Machine Learning (CE 40717)

Fall 2024

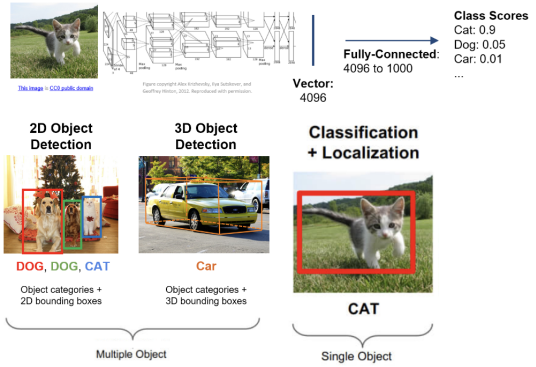
Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 30, 2024

Problem Setup

Before...

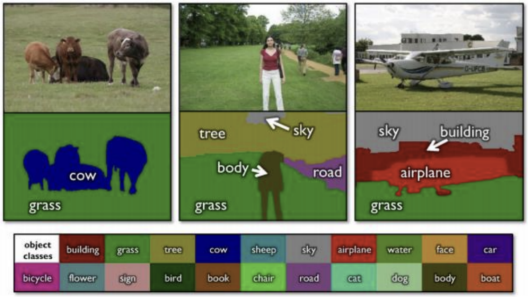


Now...

Problem: We want the output to have the same resolution as the input!

- Label every single pixel with its class. Actually simpler in some sense:
  - No longer variable # of outputs.
  - Every pixel has a label.

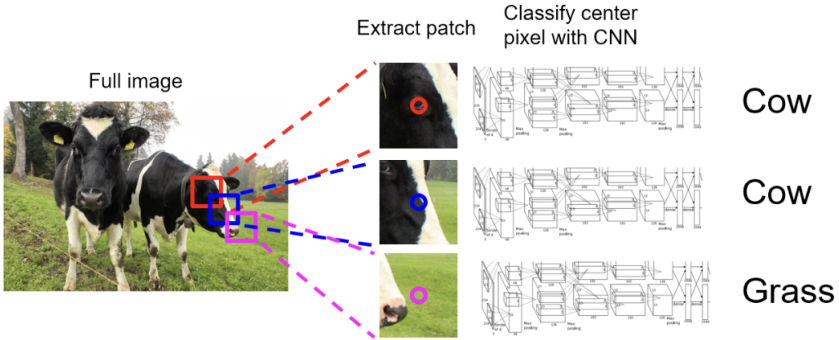
Problem Setup



Classify every point with a class
Don't worry for now about instances (e.g., two adjacent cows are just one "cow blob" and that's OK for some reason)

The challenge: design a network architecture
that makes this **"per-pixel classification"** problem computationally tractable

Semantic Segmentation: Sliding Window

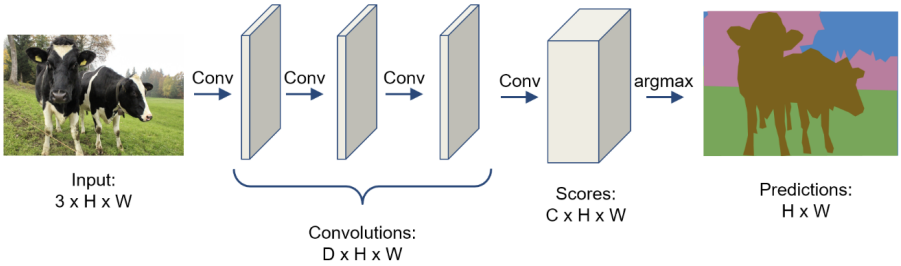


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al., "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling," ICML 2014

Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!

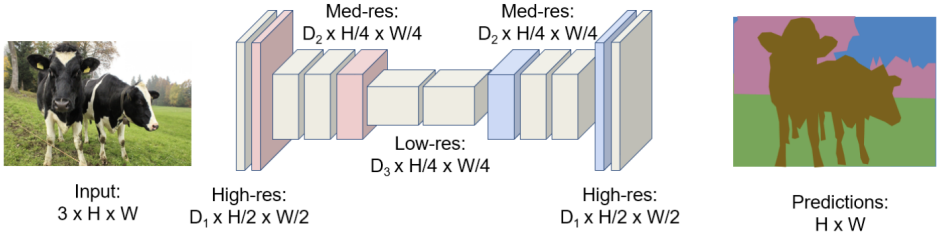


Problem: convolutions at original image resolution will be very expensive ...

