Convolutional Networks I

COMPUTER VISION

Topics: computer vision, object recognition

- GOAL: 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

112 pixels

150 pixels

"sun flower"

COMPUTER VISION

Topics: computer vision

- We can design neural networks that are specifically adapted for such problems
  - can 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

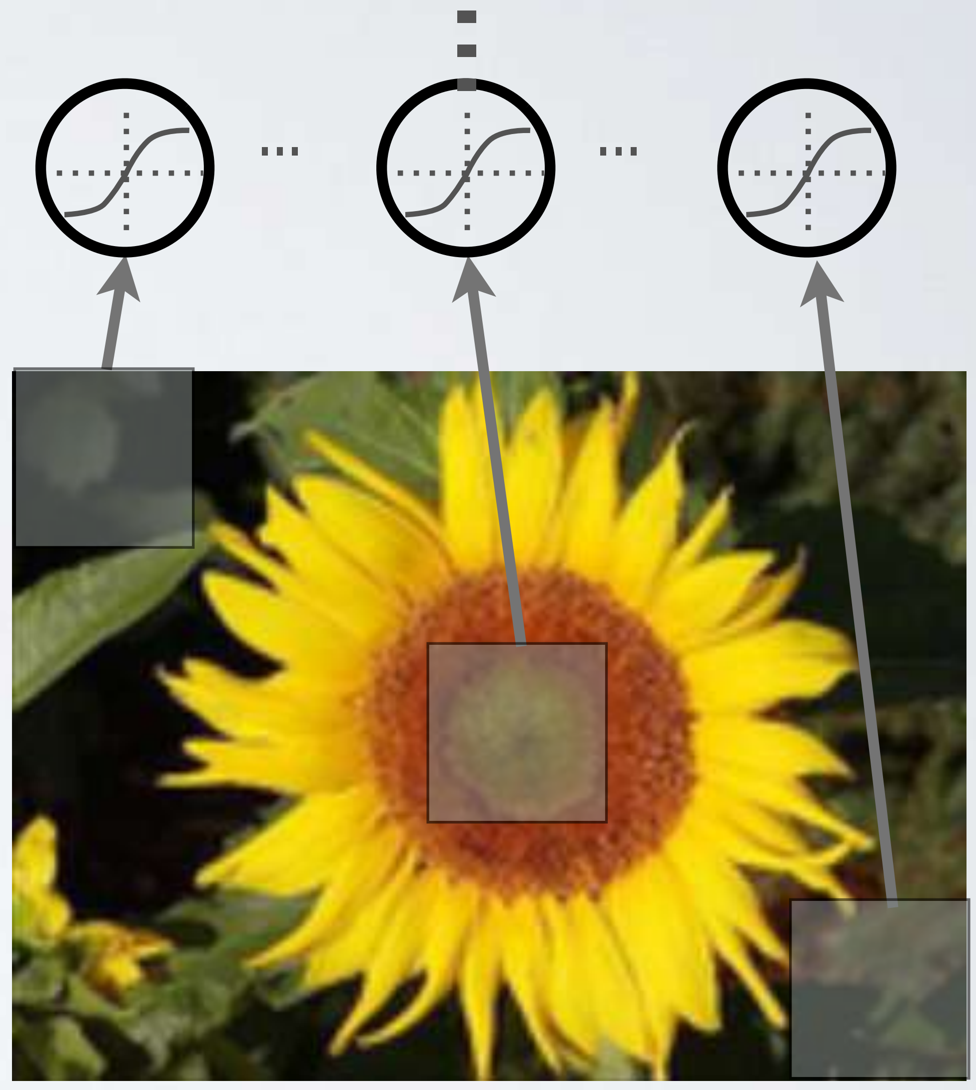
Topics: computer vision

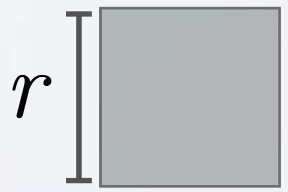
- We can design neural networks that are specifically adapted for such problems
    - can deal with very high-dimensional inputs
        - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
    - can exploit the 2D topology of pixels (or 3D for video data)
    - can build in invariance to certain 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



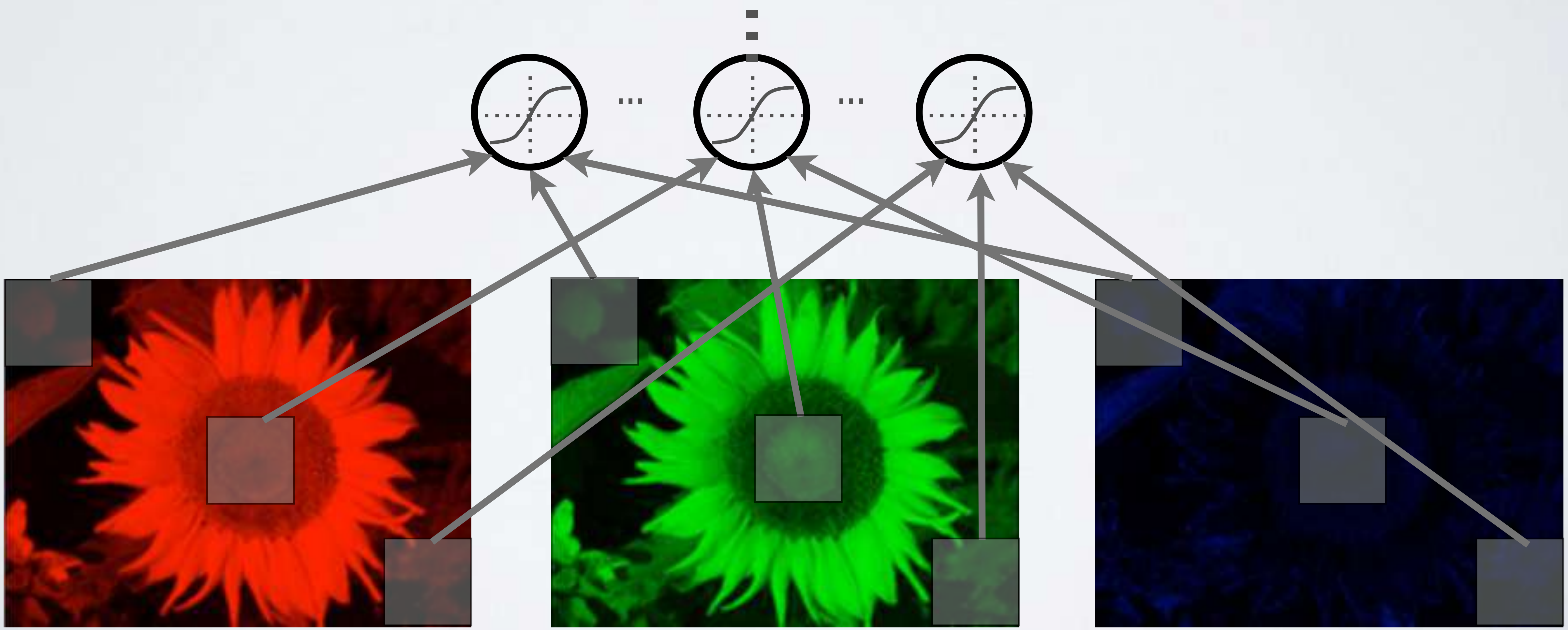


= receptive field

COMPUTER VISION

Topics: local connectivity

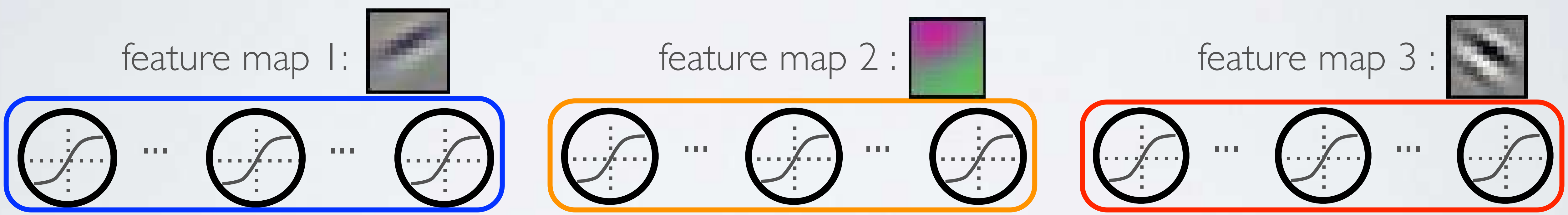
- Units are connected to all channels:
  - 1 channel if grayscale image, 3 channels (R, G, B) if color image



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



same color

=

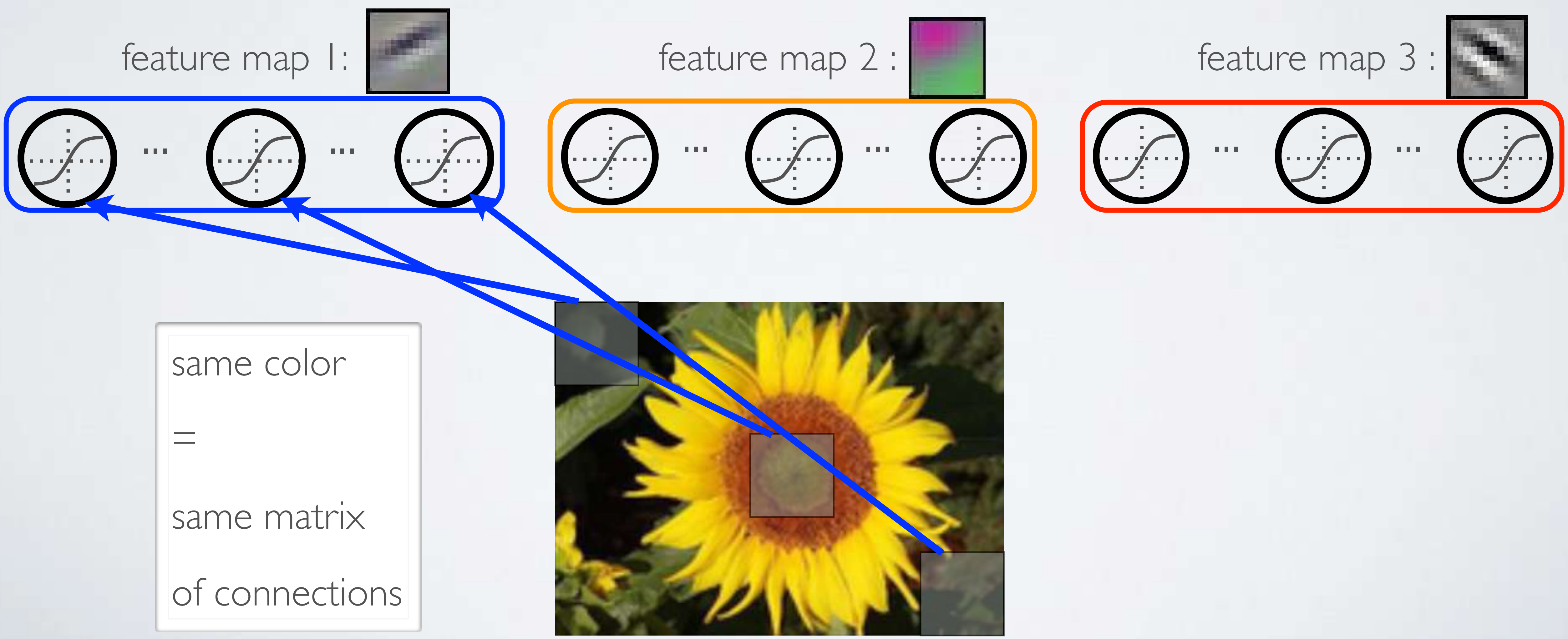
same matrix

of connections

COMPUTER VISION

Topics: parameter sharing

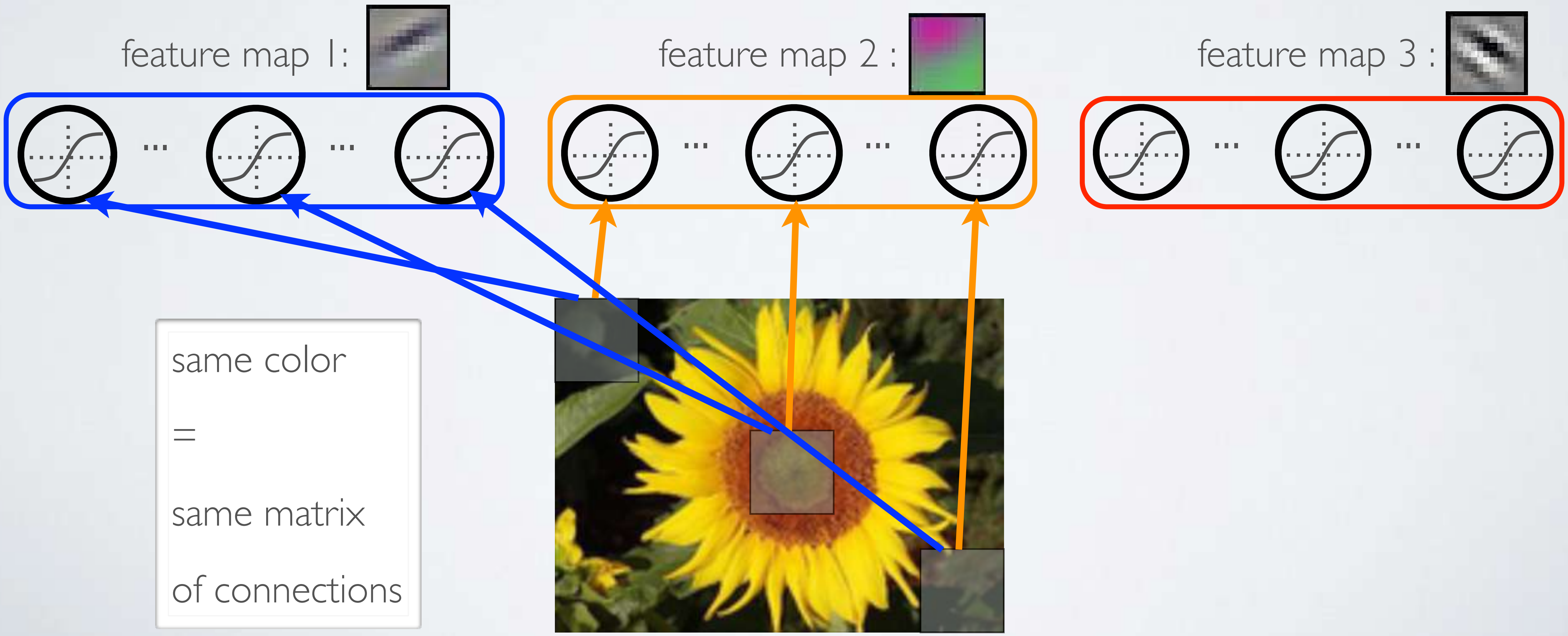
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

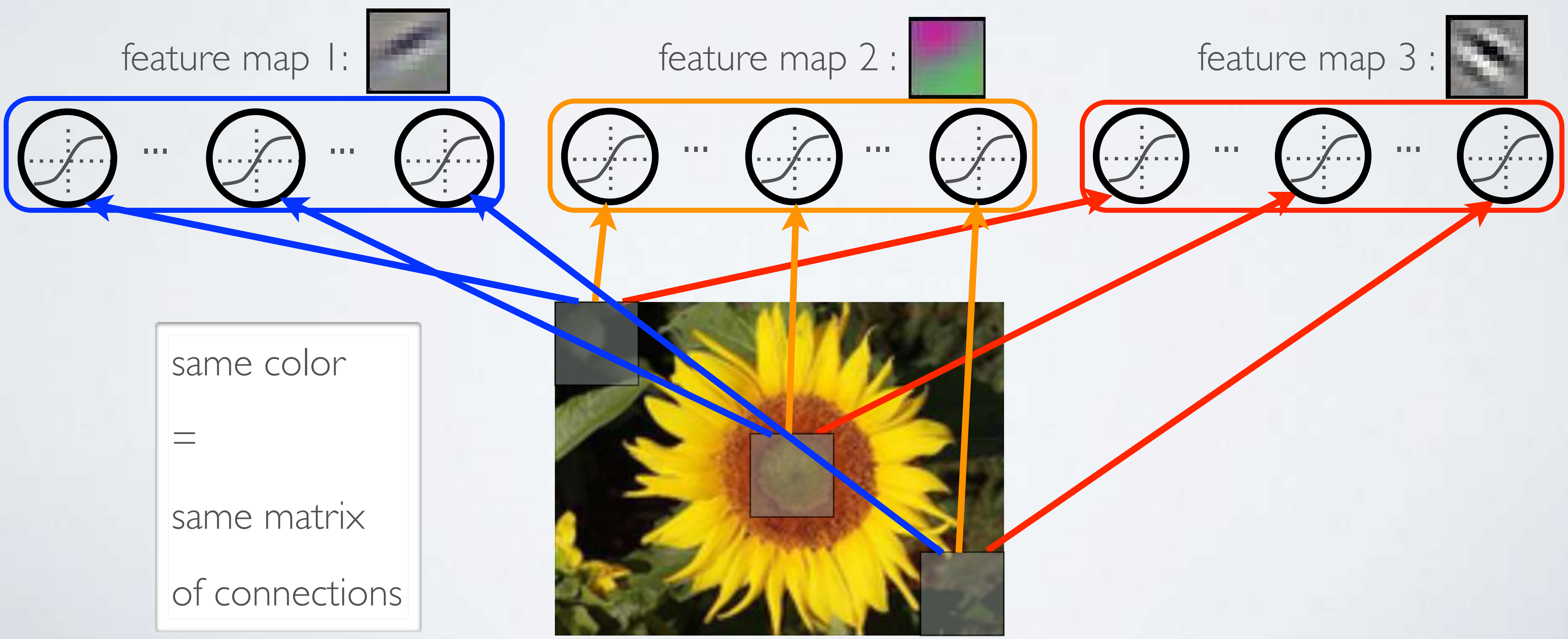
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



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



same color

=

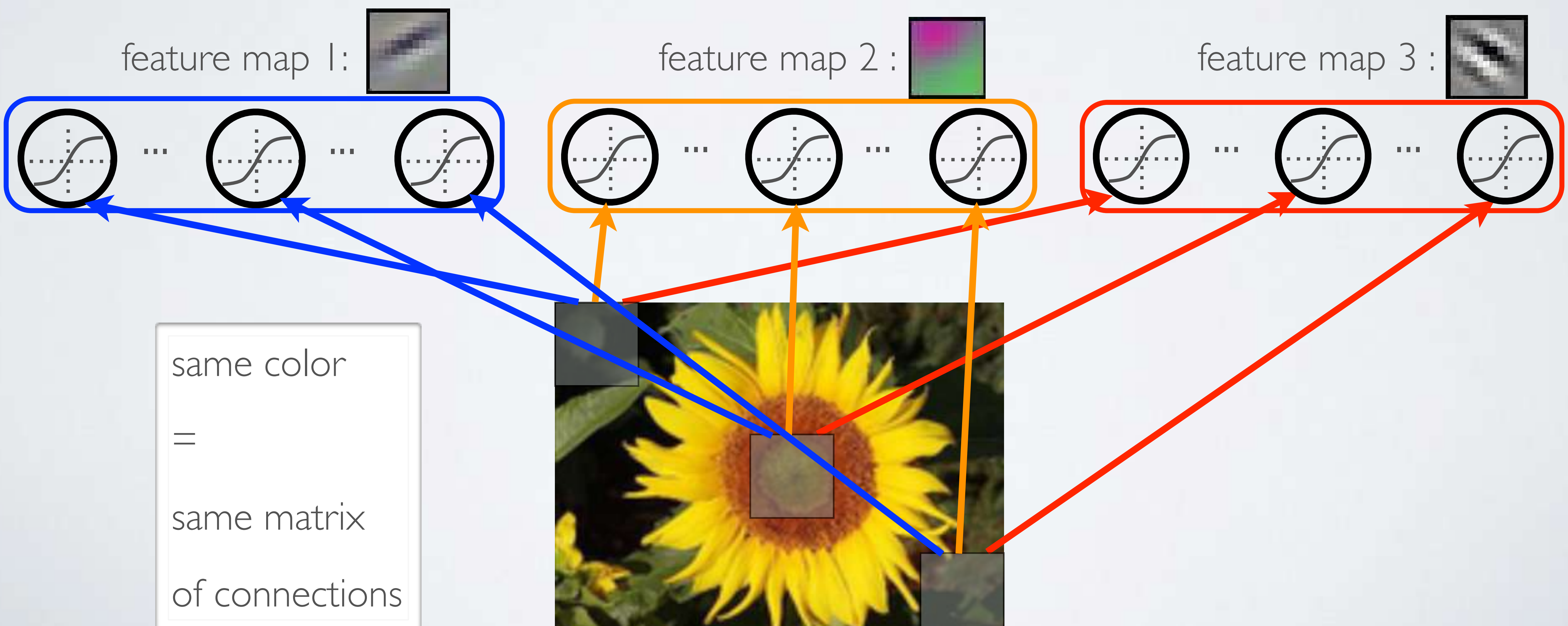
same matrix

of connections

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")

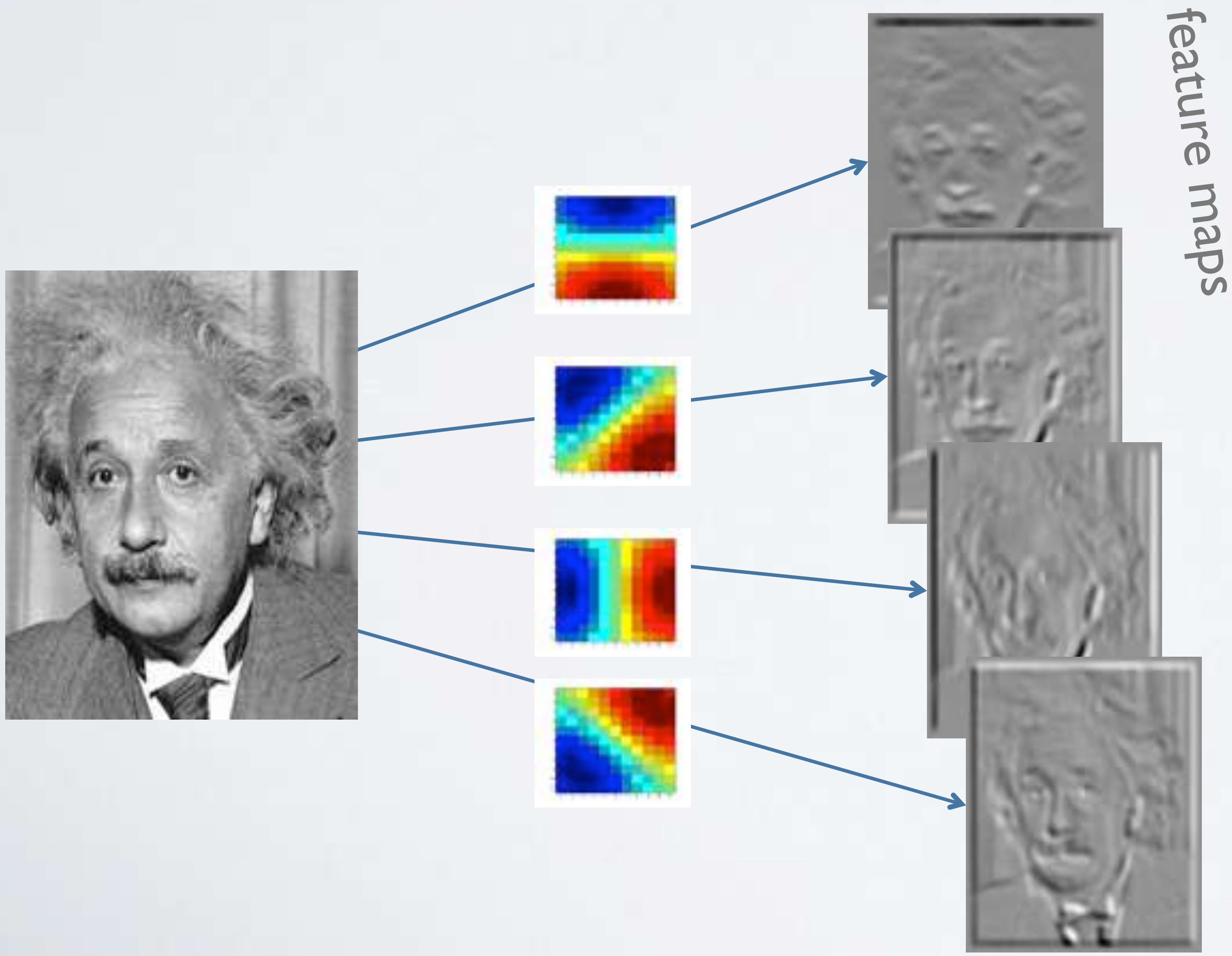


COMPUTER VISION

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

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

COMPUTER VISION

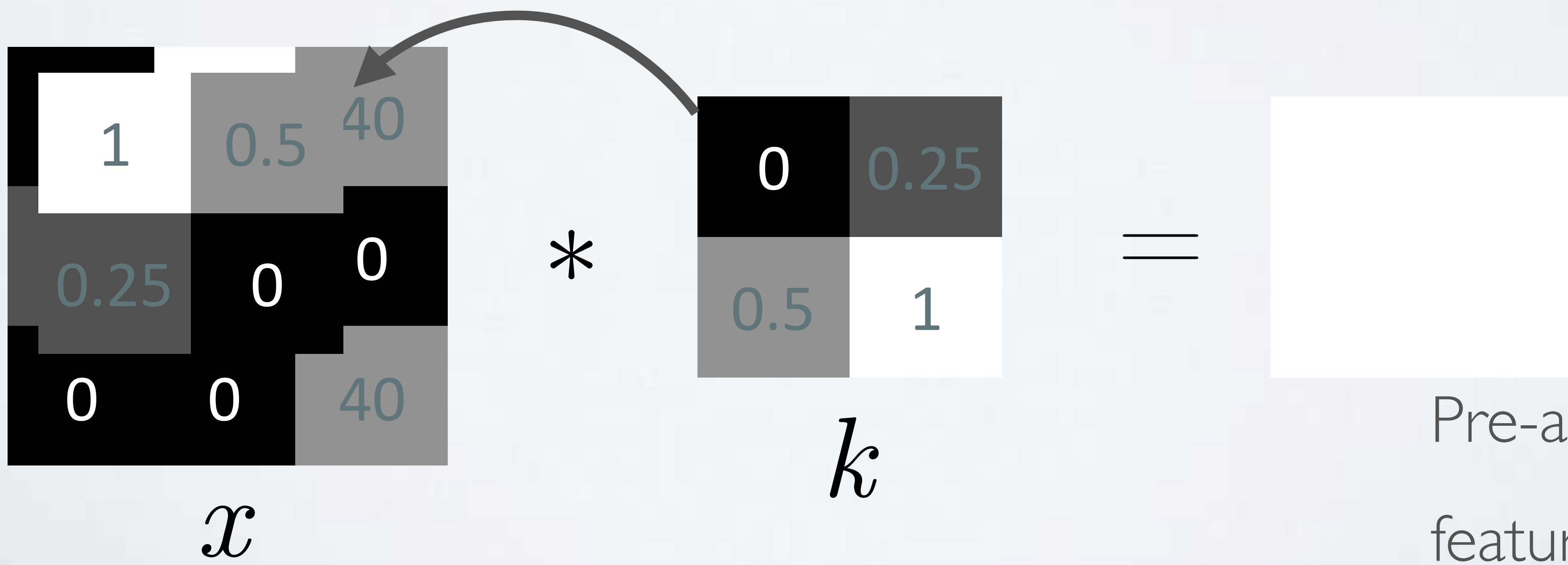
Topics: discrete convolution

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

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

- Example:

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



Pre-activation feature map

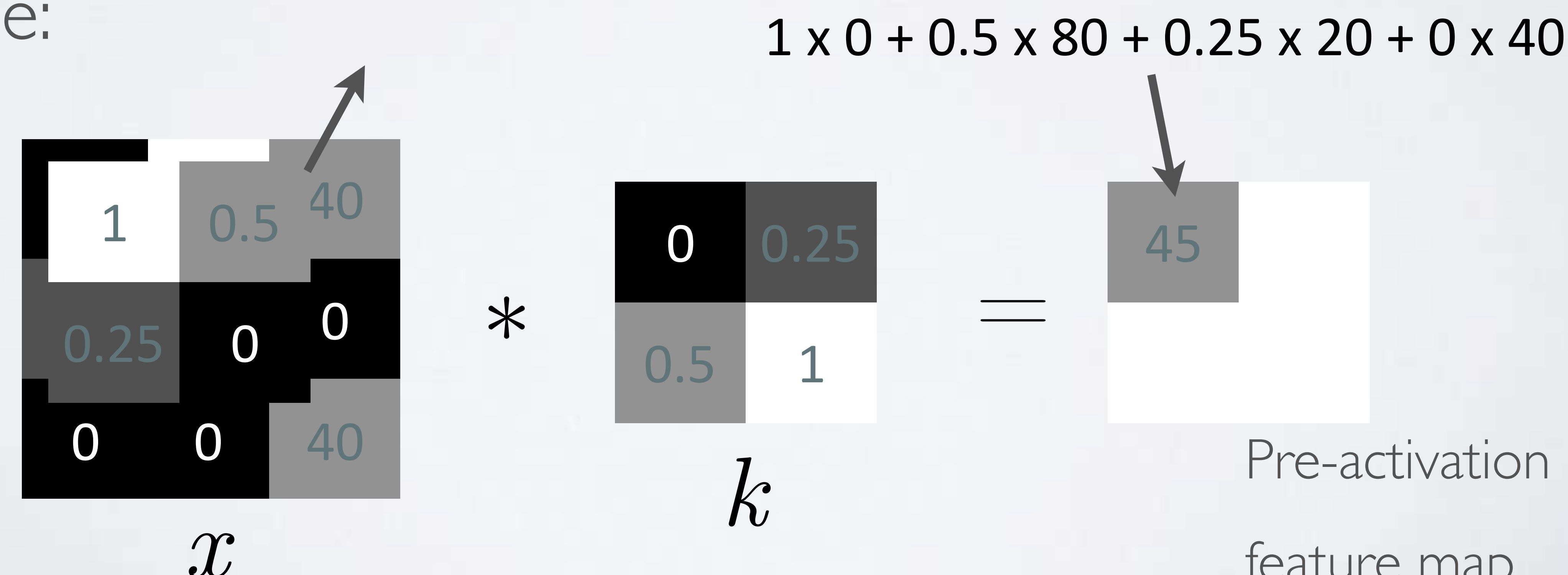
COMPUTER VISION

Topics: discrete convolution

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

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

- Example:



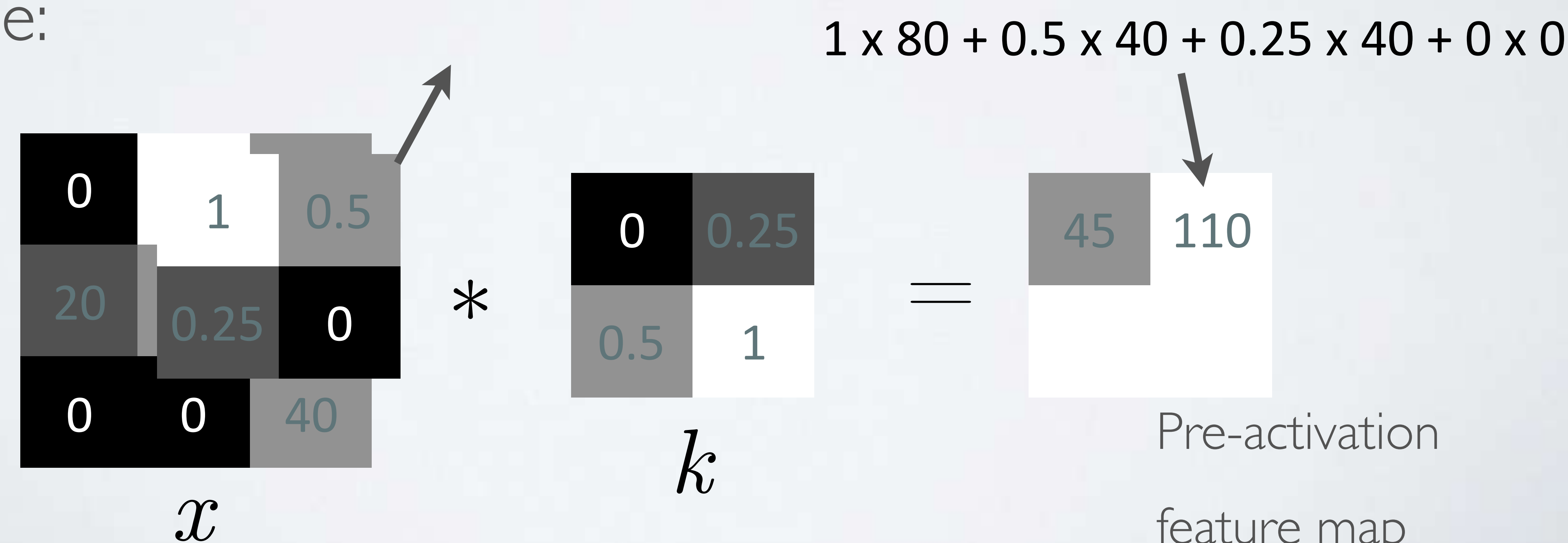
COMPUTER VISION

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:



