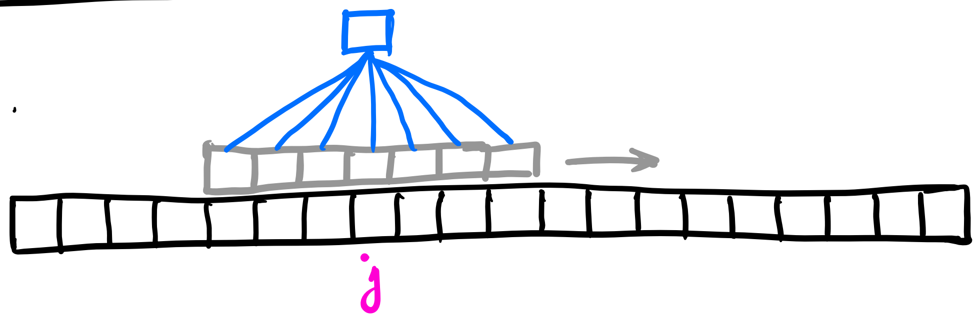
Convolution Layers



Kernel = $w_{-\ell}, \dots, w_\ell \in \mathbb{R}^{2\ell+1}$

Value at location i:

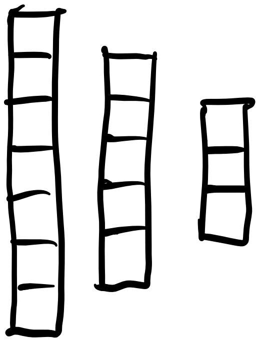
$$y_i = \sum_{i'=-\ell}^{+\ell} w_{i'} \, x_{i-i'}$$

$\rightarrow$ $(i-i')$ construction induces a flip. Flip is required for a convolution. Without the flip it is cross-correlation. In DL it doesn't matter, since the network can learn the flipped version.

The kernel is shifted and the amount of pixels (values) that are skipped between any two shifted locations is called a stride.

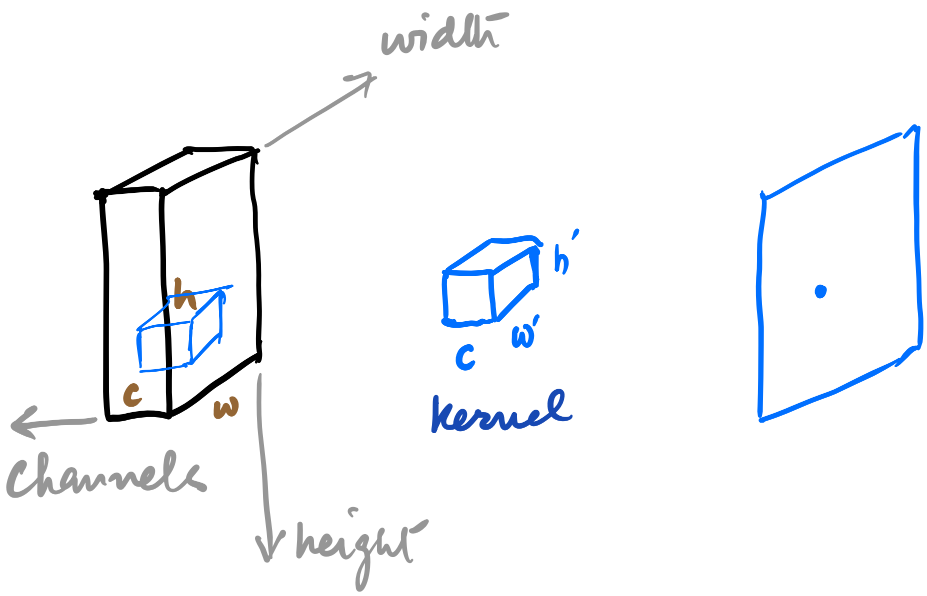
The output is 2ℓ smaller than the input since the kernel falls off the signal in the first ℓ and the last ℓ locations. There are many ways to address this issue using a procedure called padding.

