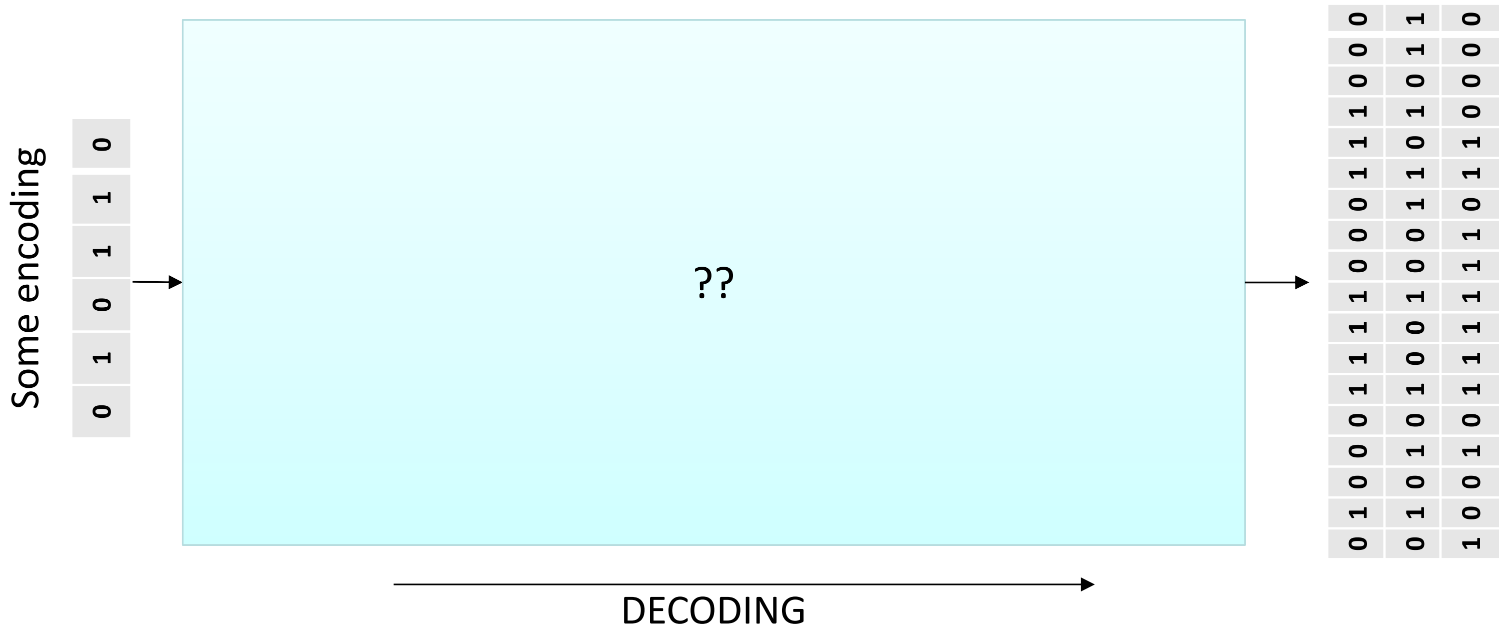
Neuroengineering (I)
6. Encoder-Decoder networks

- **Scuola di Ingegneria Industriale e dell'Informazione** – Politecnico di Milano
- Prof. Pietro Cerveri

Generative process



- Process of decoding the information represented into a vector whose elements are extracted from predefined “alphabet”
- How uncompressing/decode the encoded information?
    - Complex information requires appropriate transforms
    - Single (interpolation) against progressive decoding (multiple sequential transforms)

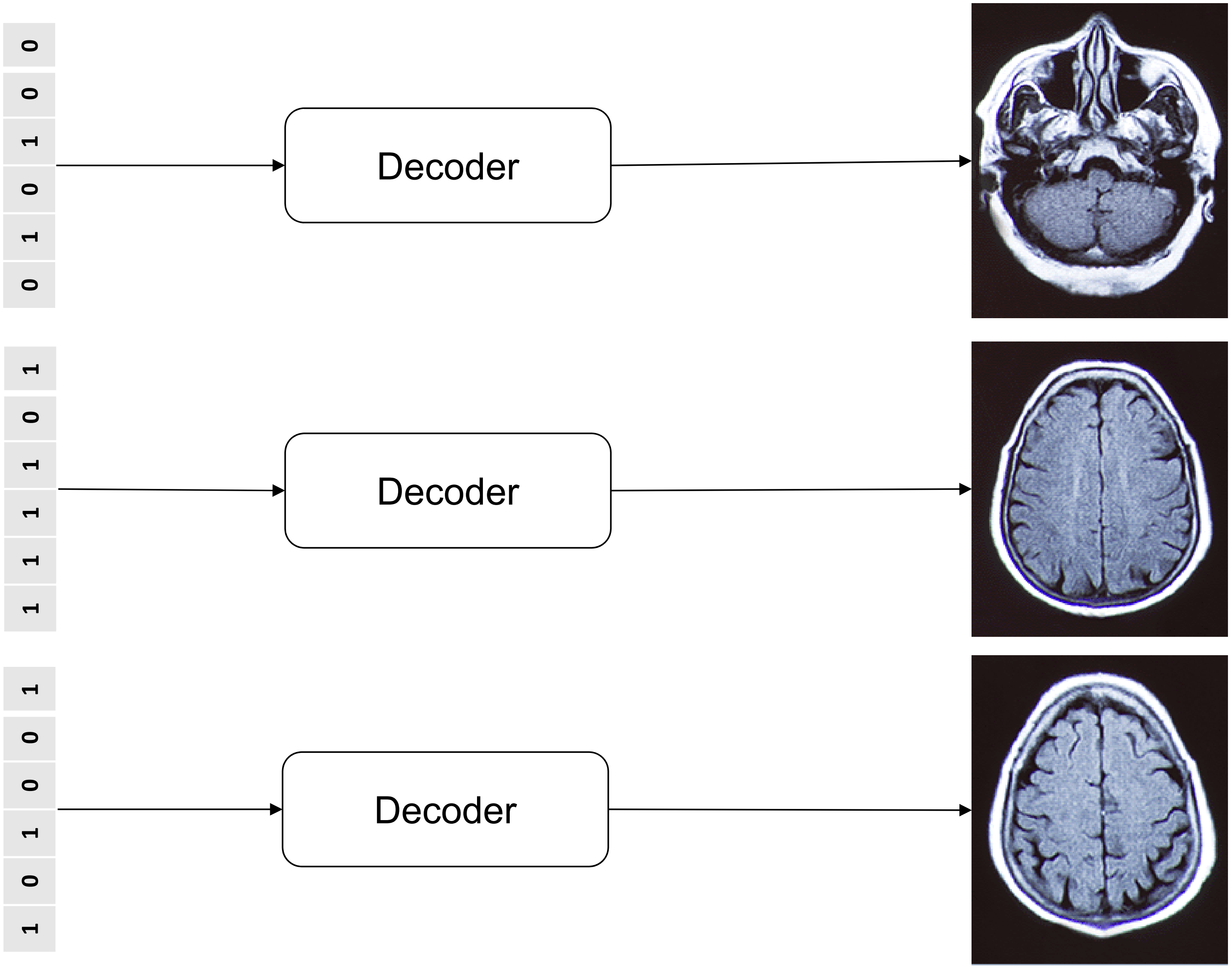
Up-sampling as a decoding process

Encoding: spatial compressed data

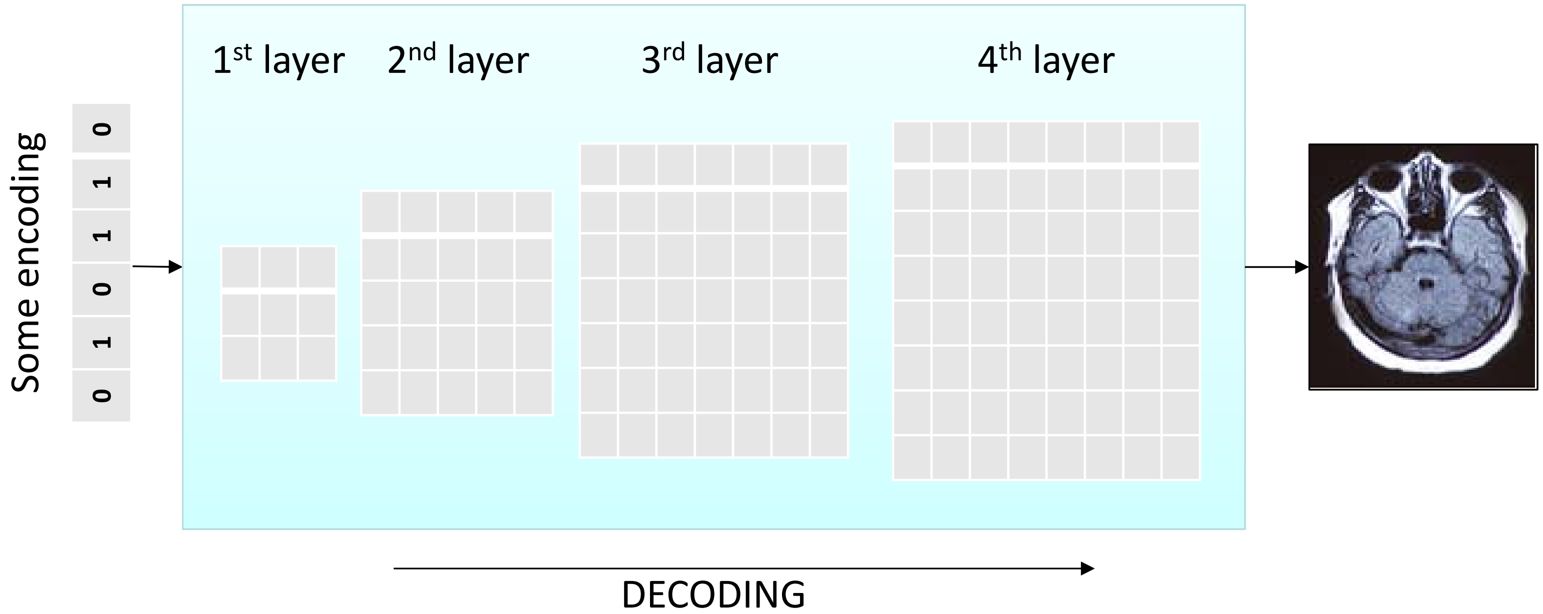
Up-sampling from a low-resolution image
towards an high resolution image

interpolation map (nearest neighbour, linear, bi-linear)
one-to-many relationship

Encoding alphabet



AN networks for decoding data



- Deep learning approach to optimal up-sampling: transposed convolution (deconvolution) which does not use a predefined interpolation method
- Learning will allow to tuning of the parameters (weights of the neurons)

Filter

-1	1	-1
1	2	1
-1	1	-1

Input image

2	2	1	1
2	2	3	1
4	5	1	1
4	1	0	4

Revise 2D convolution

3x3 filter

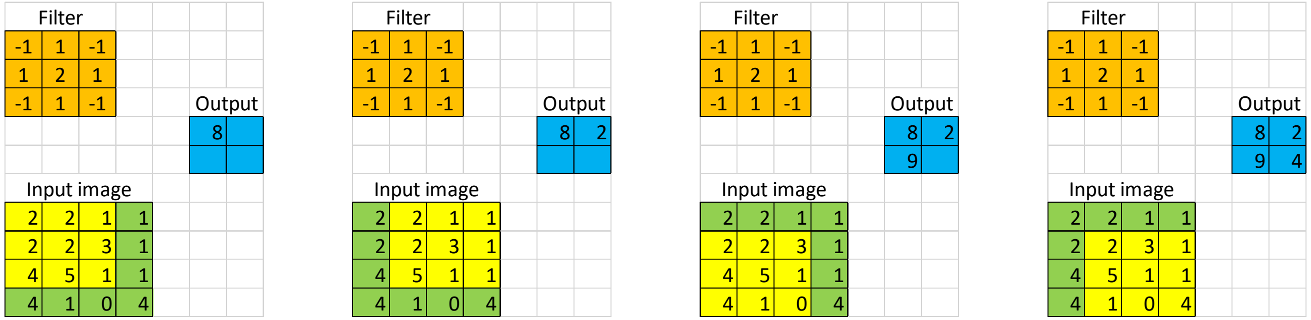
4x4 input image

Stride length: 1

No padding

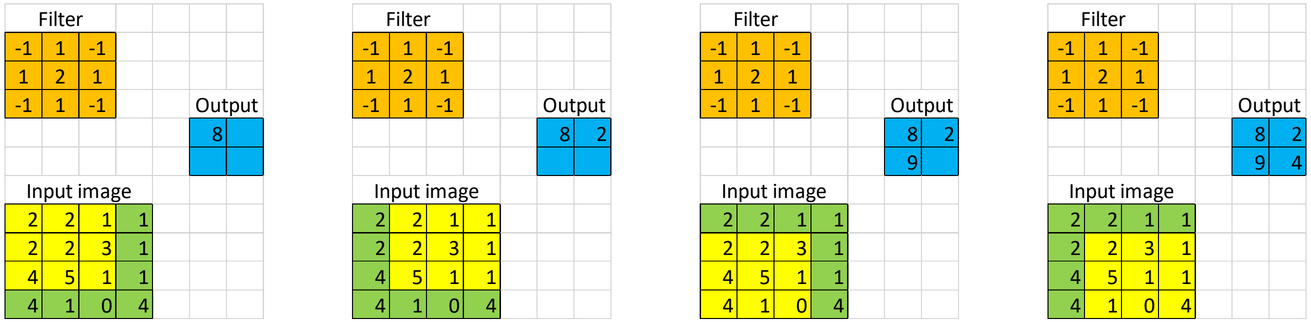
Output size??