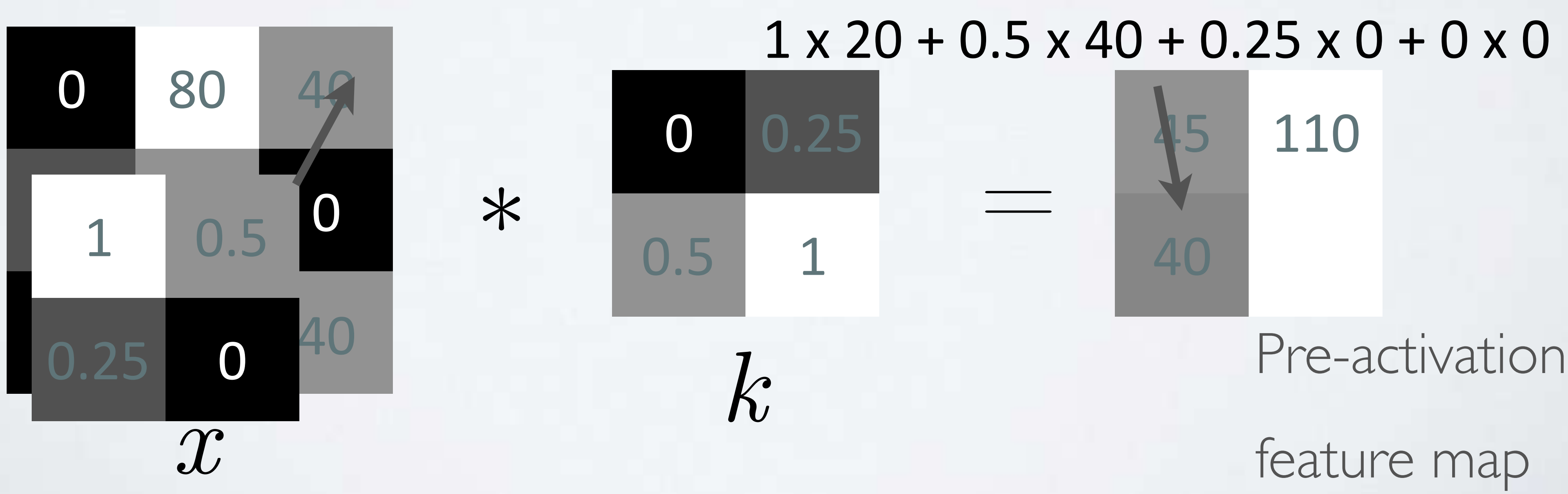
COMPUTER VISION

Topics: discrete convolution

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

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

- Example:



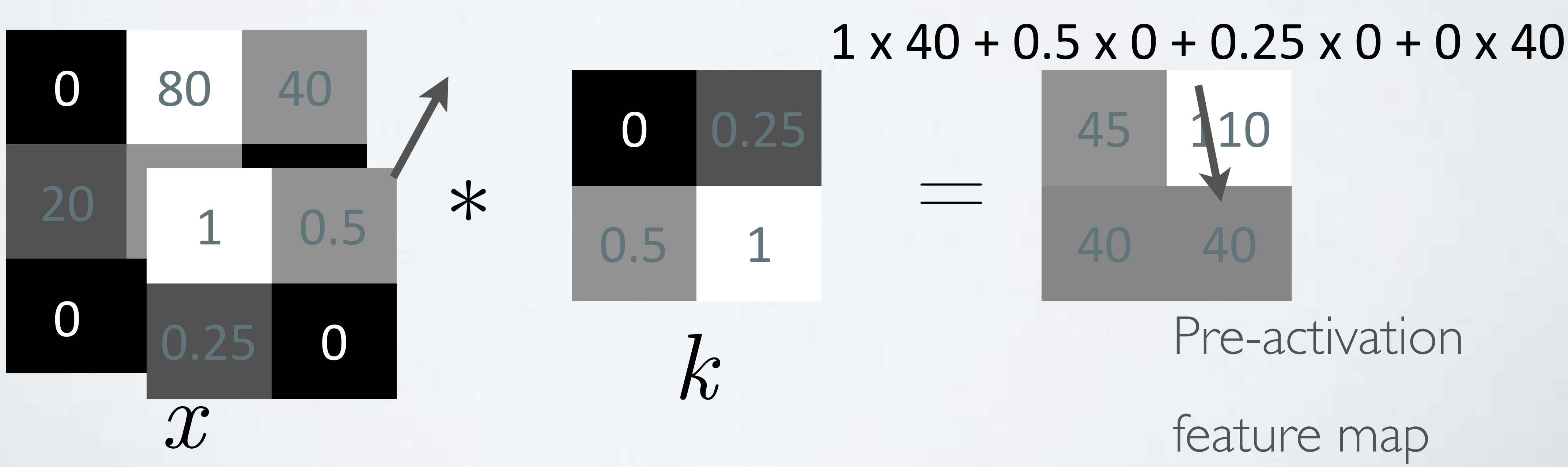
COMPUTER VISION

Topics: discrete convolution

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

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

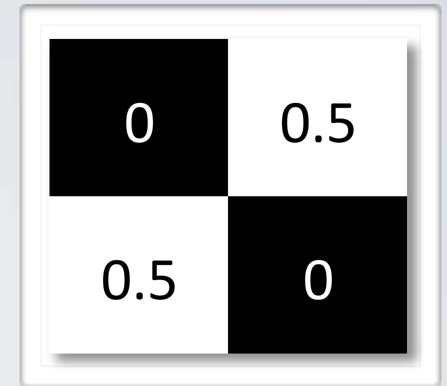
- Example:

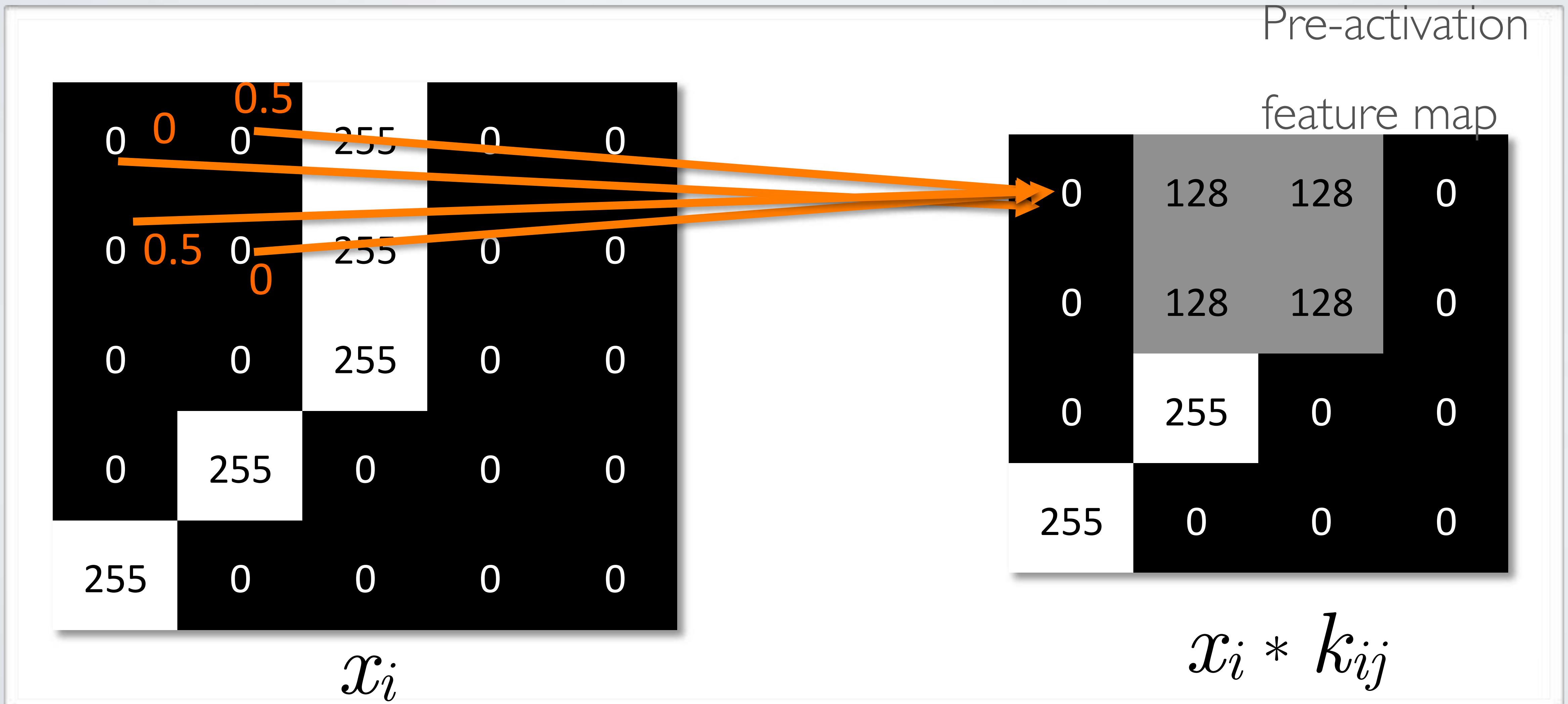


COMPUTER VISION

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

0	0.5
0.5	0

0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	255	0	0	0
255	0	0	0	0

x_i

feature map
(post-activation)

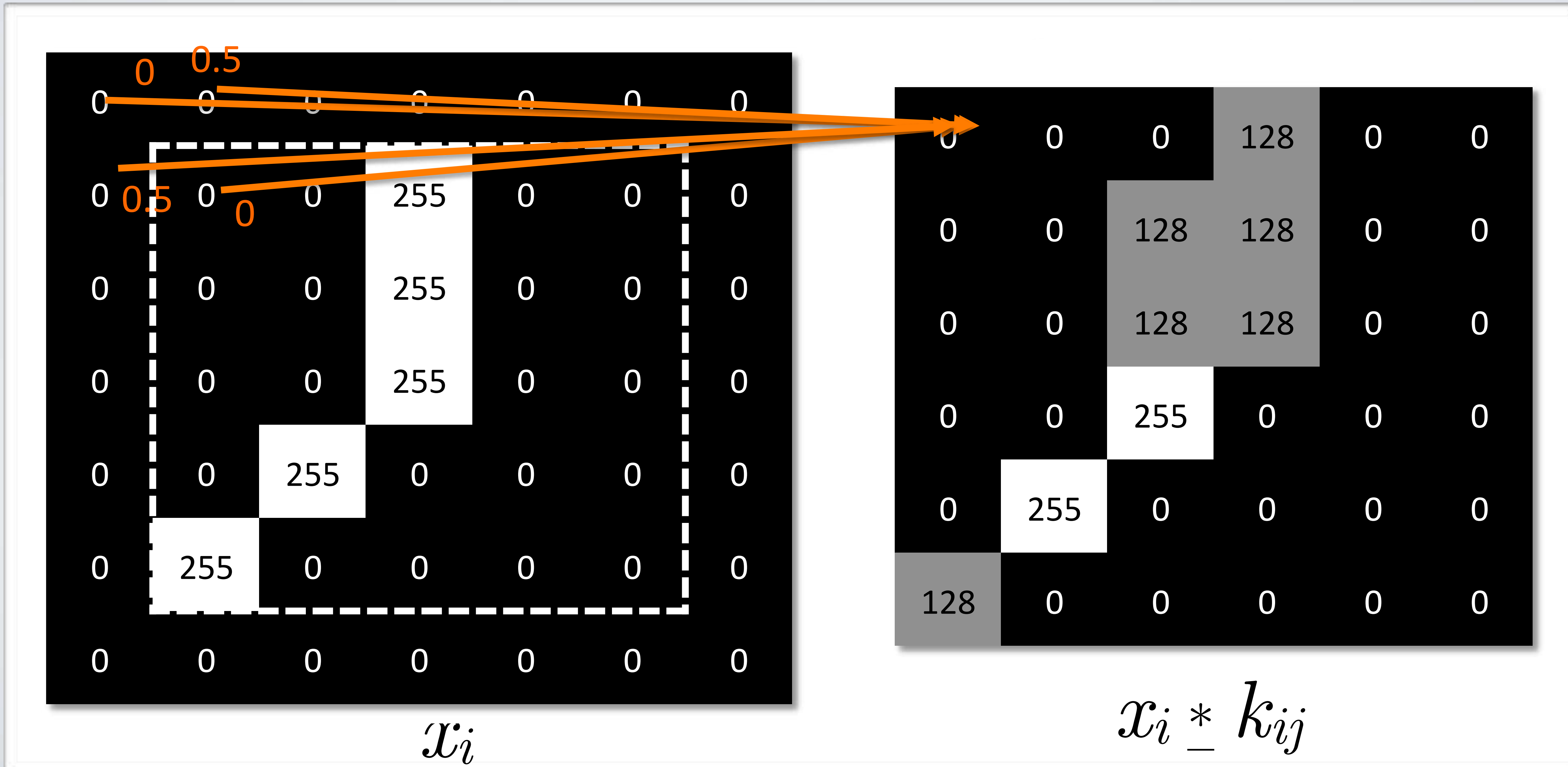
0.02	0.19	0.19	0.02
0.02	0.19	0.19	0.02
0.02	0.75	0.02	0.02
0.75	0.02	0.02	0.02

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

COMPUTER VISION

Topics: discrete convolution

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

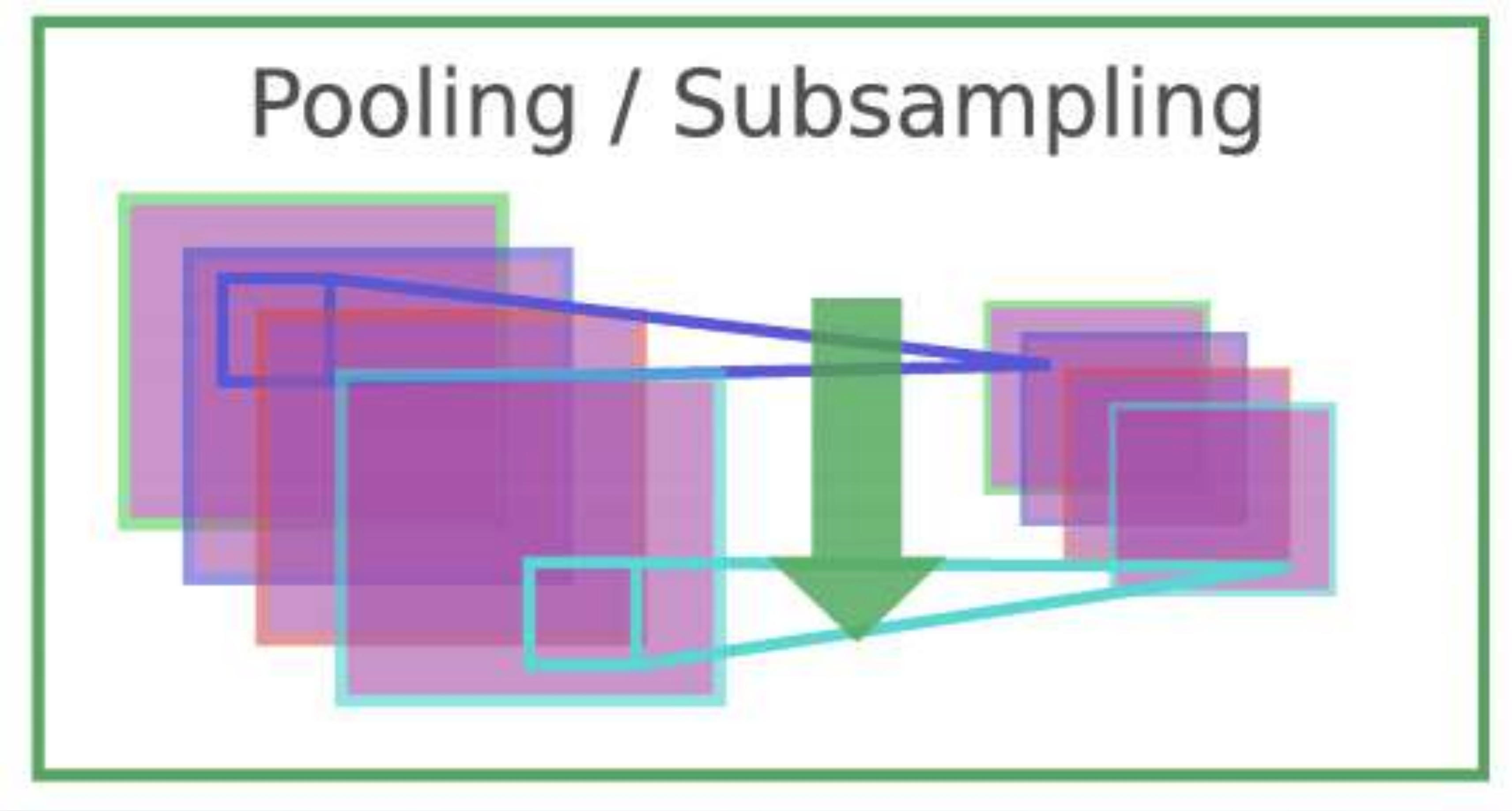


COMPUTER VISION

Topics: pooling and subsampling

Jarret et al. 2009

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



- $x_{i,j,k}$ 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

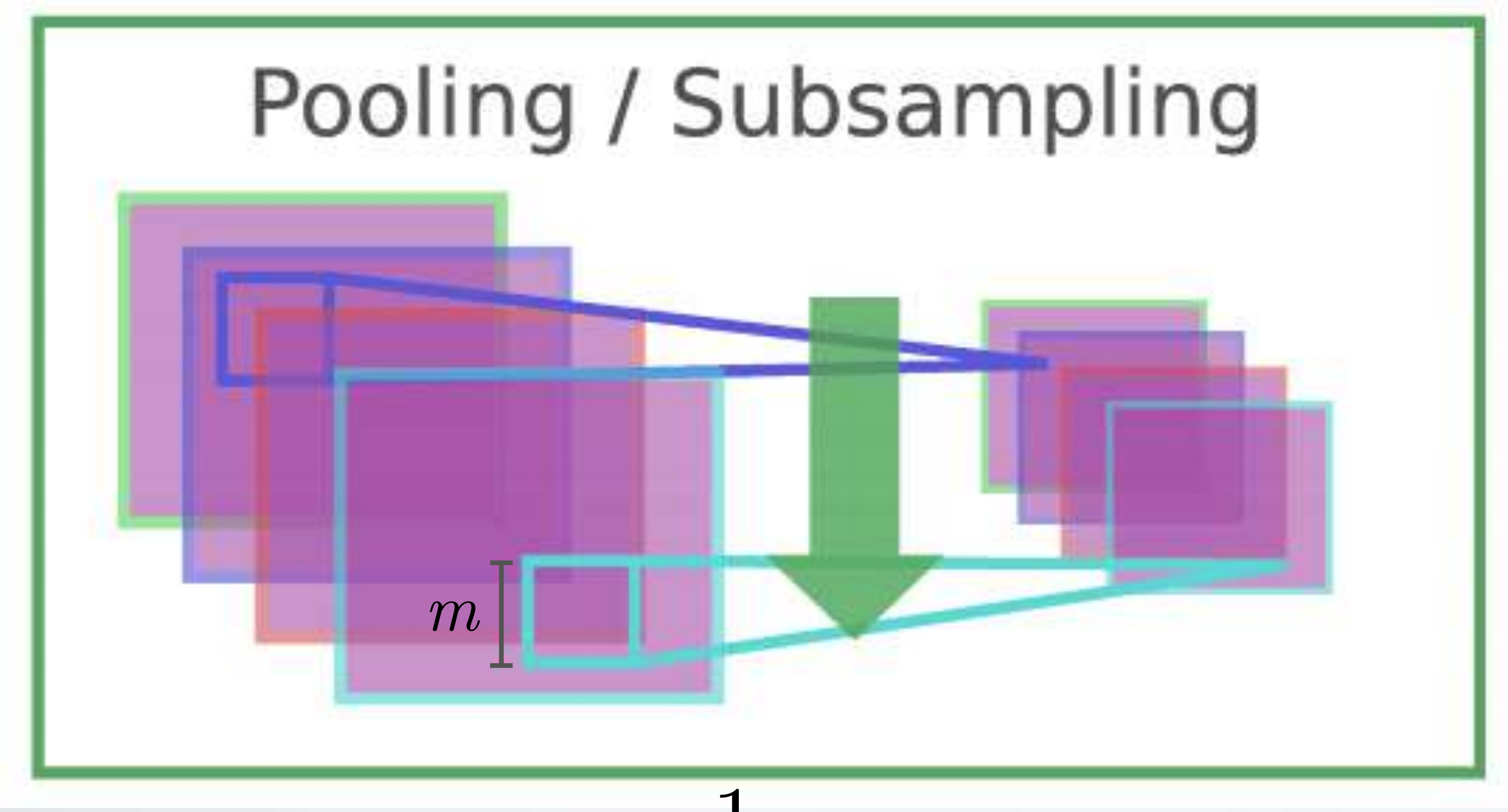
$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$$

COMPUTER VISION

Topics: pooling and subsampling

Jarret et al. 2009

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



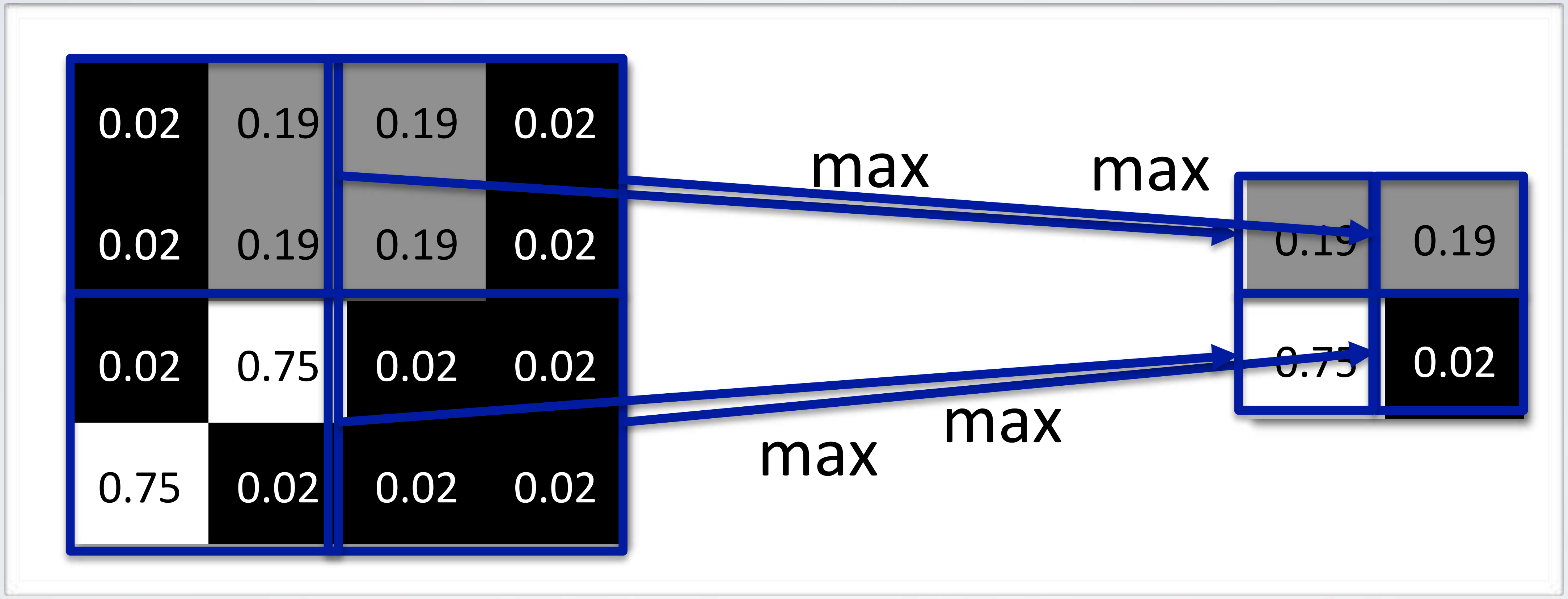
$$y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$$

- $x_{i,j,k}$ 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

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

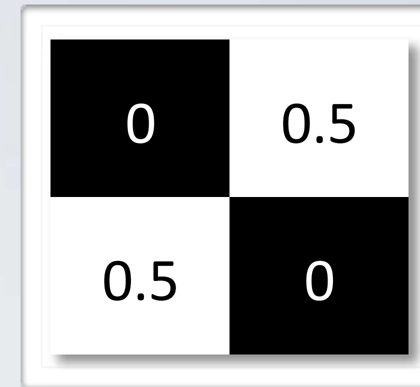


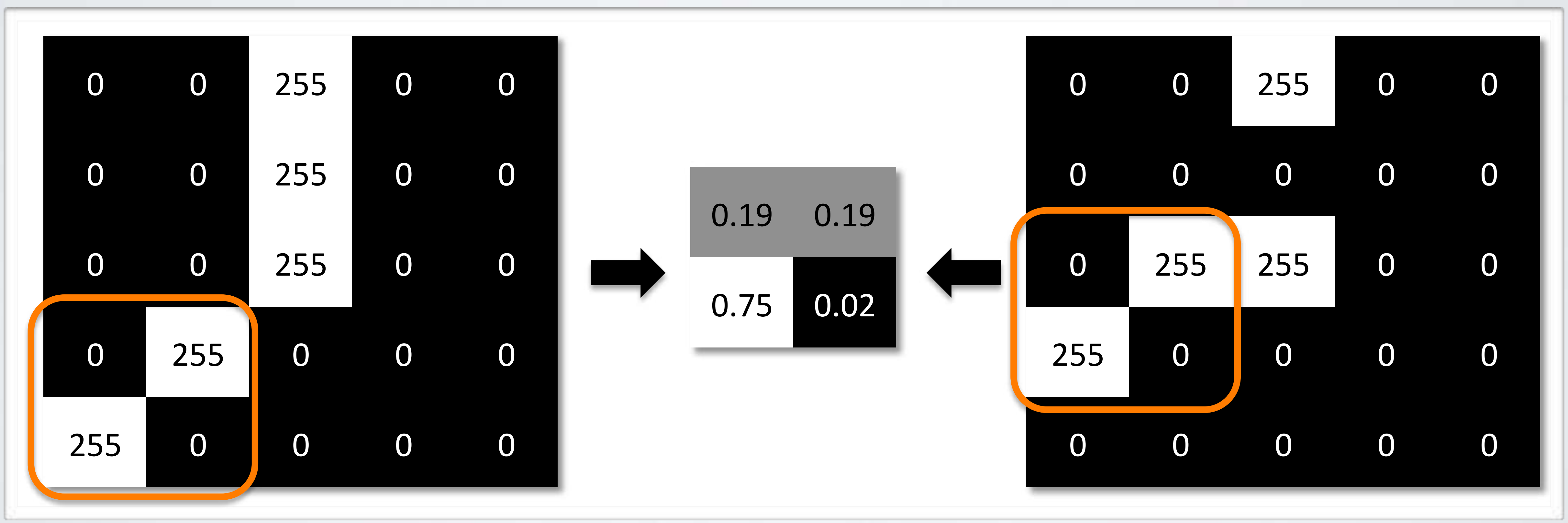
- Solves the following problems:
  - introduces invariance to local translations
  - reduces the number of hidden units in hidden layer

COMPUTER VISION

Topics: pooling and subsampling

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



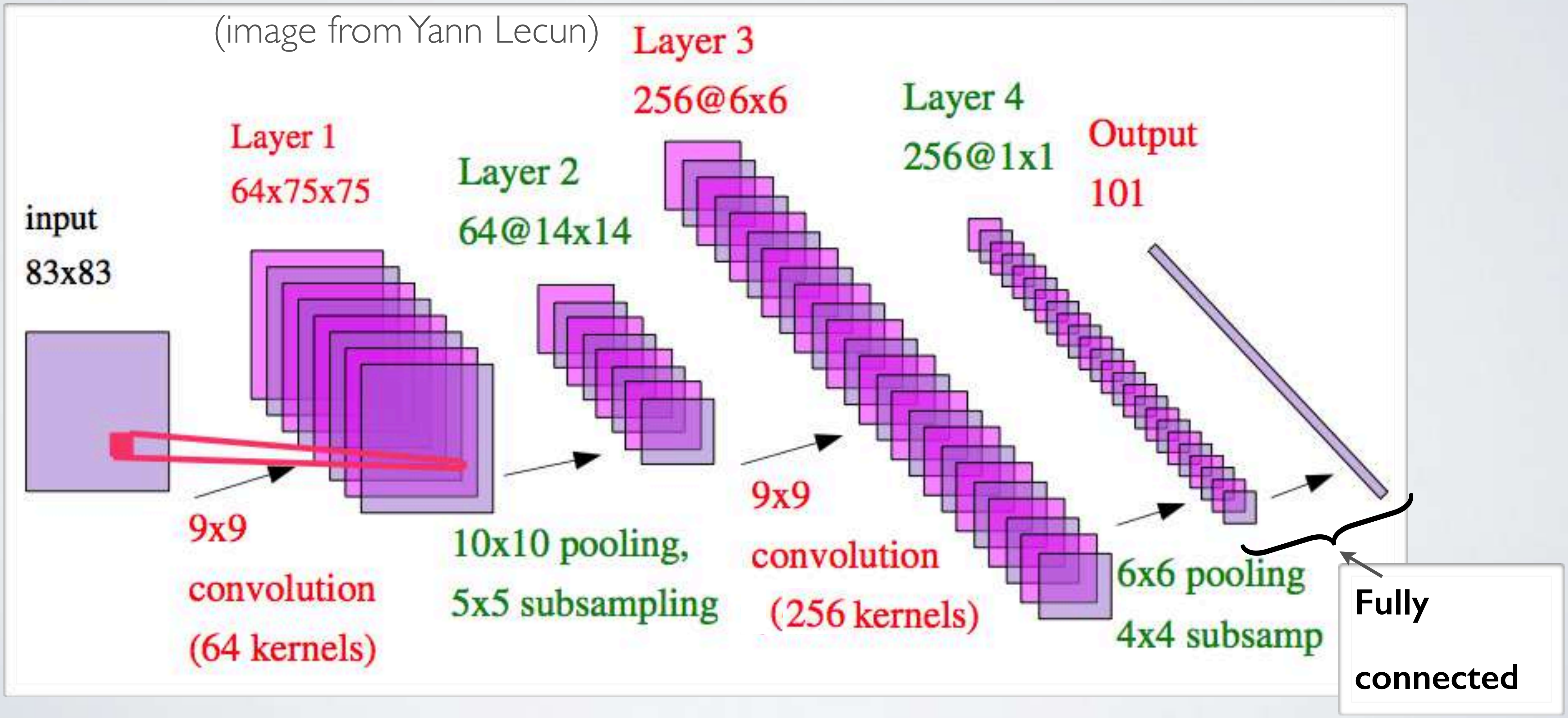


CONVOLUTIONAL NETWORK

Topics: convolutional network

- Convolutional neural network alternates between the convolutional and pooling layers

(image from Yann Lecun)



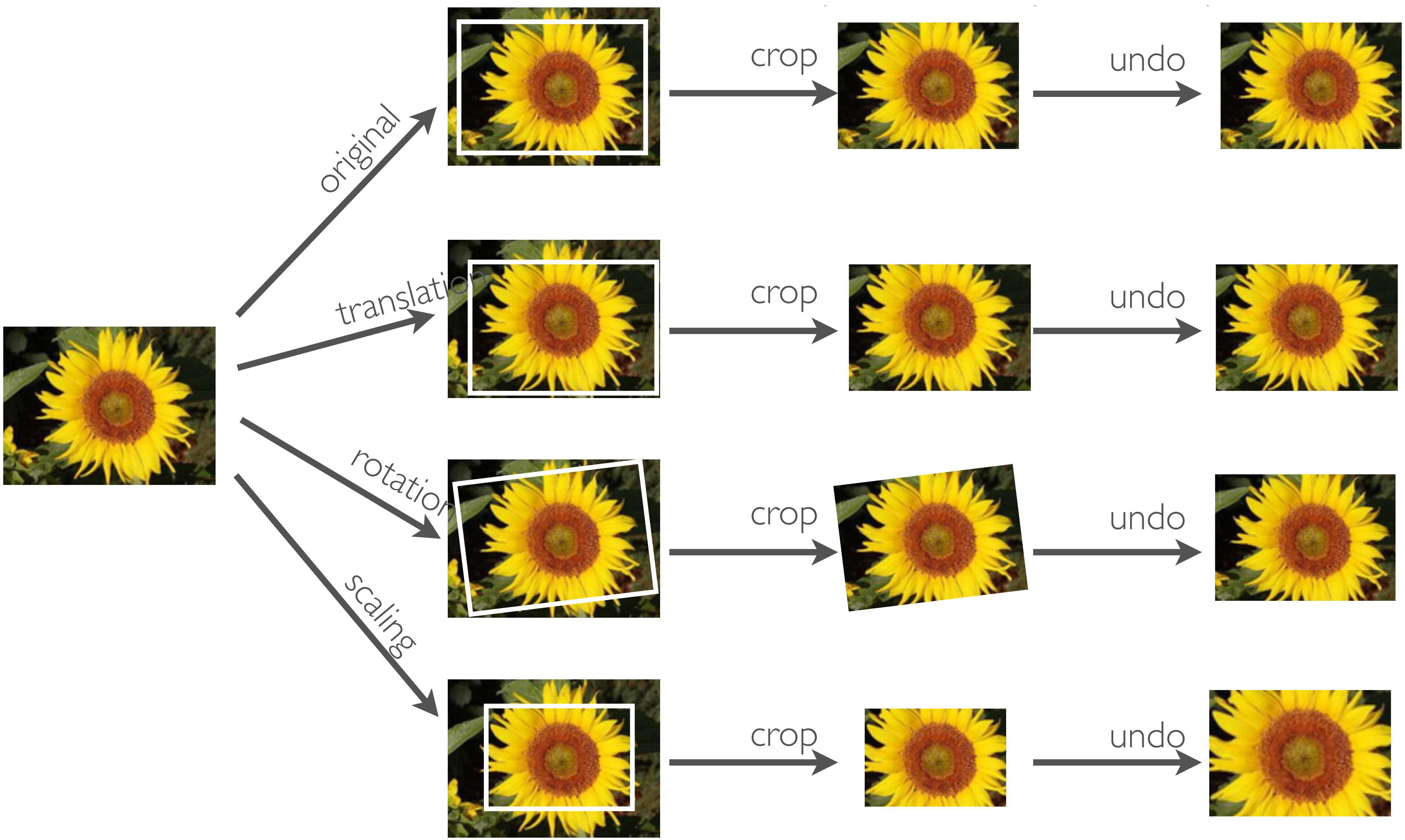
INVARIANCE BY DATA SET EXPANSION

Topics: generating additional examples

- Invariances built-in in convolutional network:
    - small translations: due to convolution and max pooling
- 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

