Memo: Backpropagation in Convolutional Neural Network

Hiroshi Kuwajima

13-03-2014 Created
14-08-2014 Revised

Note

■ Purpose

The purpose of this memo is trying to understand and remind the backpropagation algorithm in Convolutional Neural Network based on a discussion with Prof. Masayuki Tanaka.

■ Table of Contents

In this memo, backpropagation algorithms in different neural networks are explained in the following order.

■ Notation

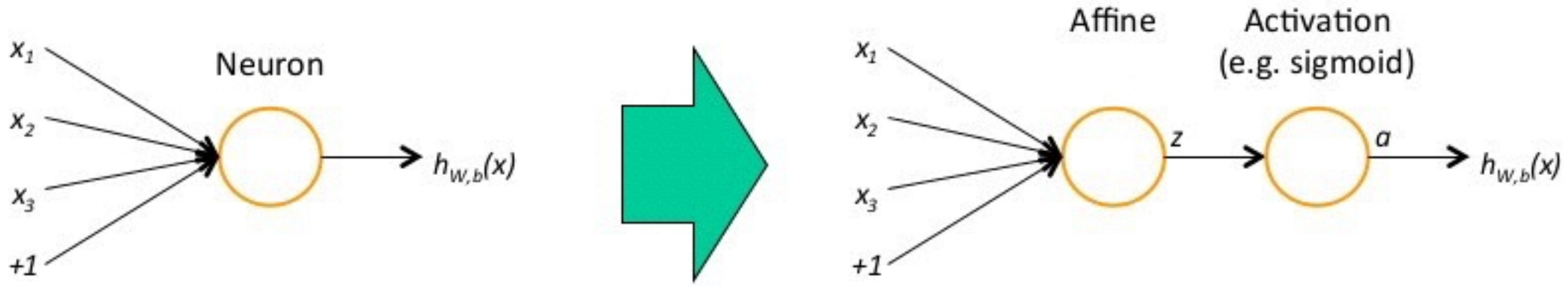
This memo follows the notation in UFLDL tutorial (http://ufldl.stanford.edu/tutorial)

Neural Network as a Composite Function

A neural network is decomposed into a composite function where each function element corresponds to a differentiable operation.

- Single neuron (the simplest neural network) example

  A single neuron is decomposed into a composite function of an affine function element parameterized by W and b and an activation function element f which we choose to be the sigmoid function.



Standard network representation Decomposition Composite function representation

$$h_{W,b}(x) = f\left(W^T x + b\right) = \text{sigmoid}\left(\text{affine}_{W,b}(x)\right) = \left(\text{sigmoid} \circ \text{affine}_{W,b}\right)(x)$$

Derivatives of both affine and sigmoid function elements w.r.t. both inputs and parameters are known. Note that sigmoid function does not have neither parameters nor derivatives parameters.

Sigmoid function is applied element-wise. '•' denotes Hadamard product, or element-wise product.

$$\frac{\partial a}{\partial z} = a \bullet (1 - a) \text{ where } a = h_{w,b}(x) = \text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$$

$$\frac{\partial z}{\partial x} = W,\ \frac{\partial z}{\partial W} = x,\ \frac{\partial z}{\partial b} = I \text{ where } z = \text{affine}_{W,b}(x) = W^T x + b, \text{ and } I \text{ is identity matrix}$$

Chain Rule of Error Signals and Gradients

Error signals are defined as the derivatives of any cost function *J* which we choose to be the square error. Error signals are computed (propagated backward) by the chain rule of derivative and useful for computing the gradient of the cost function.

- Single neuron example

  Suppose we have *m* labeled training examples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$. Square error cost function for each example is as follows. Overall cost function is the summation of cost functions over all examples.

$$J(W,b;x,y) = \frac{1}{2}\left\|y - h_{w,b}(x)\right\|^2$$

  Error signals of the square error cost function for each example are propagated using derivatives of function elements w.r.t. inputs.

$$\delta^{(a)} = \frac{\partial}{\partial a} J(W,b;x,y) = -(y-a)$$

$$\delta^{(z)} = \frac{\partial}{\partial z} J(W,b;x,y) = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z} = \delta^{(a)} \bullet a \bullet (1-a)$$

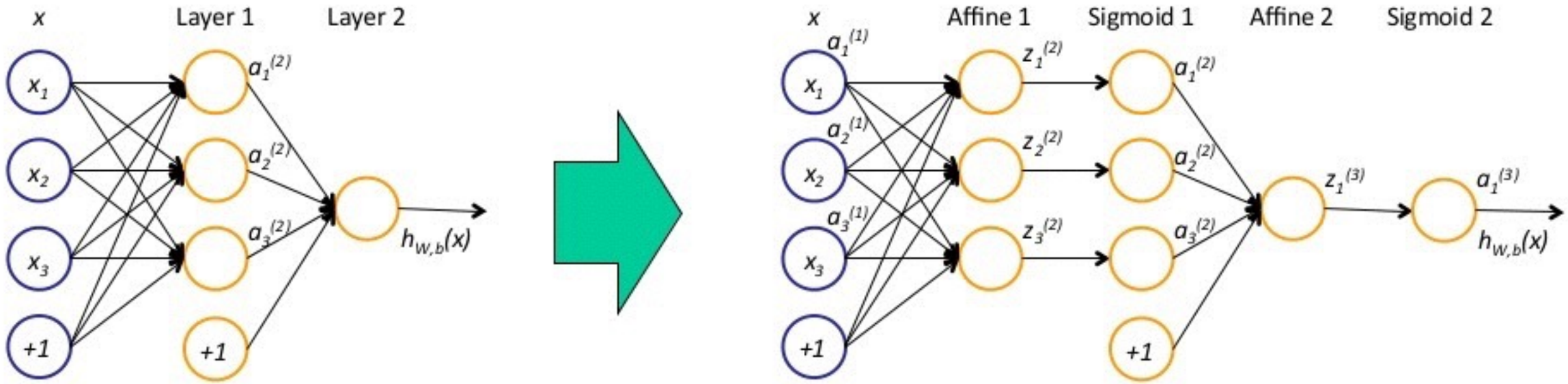
  Gradient of the cost function w.r.t. parameters for each example is computed using error signals and derivatives of function elements w.r.t. parameters. Summing gradients for all examples gets overall gradient.

$$\nabla_W J(W,b;x,y) = \frac{\partial}{\partial W} J(W,b;x,y) = \frac{\partial J}{\partial z}\frac{\partial z}{\partial W} = \delta^{(z)} x^T$$

$$\nabla_b J(W,b;x,y) = \frac{\partial}{\partial b} J(W,b;x,y) = \frac{\partial J}{\partial z}\frac{\partial z}{\partial b} = \delta^{(z)}$$

Decomposition of Multi-Layer Neural Network

- Composite function representation of a multi-layer neural network



Standard network representation Decomposition Composite function representation

$$h_{W,b}(x) = \left(\text{sigmoid} \circ \text{affine}_{W^{(2)},b^{(2)}} \circ \text{sigmoid} \circ \text{affine}_{W^{(1)},b^{(1)}}\right)(x)$$

- Derivatives of function elements w.r.t. inputs and parameters

$$a^{(1)} = x, a^{(l_{\max})} = h_{w,b}(x)$$

$$\frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} = a^{(l+1)} \bullet \left(1 - a^{(l+1)}\right) \text{ where } a^{(l+1)} = \text{sigmoid}\left(z^{(l+1)}\right) = \frac{1}{1+\exp\left(-z^{(l+1)}\right)}$$

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = W^{(l)}, \frac{\partial z^{(l+1)}}{\partial W^{(l)}} = a^{(l)}, \frac{\partial z^{(l+1)}}{\partial b^{(l)}} = I \text{ where } z^{(l+1)} = \left(W^{(l)}\right)^T a^{(l)} + b^{(l)}$$

Error Signals and Gradients in Multi-Layer NN

- Error signals of the square error cost function for each example

$$\delta^{\left(a^{(l)}\right)} = \frac{\partial}{\partial a^{(l)}} J\left(W,b;x,y\right) = \begin{cases} -\left(y - a^{(l)}\right) \text{ for } l = l_{\max} \\ \dfrac{\partial J}{\partial z^{(l+1)}} \dfrac{\partial z^{(l+1)}}{\partial a^{(l)}} = \left(W^{(l)}\right)^T \delta^{\left(z^{(l+1)}\right)} \text{ otherwise} \end{cases}$$

$$\delta^{\left(z^{(l)}\right)} = \frac{\partial}{\partial z^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} = \delta^{\left(a^{(l)}\right)} \bullet a^{(l)} \bullet \left(\mathbf{1} - a^{(l)}\right)$$

- Gradient of the cost function w.r.t. parameters for each example

$$\nabla_{W^{(l)}} J\left(W,b;x,y\right) = \frac{\partial}{\partial W^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial W^{(l)}} = \delta^{\left(z^{(l+1)}\right)} \left(a^{(l)}\right)^T$$

$$\nabla_{b^{(l)}} J\left(W,b;x,y\right) = \frac{\partial}{\partial b^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial b^{(l)}} = \delta^{\left(z^{(l+1)}\right)}$$

Backpropagation in General Cases

1. Decompose operations in layers of a neural network into function elements whose derivatives w.r.t inputs are known by symbolic computation.

$$h_\theta(x) = \left(f^{(l_{\max})} \circ \ldots \circ f^{(l)}_{\theta^{(l)}} \circ \ldots \circ f^{(2)}_{\theta^{(2)}} \circ f^{(1)}\right)(x) \text{ where } f^{(1)} = x,\ f^{(l_{\max})} = h_\theta(x) \text{ and } \forall l : \frac{\partial f^{(l+1)}}{\partial f^{(l)}} \text{ is known}$$

2. Backpropagate error signals corresponding to a differentiable cost function by numerical computation (Starting from cost function, plug in error signals backward).

$$\delta^{(l)} = \frac{\partial}{\partial f^{(l)}} J(\theta; x, y) = \frac{\partial J}{\partial f^{(l+1)}} \frac{\partial f^{(l+1)}}{\partial f^{(l)}} = \delta^{(l+1)} \frac{\partial f^{(l+1)}}{\partial f^{(l)}} \text{ where } \frac{\partial J}{\partial f^{(l_{\max})}} \text{ is known}$$

3. Use backpropagated error signals to compute gradients w.r.t. parameters only for the function elements with parameters where their derivatives w.r.t parameters are known by symbolic computation.

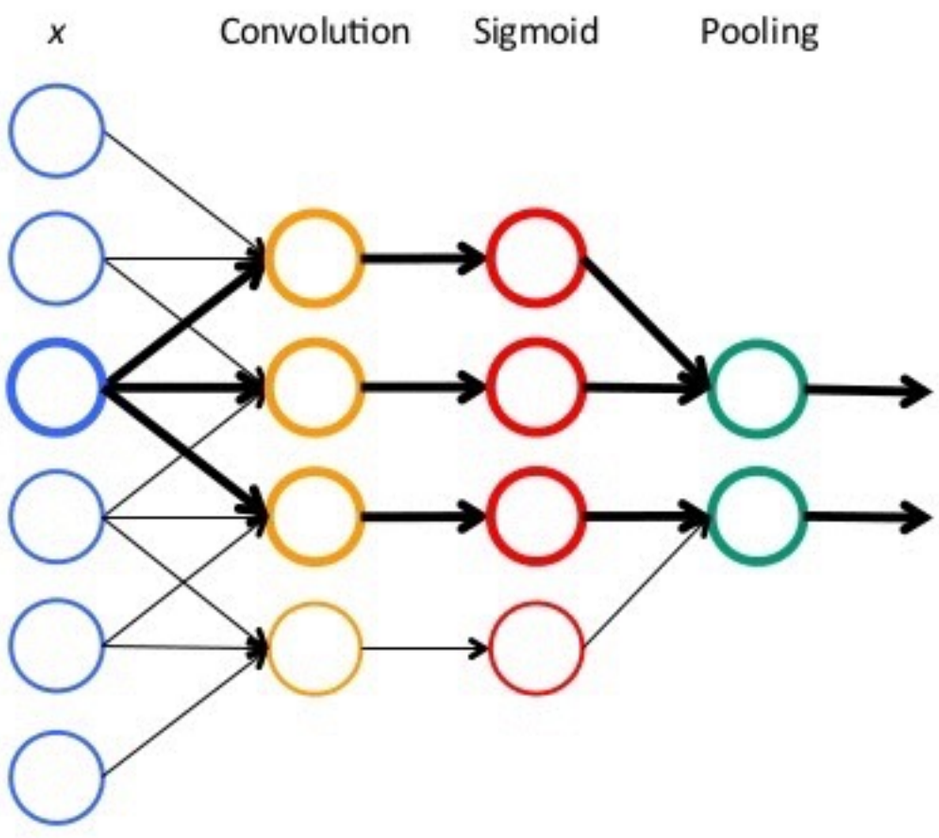
$$\nabla_{\theta^{(l)}} J(\theta; x, y) = \frac{\partial}{\partial \theta^{(l)}} J(\theta; x, y) = \frac{\partial J}{\partial f^{(l)}} \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} = \delta^{(l)} \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} \text{ where } \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} \text{ is known}$$