2D Convolution, no padding, stride: 1



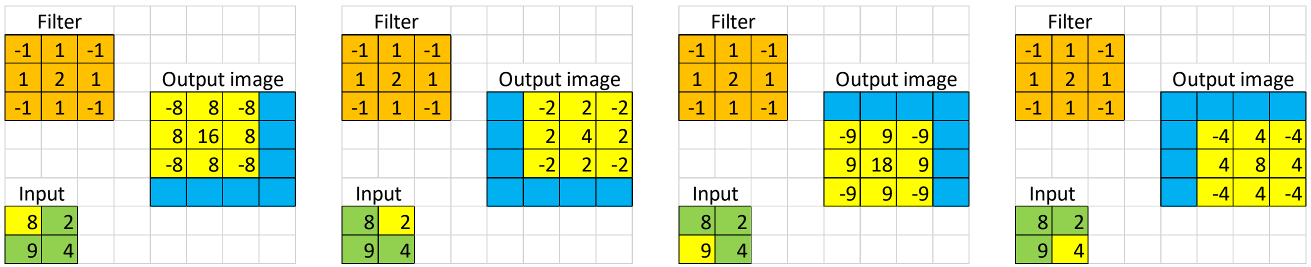
the 3×3 filter is used to connect the 9 values in the 4×4 input matrix to 1 value in the 2×2 output matrix

2D Convolution, no padding, stride: 1



the 3×3 filter is used to connect the 9 values in the 4×4 input matrix to 1 value in the 2×2 output matrix

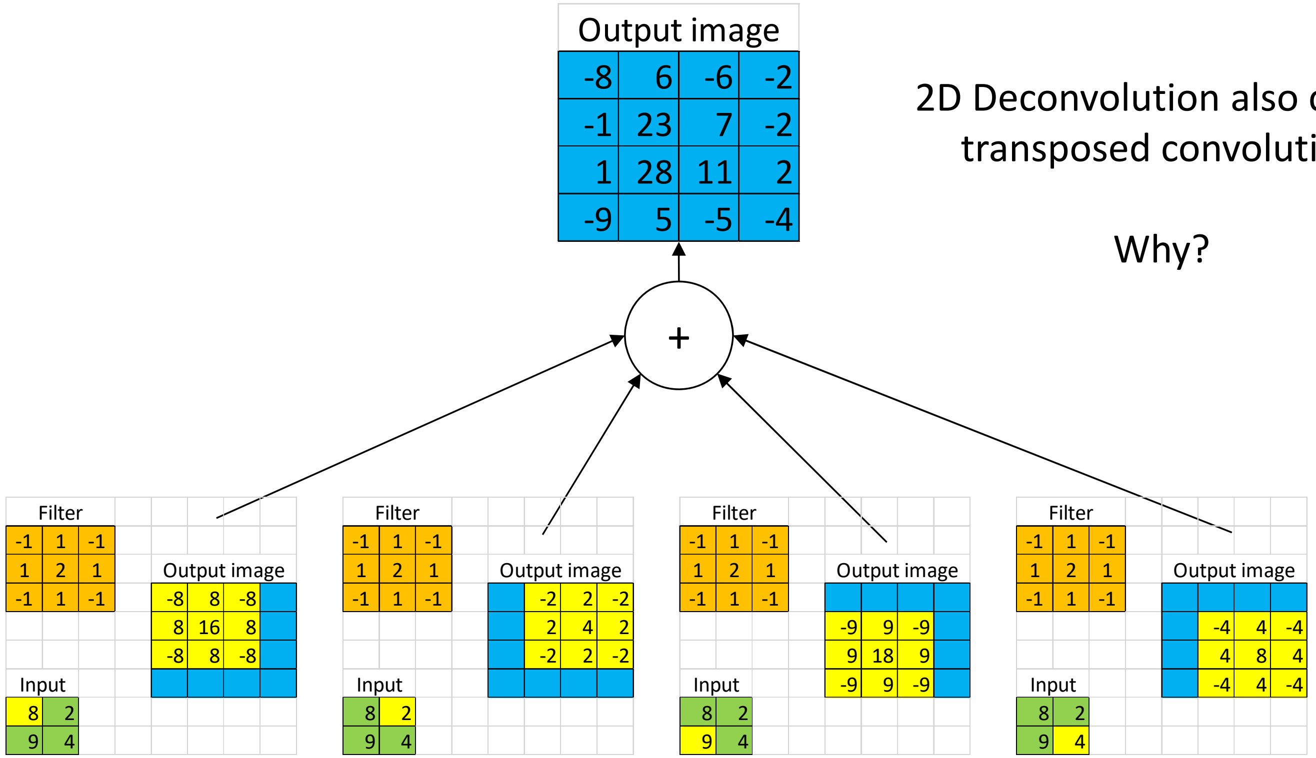
Going backward (pixel-wise multiplication, stride 1)



the 3×3 filter is used to up-sampling the 2×2 input matrix by pixel-wise multiplication, kernel striding and summation

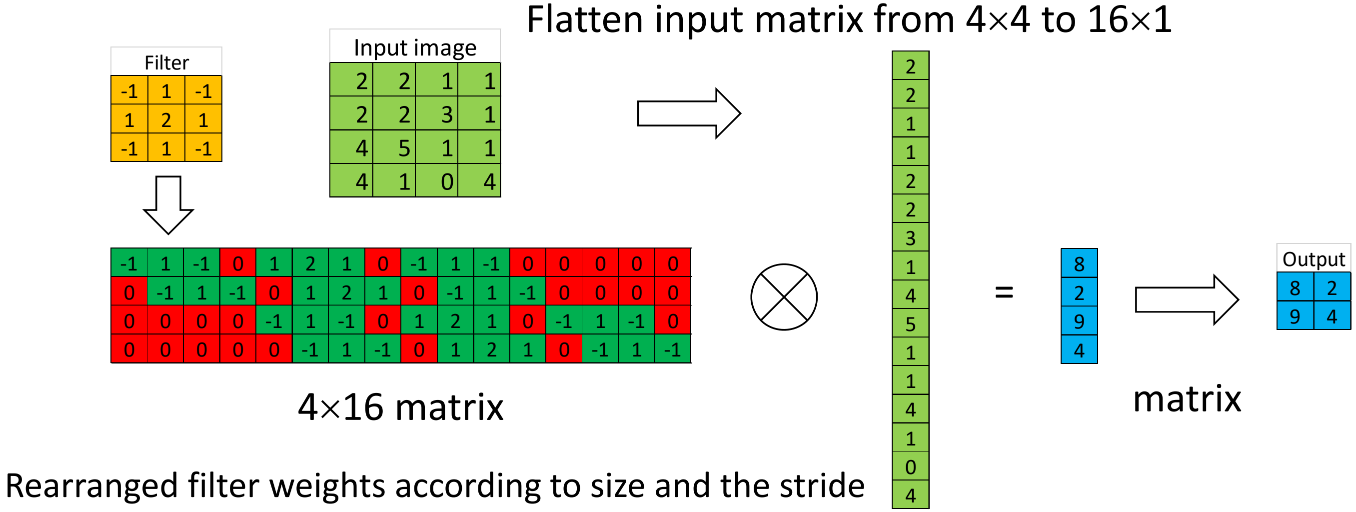
2D Deconvolution also called transposed convolution

Why?



the 3×3 filter is used to up-sampling the 2×2 input matrix by pixel-wise multiplication, kernel striding and summation

Matrix convolution



Matrix convolution

Filter

-1	1	-1
1	2	1
-1	1	-1

Input image

2	2	1	1
2	2	3	1
4	5	1	1
4	1	0	4

Flatten input matrix from 4×4 to 16×1

$$\begin{bmatrix} -1 & 1 & -1 & 0 & 1 & 2 & 1 & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 & 1 & 2 & 1 & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 & 1 & 2 & 1 & 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 & 1 & 2 & 1 & 0 & -1 & 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 1 \\ 4 \\ 5 \\ 1 \\ 1 \\ 4 \\ 1 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \\ 9 \\ 4 \end{bmatrix}$$

4×16 matrix

Rearranged filter weights according to size and the stride

matrix

Output

8	2
9	4

Matrix deconvolution (transposed convolution)

Filter

-1	1	-1
1	2	1
-1	1	-1

Flatten input matrix from 2×2 to 4×1

Input

8	2
9	4

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 2 & 1 & 1 & -1 \\ 1 & 2 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 1 & -1 & 2 & 1 \\ -1 & 1 & 1 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \otimes \begin{bmatrix} 8 \\ 2 \\ 9 \\ 4 \end{bmatrix} = \begin{bmatrix} -8 \\ 6 \\ -6 \\ -2 \\ -1 \\ 23 \\ 7 \\ -2 \\ 1 \\ 28 \\ 11 \\ 2 \\ -9 \\ 5 \\ -5 \\ -4 \end{bmatrix}$$

16×4 matrix

matrix

Output image

-8	6	-6	-2
-1	23	7	-2
1	28	11	2
-9	5	-5	-4

Remarks

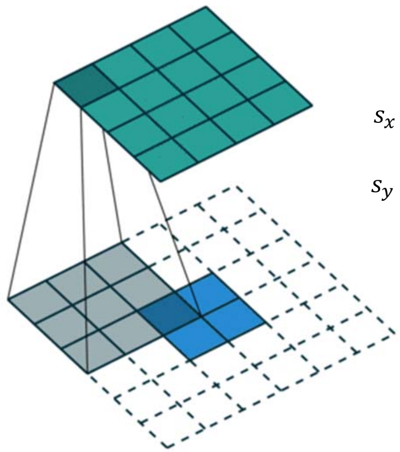
- The transposed convolution operation forms the same connectivity as the normal convolution but in the backward direction
  - one-to-many rather than many-to-one association

- As such, the transposed convolution is not a convolution. But we can emulate the transposed convolution using a convolution. We up-sample the input by adding zeros between the values in the input matrix in a way that the direct convolution produces the same effect as the transposed convolution.

- We can use it to conduct up-sampling. Moreover, the weights in the transposed convolution are learnable. So we do not need a predefined interpolation method.

