

**10417/10617**

**Intermediate Deep Learning:  
Fall2020**

Russ Salakhutdinov

Machine Learning Department  
[rsalakhu@cs.cmu.edu](mailto:rsalakhu@cs.cmu.edu)

Convolutional Neural  
Networks I

# Used Resources

- **Disclaimer:** Much of the material in this lecture was borrowed from Hugo Larochelle's class on Neural Networks:  
<https://sites.google.com/site/deeplearningsummerschool2016/>
- Some tutorial slides were borrowed from Rob Fergus' CIFAR tutorial on ConvNets:  
<https://sites.google.com/site/deeplearningsummerschool2016/speakers>
- Some slides were borrowed from Marc'Aurelio Ranzato's CVPR 2014 tutorial on Convolutional Nets  
<https://sites.google.com/site/lsvrtutorialcvpr14/home/deeplearning>

# Computer Vision

- Design algorithms that can process visual data to accomplish a given task:
  - For example, **object recognition**: Given an input image, identify which object it contains



# Computer Vision

- Our goal is to design neural networks that are specifically adapted for such problems
  - Must deal with very high-dimensional inputs:  $150 \times 150$  pixels = 22500 inputs, or  $3 \times 22500$  if RGB pixels
  - Can exploit the 2D topology of pixels (or 3D for video data)
  - Can build in invariance to certain variations: translation, illumination, etc.
- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units

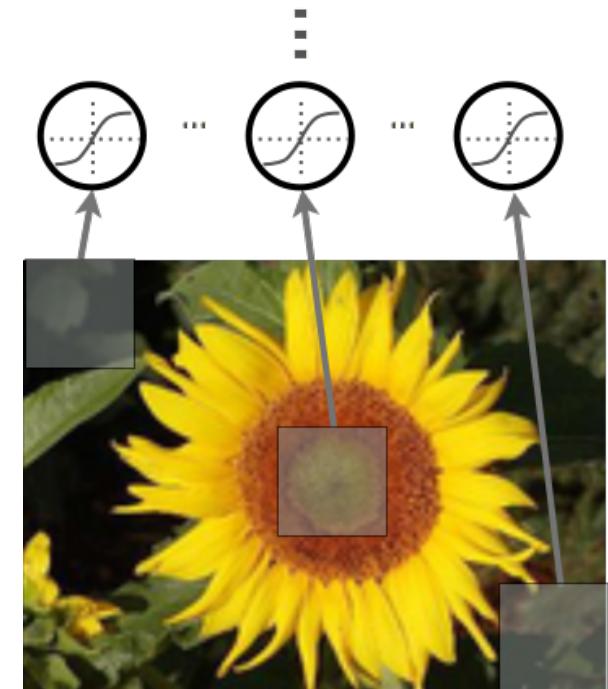
# Local Connectivity

- Use a **local connectivity** of hidden units

- Each hidden unit is connected only to a sub-region (patch) of the input image
- It is connected to all channels: 1 if grayscale, 3 (R, G, B) if color image

- Why local connectivity?

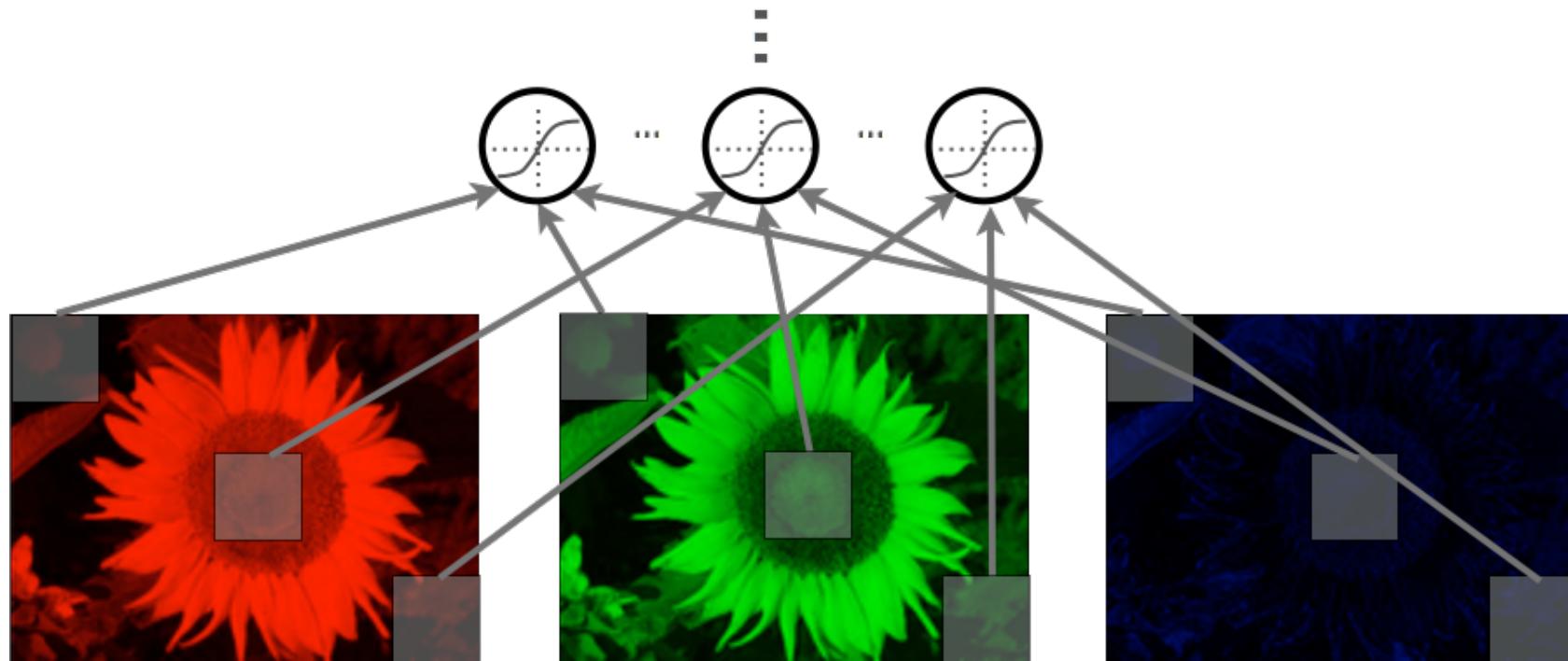
- Fully connected layer has **a lot of parameters** to fit, requires a lot of data
- Spatial correlation is local



$r$   = receptive field

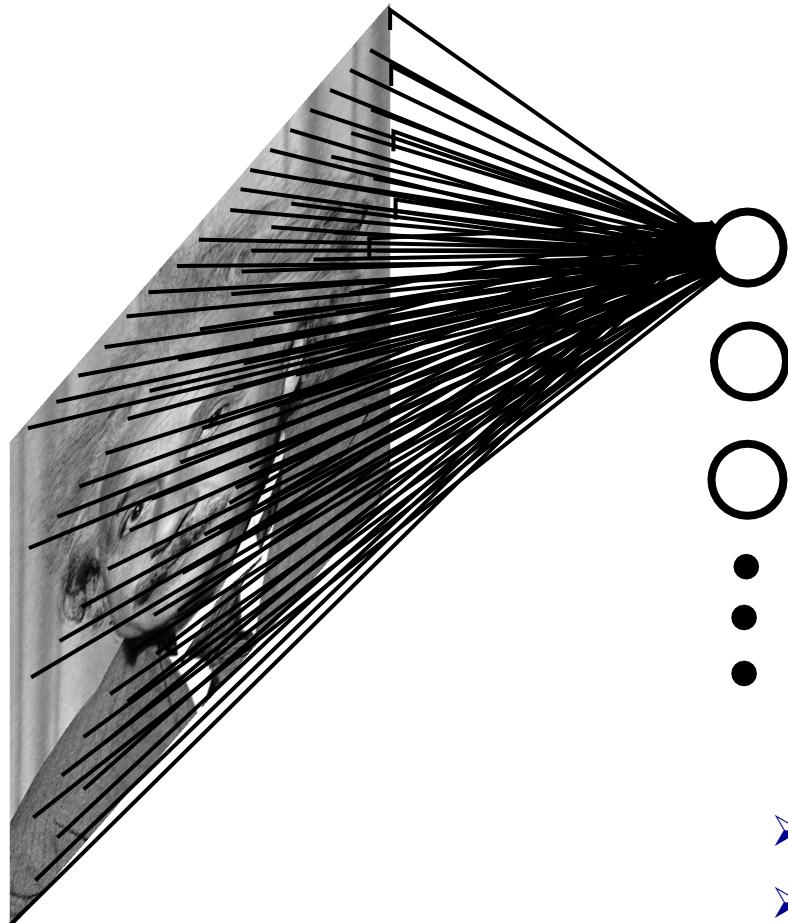
# Local Connectivity

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



# Local Connectivity

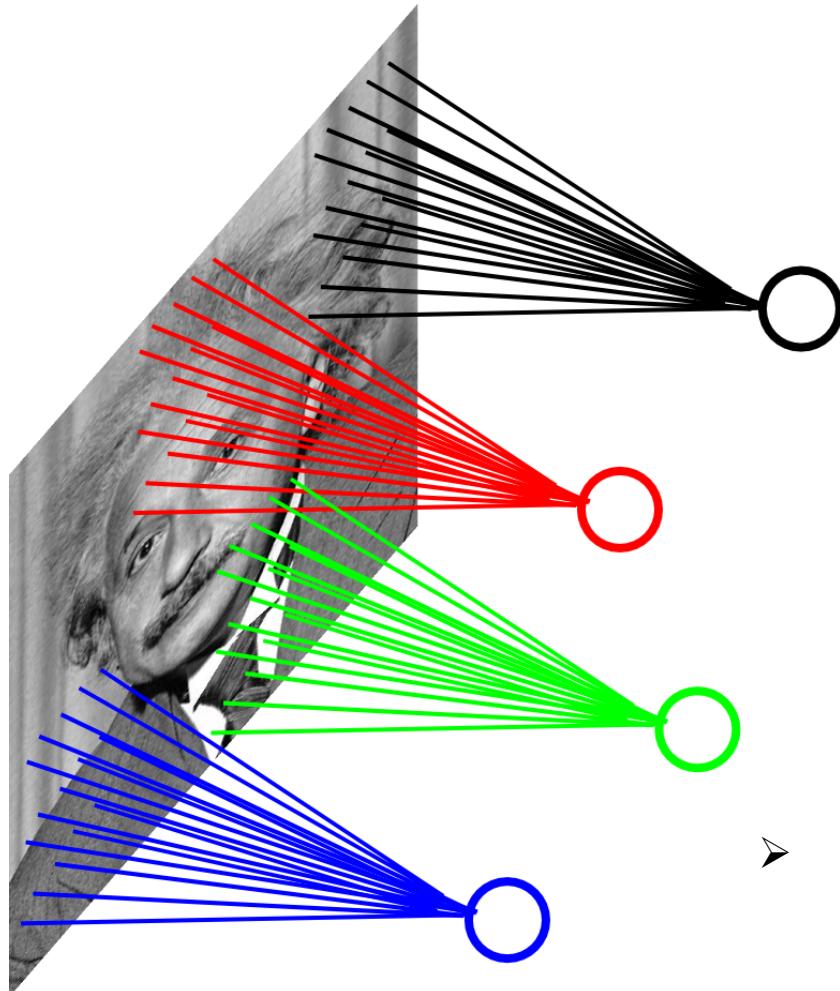
- Example: 200x200 image, 40K hidden units, **~2B parameters!**



- Spatial correlation is local
- Too many parameters, will require a lot of training data!