Semantic Segmentation Idea: Fully Convolutional

So far we learn about downsampling...

Design network as a bunch of convolutional layers,
with downsampling and upsampling inside the network!



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation," CVPR 2015
Noh et al., "Learning Deconvolution Network for Semantic Segmentation," ICCV 2015

Recap: Types of Downsampling

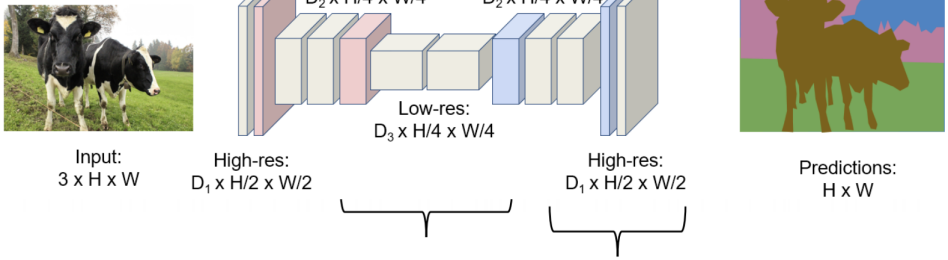
- Max Pooling
- Average Pooling
- Global Average Pooling
- Strided Convolution
- Spatial Pyramid Pooling
- ...

Details of Downsampling Methods

Pooling Method	Key Features	Best Applications	Rule of Thumb
Max Pooling	Captures the most important feature in a region (e.g., edges)	Image classification, object detection	Use when sharp features (e.g., edges) are essential.
Average Pooling	Smooths the feature map, averages over a region	Smoothing features, object detection	Use for smoother output, where fine detail is less important.
Global Average Pooling (GAP)	Replaces fully connected layers, outputs a single value per channel	Final layer in classification networks (e.g., ResNet, Inception)	Ideal for reducing model complexity and overfitting.
Strided Convolution	Combines downsampling and feature extraction	Object detection, fully convolutional networks (FCNs)	Use for efficiency when feature extraction and downsampling can be combined.
Spatial Pyramid Pooling (SPP)	Captures multi-scale features with pooling at multiple scales	Multi-scale object detection, semantic segmentation	Use when recognizing objects at various scales is critical.

Fully Convolutional Networks (FCN)

Design network as a bunch of convolutional layers,
with downsampling and upsampling inside the network!



Up Samplings

We can divide methods into:

- Static up sampling's
- Learnable up sampling's

Static Upsampling: “Unpooling”

"Nearest Neighbor"

1	2
3	4

→

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input: 2 x 2 Output: 4 x 4

“Bed of Nails”

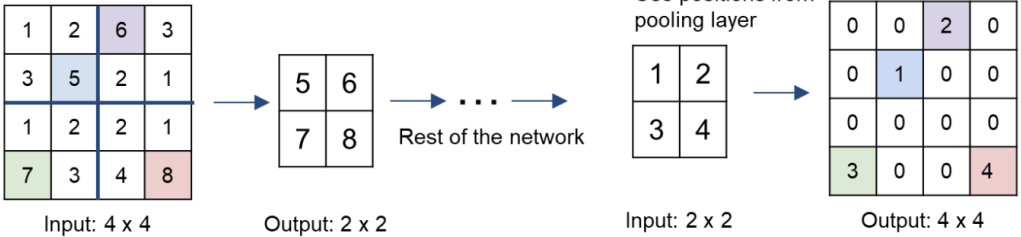
1	2
3	4

→

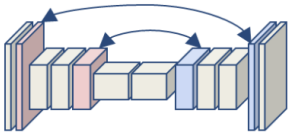
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input: 2 x 2 Output: 4 x 4

Static Upsampling: “Max Unpooling”