4. Sum gradients over all example to get overall gradient. $\nabla_{\theta^{(l)}} J(\theta) = \sum_{i=1}^{m} \nabla_{\theta^{(l)}} J\left(\theta; x^{(i)}, y^{(i)}\right)$

Convolutional Neural Network

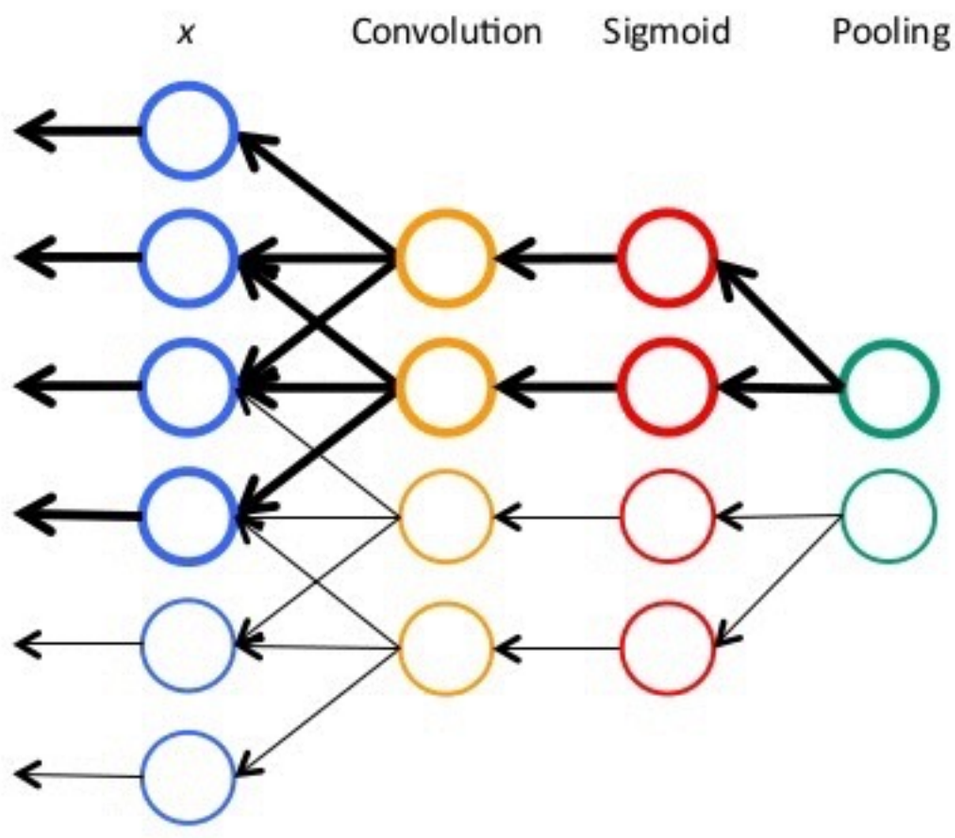
A convolution-pooling layer in Convolutional Neural Network is a composite function decomposed into function elements $f^{(conv)}$, $f^{(sigm)}$, and $f^{(pool)}$.

Let x be the output from the previous layer. Sigmoid nonlinearity is optional.



$$\left(f^{(pool)} \circ f^{(sigm)} \circ f_w^{(conv)}\right)(x)$$

Forward propagation



Backward propagation

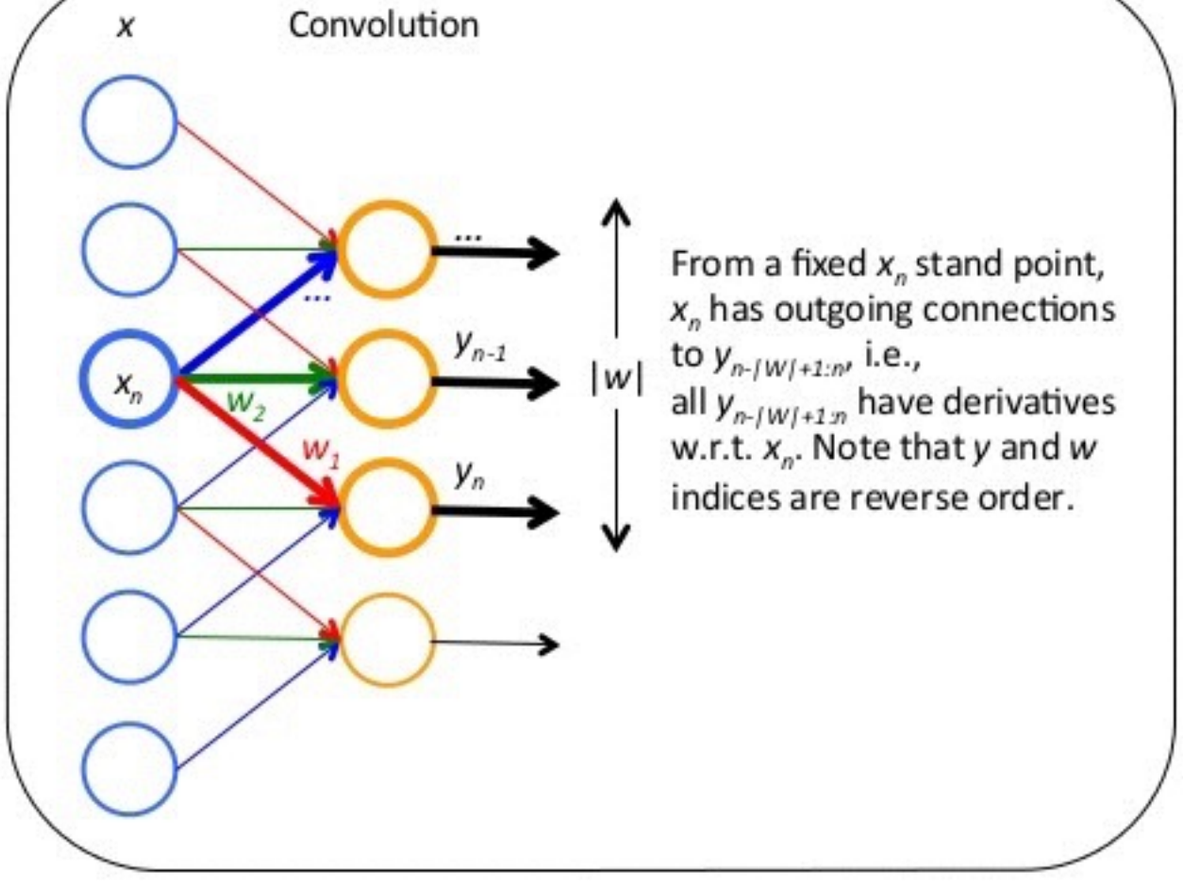
Derivatives of Convolution

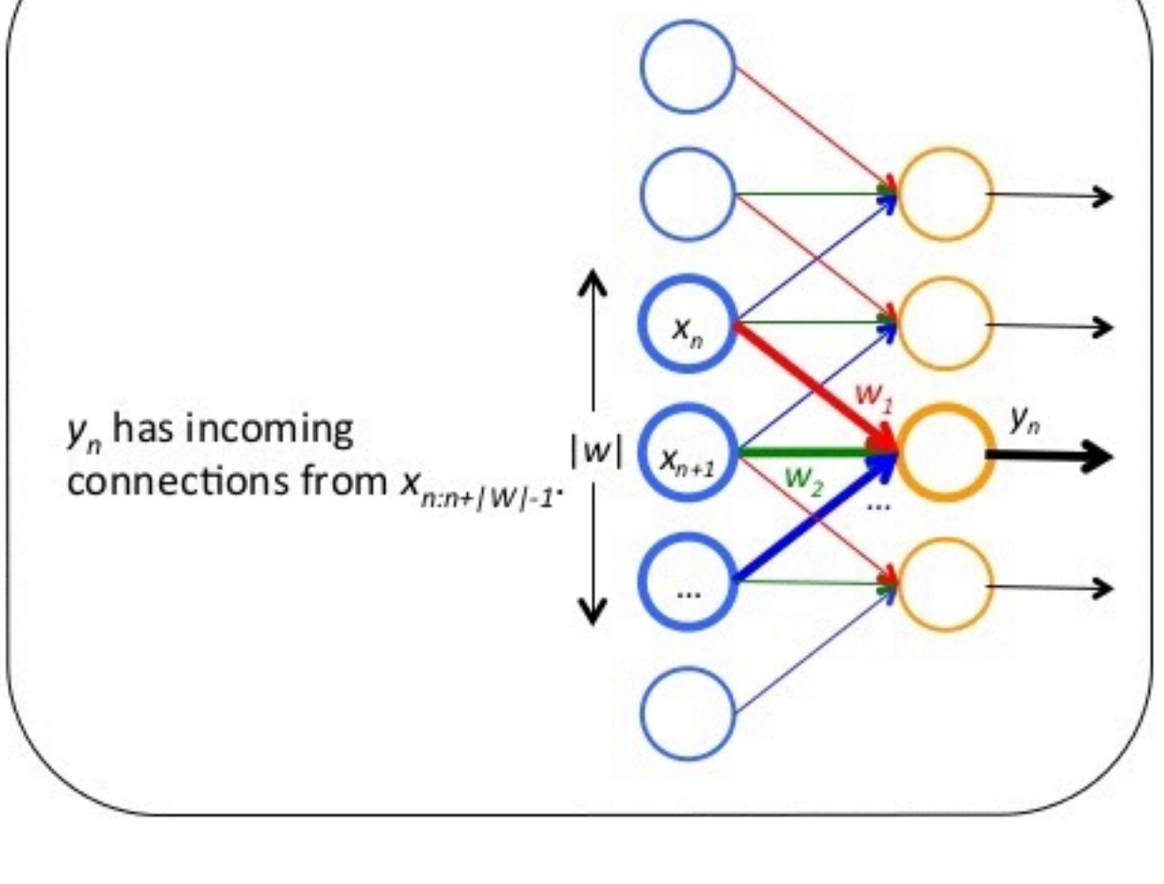
- Discrete convolution parameterized by a feature *w* and its derivatives

  Let *x* be the input, and *y* be the output of convolution layer. Here we focus on only one feature vector *w*, although a convolution layer usually has multiple features $W = [w_1\ w_2\ \dots\ w_n]$. *n* indexes *x* and *y* where $1 \le n \le |x|$ for x_n, $1 \le n \le |y| = |x| - |w| + 1$ for y_n. *i* indexes *w* where $1 \le i \le |w|$. $(f*g)[n]$ denotes the *n*-th element of $f*g$.

$$y = x * w = [y_n], y_n = (x * w)[n] = \sum_{i=1}^{|w|} x_{n+i-1} w_i = w^T x_{n:n+|w|-1}$$

$$\frac{\partial y_{n-i+1}}{\partial x_n} = w_i,\ \frac{\partial y_n}{\partial w_i} = x_{n+i-1} \text{ for } 1 \le i \le |w|$$





Backpropagation in Convolution Layer

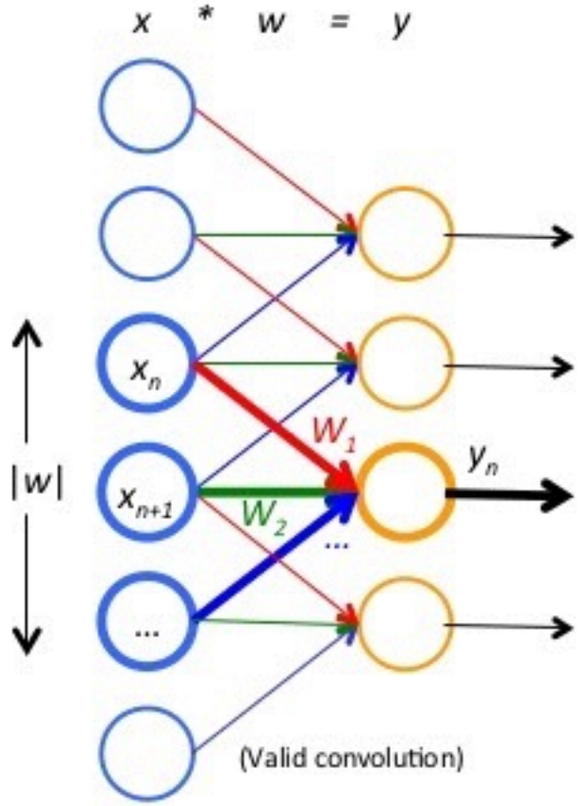
Error signals and gradient for each example are computed by convolution using the commutativity property of convolution and the multivariable chain rule of derivative.

Let's focus on single elements of error signals and a gradient w.r.t. *w*.

