Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

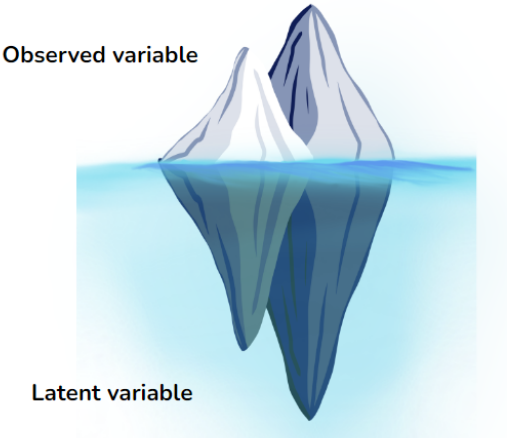
CE Department
Sharif University of Technology

November 18, 2024

What Is Latent Variable?

- **Definition:** Latent variables are variables that are not directly observed but are inferred from other variables that are observed (measured).
- **Origin:** Derived from the Latin word *latēre* meaning "hidden".
- **Examples:**
  - Personality traits (in psychology)
  - Economic factors like inflation (in economics)
  - Topic of a document (in natural language processing)

What Is Latent Variable?



What Does This Have To Do With Our Lesson?

How can we combine neural networks and unlabeled data to perform compression?

Autoencoder: Background

Using latent variables as a foundation, an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data.

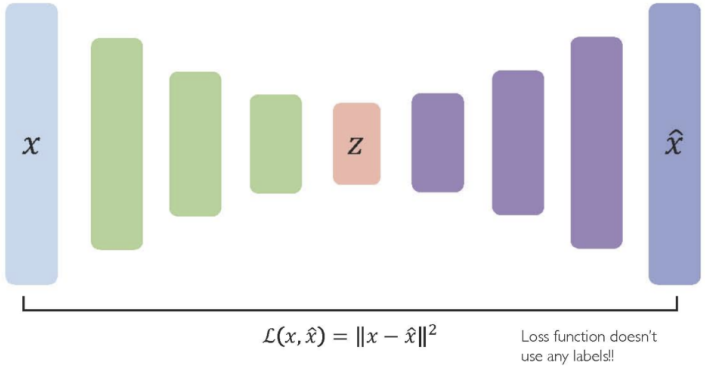
Why do we care about a low-dimensional z?

"Encoder" learns mapping from the data, x, to a low-dimensional latent space, z

Autoencoders: Background

How can we learn this latent space?

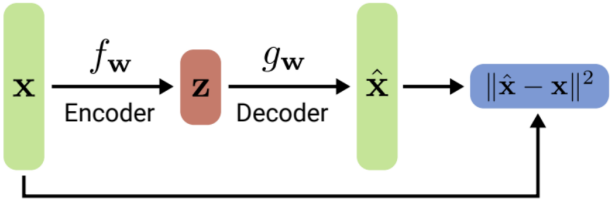
Train the model to use these features to **reconstruct the desired (usually original) data**.



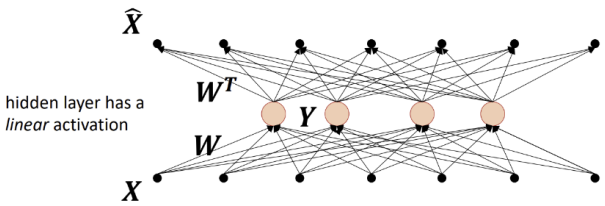
"Decoder" learns mapping back from latent z, to a reconstructed observation, x

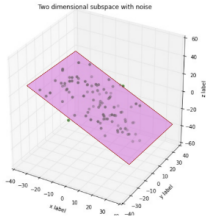
Autoencoders

1. These models map between **observation space** $\mathbf{x} \in \mathbb{R}^D$ and **latent space** $\mathbf{z} \in \mathbb{R}^Q$:
   - **Encoder** $f_w : \mathbf{x} \mapsto \mathbf{z}$ The "Analysis" net which computes the hidden representation
   - **Decoder** $g_w : \mathbf{z} \mapsto \mathbf{x}$ The "Synthesis" which recomposes the data from the hidden representation
2. Each latent variable gets associated with each data point in the training set.
3. The latent vectors are smaller than the observations ($Q < D$) $\Rightarrow$ compression.
4. Models are linear or non-linear, deterministic or stochastic, with/without encoder.



Autoencoder Without a Non-linearity



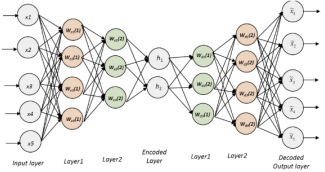


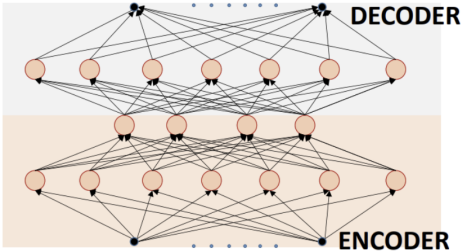
$\mathbf{Y} = \mathbf{WX}$ $\quad$ $\hat{\mathbf{X}} = W^T\mathbf{Y}$ $\quad$ $E = \|\mathbf{X} - W^T W\mathbf{X}\|^2$ $\quad$ Find W to minimize Avg$[E]$

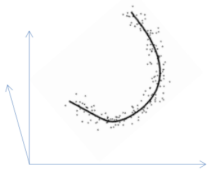
- **This is similar to PCA**
  - The output of the hidden layer will be in the principal subspace
    - Even if the recomposition weights are different from the "analysis" weights

“Deep” AE

- **Deep nonlinear autoencoders** learn to project the data, not onto a subspace, but onto a nonlinear manifold.
- This manifold is the **image of the decoder**.
- This is a kind of nonlinear dimensionality reduction.

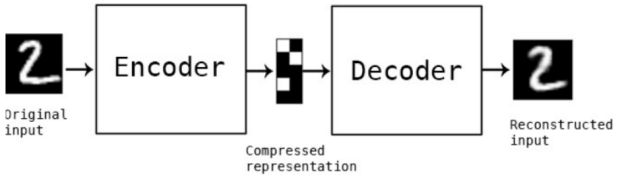






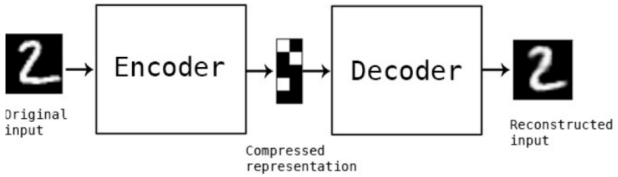
AE vs PCA

As we said, **PCA** can be described as



$$\min_W \sum \|\hat{x} - x\|^2$$
$$W^T W = I$$
$$\min_W \sum \|W^T W x - x\|^2$$

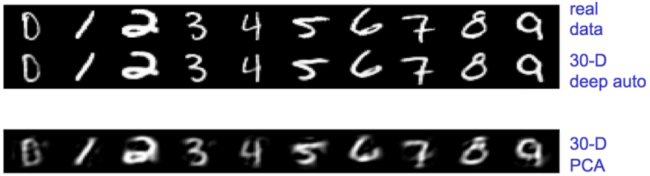
And also **Autoencoders** can be thought of as a **non-linear PCA**.

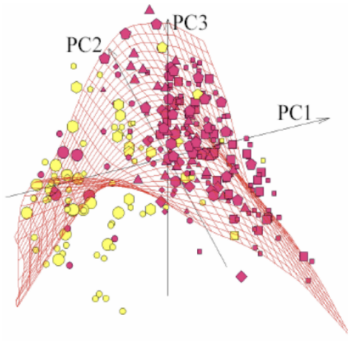


$$\min_{h,g} \sum \|\hat{x} - x\|^2$$
$$\min_{h,g} \sum \|g(f(x)) - x\|^2$$

AE vs PCA

Nonlinear autoencoders can learn more powerful codes for a given dimensionality, compared with linear autoencoders (PCA).





Autoencoder Key Characteristics

- **Specialized for Specific Data Type**
  - Autoencoders excel at recognizing patterns in the type of data they were trained on, but may fail on different data types.
  - Example: A model trained on animal images will not perform well when applied to landscapes.

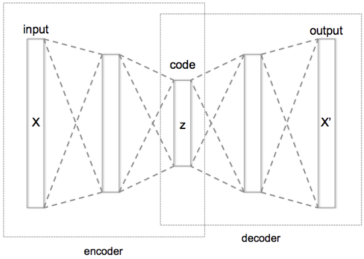
- **Tailored Compression vs. General Algorithms**
  - Unlike generic compression techniques (MP3, JPEG), which apply universal rules, autoencoders customize their approach to fit the training data structure.