# Local Connectivity

- Example: 200x200 image, 40K hidden units, filter size 10x10, 4M parameters!



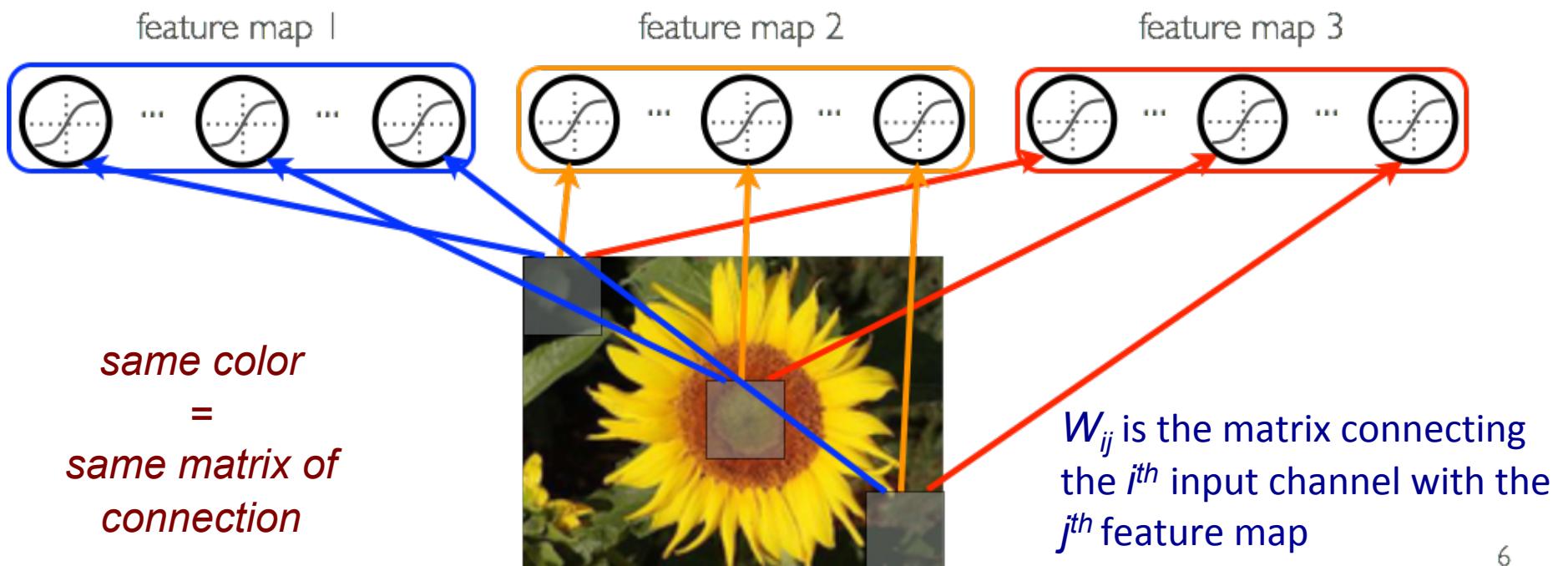
➤ This parameterization is good  
when input **image is registered**

# Computer Vision

- Our goal is to design neural networks that are specifically adapted for such problems
  - Must deal with very high-dimensional inputs:  $150 \times 150$  pixels = 22500 inputs, or  $3 \times 22500$  if RGB pixels
  - Can exploit the 2D topology of pixels (or 3D for video data)
  - Can build in invariance to certain variations: translation, illumination, etc.
- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units

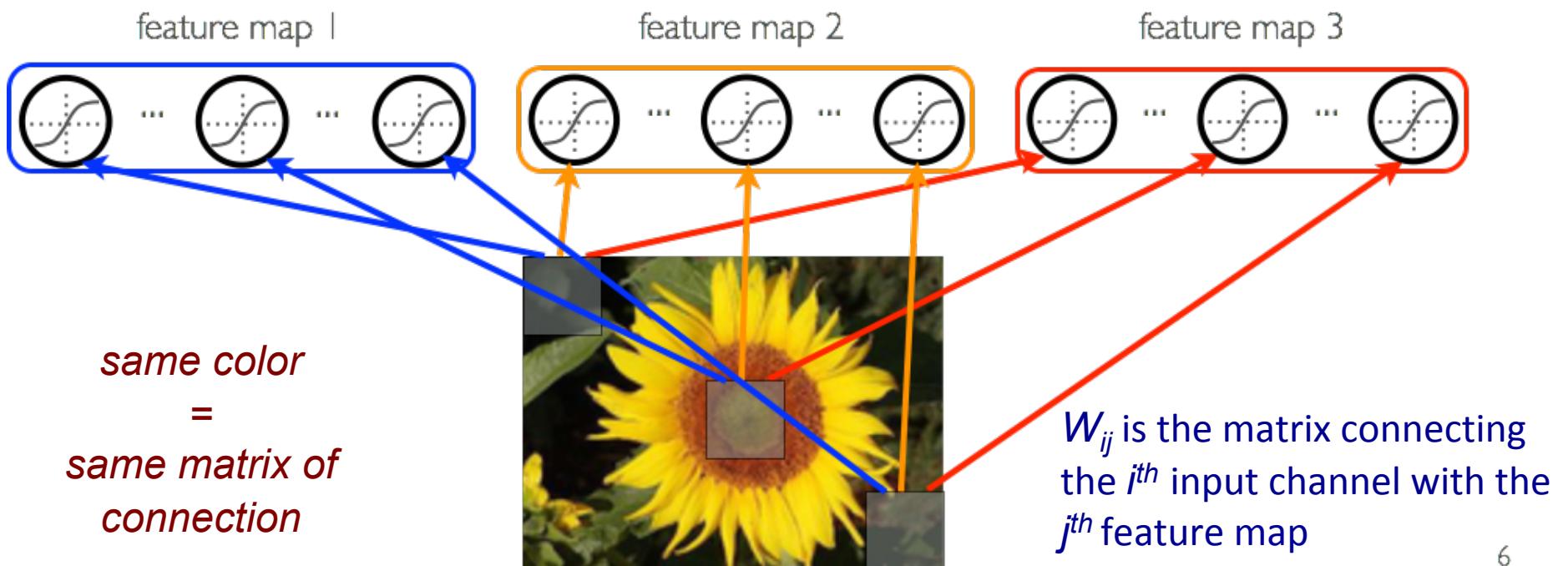
# Parameter Sharing

- Share matrix of parameters across some units
  - Units that are organized into the ‘feature map’ share parameters
  - Hidden units within a feature map cover different positions in the image



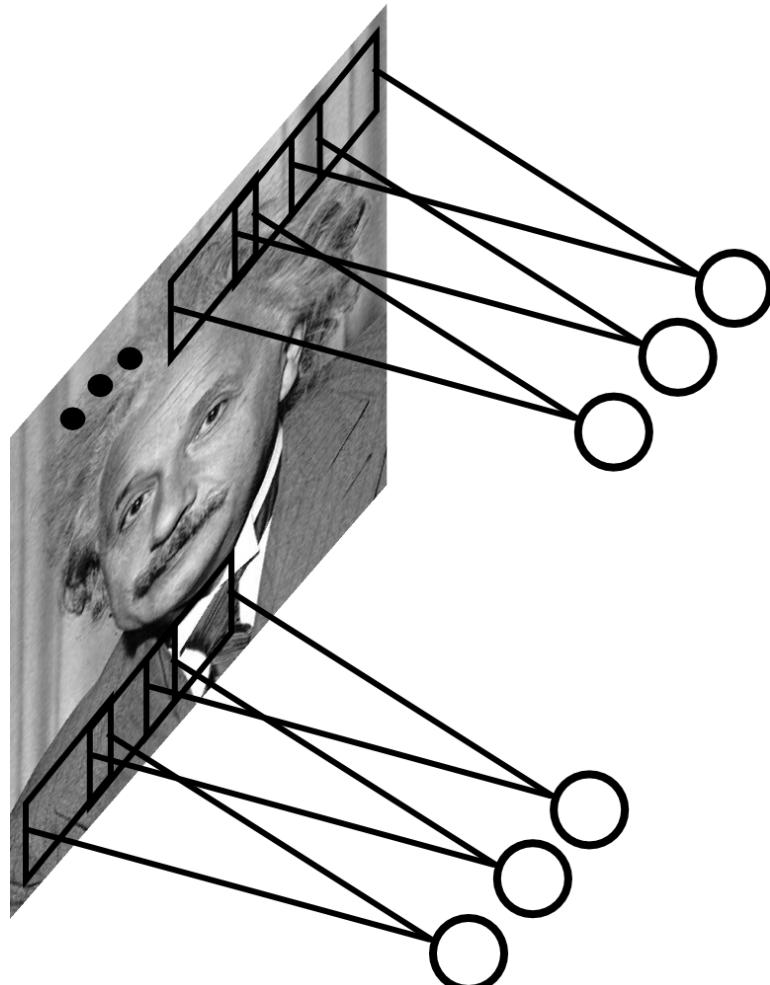
# Parameter Sharing

- Why parameter sharing?
  - Reduces even more the number of parameters
  - Will extract the same features at every position (**features are “equivariant”**)



