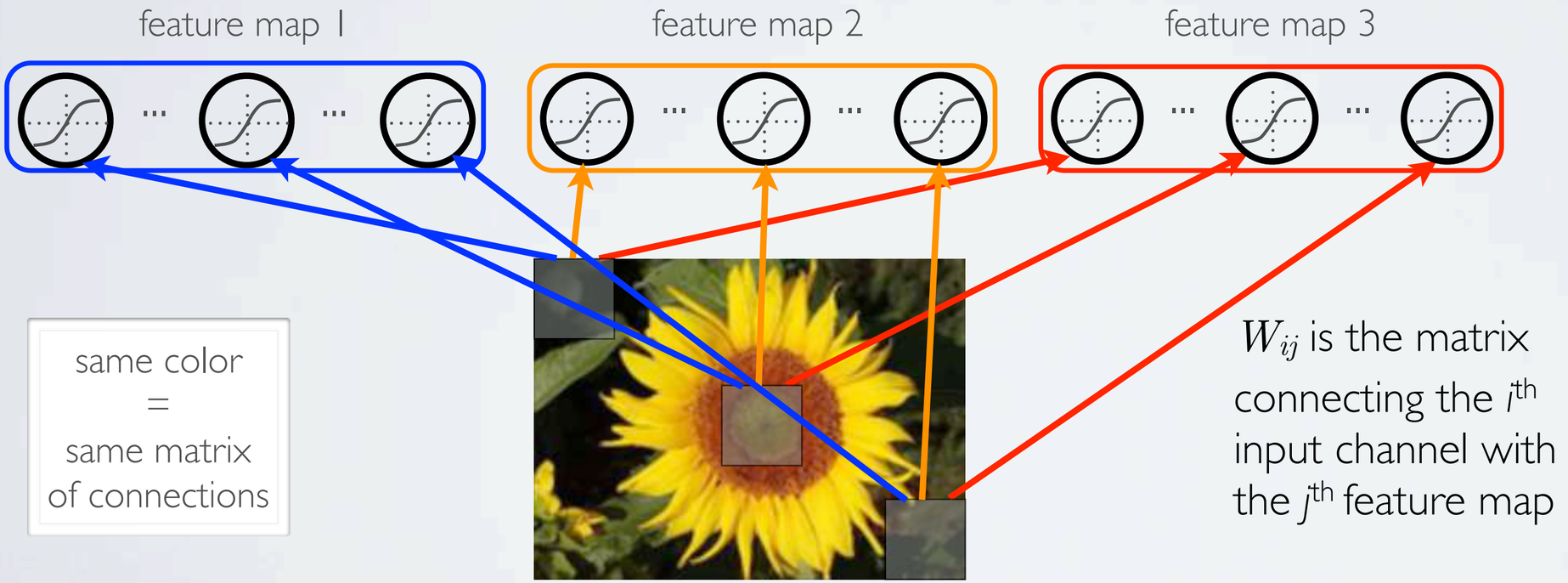
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



# COMPUTER VISION

**Topics:** parameter sharing

- Solves the following problems:
  - reduces even more the number of parameters
  - will extract the same features at every position (features are "equivariant")

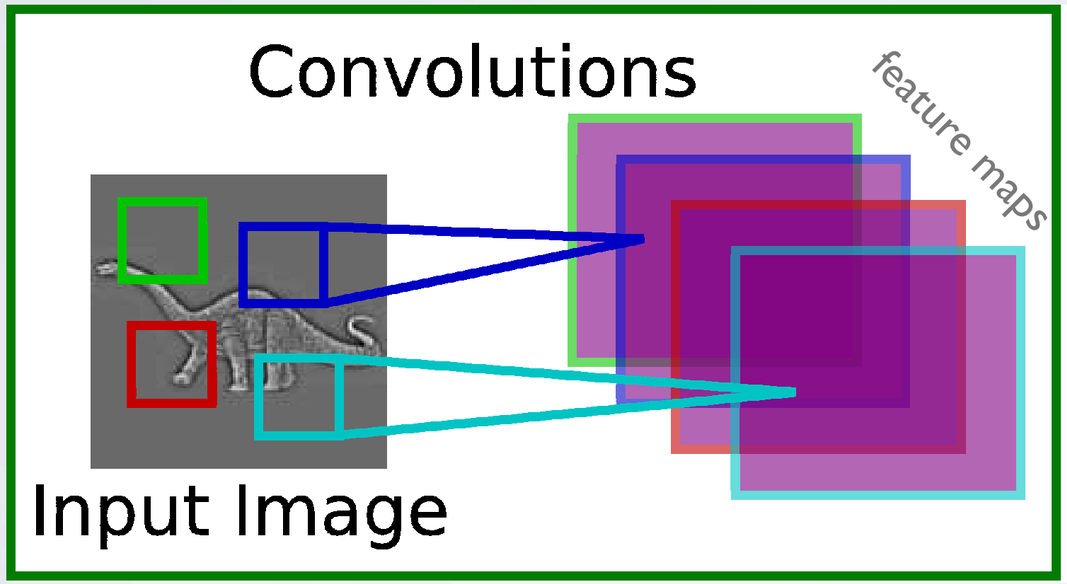


# COMPUTER VISION

**Topics:** parameter sharing

Jarret et al. 2009

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix $k_{ij}$ which is the hidden weights matrix $W_{ij}$ with its rows and columns flipped




- $x_i$ is the i[th] channel of input
- $k_{ij}$ is the convolution kernel
- $g_j$ is a learned scaling factor
- $y_j$ is the hidden layer

(could have added a bias)

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

# Neural networks

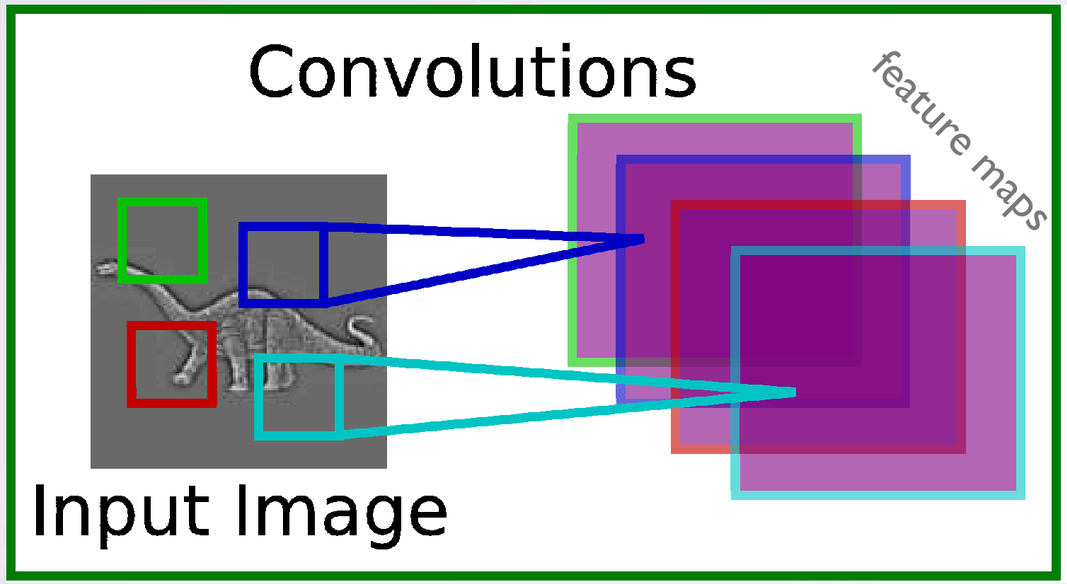
## Computer vision - discrete convolution

# COMPUTER VISION

**Topics:** parameter sharing

Jarret et al. 2009

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix $k_{ij}$ which is the hidden weights matrix $W_{ij}$ with its rows and columns flipped




- $x_i$ is the i[th] channel of input
- $k_{ij}$ is the convolution kernel
- $g_j$ is a learned scaling factor
- $y_j$ is the hidden layer

(could have added a bias)

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

# COMPUTER VISION

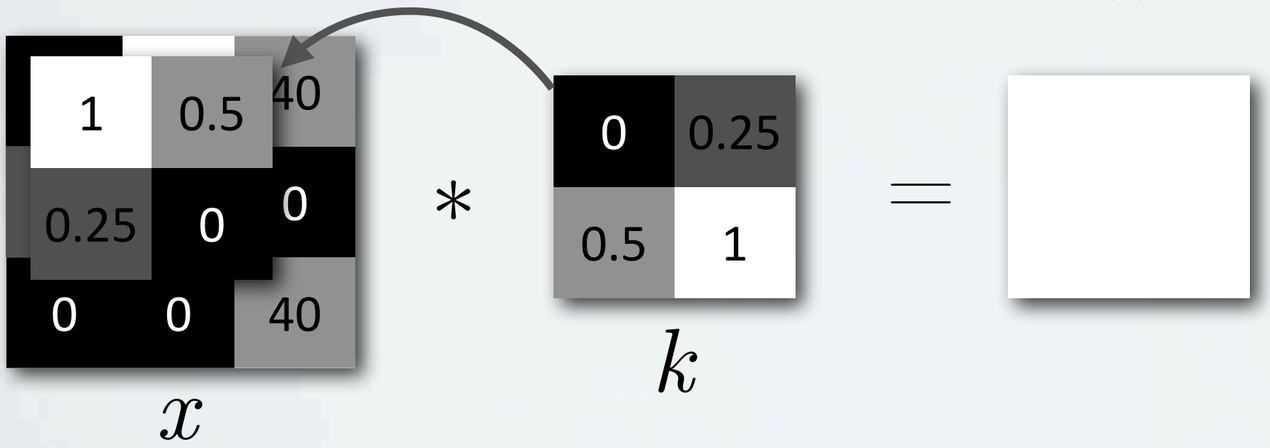
**Topics:** discrete convolution

- The convolution of an image $x$ with a kernel $k$ is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:

$\tilde{k} = k$ with rows and columns flipped



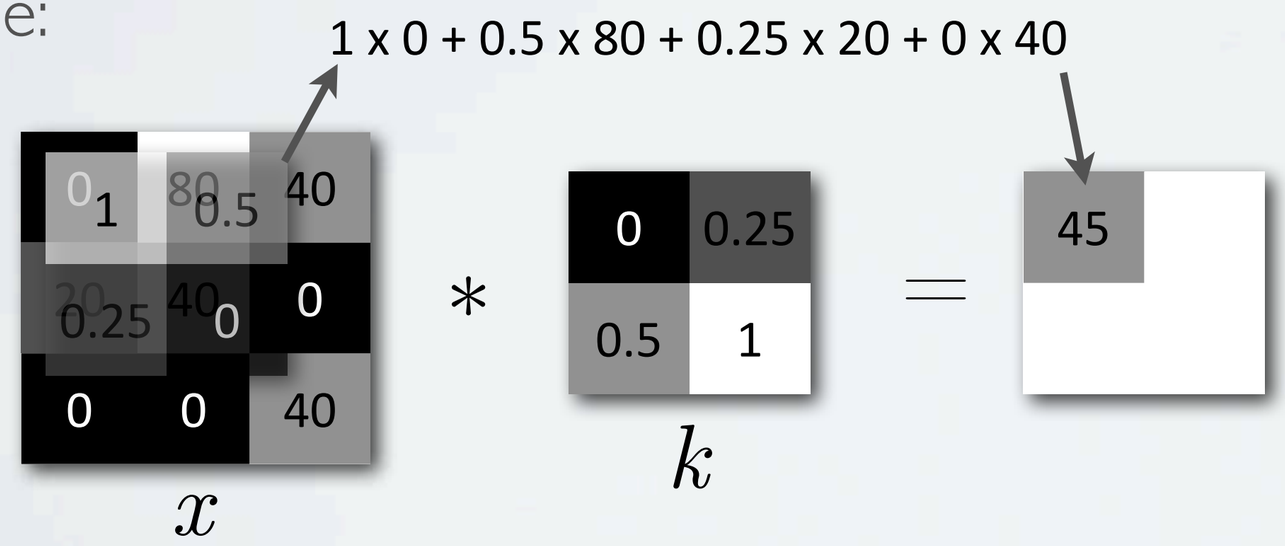
# COMPUTER VISION

**Topics:** discrete convolution

- The convolution of an image $x$ with a kernel $k$ is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



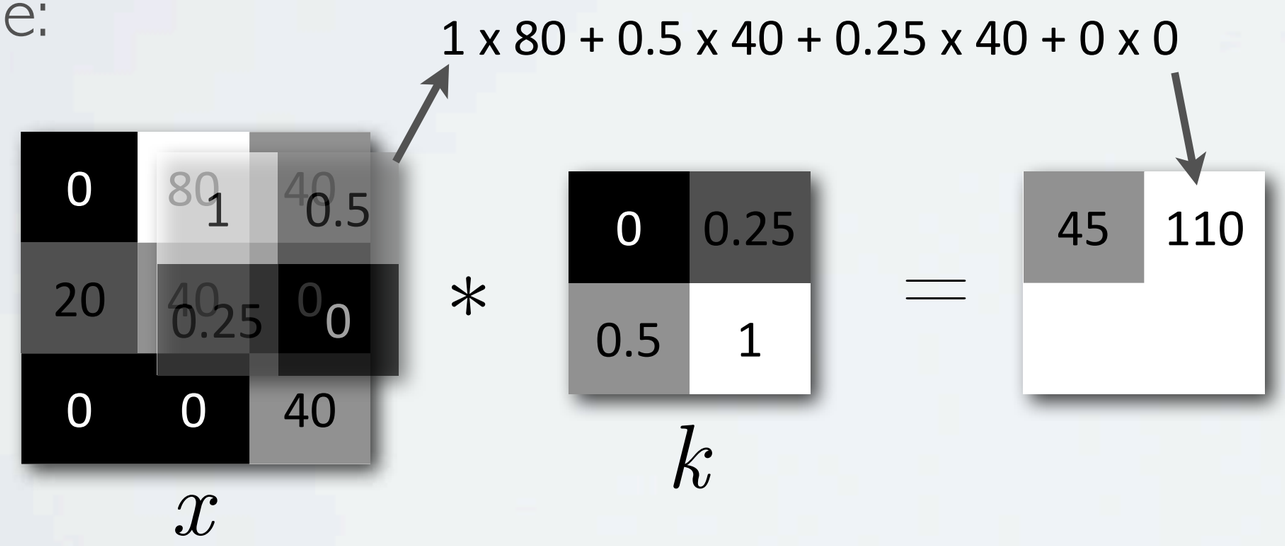
# COMPUTER VISION

**Topics:** discrete convolution

- The convolution of an image $x$ with a kernel $k$ is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



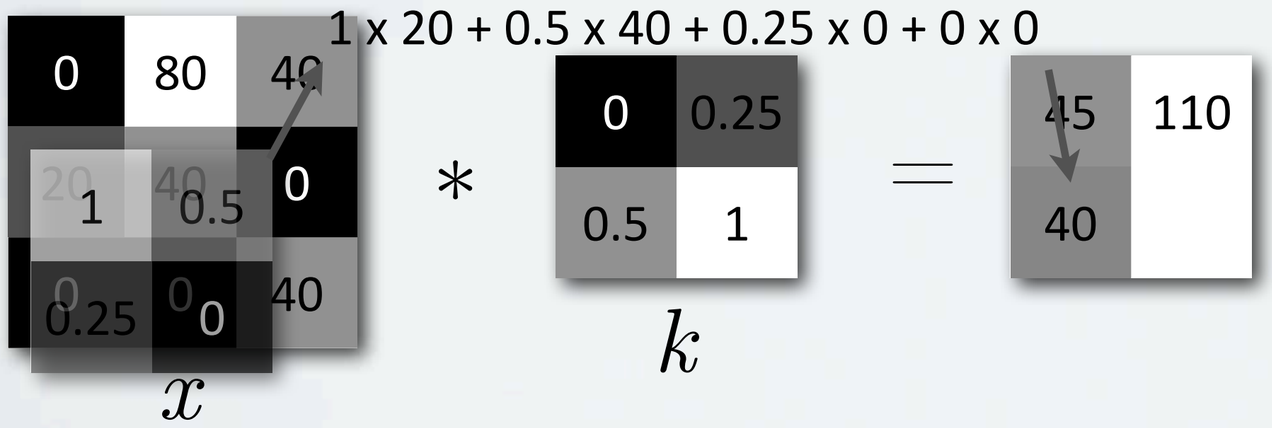
# COMPUTER VISION

**Topics:** discrete convolution

- The convolution of an image $x$ with a kernel $k$ is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



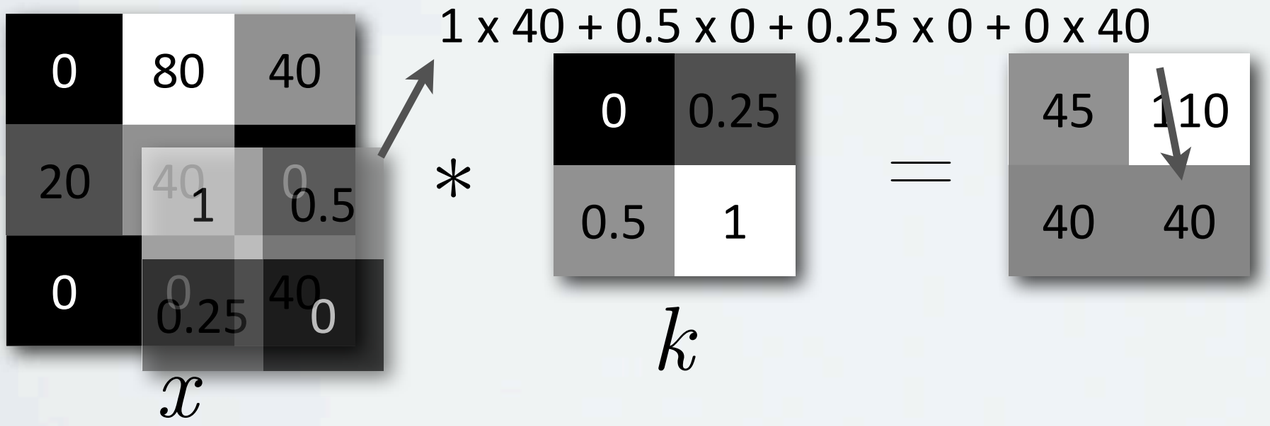
# COMPUTER VISION

**Topics:** discrete convolution

- The convolution of an image $x$ with a kernel $k$ is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



# COMPUTER VISION

**Topics:** discrete convolution

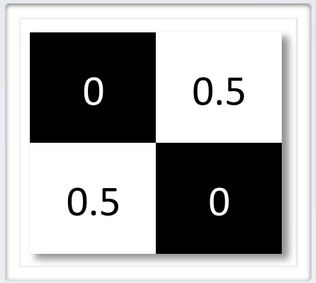
- Pre-activations from channel $x_i$ into feature map $y_j$ can be computed by:
  - getting the convolution kernel where $k_{ij} = \widetilde{W}_{ij}$ from the connection matrix $W_{ij}$
  - applying the convolution $x_i * k_{ij}$