- **Loss of Detail in Output**
  - Autoencoders don't retain all the original information, leading to some degradation in the reconstructed data.
  - This makes them "lossy" – similar to formats like MP3 or JPEG, which prioritize size over perfect fidelity.

Autoencoder Training

Encoder f and decoder g.

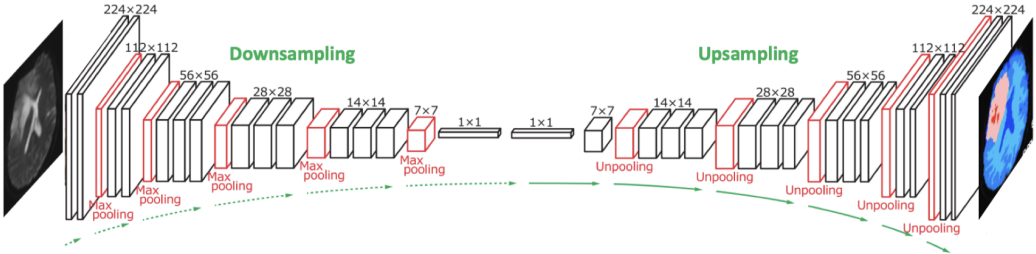
$$f : X \mapsto z$$
$$g : z \mapsto X$$
$$\arg\min_{f,g} \|x - (f \circ g)(x)\|^2$$



An autoencoder is a feed-forward non-recurrent neural network.

- With an input layer, an output layer, and one or more hidden layers
- It can be trained using the following technique:
  - Compute gradients using back-propagation, followed by mini-batch gradient descent

Fully Convolutional AE

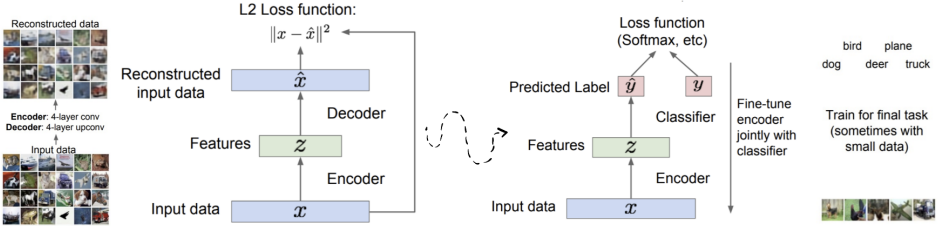


We will discuss more about **"Downsampling"** and **"Upsampling"** in the next section

Image from this link

Autoencoders As Pretrained Models

- **Feature Extraction:** Encoder **learns** data patterns and **compresses** them into useful features.
- **Supervised Model Initialization:** Use the **pretrained encoder** to initialize a supervised learning model.
- **Fine-Tuning:** Train the encoder and classifier **together** to improve performance.
- **Useful with Small Data:** Effective when training on **small datasets** by leveraging pretrained features.

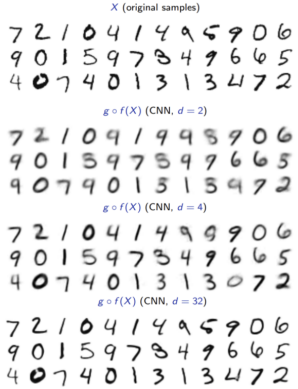


AE Performance & Bottleneck Size (MNIST Example)

- **Reconstruction Error and Bottleneck Size:**
  - A good autoencoder should minimize reconstruction error.
  - The error depends heavily on the size of the z (the dimension of the latent space).
- **Experiment Setup:**
  - CNN-based AE trained on MNIST with different bottleneck sizes:
    - $(2, 1, 1)$, $(4, 1, 1)$ & $(32, 1, 1)$

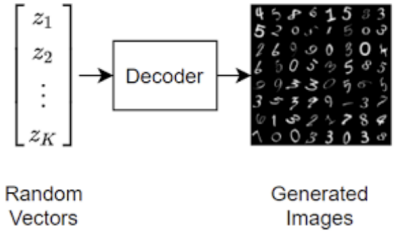
AE Performance & Bottleneck Size (MNIST Example) - Results

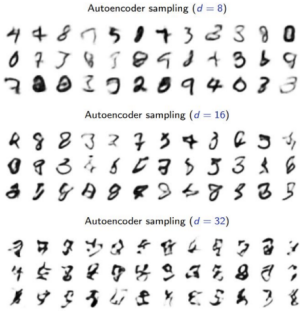
- **Results:**
  - With $d = 2$: Reconstruction is still good; a classifier could identify digits decently.
  - With $d = 32$: Reconstruction is nearly perfect; despite a compression ratio of ~ 0.04 (from 32 to 28x28).
- **Larger latent spaces** generally lead to **better reconstructions**, as more variables allow more information to be preserved during compression.



Can The Decoder Of AE Be Used For Data Generation?

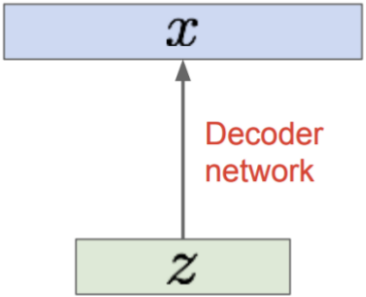
- Autoencoders are effective data compressors but up to now do not yet show **data generation capabilities**.
- We already trained an AE on MNIST; then we use **random vectors** as input to the Decoder to generate new data.
  - Is the **size** of the bottleneck affecting the **quality** of the generated sample?

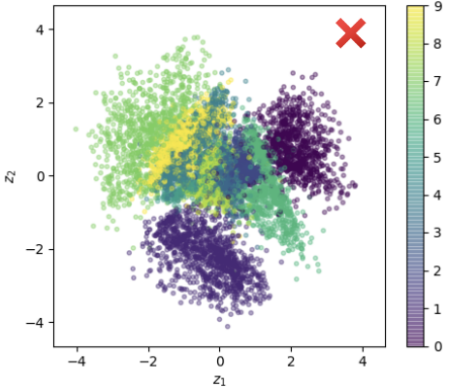




Can The Decoder Of AE Be Used For Data Generation?

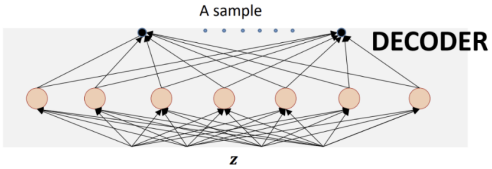
Is **any random vector** we feed into decoder networks results in a valid output?

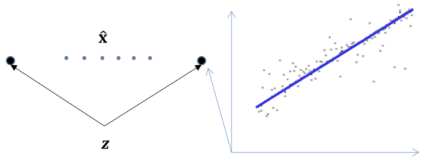




Can The Decoder Of AE Be Used For Data Generation?

- **Varying the Latent Representation (z):** By changing the values in the latent space (z), the decoder can generate outputs that move along a **learned non-linear manifold** (a curved space that represents the structure of the data).
- **Manifold Constraint:** The decoder can only generate data that lies on **the manifold learned from the training data**. This means it cannot generate data outside of the patterns it was trained on.
- **Data Generation:** The decoder is an effective generator of data that follows the **distribution of the training set**. Any value you feed into the latent space will result in an output that resembles the training data.



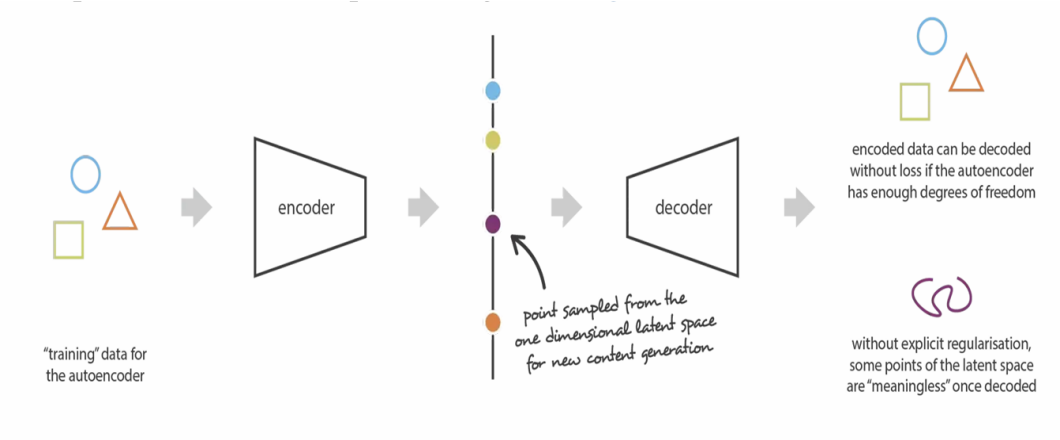


Latent Space Properties

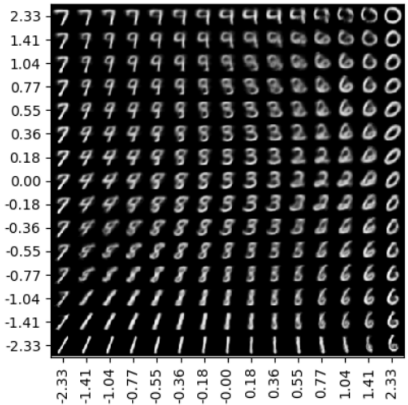
- For the generation purpose, we need the latent space to show the following properties:
  - **Continuity:** Similar points in the latent space will be similar after decoding.
  - **Completeness:** Samples of the latent space lead to meaningful content after decoding.