# Parameter Sharing

- Share matrix of parameters across certain units



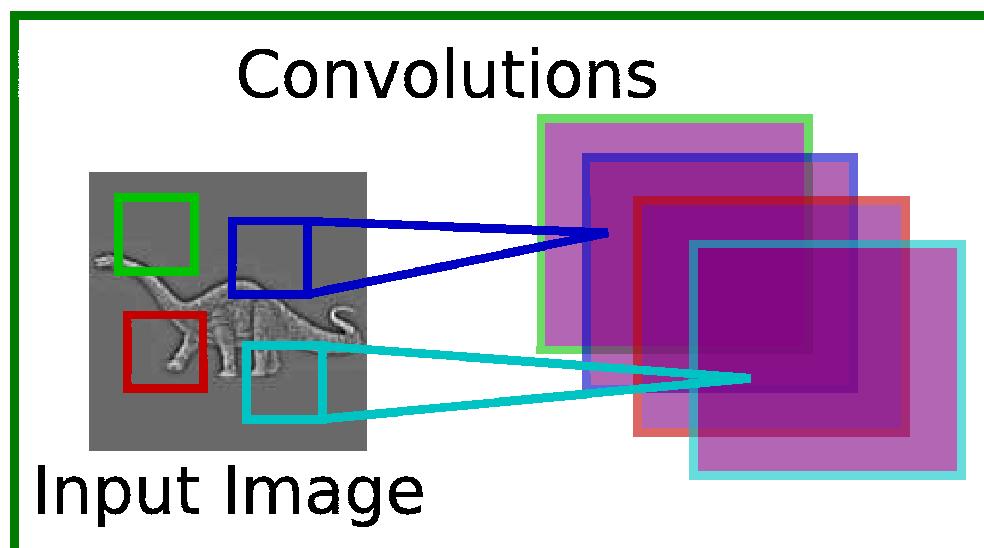
➤ **Convolutions** with certain kernels

# Computer Vision

- Our goal is to design neural networks that are specifically adapted for such problems
  - Must deal with very high-dimensional inputs:  $150 \times 150$  pixels = 22500 inputs, or  $3 \times 22500$  if RGB pixels
  - Can exploit the 2D topology of pixels (or 3D for video data)
  - Can build in invariance to certain variations: translation, illumination, etc.
- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units

# Parameter Sharing

- 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



Jarret et al. 2009

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

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

can add bias

# 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:

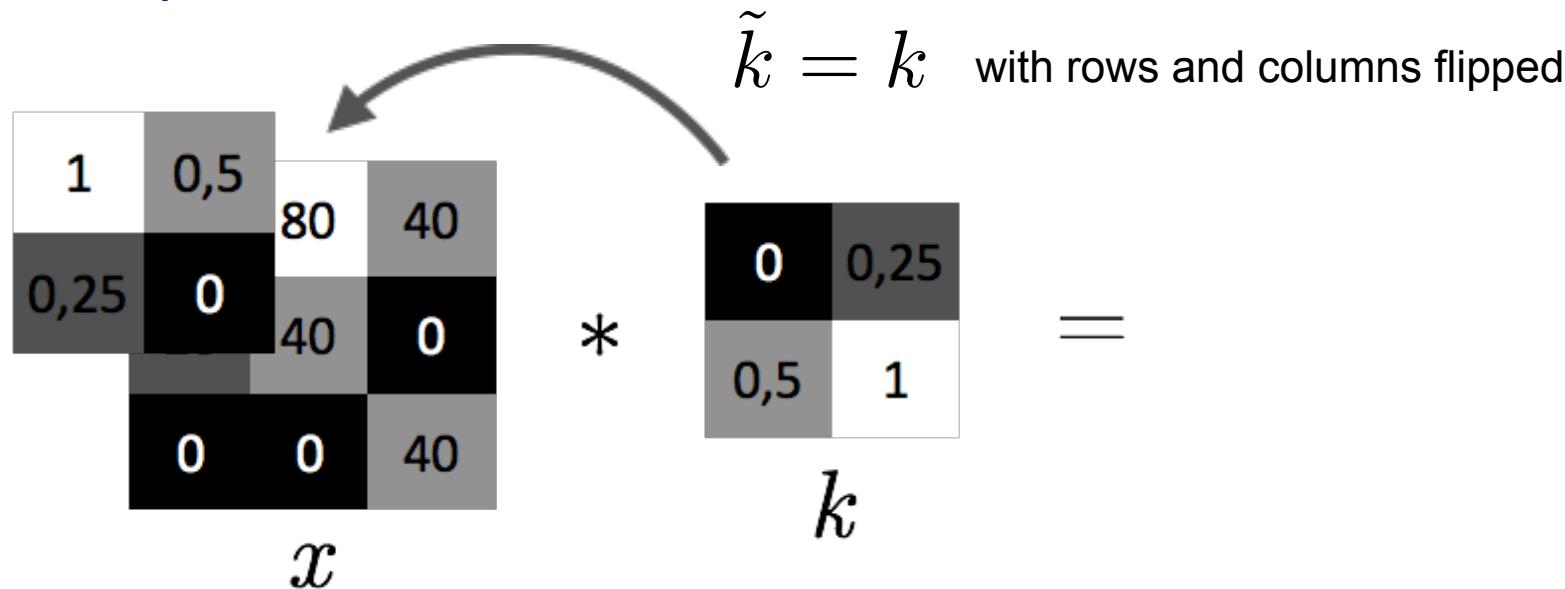
$$\begin{matrix} 0 & 80 & 40 \\ 20 & 40 & 0 \\ 0 & 0 & 40 \end{matrix} \quad * \quad \begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix} \quad = \quad k$$

# 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:



# 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:**  $1 \times 0 + 0.5 \times 80 + 0.25 \times 20 + 0 \times 40 = 45$

The diagram shows the convolution process between an input image  $x$  and a kernel  $k$ . The input image  $x$  is a 3x3 matrix with values:  $\begin{matrix} 1 & 0,5 \\ 0,25 & 0 \\ 0 & 0 \end{matrix}$ ,  $\begin{matrix} 80 & 40 \\ 40 & 0 \\ 0 & 40 \end{matrix}$ . The kernel  $k$  is a 2x2 matrix with values:  $\begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix}$ . The result of the convolution is a single value 45.

$$\begin{matrix} 1 & 0,5 \\ 0,25 & 0 \\ 0 & 0 \end{matrix} \times \begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix} = 45$$

$x$        $*$        $k$

# 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:**  $1 \times 80 + 0.5 \times 40 + 0.25 \times 40 + 0 \times 0 = 110$

The diagram shows the convolution of a 3x3 input image  $x$  with a 2x2 kernel  $k$ . The input  $x$  has values [1, 0.5, 40; 0.25, 0, 0; 0, 0, 40]. The kernel  $k$  has values [0, 0.25; 0.5, 1]. The result of the convolution is 110.

$$\begin{matrix} 1 & 0,5 & 40 \\ 0,25 & 0 & 0 \\ 0 & 0 & 40 \end{matrix} * \begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix} = \begin{matrix} 45 & 110 \end{matrix}$$

$x$                                      $k$

# 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:**  $1 \times 20 + 0.5 \times 40 + 0.25 \times 0 + 0 \times 0 = 40$

The diagram shows the convolution process between two 3x3 matrices. On the left is the input image  $x$ , with values: top row [0, 80, 40], second row [20, 40, 0], bottom row [0, 0, 40]. Middle column values are 1, 0.5, 0.25; rightmost column values are 0.5, 1, 0. The right side shows the kernel  $k$  with values: top row [0, 0.25], second row [0.5, 1]. The result of the convolution is shown on the right, resulting in values [45, 110] in the first two columns, and an empty third column.

$$\begin{matrix} 0 & 80 & 40 \\ 20 & 40 & 0 \\ 0 & 0 & 40 \end{matrix} * \begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix} = \begin{matrix} 45 & 110 \\ 40 & \end{matrix}$$

$x$        $k$

# 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:**  $1 \times 40 + 0.5 \times 0 + 0.25 \times 0 + 0 \times 40 = 40$

The diagram illustrates the computation of a discrete convolution. On the left, an input image  $x$  is shown as a 3x3 grid of values: 0, 80, 40; 20, 40, 0; 1, 0.5, 0. The middle part shows the convolution operation with a kernel  $k$ , which is a 2x2 grid of values: 0, 0.25; 0.5, 1. The result of the convolution is shown on the right as a 2x2 grid: 45, 110; 40, 40.

$$\begin{matrix} 0 & 80 & 40 \\ 20 & 40 & 0 \\ 1 & 0,5 & 0 \end{matrix} * \begin{matrix} 0 & 0,25 \\ 0,5 & 1 \end{matrix} = \begin{matrix} 45 & 110 \\ 40 & 40 \end{matrix}$$