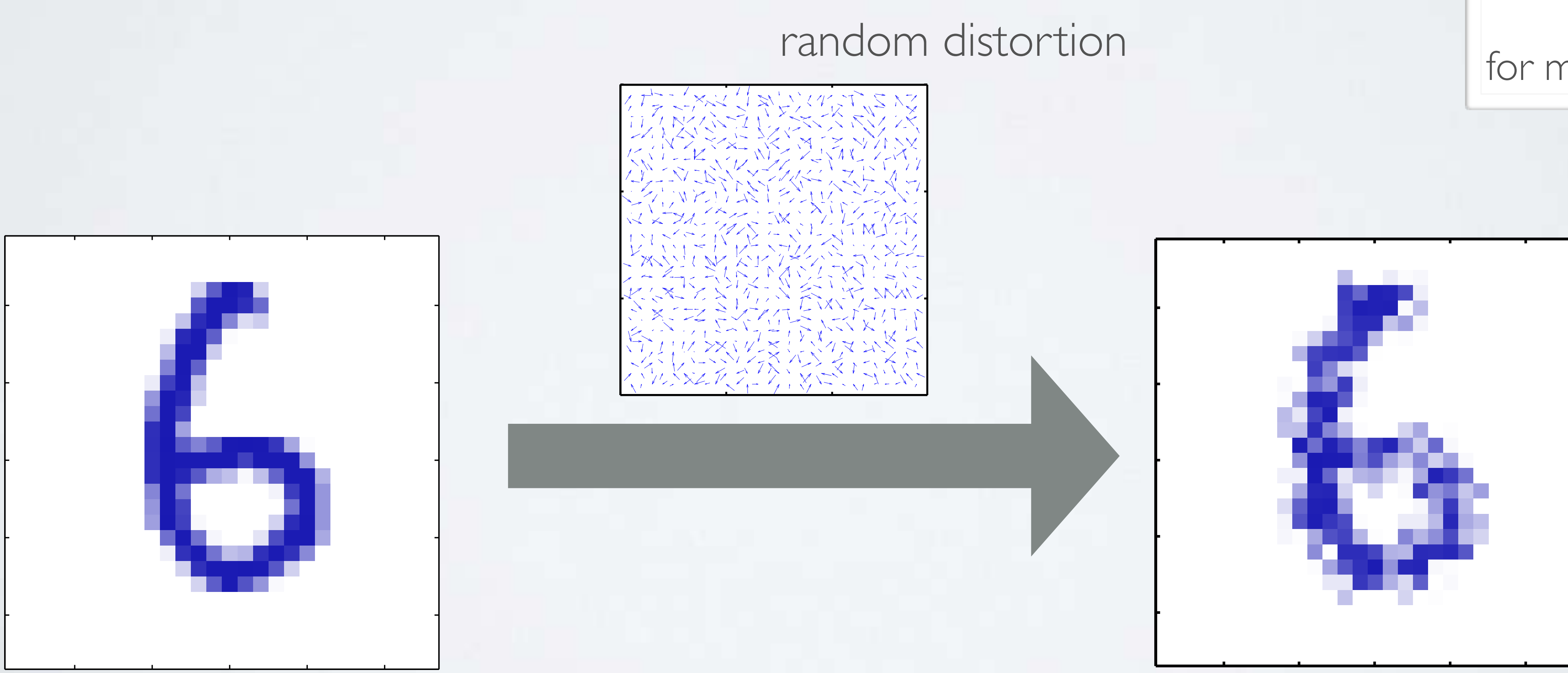


INVARIANCE BY DATA SET EXPANSION

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)

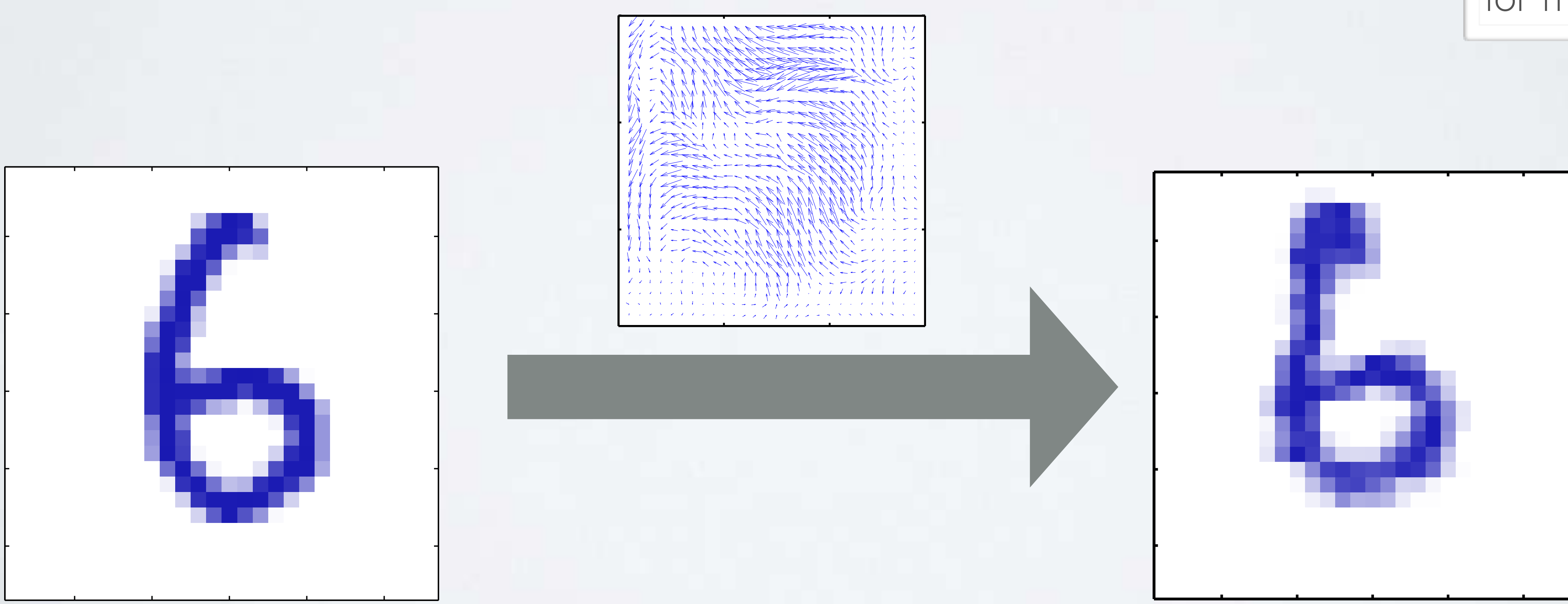
INVARIANCE BY DATA SET EXPANSION

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

See Simard et al. for more detail

smoothed
random distortion



(from Bishop's book)

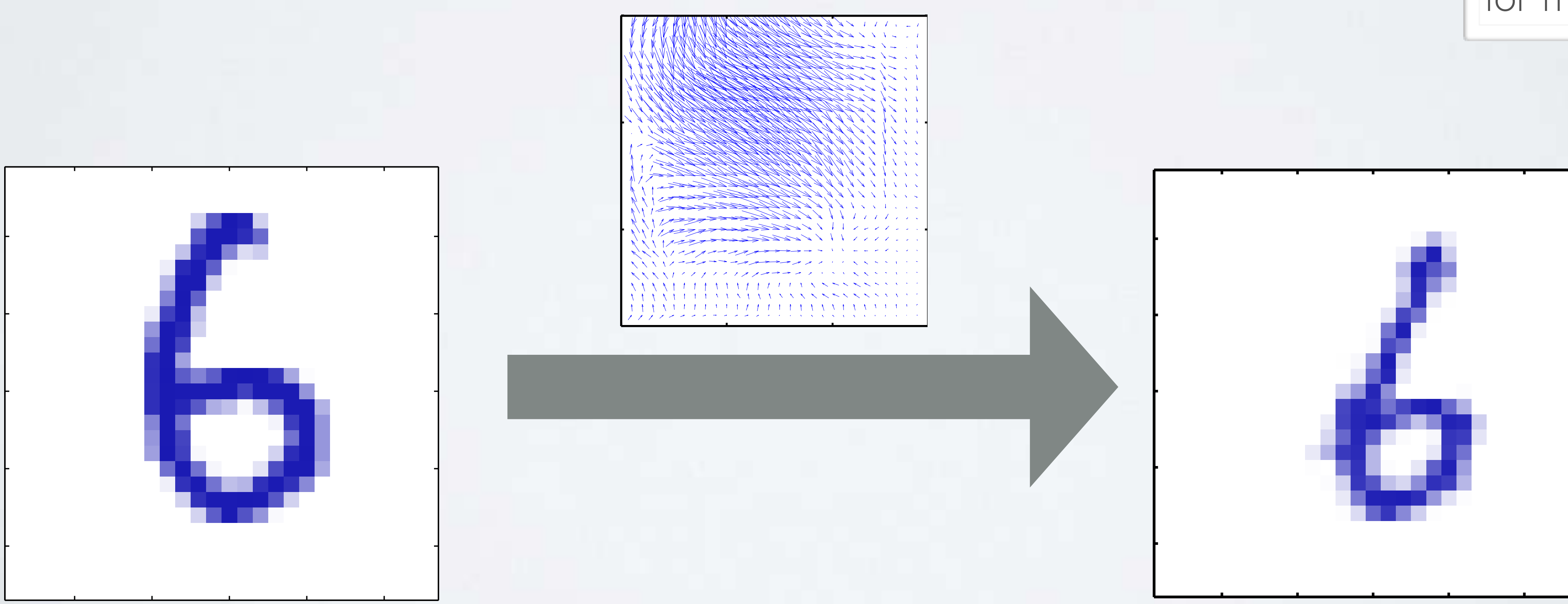
INVARIANCE BY DATA SET EXPANSION

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

See Simard et al. for more detail

smoothed
random distortion



(from Bishop’s book)

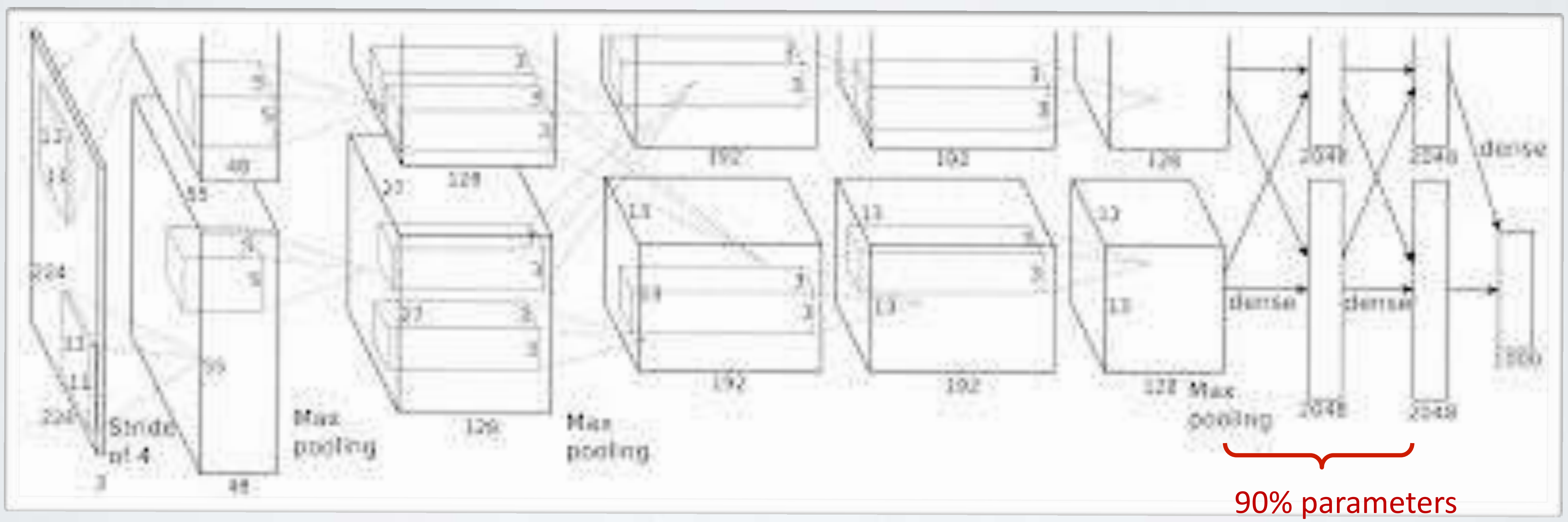
CONVOLUTIONAL NETWORK

Topics: convolutional network

- The network is trained by stochastic gradient descent
  - backpropagation is used similarly as in a fully connected network
    - ➡ need to pass gradients through the convolution operation and the pooling operation.
- Rectified linear activation functions and variants such as maxout (Goodfellow et al., 2013) are popular choices.
  - promote deeper models by allowing gradients to flow better.
- **Network-in-network** (Lin et al.; ICLR2014):
  - instead of convolving with a generalized linear unit (linear filter + nonlinear activation function), convolve a small network that includes internal hidden units.
  - Used in GoogLenet, the current state-of-the-art in the ImageNet challenge.

CONVNET IN ACTION

- SuperVision (a.k.a. AlexNet) CNN by the numbers
  - Trained on 1.2 million images, roughly 1K images for each of the 1K classes.
  - Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
  - 650,000 neurons, 60,000,000 parameters, 630,000,000 connections



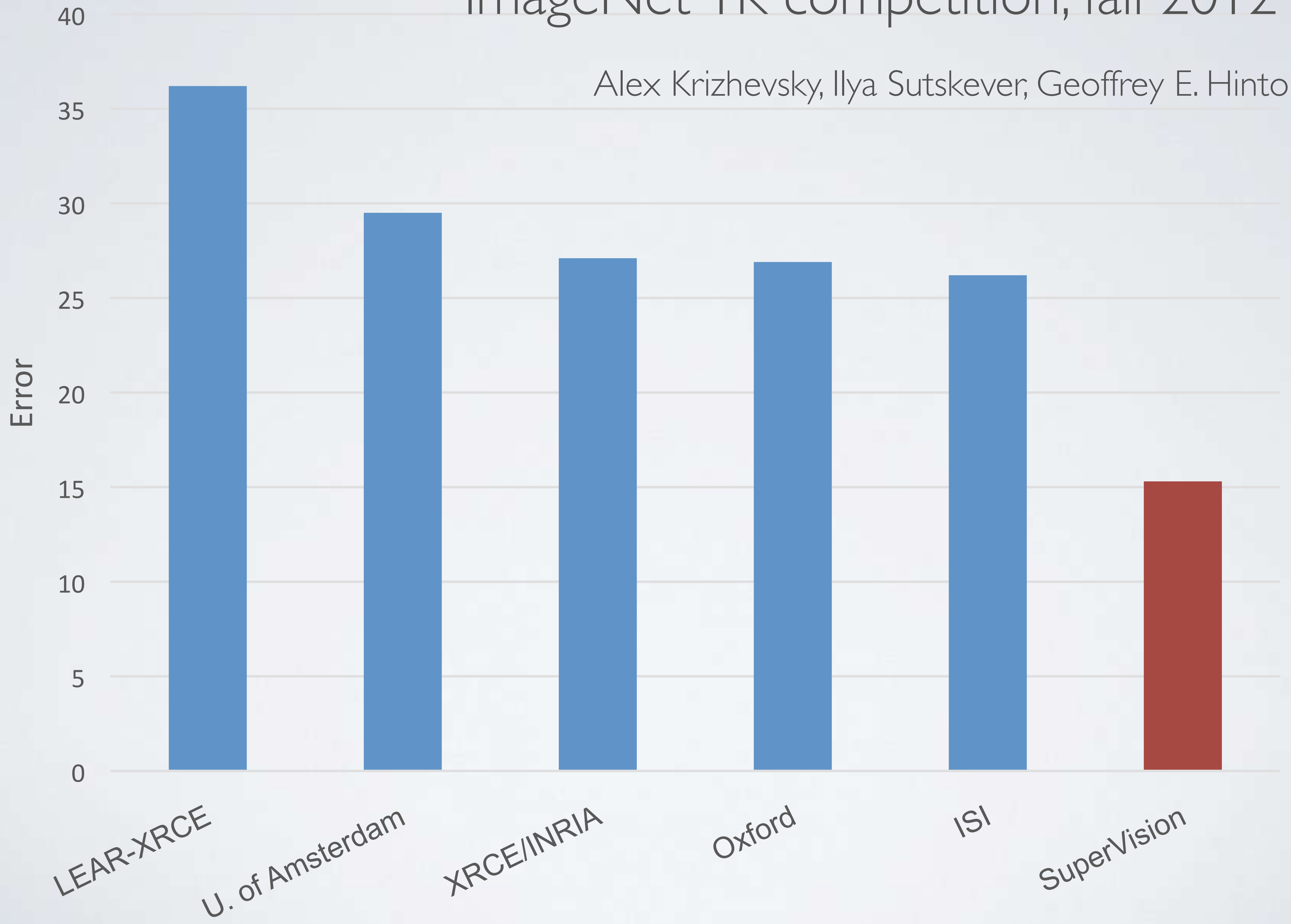
Input image

Output class prediction

CONVNET IN ACTION

ImageNet 1K competition, fall 2012

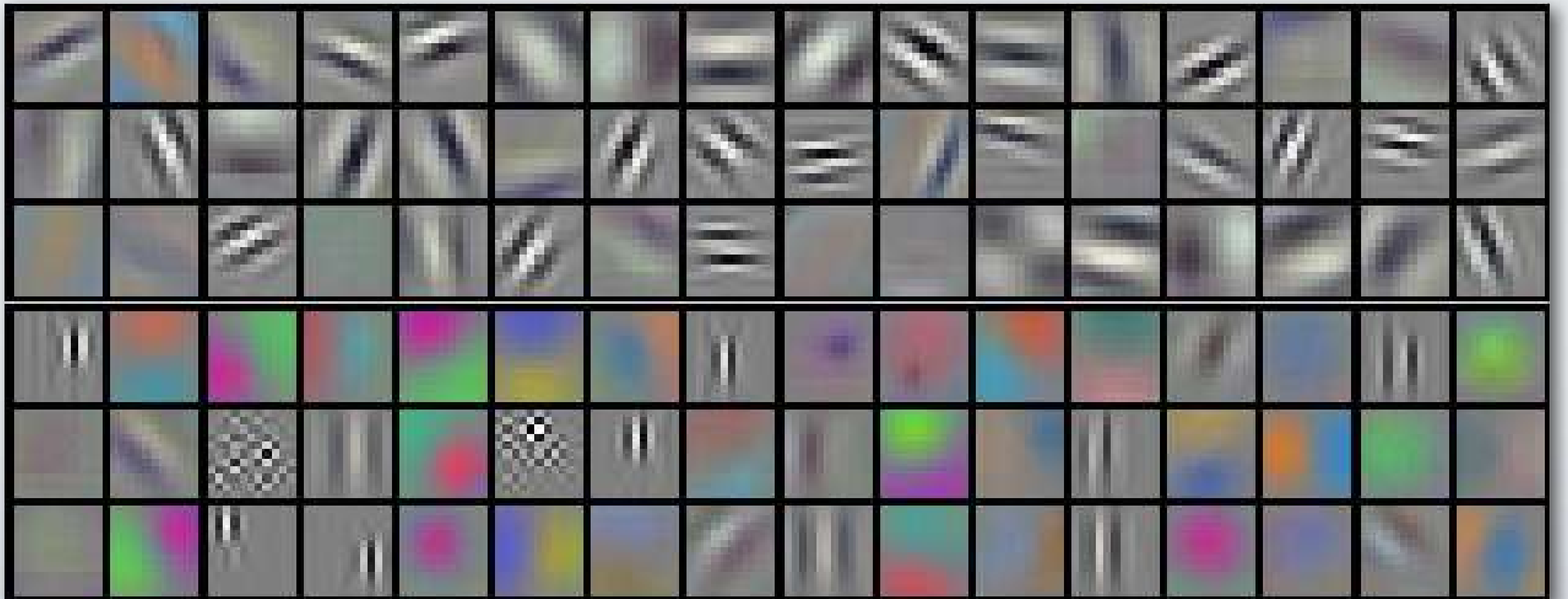
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012



CONVNET IN ACTION

- Training paradigm:
  - Rectified linear activation functions.
  - Trained with **Dropout**.
  - Dataset expansion (data augmentation) employed.

96 low-level learned features:



ONE YEAR LATER

ImageNet 1K competition, fall 2013

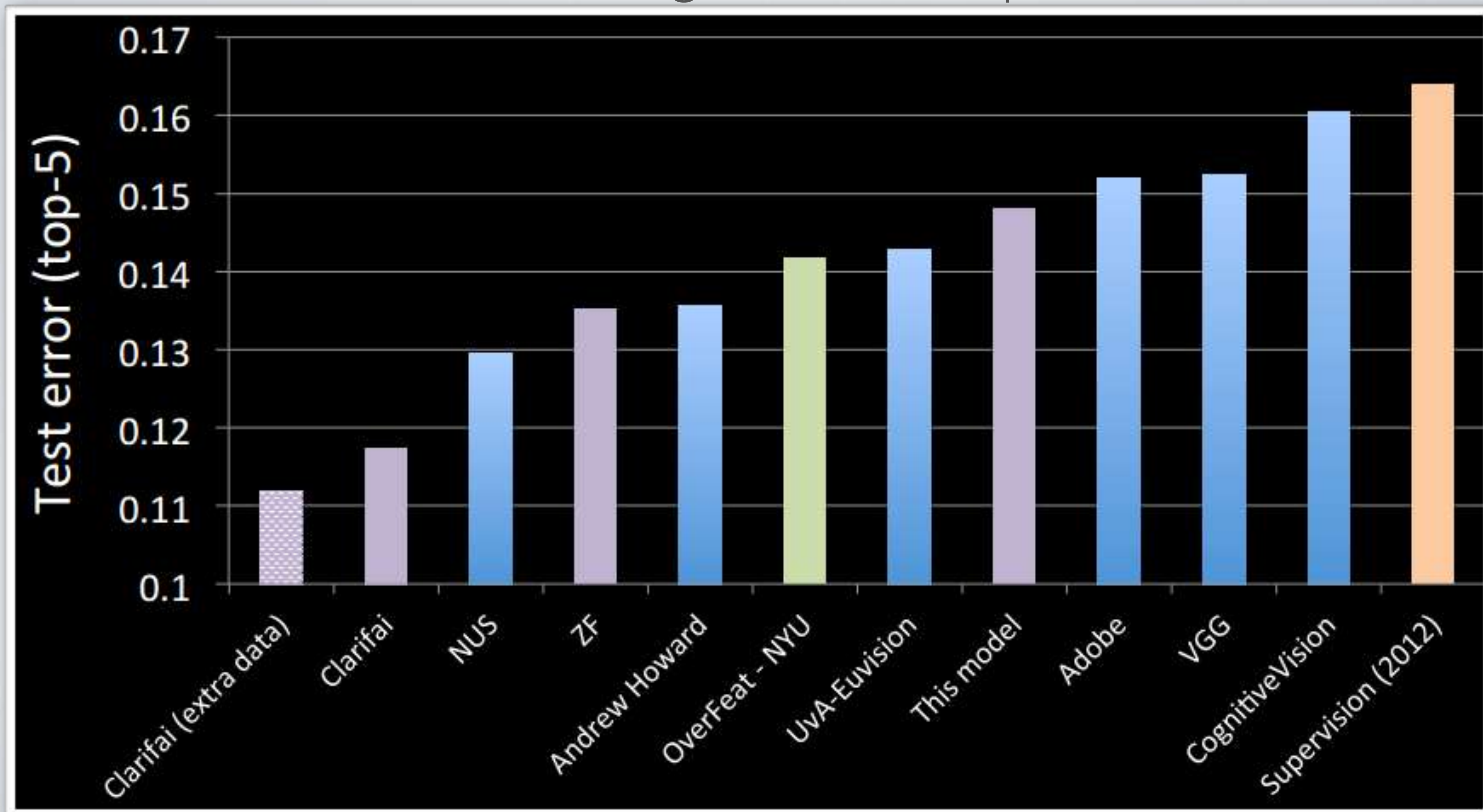
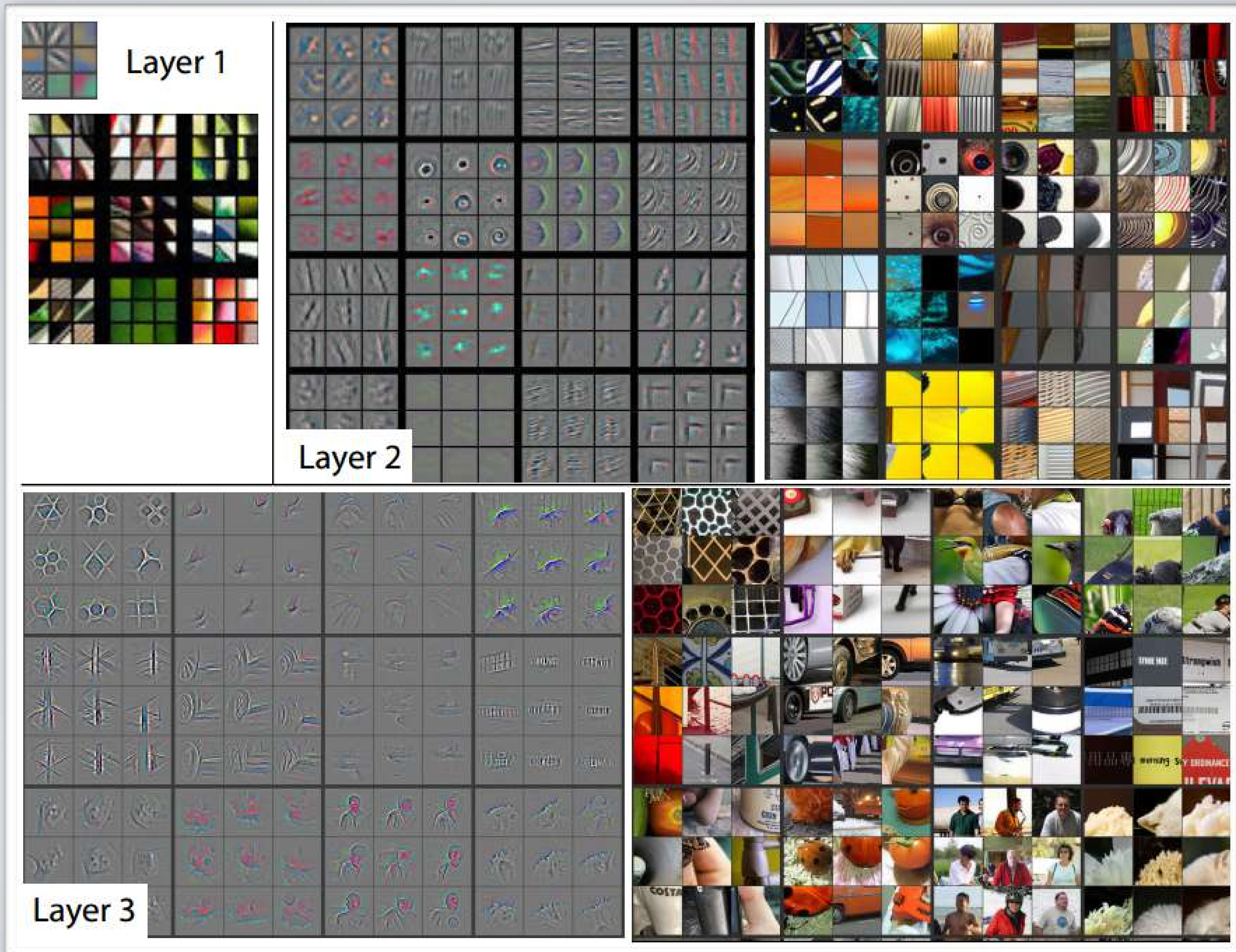


