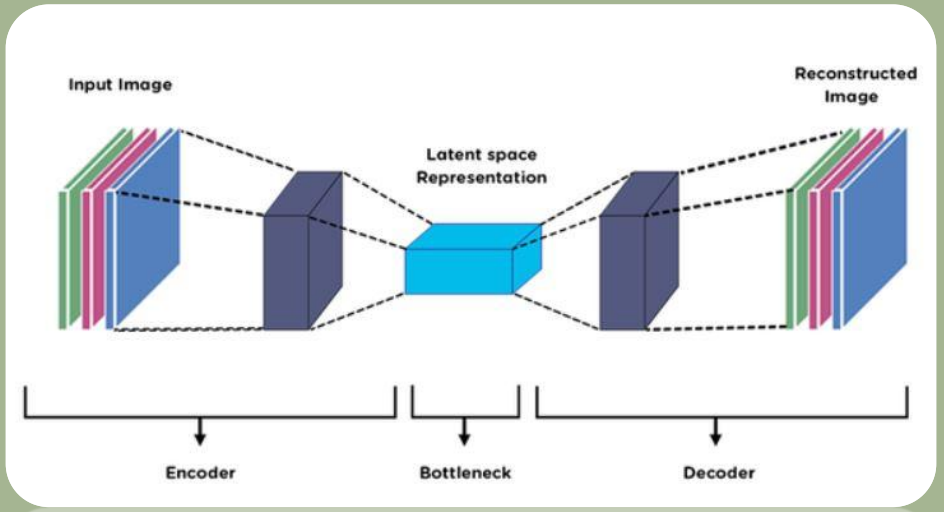
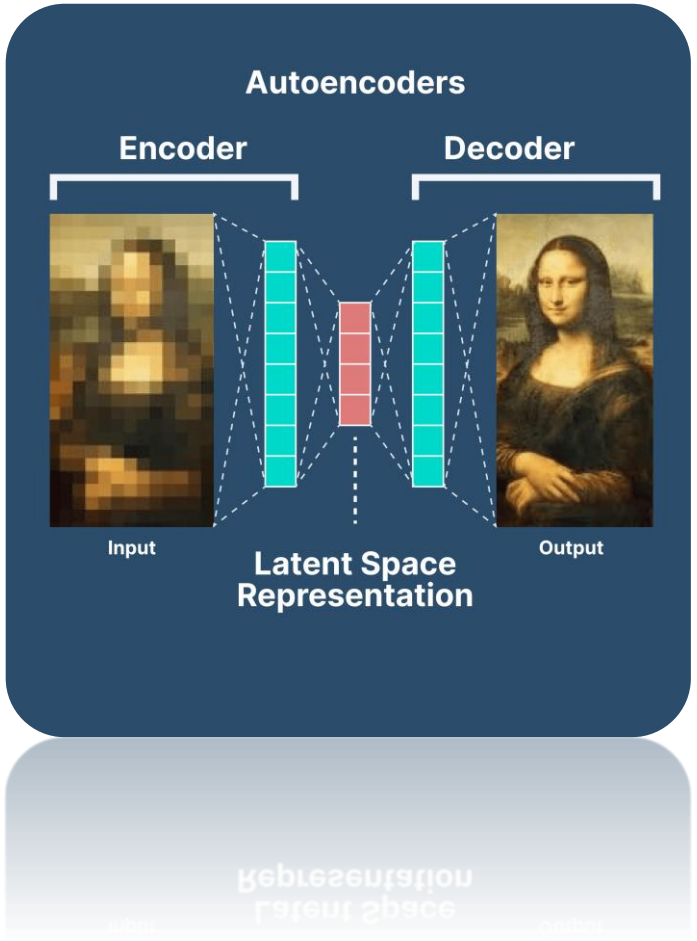
AUTOENCODERS & DIMENSIONALITY REDUCTION

MSSC 6931 – Fall 2025



Lecture Overview

- Motivation
- Dimensionality Reduction
- Fully-connected Autoencoders
- Implementation
- Different types of autoencoders