Corresponding pairs of downsampling and upsampling layers



Static Upsamplings

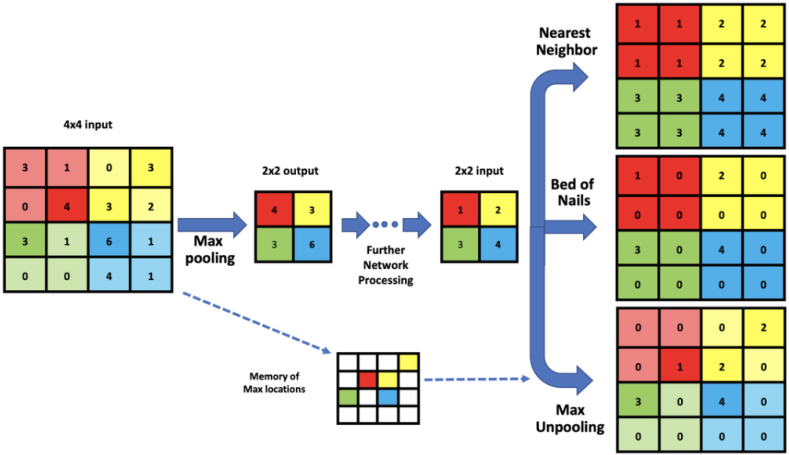
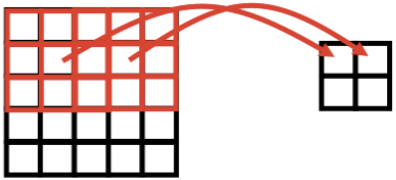


Image from this link

Convolution and Transposed Convolution

Normal Convolutions: reduce resolution with **stride**

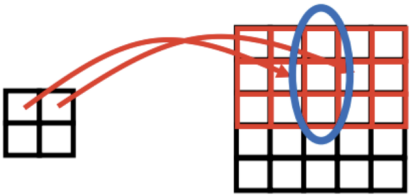
Stride = 2



Input: $H_f \times W_f \times C_{in}$
Output: $1 \times 1 \times C_{out}$
Filter: $H_f \times W_f \times C_{in} \times C_{out}$

Transposed Convolutions: increase resolution with **fractional "stride"**

Stride = 1/2



We have two sets of values here, **sum up them!**

Input: $1 \times 1 \times C_{in}$
Output: $H_f \times W_f \times C_{out}$
Filter: $C_{in} \times H_f \times W_f \times C_{out}$

1D Convolution: A Matrix Perspective

- Display the convolution matrix with a 1D kernel $[1, 1, 1]$ sliding over an input vector.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 1 & 1 & \dots & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

- This representation mimics a sliding window operation where the kernel moves across the input, performing element-wise multiplications and summing up the results.

1D Transposed Convolution: A Matrix Perspective

- The operation is equivalent to taking the transpose of the kernel matrix and applying it to the input:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 0 & \cdots & 0 \\ \cdots & 1 & 1 & 1 & 0 & 0 \\ \cdots & \cdots & 1 & 1 & 1 & 0 \\ \cdots & \cdots & \cdots & 1 & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} A_1 & A_2 & \cdots & A_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \sum_{i=1}^{k} x_i A_i$$

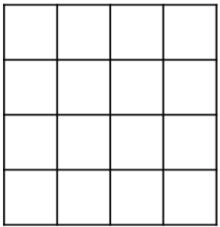
- Since the transposed convolution uses the **transpose** of the kernel matrix and effectively reverses the compression process (convolution), it's called **"Transposed Convolution"**.

Other names:

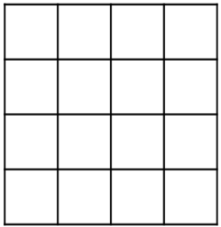
- Deconvolution (bad name)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 1 pad 1



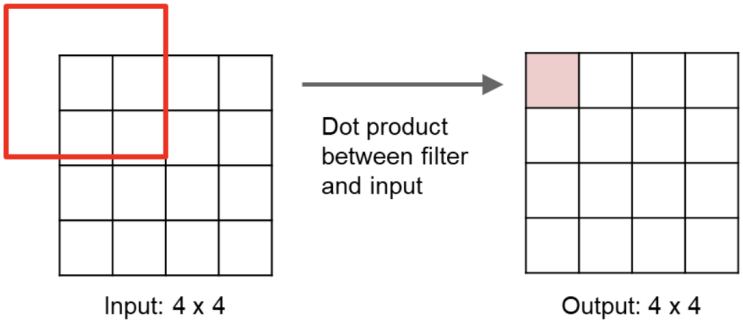
Input: 4 x 4



Output: 4 x 4

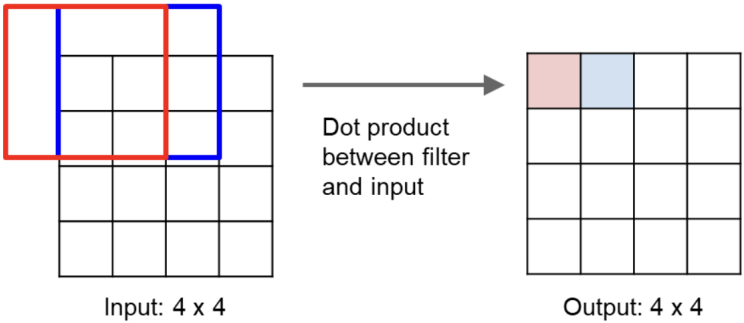
Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 1 pad 1