image from Li Deng, 2014

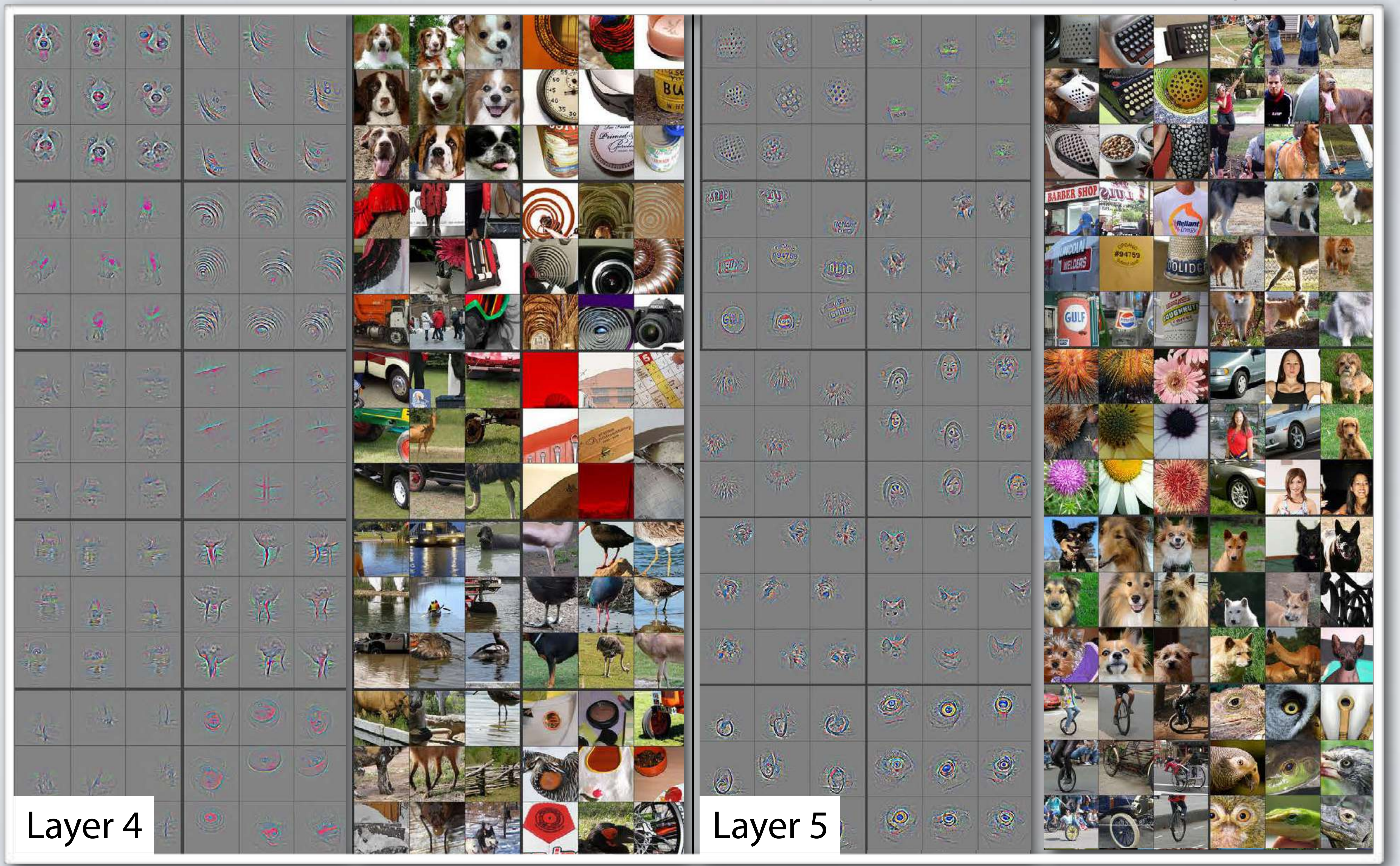
UNDERSTANDING CNNS

Image from Zeiler and Fergus, 2013



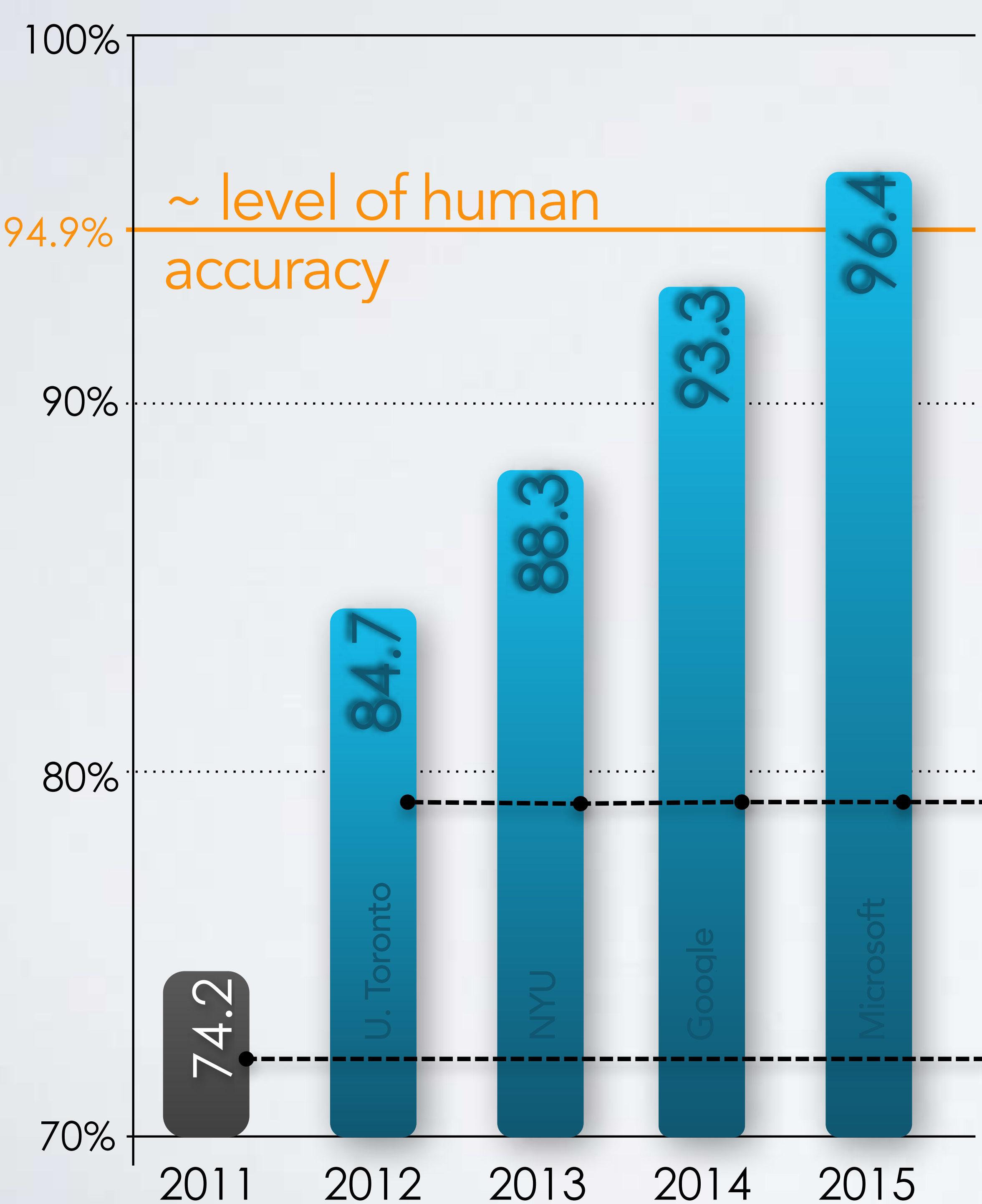
UNDERSTANDING CNNS

Image from Zeiler and Fergus, 2013



IMAGENET ACCURACY STILL IMPROVING

TOP-5 CLASSIFICATION TASK



ResNet

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Use

Deep Learning

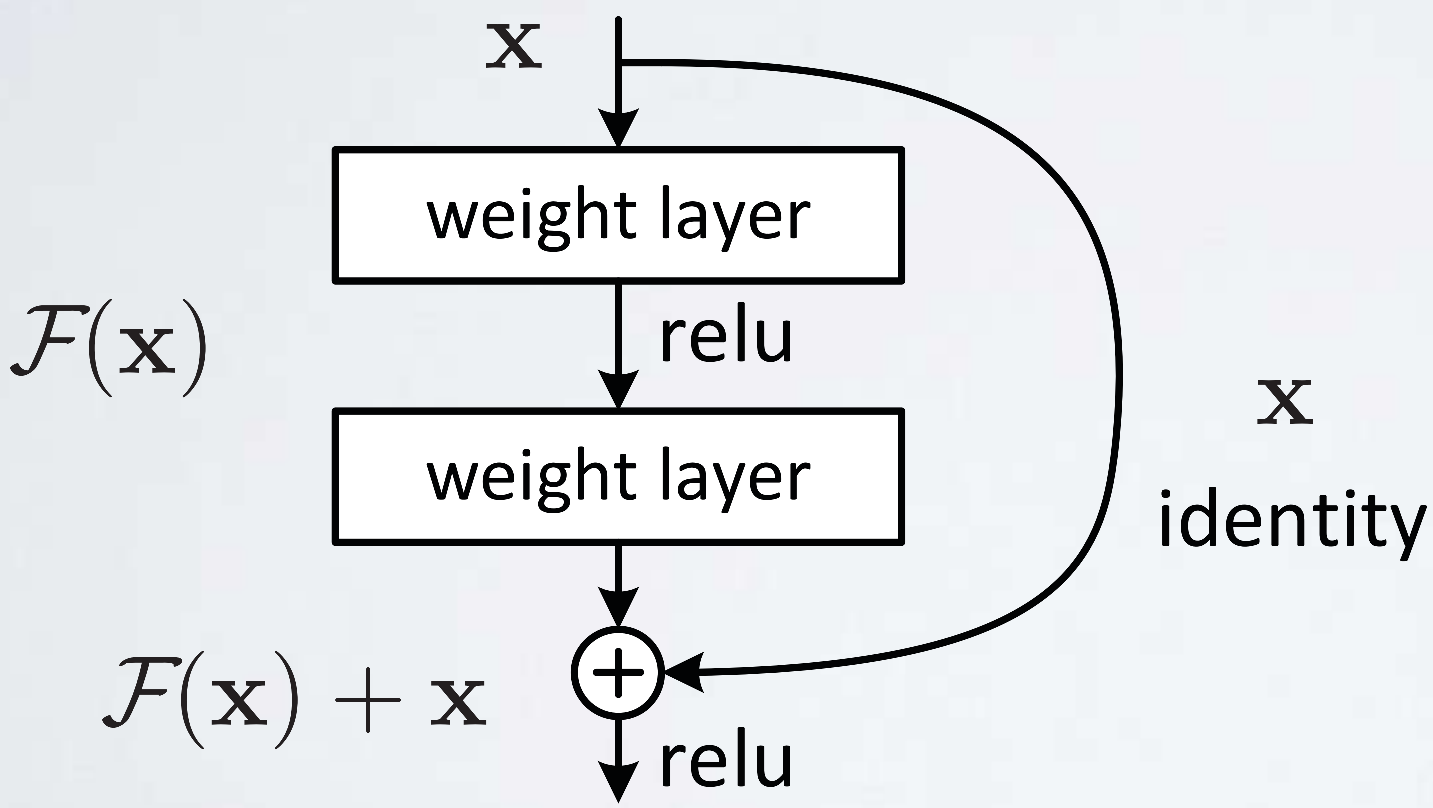
over

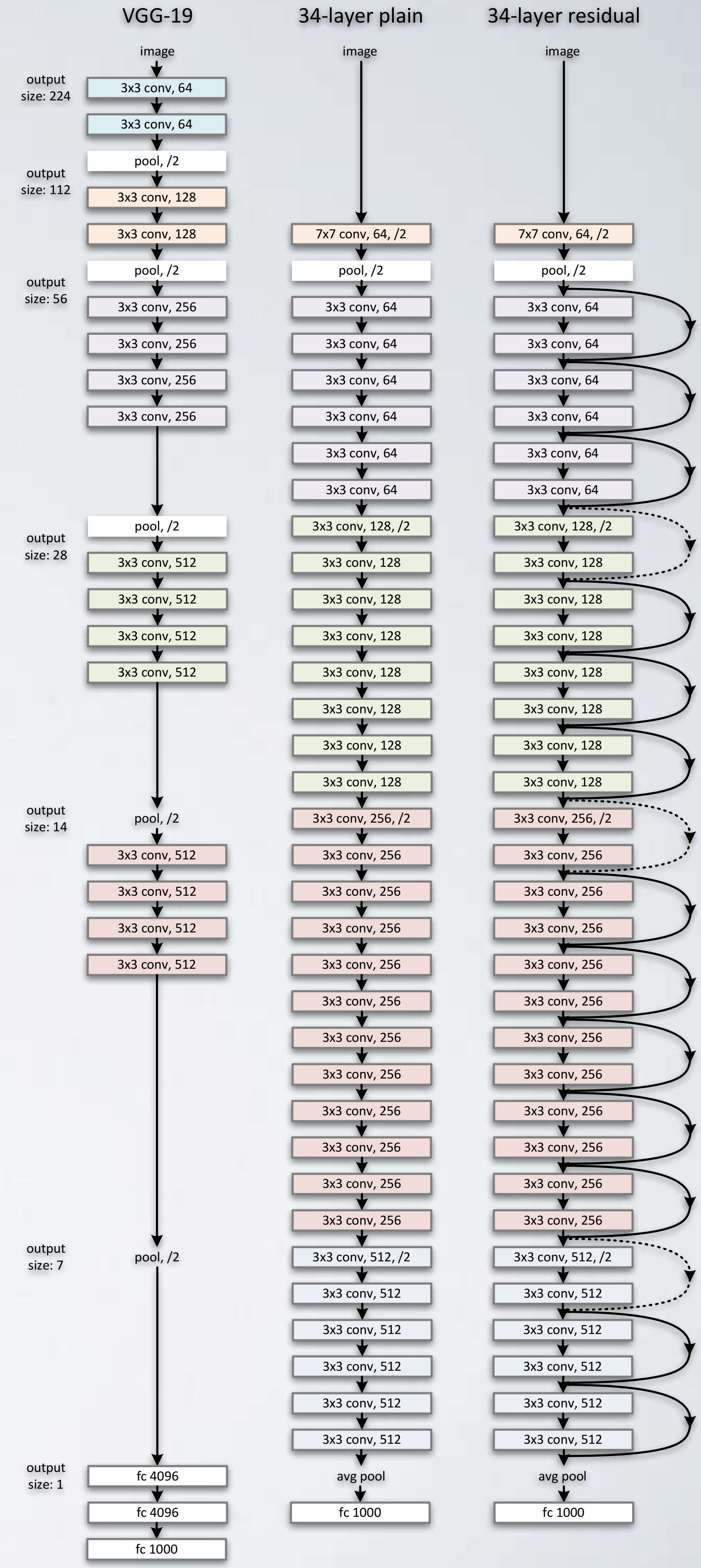
Conventional Computer Vision

RESNETS

(HE, ZHANG, REN AND SUN, 2015)

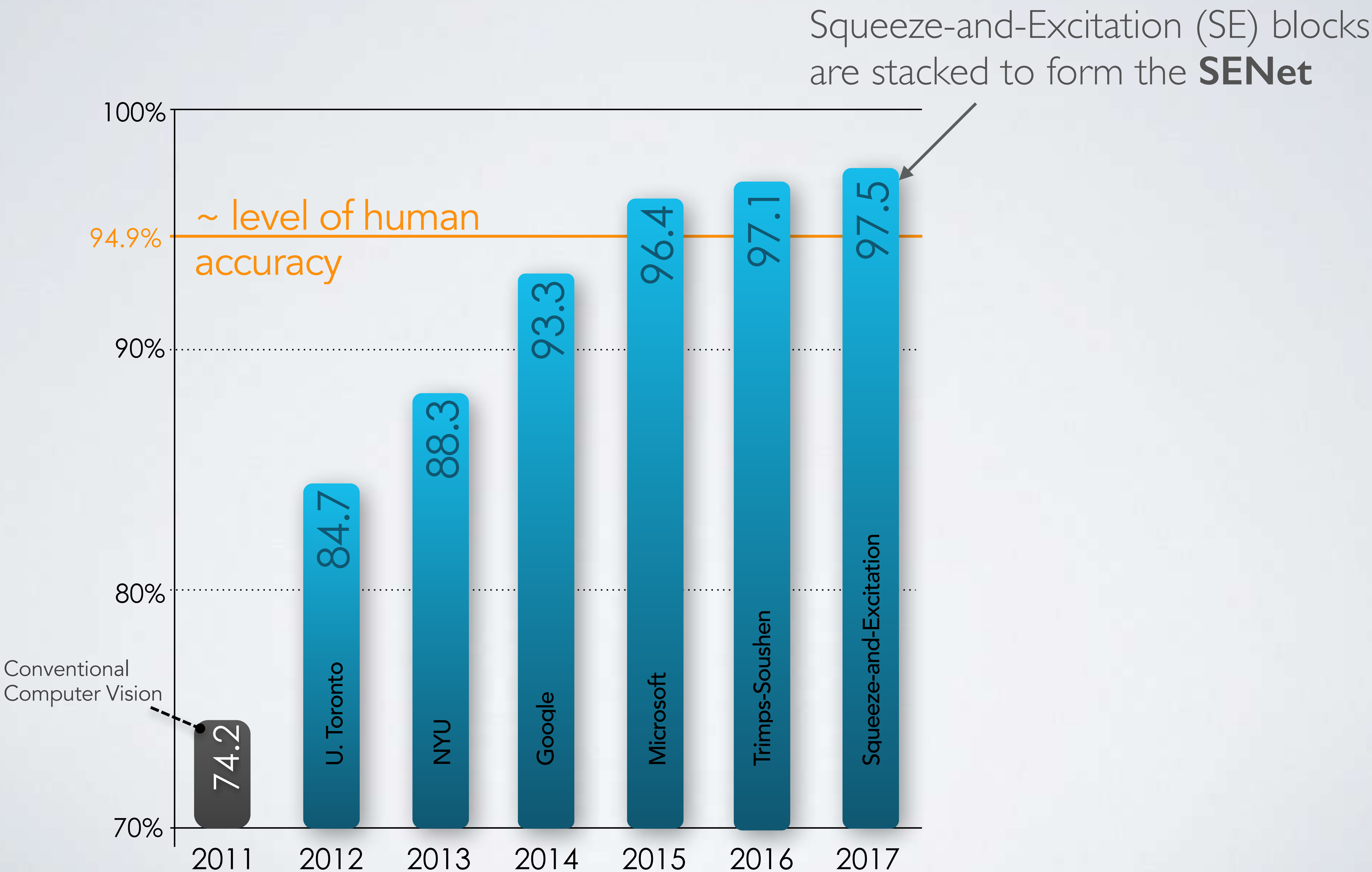
- Lastest state-of-the-art for ImageNet object recognition challenge.
- Uses "shortcut" connections that bypass the nonlinearity





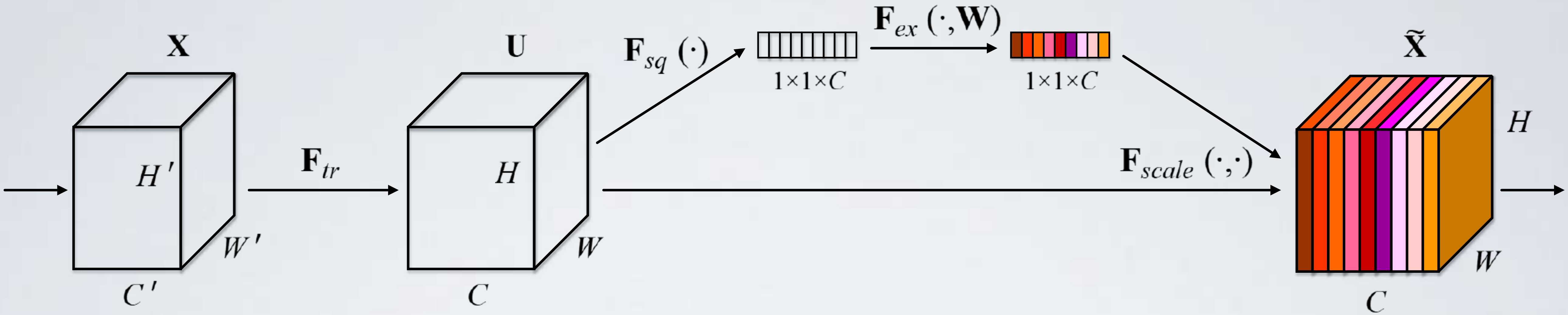
IMAGENET ACCURACY STILL IMPROVING

TOP-5 CLASSIFICATION TASK



SQUEEZE-AND-EXCITATION

(HU, SHEN AND SUN, 2017)



$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s.$$

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} u_c(i,j)$$

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z}))$$

$$\widetilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c$$

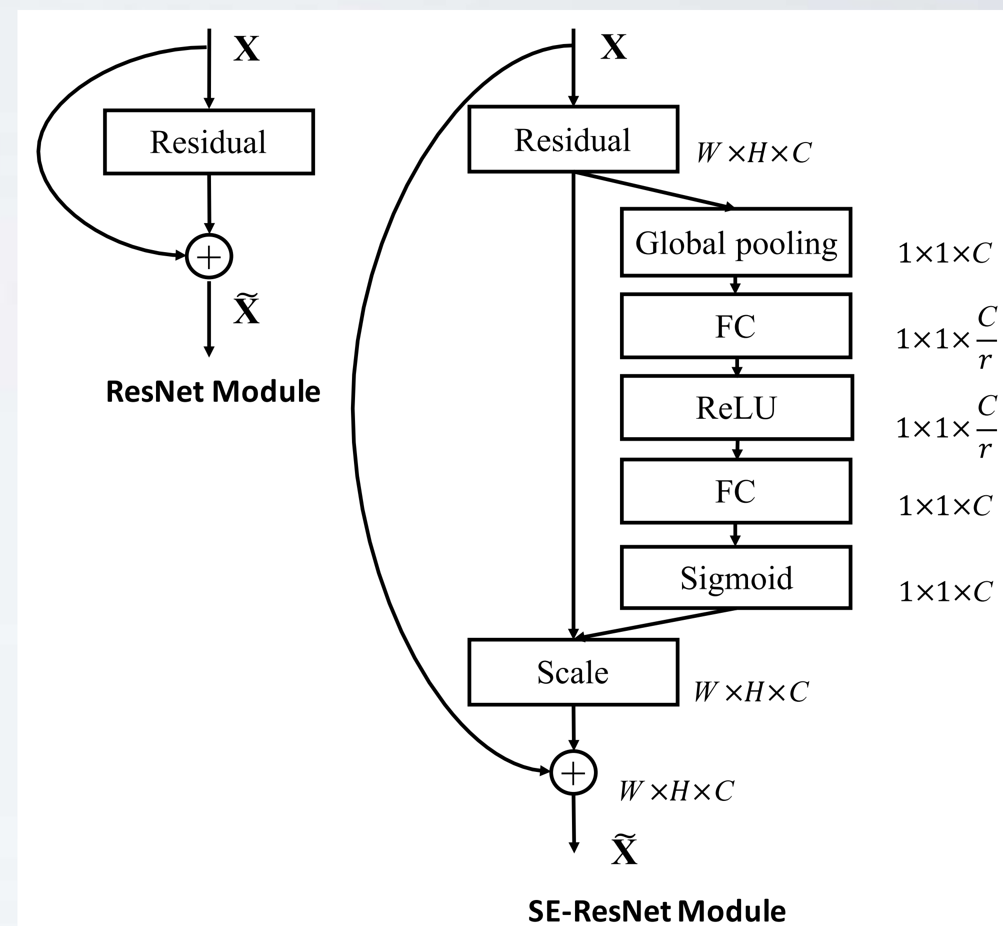


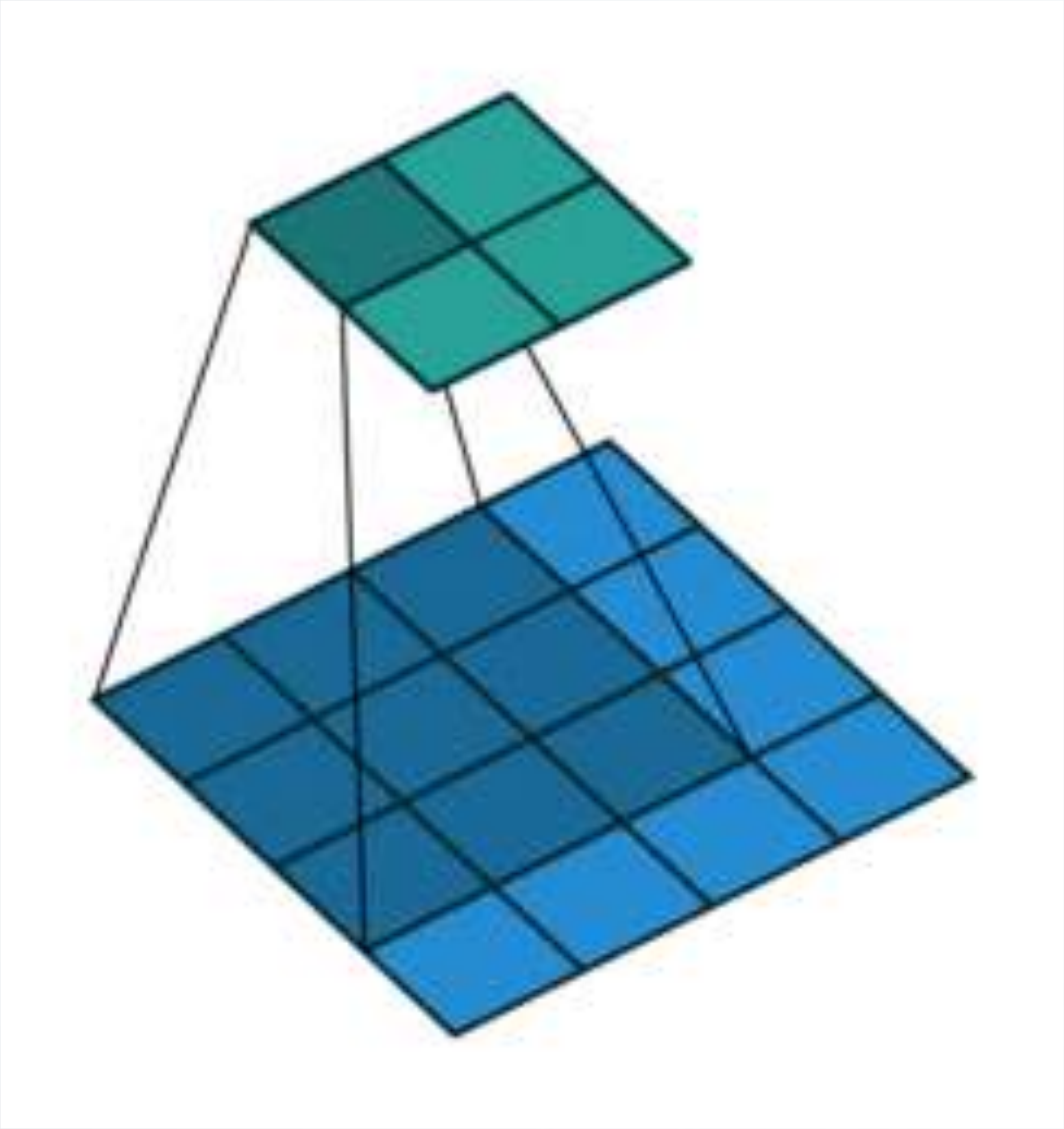
Figure 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Convolutions in more detail

material for this part of the lecture is taken mainly from the theano tutorial:
http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html

CONVOLUTION AS A MATRIX OPERATION

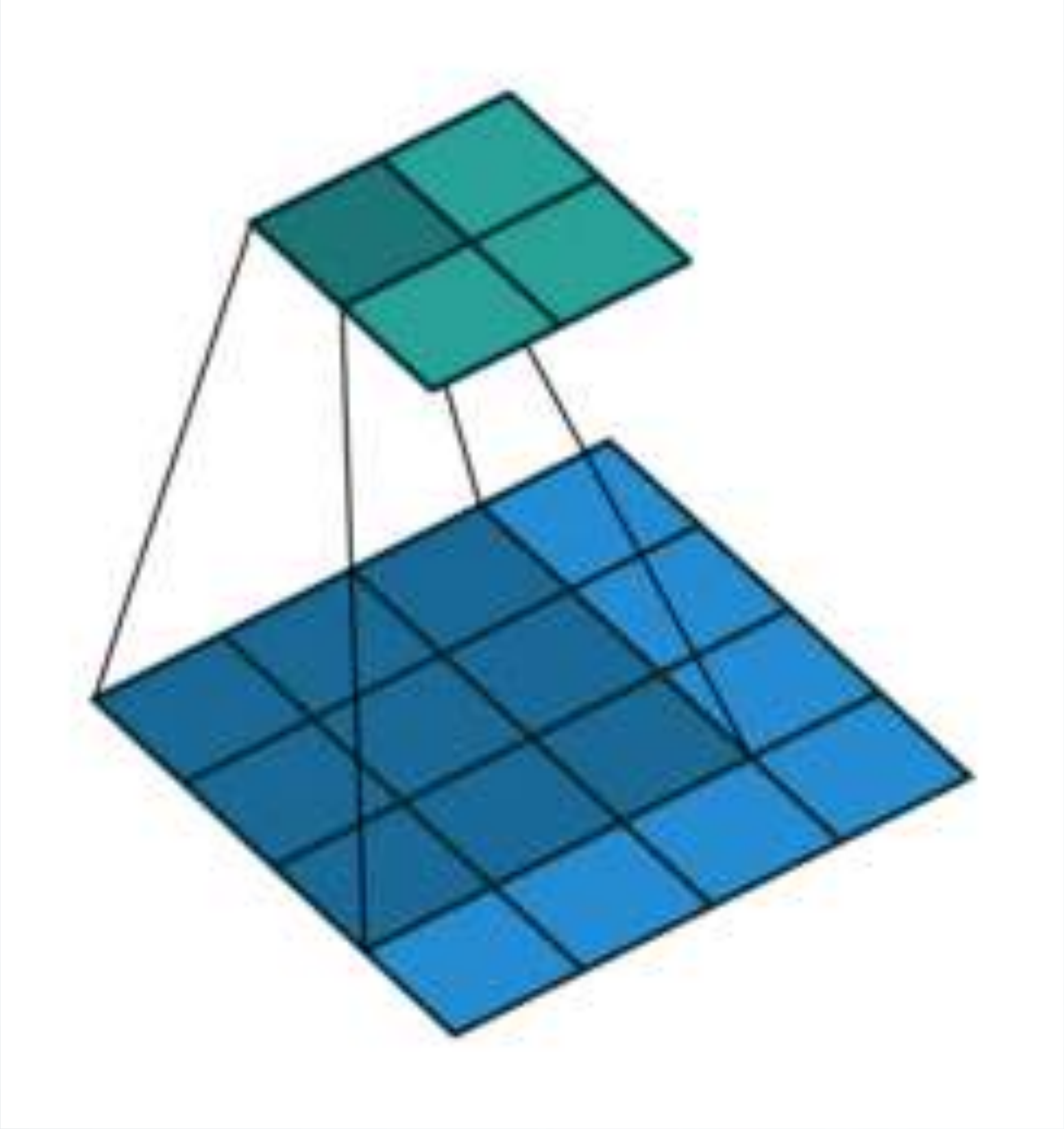
- Convolutions can be represented as a sparse matrix multiply:
  - unrolling the input and output into vectors (from left to right, top to bottom).



$$\begin{pmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ 0 & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{pmatrix}^T$$

CONVOLUTION AS A MATRIX OPERATION

- Convolutions can be represented as a sparse matrix multiply:
  - unrolling the input and output into vectors (from left to right, top to bottom).



$$\begin{pmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ 0 & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{pmatrix}^T$$

CNN: DYNAMIC SIZING

- How can we deal with variable sized inputs?
  - Use variable sized pooling regions to a fixed size (post-pooling) feature map.
    - lower feature map sizes will be a function of the input size.
  - Avoid fully-connected layers and use pooling to reduce down to a set of 1x1 feature maps.

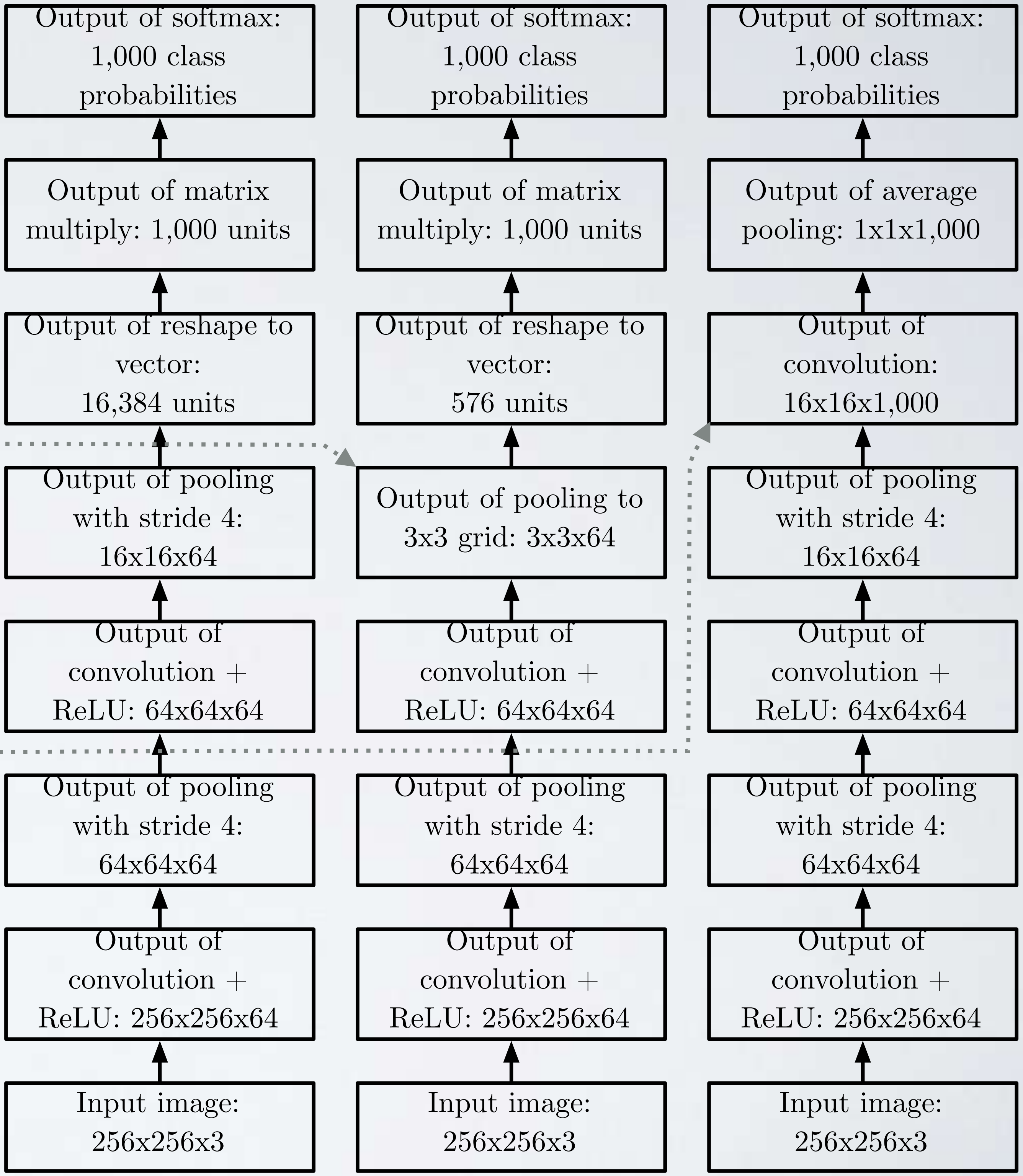
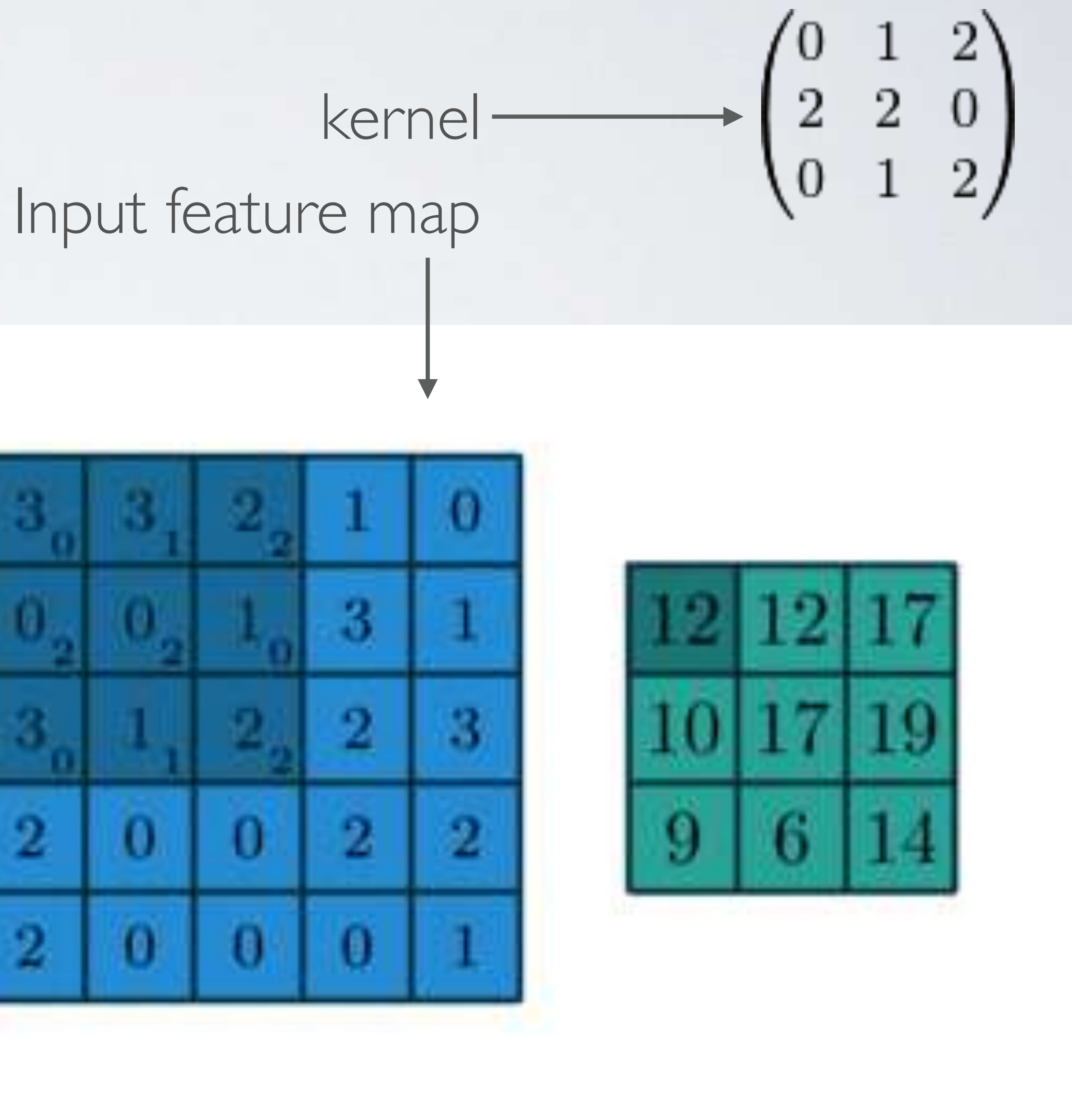


Image from Deep Learning Textbook (Goodfellow et al, 2016)

CONVOLUTION BASICS

Properties of images

- Stored as multi-dimensional arrays.
- Feature one or more axes for which ordering matters
  - IMAGE: width and height axes
  - AUDIO: time axis.
- One axis, called the channel axis, is used to access different views of the data
  - IMAGE: red, green and blue channels of color.
  - AUDIO: left and right channels of a stereo track.