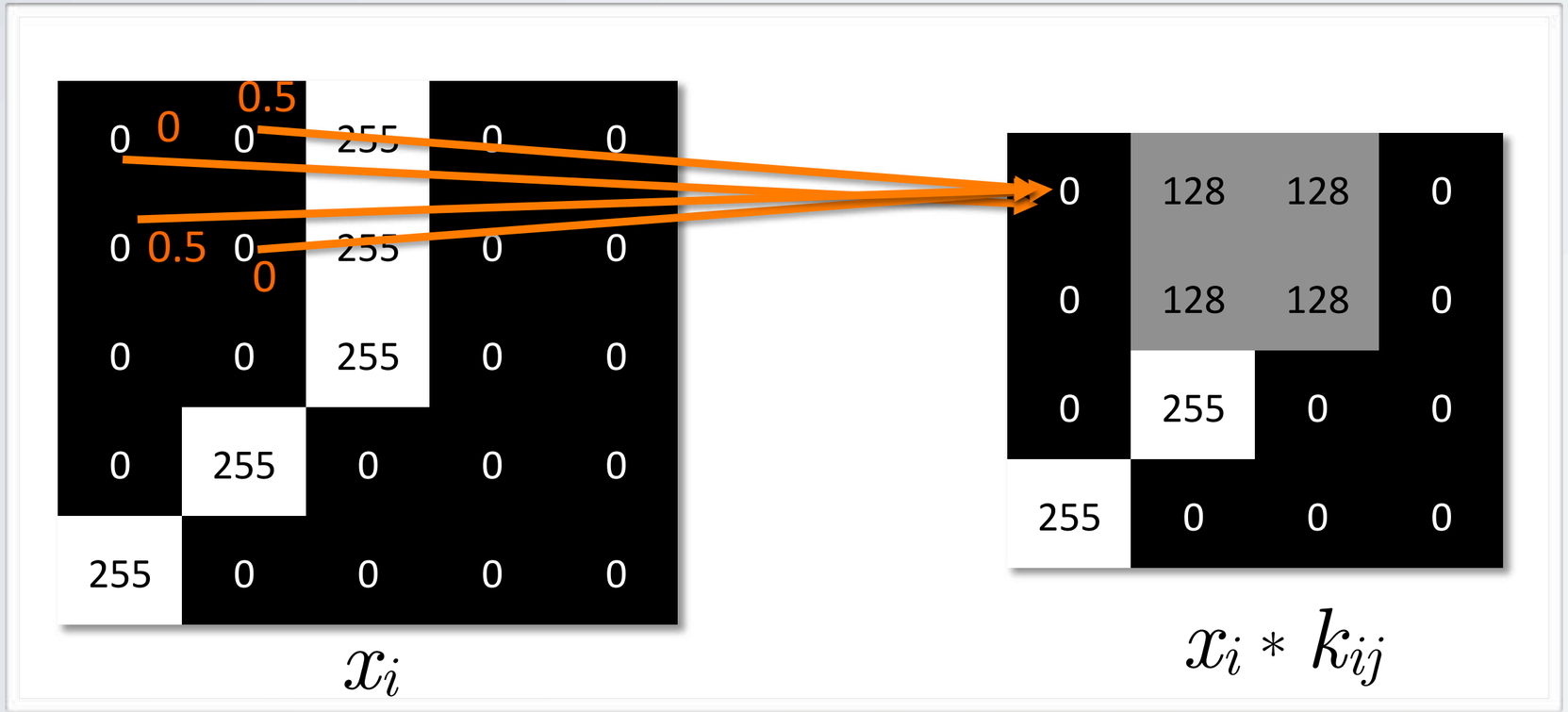
- This is equivalent to computing the discrete correlation of $x_i$ with $W_{ij}$

# COMPUTER VISION

**Topics:** discrete convolution

- Simple illustration: $x_i * k_{ij}$ where $W_{ij} = \tilde{W}_{ij}$

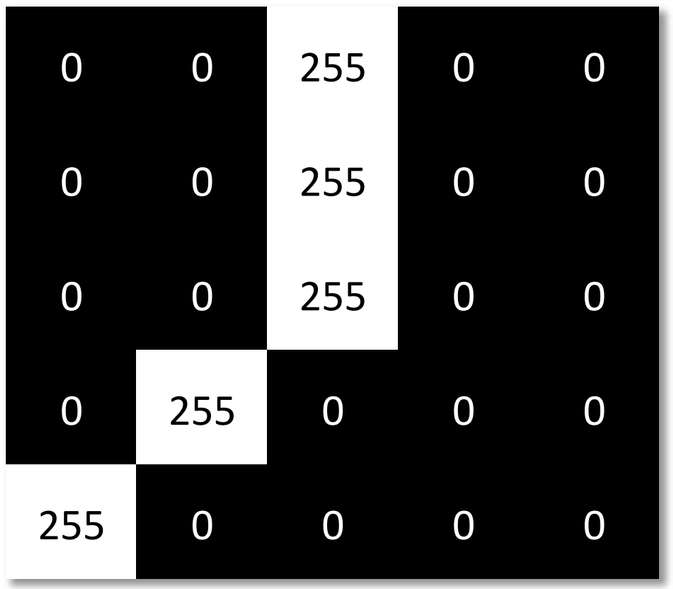


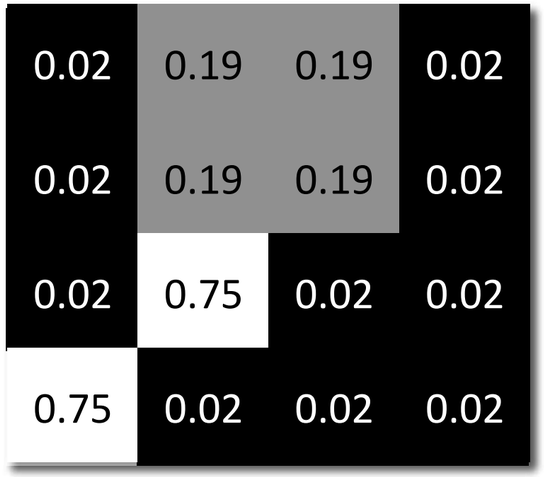
# COMPUTER VISION

**Topics:** discrete convolution

- With a non-linearity, we get a detector of a feature at any position in the image




$x_i$

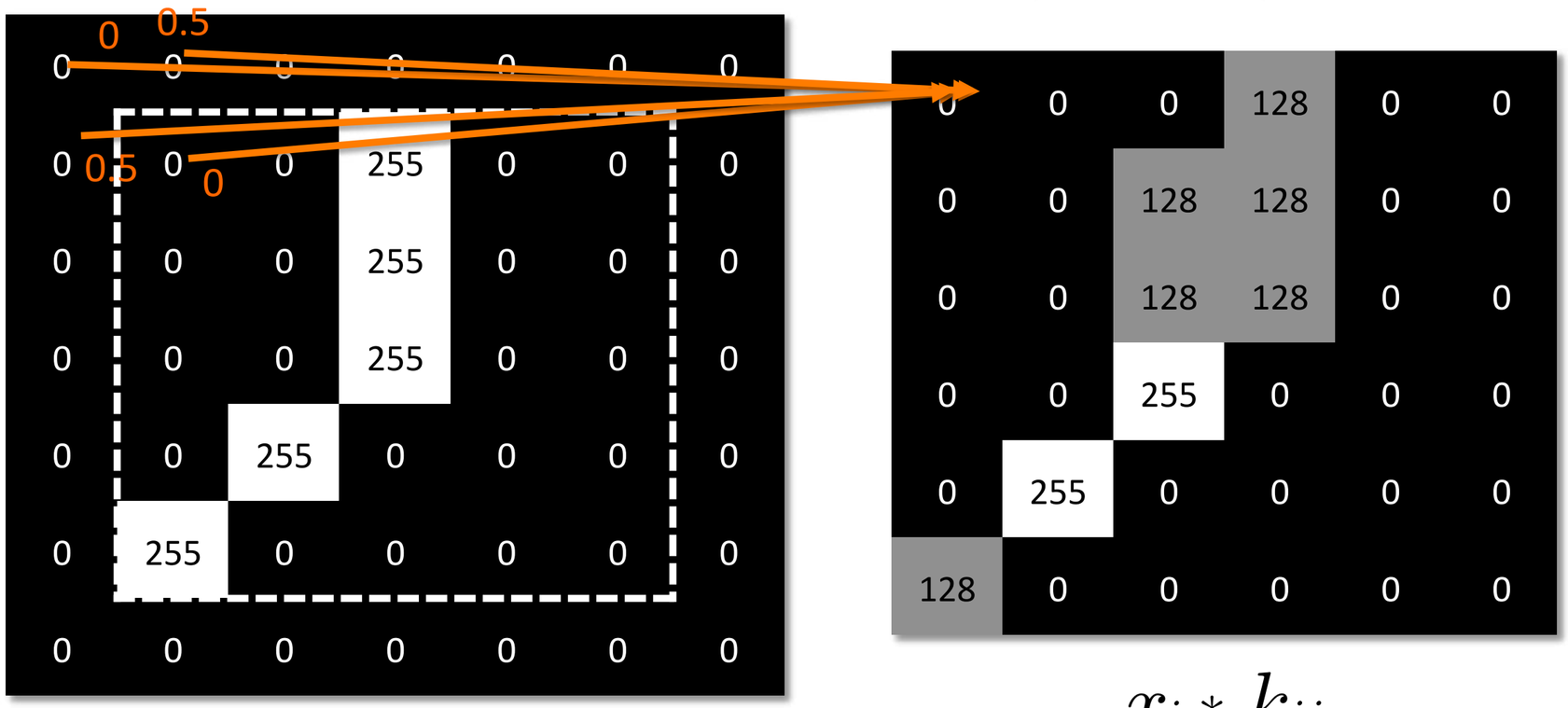



$$\mathrm{sigm}(0.02\ x_i * k_{ij} \text{-}4)$$

# COMPUTER VISION

**Topics:** discrete convolution

- Can use "zero padding" to allow going over the borders ( $\underline{*}$ )



# Neural networks

## Computer vision - pooling and subsampling

# COMPUTER VISION

**Topics:** computer vision

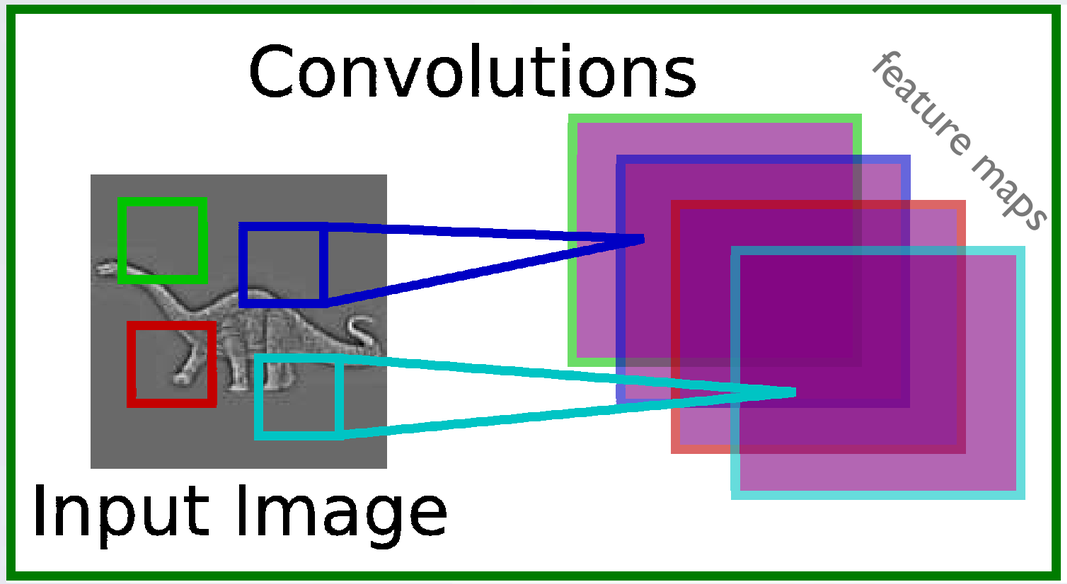
- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - local connectivity
  - parameter sharing
  - **pooling / subsampling hidden units**

# COMPUTER VISION

**Topics:** parameter sharing

Jarret et al. 2009

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix $k_{ij}$ which is the hidden weights matrix $W_{ij}$ with its rows and columns flipped