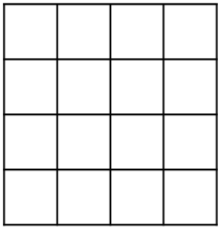
Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 1 pad 1

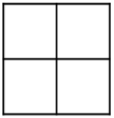


Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 2 pad 1



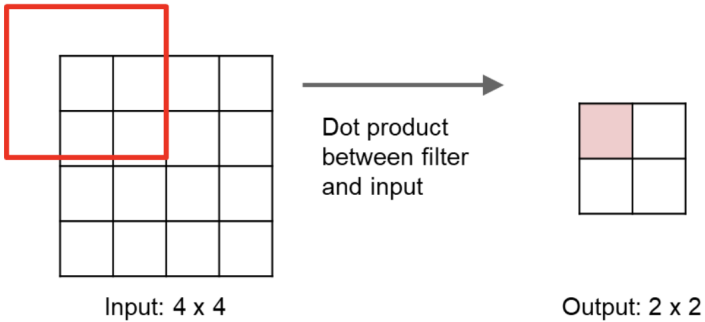
Input: 4 x 4



Output: 2 x 2

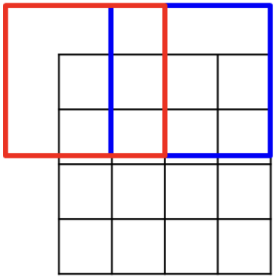
Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Learnable Upsampling: “Transpose Convolution”

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

Dot product between filter and input

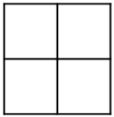
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

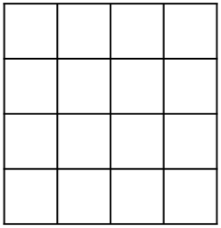
Stride gives ratio between movement in input and output

Learnable Upsampling: “Transpose Convolution”