We often don't care about padding since we do want to reduce the spatial resolution of the input.



Successive convolution with a tap-3 kernel. l for tap-3 kernel is 1 ($\because 2l+1=3$); therefore, each output is 2 less than the input.

Convolutional layers used in many DL architectures.



$$y_{ij} = \sum_{i'} \sum_{j'} \sum_{c'} w_{i'j'c'} \; x_{i-i' \; j'-j \; c'-c}$$

This can be extended to deal with the situation when the output has many channels.

$$y_{ijf} = \sum_{i'} \sum_{j'} \sum_{c'} w_{i'j'c'f} \; x_{i-i' \; j'-j \; c'-c}$$

Lets consider the 1D convolution case and see if we can derive the backpropagation rules.

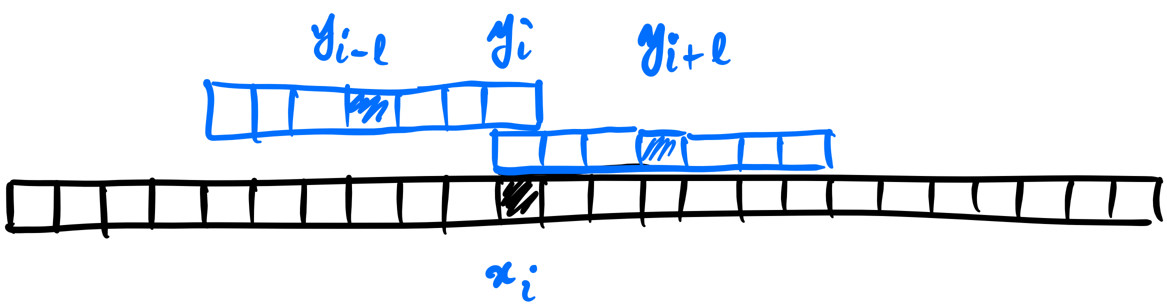
$$y_i = \sum_{i'=-l}^{+l} w_{i'} x_{i-i'}$$

$$\Rightarrow \quad y_i = w_{-l} x_{i+l} + w_{-l+1} x_{i+l-1} + \cdots + w_l x_{i-l}$$

Recall that we already have $\frac{\partial c}{\partial Y}$. Lets apply the chain-rule

$$\frac{\partial c}{\partial X} = \frac{\partial c}{\partial Y} \cdot \frac{\partial Y}{\partial X} \quad \text{and} \quad \frac{\partial c}{\partial W} = \frac{\partial c}{\partial Y} \frac{\partial Y}{\partial W}$$

$$\frac{\partial y_i}{\partial x_{i+l}} = w_{-l} \quad \cdots \quad \frac{\partial y_i}{\partial x_{i-l}} = w_l$$



Notice how x_i influences outputs between y_{i-l} and y_{i+l}. So we need to backpropagate gradients through these when computing $\frac{\partial c}{\partial x_i}$.

$$\frac{\partial c}{\partial x_i} = \frac{\partial c}{\partial y_{i-l}} \frac{\partial y_{i-l}}{\partial x_i} + \cdots + \frac{\partial c}{\partial y_{i+l}} \frac{\partial y_{i+l}}{\partial x_i}$$

Now lets try to find what $\frac{\partial y_{i-\ell}}{\partial x_i}$ and $\frac{\partial y_{i+\ell}}{\partial x_i}$ look like:

$$\frac{\partial y_{i-\ell}}{\partial x_i} = w_\ell \quad \text{and} \quad \frac{\partial y_{i+\ell}}{\partial x_i} = w_{-\ell}$$