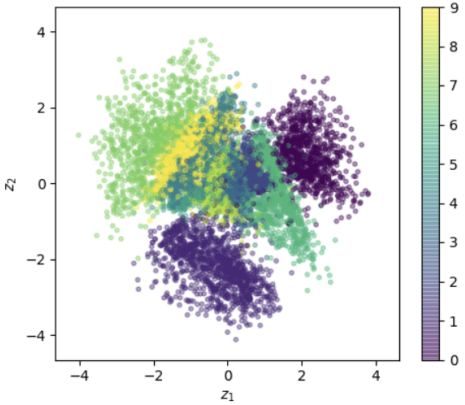
Latent Space Properties: Continuity

Latent Space Properties: Completeness



AE Latent Space - MNIST





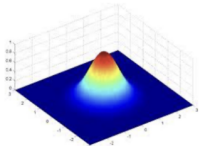
What is the problem?

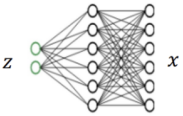
Latent Variable

- We define a simple distribution on the latent space, i.e., $p(z)$ and a decoder network $p_\theta(\mathbf{x}|\mathbf{z})$ to map the latent variable to the data space.

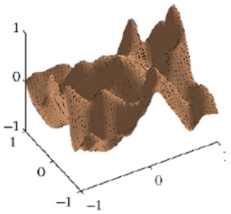
$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$$

$p(z)$





$p(x)$



AE's: Variational Autoencoder (Intuition)

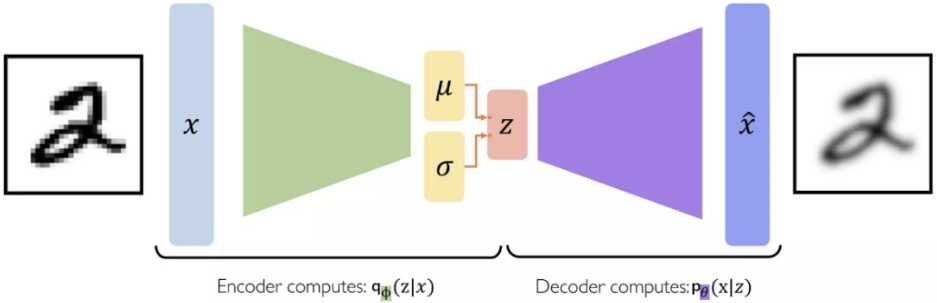
The key change that they added was that instead of having an encoder that maps points in X to points in Z, VAE's map points in X to distributions in Z. This allows a point in X to be indirectly mapped to several close neighbour points in Z.

Simple AE: input x $\xrightarrow{\text{Encoding}}$ latent representation $z = e(x)$ $\xrightarrow{\text{Decoding}}$ input reconstruction $d(z)$

Variational AE: input x $\xrightarrow{\text{Encoding}}$ latent distribution $p(z|x)$ $\xrightarrow{\text{Sampling}}$ sampled latent representation $z \sim p(z|x)$ $\xrightarrow{\text{Decoding}}$ input reconstruction $d(z)$

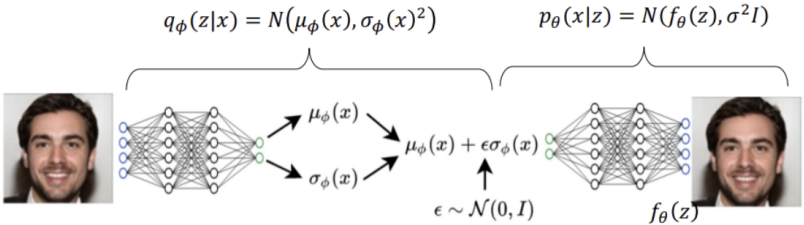
AE's: Variational Autoencoder (Big Picture)

The key change that they added was that instead of having an encoder that maps points in X to points in Z, VAE's map points in X to distributions in Z. This allows a point in X to be indirectly mapped to several close neighbour points in Z.



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(regularization term)}$$

AE's: Variational Autoencoder (Details)



VAE loss: $\mathscr{L}(x, \theta, \phi) = \underbrace{\|x^{(i)} - \hat{x}^{(i)}\|^2}_{\text{reconstruction loss}} + \underbrace{KL\left(q_\phi(z|x^{(i)}) \,\|\, \mathcal{N}(0, I)\right)}_{\text{regularization term}}$

$\downarrow$

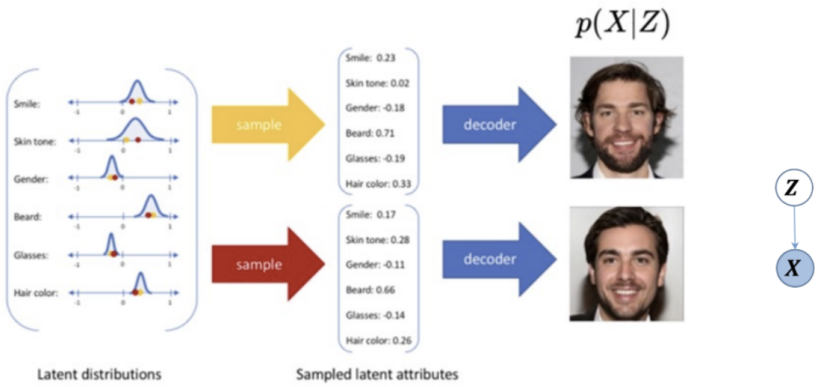
$f_\theta(z) = f_\theta\left(\mu_\phi(x^{(i)}) + \epsilon\sigma_\phi(x^{(i)})\right)$

$\downarrow$

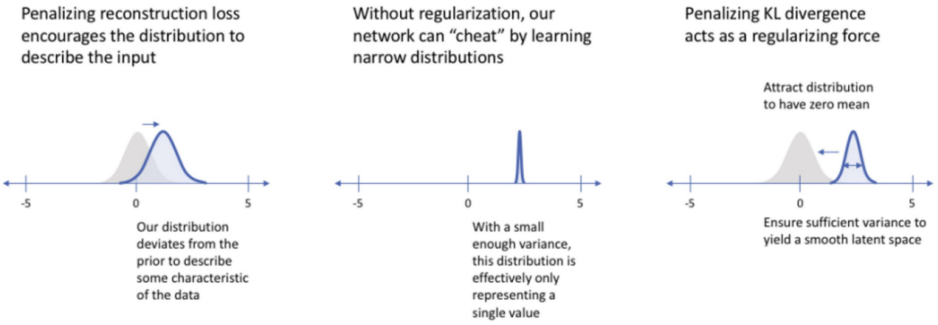
Common choice of prior on the latent distribution.

AE's: Variational Autoencoder (Details)

- Encourage encodings to be evenly distributed around the center of the latent space
- Penalize the network when it tries to memorize data



AE's: Variational Autoencoder (Details)



The VAE objectives arranges data on a compact manifold (we can sample from) in a continuous smooth way.

AE's: Undercomplete Autoencoders

1. An autoencoder whose code dimension is less than the input dimension is called undercomplete.
2. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.
3. The learning process is described simply as minimizing a loss function

$$L(x, g(f(x)))$$

where L is a loss function penalizing $g(f(x))$ for being dissimilar from x, such as the mean squared error.

AE's: Overcomplete Autoencoders

1. consider encoder f and decoder g.

$$X \in \mathbb{R}^d \quad h \in \mathbb{R}^k$$

2. When $k > d$, the autoencoder is called Overcomplete Autoencoders.

3. There are other ways we can constrain the reconstruction of an autoencoder than to impose a hidden layer of smaller dimension than the input.

4. Regularized Autoencoders use a loss function that encourages the model to have some properties besides reproducing inputs.
   - Sparsity representation (Sparse Autoencoders)
   - Smallness of derivative of representation (Contractive Autoencoders)
   - Robustness to noise or to missing inputs (Denoising Autoencoders)

AE's: Sparse Autoencoders (Intuition)

Idea: Can we describe the input with a small set of *"attributes"*?
This might be a more *compressed* and *structured* representation.