Deconvolution by zero-padding the input image

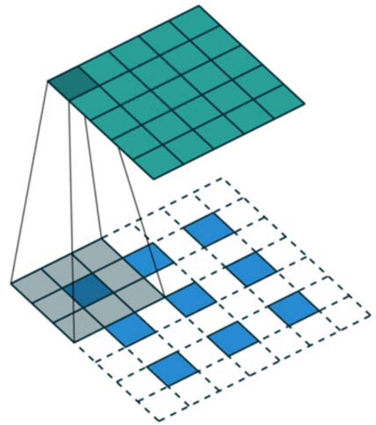
A deconvolution for one 3×3 filter with stride 1 and image (blue entries) with zero-padding of 2 (white entries)



$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1$$
$$s_y = \frac{(I_y + 2z_y - f_y)}{L_y} + 1$$

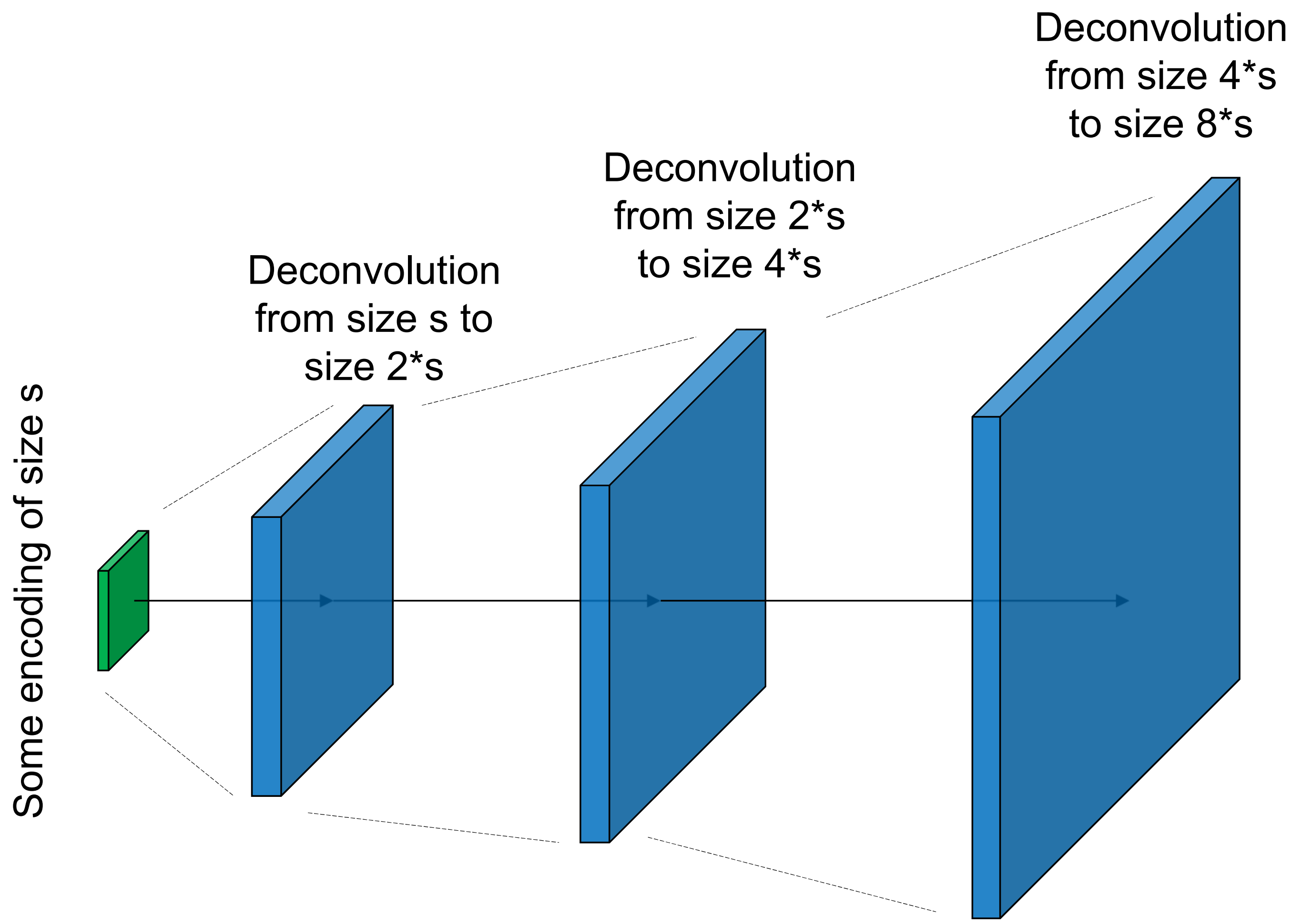
Inter-pixel zero-padding of 1, the deconvolution would look like this
Also called fractional strided convolution

$$s_x = \frac{(I_x + 4z_x - f_x)}{L_x} + 1$$
$$s_y = \frac{(I_y + 4z_y - f_y)}{L_y} + 1$$

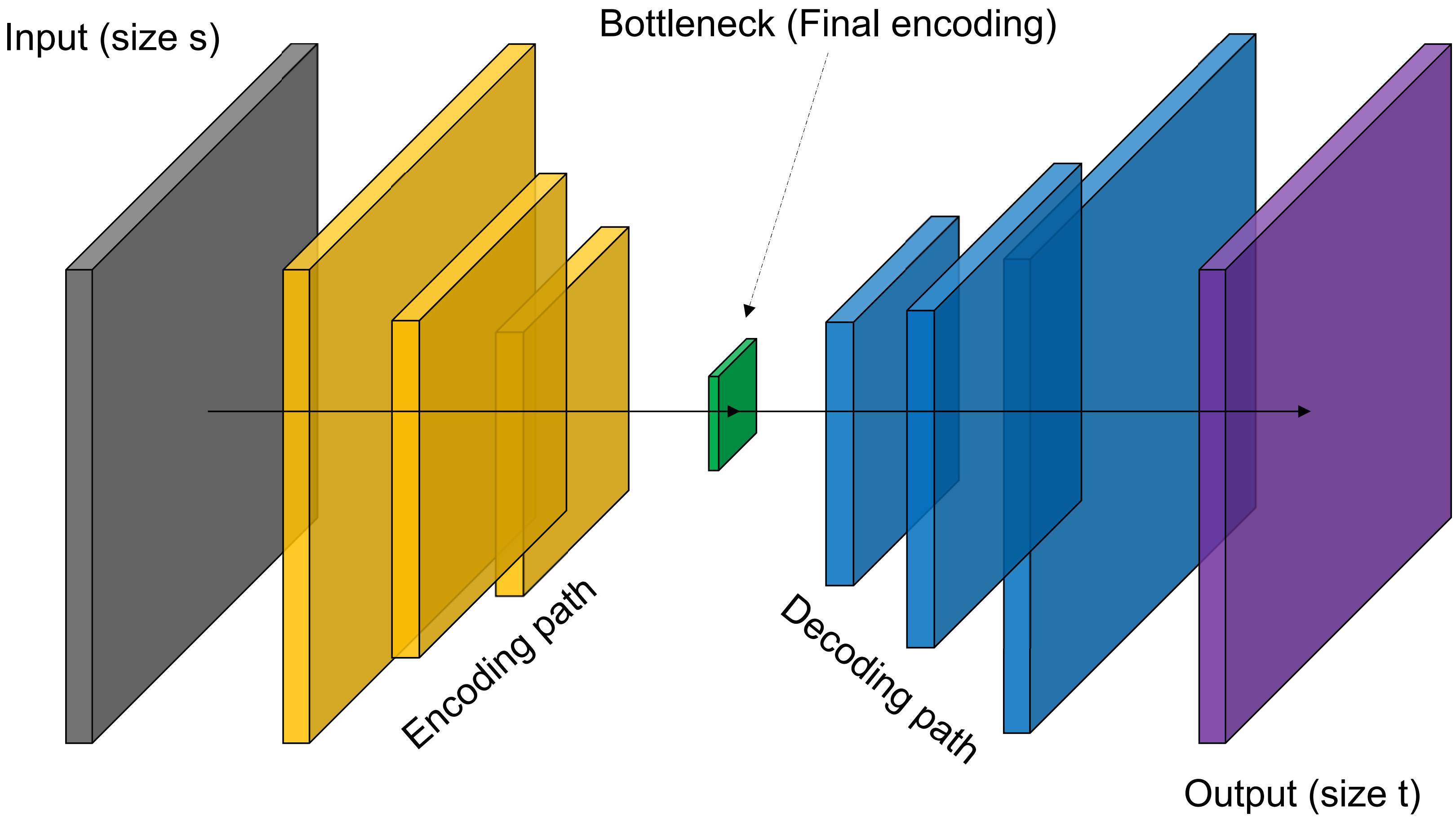


Upsampling convolution

Deconvolution networks

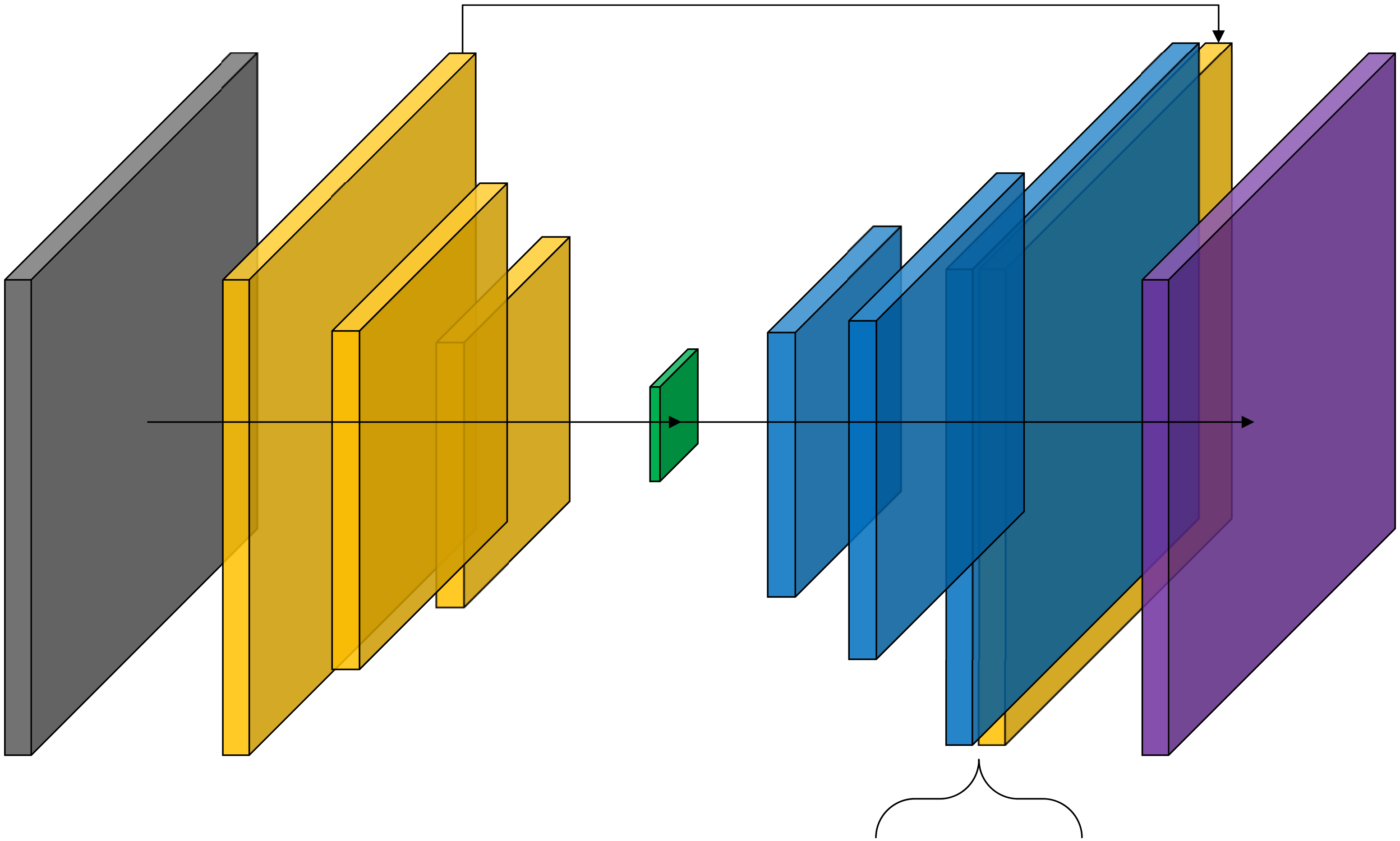


Encoder-decoder network



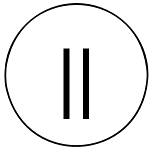
Output: Binary, Linear or Softmax (?????) layer