$x$                                      $k$

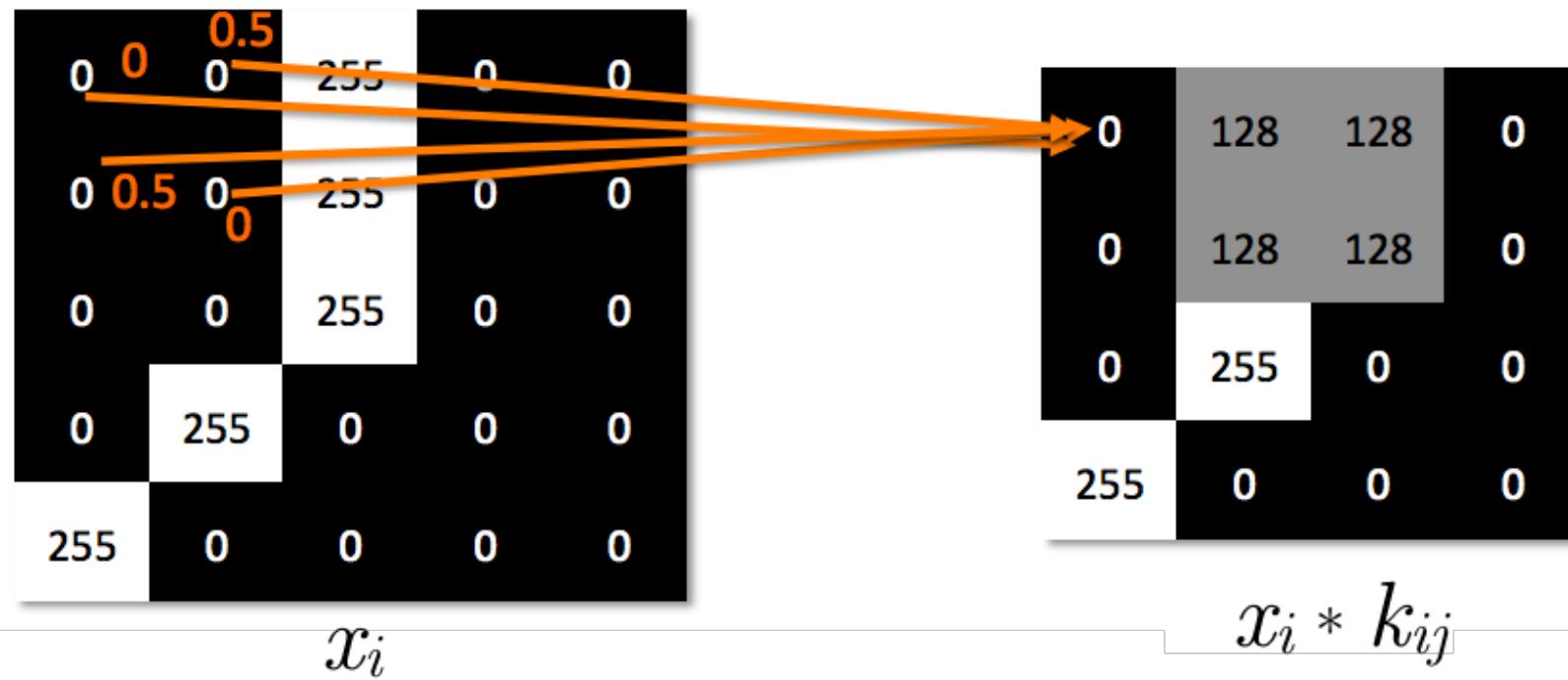
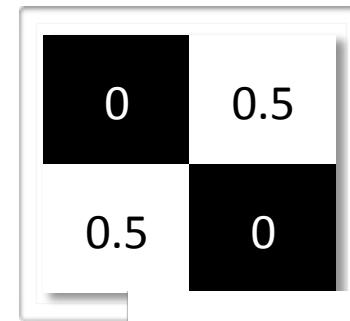
# Discrete Convolution

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

# Example

- Illustration:

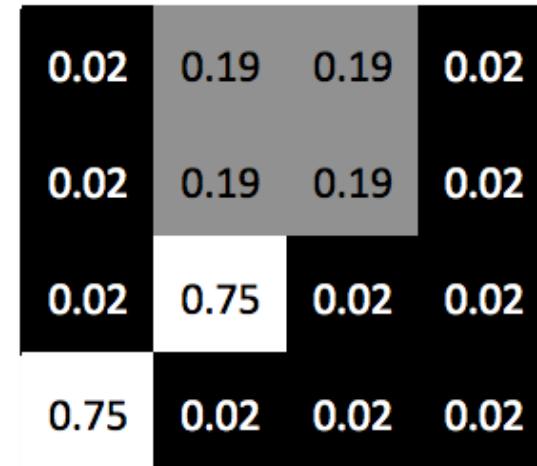
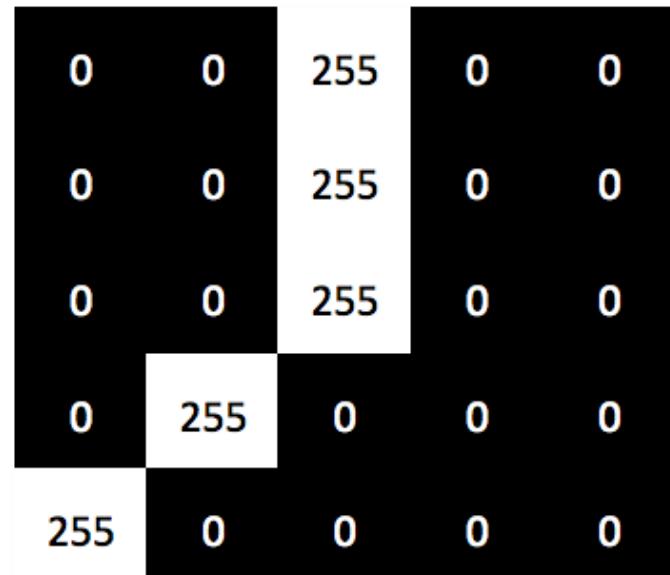
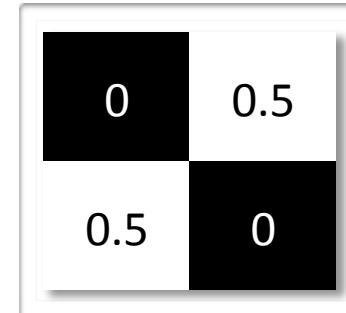
$$x * k_{ij}, \text{ where } W_{ij} = \tilde{W}_{ij}$$



# Example

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

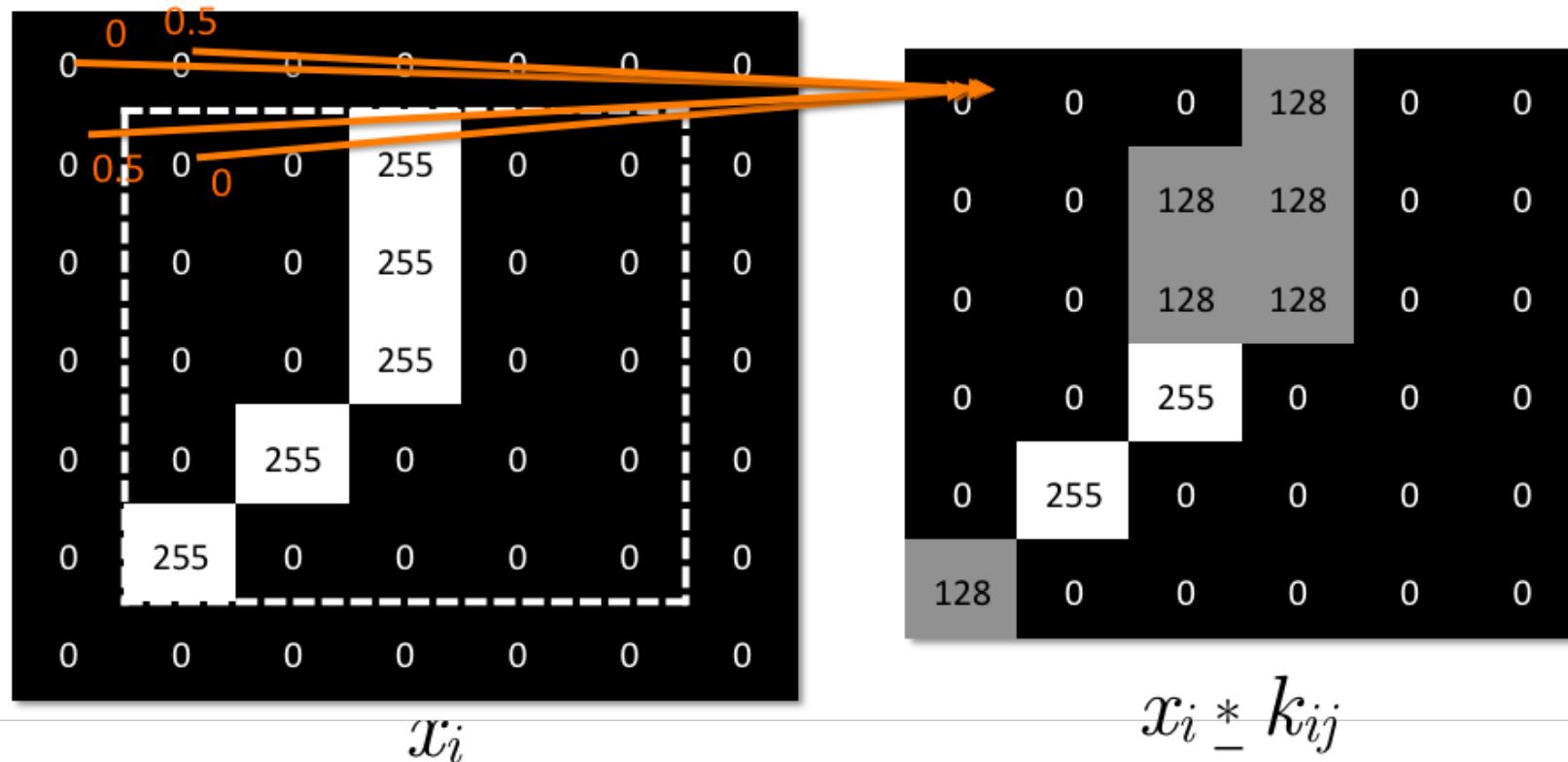
$$x * k_{ij}, \text{ where } W_{ij} = \tilde{W}_{ij}$$