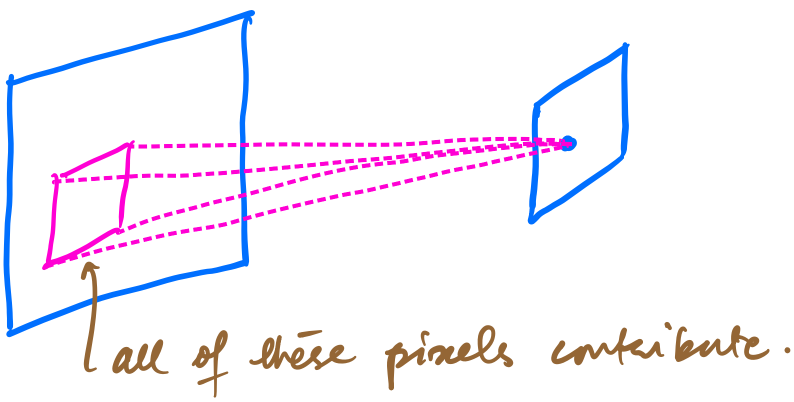
We can therefore write:

$$\frac{\partial e}{\partial x_i} = \begin{bmatrix} \frac{\partial e}{\partial y_{i-\ell}} & \cdots & \frac{\partial e}{\partial y_{i+l}} \end{bmatrix} \begin{bmatrix} w_\ell \\ \vdots \\ w_{-\ell} \end{bmatrix}$$

But what about other x_i s. We can similarly construct relevant $\frac{\partial e}{\partial y_{i-\ell}}$.

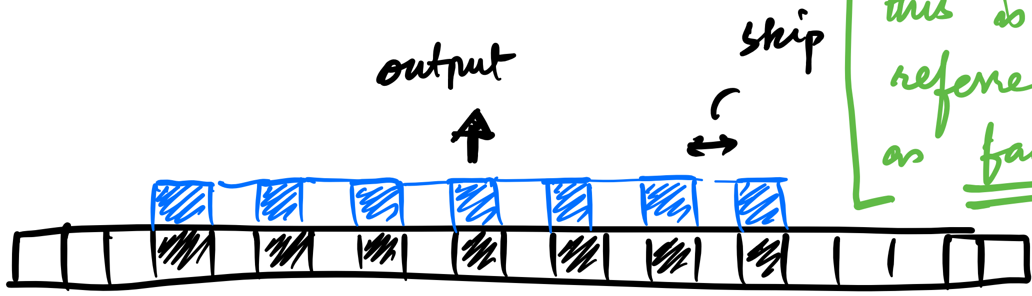
Other considerations

1. Don't forget to add a non-linearity
2. Field of regard.



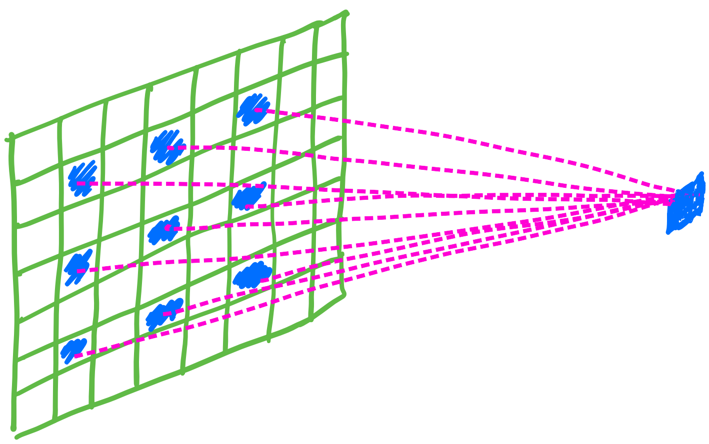
3. Good at capturing local, spatial content but what about global context? How can we extend convolution to also include global context?

Use atrous or dilated convolutions.