3 x 3 **transpose** convolution, stride 2 pad 1

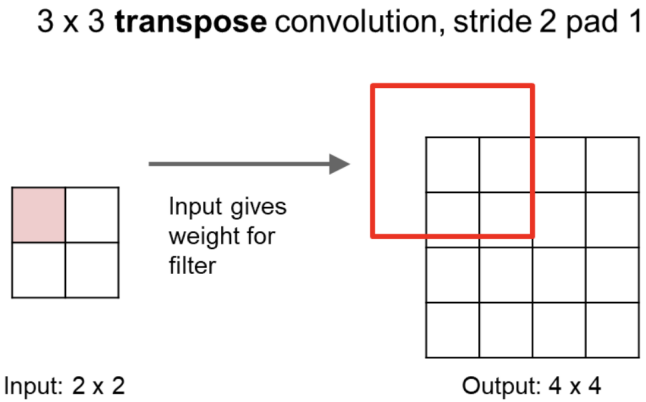


Input: 2 x 2

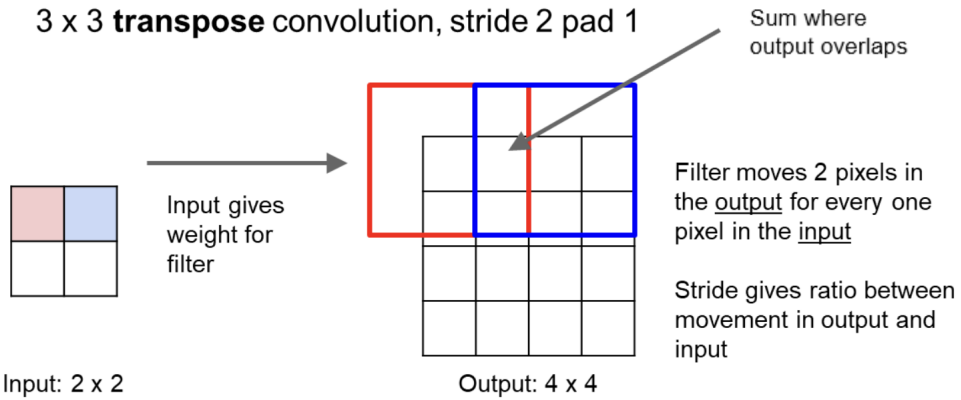


Output: 4 x 4

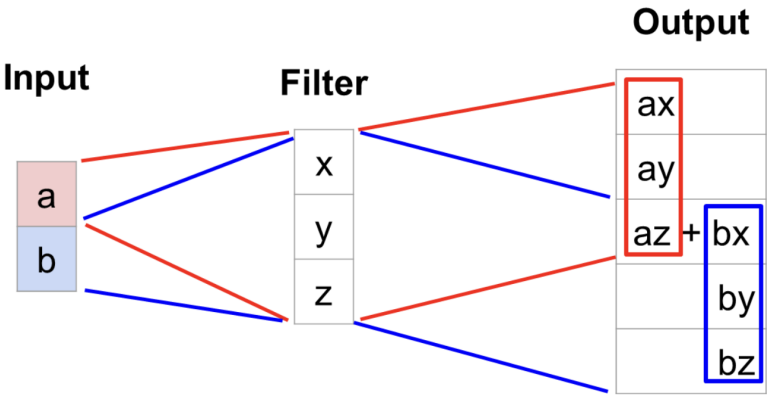
Learnable Upsampling: “Transpose Convolution”



Learnable Upsampling: “Transpose Convolution”



Transpose Convolution: 1D Example



- **Output** contains copies of the filter weighted by the **input**, summing at overlaps in the output.
- **Need to crop** one pixel from the output to make it exactly **2x input**.

Transpose Convolution: 2D Example

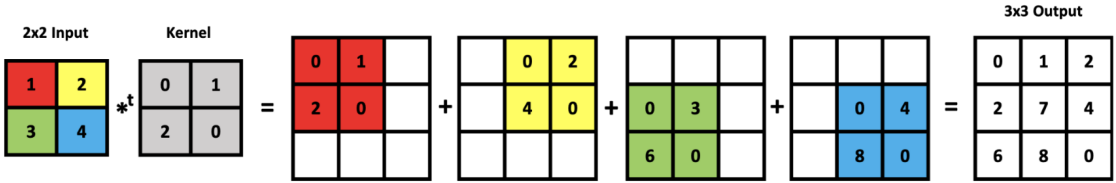
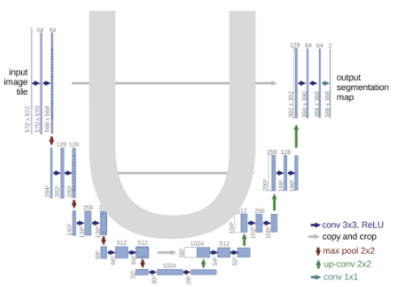
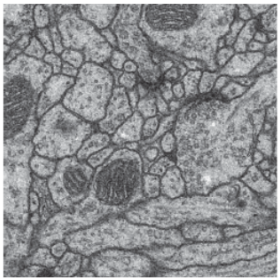


Image from this link

U-Net: Purpose & Origin

- Developed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
- First released in 2015, presented at the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI).
- Designed specifically for biomedical image segmentation tasks.





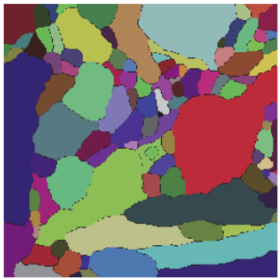
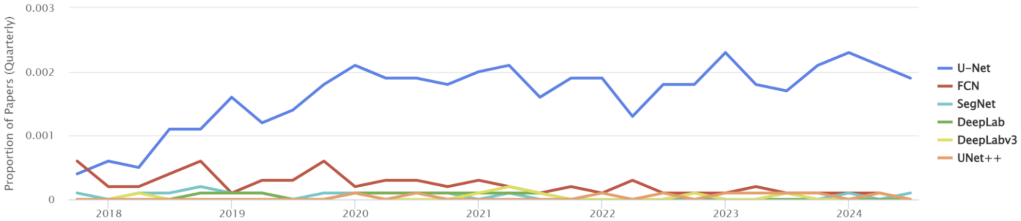


Image from this link

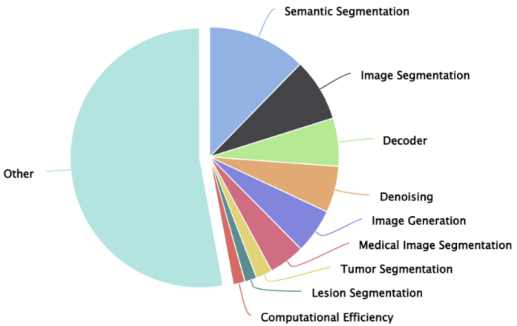
U-Net: Usage Over Time

- The rise in its usage is due to its simplicity, effectiveness, and adaptability to various image processing tasks beyond medical imaging.