- $x_i$ is the i[th] channel of input
- $k_{ij}$ is the convolution kernel
- $g_j$ is a learned scaling factor
- $y_j$ is the hidden layer

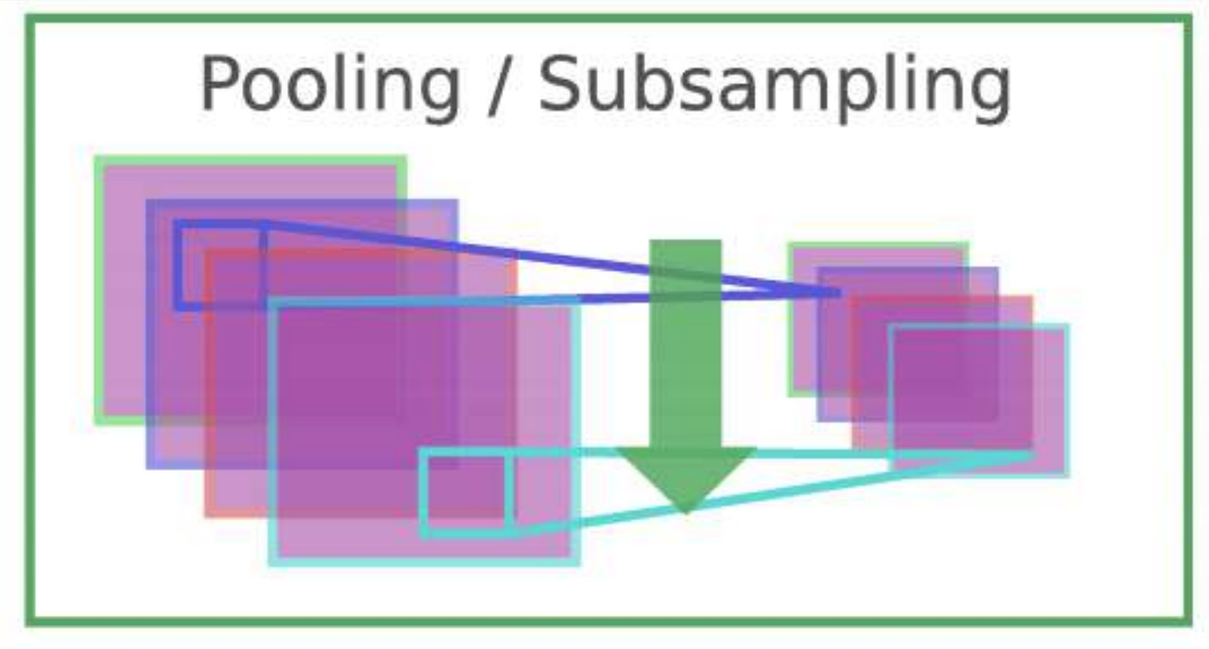
(could have added a bias)

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

# COMPUTER VISION

**Topics:** pooling and subsampling

- Third idea: pool hidden units in same neighborhood
  - pooling is performed in non-overlapping neighborhoods (subsampling)




Jarret et al. 2009

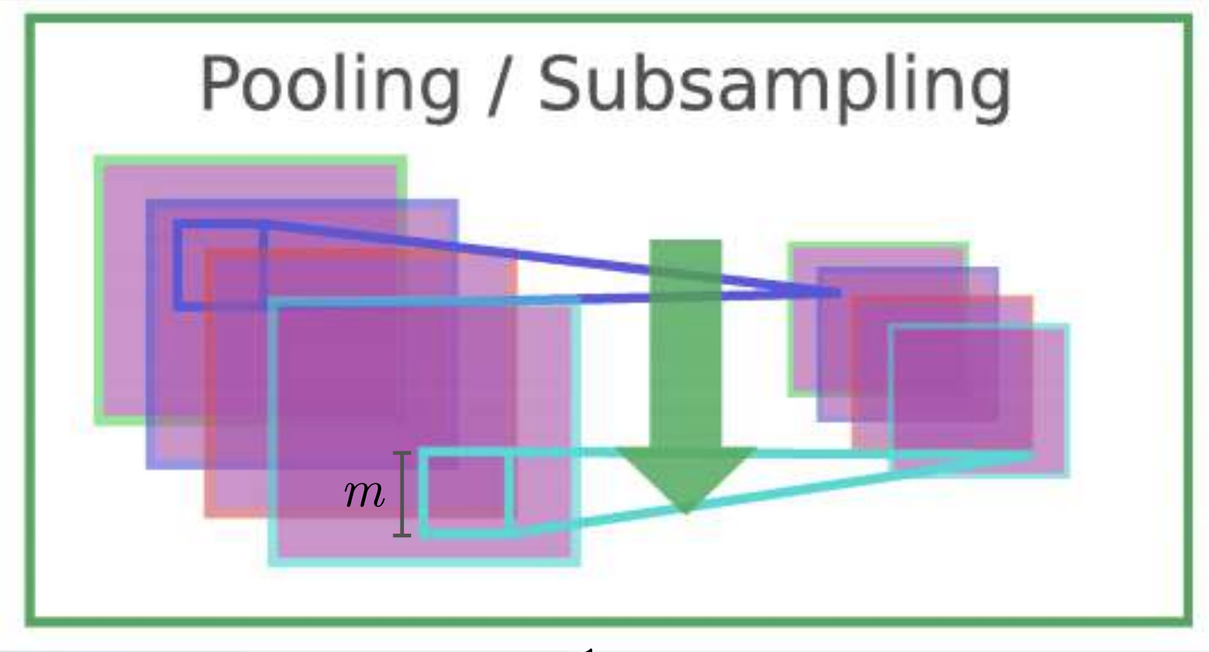
- $x_{i,j,k}$ is value of the i[th] feature map at position j,k
- $p$ is vertical index in local neighborhood
- $q$ is horizontal index in local neighborhood
- $y_{ijk}$ is pooled and subsampled layer

$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$$

# COMPUTER VISION

**Topics:** pooling and subsampling

- Third idea: pool hidden units in same neighborhood
  - an alternative to "max" pooling is "average" pooling




Jarret et al. 2009

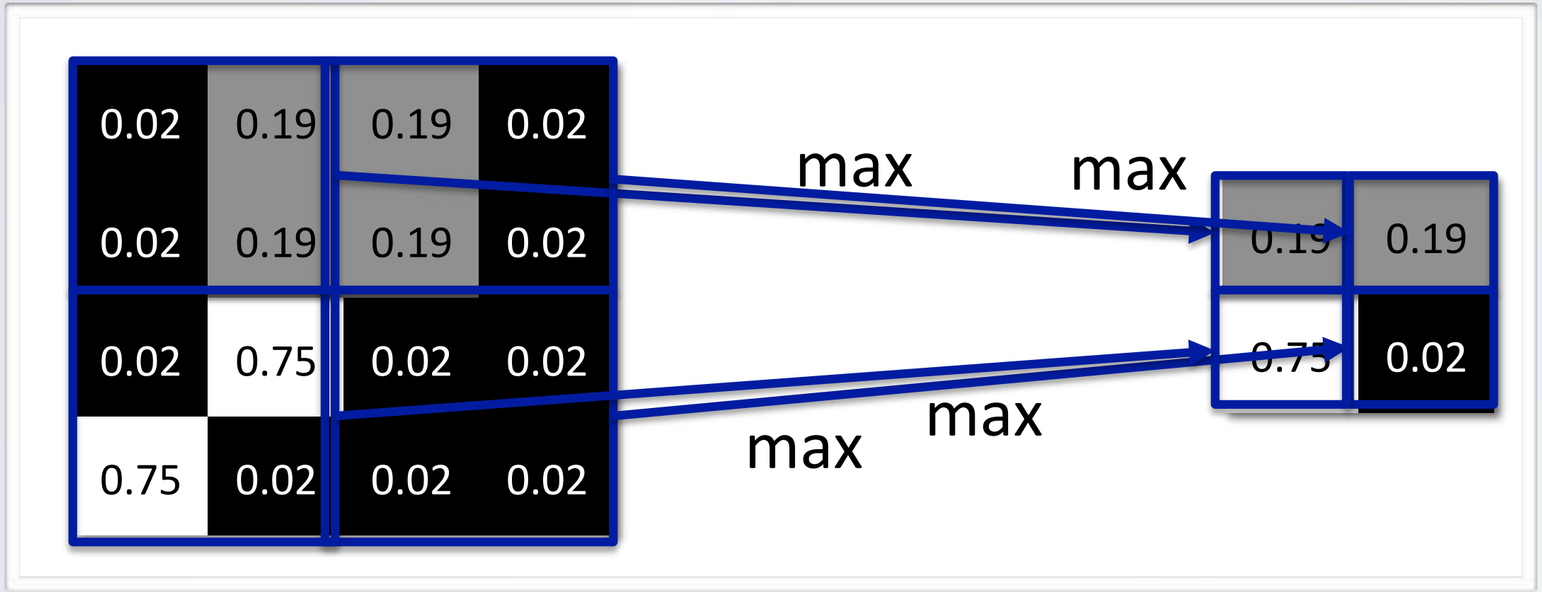
$$y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$$

- $x_{i,j,k}$ is value of the i[th] feature map at position j,k
- $p$ is vertical index in local neighborhood
- $q$ is horizontal index in local neighborhood
- $y_{ijk}$ is pooled and subsampled layer
- $m$ is the neighborhood height/width