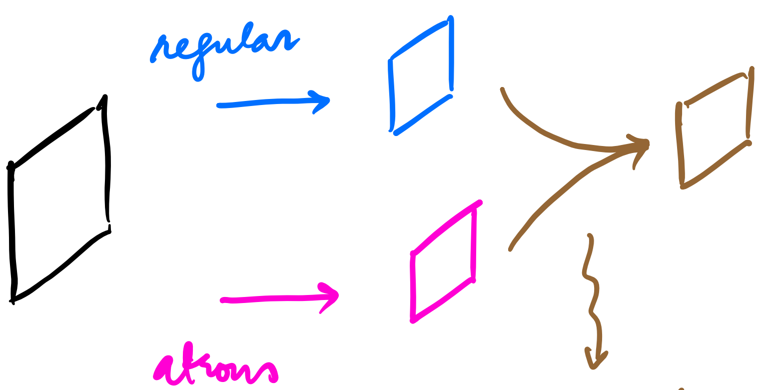
[this is also referred to as factor.]

Notice how the output includes information from pixels that are further away while keeping the number of parameters constant.

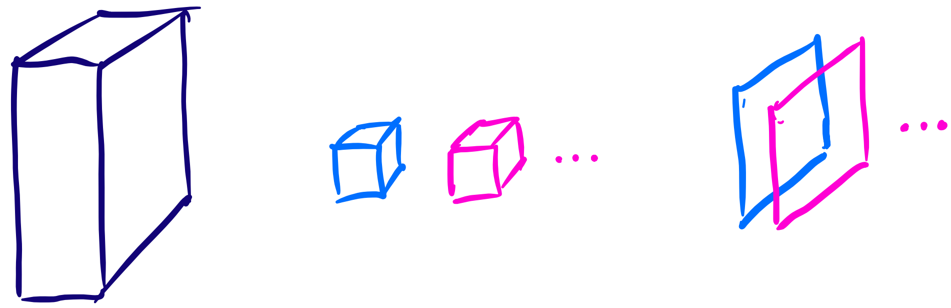


We can easily combine atrous and "regular" convolutions to create an output.

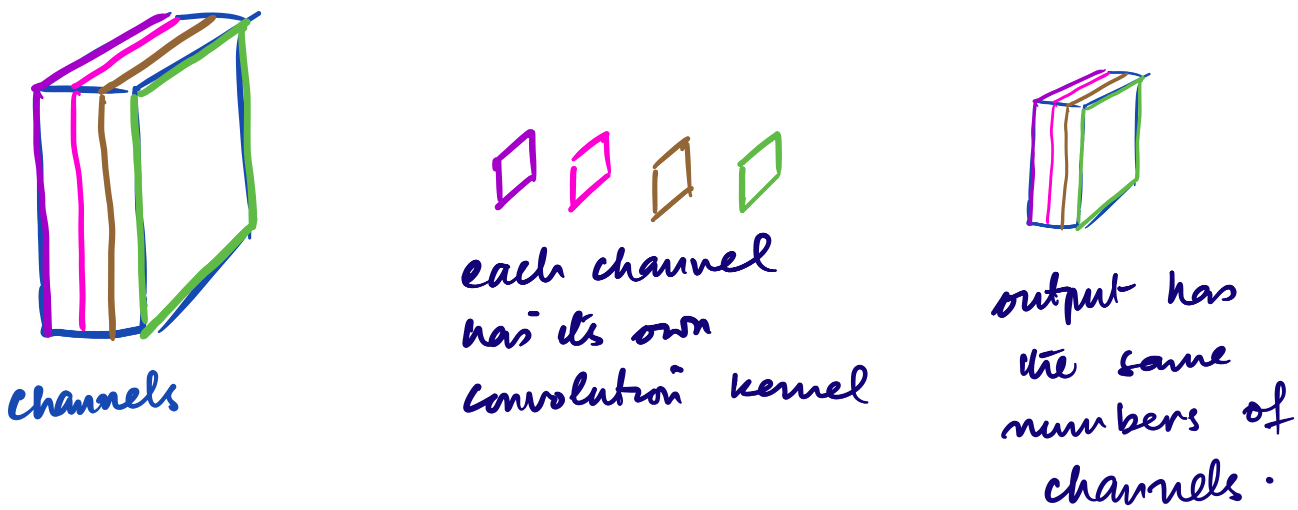
- Used often in real-time image segmentation.