1. NOT structured - *"dense": most values non-zero*

```
Pixel (0,0):  #FE057D
Pixel (0,1):  #FD0263
Pixel (0,2):  #E1065F
```

Idea: "Sparse" representations are going to be more structured!

2. Very structured! - *"sparse": most values are zero*

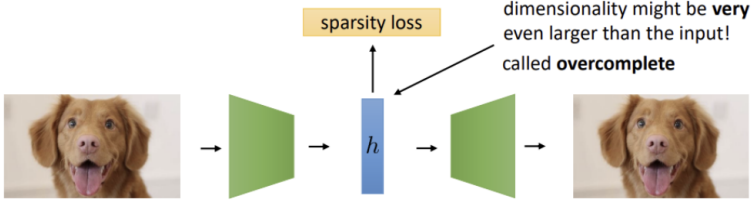
```
has_ears:  1
has_wings:  0
has_wheels:  0
```

There are many possible "attributes,"& most images don't have most of the attributes.

Aside:

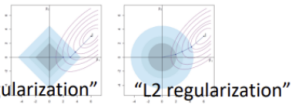
This idea originated in neuroscience, where researchers believe that the brain uses **sparse** representations (see "sparse coding").

AE's: Sparese Autoencoder (Big Picture)



ℓ_1 norm of h
gradient is always $\text{sgn}(h_j)$

simple sparsity loss: $\sum_{j=1}^{D} |h_j|$



There are other kinds of sparsity losses/models:

- Lifetime sparsity
- Spike and slab models

AE's: Sparse Autoencoders (Details)

1. Sparse Autoencoders try to minimize the following function.

$$L(x, g(f(x))) + \Omega(h)$$

2. The first term is loss for copying inputs.
3. The second term is sparsity penalty.
4. In general neural networks, we are trying to find the **maximum likelihood:** $p(x|\theta)$.
5. We often use $\log p(x|\theta)$ for simplification, from which we can get the loss function without regularization.
6. What about MAP (Maximum a posteriori)?

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

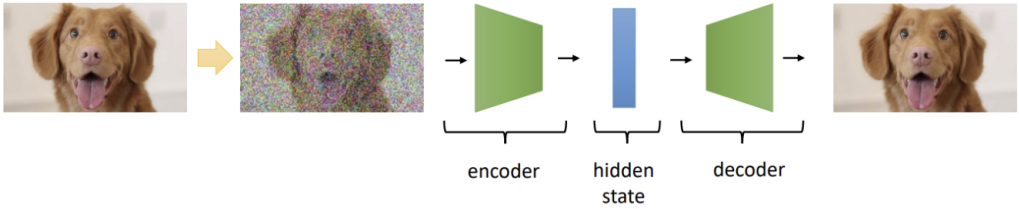
7. Maximizing the log of the above function yields:

$$\text{maximize } (\log p(x|\theta) + \log p(\theta))$$

8. The first term is the loss function, and the second term is the regularization penalty.

AE's: Denoising Autoencoders (Intuition)

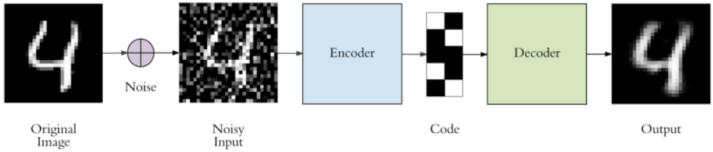
Idea: a good model that has learned meaningful structure should "fill in the blanks"



There are many variants on this basic idea, and this is one of the most widely used simple autoencoder designs.

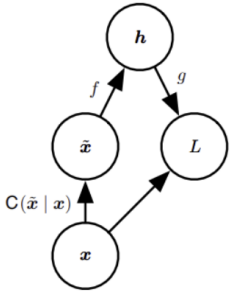
AE's: Denoising Autoencoders (Details)

1. **The denoising autoencoder (DAE)** is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
2. **Traditional autoencoders minimize** $L(x, g(f(x)))$
   - L is a loss function penalizing $g(f(x))$ for being dissimilar from x, such as L_2 norm of difference: mean squared error.
3. **A DAE minimizes** $L(x, g(f(\tilde{x})))$
   - $\tilde{x}$ is a copy of x that is corrupted by some form of noise.
   - The autoencoder must undo this corruption rather than simply copying their input.



AE's: Denoising Autoencoders (Training Procedure)

1. The DAE training procedure is

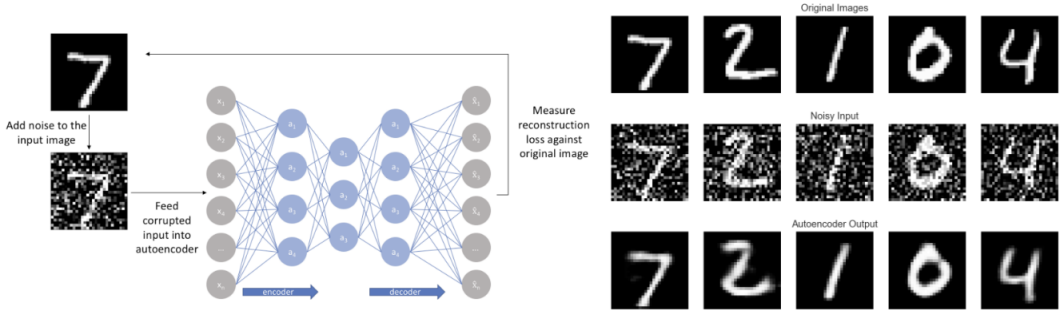


2. We introduce a corruption process $C(\tilde{x}|x)$.

AE's: Denoising Autoencoders (Training Procedure)

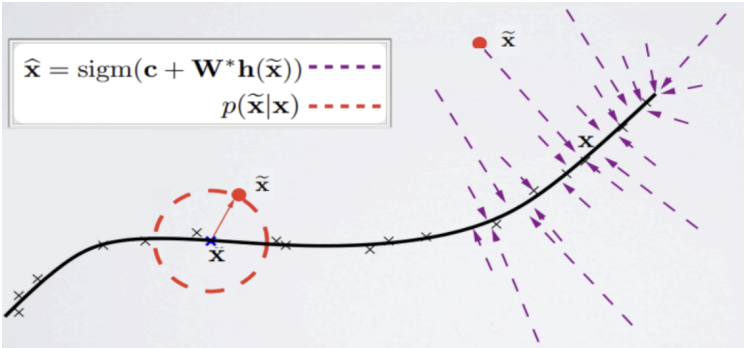
1. We introduce a corruption process $C(\tilde{x}|x)$.
2. The autoencoder then learns a reconstruction distribution $p_{\text{reconstruct}}(x|\tilde{x})$ estimated from training pairs $(x, \tilde{x})$ as follows:
   - Sample a training example x_i from the training data.
   - Sample a corrupted version $\tilde{x}_i$ from $C(\tilde{x}_i|x = x_i)$.
   - Use $(x, \tilde{x})$ as a training example for estimating the autoencoder reconstruction distribution $p_{\text{reconstruct}}(x|\tilde{x}) = p_{\text{decoder}}(x|h)$ with h the output of encoder $f(\tilde{x})$ and p_{decoder} typically defined by a decoder $g(h)$.
   - Typically we can simply perform gradient-based approximate minimization on the negative log-likelihood $-\log p_{\text{decoder}}(x|h)$.

AE's: Denoising Autoencoders (Result On MNIST)



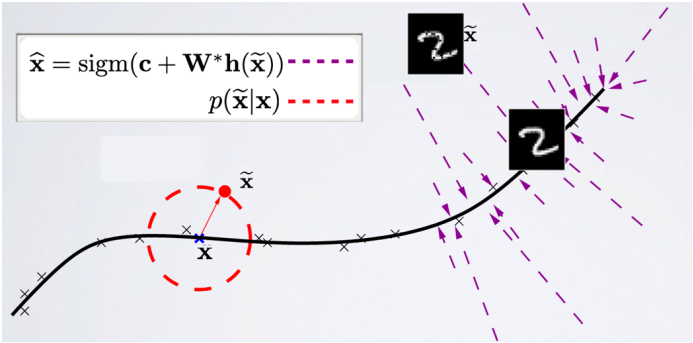
AE's: Denoising Autoencoders (It Isn't Lazy Network!)