# COMPUTER VISION

**Topics:** pooling and subsampling

- Illustration of pooling/subsampling operation

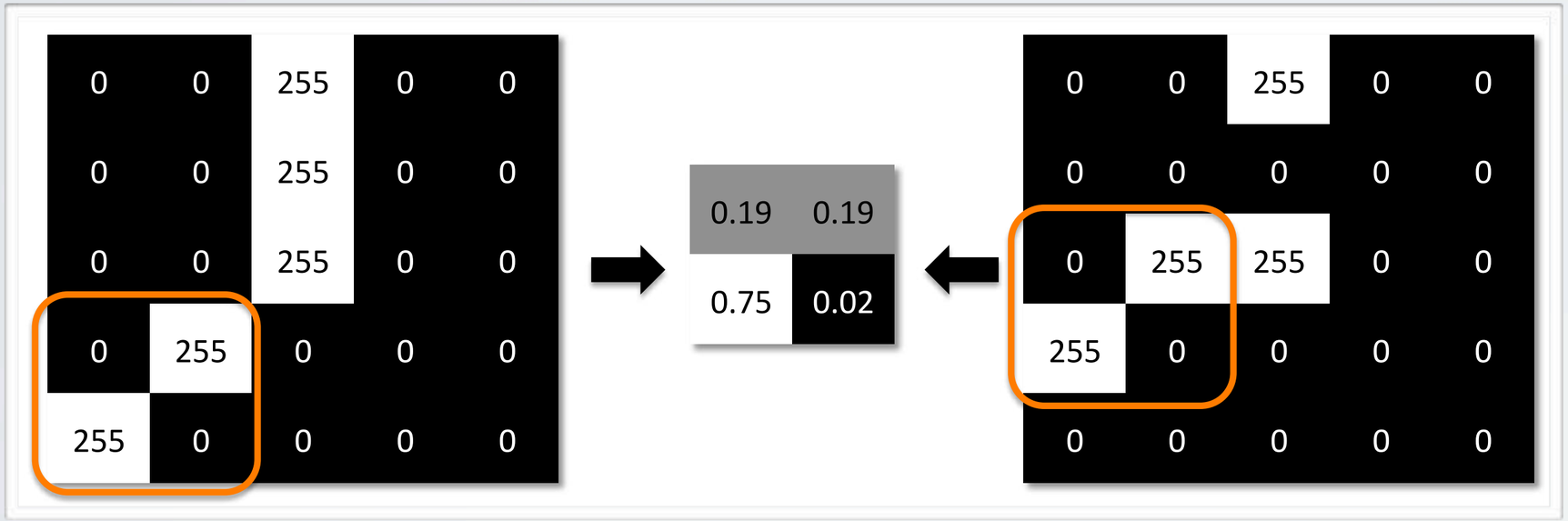



- Solves the following problems:
  - introduces invariance to local translations
  - reduces the number of hidden units in hidden layer

# COMPUTER VISION

**Topics:** pooling and subsampling

- Illustration of local translation invariance
  - both images given the same feature map after pooling/subsampling



# Neural networks

## Computer vision - convolutional network

# COMPUTER VISION

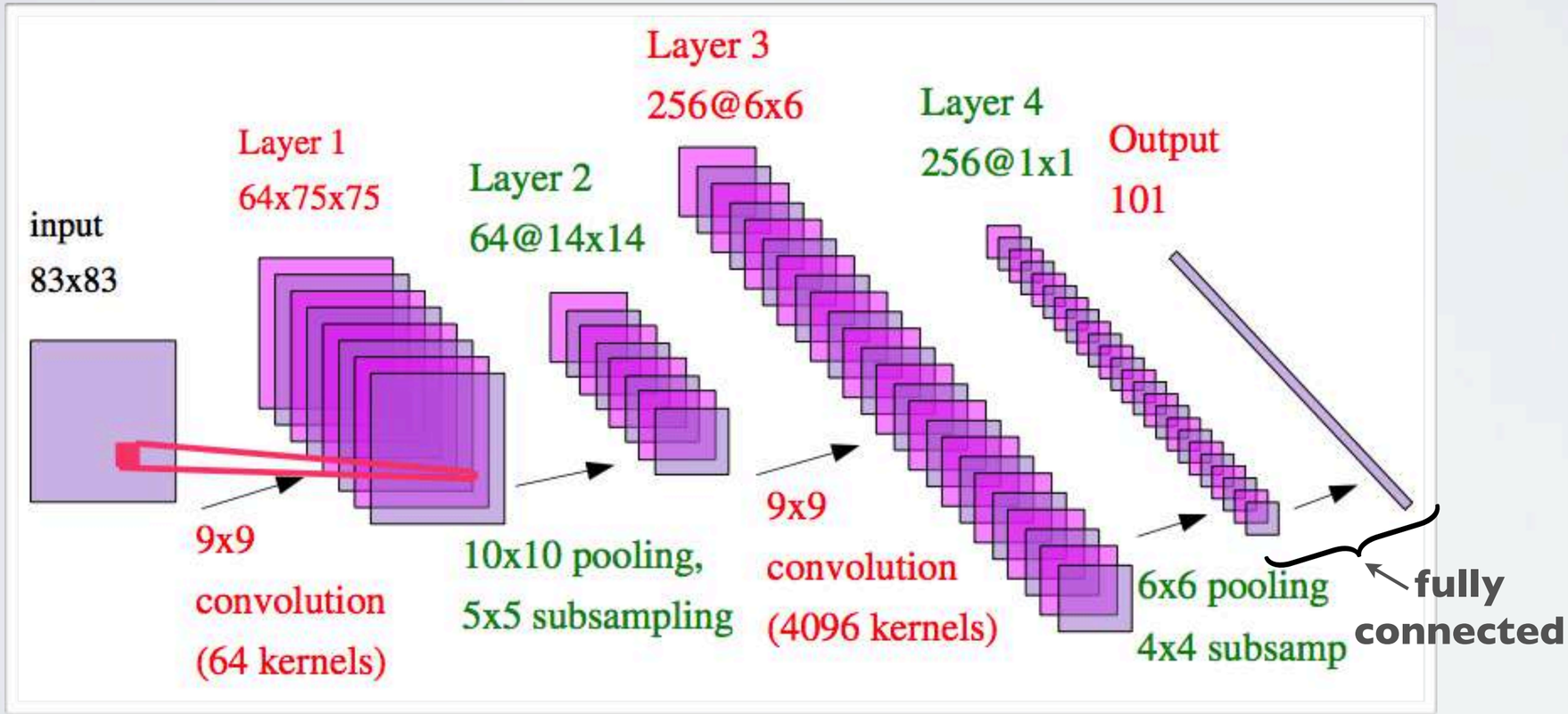
**Topics:** computer vision

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

- Convolutional neural network alternates between the convolutional and pooling layers




(from Yann Lecun)

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

- Output layer is a regular, fully connected layer with softmax non-linearity
  - output provides an estimate of the conditional probability of each class

- The network is trained by stochastic gradient descent
  - backpropagation is used similarly as in a fully connected network
  - we have seen how to pass gradients through element-wise activation function
  - we also need to pass gradients through the convolution operation and the pooling operation

# CONVOLUTIONAL NETWORK

**Topics:** gradient of convolution layer

- Let $l$ be the loss function
  - for convolution operation $y_j = x_i * k_{ij}$ the gradient for $x_i$ is

$$\nabla_{x_i} l = \sum_j (\nabla_{y_j} l) \underline{*} (W_{ij})$$

and the gradient for $W_{ij}$ is

$$\nabla_{W_{ij}} l = (\nabla_y l) * \widetilde{x}_i$$

where $\underline{*}$ is the convolution with zero padding and $\widetilde{x}_i$ is the row/column flipped version of $x_i$

# CONVOLUTIONAL NETWORK

**Topics:** gradient of pooling layer

- Let $l$ be the loss function
  - for max pooling operation $y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$ the gradient for $x_{ijk}$ is

$$\nabla_{x_{ijk}} l = 0 \text{ except for } \nabla_{x_{i,j+p',k+q'}} l = \nabla_{y_{ijk}} l$$

    where $p', q' = \operatorname{argmax} x_{i,j+p,k+q}$

    - in words, only the "winning" units in layer $x$ get the gradient from the pooled layer

  - for average pooling operation $y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$ the gradient for $x_{ijk}$ is

$$\nabla_x l = \frac{1}{m^2} \operatorname{upsample}(\nabla_y l)$$

    where $\operatorname{upsample}$ inverts subsampling

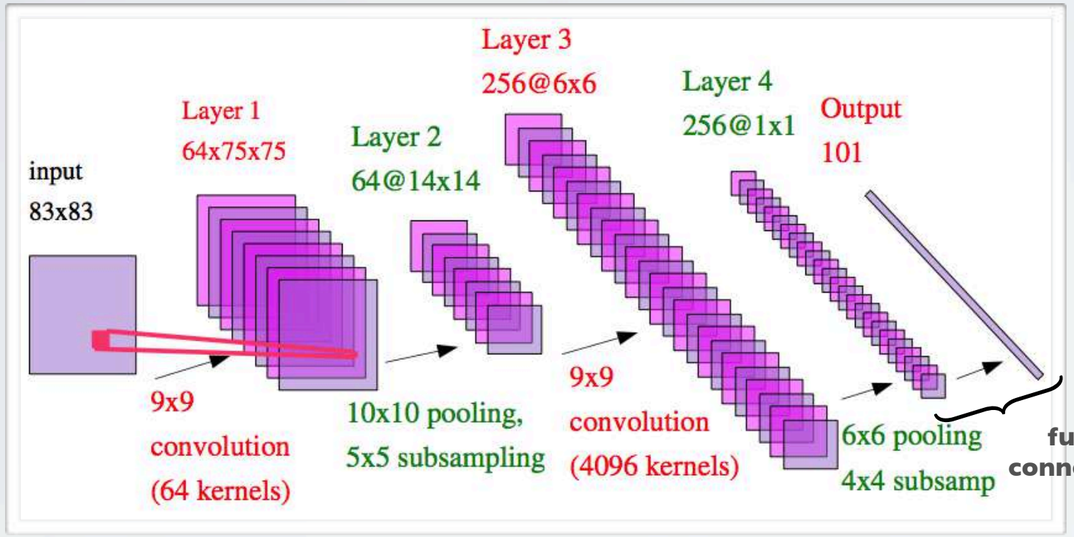
# Neural networks

## Computer vision - object recognition

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

- This architecture works well for handwritten character recognition




(from Yann Lecun)

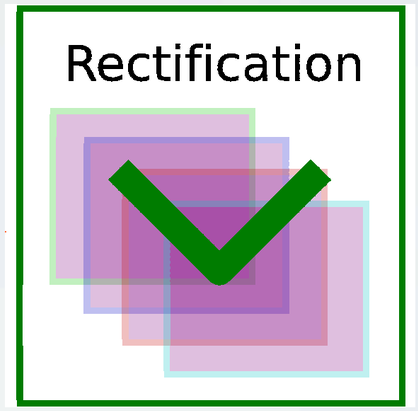
- It performs poorly on object recognition in general
  - we need to introduce other operations between

# CONVOLUTIONAL NETWORK

**Topics:** rectification layer

Jarret et al. 2009

- Rectification layer: $y_{ijk} = |x_{ijk}|$
  - introduces invariance to the sign of the unit in the previous layer
    - for instance, lose information of whether an edge is black-to-white or white-to-black