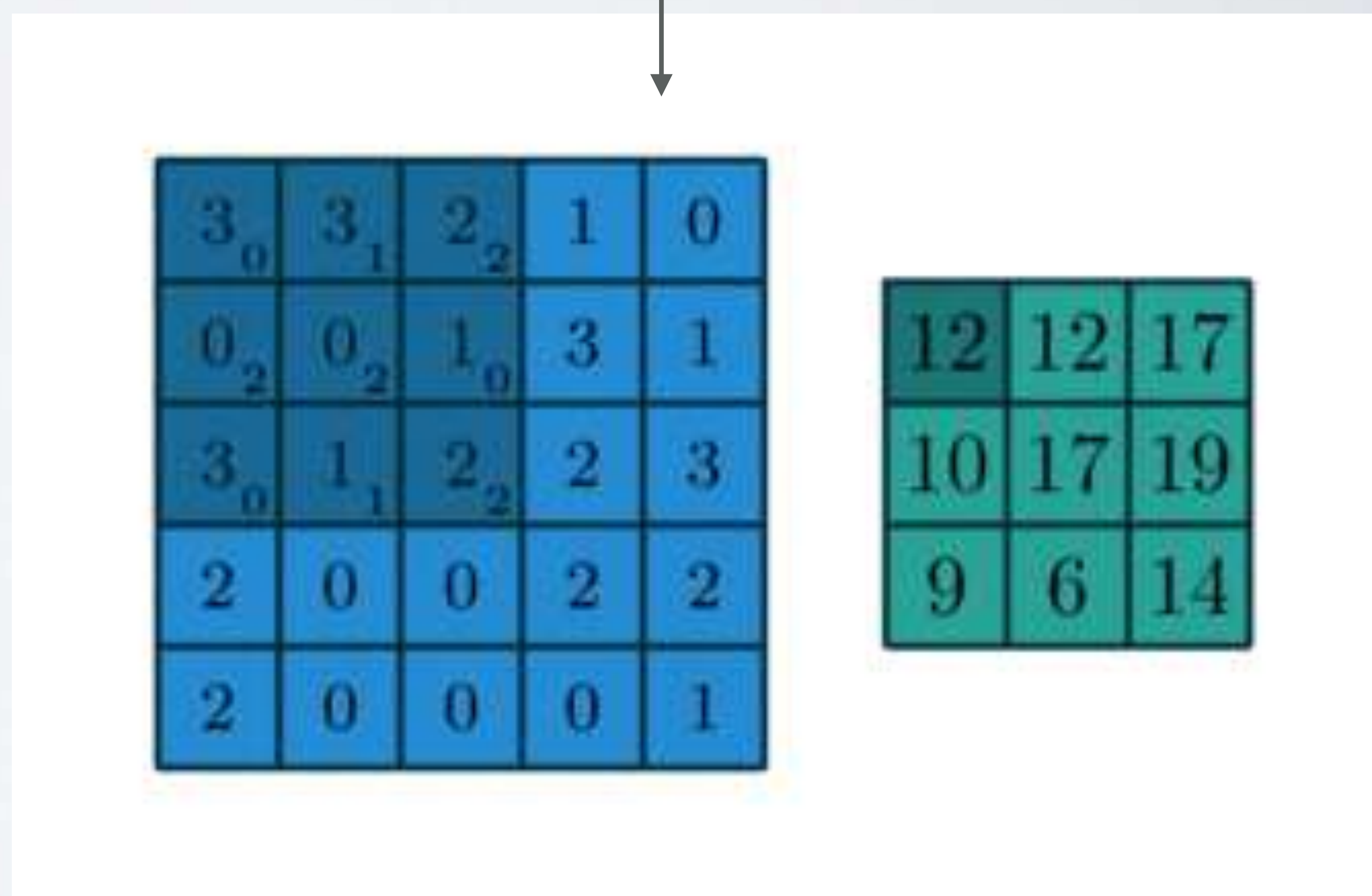
CONVOLUTION BASICS

Properties of images

- Stored as multi-dimensional arrays.
- Feature one or more axes for which ordering matters
  - IMAGE: width and height axes
  - AUDIO: time axis.
- One axis, called the channel axis, is used to access different views of the data
  - IMAGE: red, green and blue channels of color.
  - AUDIO: left and right channels of a stereo track.

kernel → $\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$

Input feature map

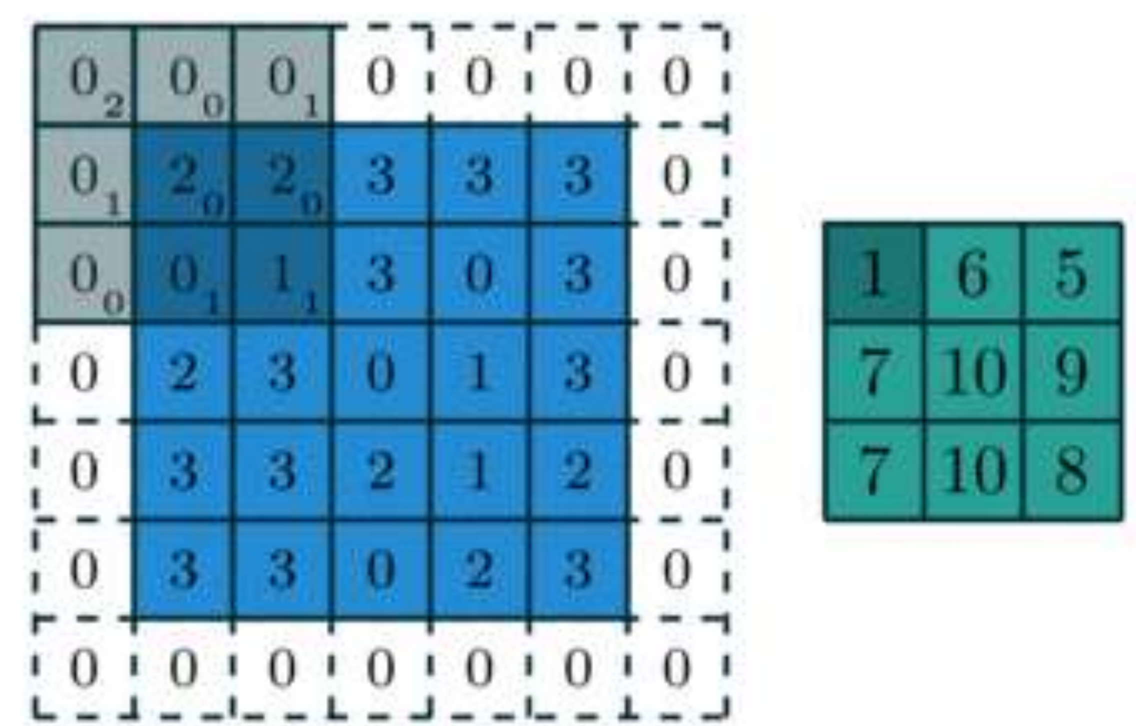


TERMINOLOGY

- Properties affecting the output size o_j along axis j:
  - i_j: input size along axis j,
  - k_j: kernel size along axis j,
  - s_j: stride (distance between two consecutive positions of the kernel) along axis j,
  - p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

$$n \equiv \text{number of output feature maps},$$
$$m \equiv \text{number of input feature maps},$$
$$k_j \equiv \text{kernel size along axis } j.$$

E.g. 3x3 kernel, 5x5 input, zero padding 1x1, stride: 2x2:

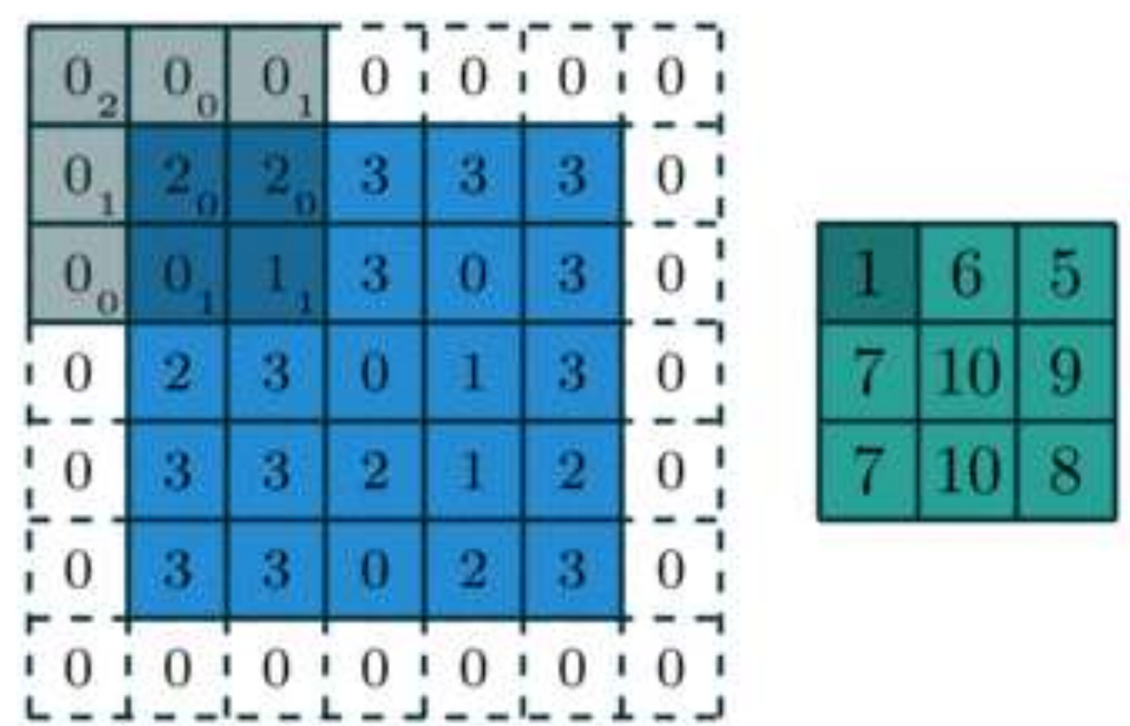


TERMINOLOGY

- Properties affecting the output size o_j along axis j:
  - i_j: input size along axis j,
  - k_j: kernel size along axis j,
  - s_j: stride (distance between two consecutive positions of the kernel) along axis j,
  - p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

$$n \equiv \text{number of output feature maps,}$$
$$m \equiv \text{number of input feature maps,}$$
$$k_j \equiv \text{kernel size along axis } j.$$

E.g. 3x3 kernel, 5x5 input, zero padding 1x1, stride: 2x2:



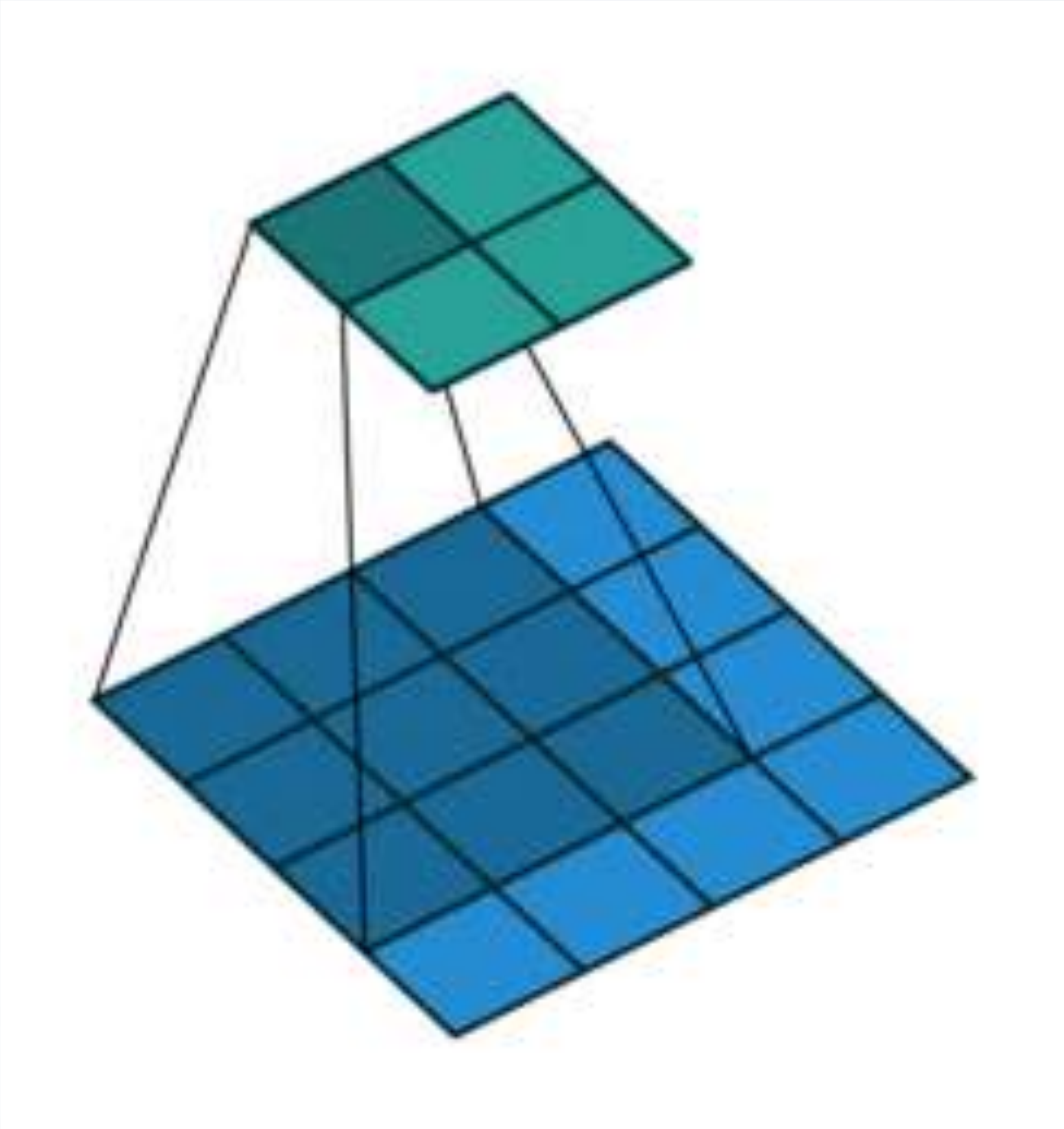
TERMINOLOGY

- Simplifying assumptions:
  - 2-D discrete convolutions ($N = 2$),
  - square inputs ($i_1 = i_2 = i$),
  - square kernel size ($k_1 = k_2 = k$),
  - same strides along axes ($s_1 = s_2 = s$)
  - same zero padding along both axes ($p_1 = p_2 = p$).
- Can comput the output dimension as a function of these quantities:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

floor function

E.g. $i = 4$ and $k = 3$, $s = 1$ and $p = 0$.



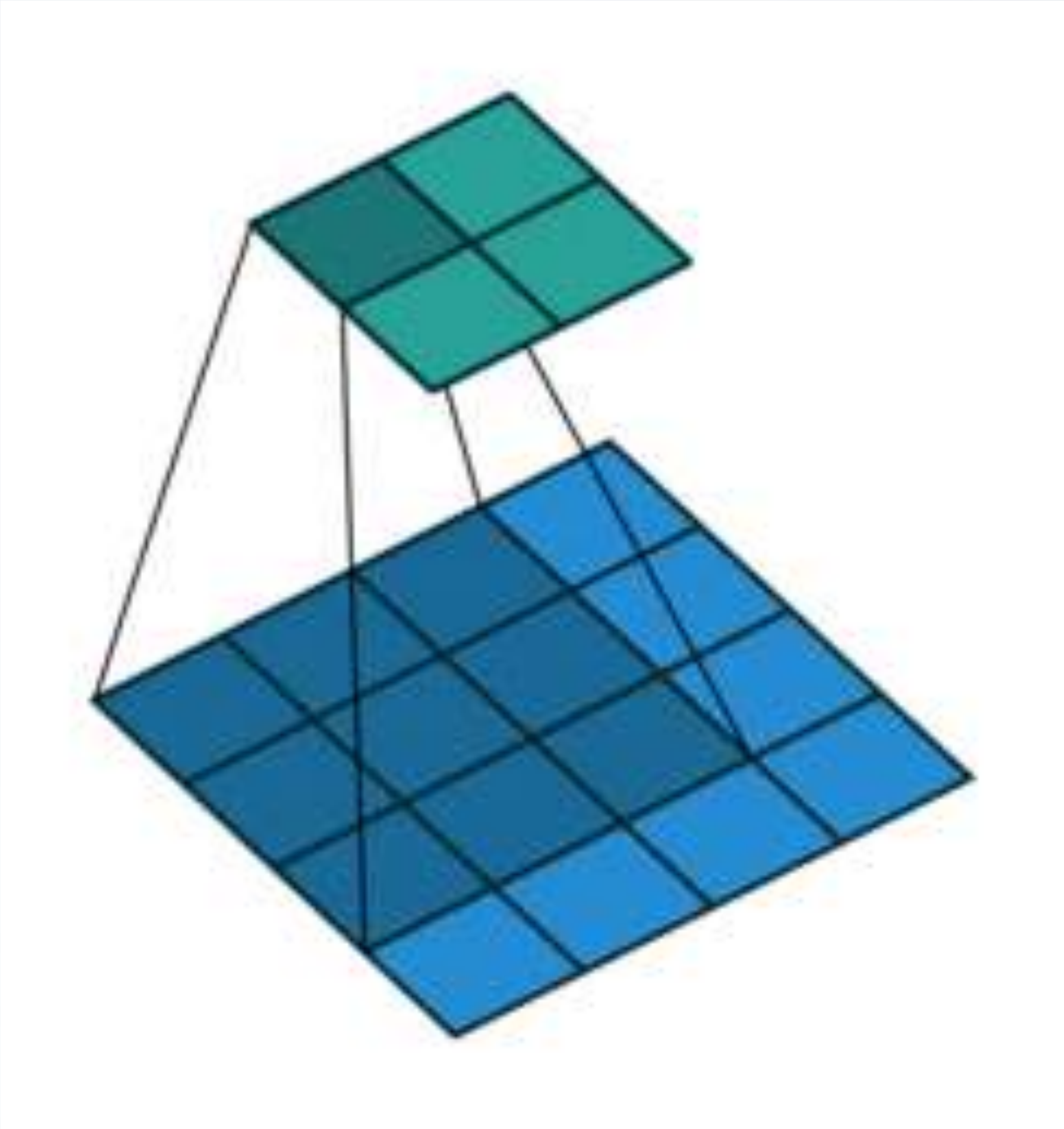
TERMINOLOGY

- Simplifying assumptions:
  - 2-D discrete convolutions ($N = 2$),
  - square inputs ($i_1 = i_2 = i$),
  - square kernel size ($k_1 = k_2 = k$),
  - same strides along axes ($s_1 = s_2 = s$)
  - same zero padding along both axes ($p_1 = p_2 = p$).
- Can comput the output dimension as a function of these quantities:

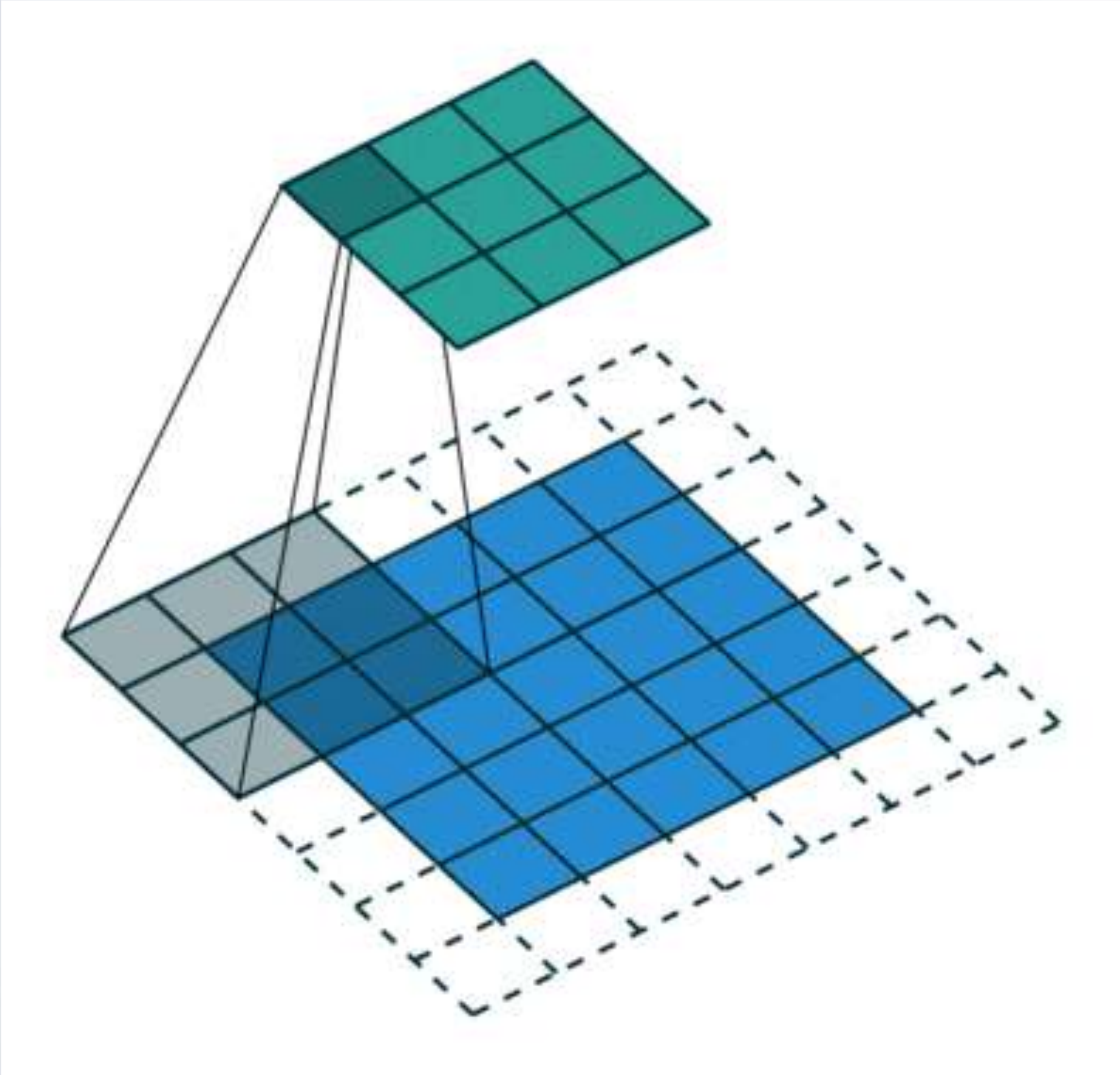
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

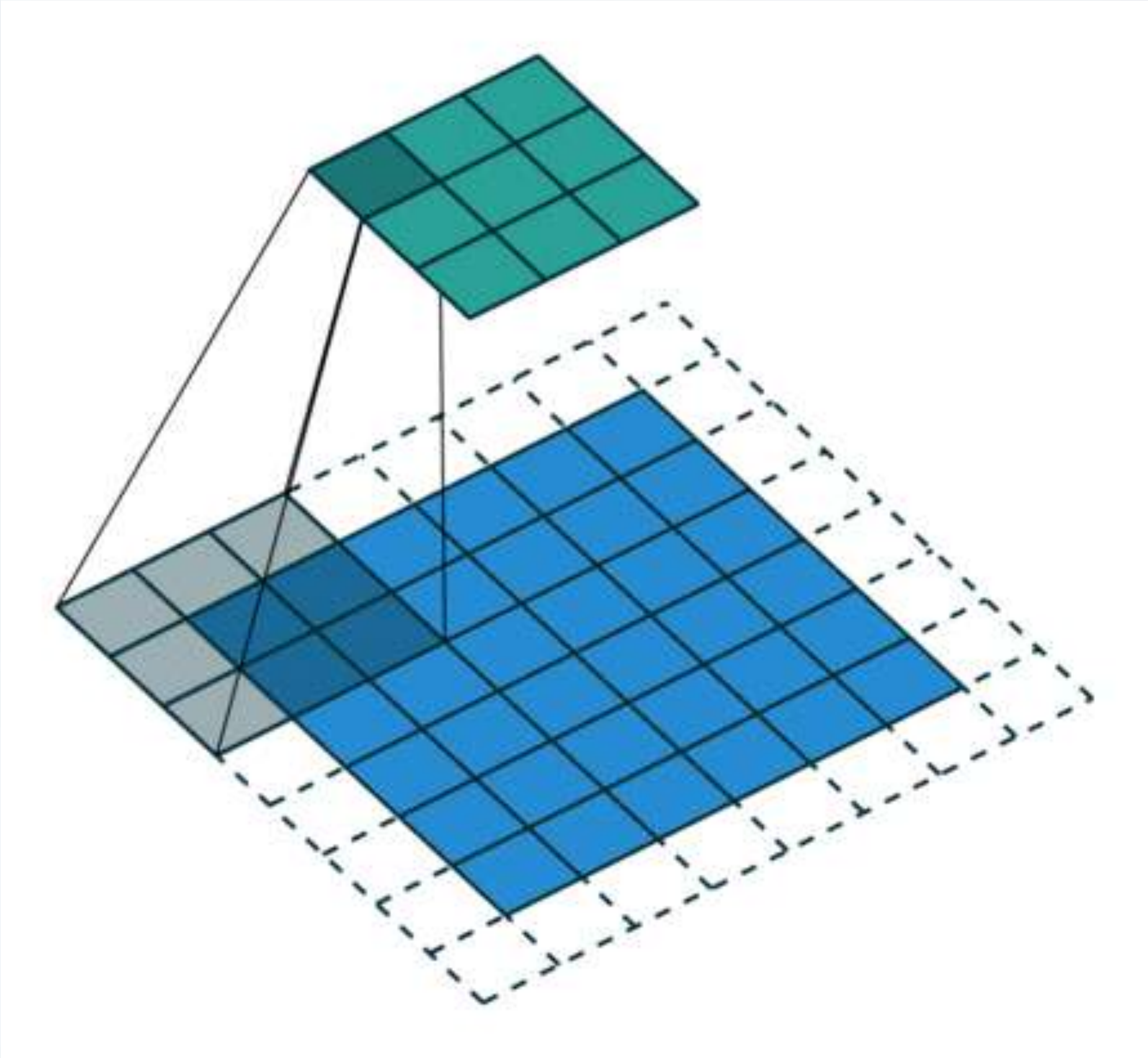
floor function

E.g. $i = 4$ and $k = 3$, $s = 1$ and $p = 0$.

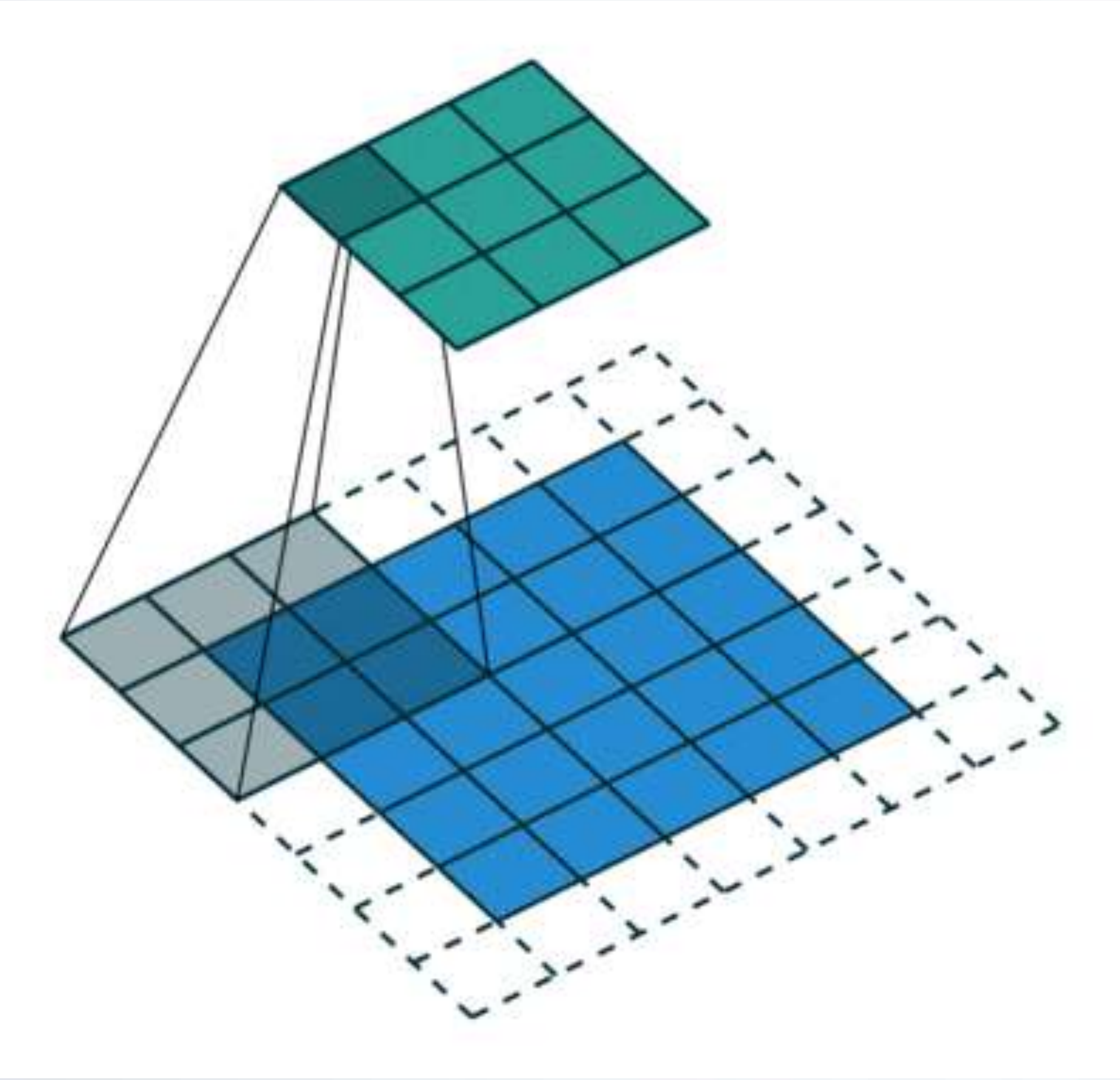


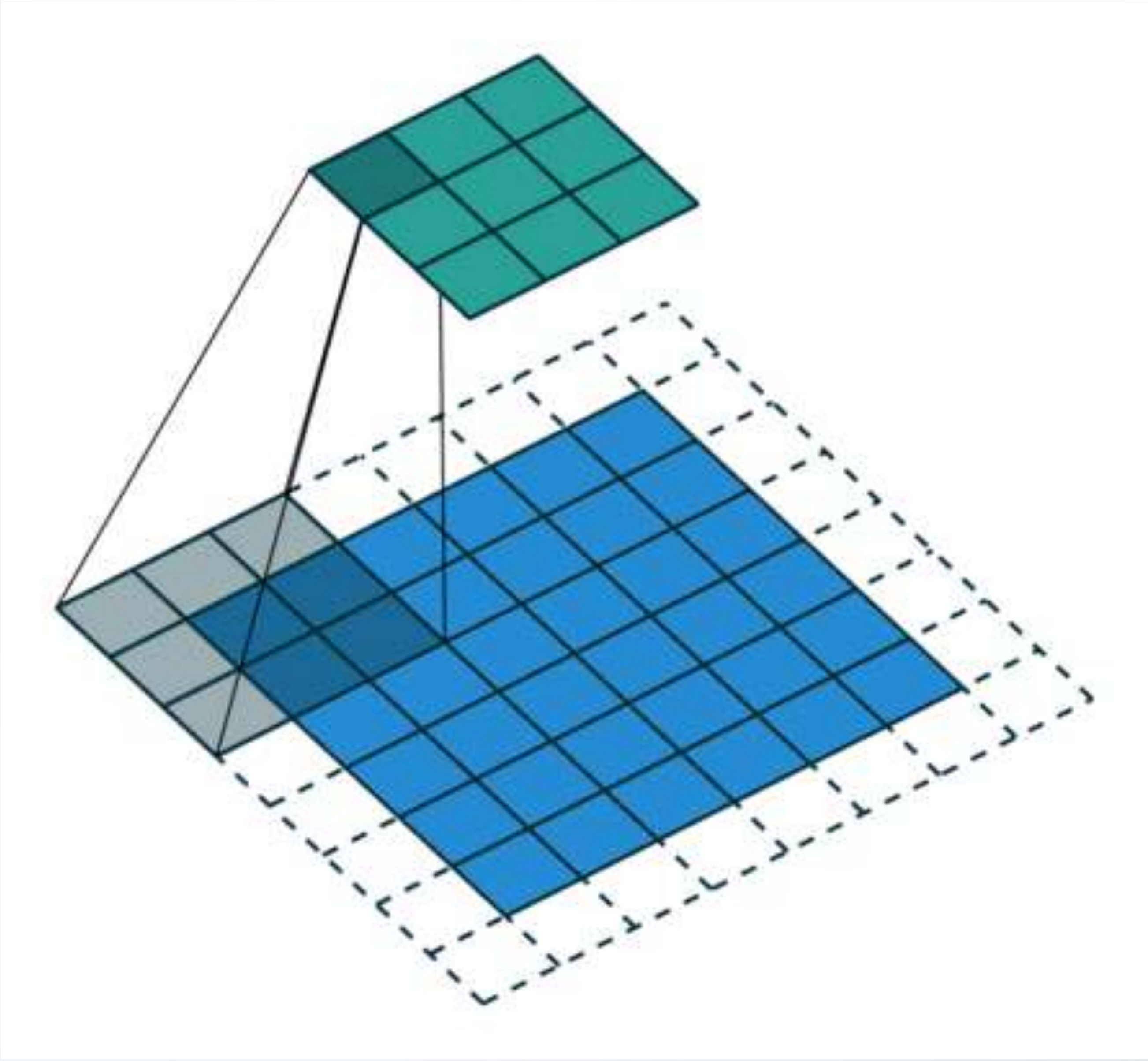
CONVOLUTIONS…





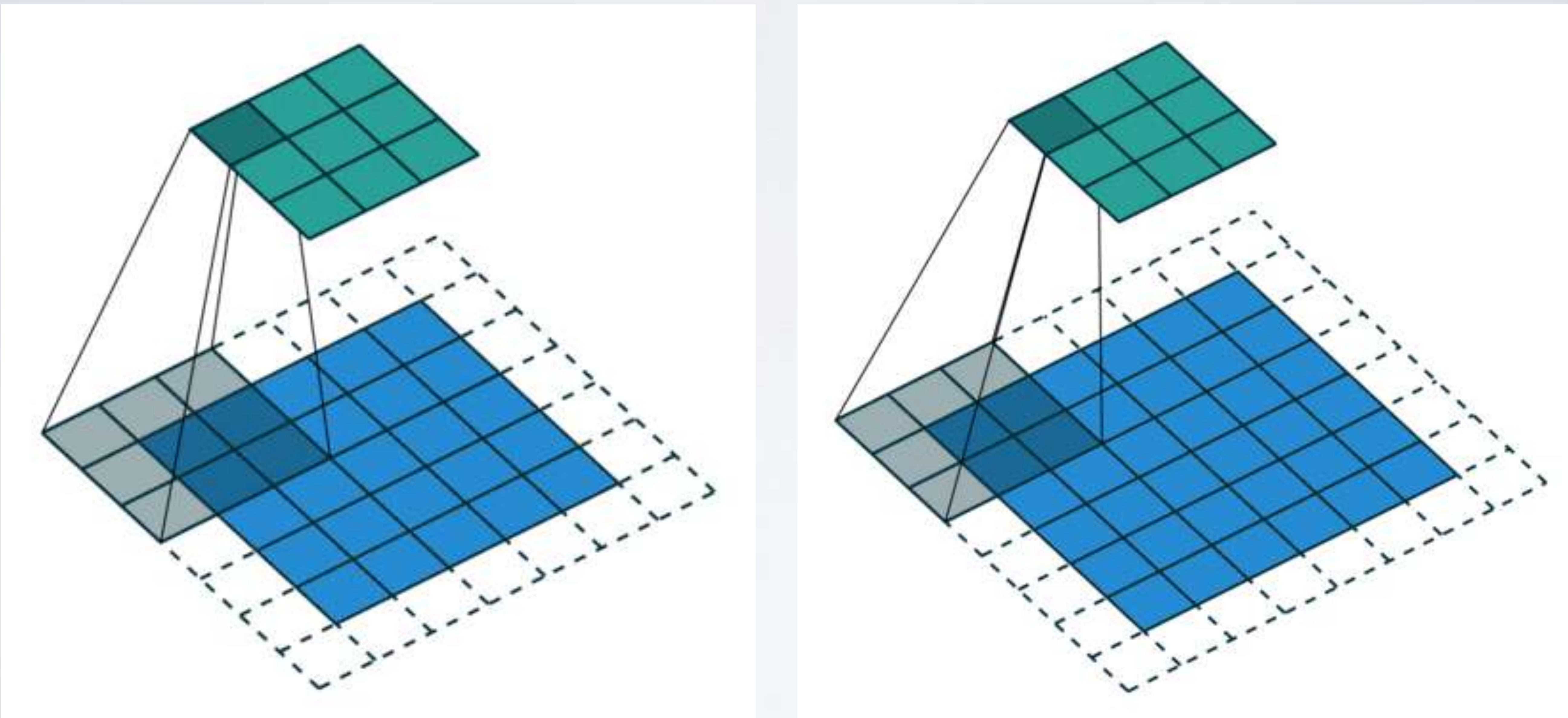
CONVOLUTIONS…





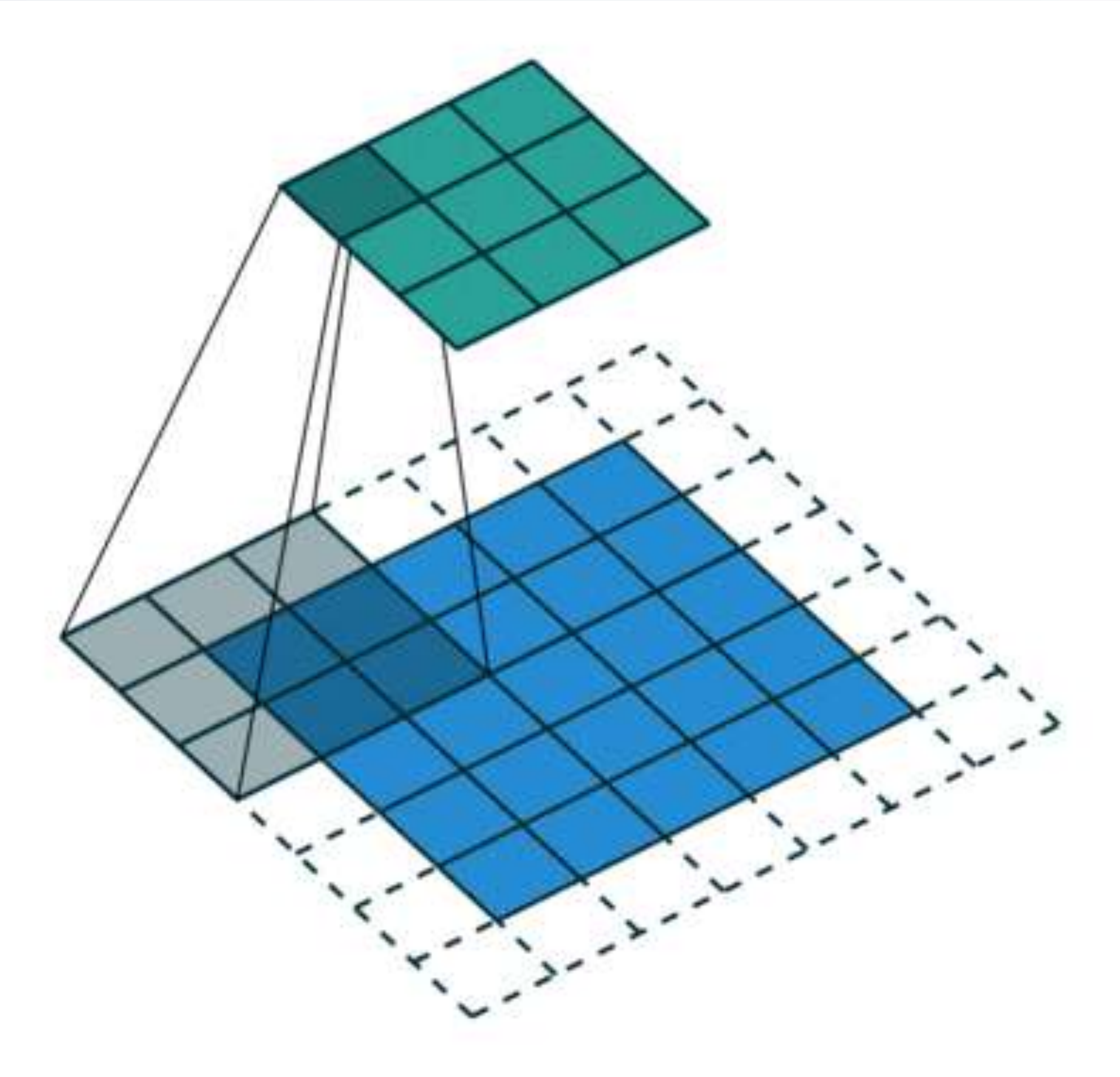
CONVOLUTIONS…

$i = 5$ and $k = 3$, $s = 2$ and $p = 1$.