- DAEs won't simply memorize the inputs and outputs (More robustness)
- Intuitively, a DAE learns a projection from a neighborhood of our training data back onto the training data



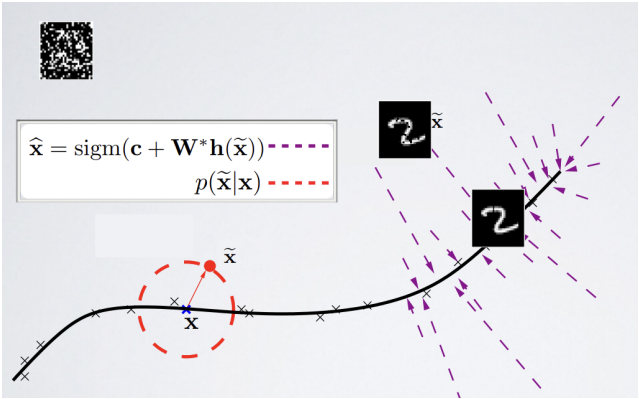
AE's: Denoising Autoencoders (It Isn't Lazy Network!)

- DAEs won't simply memorize the inputs and outputs (More robustness)
- Intuitively, a DAE learns a projection from a neighborhood of our training data back onto the training data



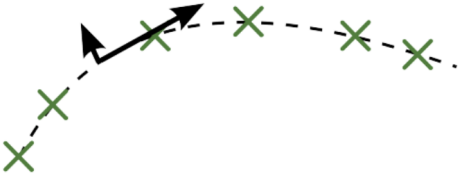
AE's: Denoising Autoencoders (It Isn't Lazy Network!)

- DAEs won't simply memorize the inputs and outputs (More robustness)
- Intuitively, a DAE learns a projection from a neighborhood of our training data back onto the training data

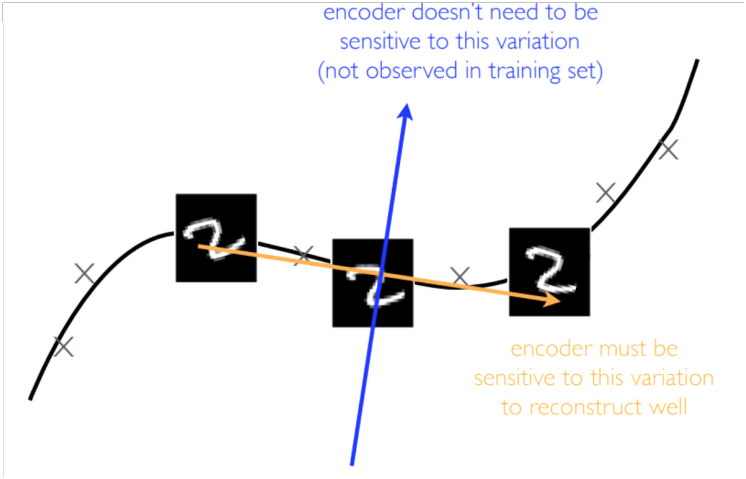


AE's: Contractive Autoencoder (Intuition)

1. Contractive autoencoders are explicitly encouraged to learn a manifold through their loss function (Alain and Bengio 2014).
2. Desirable property: Points close to each other in input space maintain that property in the latent space.
3. Method to avoid uninteresting solutions
4. Add an explicit term in the loss that penalizes that solution
5. We wish to extract features that only reflect variations observed in the training set
6. We would like to be invariant to other variations

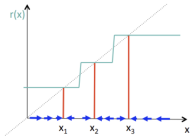


AE's: Contractive Autoencoder (Intuition)



AE's: Contractive Autoencoder (Details)

1. Contractive autoencoder has an explicit regularizer on $h = f(x)$, encouraging the derivatives of f to be as small as possible.
2. This will be true if $f(x) = h$ is continuous, has small derivatives.



3. We can use the Frobenius Norm of the Jacobian Matrix as a regularization term:

$$\Omega(f, x) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

4. These autoencoders are called *contractive* because they contract the neighborhood of input space into a smaller, localized group in latent space.
5. **Exercise:** What is the difference between DAE and CAE?

AE's: Contractive Autoencoder (Details)

- New loss function:

$$\underbrace{l\Big(f\Big(x^{(t)}\Big)\Big)}_{\texttt{autoencoder reconstruction}} + \lambda \underbrace{\|\nabla_{x^{(t)}} h(x^{(t)}\|_F^2}_{\texttt{Jacobian of encoder}}$$

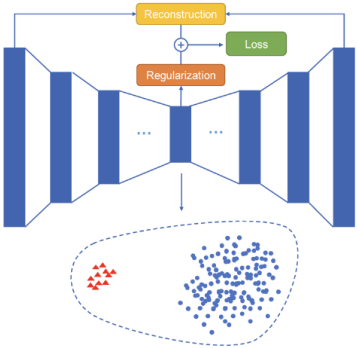
- where, for binary observations:

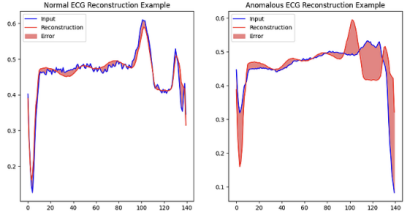
$$l\Big(f\Big(x^{(t)}\Big)\Big) = -\Sigma_k \left(x_k^{(t)} \log\Big(\hat{x}_k^{(t)}\Big) + \Big(1 - x_k^{(t)}\Big)\log\Big(1 - \hat{x}_k^{(t)}\Big)\right) \Big\} \texttt{ encoders keeps good information}$$

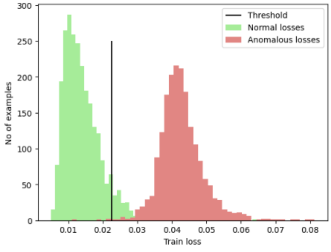
$$\|\nabla_{x^{(t)}} h(x^{(t)}\|_F^2 = \Sigma_j \Sigma_k \left(\frac{\partial h\Big(x^{(t)}\Big)_j}{\partial x_k^{(t)}}\right)^2 \Big\} \texttt{ encoder throws away all information}$$

$$\rightarrow \texttt{ encoders keep only good information}$$

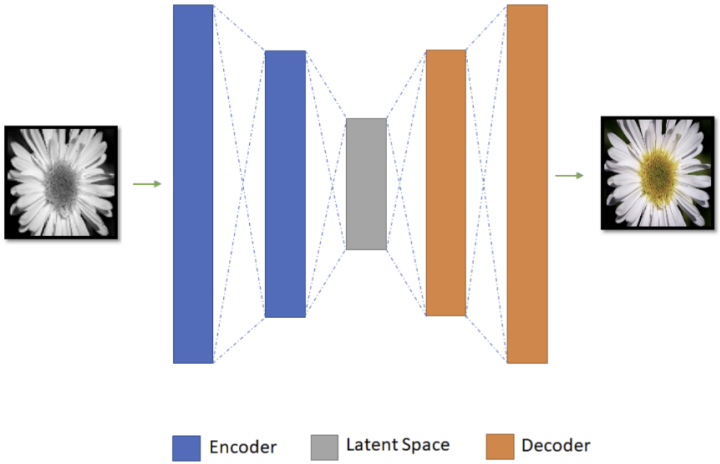
AE other Applications: Anomaly Detection



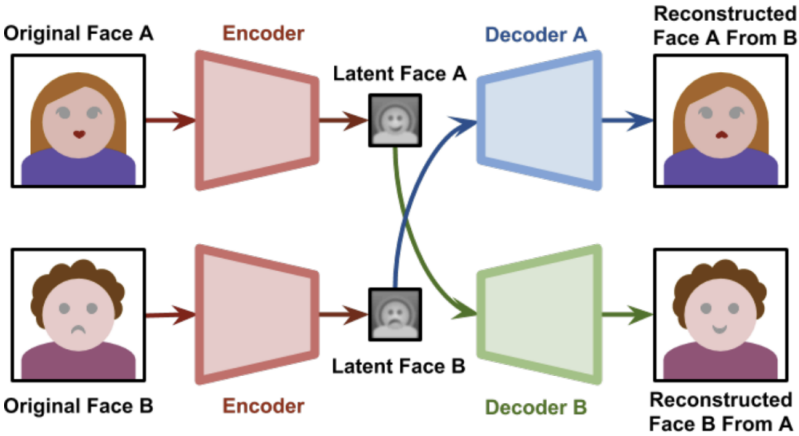




AE other Applications: Image Colorization

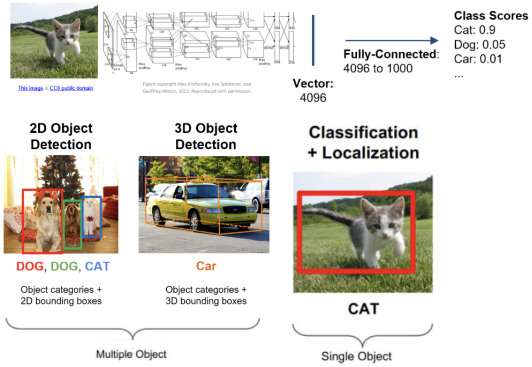


AE other Applications: Fake Lough



Problem Setup

Before...