# CONVOLUTIONAL NETWORK

**Topics:** local contrast normalization layer

Jarret et al. 2009

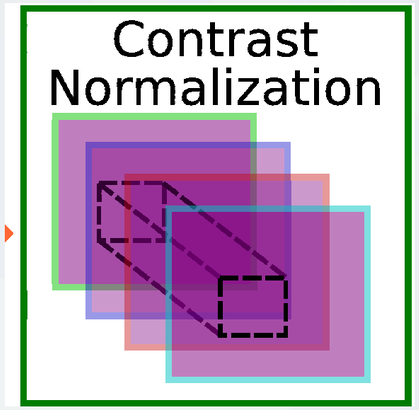
- Local contrast normalization:

$$v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} x_{i,j+p,k+q}$$

$$y_{ijk} = v_{ijk} / \max(c, \sigma_{jk})$$

$$\sigma_{jk} = (\sum_{ipq} w_{pq} v^2_{i,j+p,k+q})^{1/2}$$

$$\sum_{pq} w_{pq} = 1$$




where $c$ is a small constant to prevent division by 0

- reduces unit's activation if neighbors are also active
- creates competition between feature maps

# CONVOLUTIONAL NETWORK

**Topics:** local contrast normalization layer

Jarret et al. 2009

- Local contrast normalization:

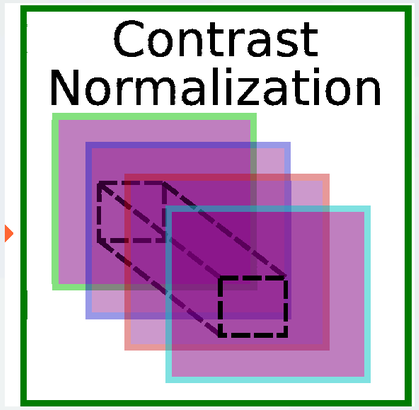
local average

$$v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} x_{i,j+p,k+q}$$

$$y_{ijk} = v_{ijk} / \max(c, \sigma_{jk})$$

$$\sigma_{jk} = (\sum_{ipq} w_{pq} v^2_{i,j+p,k+q})^{1/2}$$

$$\sum_{pq} w_{pq} = 1$$




where $c$ is a small constant to prevent division by 0

- reduces unit's activation if neighbors are also active
- creates competition between feature maps

# CONVOLUTIONAL NETWORK

**Topics:** local contrast normalization layer

Jarret et al. 2009

- Local contrast normalization:

local average

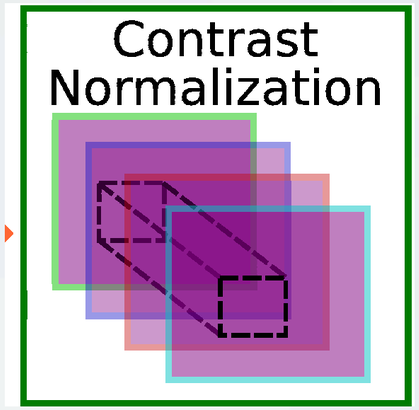
$$v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} x_{i,j+p,k+q}$$

$$y_{ijk} = v_{ijk} / \max(c, \sigma_{jk})$$

local std dev.

$$\sigma_{jk} = \left(\sum_{ipq} w_{pq} v^2_{i,j+p,k+q}\right)^{1/2}$$

$$\sum_{pq} w_{pq} = 1$$




where $c$ is a small constant to prevent division by 0

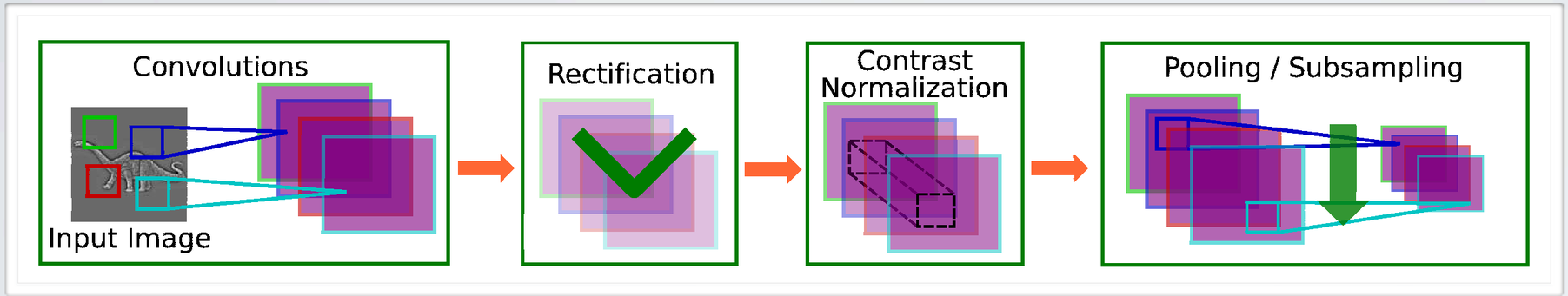
- reduces unit's activation if neighbors are also active
- creates competition between feature maps

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

Jarret et al. 2009

- These operations are inserted after the convolutions and before the pooling




- Images should also be preprocessed by
  - converting to grayscale (if appropriate)
  - resizing images to 150 x 150 pixels (use zero padding for non-square images)
  - removing (intra image) mean and dividing by standard deviation of the image
  - applying local contrast normalization

# Neural networks

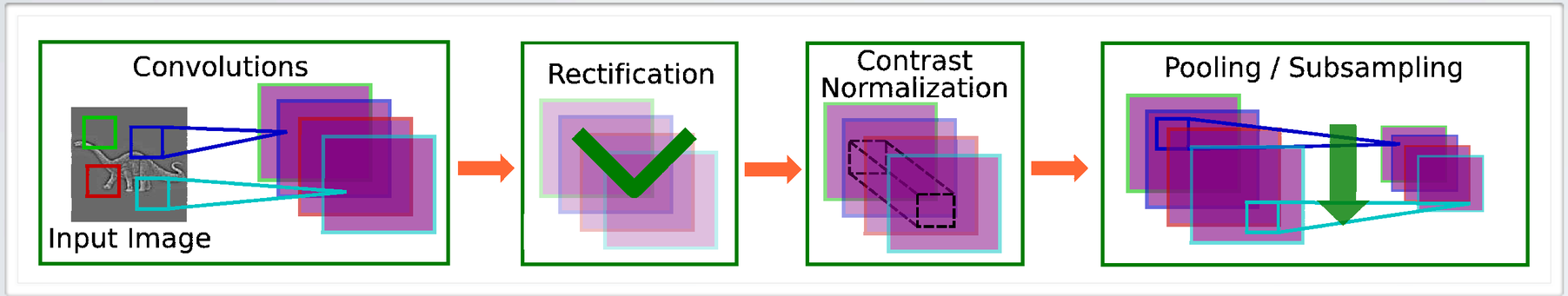
## Computer vision - example

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

Jarret et al. 2009

- These operations are inserted after the convolutions and before the pooling




- Images should also be preprocessed by
  - converting to grayscale (if appropriate)
  - resizing images to 150 x 150 pixels (use zero padding for non-square images)
  - removing (intra image) mean and dividing by standard deviation of the image
  - applying local contrast normalization

# CONVOLUTIONAL NETWORK

**Topics:** initialization of parameters

Jarret et al. 2009

- Initialization of parameters:
  - can do as in regular neural network and initialize them randomly
  - can also use unsupervised pretraining approach
    - to use unsupervised neural networks we've seen so far, we have to convert pretraining as a patch-wise learning problem
      - ✓ extract patches of the same as the receptive fields of the hidden units, at random positions
      - ✓ train an unsupervised neural network (RBM, autoencoder, sparse coding) on those patches
      - ✓ use weights connecting an input patch to each hidden unit to initialize each feature map parameters
      - ✓ map images through all feature maps and repeat previous steps, for as many layers as desired
- We will compare:
  - using random initialization ($R$) or unsupervised pretraining ($U$)
  - using fine-tuning of whole network ($^+$) or only training output layer (no $^+$)

# CONVOLUTIONAL NETWORK

**Topics:** convolutional network

Jarret et al. 2009

- Results on Caltech: $F_{CSG}$ = convolution layer
  $R$ = rectification layer
  $N$ = local contrast normalization layer
  $P_M$ = max pooling layer, $P_A$ = average pooling layer

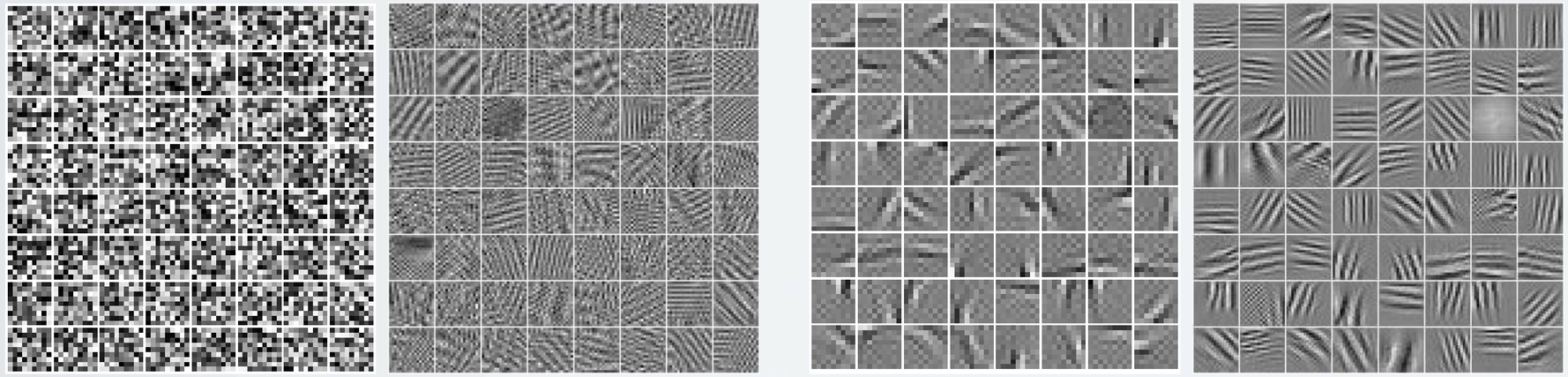
| **Single Stage System:** $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P}^{5\times5}]$ **- log_reg** | | | | | |
|---|---|---|---|---|---|
| | $\mathbf{R_{abs} - N - P_A}$ | $\mathbf{R_{abs} - P_A}$ | $\mathbf{N - P_M}$ | $\mathbf{N - P_A}$ | $\mathbf{P_A}$ |
| $\mathbf{U^+}$ | 54.2% | 50.0% | 44.3% | 18.5% | 14.5% |
| $\mathbf{R^+}$ | 54.8% | 47.0% | 38.0% | 16.3% | 14.3% |
| $\mathbf{U}$ | 52.2% | 43.3%($\pm$1.6) | 44.0% | 17.2% | 13.4% |
| $\mathbf{R}$ | 53.3% | 31.7% | 32.1% | 15.3% | 12.1%($\pm$2.2) |
| $\mathbf{G}$ | 52.3% | | | | |
| **Two Stage System:** $[64.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P}^{5\times5}] - [256.\mathbf{F}^{9\times9}_{\mathbf{CSG}} - \mathbf{R/N/P}^{4\times4}]$ **- log_reg** | | | | | |
| | $\mathbf{R_{abs} - N - P_A}$ | $\mathbf{R_{abs} - P_A}$ | $\mathbf{N - P_M}$ | $\mathbf{N - P_A}$ | $\mathbf{P_A}$ |
| $\mathbf{U^+U^+}$ | 65.5% | 60.5% | 61.0% | 34.0% | 32.0% |
| $\mathbf{R^+R^+}$ | 64.7% | 59.5% | 60.0% | 31.0% | 29.7% |
| $\mathbf{UU}$ | 63.7% | 46.7% | 56.0% | 23.1% | 9.1% |
| $\mathbf{RR}$ | 62.9% | 33.7%($\pm$1.5) | 37.6%($\pm$1.9) | 19.6% | 8.8% |
| $\mathbf{GT}$ | 55.8% | | | | |