From papers with code

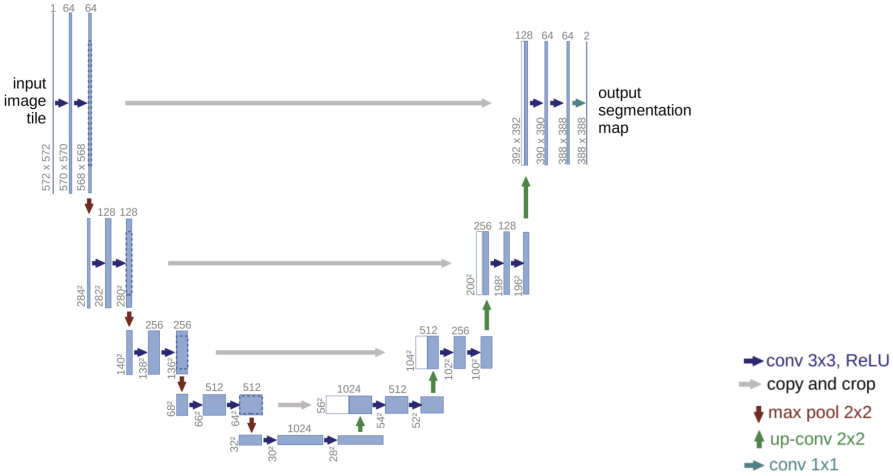
U-Net: Tasks



Task	Papers	Share
Semantic Segmentation	115	12.34%
Image Segmentation	73	7.83%
Decoder	56	6.01%
Denoising	54	5.79%
Image Generation	52	5.58%
Medical Image Segmentation	42	4.51%
Tumor Segmentation	19	2.04%
Lesion Segmentation	14	1.50%
Computational Efficiency	14	1.50%

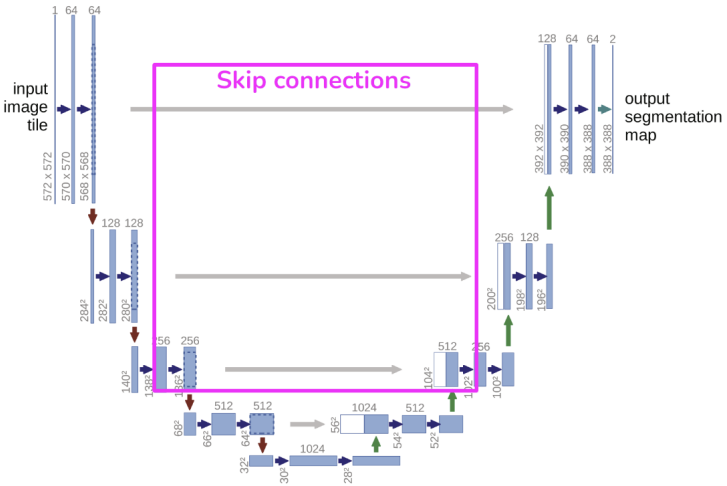
From papers with code

U-Net: Big Picture



From Unet

U-Net: Skip Connections



From Unet

U-Net vs AE's: The Skip Connection's

Preserving Spatial Features & Reducing Information Bottlenecks:

- Without skip connections, deep layers compress data heavily, leading to loss of spatial details, blurry outputs, and poor segmentation quality, especially in fine edges and boundaries.
- Skip connections bypass the bottleneck by passing high-resolution spatial details directly from encoder to decoder, helping retain sharpness and segmentation accuracy.

Improving Network Stability and Learning:

- Skip connections allow gradients to flow more effectively during backpropagation, mitigating the vanishing gradient problem common in deep networks.