Unet -> Encoder-decoder network with skip connections

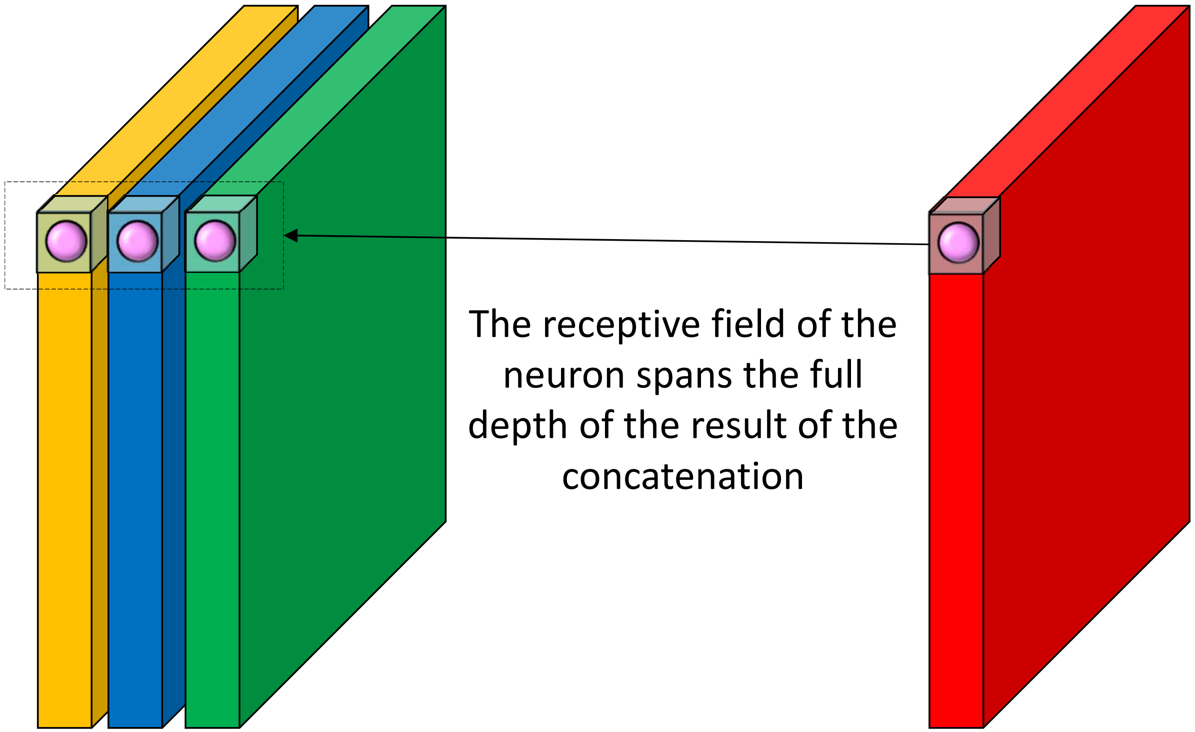


Concatenated layers

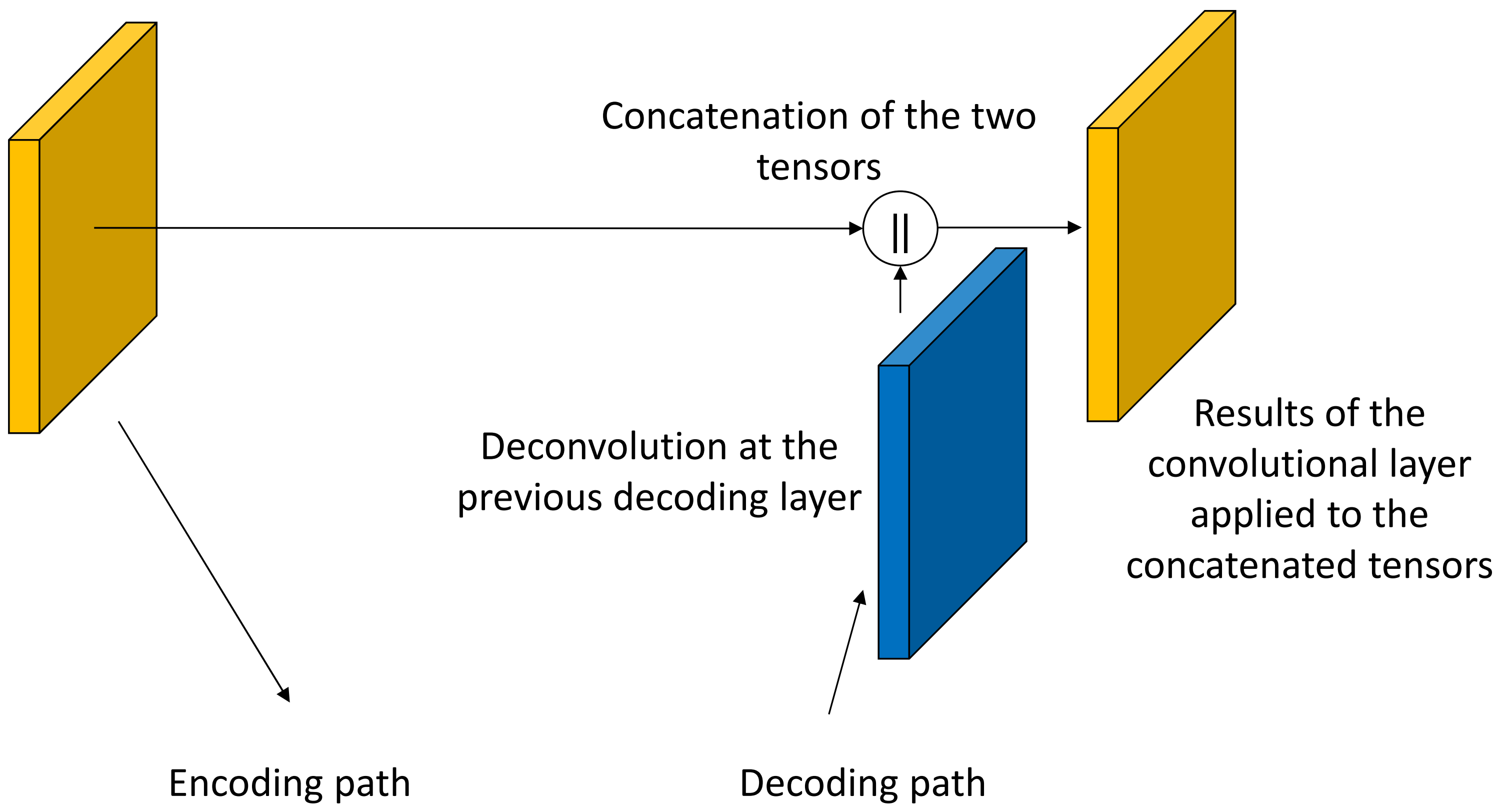
Concatenation between two or more layers



Graphical symbol for concatenation

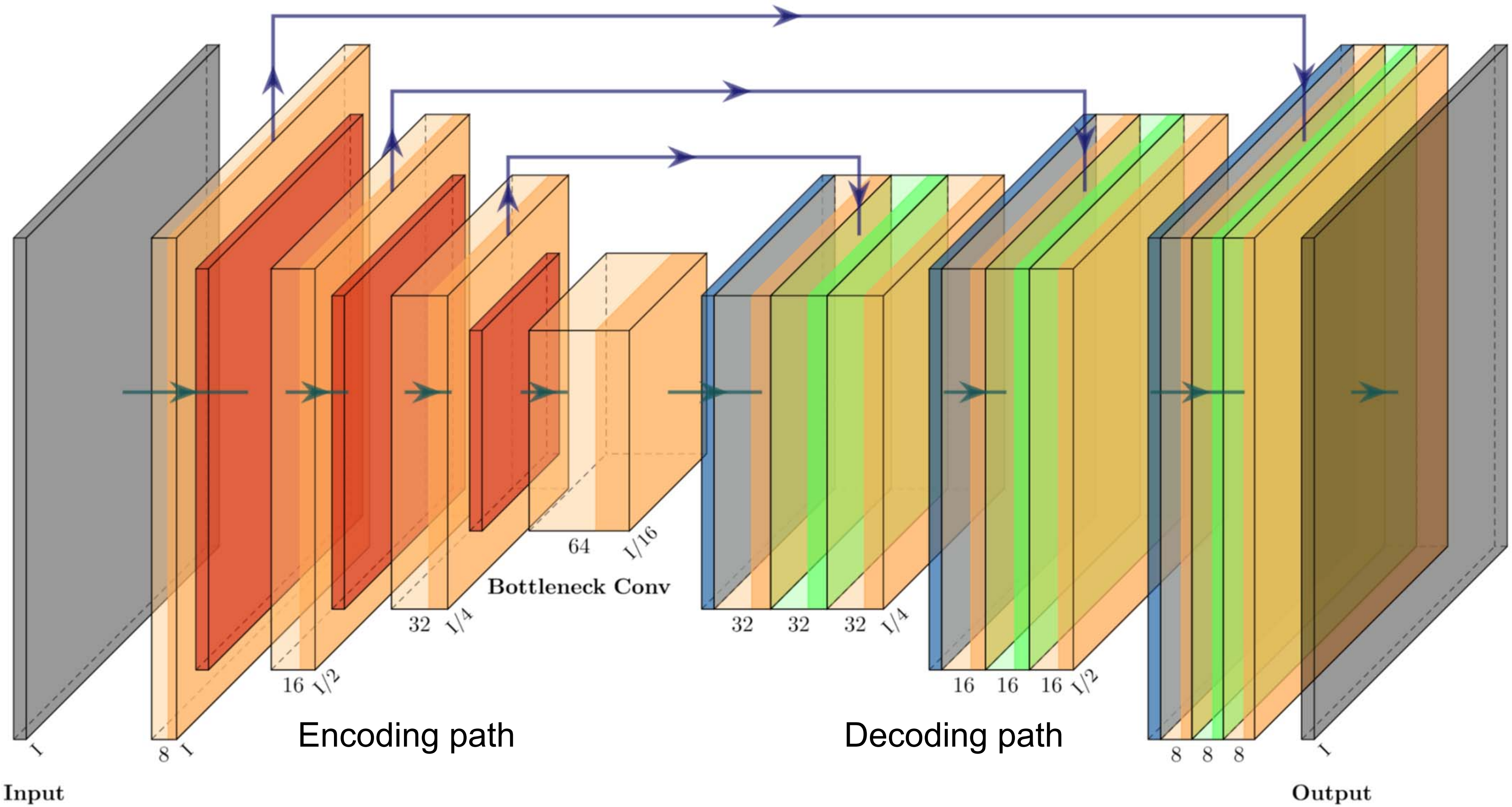


Role of the **skip connection**



Using the encoded information at the same processing level to aid the up-sampling