# CONVOLUTIONAL NETWORK

**Topics:** random filters

Jarret et al. 2009

- Results on Caltech:
  - random filters are surprisingly good
  - turns out that random filters give units that are still sensitive to a particular frequency
    - can analyze this by finding input which maximizes the activation of a given hidden unit (with gradient ascent applied in input space)



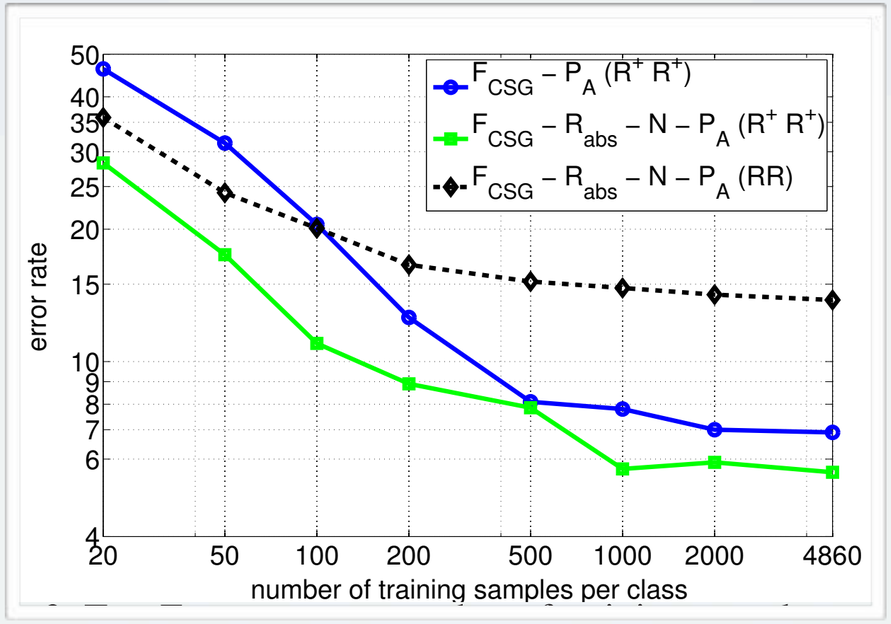
random filters | optimal input | learned filters | optimal input

# CONVOLUTIONAL NETWORK

**Topics:** importance of architecture

Jarret et al. 2009

- Results on Caltech:
  - choice of right architecture can be more important than learning algorithm
    - the use of rectification and local contrast normalization layers is important
    - this is particularly true if little training data
- Results on NORB:
  - architecture makes less of a difference with lots of training data per class
  - random filters are also not as good



# Neural networks

## Computer vision - data set expansion

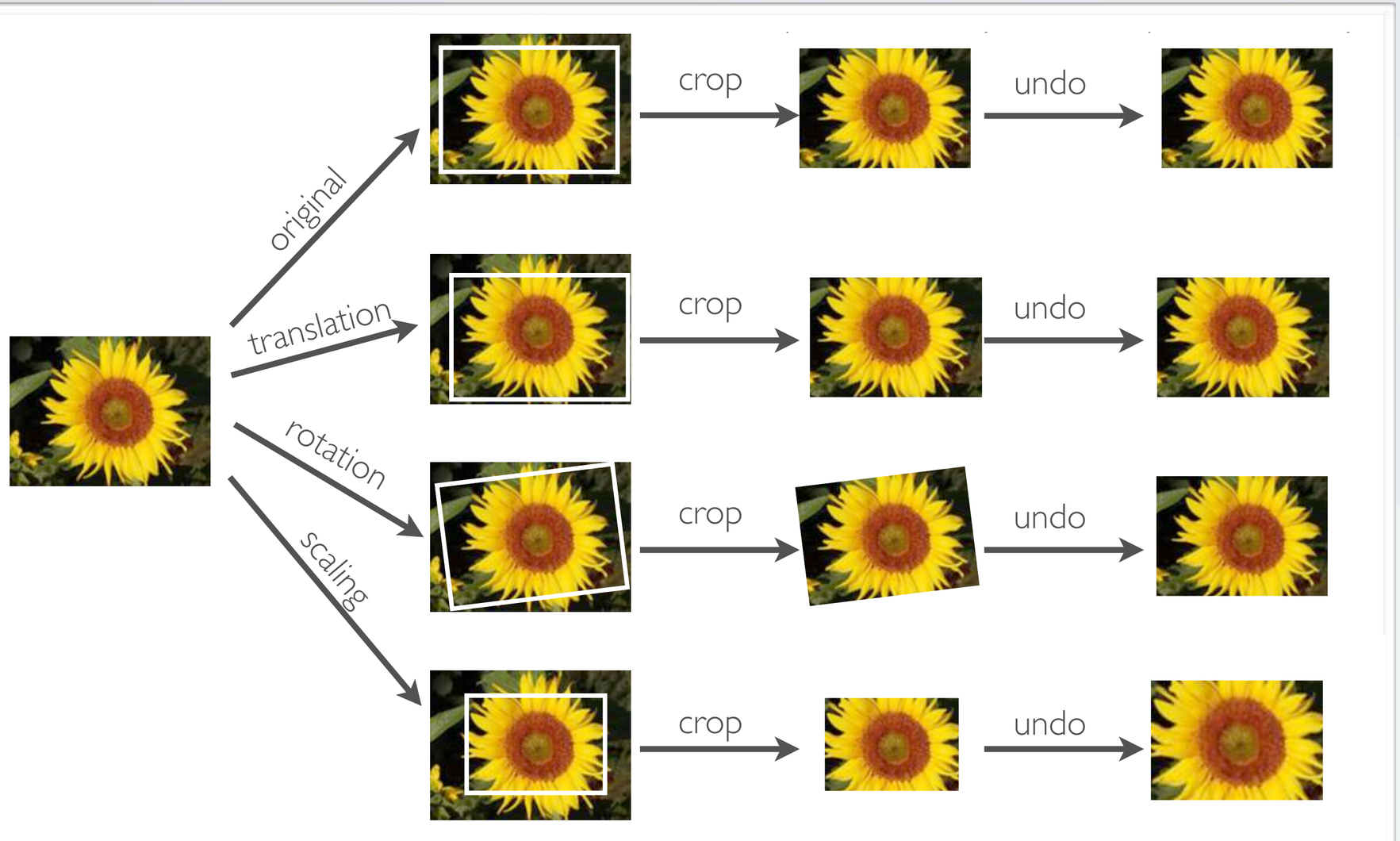
# INVARIANCE BY DATA SET EXPANSION

**Topics:** generating additional examples

- Invariances built-in in convolutional network:
  - small translations: due to convolution and max pooling
  - small illumination changes: due to local contrast normalization
- It is not invariant to other important variations such as rotations and scale changes
- However, it's easy to artificially generate data with such transformations
  - could use such data as additional training data
  - neural network will learn to be invariant to such transformations

# INVARIANCE BY DATA SET EXPANSION

**Topics:** generating additional examples

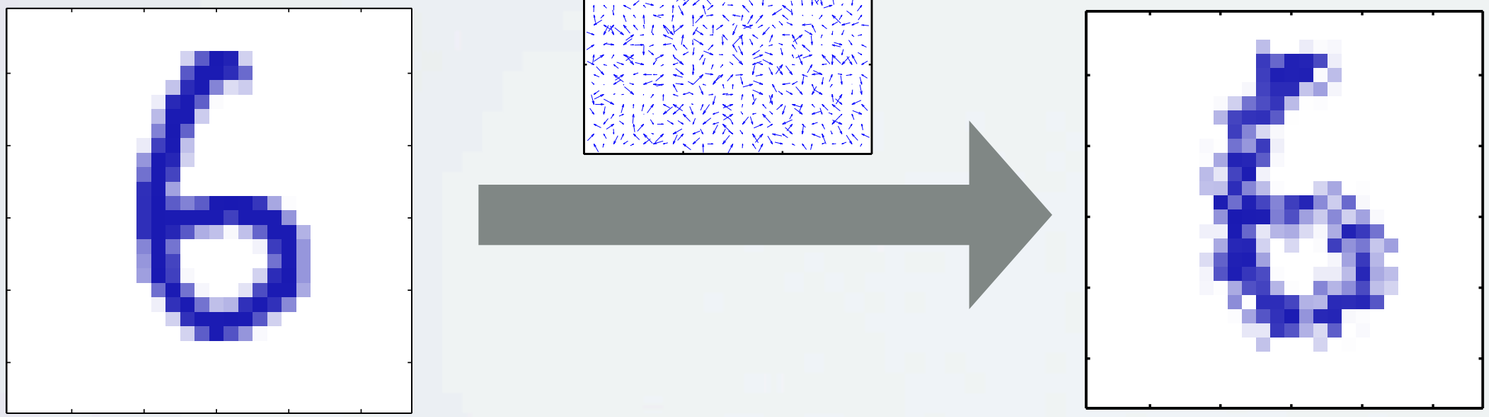


# INVARIANCE BY DATA SET EXPANSION

**Topics:** generating additional examples, distortion field

- Can add "elastic" deformations (useful in character recognition)
- We do this by applying a "distortion field" to the image
  - a distortion field specifies where to displace each pixel value