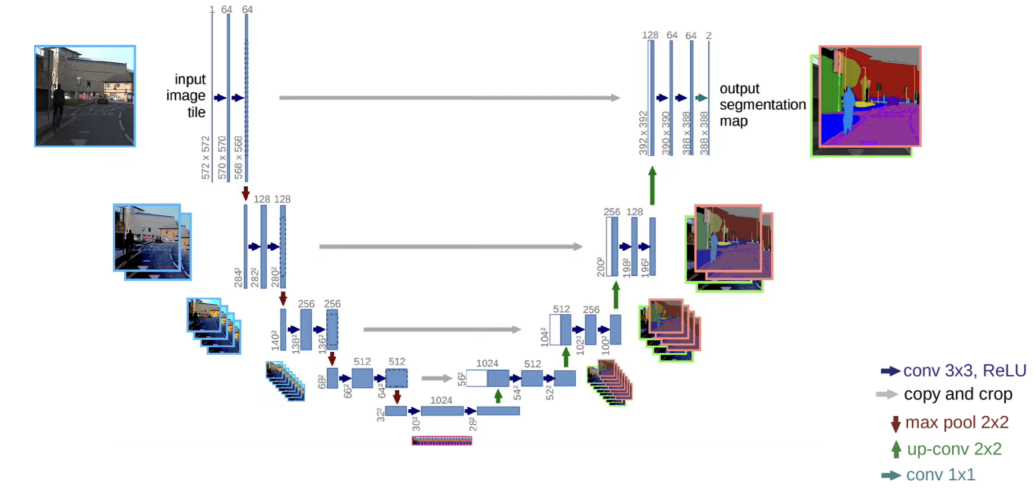
U-Net vs AE's: Latent Space

Unlike AE's (specially VAE), U-Net's latent space is less expressive:

- **Skip Connections Reduce the Bottleneck's Importance:** Because of skip connections, U-Net doesn't rely solely on the latent space for feature representation. Thus, the latent space is not required to encode as much high-frequency information as in an AE. As a result, its representation isn't as structured or "clustered," meaning a t-SNE plot of U-Net's latent space would not show the same level of clustering we see with VAEs, where every detail must be inferred from the bottleneck.

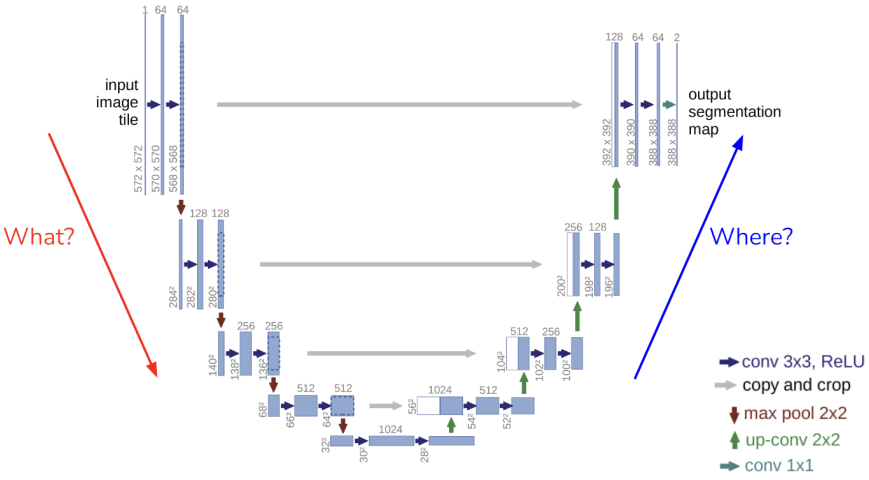
- **No Strong Encoding Constraint:** Since skip connections provide spatial detail, U-Net's latent space is not forced to develop an "expressive" representation, which is essential for models like VAEs where information passes strictly through a bottleneck.