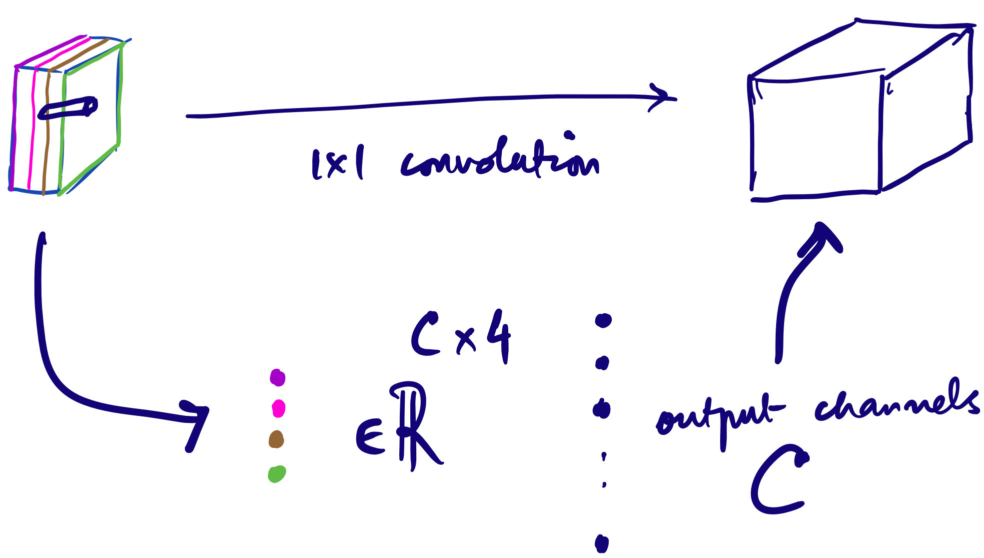


- concatenate
- sum
- linear combination



This convolution mixes information along space (i.e., height and width) and channels.
We can borrow ideas from separable convolution and separate out spatial information mixing and channel-wise information mixing.





1-d convolution is often used to increase channel-depth of a feature while maintaining its spatial resolution.

Benefits of separable convolution

Consider an input image: $32 \times 32 \times 3$.

We have a convolution kernel of size $3 \times 3 \times 3$ and we want the output to be of size $30 \times 30 \times 12$.

Therefore, we need 12 $(3 \times 3 \times 3)$ kernels.

\# total number of multiplications

$$= ((30 \times 30) \times (3 \times 3 \times 3)) \times 12$$

$$= 291600$$

Now say we separate out the channels and spatial dimensions

1. Convolve each input channel with a 3×3 kernel.

   \# multiplications $= ((30 \times 30) \times (3 \times 3)) \times 3$

   $= 24,300$

2. Now multiply each location with 12×3 matrix

   \# multiplications $= (30 \times 30) \times (12 \times 3)$

   $= 32,400$

4. Total $= 24{,}300 + 32{,}400$

$= 56{,}700$

What about the number of parameters:

\# parameters in the non-separated case

$= (3\times3\times3)\times 12 = 324$

\# parameters in the separated case

$= (3\times3)\times12 + 3\times12$

$= 108 + 36$

$= 144$

Transposed Convolution (a.k.a de-convolution)

Convolution operation reduces the spatial dimension as we go deeper. This allows convolutional layers to construct an abstract representation of the (entire) image. This is great for image-level decision making, such that image classification.

However, what if you want to make pixel-level decision making, e.g., semantic segmentation or superresolution?

We need a mechanism to maintain the spatial resolution.

How do we derive the benefits of convolutional layers (local processing and spatial structure) and maintain the spatial resolution?