General Unet architecture



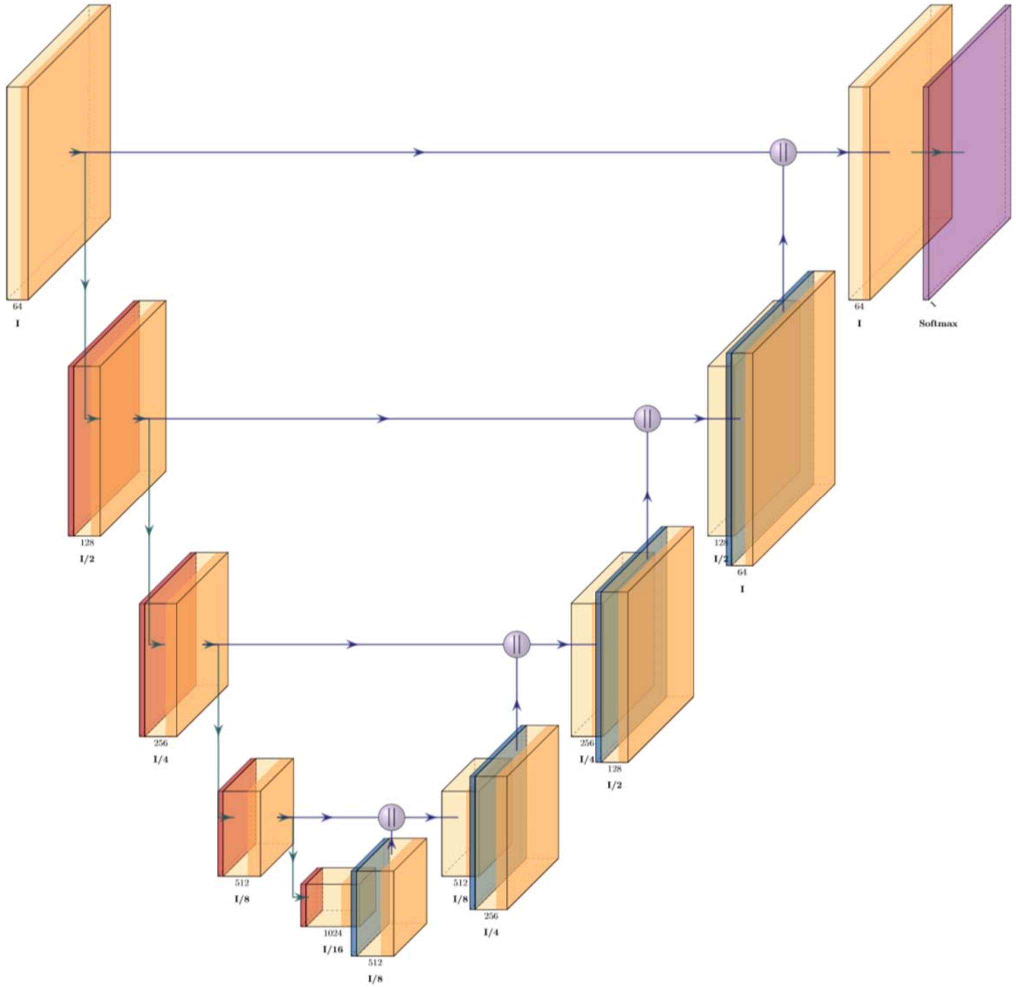
Encoding layer

```
# Convolution (i is the layer index)
X = Conv3D(nFM_i, kernel_size = (3, 3, 3), strides = (1, 1, 1), padding = 'same',activation = 'linear', )(X_from previous layer)

# ReLu
X = Activation('relu')(X)

# Skip connection
Li_X = X

# Maxpooling
X_tonextlayer = MaxPooling3D(pool_size = (2,  2,  2), strides = (2,  2,  2), padding = 'same')(X)
```

Decoding layer

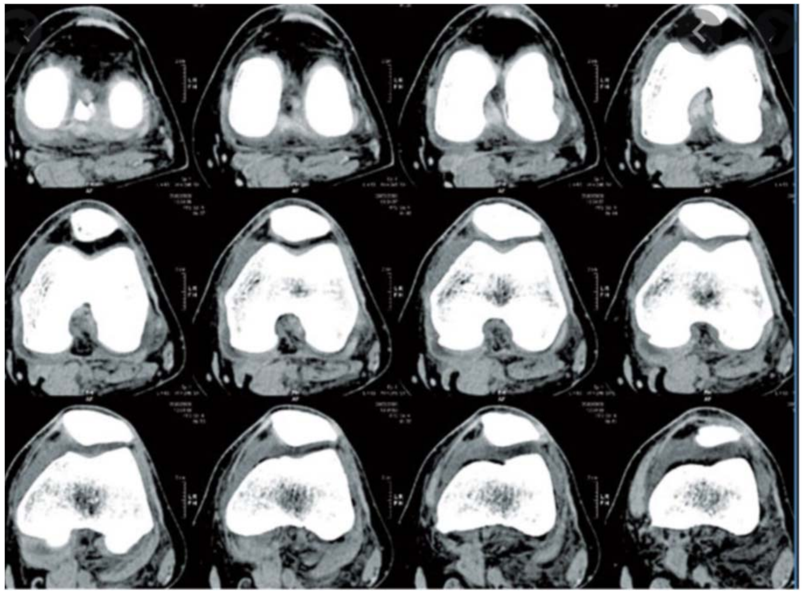
```
# Deconvolution (j is the layer index)
X = Conv3DTranspose(nFM_j, kernel_size = (2, 2, 2), strides = (2, 2, 2), padding = 'same',activation = 'linear', )(X_from previous layer)

# ReLu
X = Activation('relu')(X)

# Concatenation with Skip connection
X = Concatenate(X, Li_X)

# Convolution (j the layer index)
X = Conv3D(nFM_j, kernel_size = (3, 3, 3), strides = (1, 1, 1), padding = 'same',activation = 'linear', )(X)

# ReLu
X_tonext layer = Activation('relu')(X)
```

Another way to look at the Unet



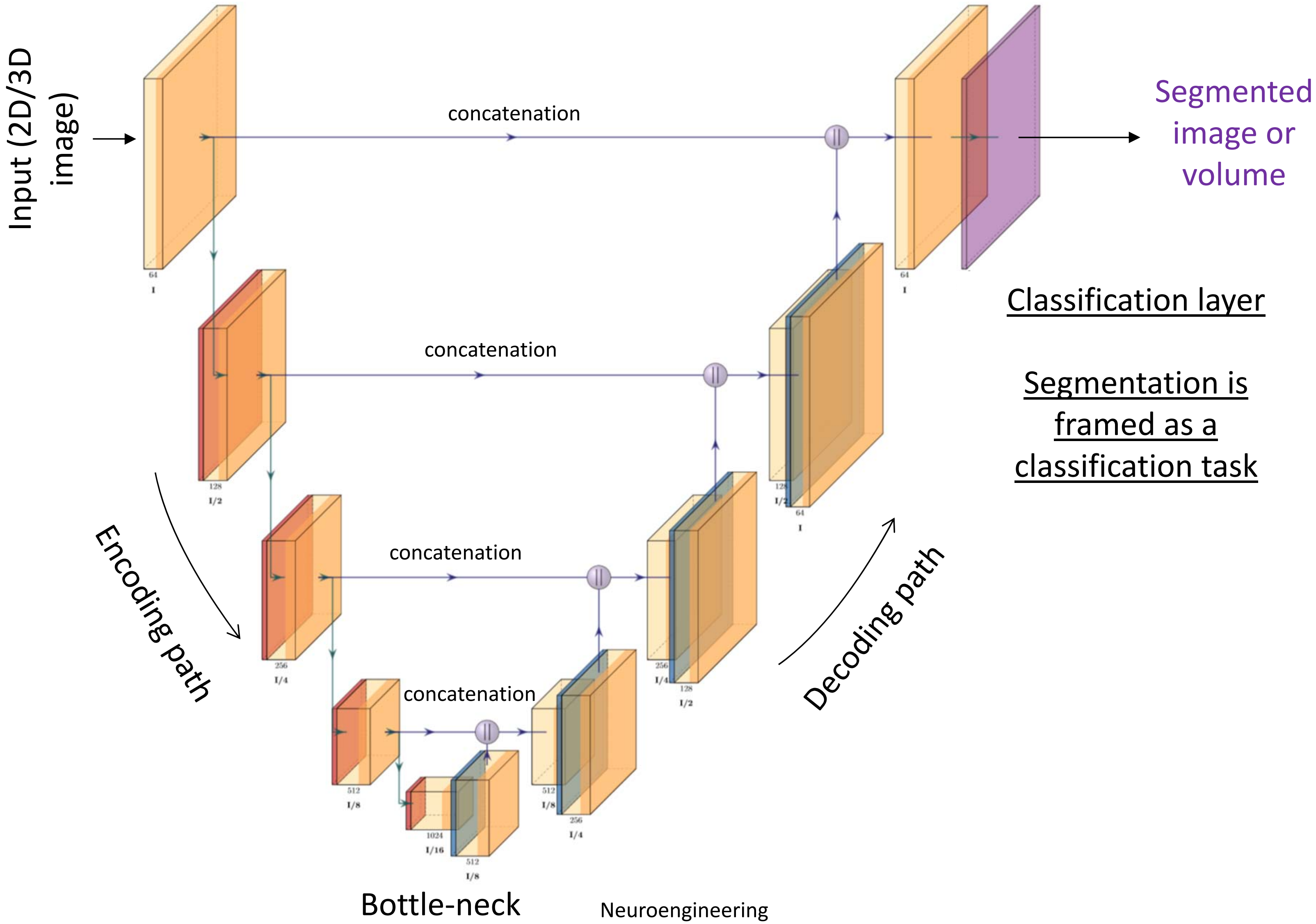
Unet in action: image segmentation

Input



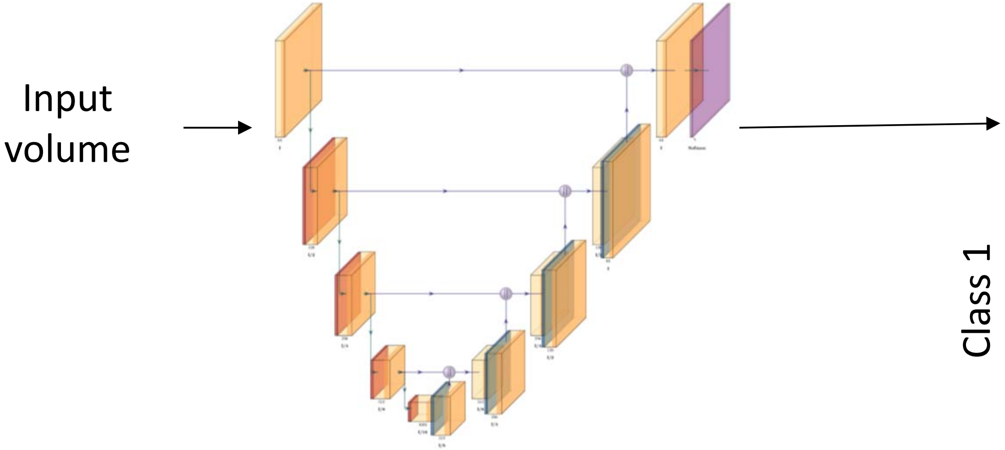
Output

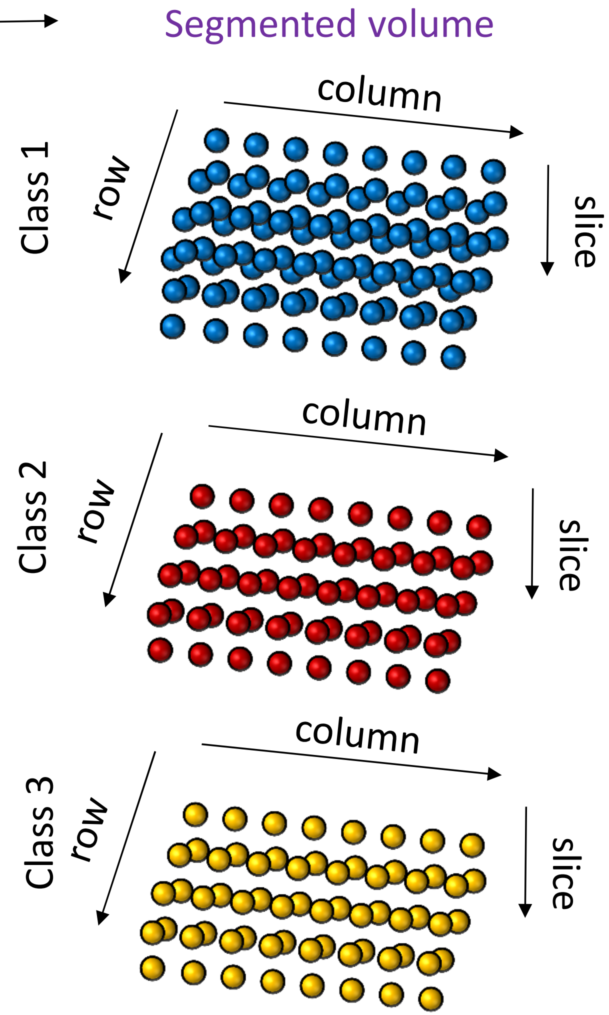
U-NET for image segmentation



Classification layer

Segmentation is framed as a classification task





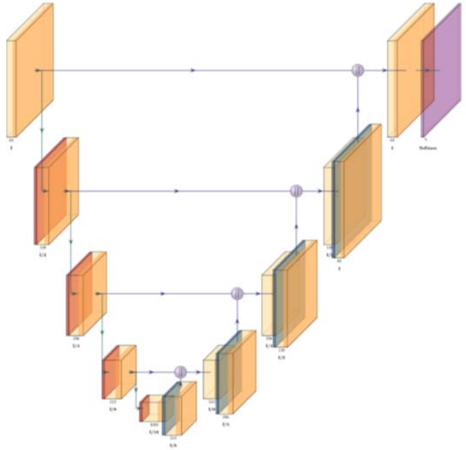
In case of binary segmentation, the classification layer is a 3D volume of sigmoidal neurons