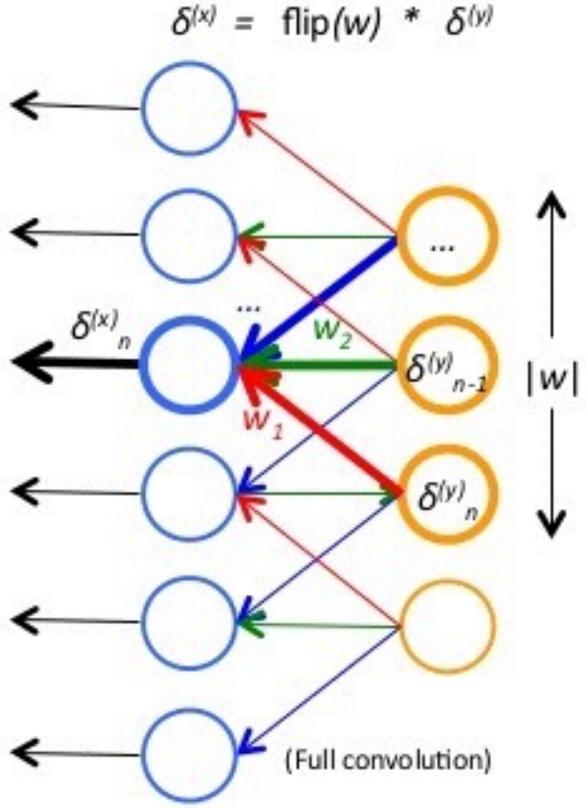
$$\delta_n^{(x)} = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=1}^{|w|} \frac{\partial J}{\partial y_{n-i+1}}\frac{\partial y_{n-i+1}}{\partial x_n} = \sum_{i=1}^{|w|} \delta_{n-i+1}^{(y)} w_i = \left(\delta^{(y)} * \mathrm{flip}(w)\right)[n],\ \delta^{(x)} = \left[\delta_n^{(x)}\right] = \delta^{(y)} * \mathrm{flip}(w)$$

↑ Reverse order linear combination

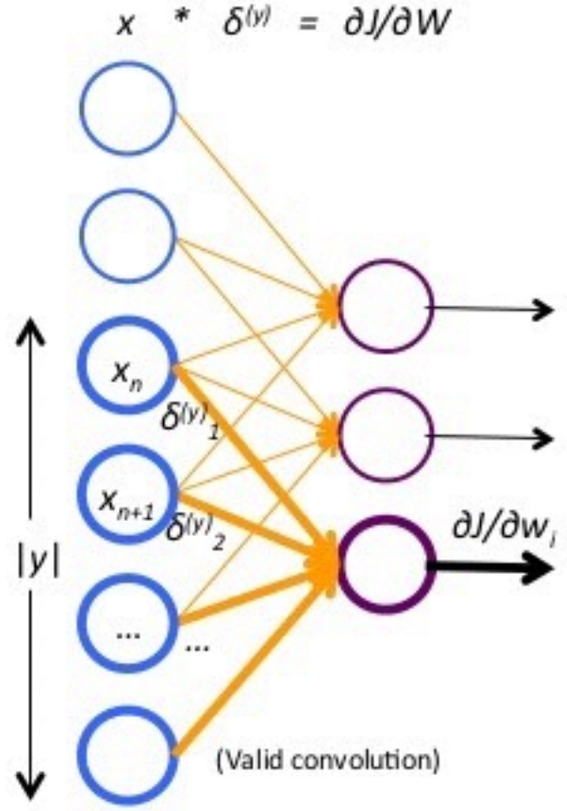
$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1} \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1} \delta_n^{(y)} x_{n+i-1} = \left(\delta^{(y)} * x\right)[i],\ \frac{\partial J}{\partial w} = \left[\frac{\partial J}{\partial w_i}\right] = \delta^{(y)} * x = x * \delta^{(y)}$$



Forward propagation (convolution)



Backward propagation



Gradient computation

Derivatives of Pooling

Pooling layer subsamples statistics to obtain summary statistics with any aggregate function (or filter) *g* whose input is vector, and output is scalar. Subsampling is an operation like convolution, however *g* is applied to disjoint (non-overlapping) regions.

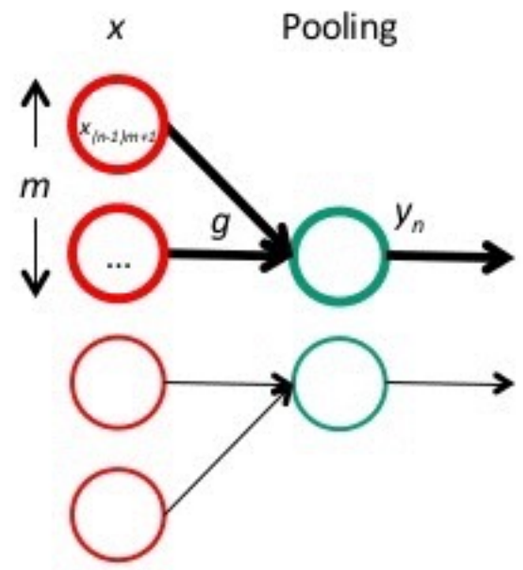
- Definition: *subsample* (or *downsample*)

  Let *m* be the size of pooling region, *x* be the input, and *y* be the output of the pooling layer. subsample(*f*, *g*)[*n*] denotes the *n*-th element of subsample(*f*, *g*).

$$y_n = \text{subsample}(x, g)[n] = g\left(x_{(n-1)m+1:nm}\right)$$

$$y = \text{subsample}(x, g) = [y_n]$$

$$g(x) = \begin{cases} \dfrac{\sum_{k=1}^{m} x_k}{m}, \dfrac{\partial g}{\partial x} = \dfrac{\mathbf{1}}{m} & \text{mean pooling} \\ \max(x), \dfrac{\partial g}{\partial x_i} = \begin{cases} 1 \text{ if } x_i = \max(x) \\ 0 \text{ otherwise} \end{cases} & \text{max pooling} \\ \|x\|_p = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p}, \dfrac{\partial g}{\partial x_i} = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p-1} |x_i|^{p-1} & \text{L}^\text{p} \text{ pooling} \\ \text{or any other differentiable } \mathbf{R}^m \to \mathbf{R} \text{ functions} \end{cases}$$



Backpropagation in Pooling Layer

Error signals for each example are computed by upsampling. Upsampling is an operation which backpropagates (distributes) the error signals over the aggregate function g using its derivatives $g'_n = \partial g / \partial x_{(n-1)m+1:nm}$. g'_n can change depending on pooling region n.

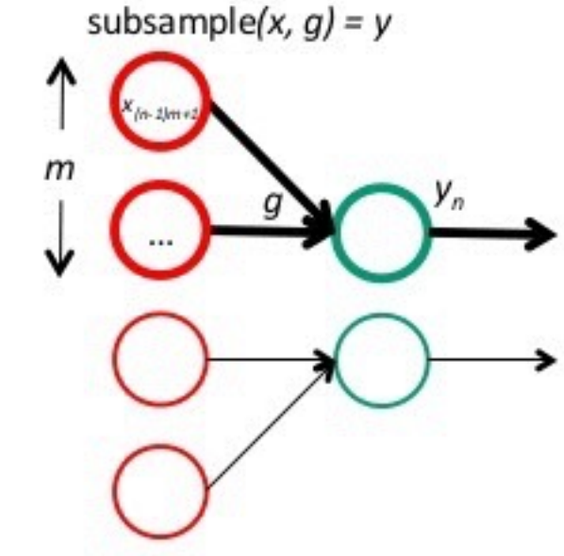
- In max pooling, the unit which was the max at forward propagation receives all the error at backward propagation and the unit is different depending on the region n.

- Definition: *upsample*

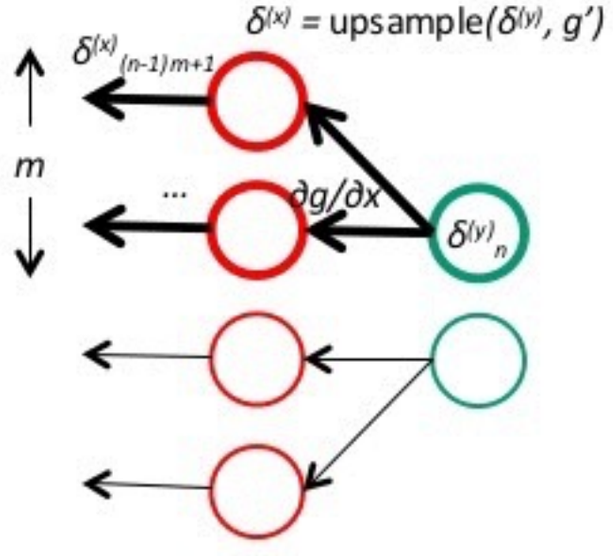
  upsample(f, g)[n] denotes the n-th element of upsample(f, g).

$$\delta^{(x)}_{(n-1)m+1:nm} = \text{upsample}\left(\delta^{(y)}, g'\right)[n] = \delta^{(y)}_n g'_n = \delta^{(y)}_n \frac{\partial g}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial x_{(n-1)m+1:nm}}$$

$$\delta^{(x)} = \text{upsample}\left(\delta^{(a)}, g'\right) = \left[\delta^{(x)}_{(n-1)m+1:nm}\right]$$



Forward propagation (subsapmling)



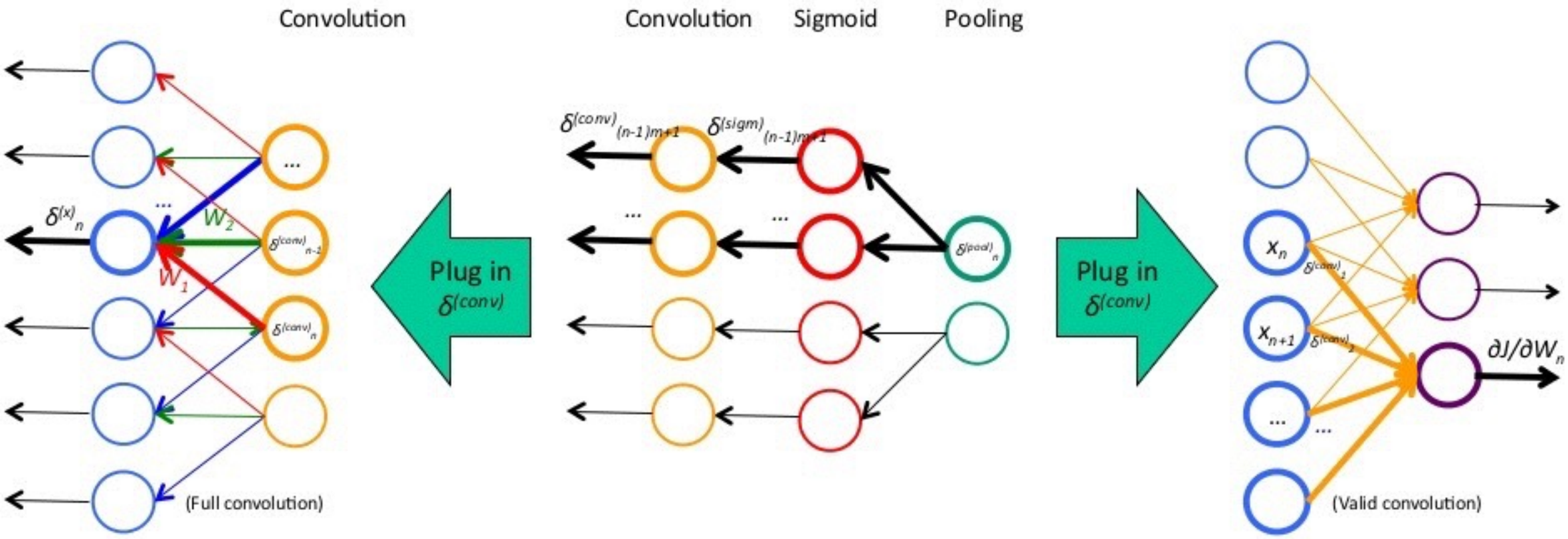
Backward propagation (upsapmling)

Backpropagation in CNN (Summary)

2. Propagate error signals $\delta^{(conv)}$

1. Propagate error signals $\delta^{(pool)}$

3. Compute gradient $\nabla_w J$



$$\delta^{(x)} = \delta^{(conv)} * \mathrm{flip}(w)$$

$$\delta^{(conv)} = \mathrm{upsample}\left(\delta^{(pool)}, g'\right) \bullet \underbrace{f^{(sigm)} \bullet \left(1 - f^{(sigm)}\right)}_{\text{Derivative of sigmoid}}$$

$$x * \delta^{(conv)} = \nabla_w J$$

Remarks

- References
  - UFLDL Tutorial, http://ufldl.stanford.edu/tutorial
  - Chain Rule of Neural Network is Error Back Propagation, http://like.silk.to/studymemo/ChainRuleNeuralNetwork.pdf