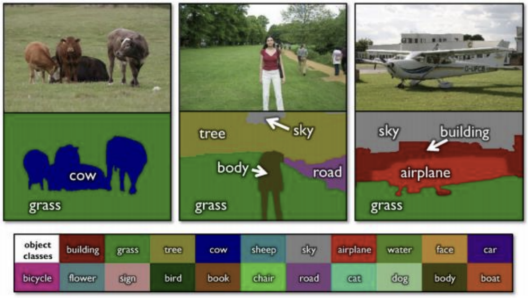


Now...

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.

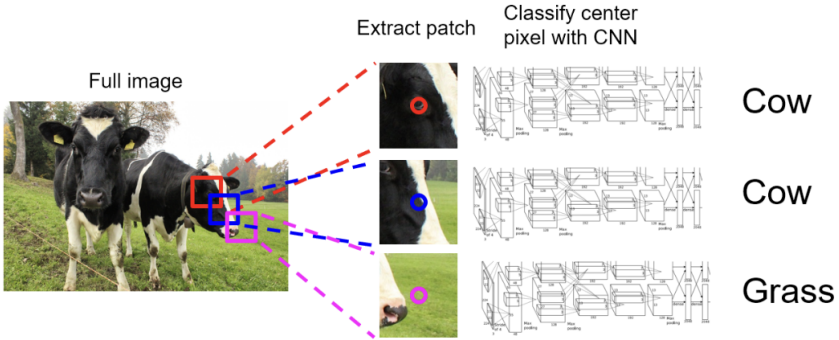
Problem Setup



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

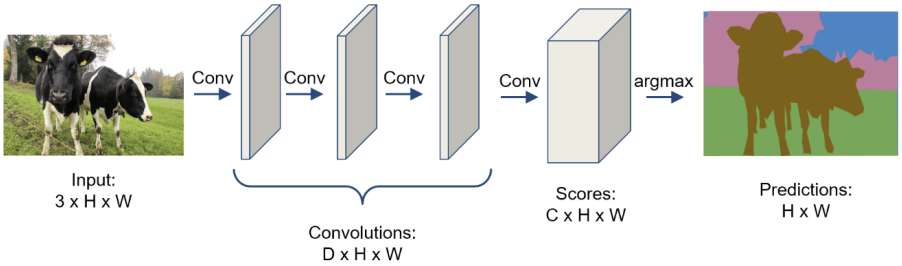


Problem: Very inefficient! Not reusing shared features between overlapping patches

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!

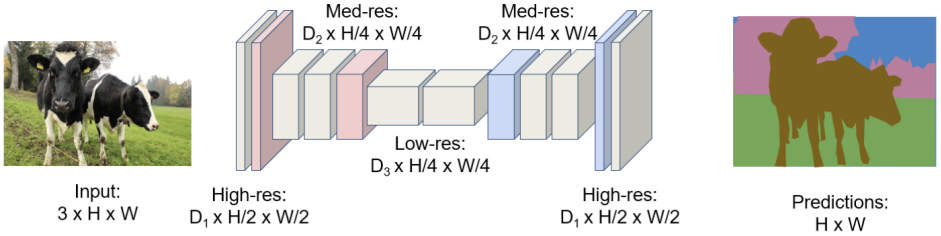


Problem: convolutions at original image resolution will be very expensive ...

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

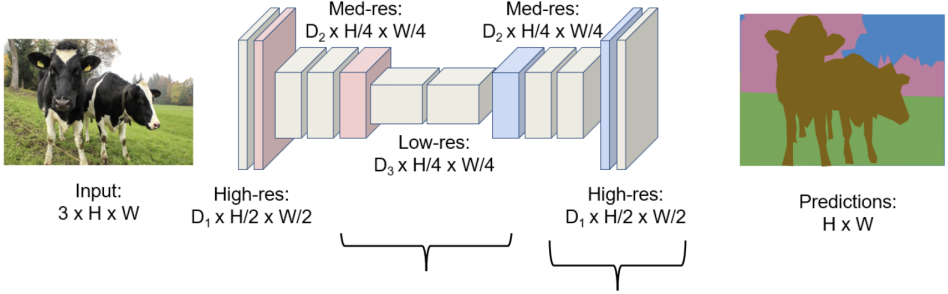
- Max Pooling
- Average Pooling
- Global Average Pooling
- Strided Convolution
- Spatial Pyramid Pooling
- ...

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”

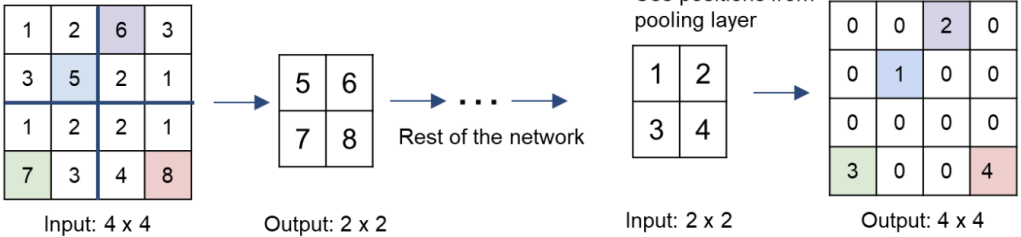
1	2
3	4

→

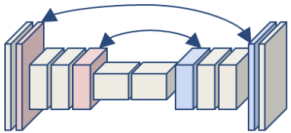
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input: 2 x 2 Output: 4 x 4

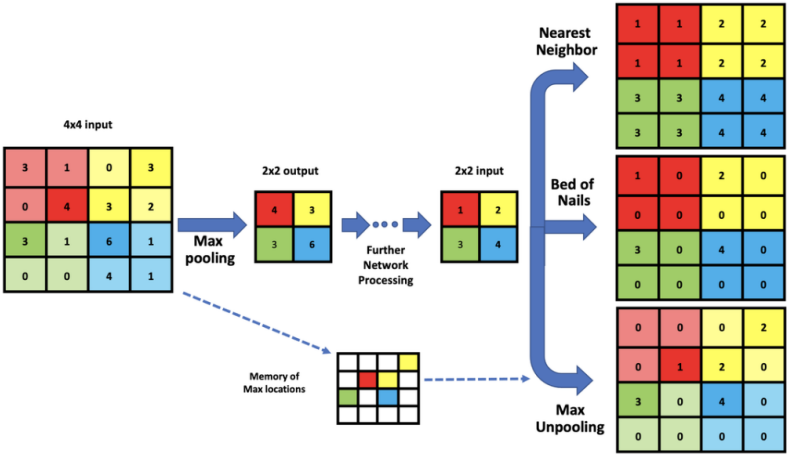
Static Upsampling: “Max Unpooling”



Corresponding pairs of downsampling and upsampling layers



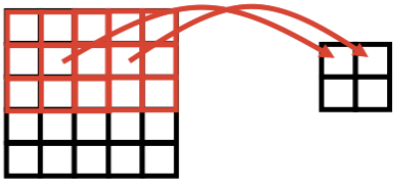
Static Upsamplings



Convolution and Transposed Convolution

Normal Convolutions: reduce resolution with **stride**

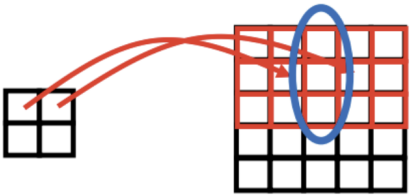
Stride = 2



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:

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

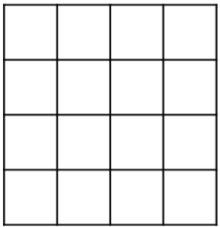
- Since the transposed convolution uses the **transpose** of the kernel matrix and effectively reverses the compression process (convolution), it's called **"Transposed Convolution"**.