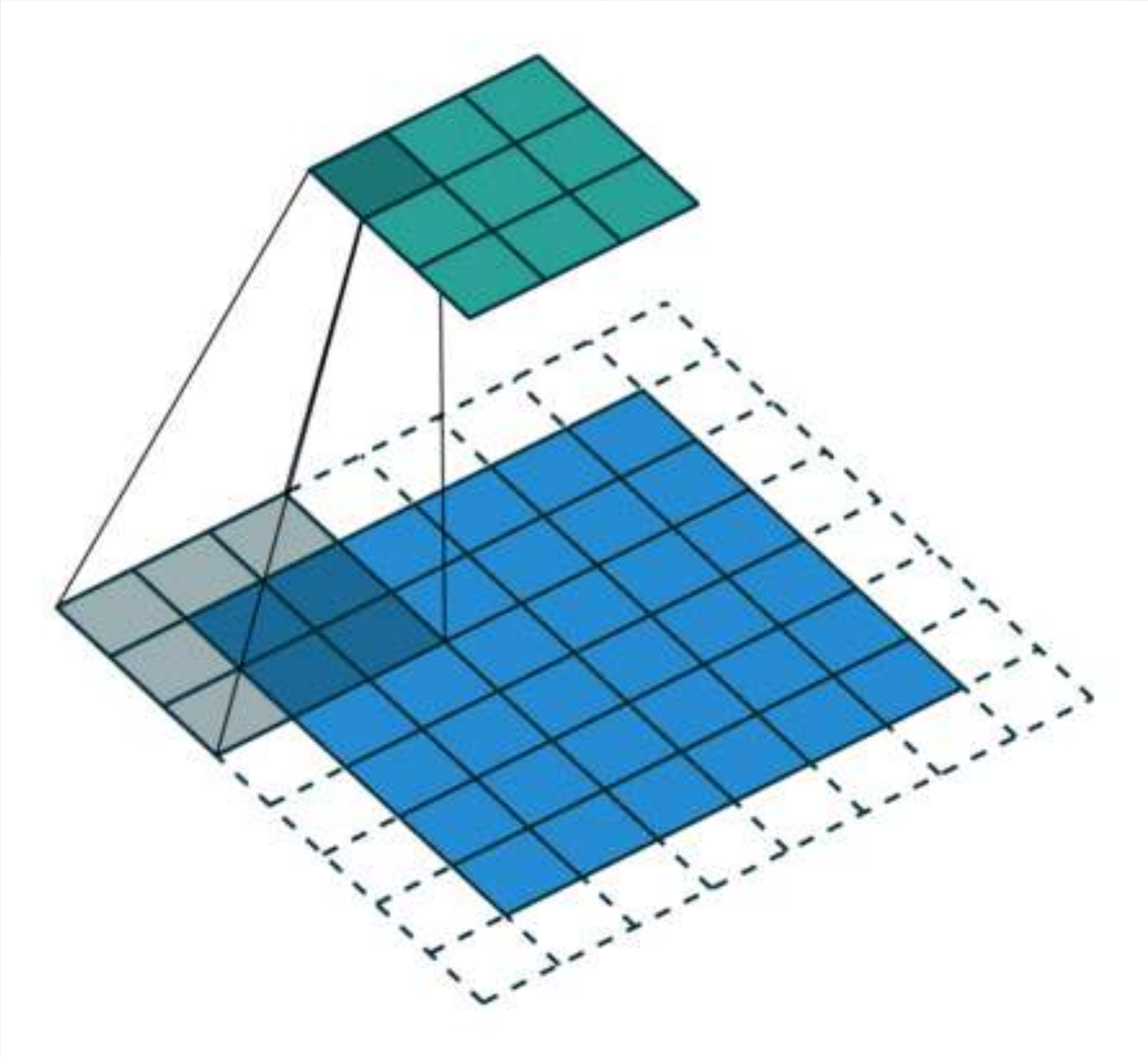


CONVOLUTIONS…

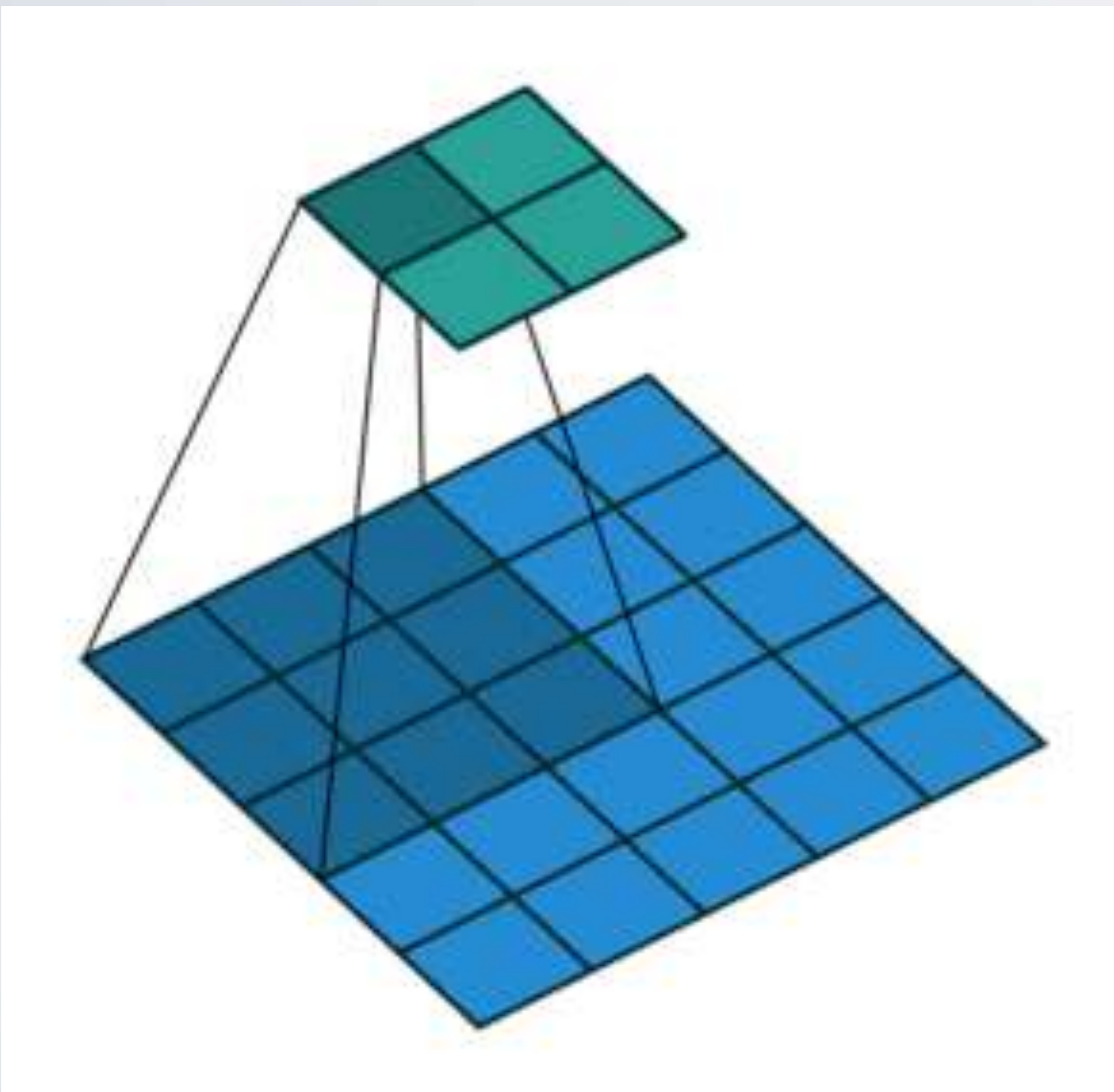
$i = 5$ and $k = 3$, $s = 2$ and $p = 1$.

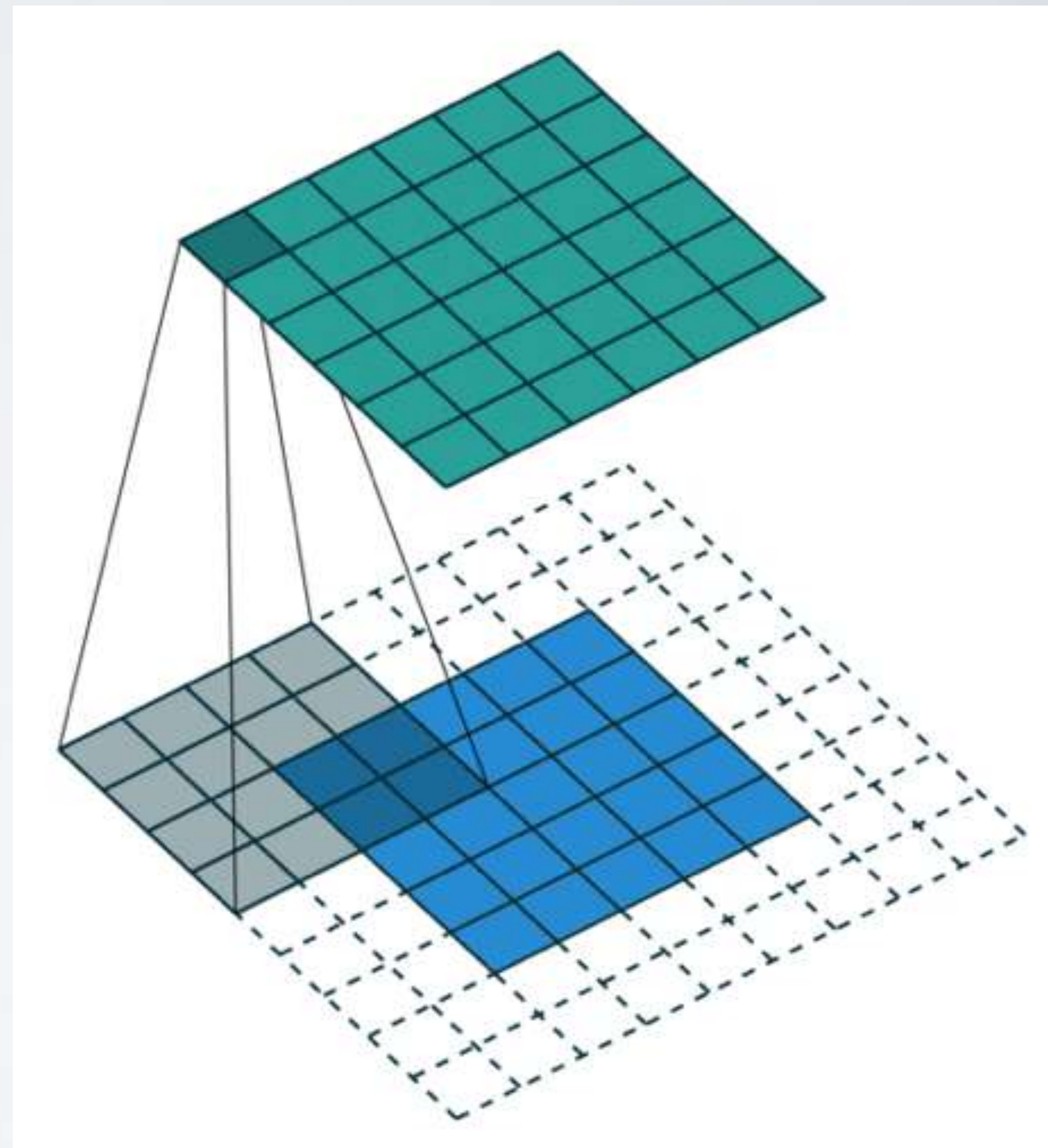


$i = 6$ and $k = 3$, $s = 2$ and $p = 1$.

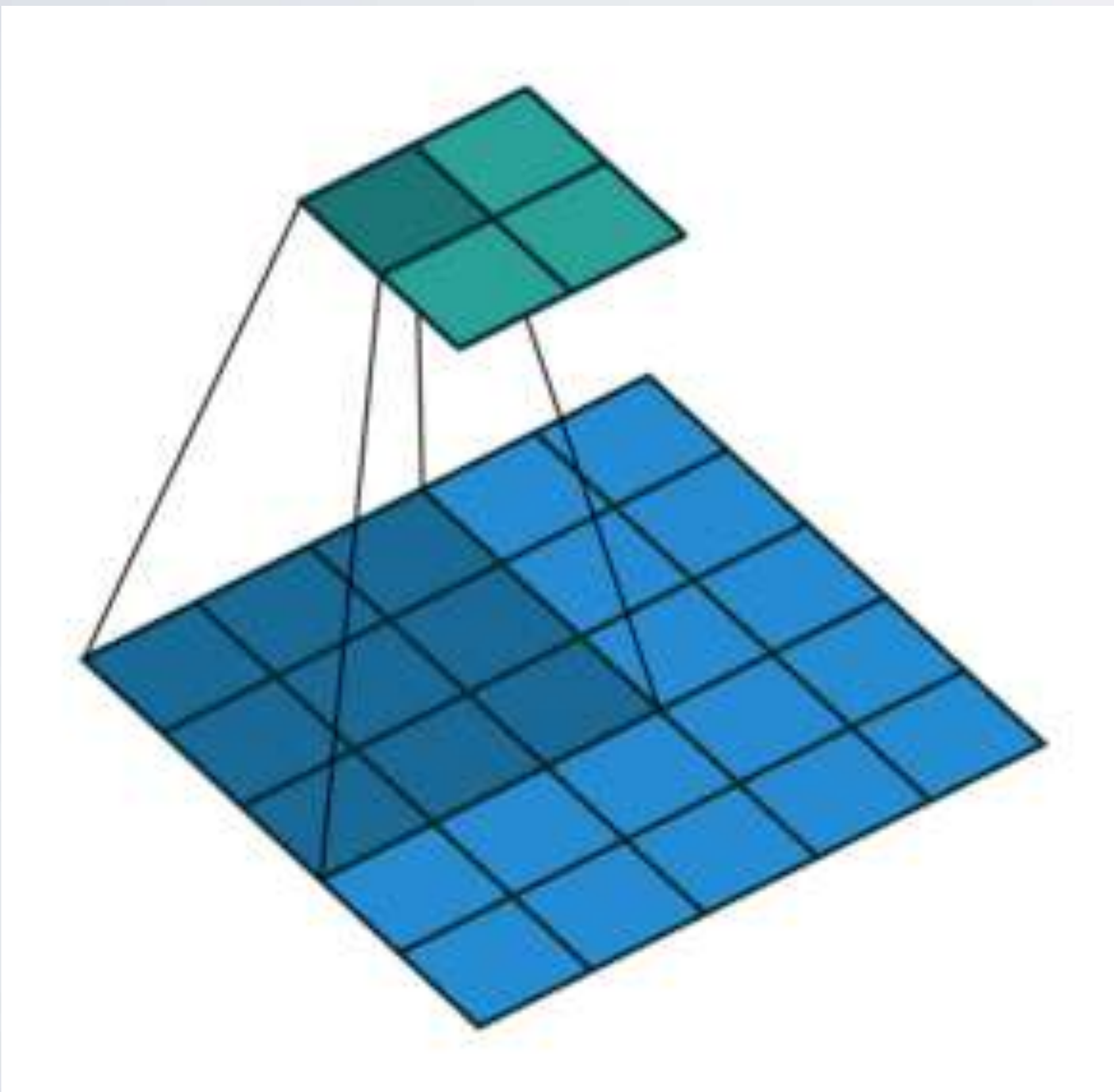


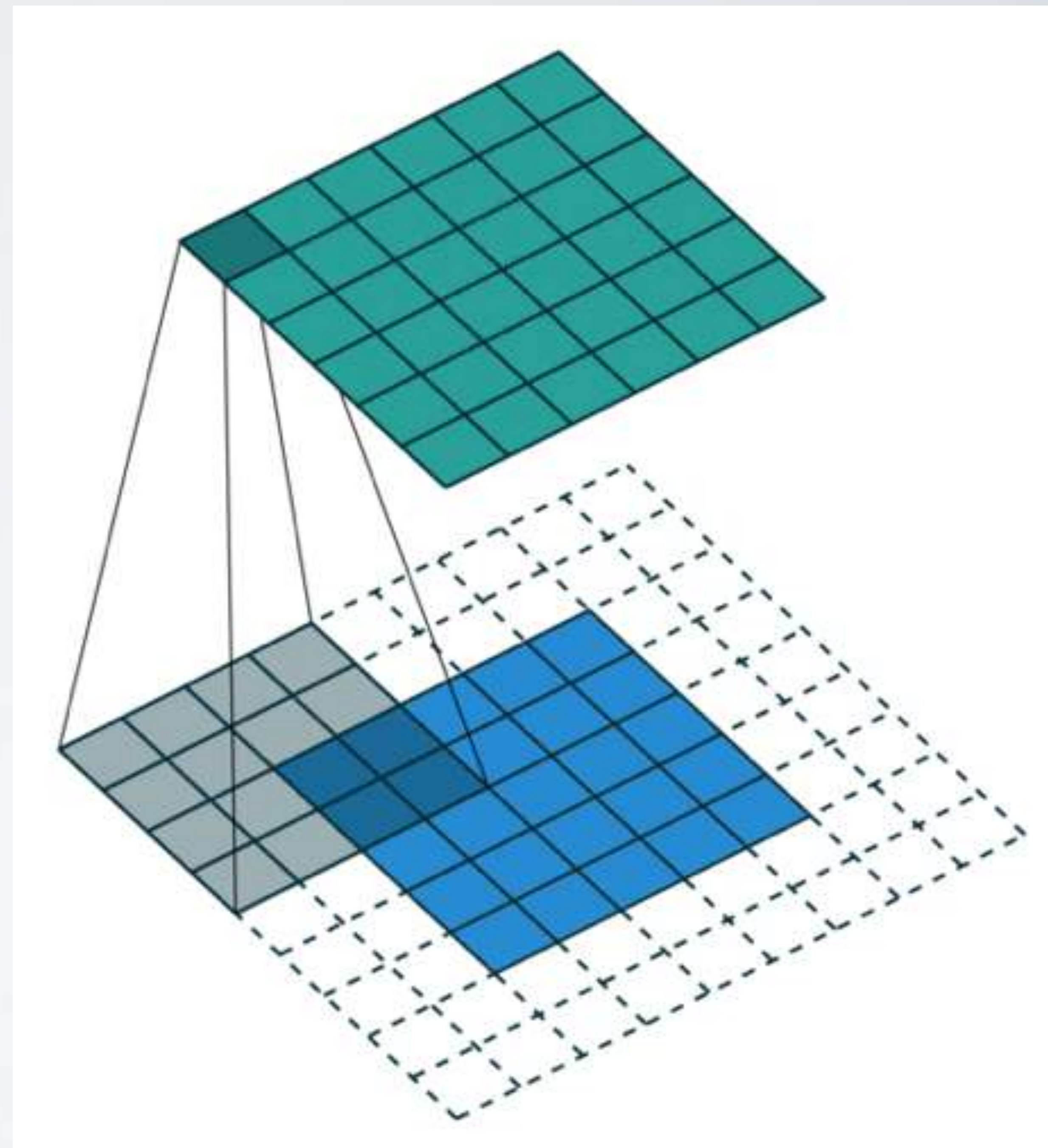
CONVOLUTIONS…





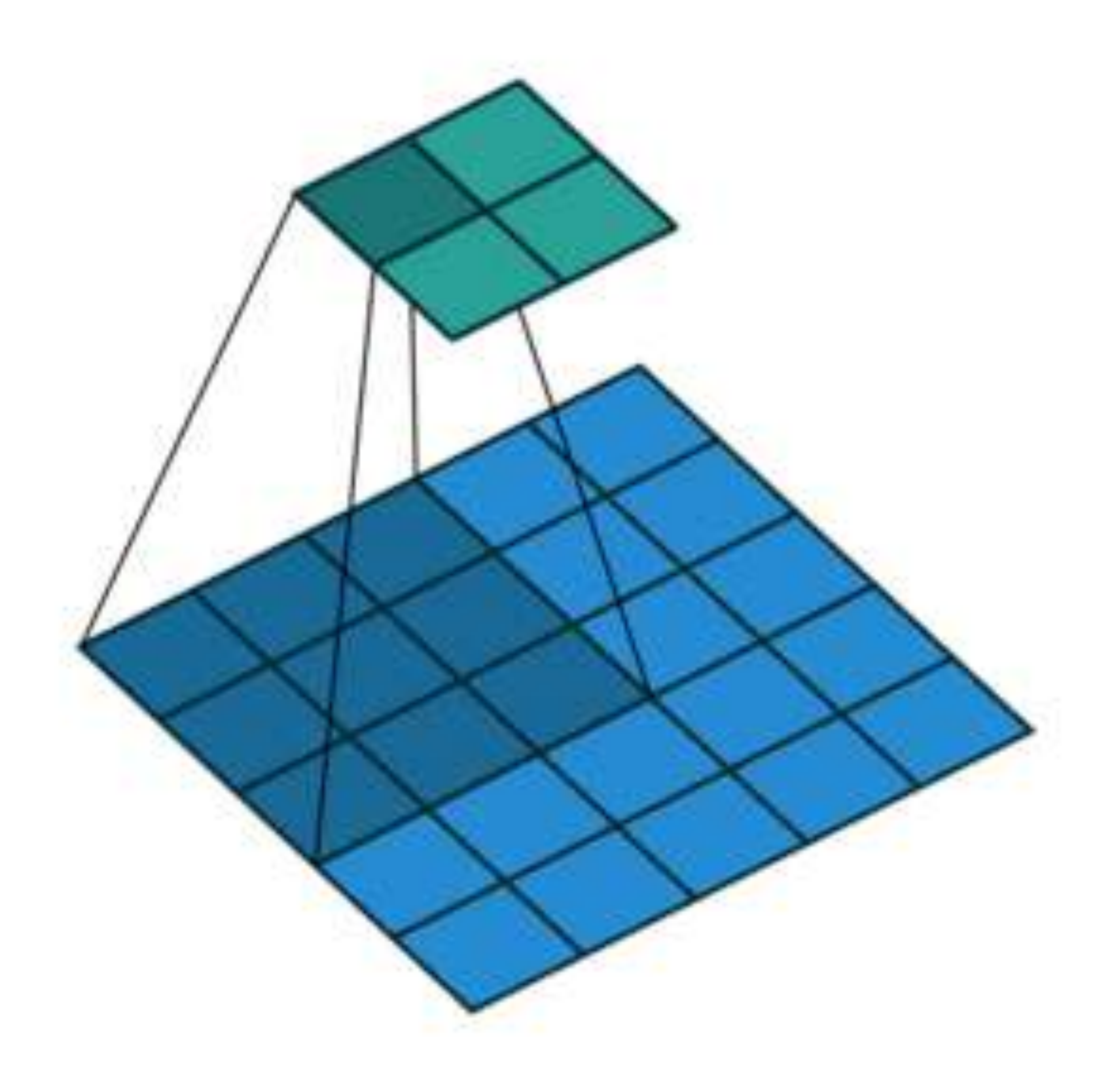
CONVOLUTIONS…

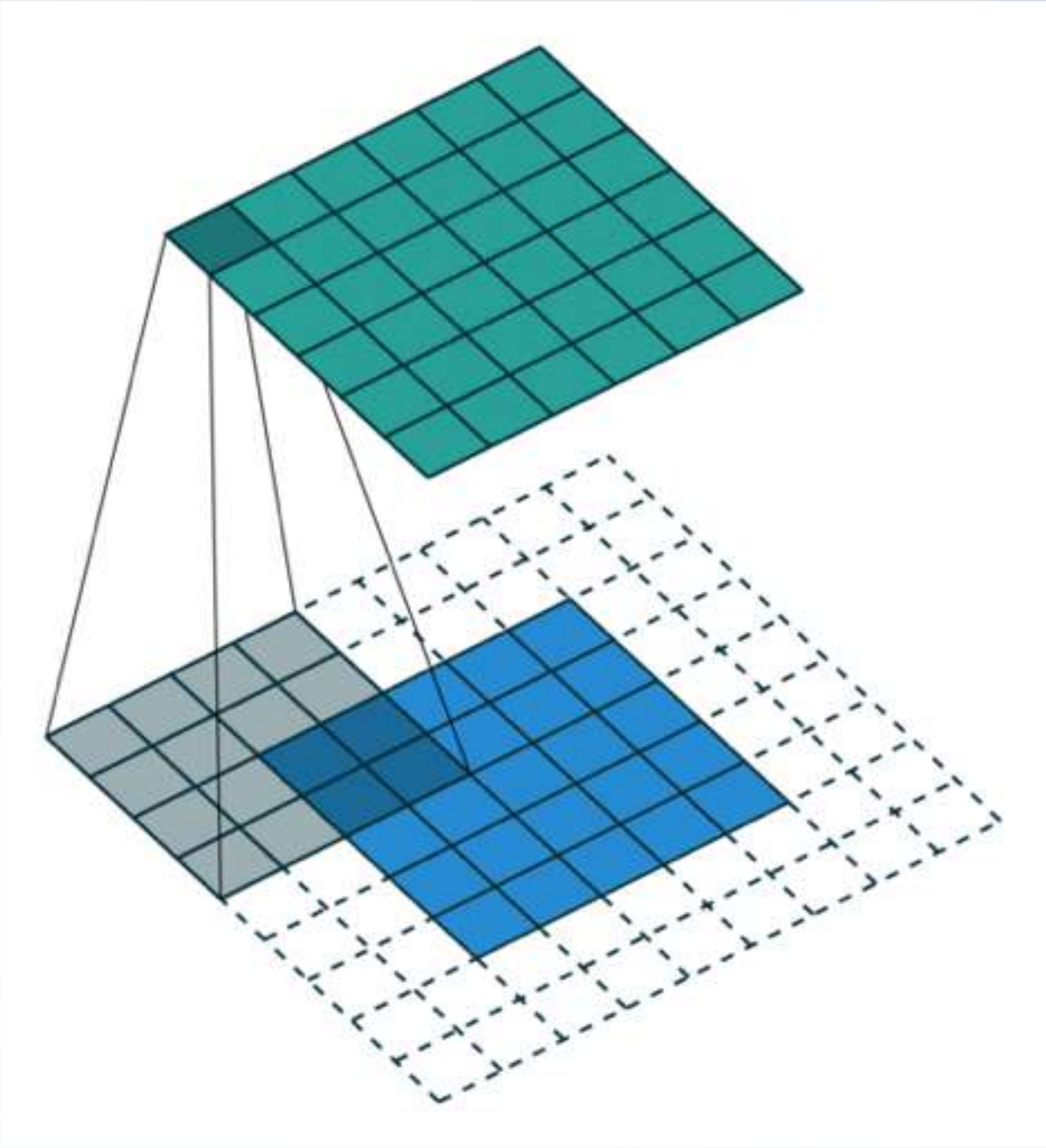




CONVOLUTIONS…

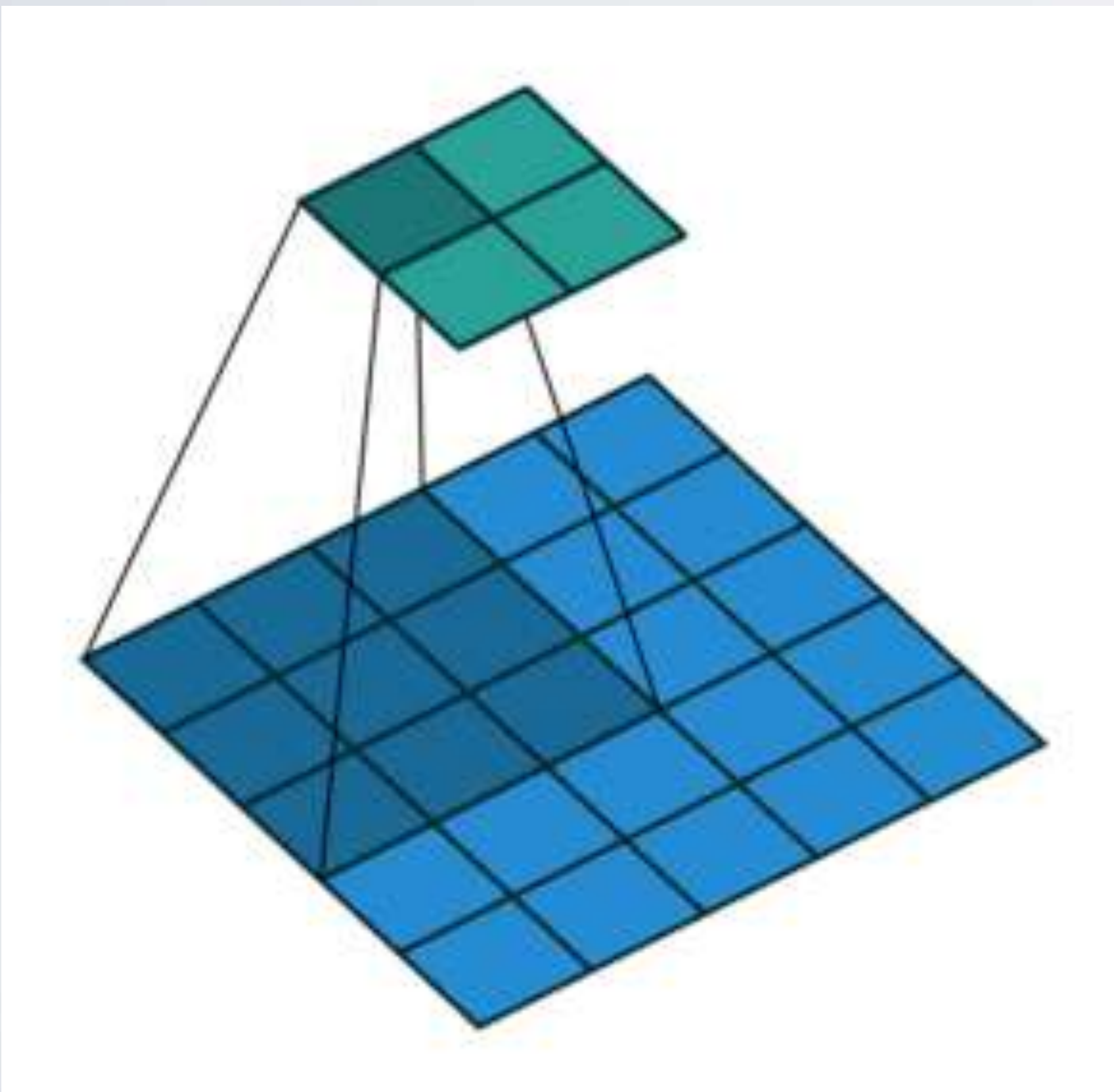
$i = 5$ and $k = 3$, $s = 2$ and $p = 0$.