- Acknowledgement

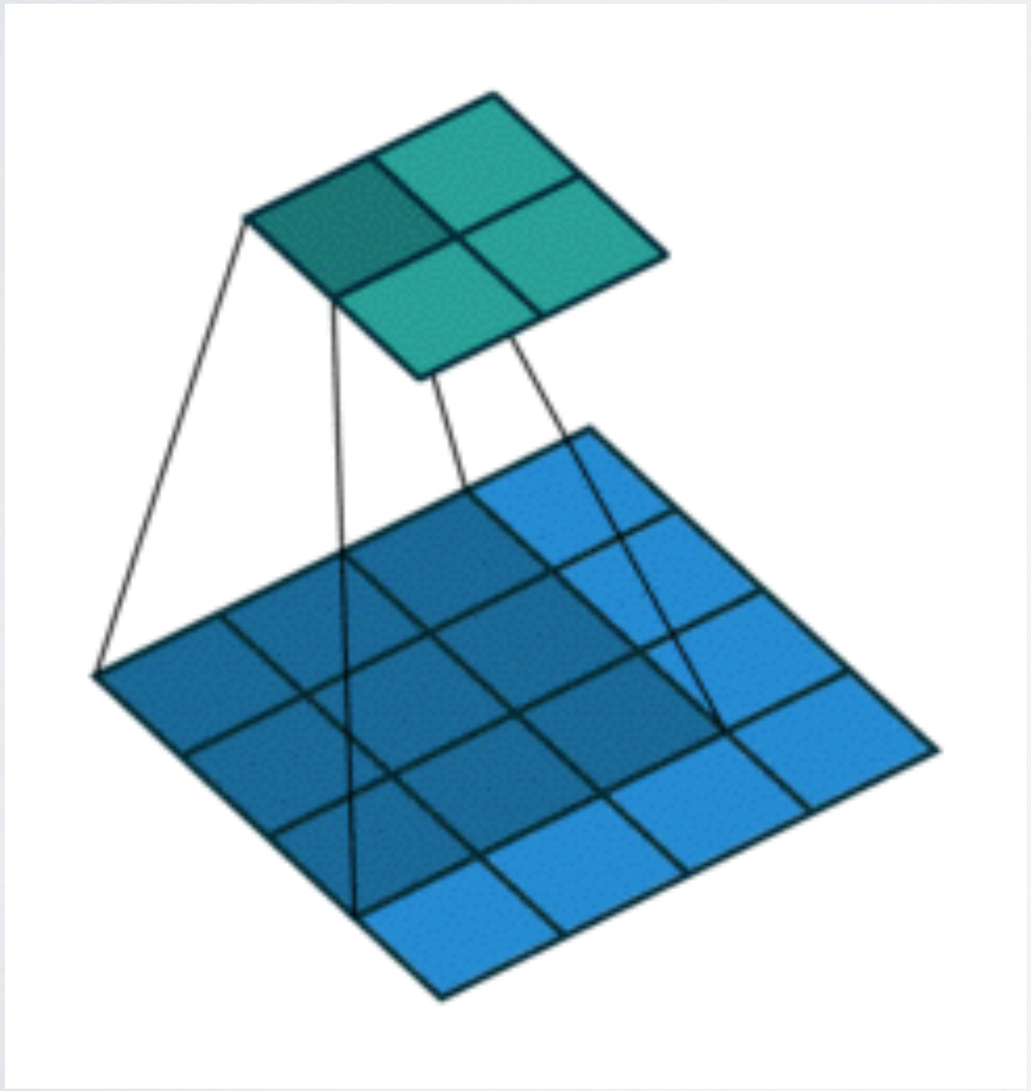
This memo was written thanks to a good discussion with Prof. Masayuki Tanaka.

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).



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

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.

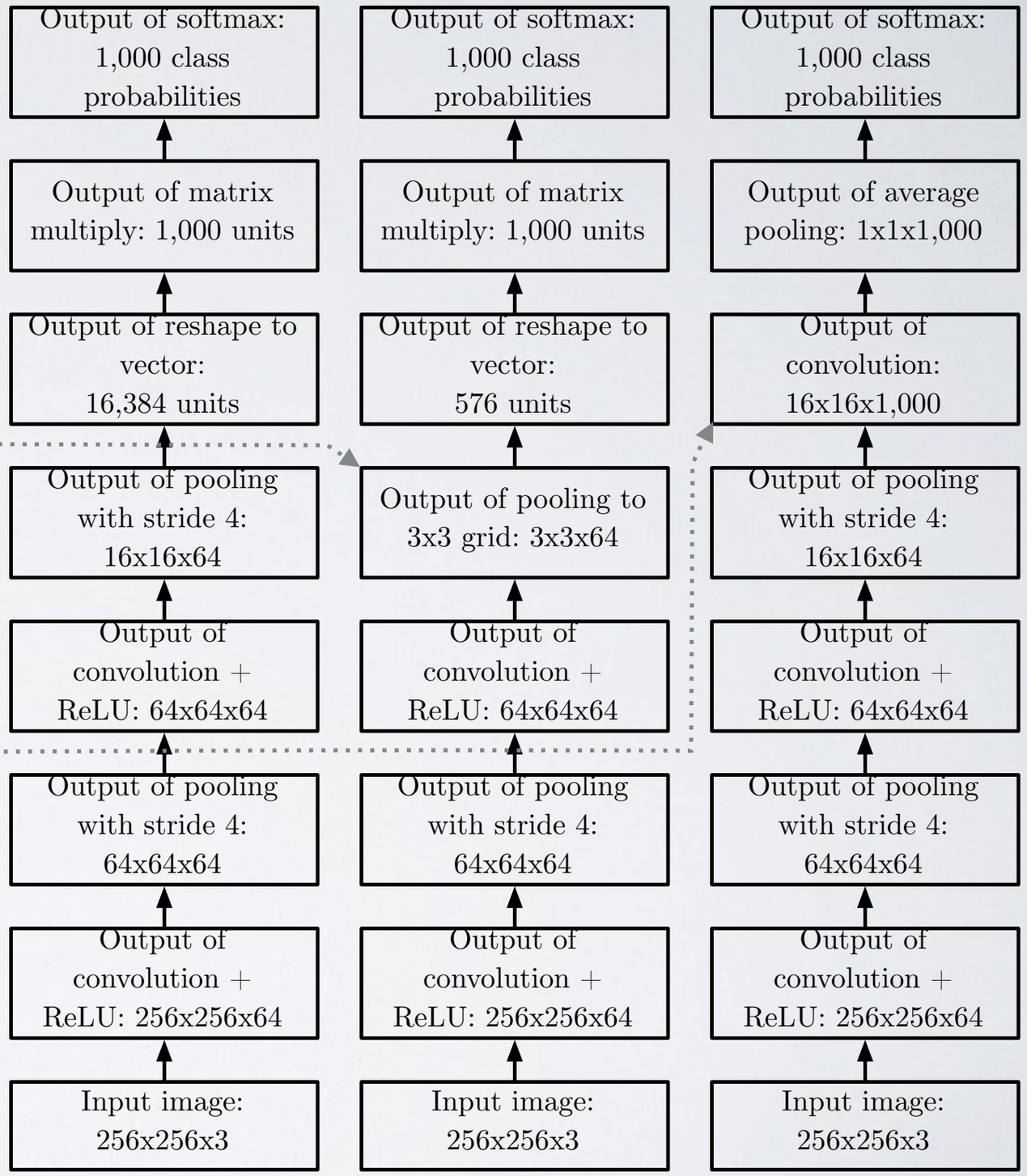


Image from Deep Learning Textbook (Goodfellow et al, 2016)

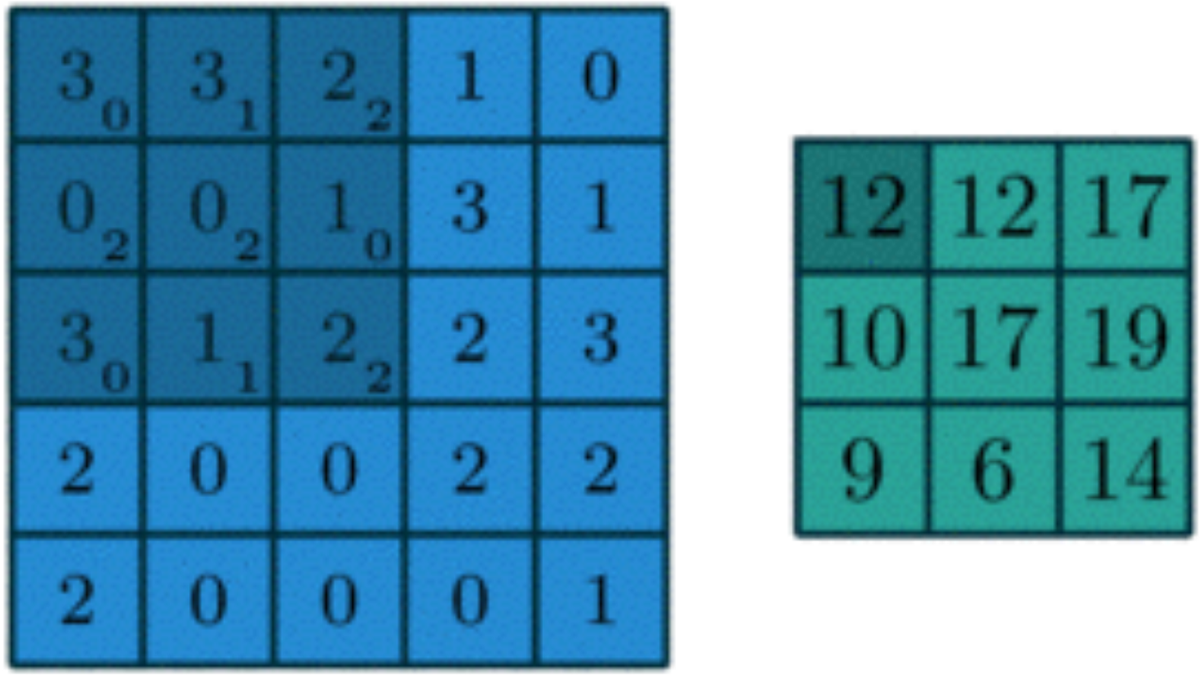
CONVOLUTION BASICS

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.

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

Input feature map

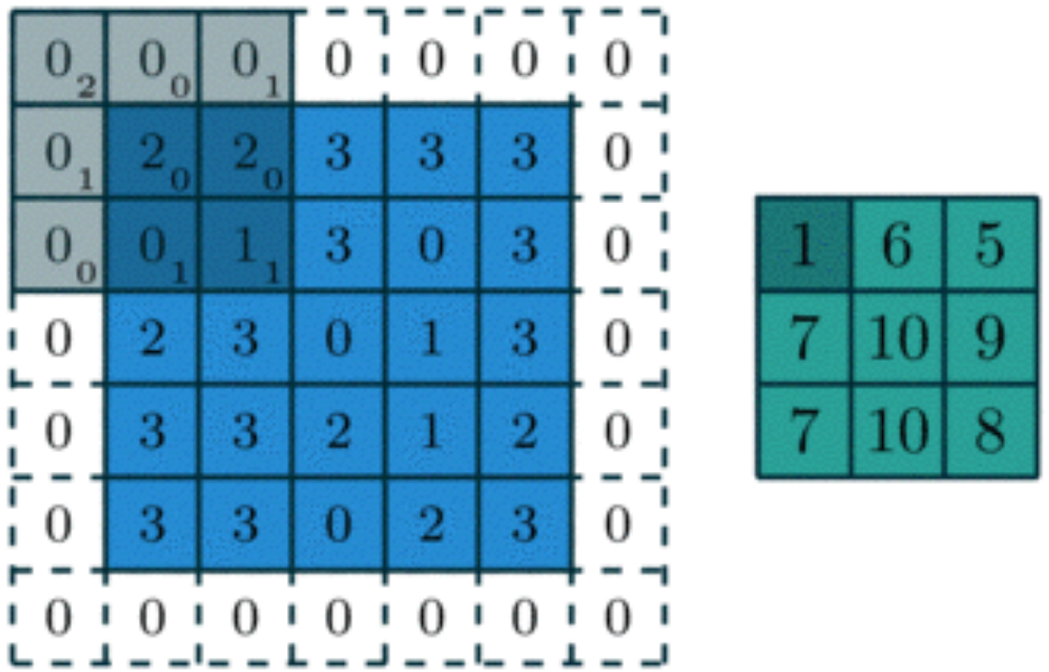


TERMINOLOGY

- 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:



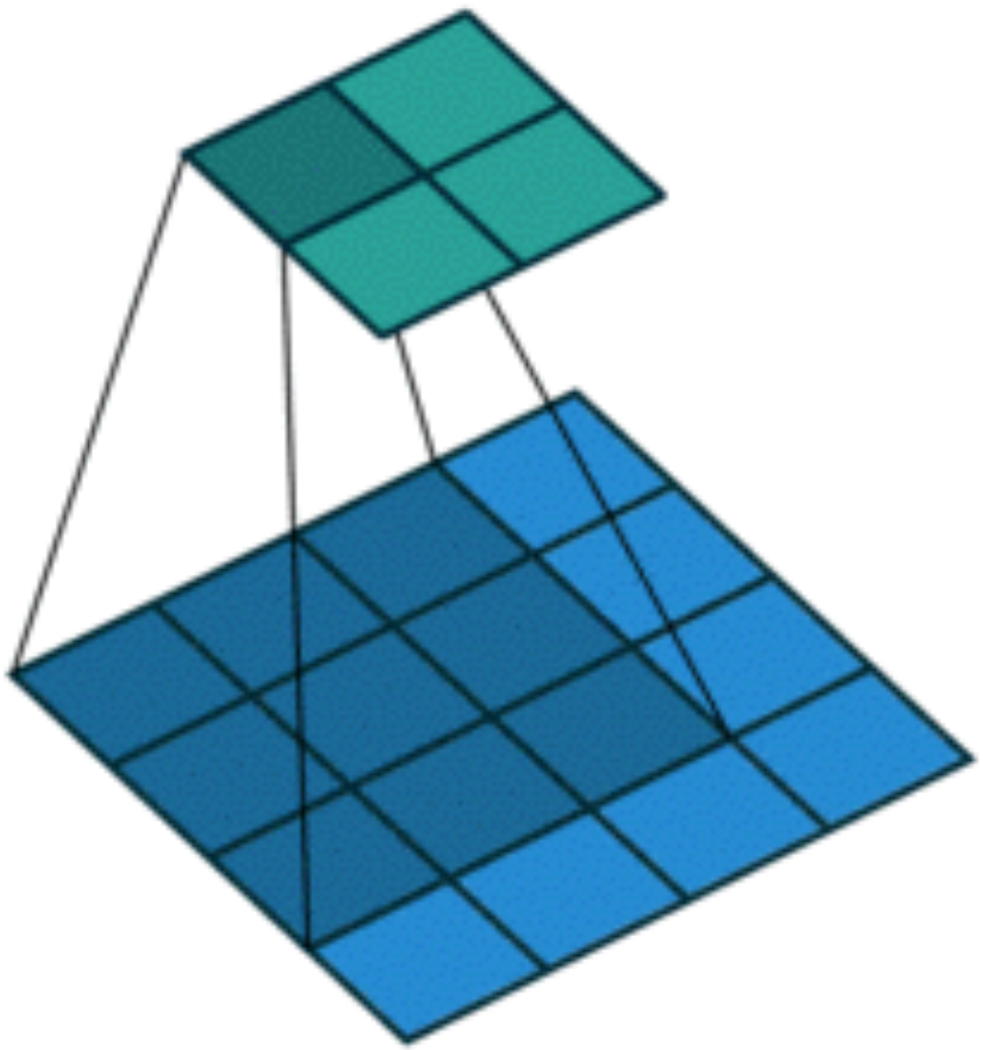
TERMINOLOGY

- 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:

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

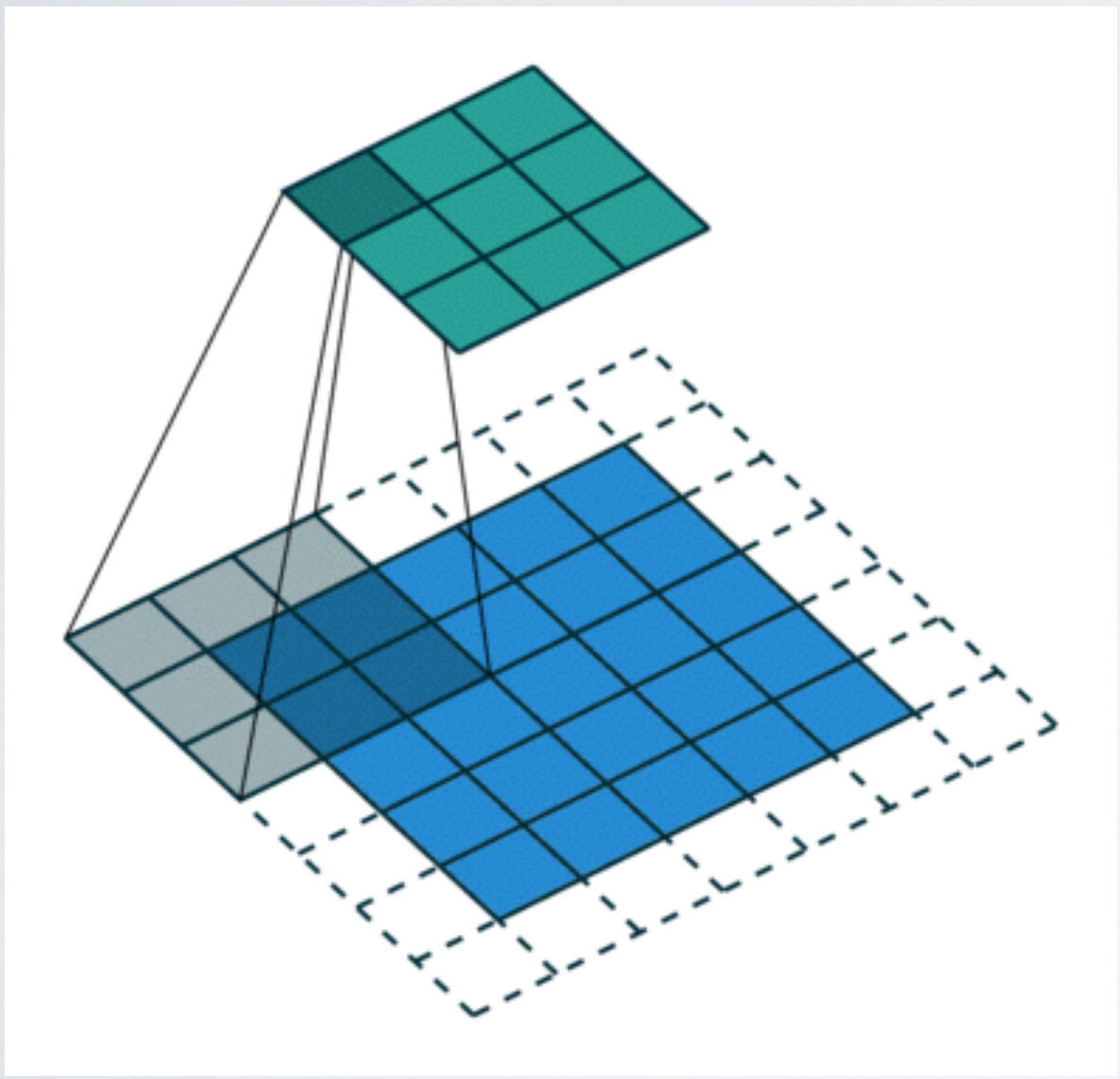
floor function

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

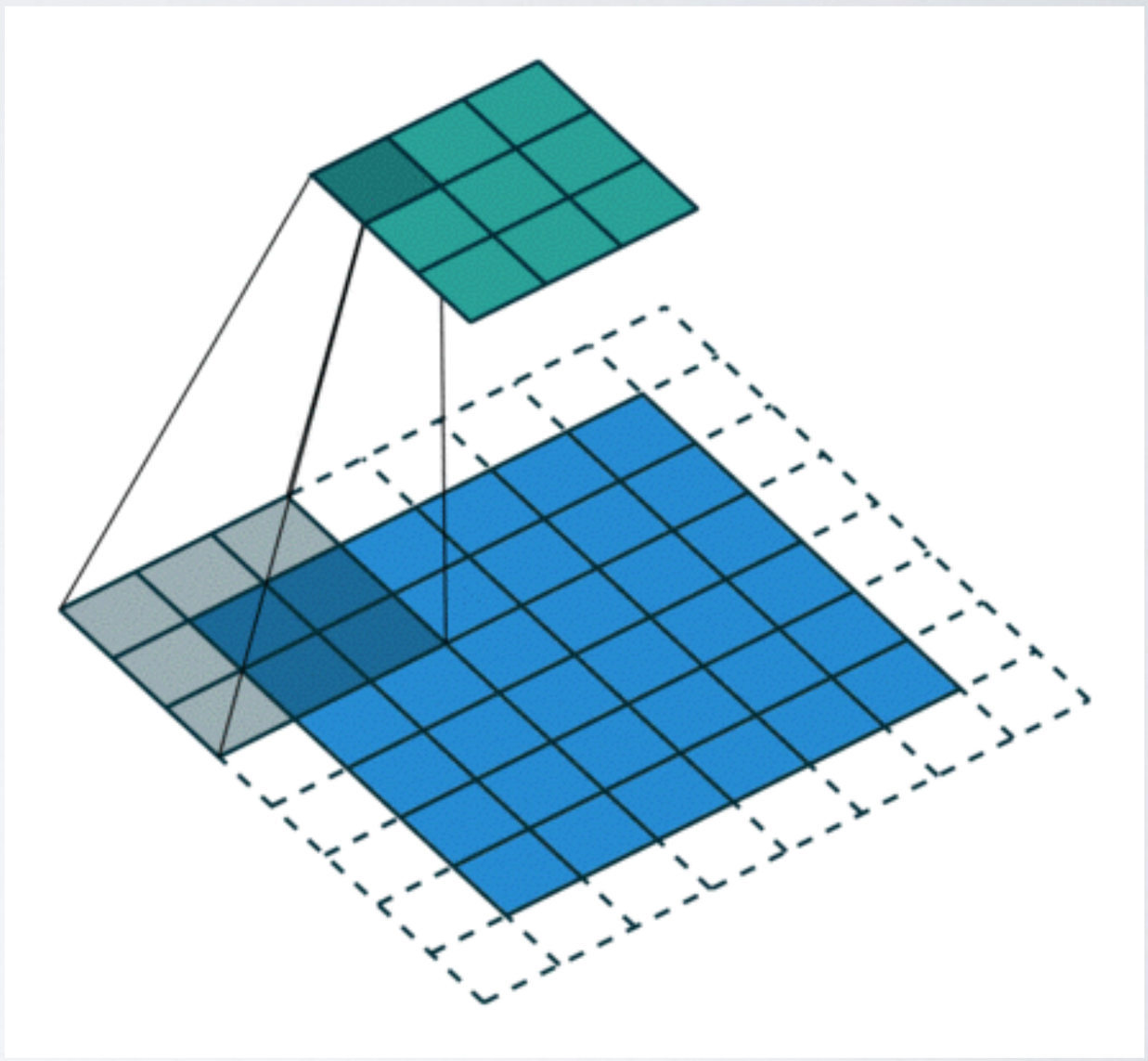


CONVOLUTIONS…

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

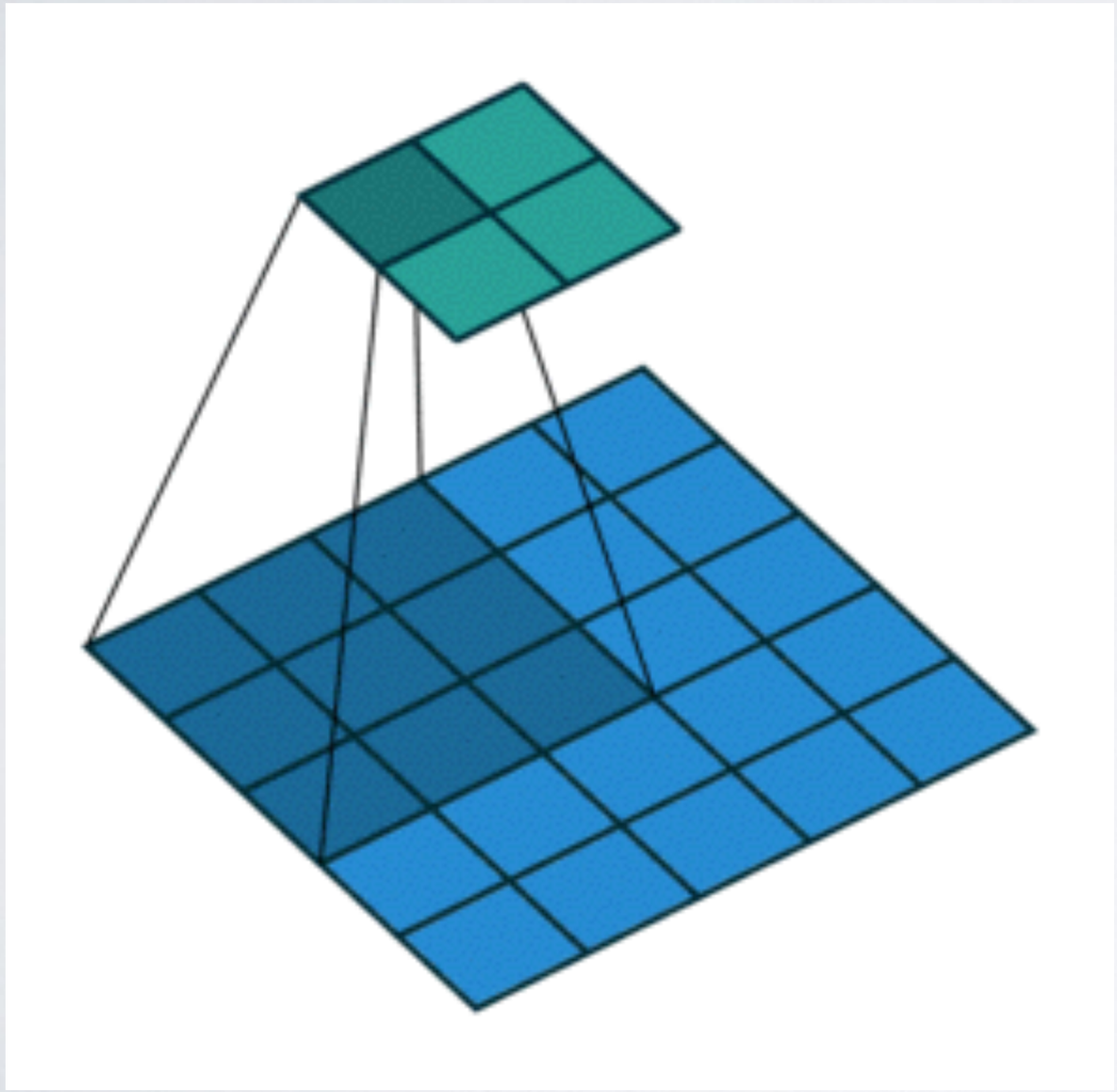


$i = 6$ and $k = 3, s = 2$ and $p = 1$.