See Simard et al. for more detail



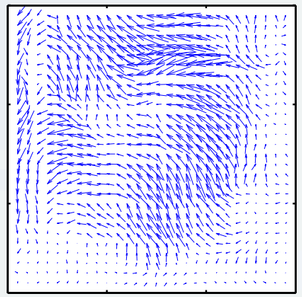

(from Bishop's book)

# INVARIANCE BY DATA SET EXPANSION

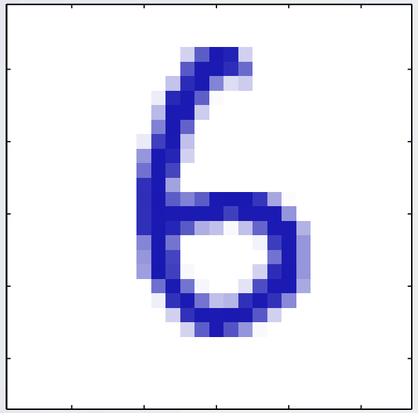
**Topics:** generating additional examples, distortion field

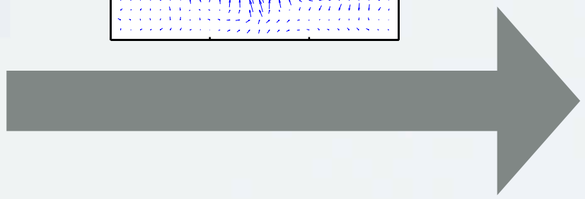
- Can add "elastic" deformations (useful in character recognition)
- We do this by applying a "distortion field" to the image
  - a distortion field specifies where to displace each pixel value

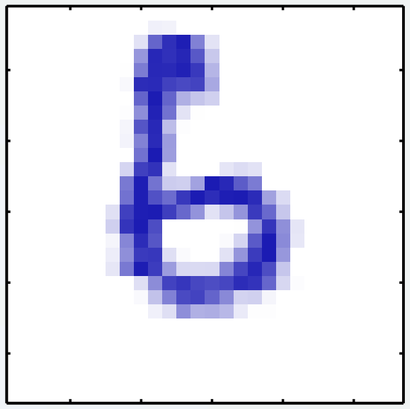
smoothed
random distortion



See Simard et al. for more detail







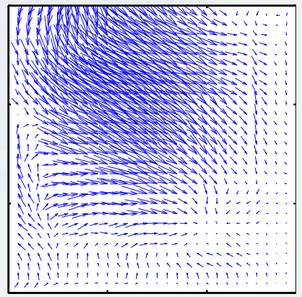
(from Bishop's book)

# INVARIANCE BY DATA SET EXPANSION

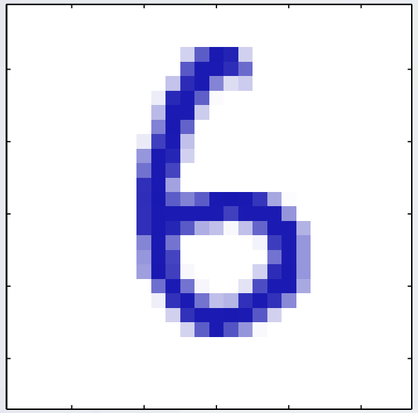
**Topics:** generating additional examples, distortion field

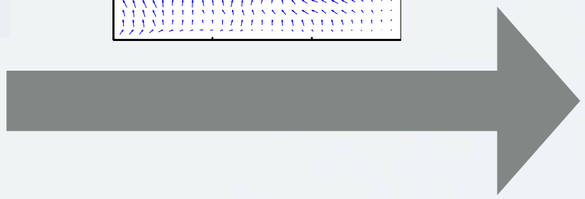
- Can add "elastic" deformations (useful in character recognition)
- We do this by applying a "distortion field" to the image
  - a distortion field specifies where to displace each pixel value

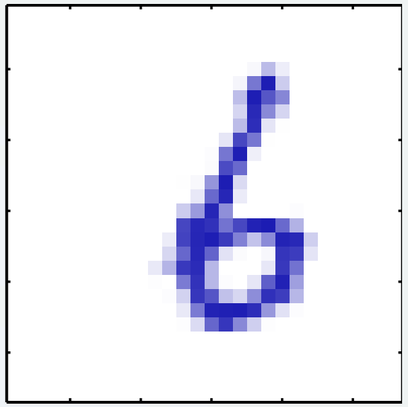
smoothed
random distortion



See Simard et al.
for more detail







(from Bishop's book)

# Neural networks

## Computer vision - convolutional RBM

# CONVOLUTIONAL RBM

**Topics:** convolutional RBM

Lee et al. 2009

- How about designing convolutional unsupervised networks
  - let's consider the case of the RBM
  - could use same convolutional connectivity between input ($\mathbf{v}$) and hidden layer ($\mathbf{h}$)

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h}))$$

$$E(\mathbf{v},\mathbf{h}) = -\sum_{k=1}^{K}\sum_{i,j=1}^{N_H}\sum_{r,s=1}^{N_W} h_{ij}^k W_{rs}^k v_{i+r-1,j+s-1}$$

$$-\sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{ij}^k - c\sum_{i,j=1}^{N_V} v_{ij}.$$

$$= -\sum_{k=1}^{K} h^k \bullet (\tilde{W}^k * v) - \sum_{k=1}^{K} b_k \sum_{i,j} h_{i,j}^k - c\sum_{i,j} v_{ij}$$

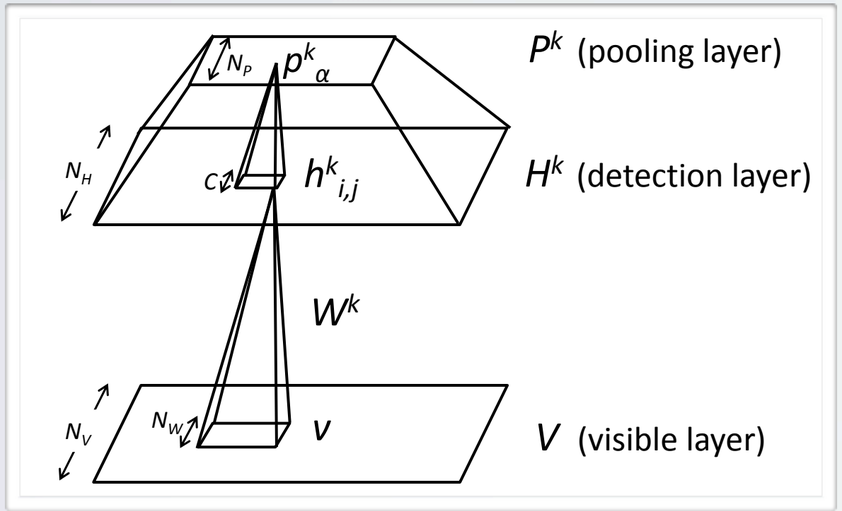
- $h^k_{ij}$ are the hidden units of the k[th] feature map
- $W^k_{rs}$ are the weights to the k[th] feature map
- $\tilde{W}^k$ are the weights with flipped rows and columns (convolution kernel)

# CONVOLUTIONAL RBM

**Topics:** convolutional RBM

Lee et al. 2009

- We can introduce a notion of probabilistic pooling
  - pooling unit $p_\alpha^k$ above is 1 only if at least one hidden unit $h_{i,j}^k$ in neighborhood is 1
  - within a pooling neighborhood, allow at most only a single unit $h_{i,j}^k$ equal to 1




$$I(h_{ij}^k) \triangleq b_k + (\tilde{W}^k * v)_{ij}$$

implies $p_\alpha^k$ is 1

$$P(h_{i,j}^k = 1|\mathbf{v}) = \frac{\exp(I(h_{i,j}^k))}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h_{i',j'}^k))}$$

$$P(p_\alpha^k = 0|\mathbf{v}) = \frac{1}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h_{i',j'}^k))}$$

implies all $h_{i,j}^k$ are 0

# CONVOLUTIONAL RBM

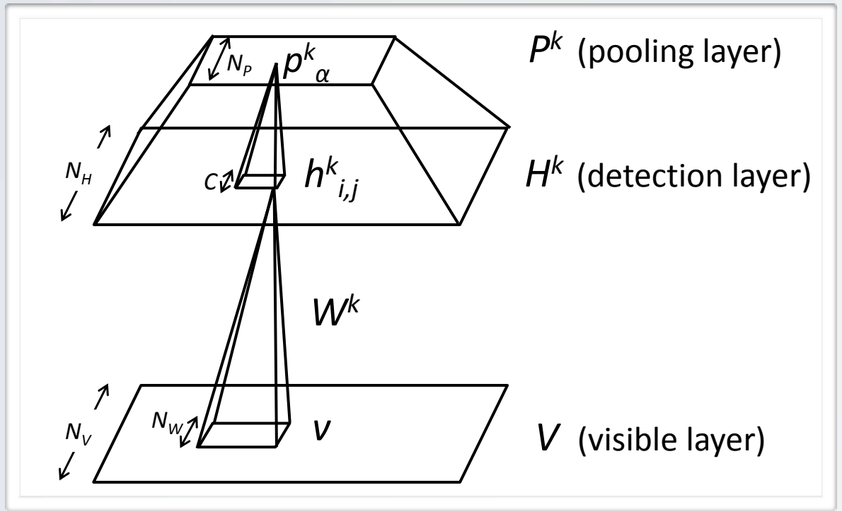
**Topics:** convolutional RBM

Lee et al. 2009

- Given the $h^k_{i,j}$ units, we sample each input independently:

$$P(v_{ij} = 1|\mathbf{h}) = \sigma((\sum_k W^k * h^k)_{ij} + c)$$

sigmoid




$$I(h^k_{ij}) \triangleq b_k + (\tilde{W}^k * v)_{ij}$$

implies $p^k_\alpha$ is 1

$$P(h^k_{i,j} = 1|\mathbf{v}) = \frac{\exp(I(h^k_{i,j}))}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h^k_{i',j'}))}$$

$$P(p^k_\alpha = 0|\mathbf{v}) = \frac{1}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h^k_{i',j'}))}$$

implies all $h^k_{i,j}$ are 0

# CONVOLUTIONAL RBM

**Topics:** convolutional RBM

Lee et al. 2009

- Using these adapted conditionals, we can perform contrastive divergence
  - energy gradients involve convolutions, similar to the backprop gradients in convolutional network
- Can stack convolutional RBMs
  - provides a pretraining procedure which doesn't require the extraction of patches
- See Lee et al. 2009 for more details