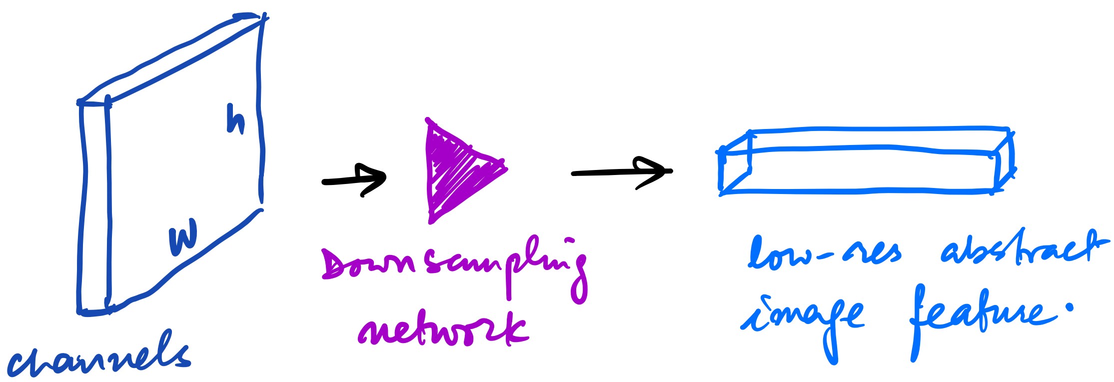
① Use padding to maintain the spatial resolution.

↓ Downside: increased computation cost.

② Combine a downsampling network with an upsampling network. (Encoder-Decoder Architecture)

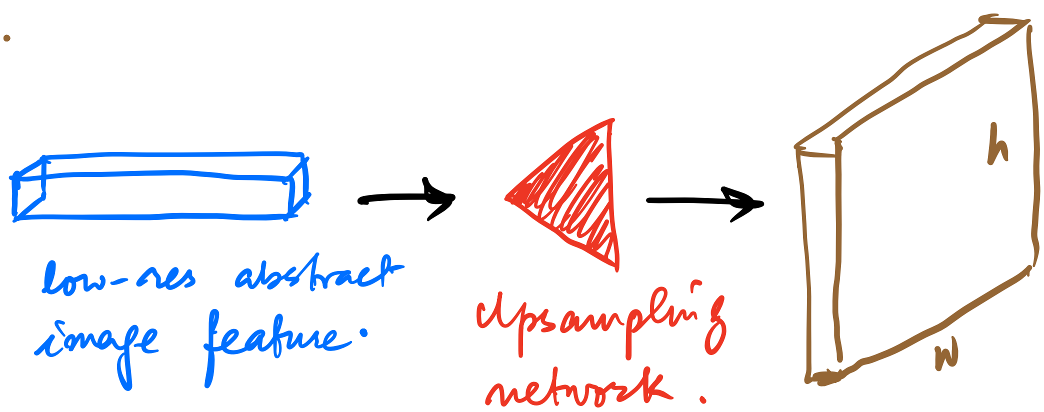
Downsampling network

CNN network that creates abstract representation of an image. The abstract representation is low-resolution.



Upsampling network

Takes a low-res abstract image feature and reconstructs a feature having the same spatial resolution as the image.