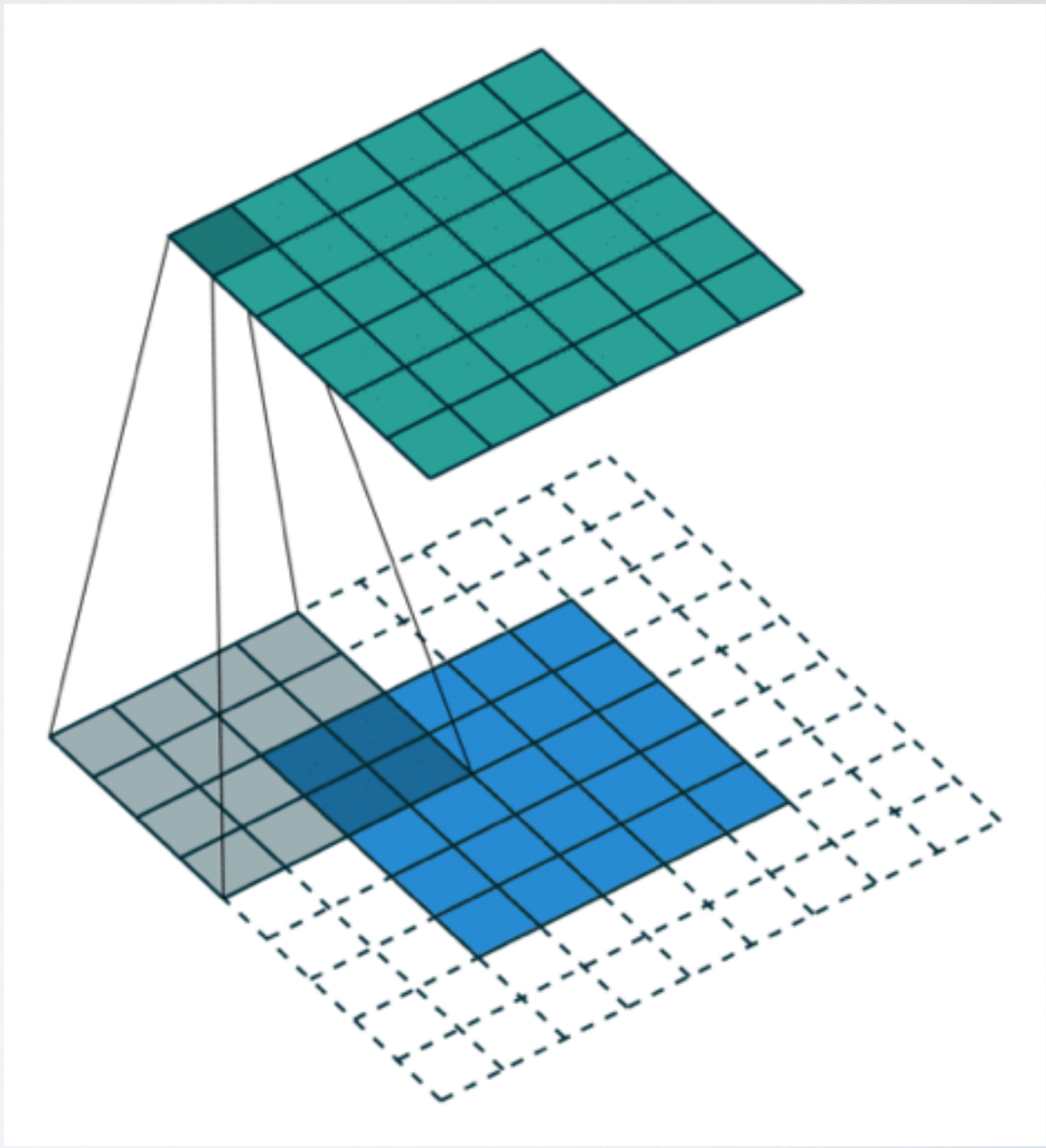


CONVOLUTIONS…

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

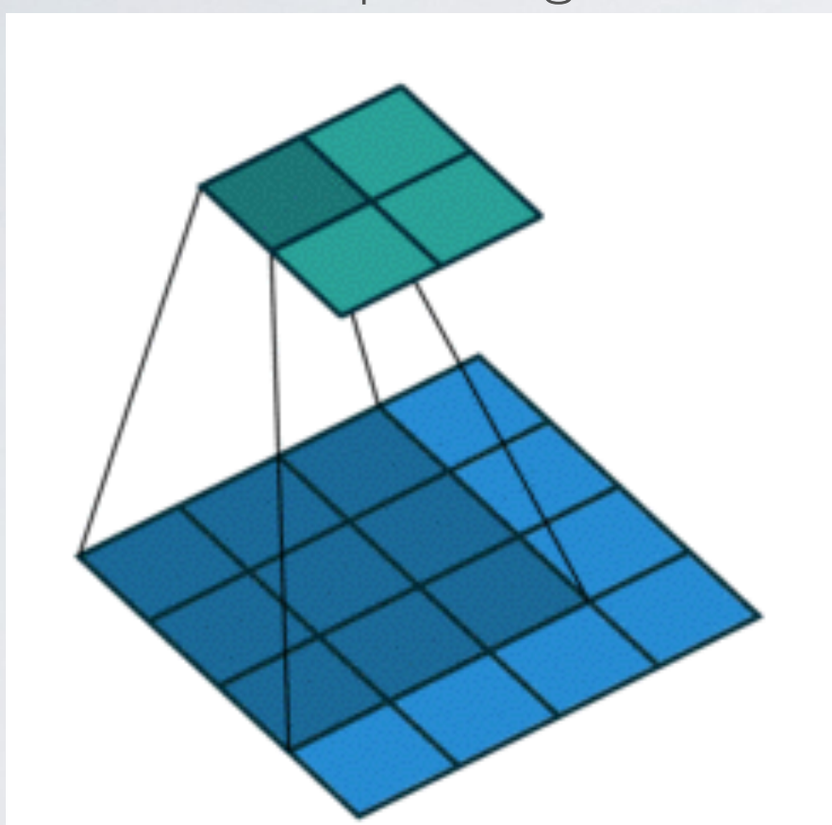


$i = 5$ and $k = 4, s = 1$ and $p = 2$.

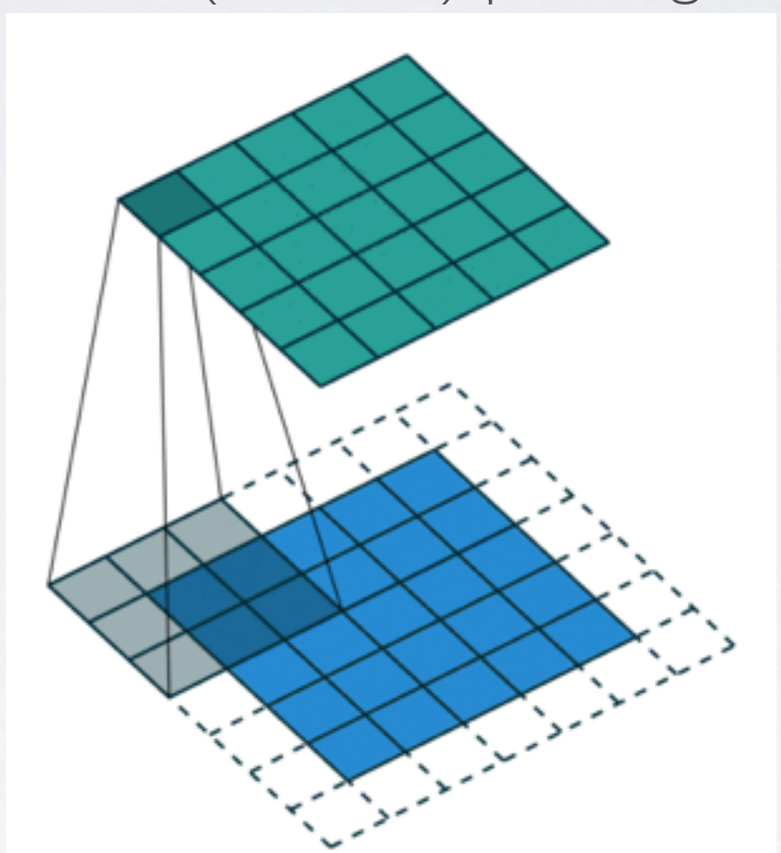


PADDING: SPECIAL CASES

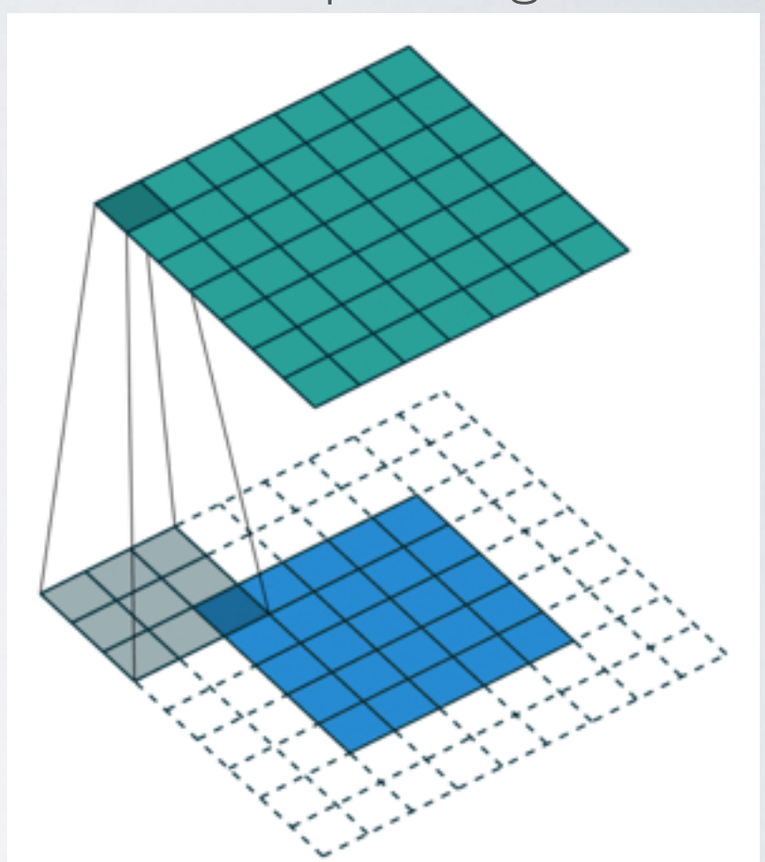
valid convolution
no padding



same convolution
half (or same) padding



full convolution
full padding



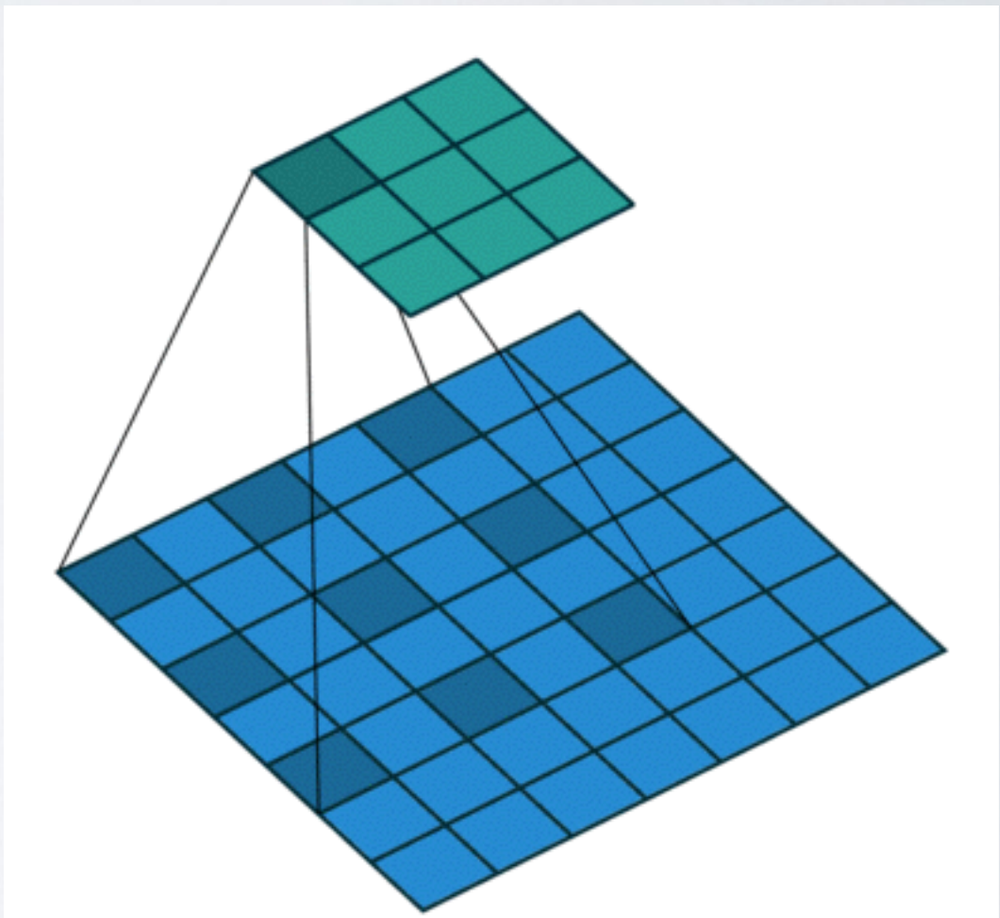
output dimension = input dimension

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:

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

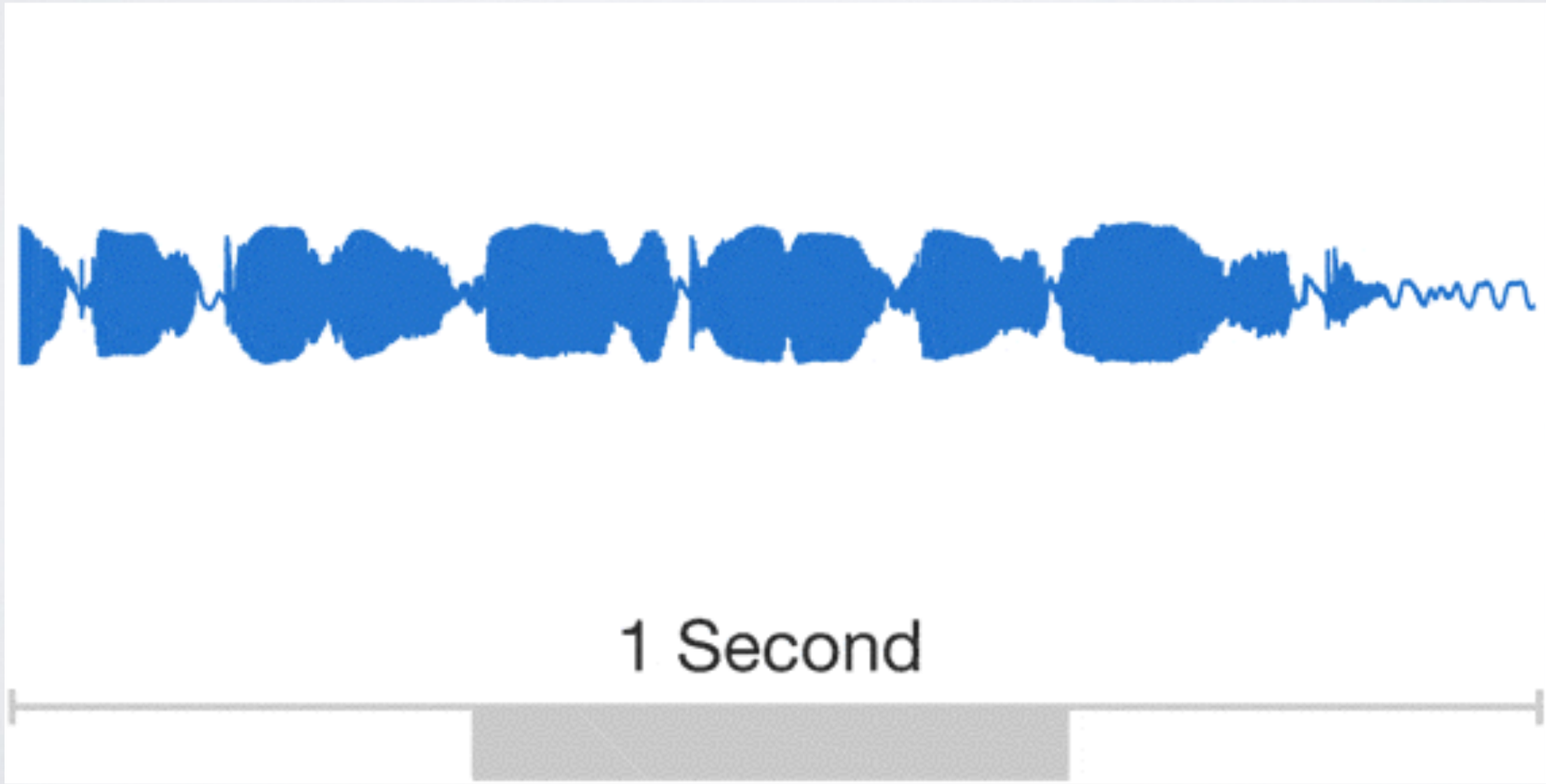
$i = 7, k = 3, d = 2, s = 1$ and $p = 0$.



DILATED CONVOLUTIONS: WAVENET

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

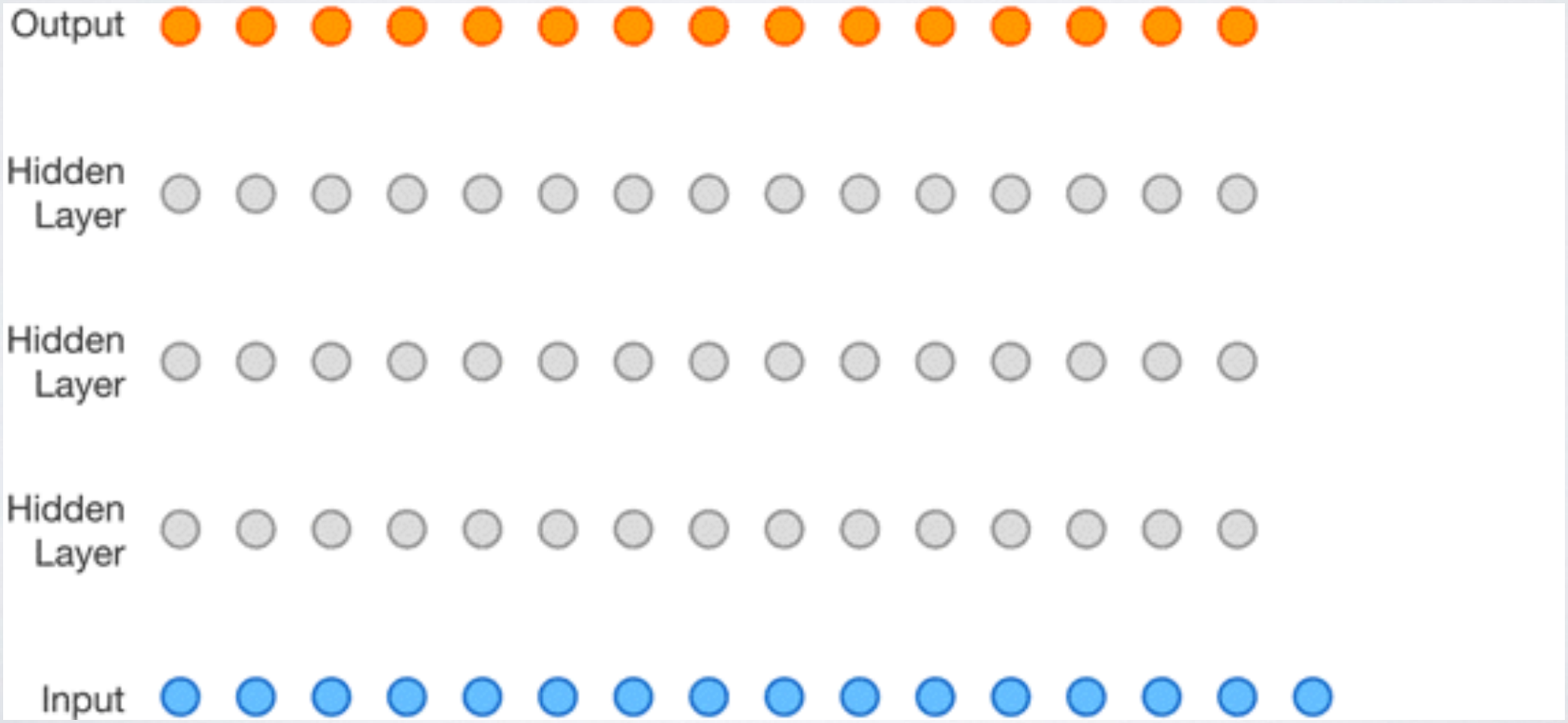
- WaveNet: Dilated CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: WAVENET

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

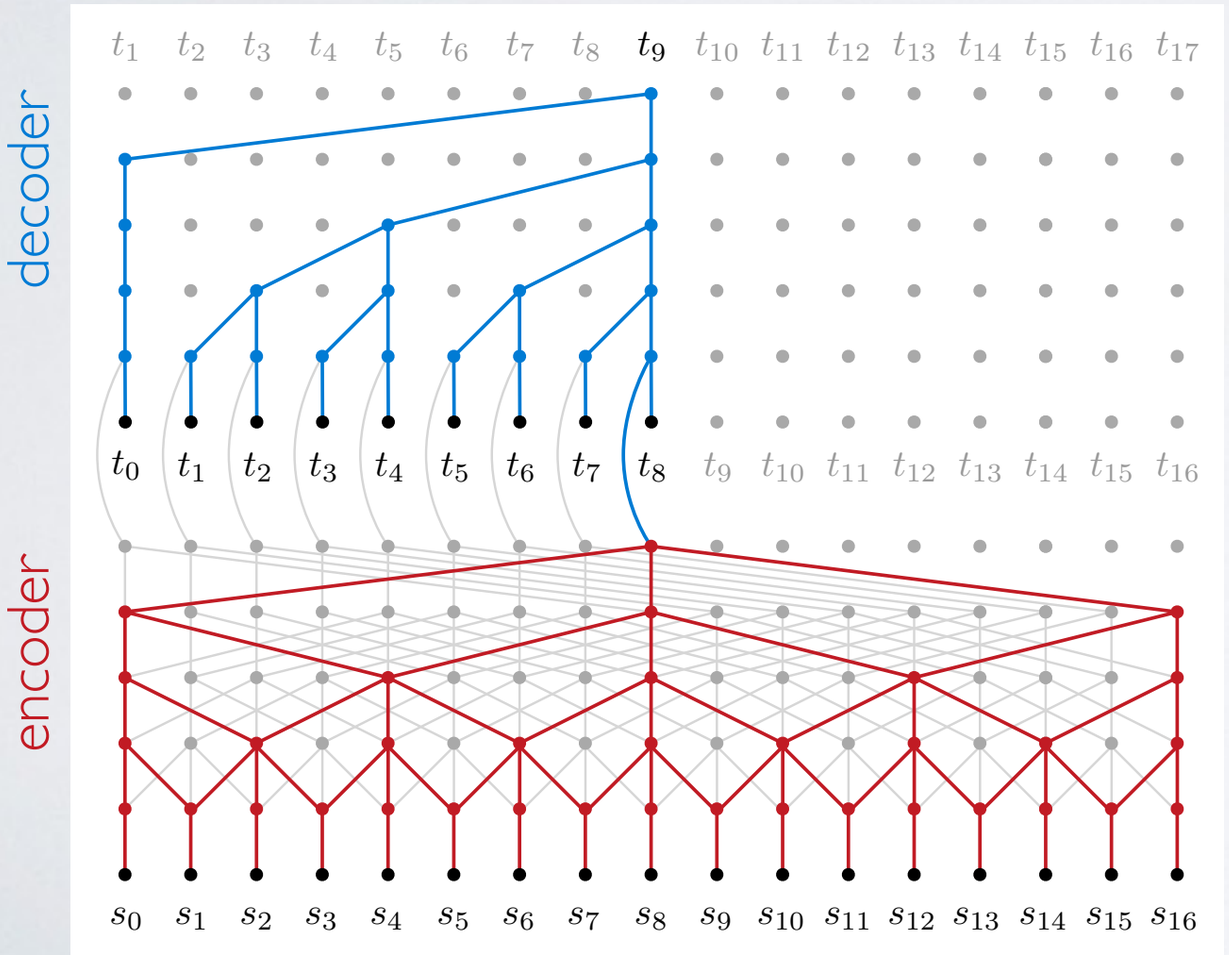
- WaveNet: dilated causal CNNs for speech synthesis
  - Modeling audio data is challenging because of the many scales over which information is encoded.
  - WaveNet uses dilated CNNs in 1-D to efficiently cover the relavent scales.