Other names:

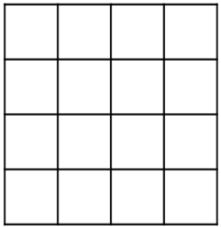
- Deconvolution (bad name)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

Learnable Upsampling: “Transpose Convolution”

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



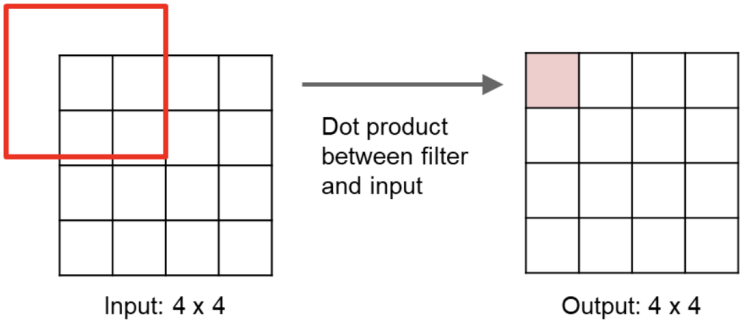
Input: 4 x 4



Output: 4 x 4

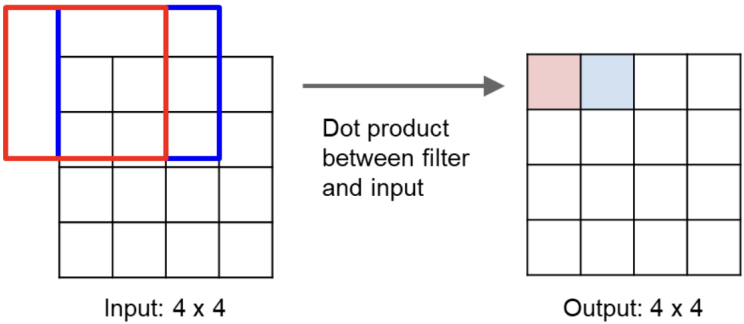
Learnable Upsampling: “Transpose Convolution”

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



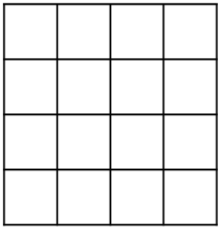
Learnable Upsampling: “Transpose Convolution”

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

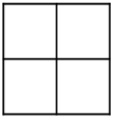


Learnable Upsampling: “Transpose Convolution”

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



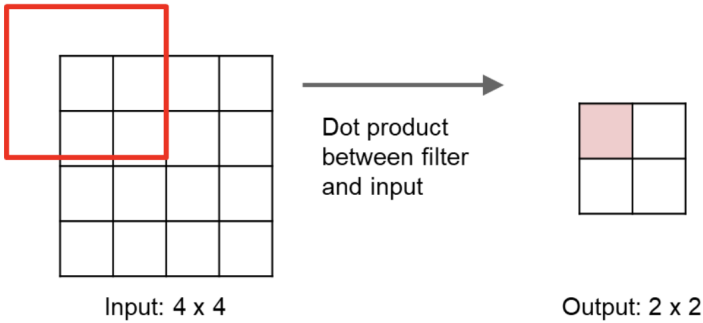
Input: 4 x 4



Output: 2 x 2

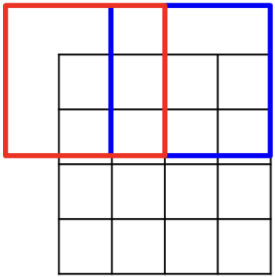
Learnable Upsampling: “Transpose Convolution”

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

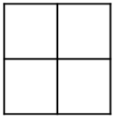
Output: 2 x 2

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

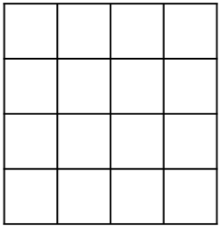
Stride gives ratio between movement in input and output

Learnable Upsampling: “Transpose Convolution”

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

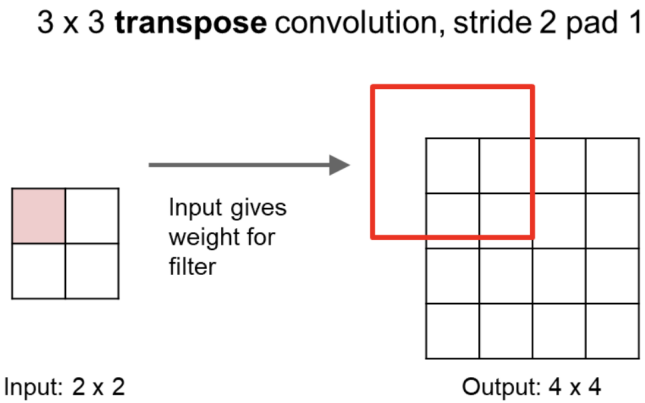


Input: 2 x 2

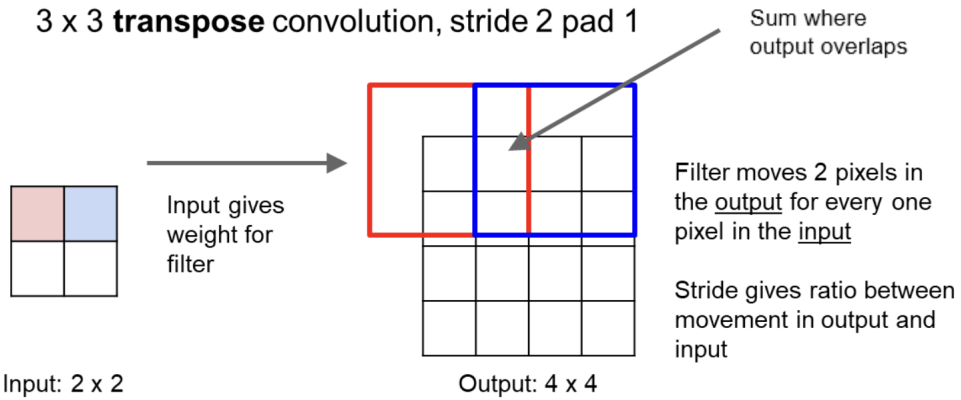


Output: 4 x 4

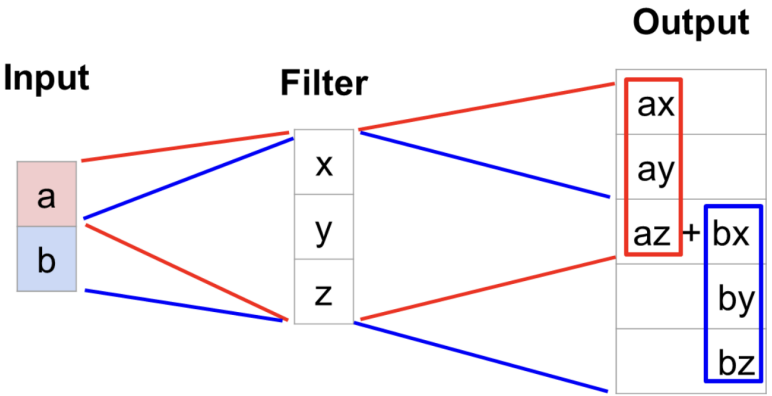
Learnable Upsampling: “Transpose Convolution”



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

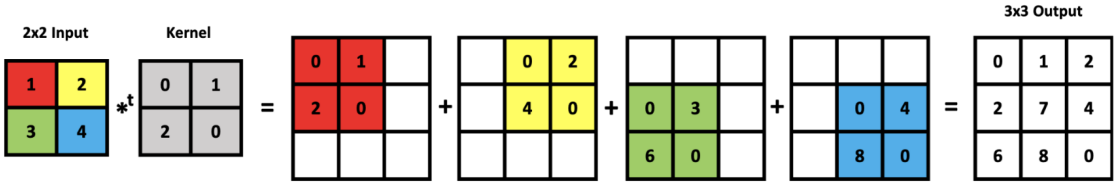
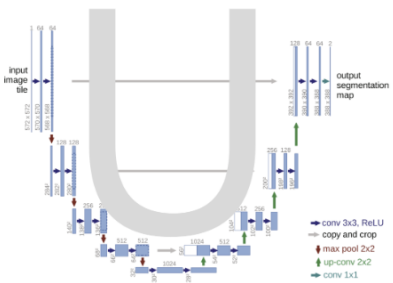
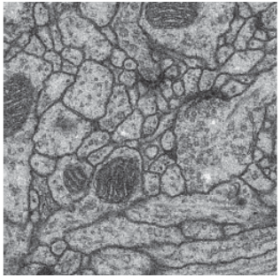


Image from this link

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.