In case of multi-label segmentation, the classification layer is a **Softmax**

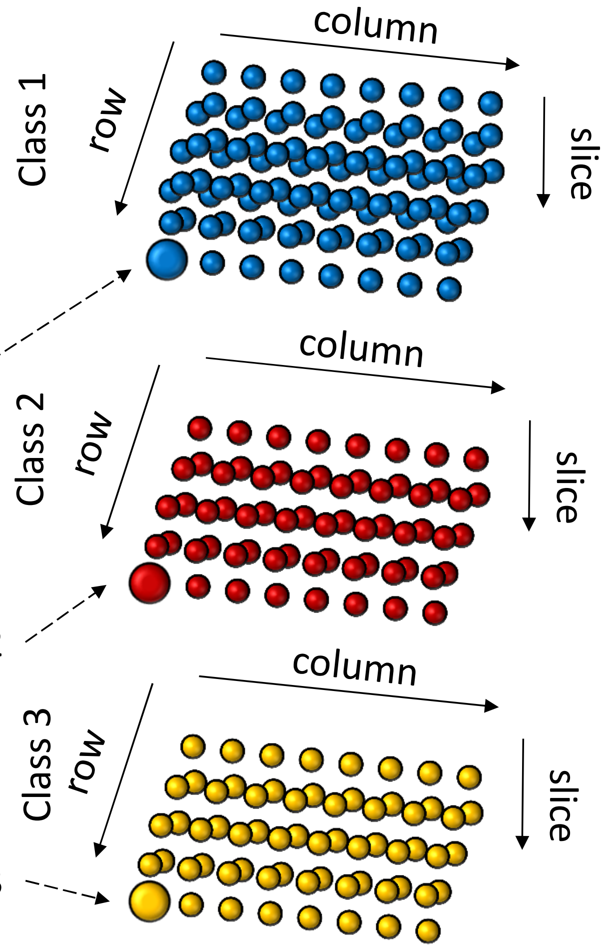
For 3D volume segmentation (the present case), the Softmax layer is a 4D tensor. For instance, considering 3 labels and r×c×s the size of the volume, the network output will be of size r×c×s×3

There will be therefore the Softmax wiring between neurons (corresponding to the same voxel)

Input
volume



Segmented volume



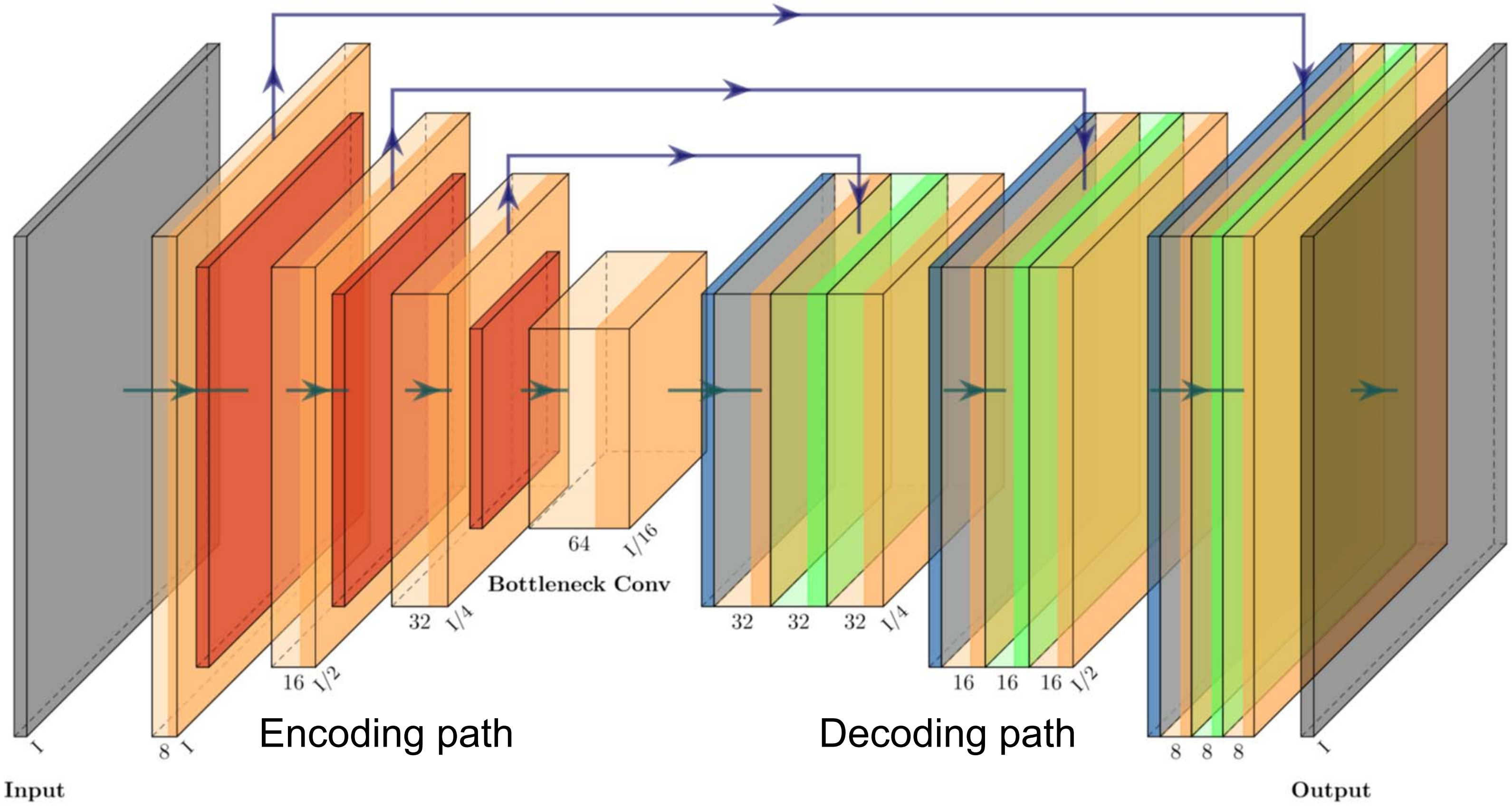
Probability of the voxel i,j,k to belong to class 1