Q. How do we upsample?

① Nearest neighbour

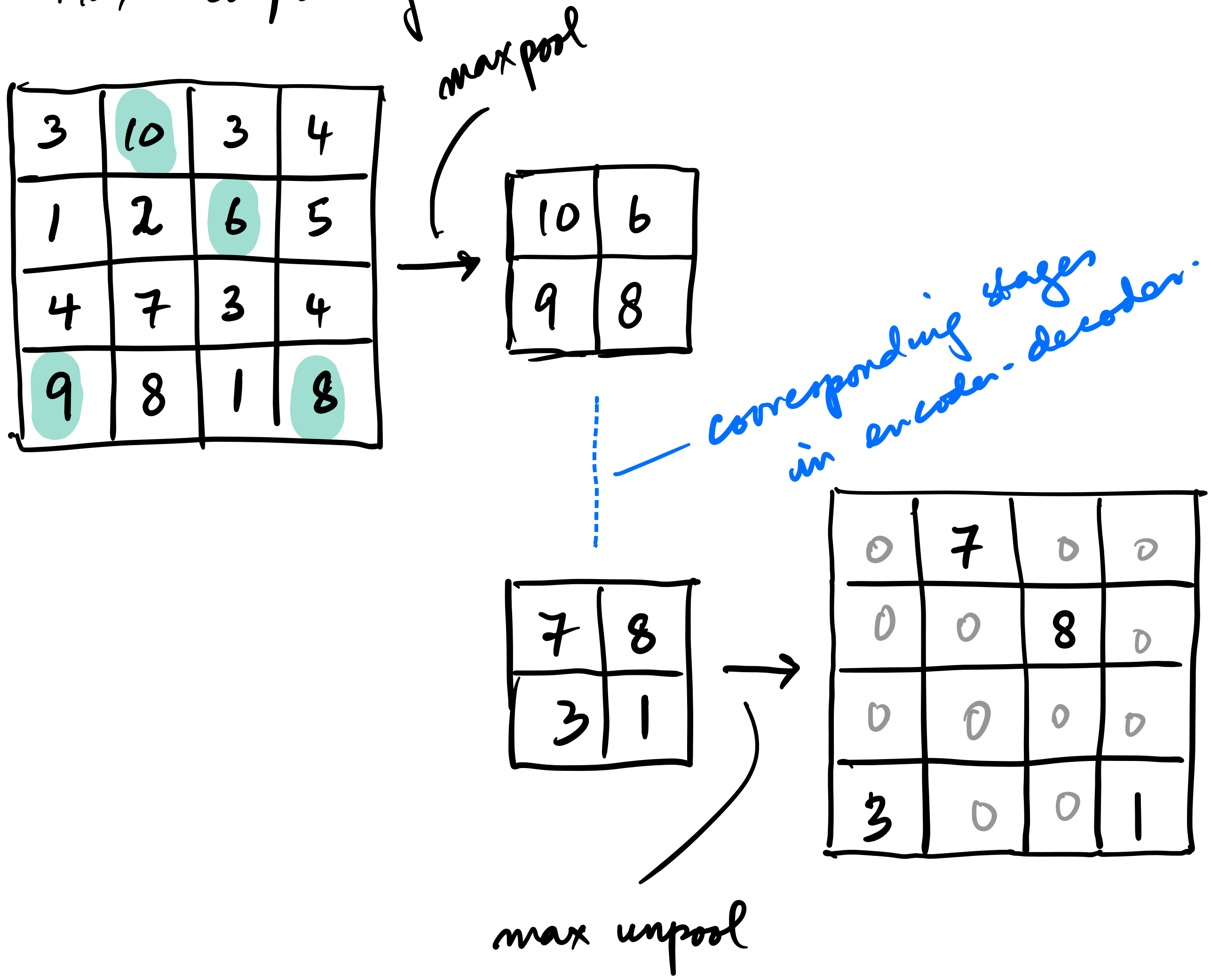
$$\begin{bmatrix} 13 & 1 \\ 4 & 0 \end{bmatrix} \xrightarrow{NN} \begin{bmatrix} 13 & 13 & 1 & 1 \\ 13 & 13 & 1 & 1 \\ 4 & 4 & 0 & 0 \\ 4 & 4 & 0 & 0 \end{bmatrix}$$