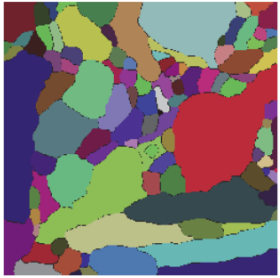
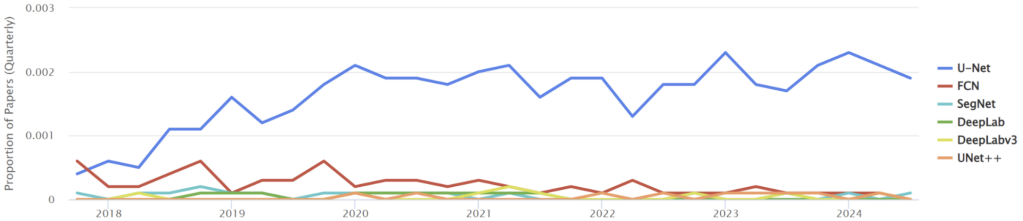


Image from this link

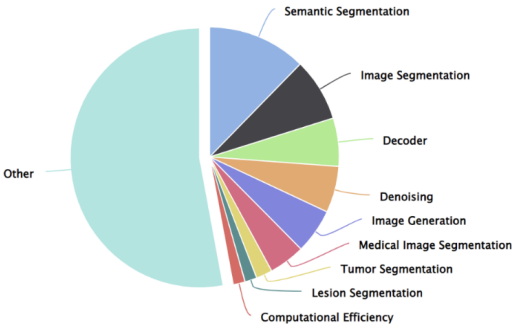
U-Net: Usage Over Time

- 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

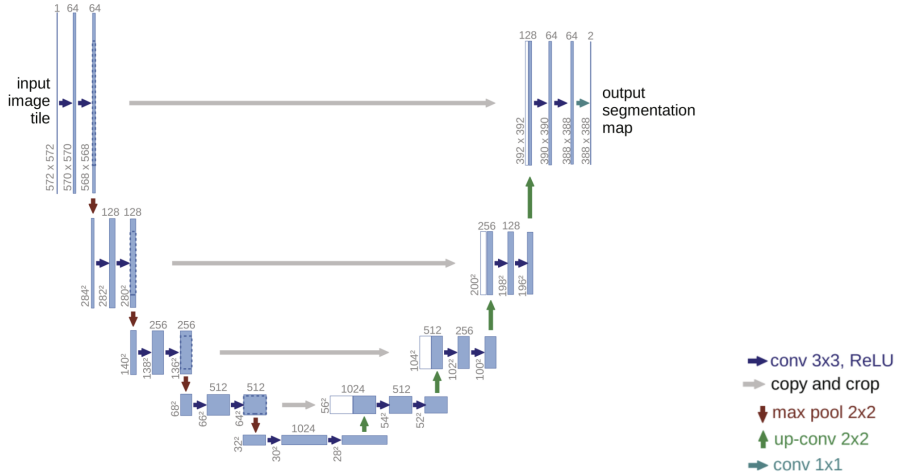
U-Net: Tasks



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%

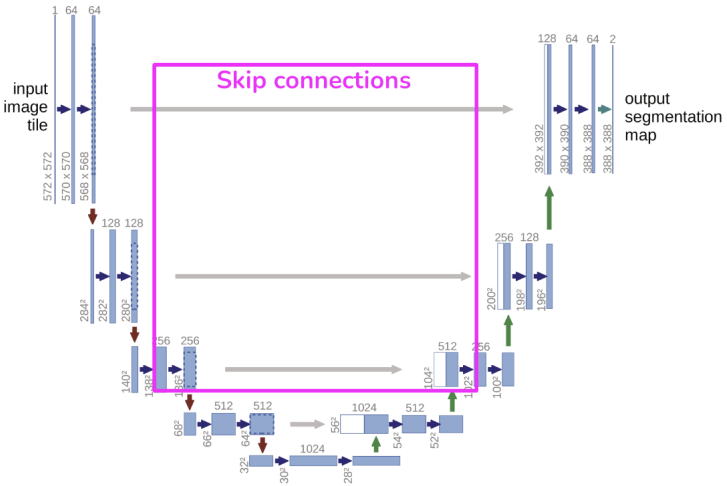
From papers with code

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.

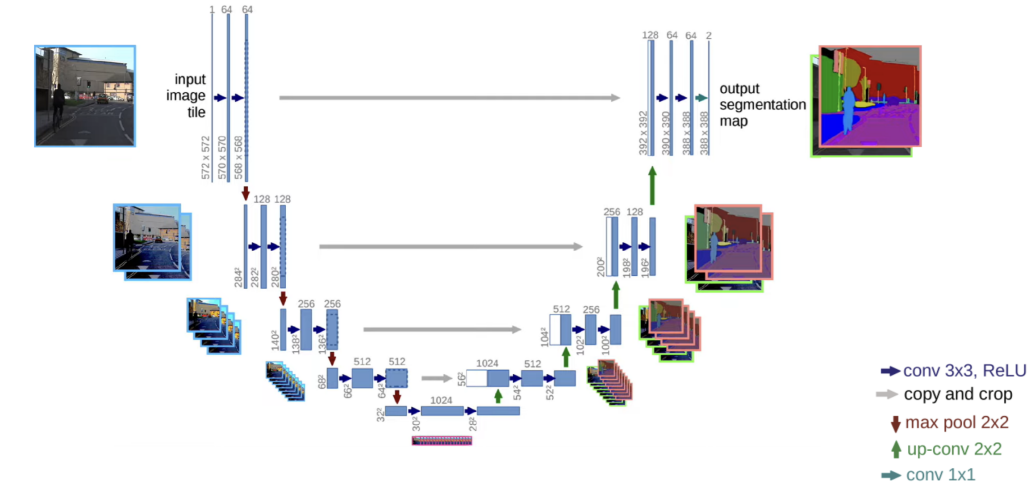
U-Net vs AE's: Latent Space

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.

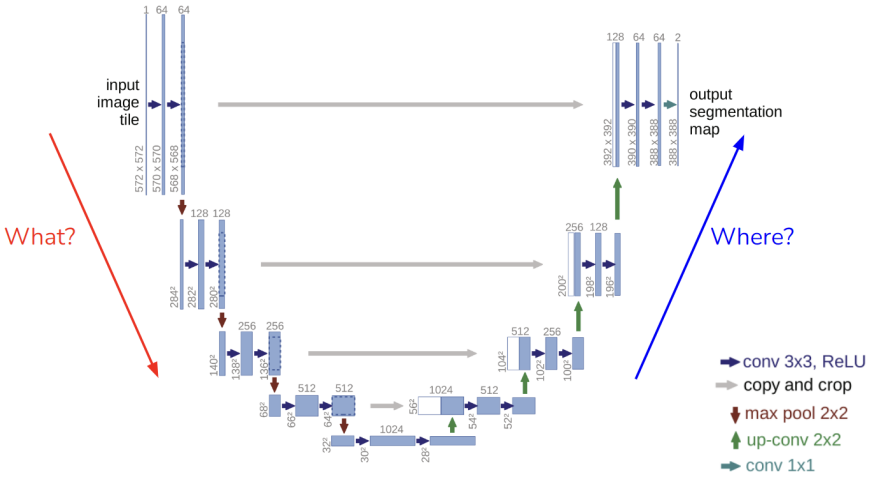
- **No Strong Encoding Constraint:** Since skip connections provide spatial detail, U-Net's latent space is not forced to develop an "expressive" representation, which is essential for models like VAEs where information passes strictly through a bottleneck.

U-Net: Big Picture



From this link

U-Net: Big Picture

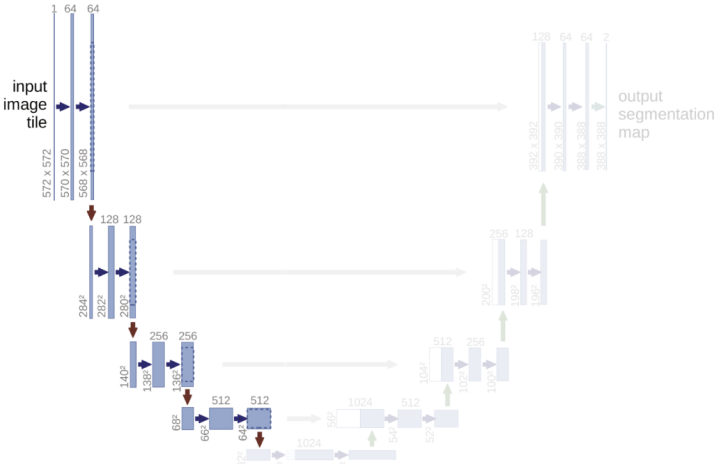


From Unet

U-Net: Big Picture

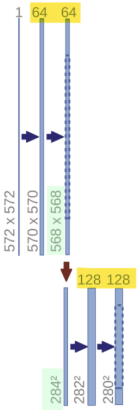
From this link

U-Net: Encoder



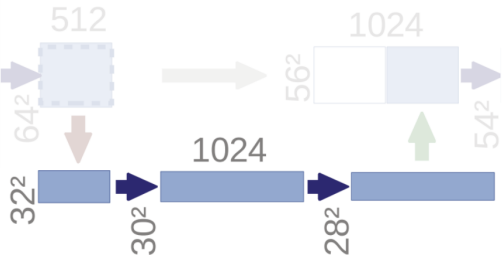
U-Net: Encoder

- 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



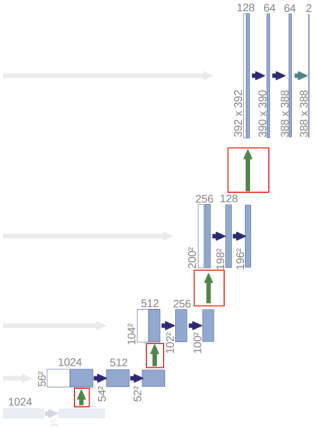
U-Net: Bottleneck

- 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.

- 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?