Probability of the same voxel i,j,k to belong to class 2

Probability of the same voxel i,j,k to belong to class 3

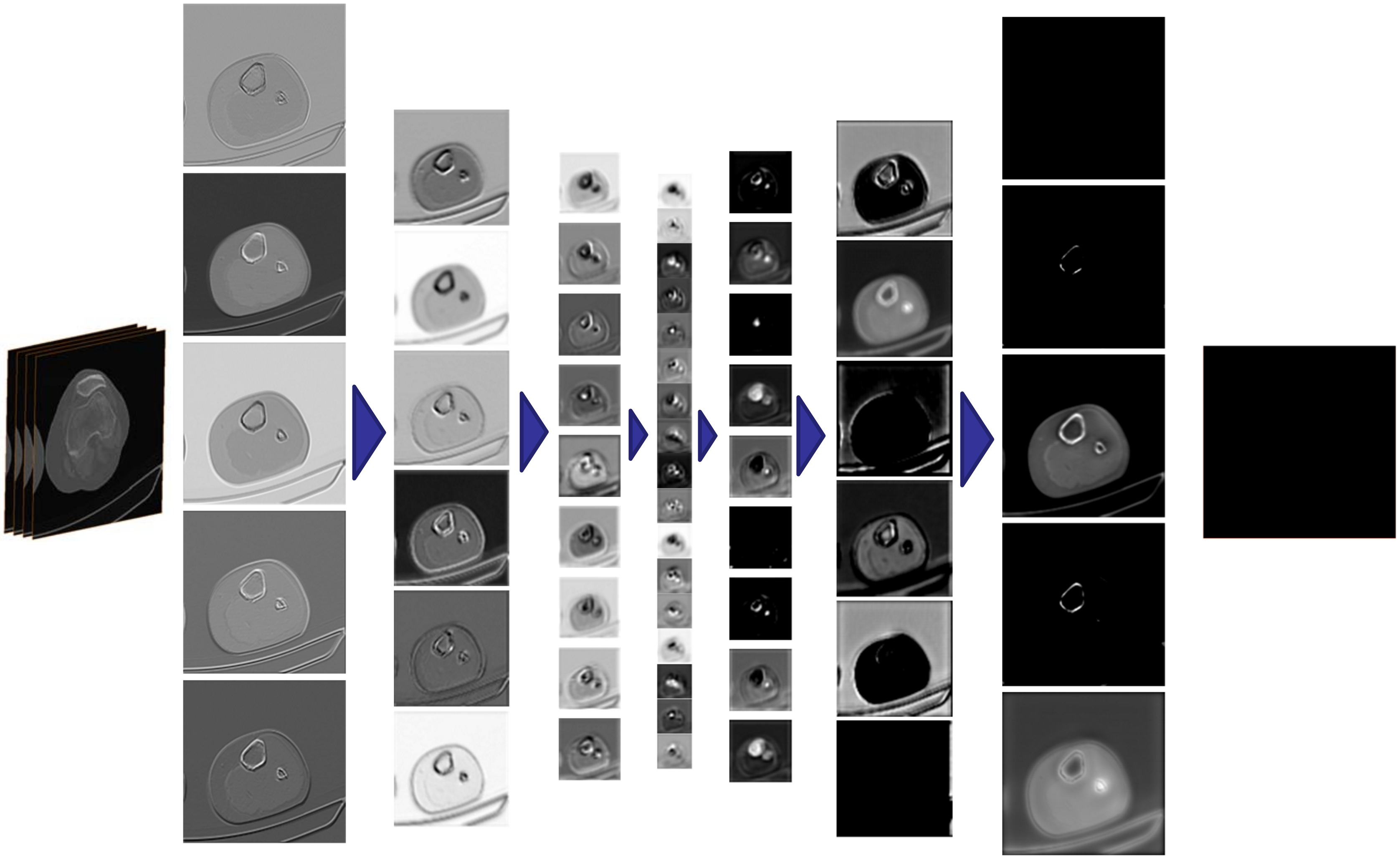
p1 + p2 + p3 = 1

U-Net for tibia and femur segmentation in the knee



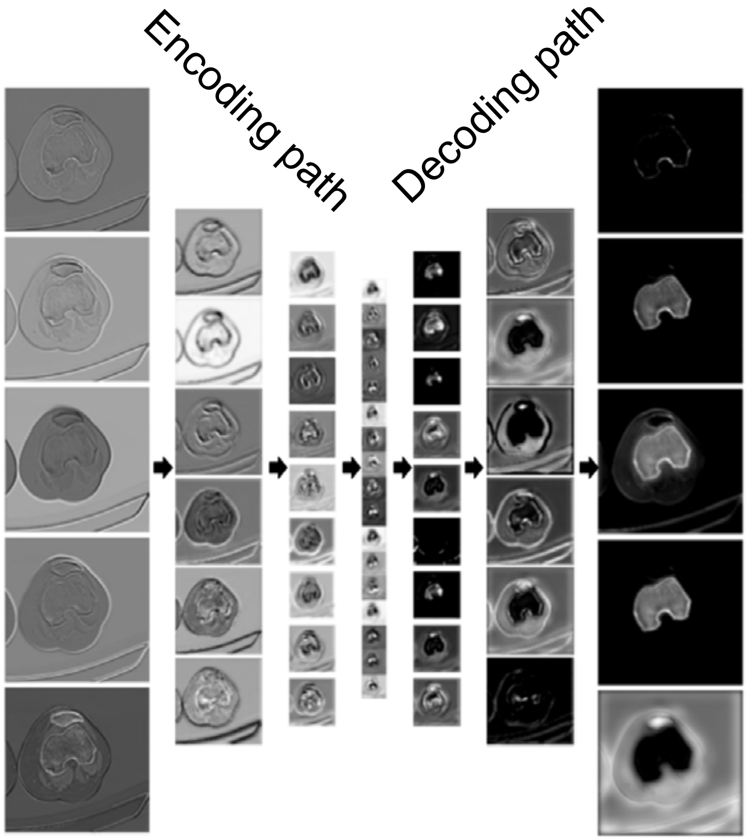
This network has 351435 parameters (filter size 3×3×3)
Assuming a volume of 128×128×128 the network output will be 128×128×128×3

8-16-32-64-32-16-8



Relation with explainable AI

Explainable AI (XAI) investigates methods for analyzing or complementing AI models to make the internal logic and output of algorithms transparent and interpretable, making these processes humanly understandable and meaningful.



The encoding path is learning filters to extract geometrical features performing edge detection (easy part)

but... what about the decoding path??

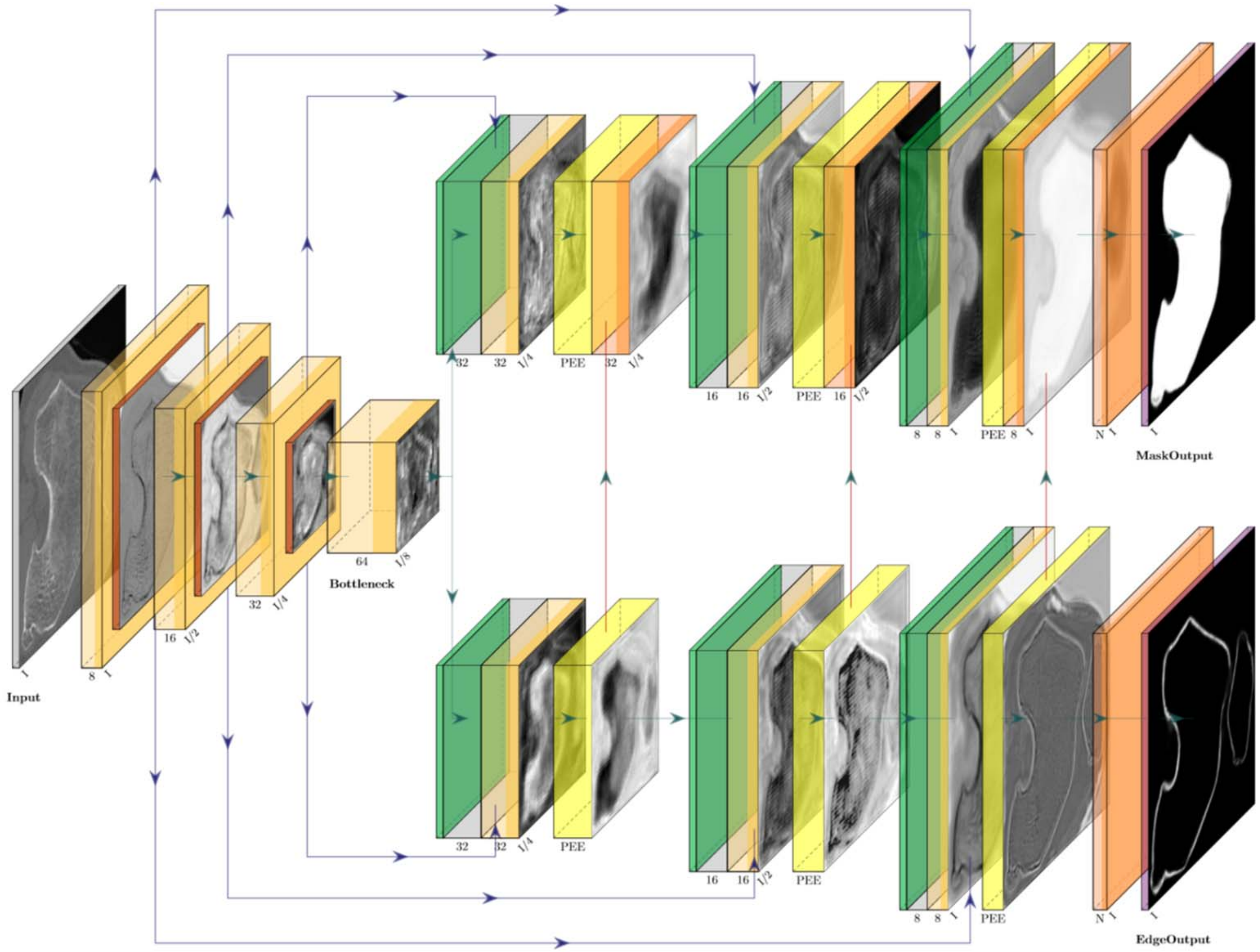
For sure it is learning semantics related to the task (femur and tibia segmentation)

Let us take into account that the 2D bone texture is very similar across all the 4 bones

So it is likely the decoding part is reconstructing the target shapes (2D/3D)

but.. what is the layer that has learnt to “remove” patella and fibula from segmentation?

Extending the U-NET with two interacting decoding paths

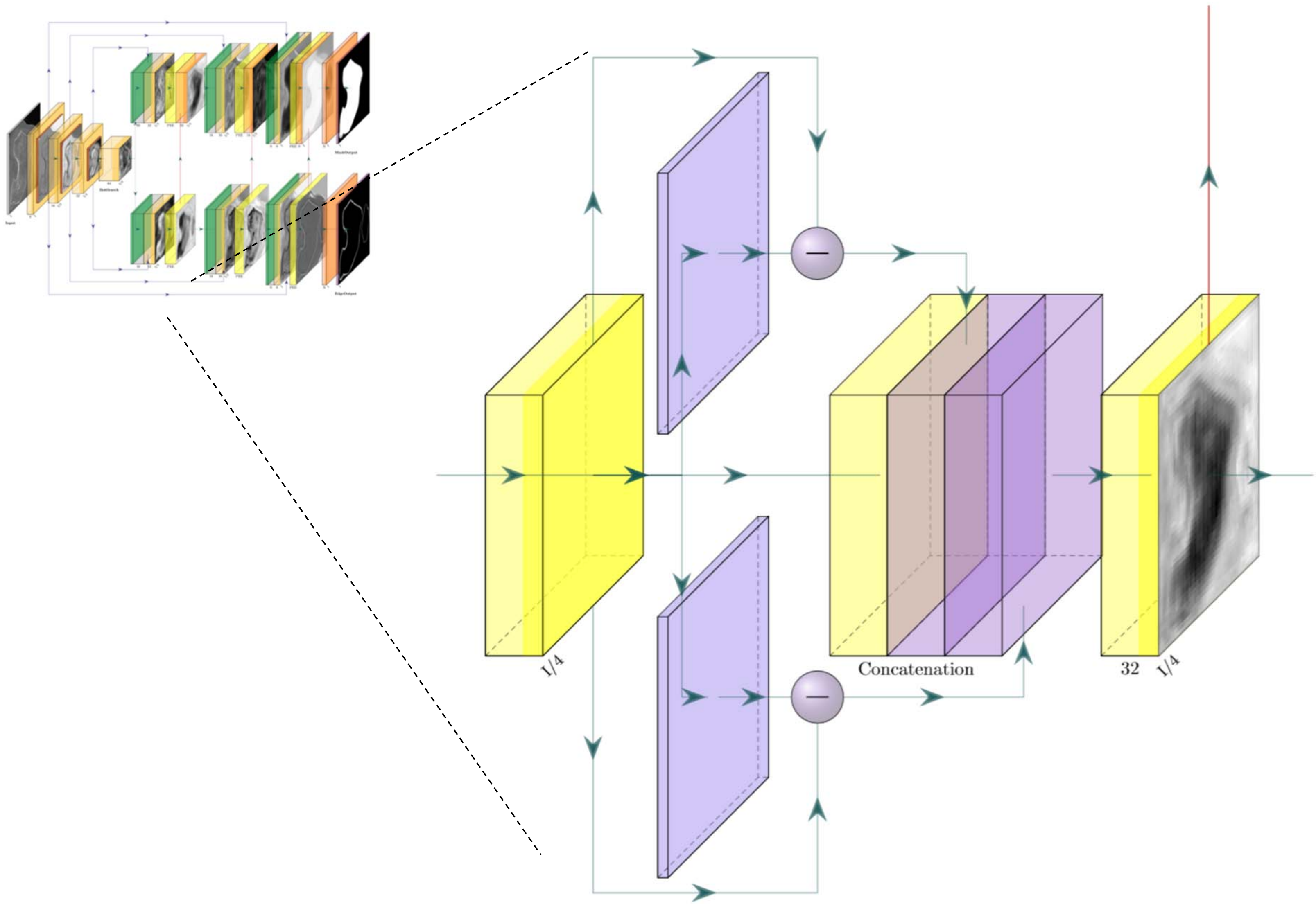


Upper decoding branch: masks
Lower decoding branch: contours

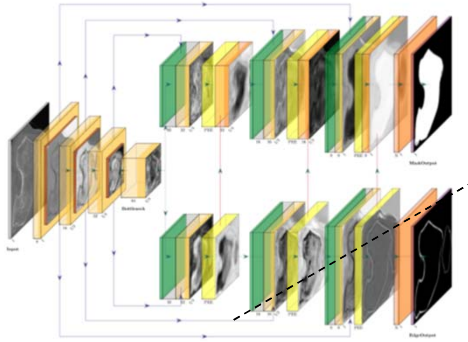
Skip connections (blue arrows)
Contour to mask concatenation (red arrows)