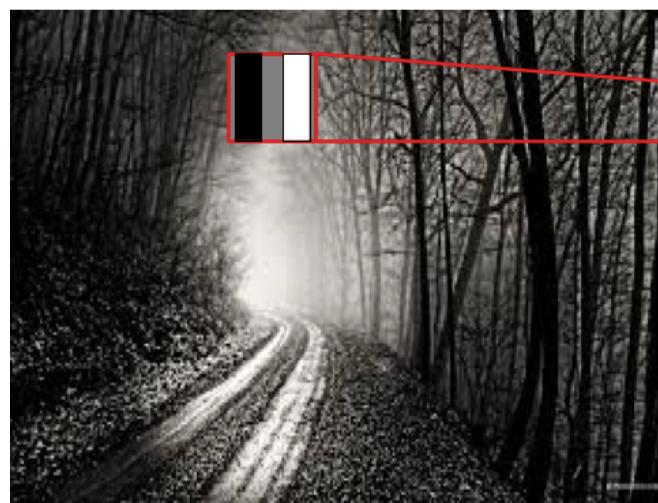
$$\text{sigm}(0.02 x_i * k_{ij} - 4)$$

# Example

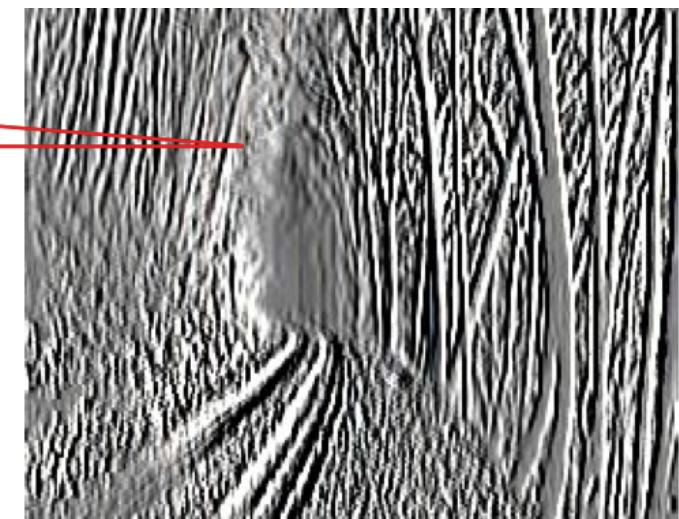
- Can use “zero padding” to allow going over the borders ( \* )



# Example

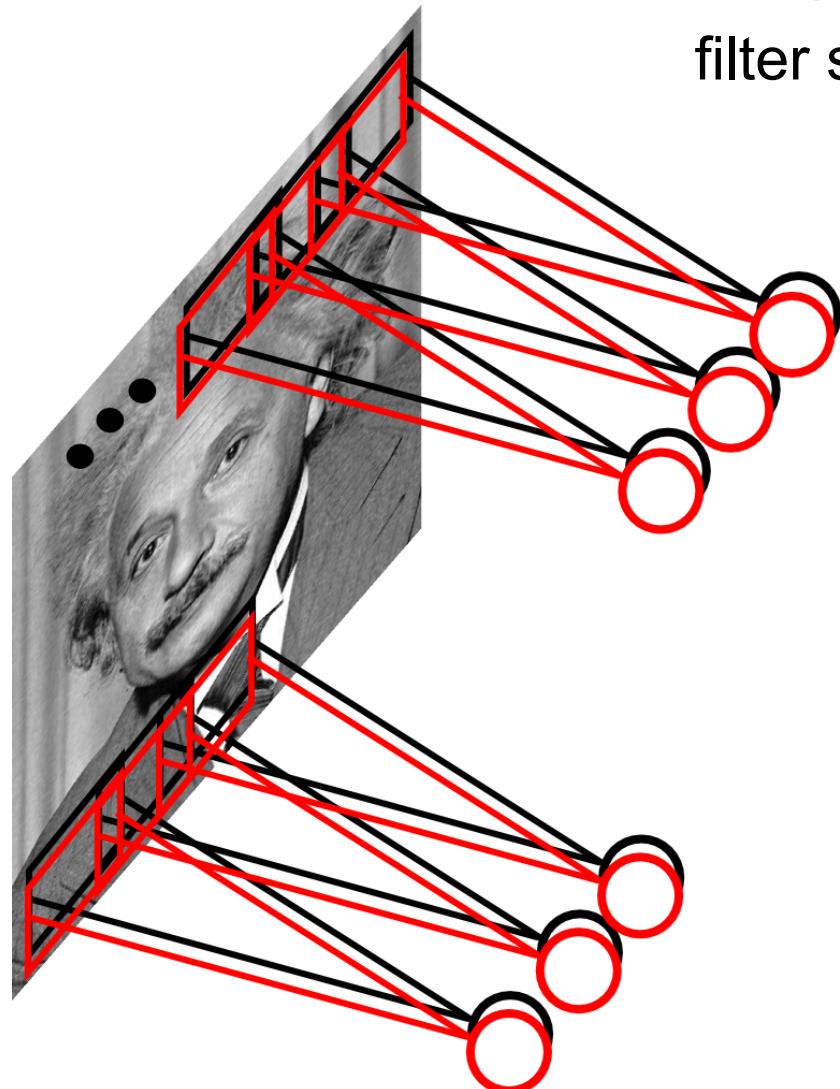


$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



# Multiple Feature Maps

- Example: 200x200 image, 100 filters, filter size 10x10, 10K parameters



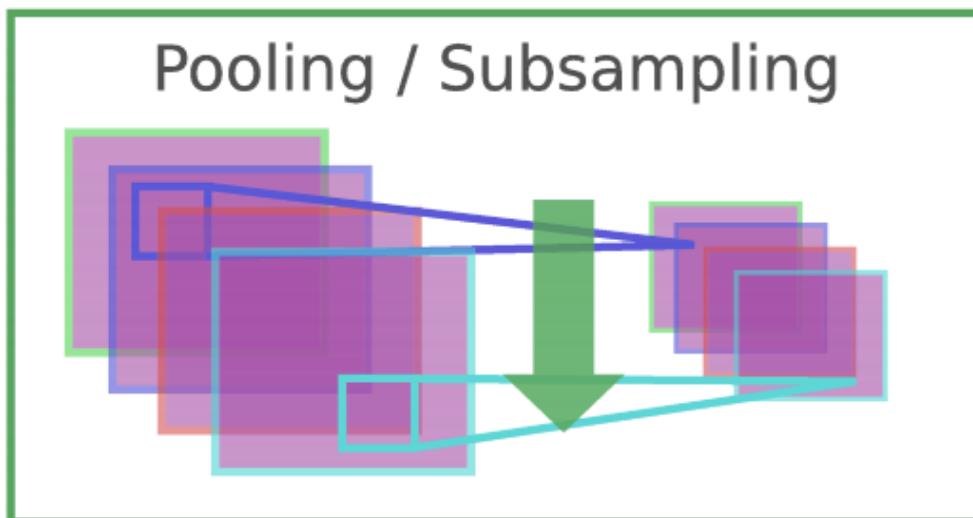
# Computer Vision

- Our goal is to design neural networks that are specifically adapted for such problems
  - Must deal with very high-dimensional inputs:  $150 \times 150$  pixels = 22500 inputs, or  $3 \times 22500$  if RGB pixels
  - Can exploit the 2D topology of pixels (or 3D for video data)
  - Can build in invariance to certain variations: translation, illumination, etc.
- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units

# Pooling

- Pool hidden units in same neighborhood
  - **pooling** is performed in non-overlapping neighborhoods (subsampling)

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



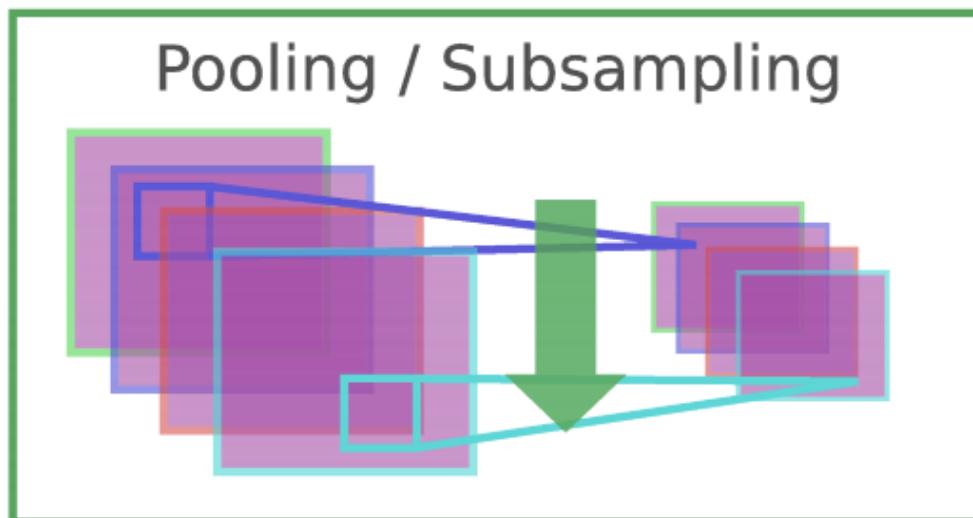
- $x_i$  is the  $i^{\text{th}}$  channel of input
- $x_{i,j,k}$  is value of the  $i^{\text{th}}$  feature map at position  $j,k$
- $p$  is vertical index in local neighborhood
- $q$  is horizontal index in local neighborhood
- $y_{ijk}$  is pooled / subsampled layer

Jarret et al. 2009

# Pooling

- Pool hidden units in same neighborhood
  - an alternative to “max” pooling is “average” pooling