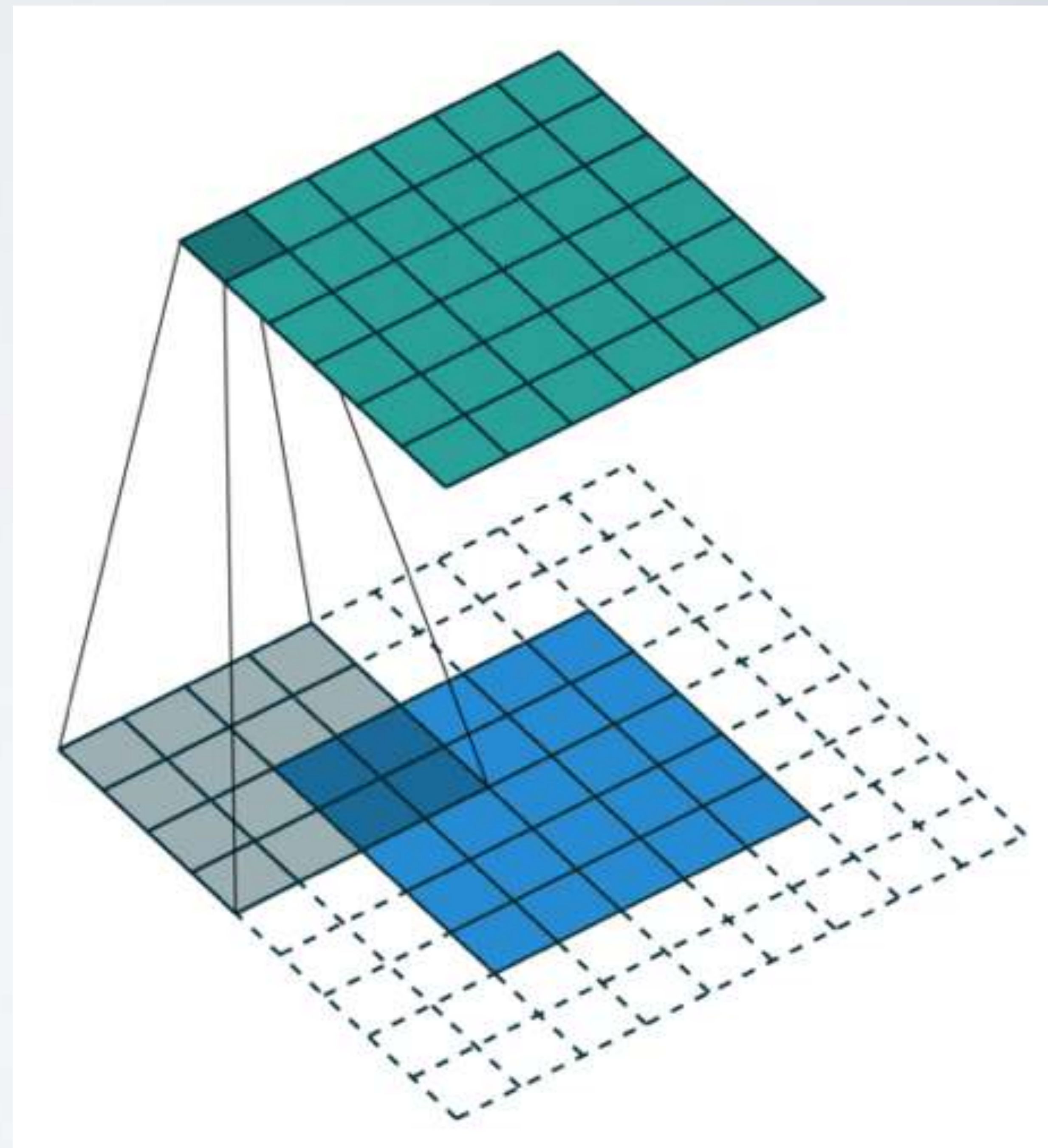


CONVOLUTIONS…

$i = 5$ and $k = 3, s = 2$ and $p = 0$.

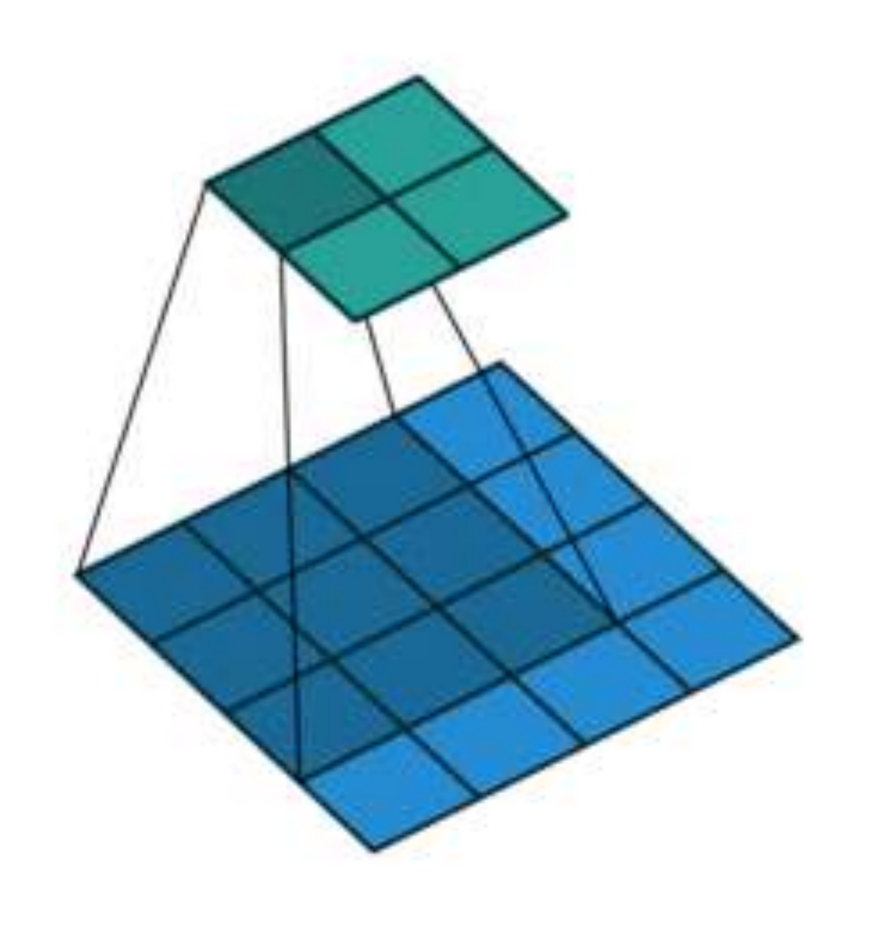


$i = 5$ and $k = 4, s = 1$ and $p = 2$.

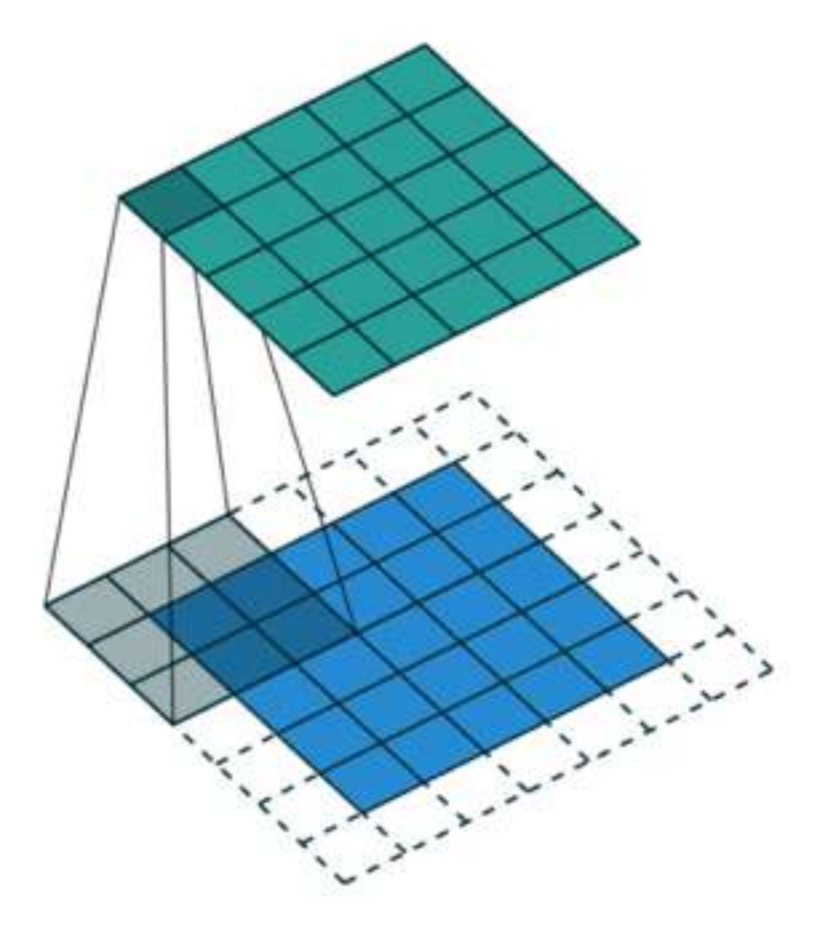


PADDING: SPECIAL CASES

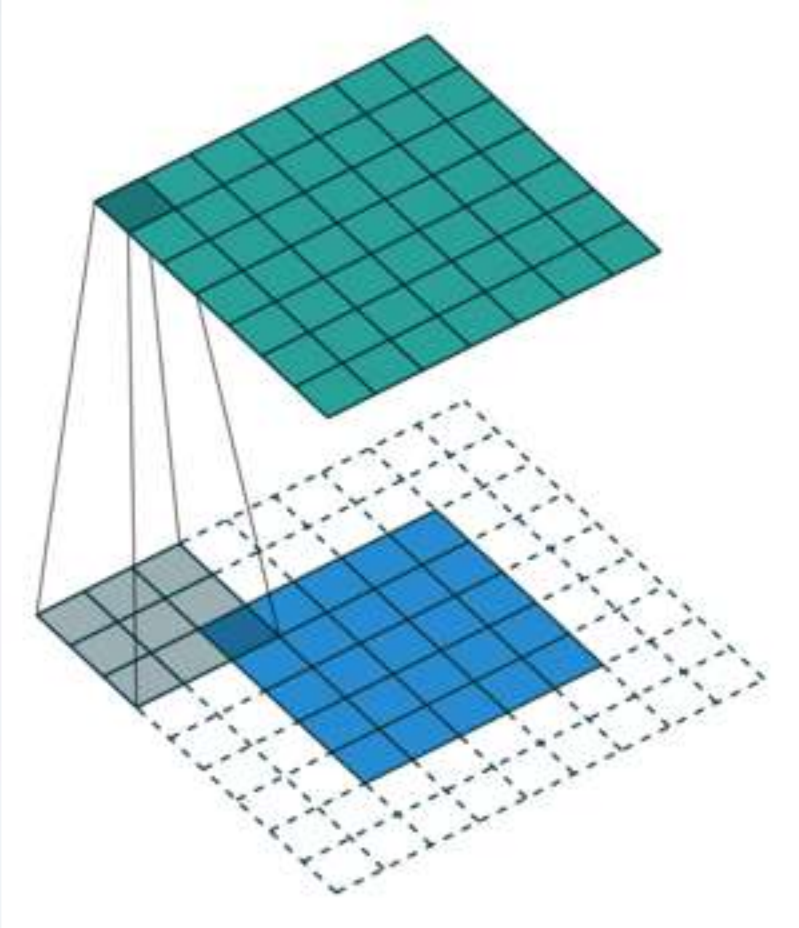
valid convolution
no padding



same convolution
half (or same) padding



full convolution
full padding

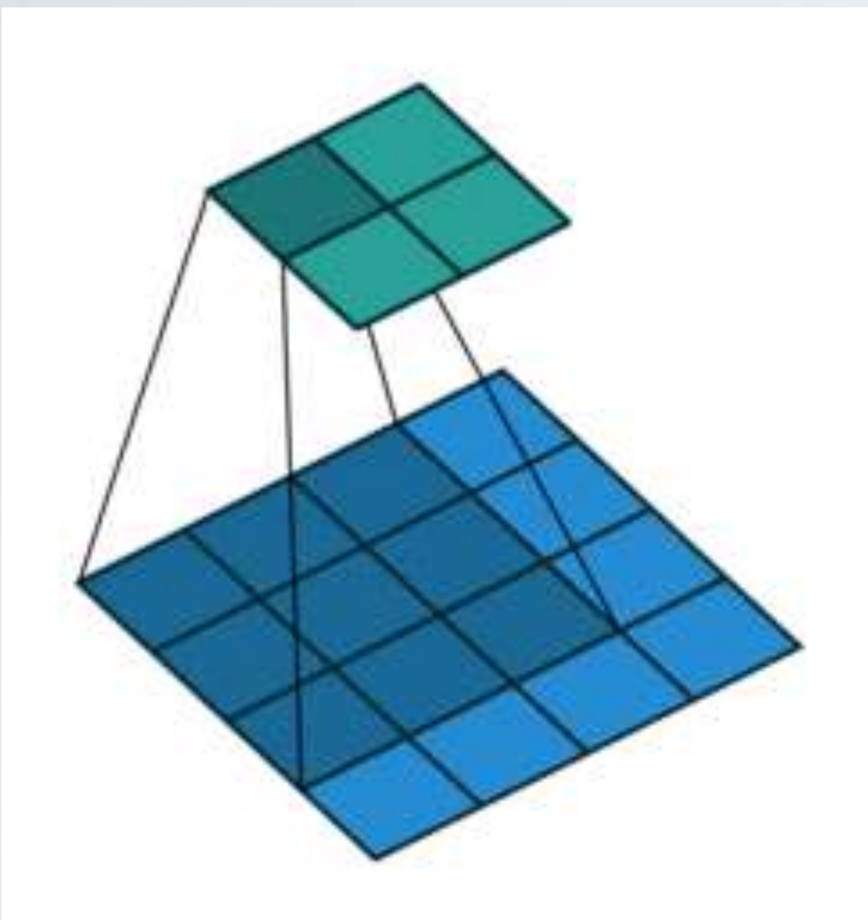


output dimension = input dimension

PADDING: SPECIAL CASES

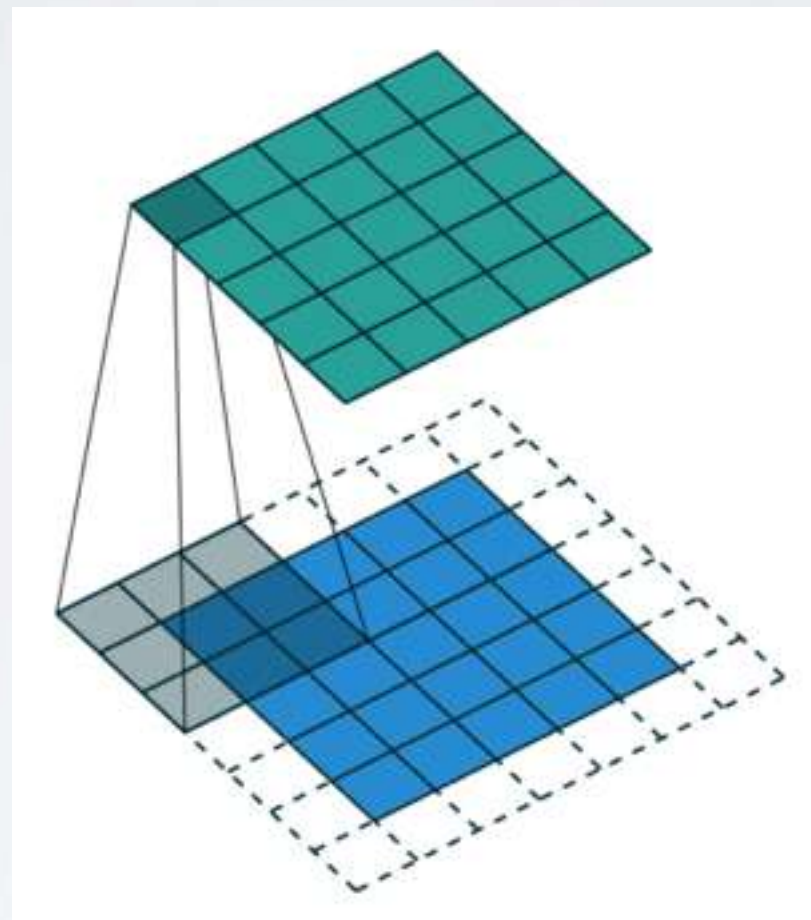
valid convolution

no padding



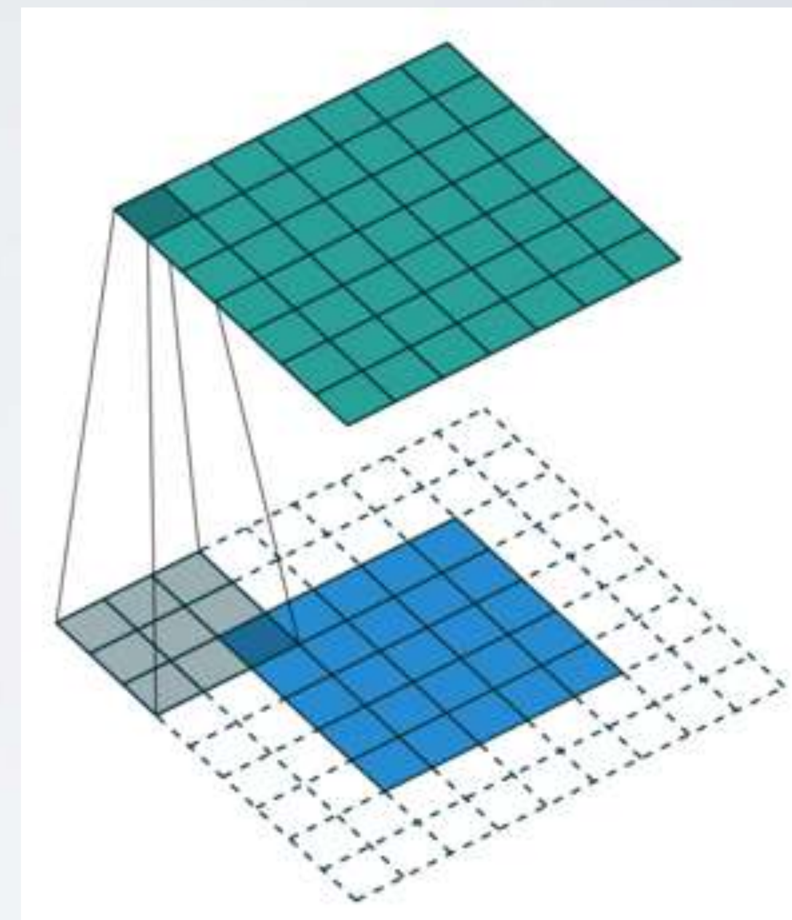
same convolution

half (or same) padding



full convolution

full padding



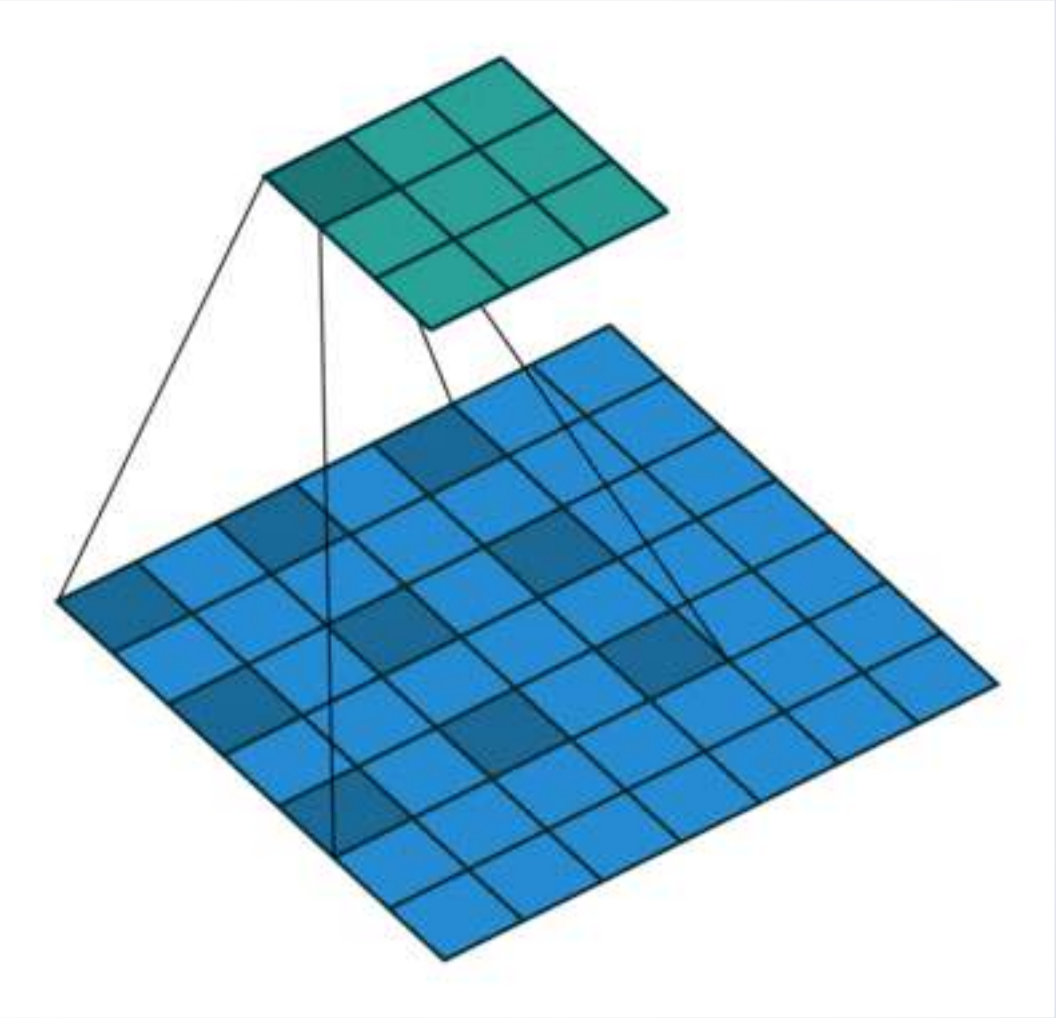
output dimension = input dimension

DILATED CONVOLUTIONS

- Dilated convolutions "inflate" the kernel, inserting spaces between the kernel elements.
  - Dilation "rate" is controlled by an additional hyperparameter d.
  - Usually d - 1 spaces inserted between kernel elements such that $d = 1$ corresponds to a regular convolution.
  - A kernel of size k dilated by a factor d has an effective size:

$$\hat{k} = k + (k - 1)(d - 1).$$

$i = 7, k = 3, d = 2, s = 1$ and $p = 0$.

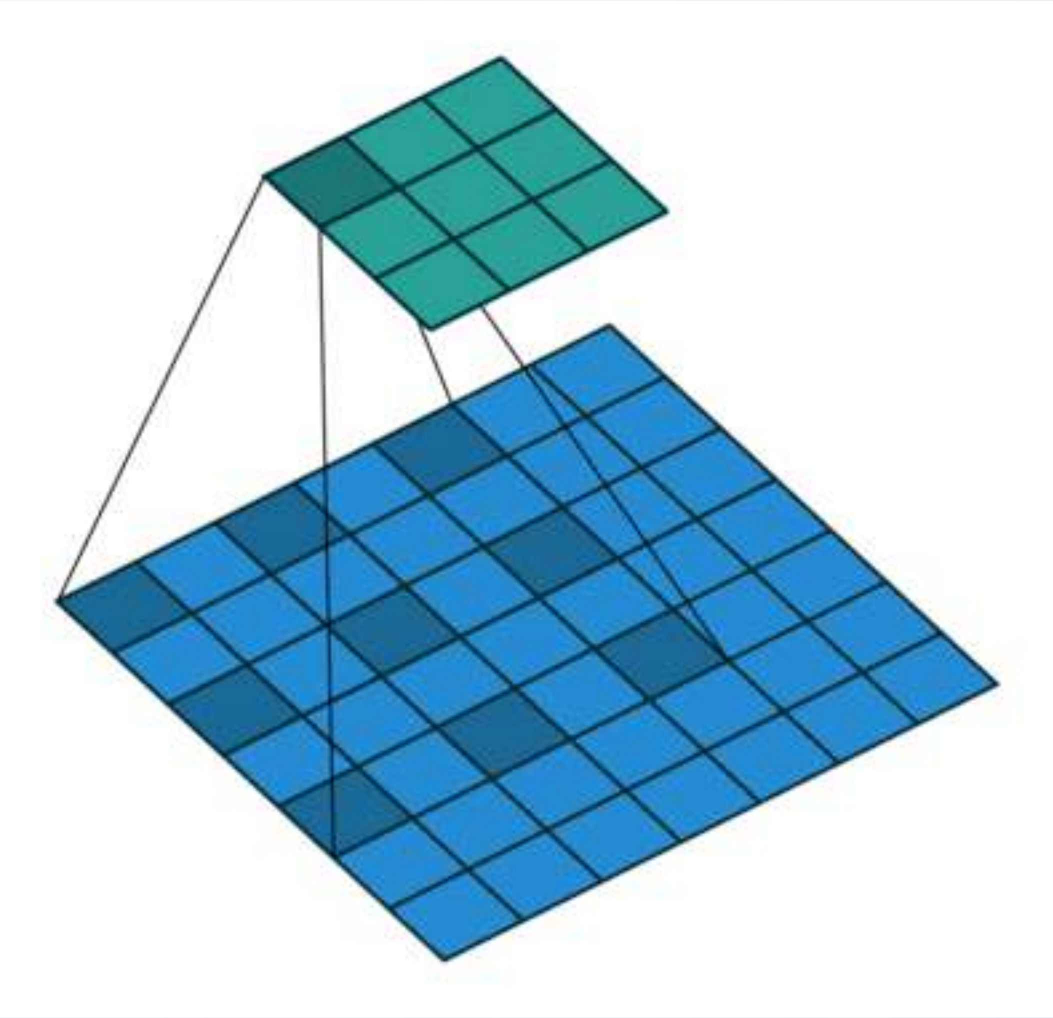


DILATED CONVOLUTIONS

- Dilated convolutions "inflate" the kernel, inserting spaces between the kernel elements.
  - Dilation "rate" is controlled by an additional hyperparameter d.
  - Usually d - 1 spaces inserted between kernel elements such that $d = 1$ corresponds to a regular convolution.
  - A kernel of size k dilated by a factor d has an effective size:

$$\hat{k} = k + (k-1)(d-1).$$

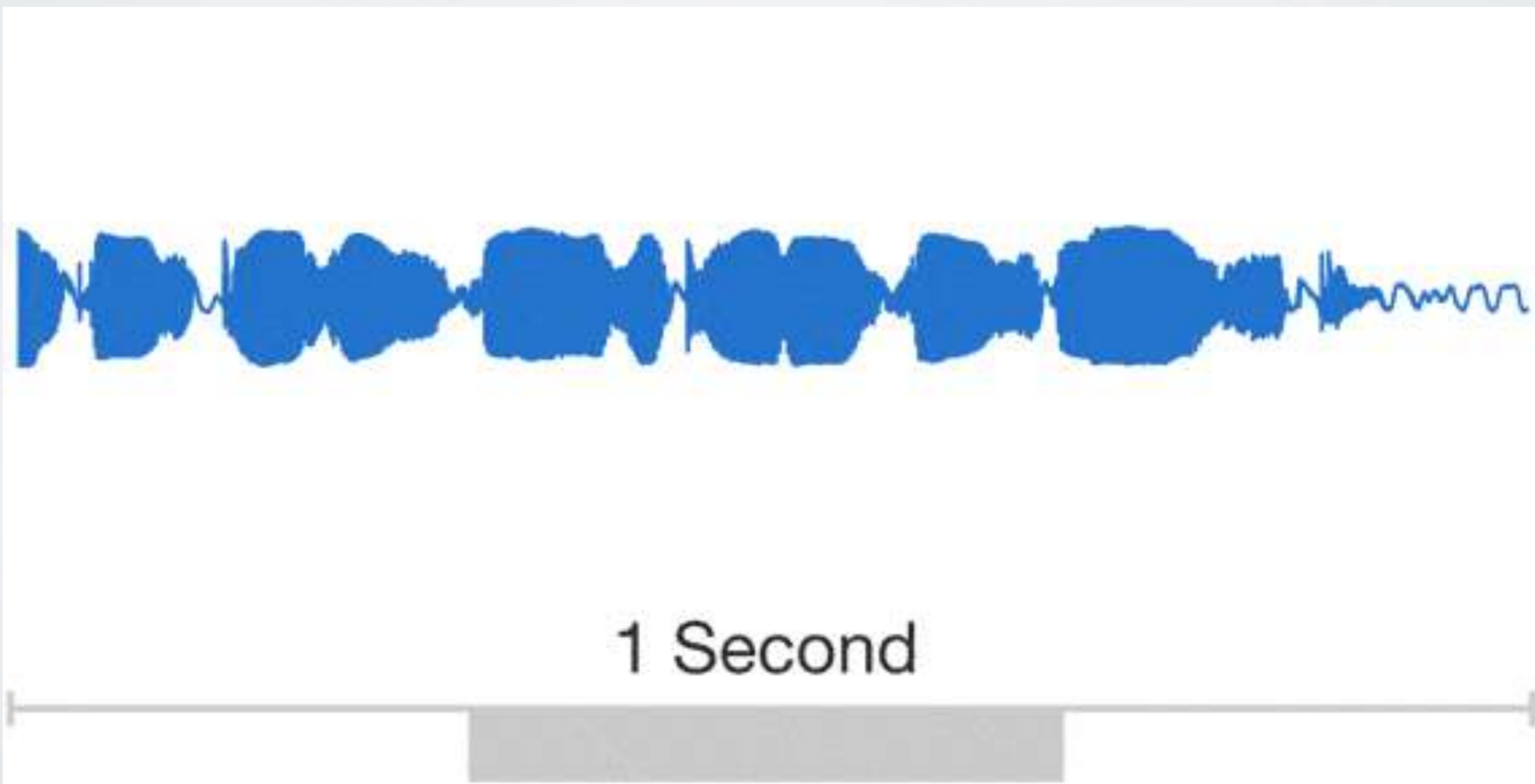
$i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$.