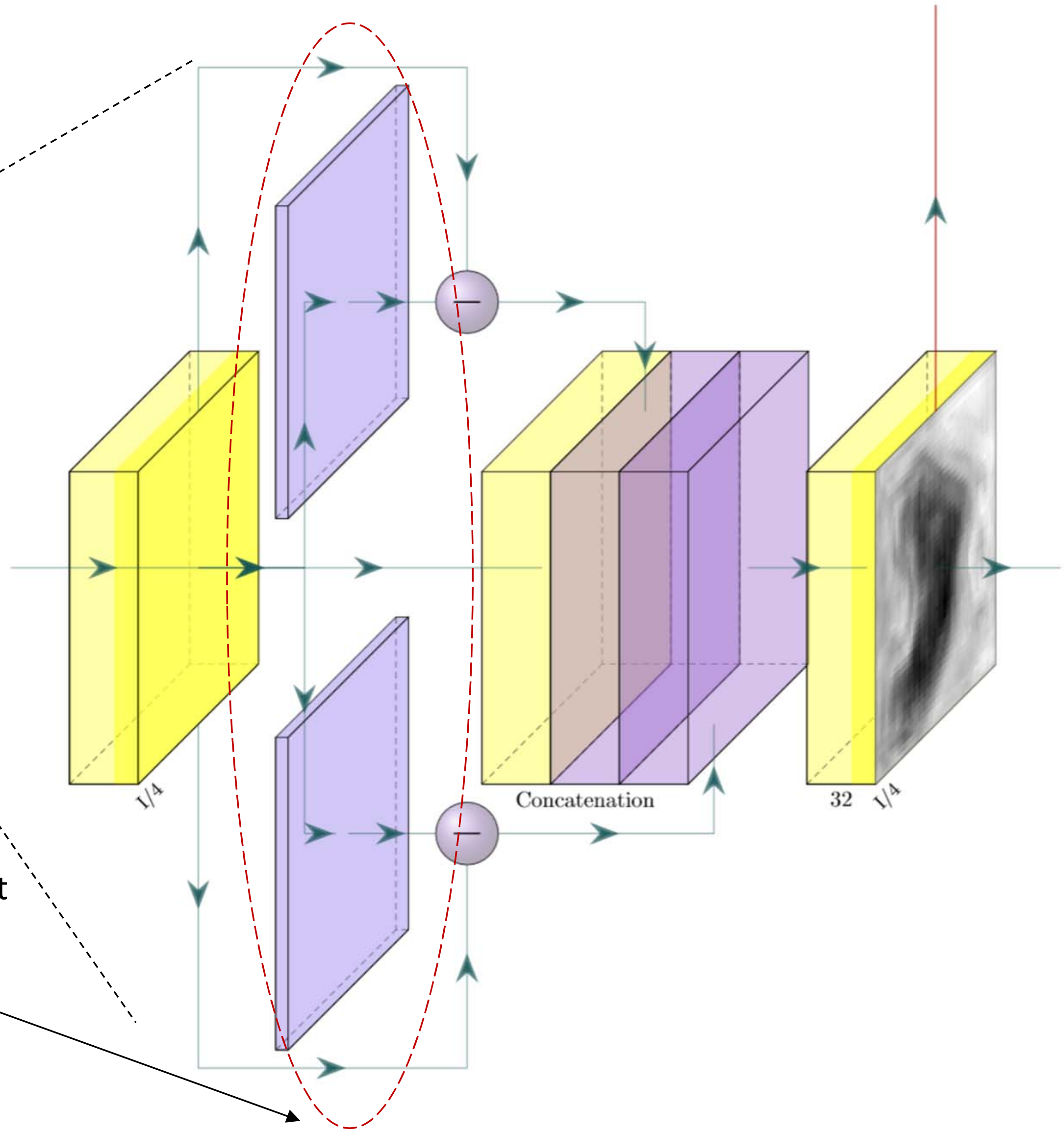
Pyramidal edge extraction (PEE) is implemented in each level of the edge detection branch.

Focus od pyramidal edge extraction module



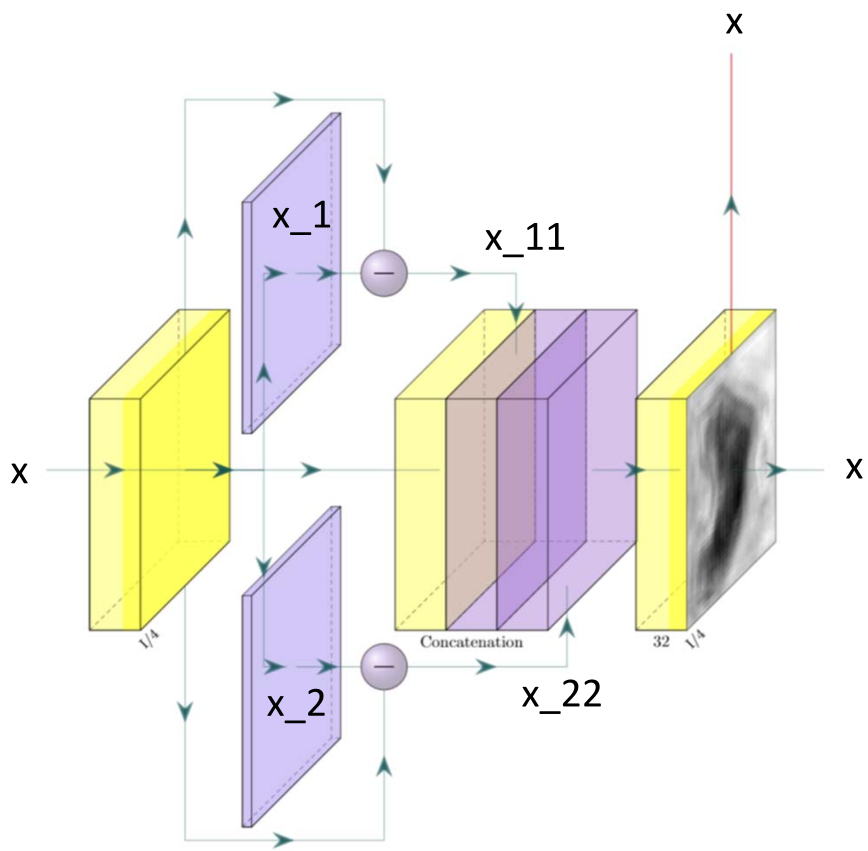
Focus od pyramidal edge extraction (PEE) module





- ❑ *“filters”* at different resolution (actually not convolution but average pooling)
- ❑ here we are using just two of them
  - (e.g 3x3x3 and 5x5x5)

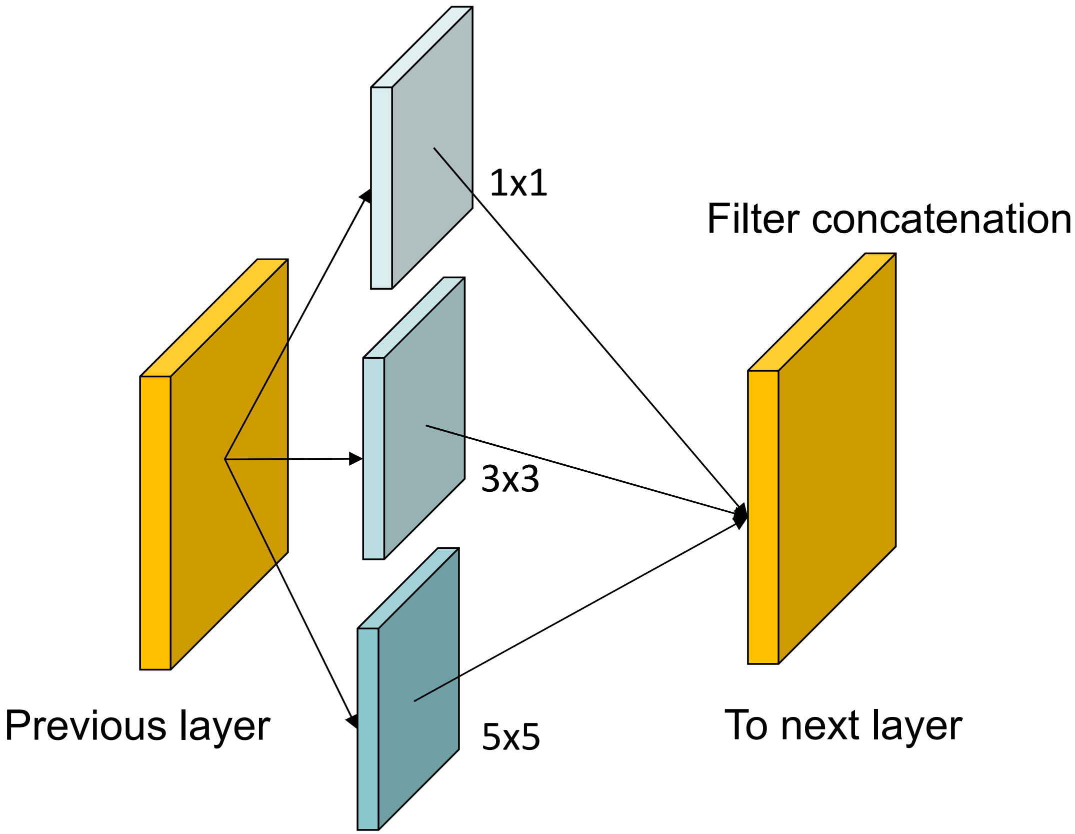
Focus od pyramidal edge extraction module



filters = 3x3x3
strides = 1

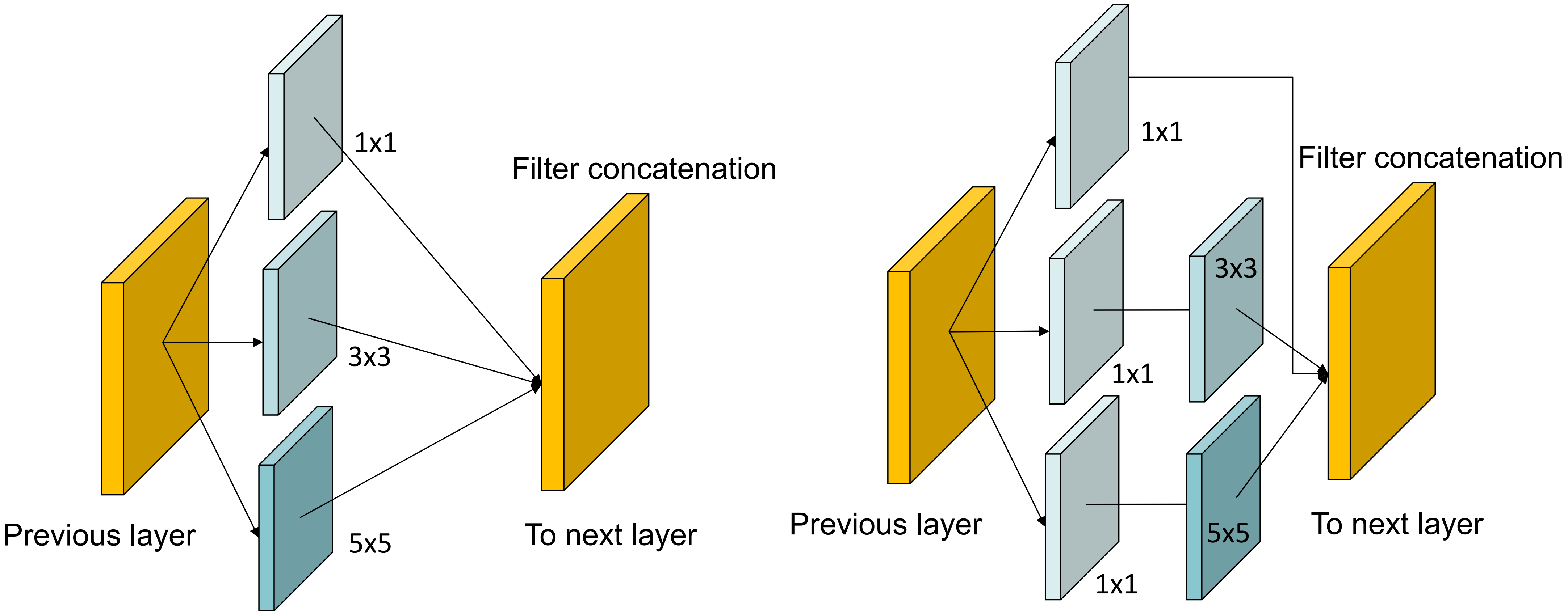
```
x = conv_layer(filters / 2, strides, padding='same')(x)

x_1 = average_layer(pool_size=pool_size_1, strides=strides,
padding='same')(x)

x_2 = average_layer(pool_size=pool_size_2, strides=strides,
padding='same')(x)


x_11 = subtract()([x, x_1])

x_22 = subtract()([x, x_2])


x = Concatenate()([x, x_11, x_22])

x = conv_layer(filters, strides, padding='same')(x)
```

Inception module: multi-resolution convolutional filters



Multiple kernel filter sizes on the same level

Inception module: multi-resolution convolutional filters and dimensionality reduction



DIFFERENCE?????