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

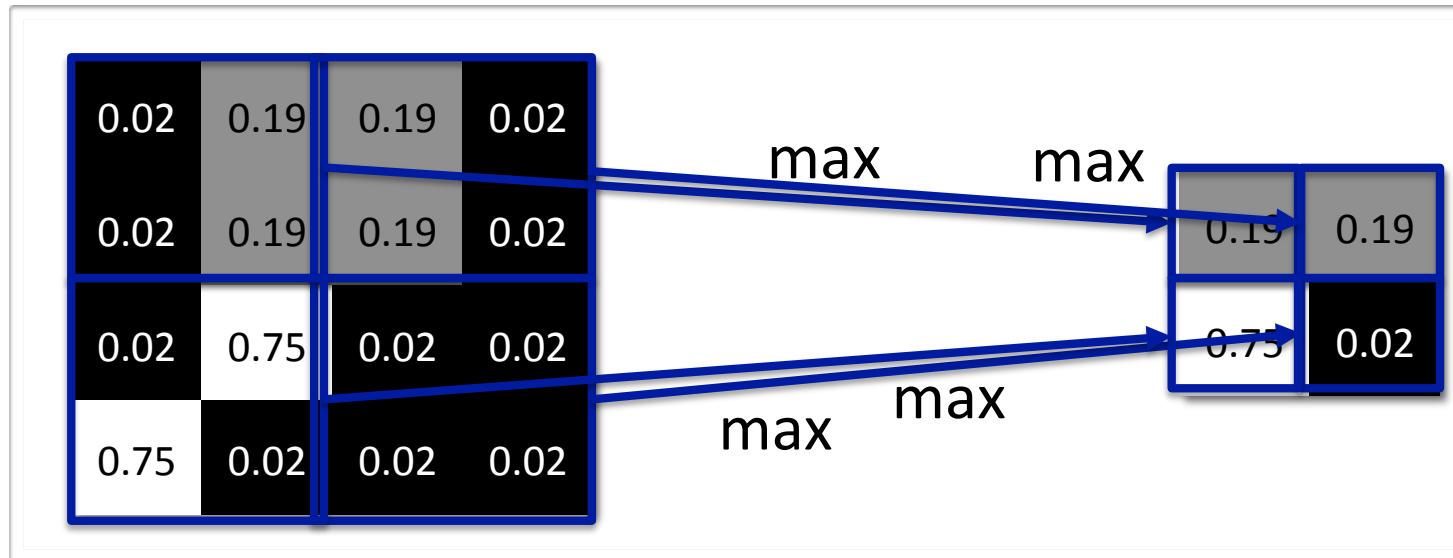


Jarret et al. 2009

- $x_i$  is the  $i^{\text{th}}$  channel of input
- $x_{i,j,k}$  is value of the  $i^{\text{th}}$  feature map at position  $j,k$
- $p$  is vertical index in local neighborhood
- $q$  is horizontal index in local neighborhood
- $y_{ijk}$  is pooled / subsampled layer
- $m$  is the neighborhood height/width

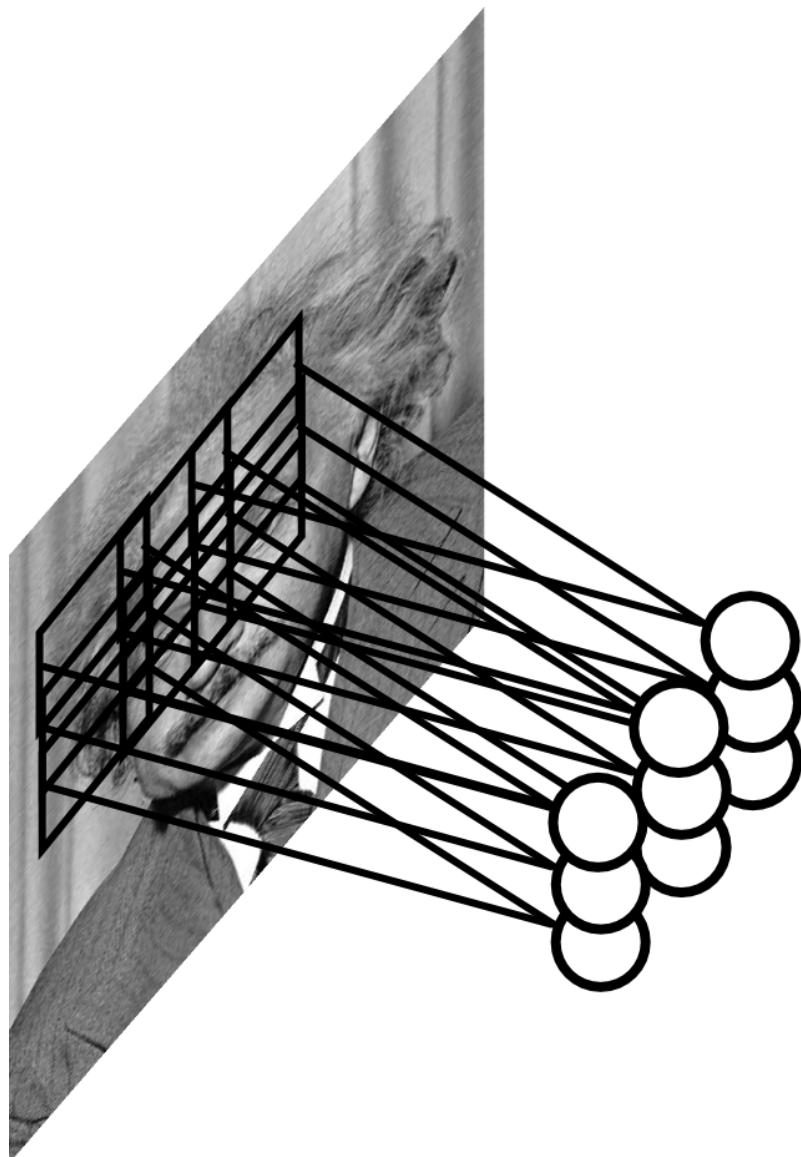
# Example: Pooling

- Illustration of pooling/subsampling operation



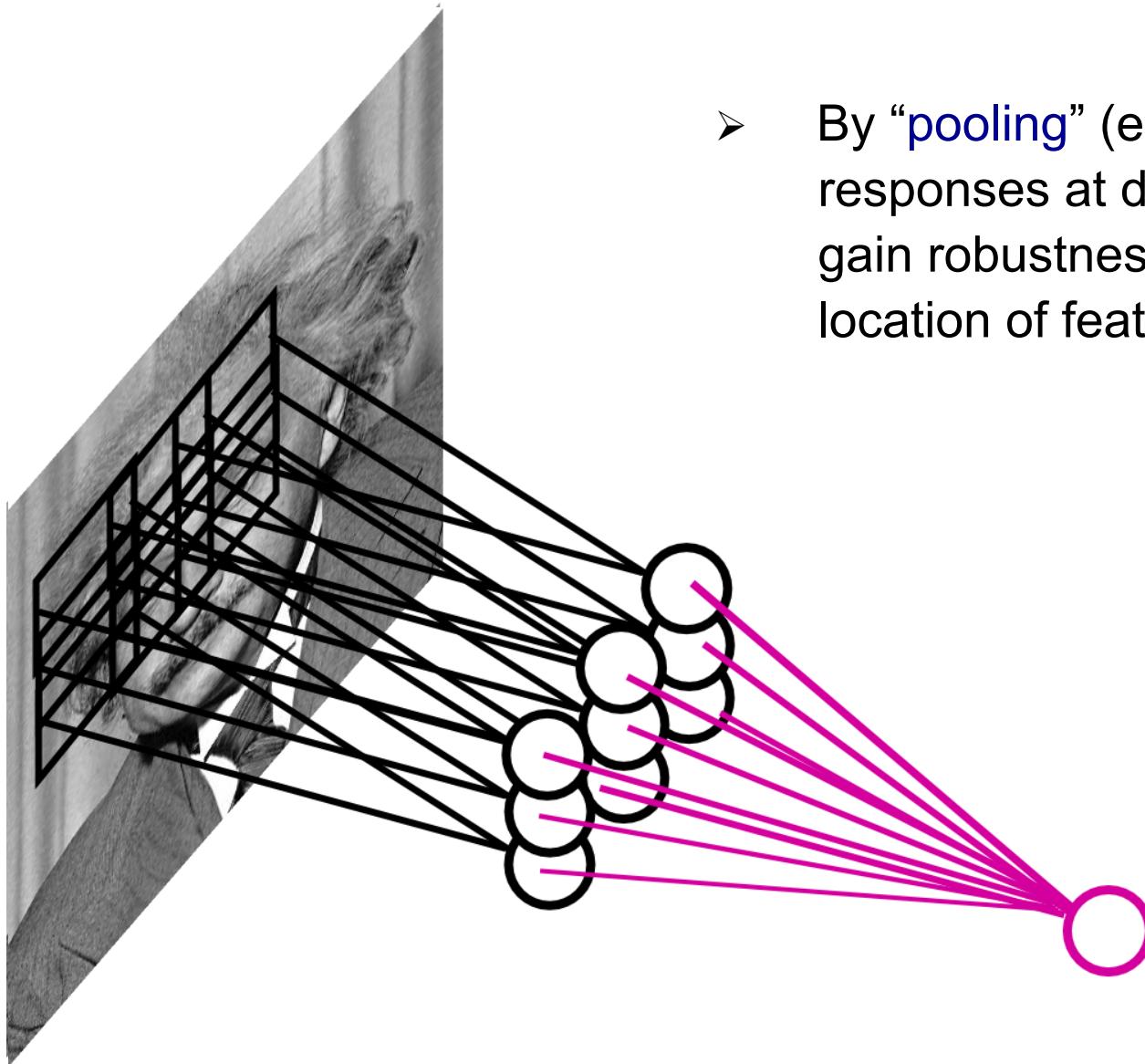
- Why pooling?
  - Introduces invariance to local translations
  - Reduces the number of hidden units in hidden layer

# Example: Pooling



- can we make the detection robust to the exact location of the eye?