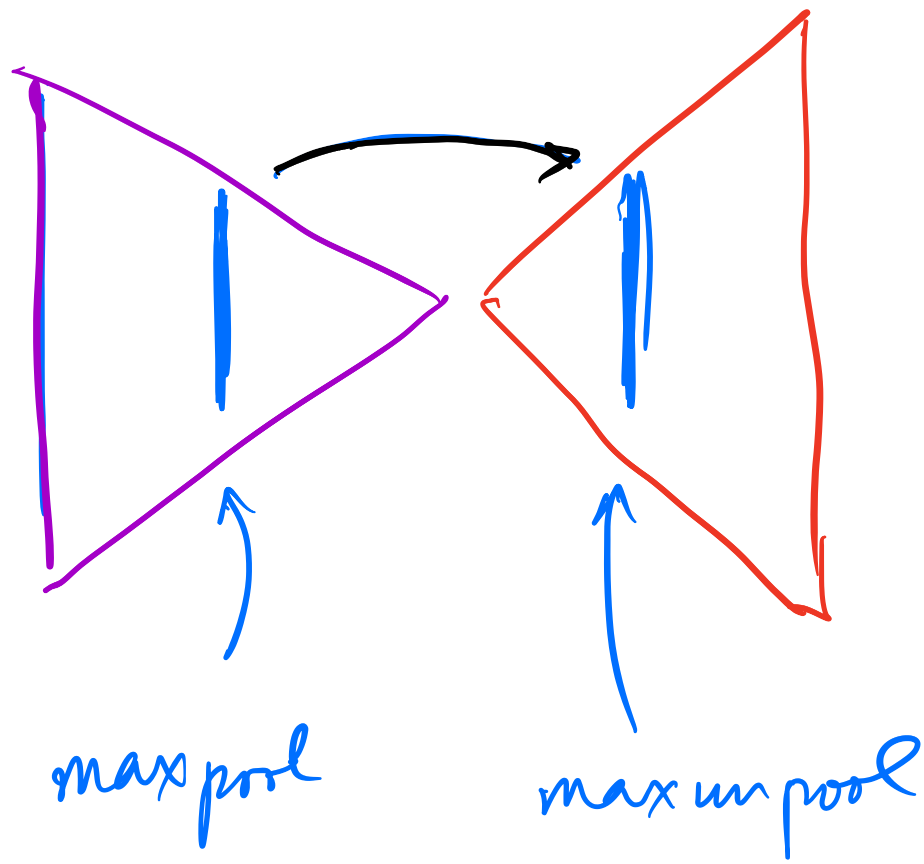
② Bi-linear interpolation

③ Bed of nails

$$\begin{bmatrix} 13 & 1 \\ 4 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 13 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

④ Max-unpooling.





None of these techniques are data-dependent.
So these donot learn from data.

(5) Transposed Convolution

Say we want to upsample a 2×2 feature map to a 3×3 feature map?

$$\begin{bmatrix} 3 & 4 \\ 1 & 7 \end{bmatrix} \longrightarrow \overset{?}{\begin{bmatrix} \; & \; & \; \\ \; & \; & \; \\ \; & \; & \; \end{bmatrix}}$$

Lets take a 2×2 kernel with unit stride and zero padding

kernel : $\begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}$

Now take every element of the feature map and multiply it with kernel and save the results as follows.

$$3 \times \begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 6 & 0 & \; \\ 3 & -3 & \; \\ \; & \; & \; \end{bmatrix}$$

$$4 \times \begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} \; & 8 & 0 \\ \; & 4 & -4 \\ \; & \; & \; \end{bmatrix}$$

We can now extend this idea as follows:

$$\begin{array}{|c|c|}\hline 3 & 4 \\ \hline 1 & 7 \\ \hline \end{array} \; \begin{array}{|c|c|}\hline 2 & 0 \\ \hline 1 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 6 & 0 & \\ \hline 3 & -3 & \\ \hline & & \\ \hline \end{array}$$

$$+ \begin{array}{|c|c|c|}\hline & 8 & 0 \\ \hline & 4 & -4 \\ \hline & & \\ \hline \end{array}$$

$$+ \begin{array}{|c|c|c|}\hline & & \\ \hline 2 & 0 & \\ \hline 1 & -1 & \\ \hline \end{array}$$

$$+ \begin{array}{|c|c|c|}\hline & & \\ \hline & 14 & 0 \\ \hline & 7 & -7 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|}\hline 6 & 8 & 0 \\ \hline 5 & 15 & -4 \\ \hline 1 & 6 & -7 \\ \hline \end{array}$$