DILATED CONVOLUTIONS: BYTENET

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

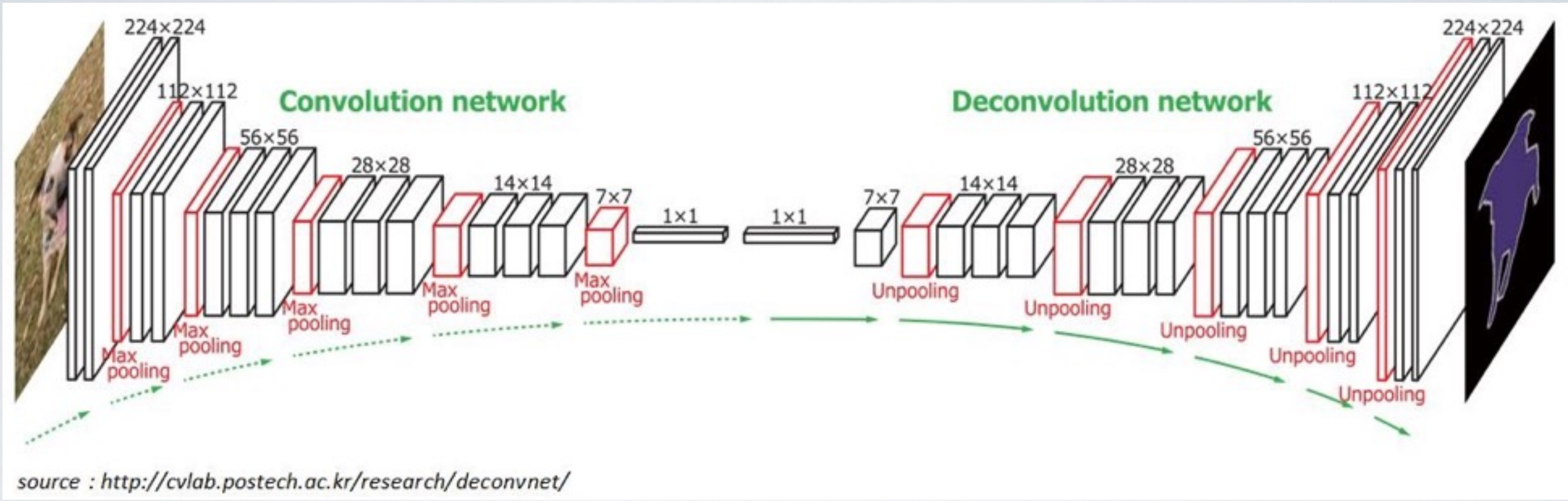
- ByteNet: dilated CNN for machine translation.
  - Uses an encoder-decoder structure with a causal convolution in the decoder.
  - State-of-the-art, computationally efficient machine translation



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.

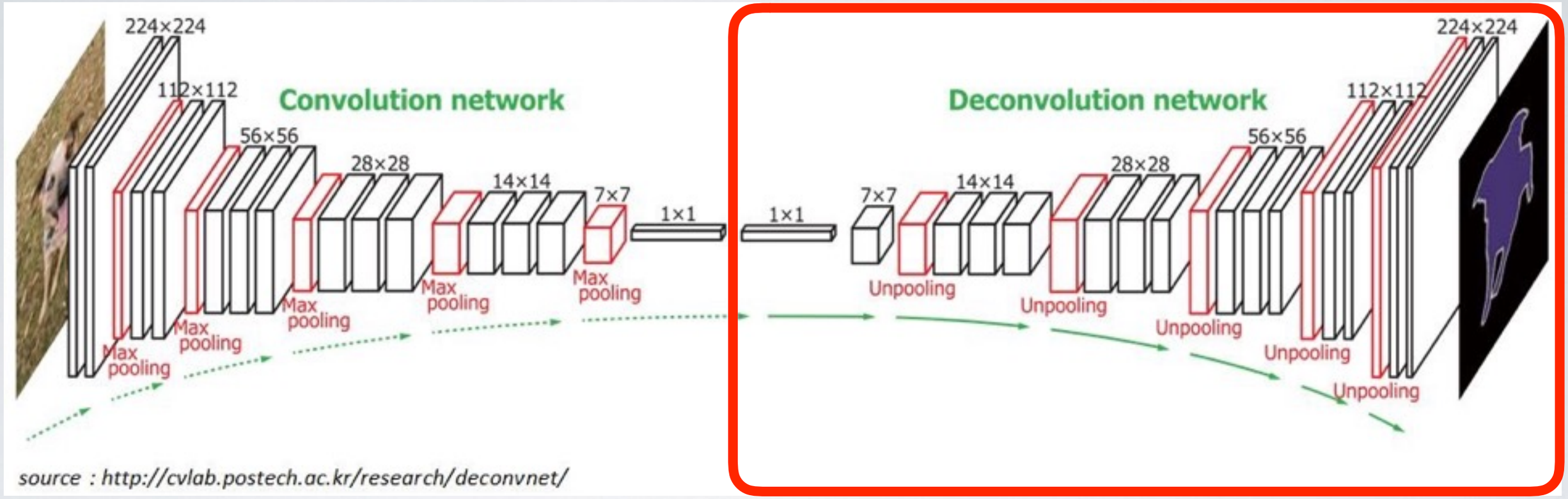
(DE)CONVOLUTIONS FOR STRUCTURED OUTPUT



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

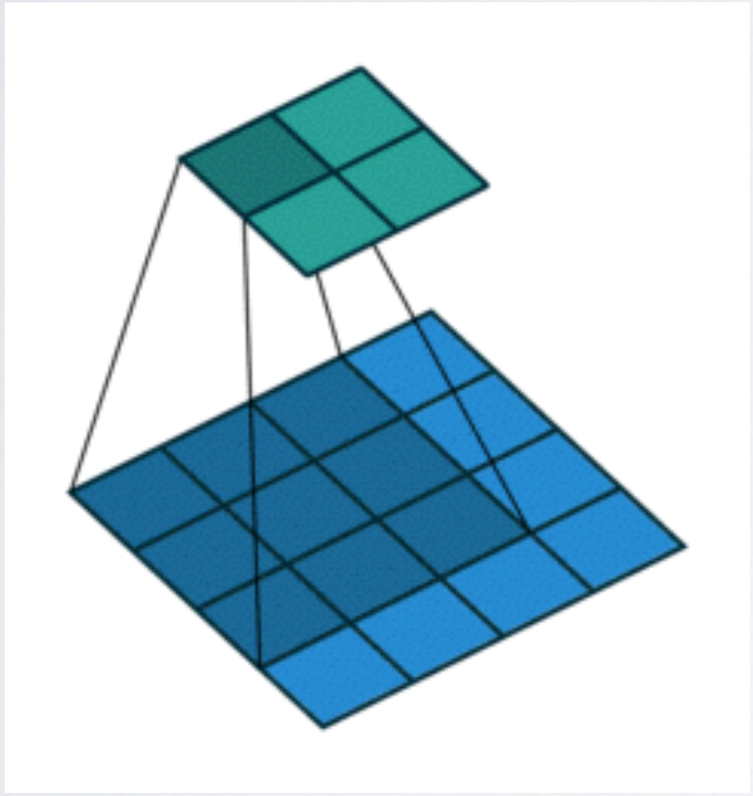


How do we do this part?

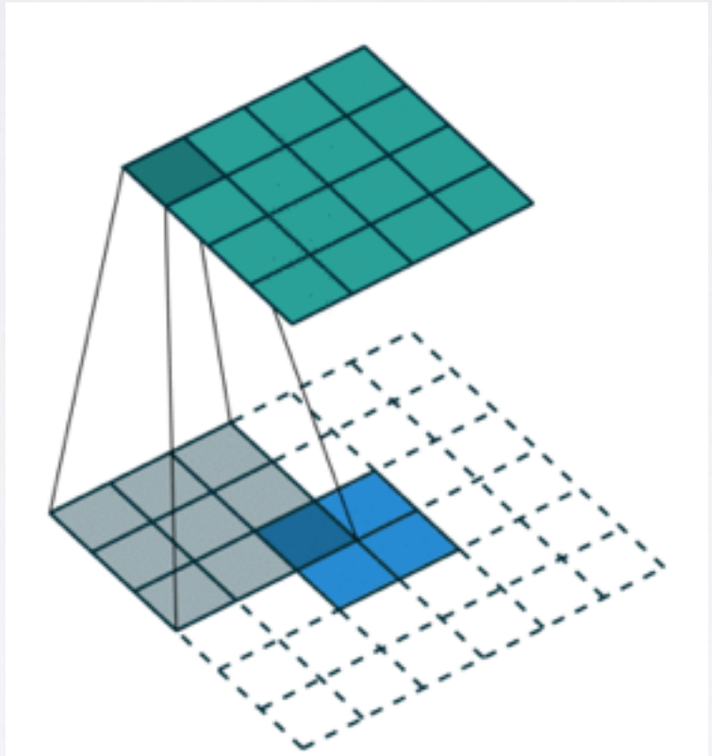
- Option 1: Transposed (a.k.a. fractionally strided) convolution.
- Option 2: Upsampling (e.g. nearest-neighbour or bilinear) + convolution.

TRANSPOSE CONVOLUTION

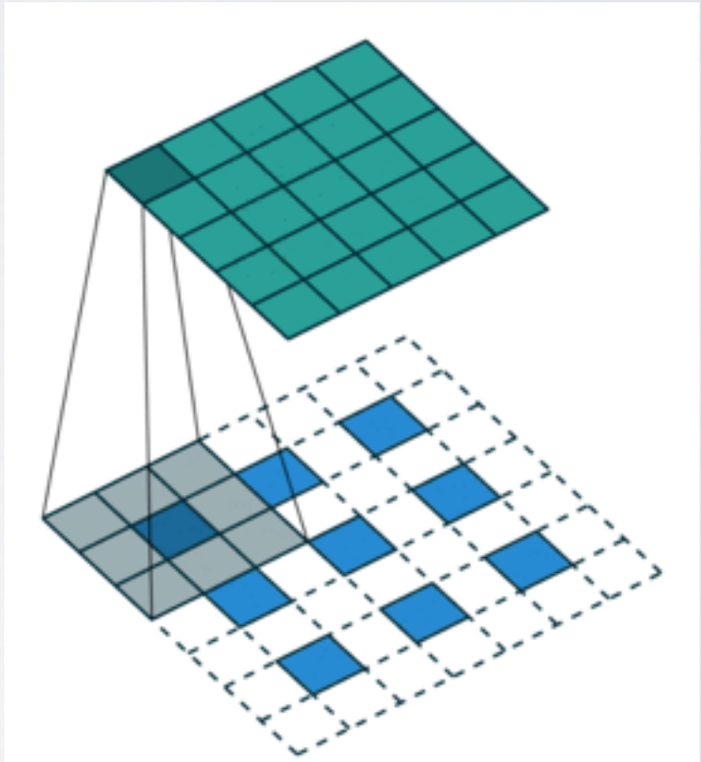
- Transposed convolution is the operation done in the backprop pass through a convolution.
- It's the natural "inverse" of the convolution operation.



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



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

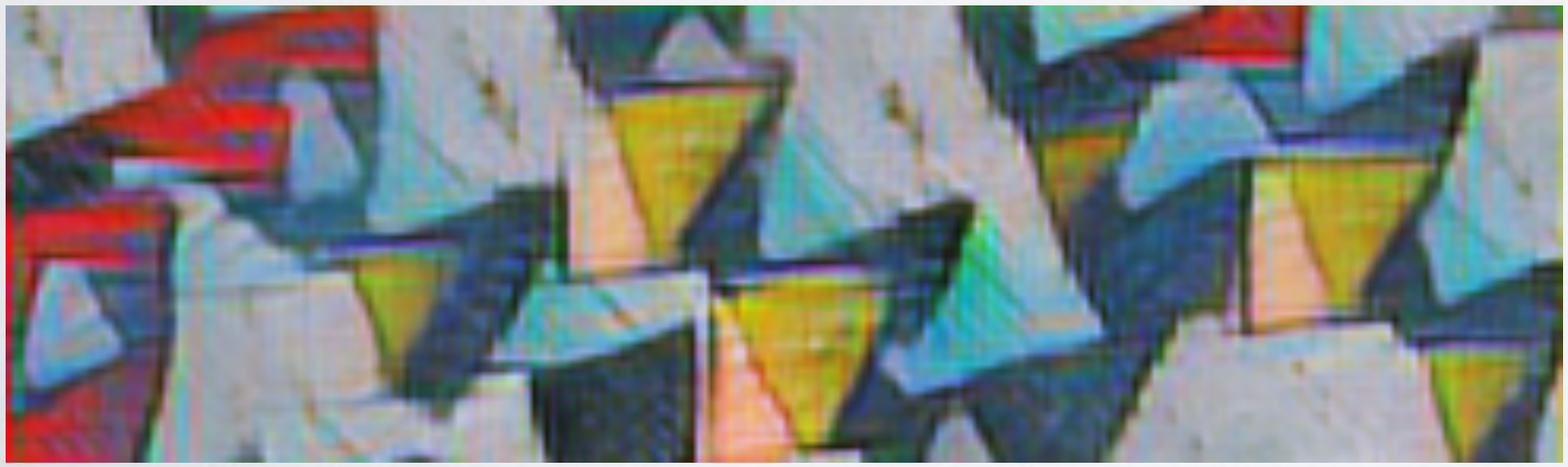


i = 5, k = 3, s = 2 and p= 1

INTERPOLATION + CONVOLUTION

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

Using transpose convolution:
Heavy checkerboard artifacts.



Using resize-convolution.
No checkerboard artifacts.