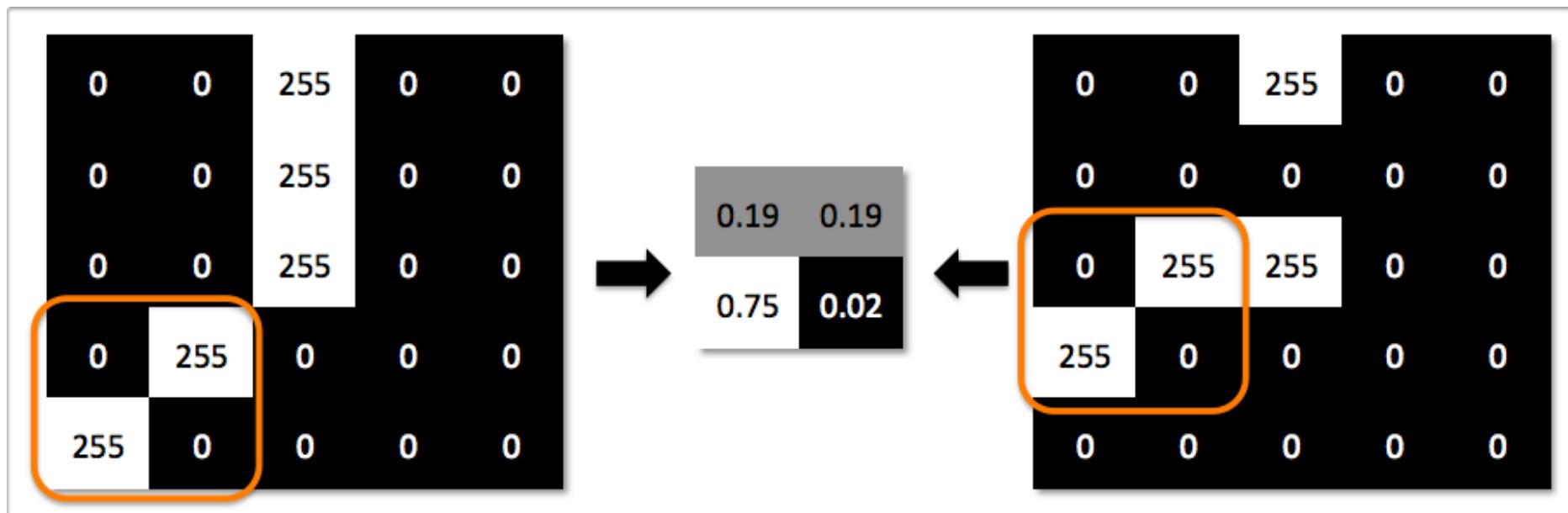
# Example: Pooling



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

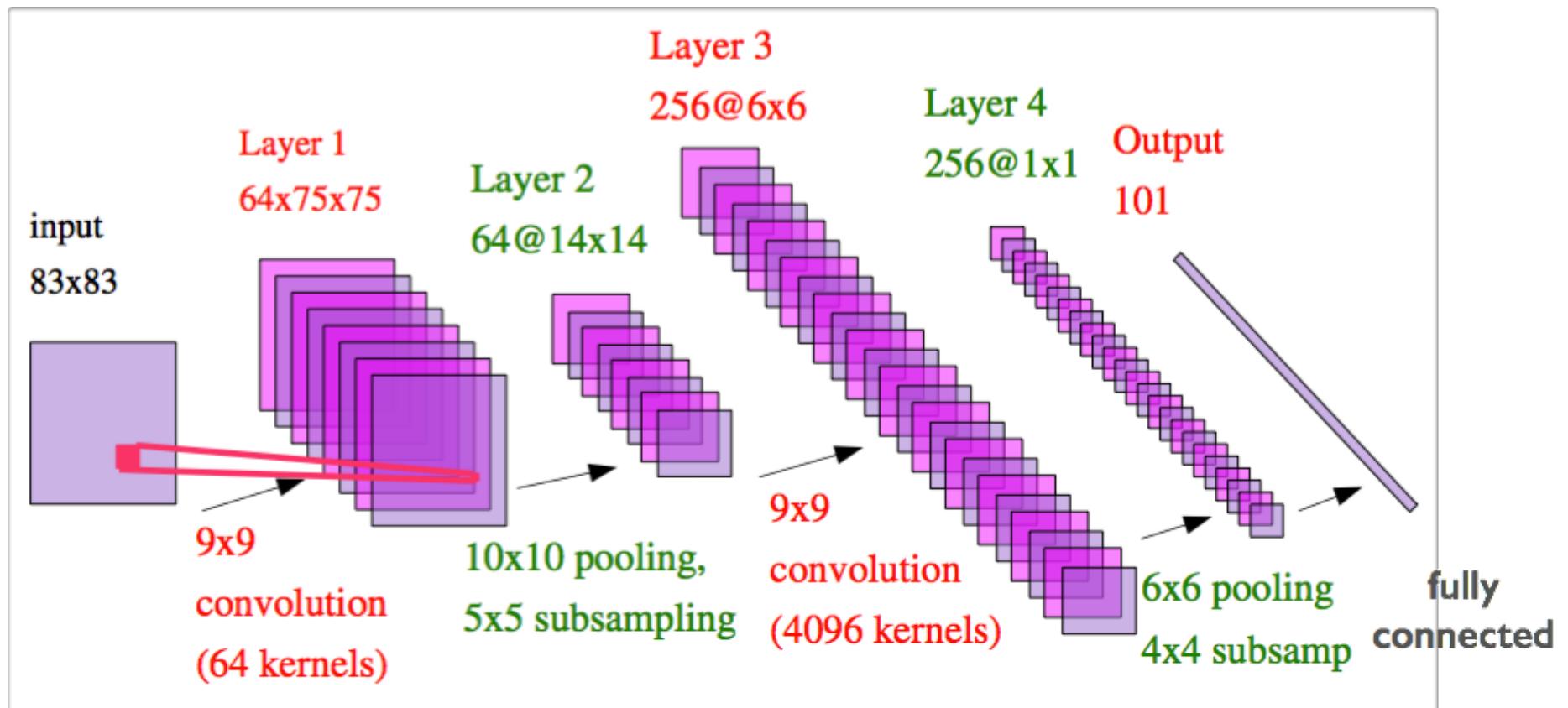
# Translation Invariance

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



# Convolutional Network

- Convolutional neural network alternates between the convolutional and pooling layers



From Yann LeCun's slides

# Convolutional Network

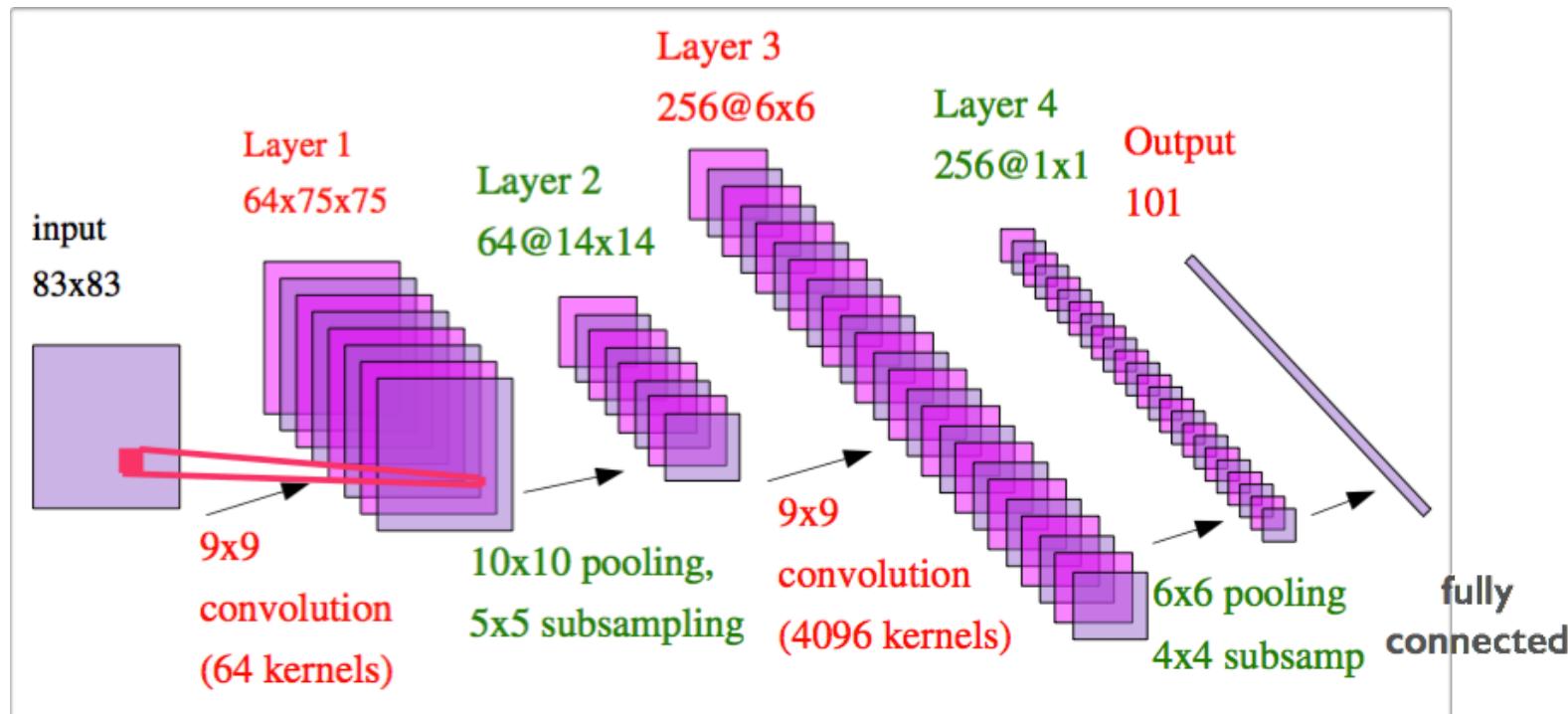
- For **classification**: Output layer is a regular, fully connected layer with softmax non-linearity
  - Output provides an estimate of the conditional probability of each class
- The network is trained by **stochastic gradient descent**
  - Backpropagation is used similarly as in a fully connected 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

# Gradient of Convolutional 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 other words, only the “**winning**” units in layer  $x$  get the gradient from the pooled layer
  - For the **average** 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

# Convolutional Network

- Convolutional neural network alternates between the convolutional and pooling layers

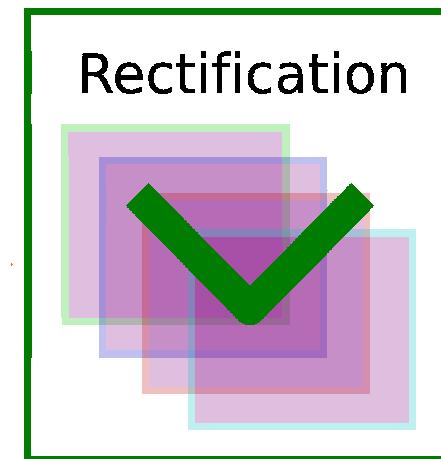


- Need to introduce **other operations** that can improve object recognition.