DILATED CONVOLUTIONS: WAVENET

(Van den Oord et al., 2016, arXiv: 1609.03499)

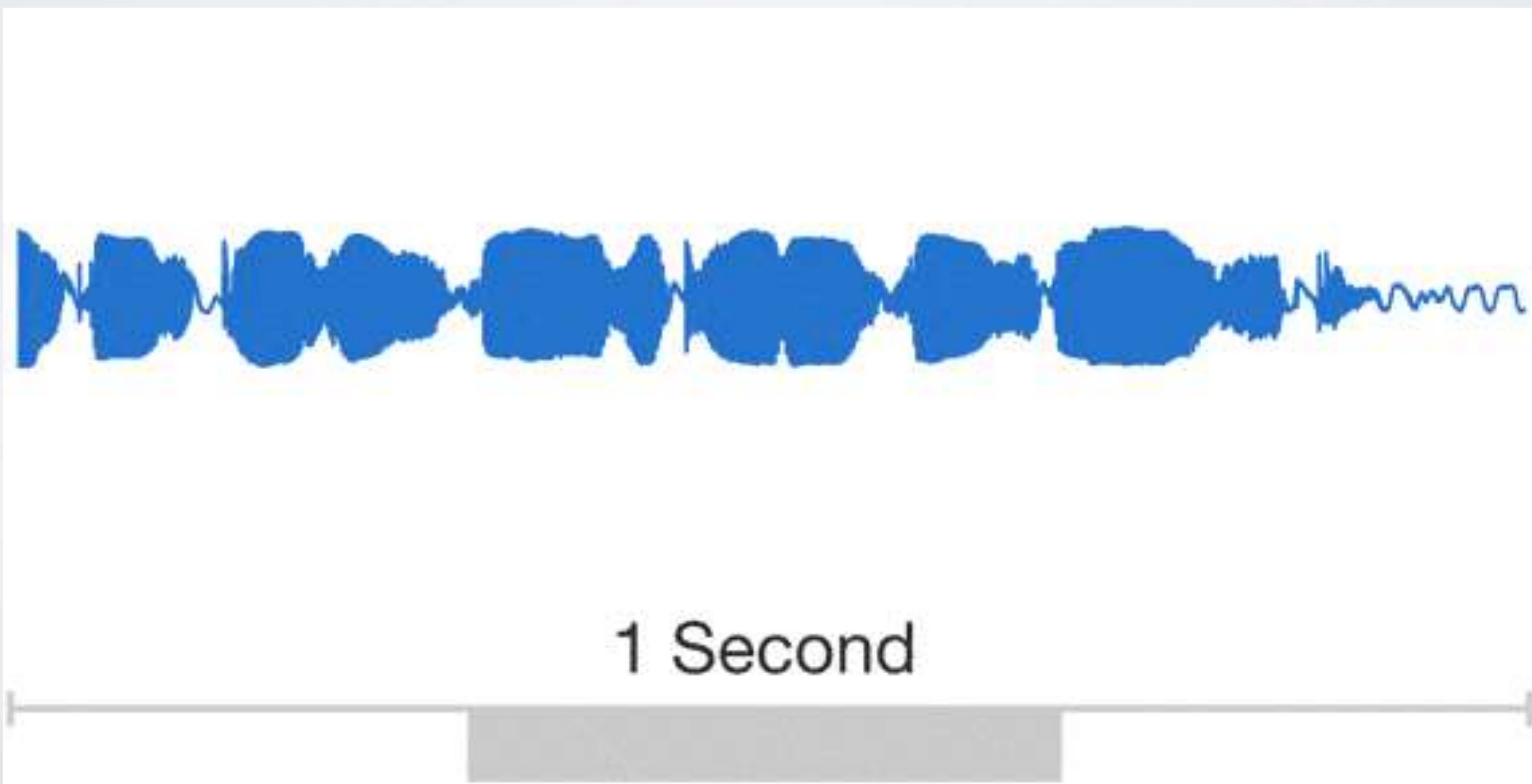
- WaveNet: Dilated CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: WAVENET

(Van den Oord et al., 2016, arXiv: 1609.03499)

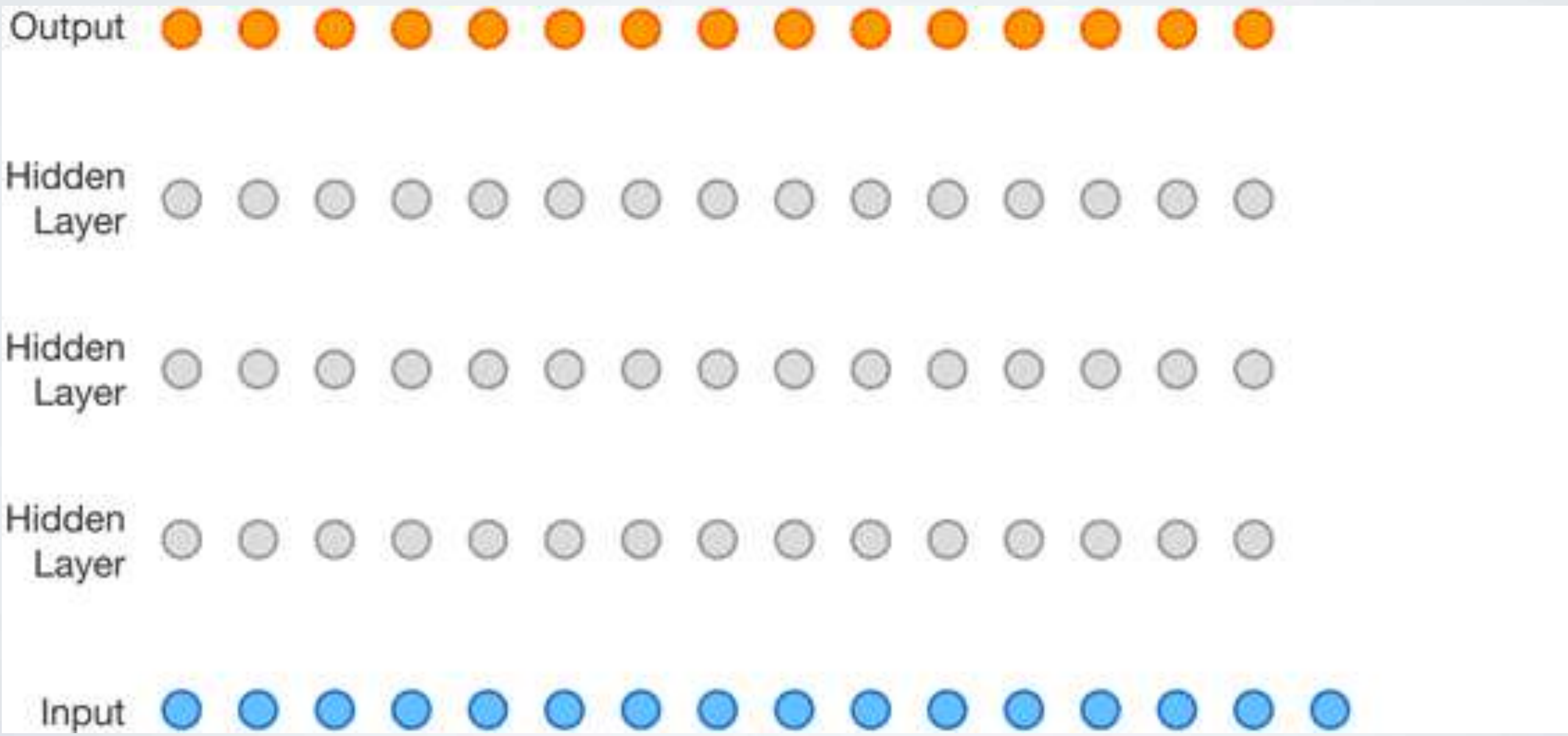
- WaveNet: Dilated CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: WAVENET

(Van den Oord et al., 2016, arXiv: 1609.03499)

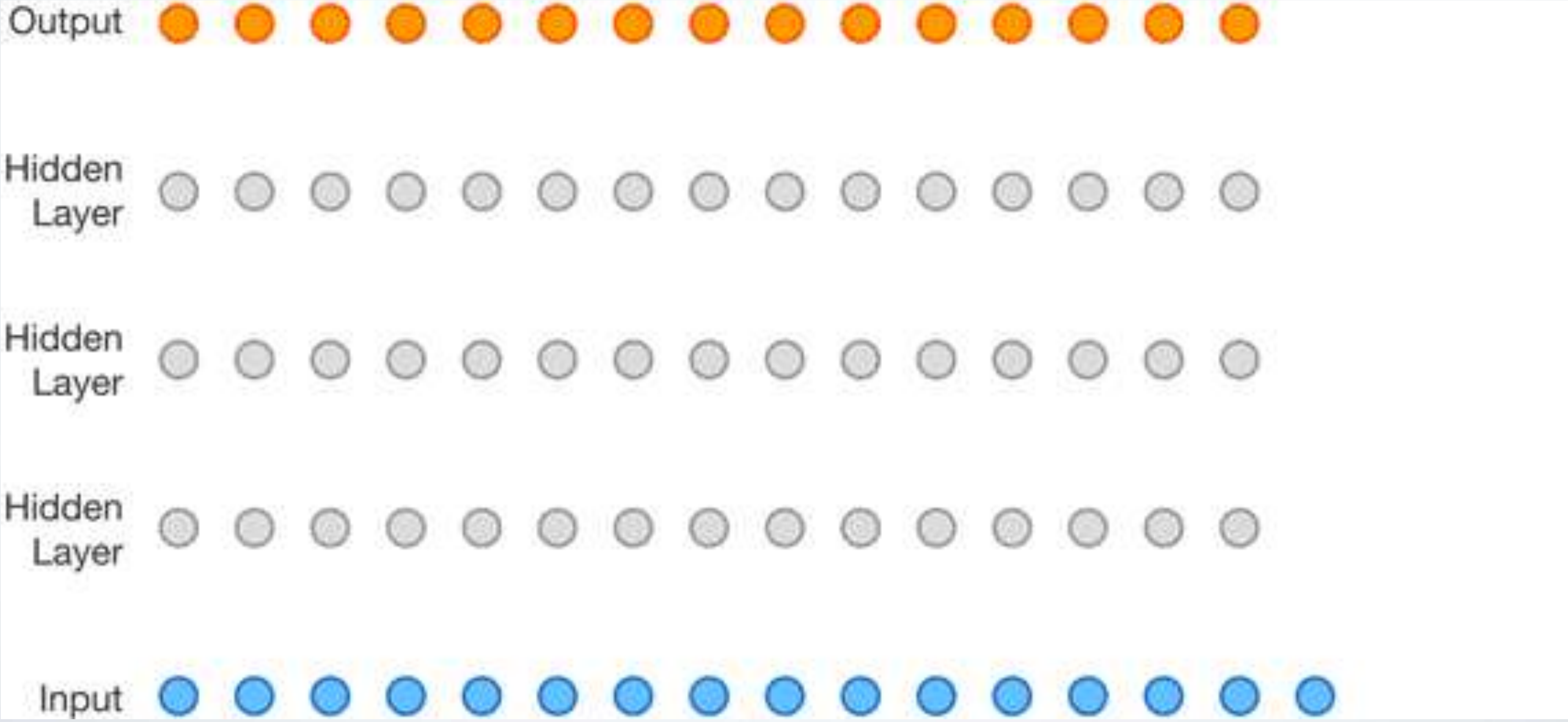
- WaveNet: dilated causal CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: WAVENET

(Van den Oord et al., 2016, arXiv: 1609.03499)

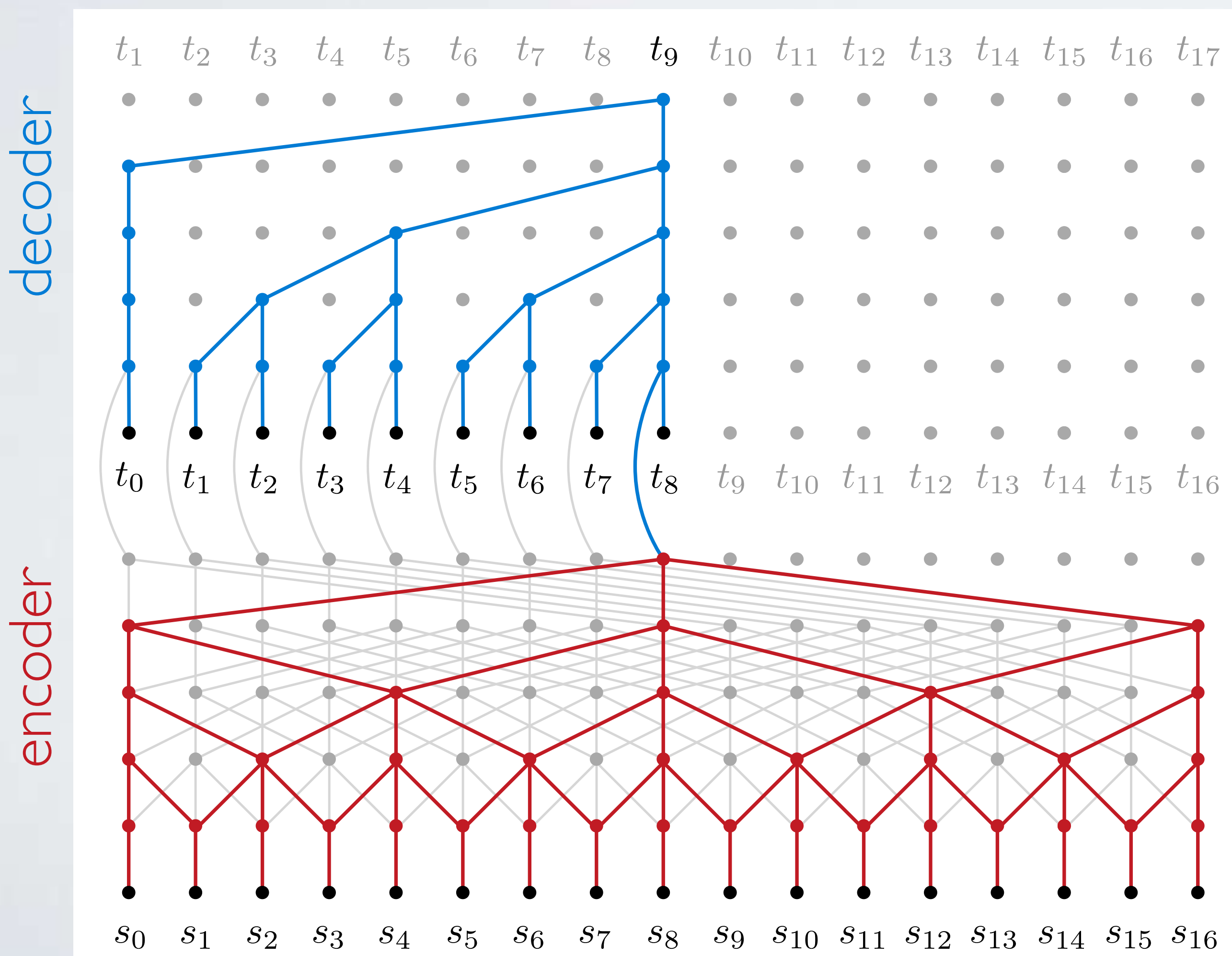
- WaveNet: dilated causal CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: BYTENET

(Kalchbrenner et al., 2016, arXiv: 1610.10099)

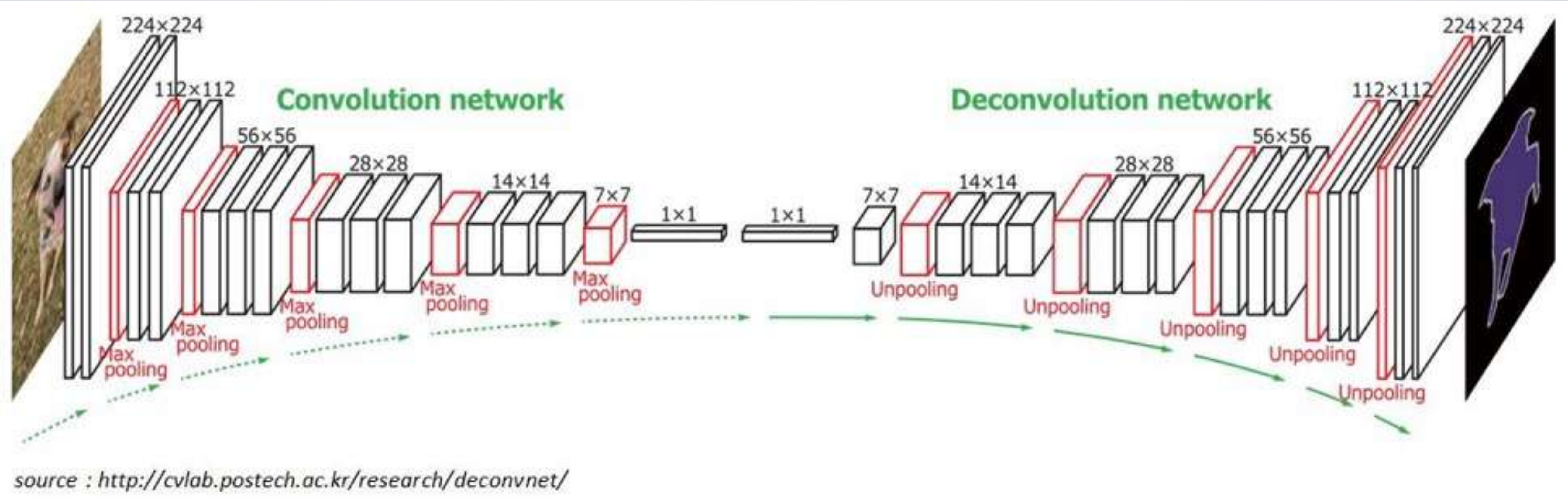
- ByteNet: dilated CNN for machine translation.
  - Uses an encoder-decoder structure with a causal convolution in the decoder.
  - State-of-the-art, computationally efficient machine translation



At the same time, around 3000 demonstrators attempted to reach the official residency of Prime Minister Nawaz Sharif.
Gleichzeitig versuchten rund 3000 Demonstranten, zur Residenz von Premierminister Nawaz Sharif zu gelangen.
Gleichzeitig haben etwa 3000 Demonstranten versucht, die offizielle Residenz des Premierministers Nawaz Sharif zu erreichen.
Just try it: Laura, Lena, Lisa, Marie, Bettina, Emma and manager Lisa Neitzel (from left to right) are looking forward to new members.
Einfach ausprobieren: Laura, Lena, Lisa, Marie, Bettina, Emma und Leiterin Lisa Neitzel (von links) freuen sich auf Mitstreiter.
Probieren Sie es aus: Laura, Lena, Lisa, Marie, Bettina, Emma und Manager Lisa Neitzel (von links nach rechts) freuen sich auf neue Mitglieder.
He could have said, "I love you," but it was too soon for that.
Er hätte sagen können "ich liebe dich", aber dafür war es noch zu früh.
Er hätte sagen können: "I love you", aber es war zu früh.

Table 4: Raw output translations generated from the ByteNet that highlight interesting reordering and transliteration phenomena. For each group, the first row is the English source, the second row is the ground truth German target, and the third row is the ByteNet translation.

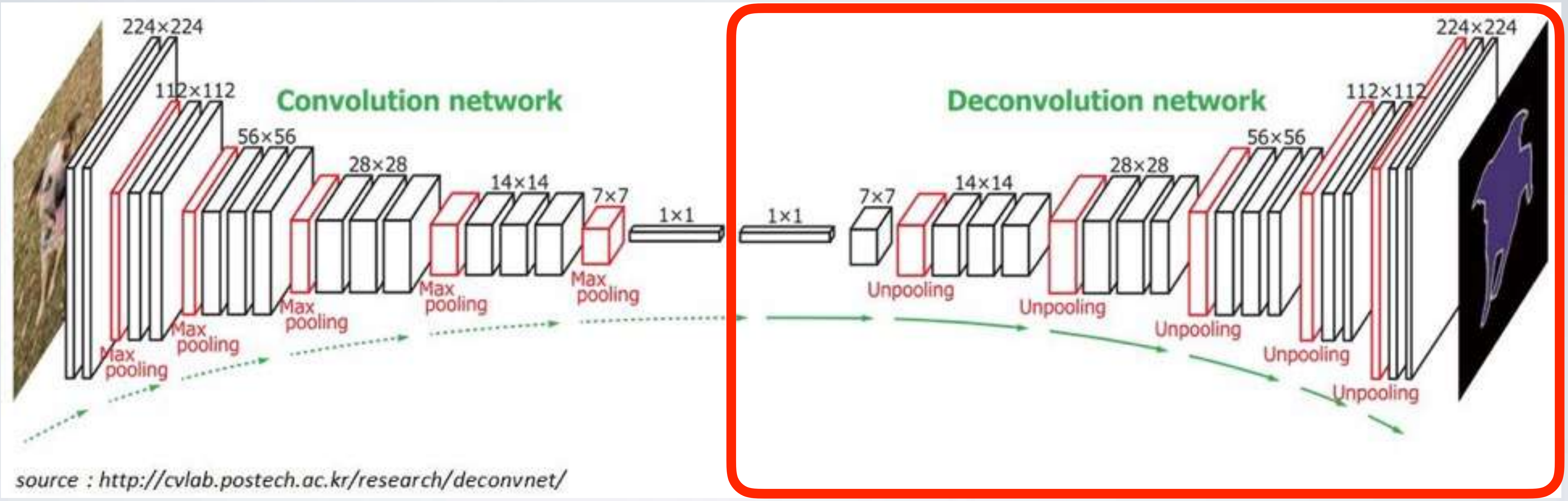
(DE)CONVOLUTIONS FOR STRUCTURED OUTPUT



semantic segmentation (example taken from https://nrupatunga.github.io/convolution-1/)

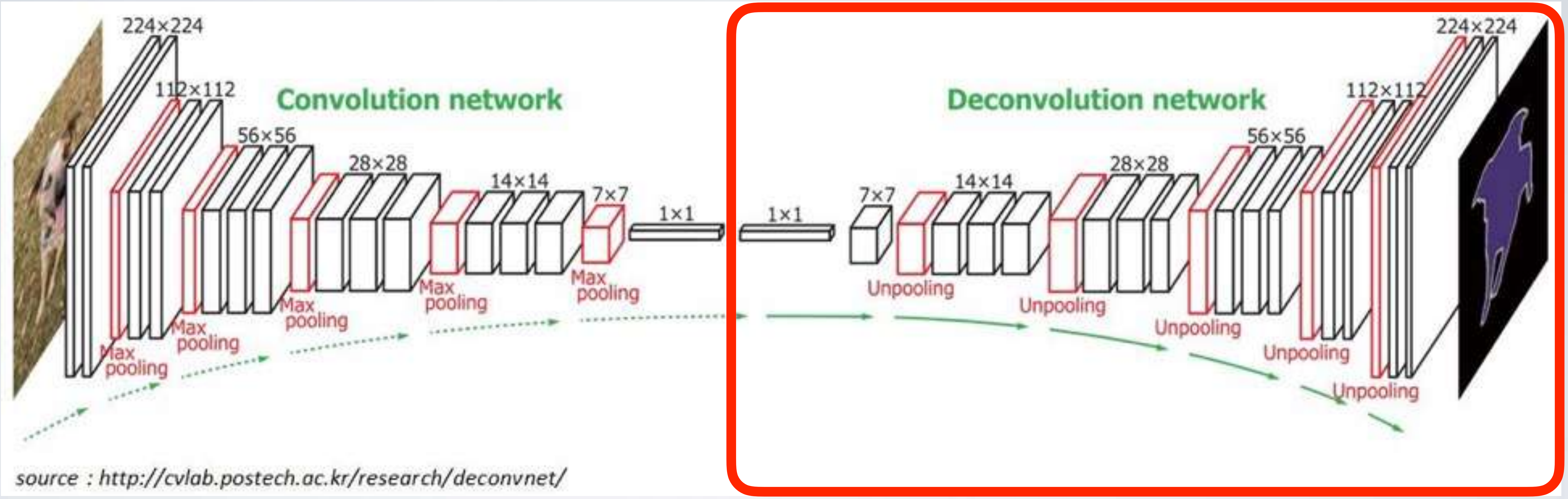
- We can use convolutions to build structure into high-dimensional outputs.
- Useful in applications such as:
  - ➡ semantic segmentation (above)
  - ➡ image generation

(DE)CONVOLUTIONS FOR STRUCTURED OUTPUT



How do we do this part?

(DE)CONVOLUTIONS FOR STRUCTURED OUTPUT

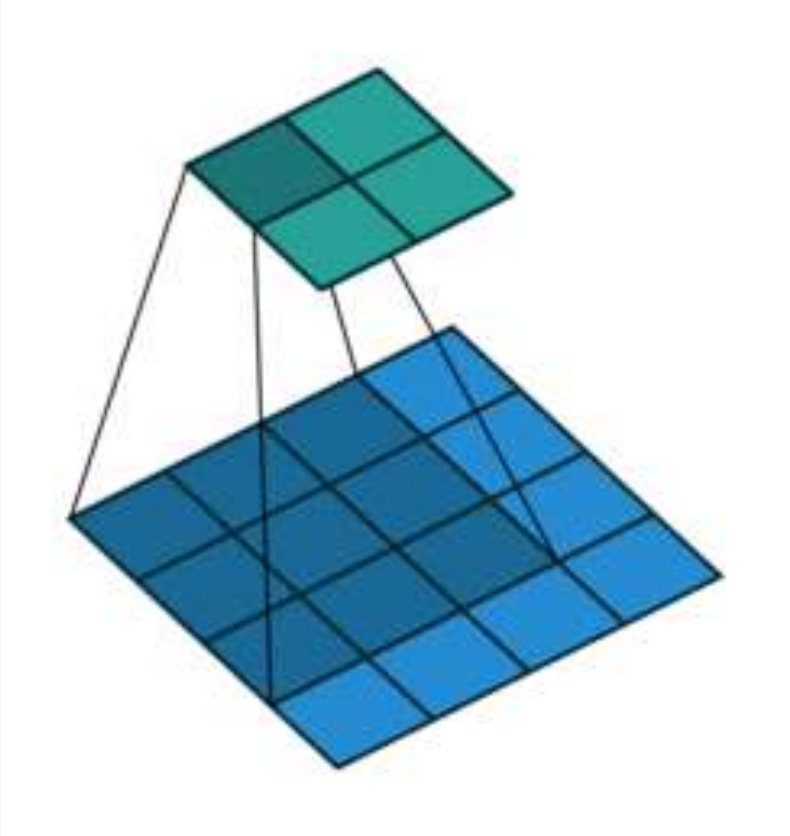


How do we do this part?

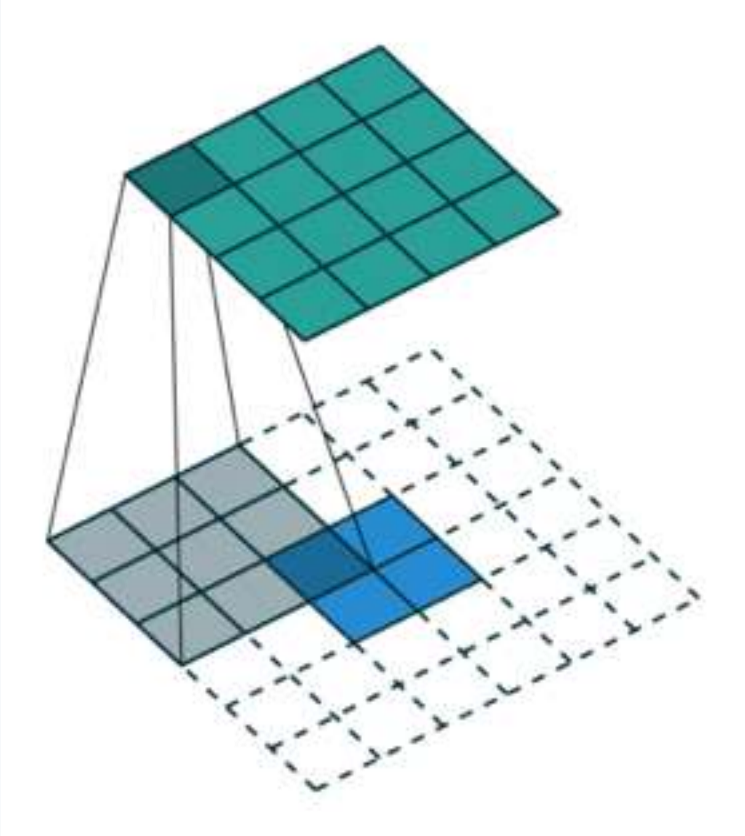
- Option 1: Transposed (a.k.a. fractionally strided) convolution.
- Option 2: Upsampling (e.g. nearest-neighbour or bilinear) + convolution.

TRANSPOSE CONVOLUTION

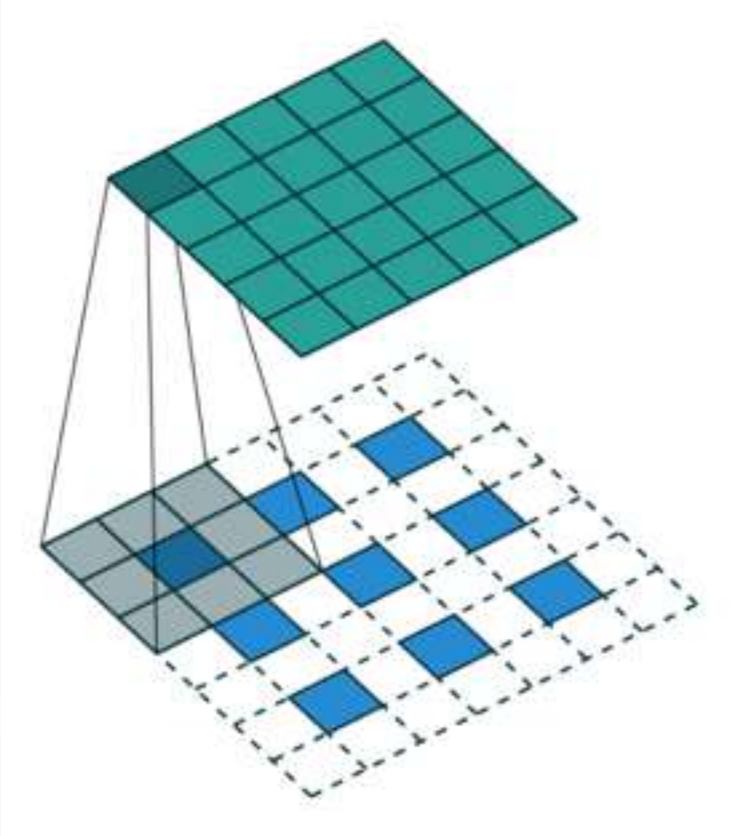
- Transposed convolution is the operation done in the backprop pass through a convolution.
- It's the natural "inverse" of the convolution operation.



Standard Convolution
$i = 5, k = 3$ and $s = 2$



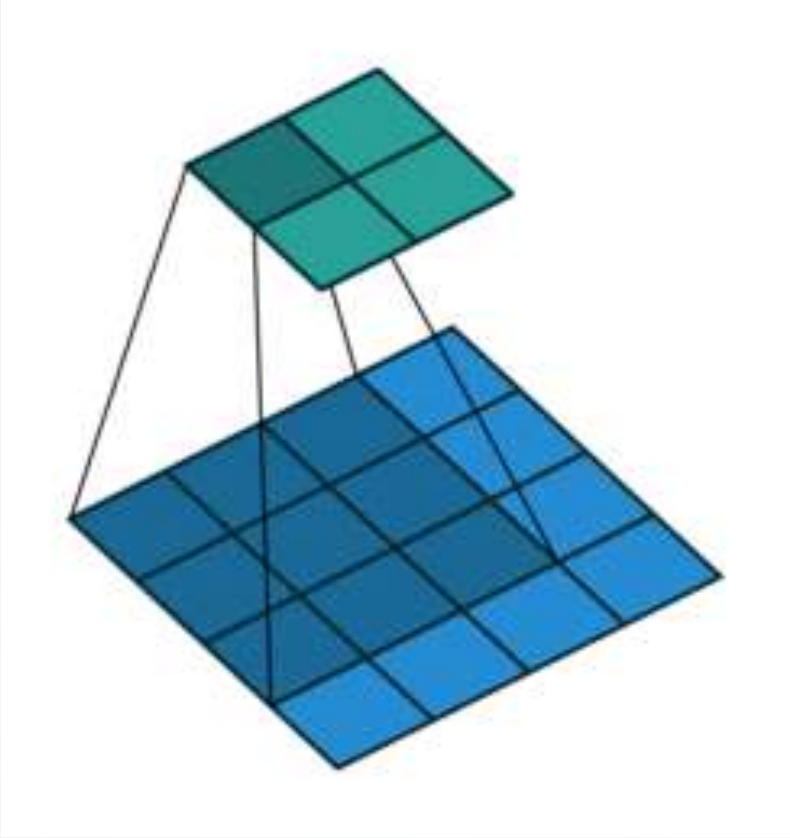
Equivalent to transpose conv. but less efficient (due to zero-padding)



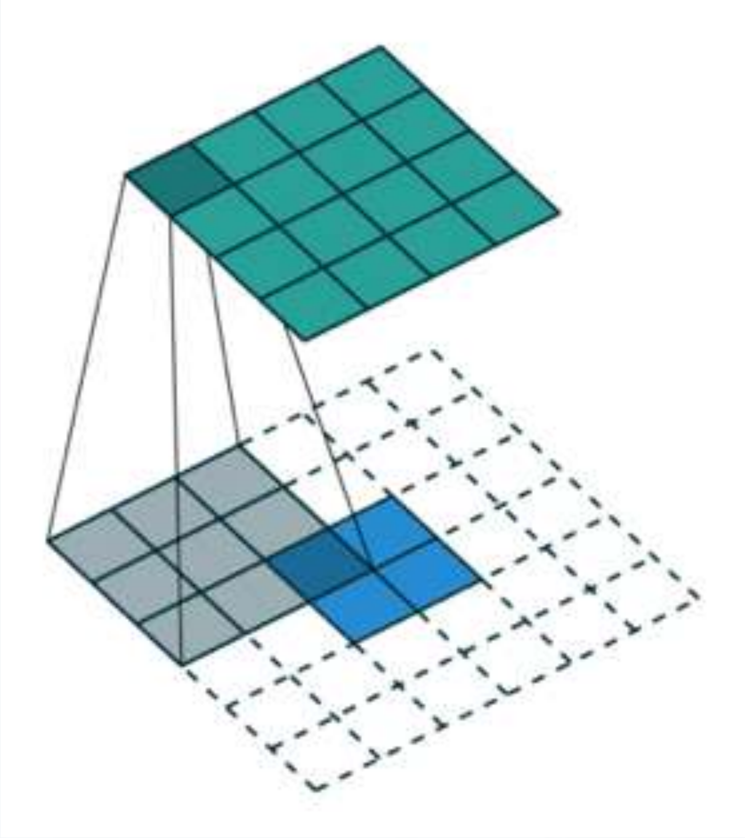
i = 5, k = 3, s = 2 and p= 1

TRANSPOSE CONVOLUTION

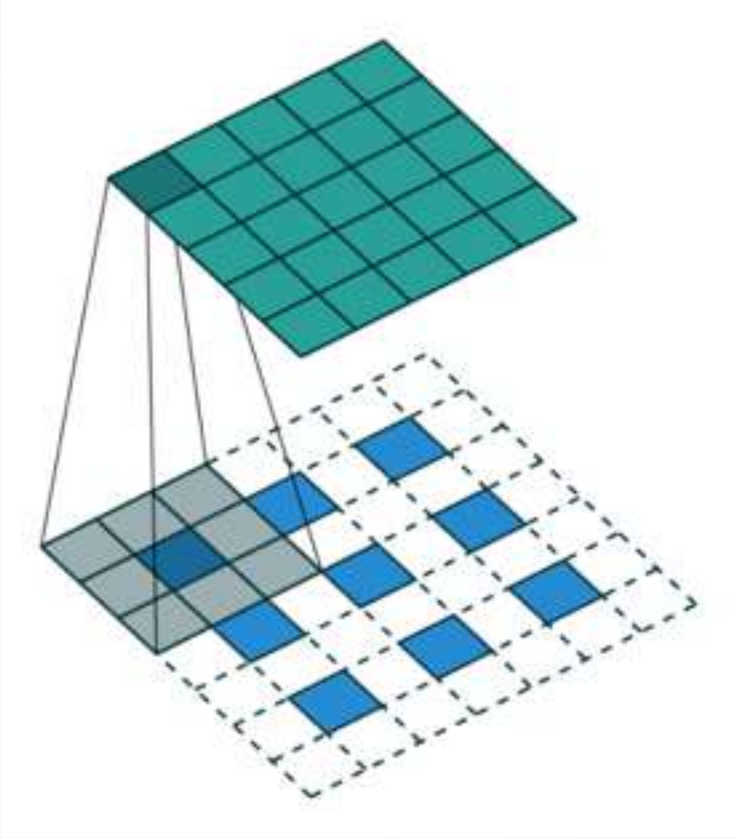
- Transposed convolution is the operation done in the backprop pass through a convolution.
- It's the natural "inverse" of the convolution operation.



Standard Convolution
$i = 5, k = 3$ and $s = 2$



Equivalent to transpose conv. but less efficient (due to zero-padding)



i = 5, k = 3, s = 2 and p= 1

INTERPOLATION + CONVOLUTION

- But transpose convolution can cause artifacts.
  - see http://distill.pub/2016/deconv-checkerboard
- Currently, interpolation and convolution is the preferred method for generative convolutional models.

Using transpose convolution:
Heavy checkerboard artifacts.

Using resize-convolution.
No checkerboard artifacts.