U-Net: Big Picture



From this link

U-Net: Big Picture

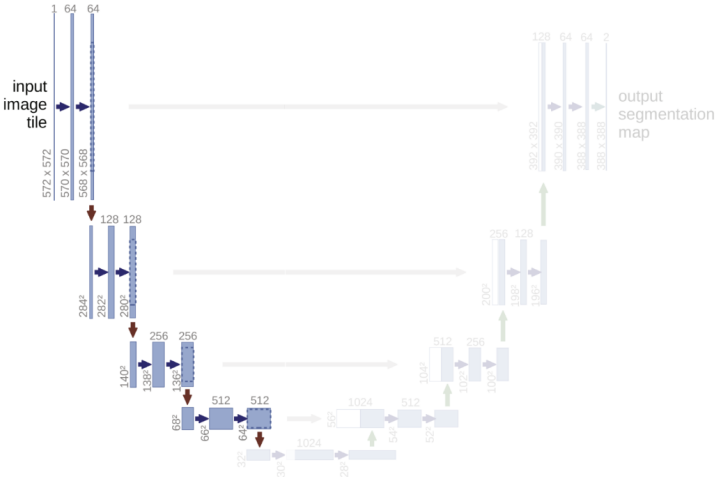


From Unet

U-Net: Big Picture

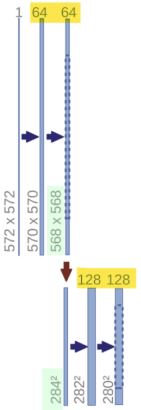
From this link

U-Net: Encoder



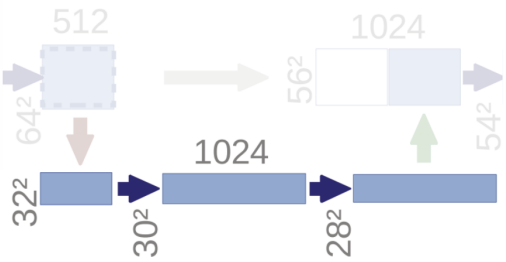
U-Net: Encoder

- Repeated 3x3 convolutional with **valid padding** (= no padding) + ReLU layers
- 2x2 max pooling layers to down sample
- **Double channels** with conv after the **max pool**
- Due to the **shrinking effect** of valid convs, **cropped part** is concatenated with upsampled feature map



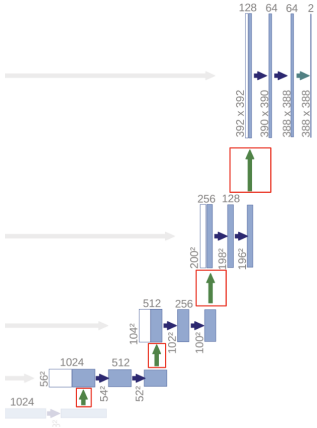
U-Net: Bottleneck

- Down sample with **2x2 max pooling**
- Repeated **3x3 convolutional** with valid pooling (= no padding) + ReLU layers
- **Double channels** with conv after the max pool



U-Net: Decoder

- Repeated 3x3 convolutional with **valid pooling** (= no padding) + ReLU layers
- Upsampling, followed by 2x2 convolutional layer
- Halve channels after upsampling convolution



U-Net: Skip Connections

Skip Connections:

- Connect encoder layers to decoder layers at the same depth.
- Preserve high-resolution features for better upsampling.

Cropping for Alignment:

- Due to valid padding, feature maps in the encoder are larger than those in the decoder.
- Cropping aligns sizes before concatenation in skip connections.

Why Important?:

- Combines local details (from encoder) with context (from decoder).
- Helps in precise pixel-wise segmentation.
- Overcomes vanishing gradient issue by allowing smooth information flow.

Data Efficiency in U-Net

- **Residual and Skip Connections:** These connections reduce the amount of information the model needs to learn, as many spatial details are preserved, requiring less adaptation from the model. The need to learn detailed spatial arrangements is mitigated by direct concatenation, which efficiently reintroduces spatial information.
- **Fully Convolutional Nature:** U-Net uses only convolutional layers, which have fewer parameters compared to fully connected networks. This allows the model to generalize well from smaller datasets.
- **Data Augmentation Advantages:** Because U-Net is often used for pixel-wise tasks like semantic segmentation, every augmentation (rotation, scaling, etc.) maintains a one-to-one correspondence with ground truth masks. This allows each augmentation to act as a "new" data point, further enriching the dataset without additional manual labeling.
- **Patch-Based Training:** Instead of processing the entire image (e.g., a 1024x1024 image), U-Net can work with smaller patches, increasing batch sizes and enabling more efficient use of each image during training.

Slides by: Amirhossein Izadi

- F.-F. Li, J. Wu, and R. Gao, “CS231n” Lecture slides, 2024, Stanford
- A. Amini, “6s191” Lecture slides, 2024, MIT.
- M. Soleymani, “Deep Learning” Lecture slides, 2024, Sharif University of Technology.
- H. Beigy, “Deep Learning” Lecture slides, 2022, Sharif University of Technology.
- S. Levine, “CS W182/282A” Lecture slides, 2024, UC Berkeley
- Y. Maziane, “Diffusion models: Seek of information and structure in latent space,” 2022, University of Liège.
- G. Buzzard, “Mathematical Aspects of Neural Networks” Lecture slides, 2019, Purdue University.

Any Questions?