# Rectification

- Rectification layer:  $y_{ijk} = |x_{ijk}|$

- introduces invariance to the sign of the unit in the previous layer
- for instance, loss of information of whether an edge is black-to-white or white-to-black



# Local Contrast Normalization

- Perform local contrast normalization

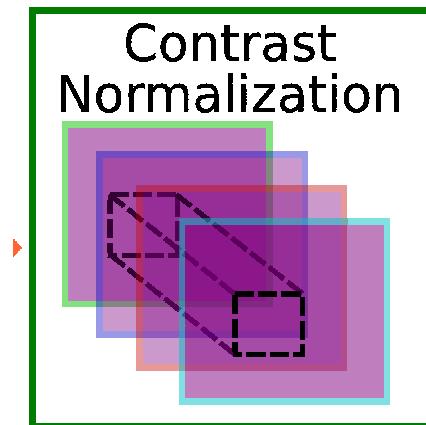
$$v_{ijk} = x_{ijk} - \left[ \sum_{ipq} w_{pq} x_{i,j+p,k+q} \right]$$

Local average

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

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

Local stdev



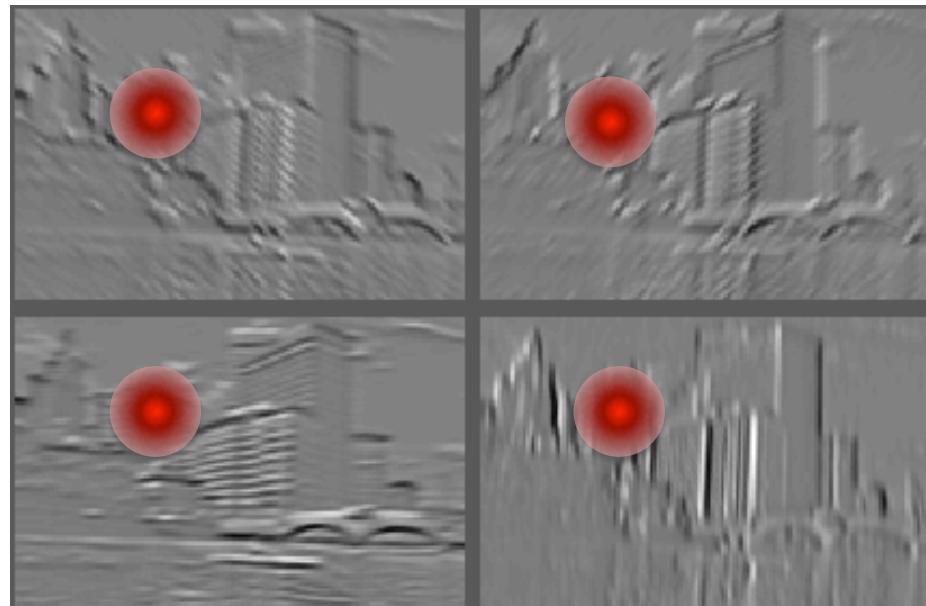
where c is a small constant to prevent division by 0

- reduces unit's activation if neighbors are also active
- creates competition between feature maps
- scales activations at each layer better for learning

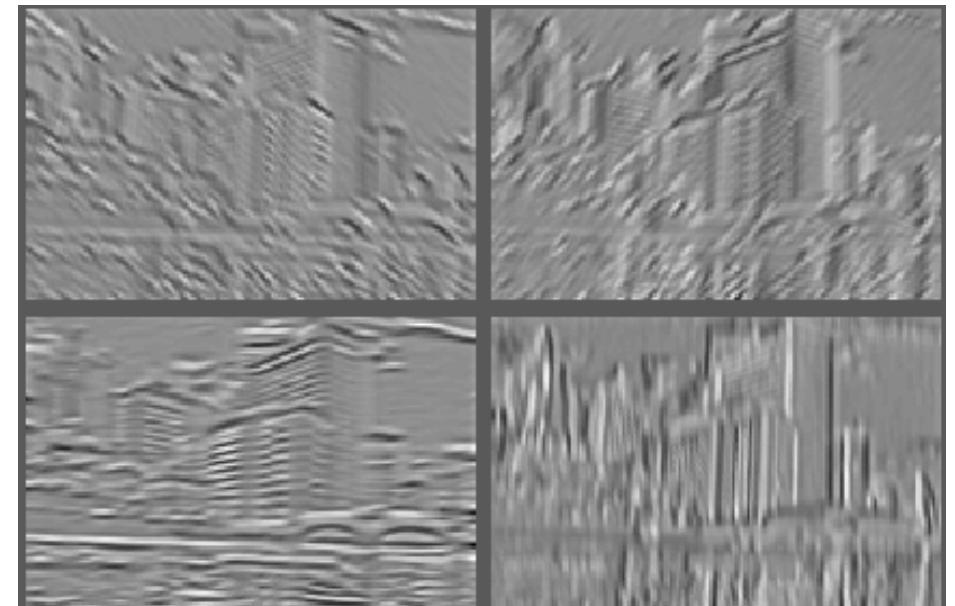
# Local Contrast Normalization

- Perform local contrast normalization
  - Local mean=0, Local std. = 1, “Local” is 7x7 Gaussian

Feature Maps

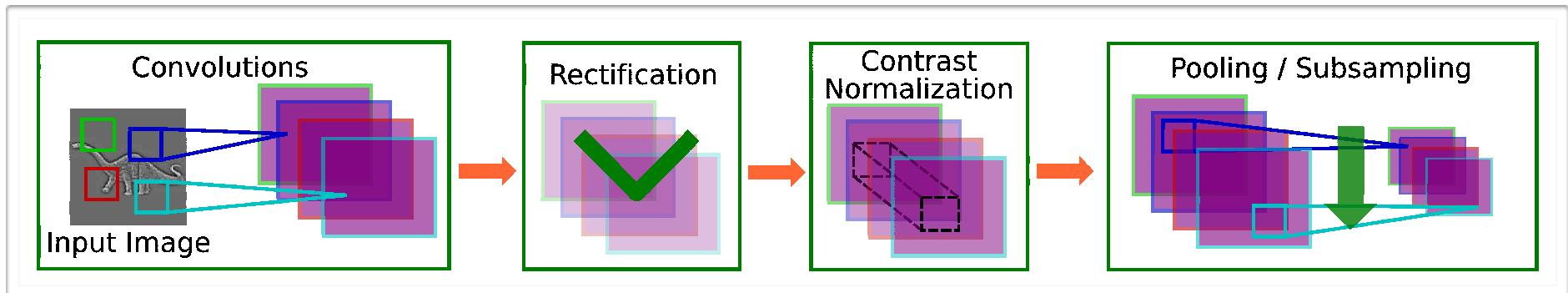


Feature Maps after  
Contrast Normalization

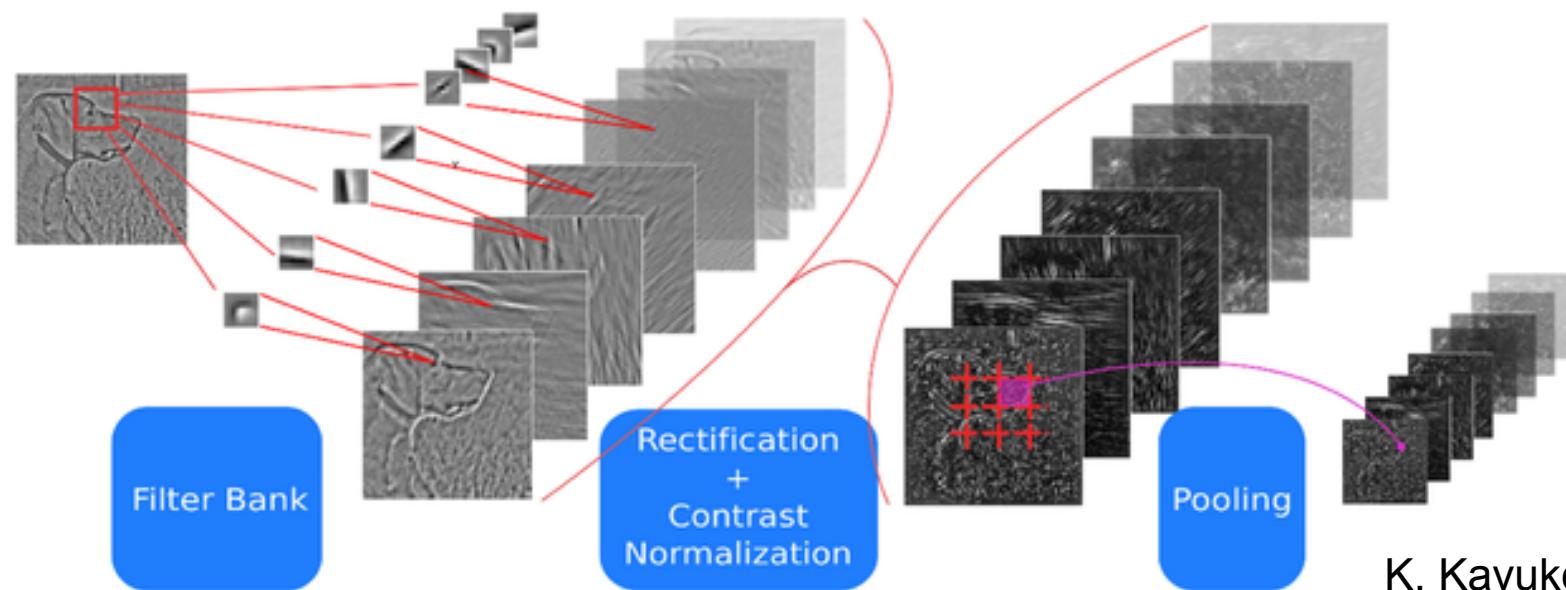


# Convolutional Network

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



Jarret et al. 2009



# Remember Batch Normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$



Learned linear transformation to adapt to non-linear activation function ( $\gamma$  and  $\beta$  are trained)