Autoencoder Applications

Denoising autoencoder

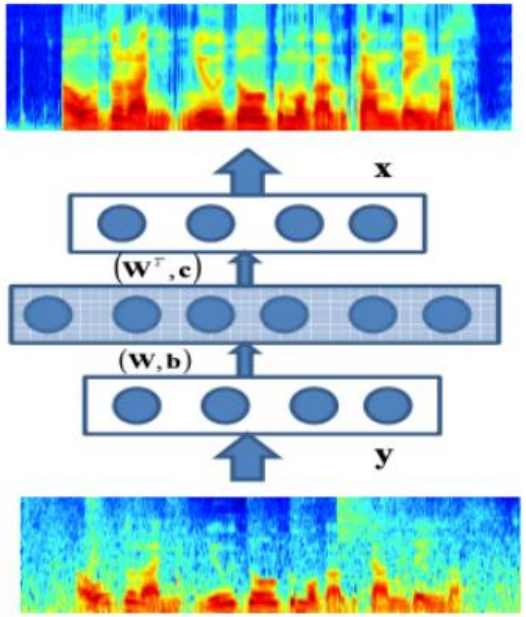
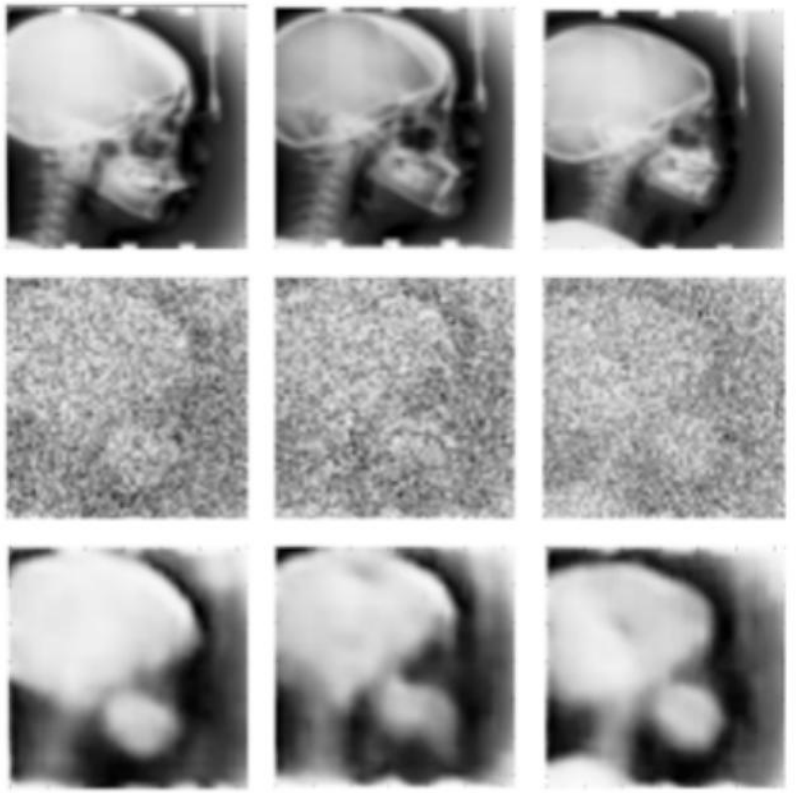


Figure 1: Training neural autoencoder with noisy-clean speech pairs.

Image enhancement



Original image

Noisy image

Recovered image

Gondara, L. (2016, December). Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)* (pp. 241-246). IEEE.

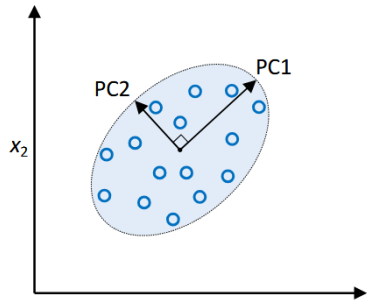
Reference: https://sebastianraschka.com/blog/2021/dl-course.html

Dimensionality Reduction

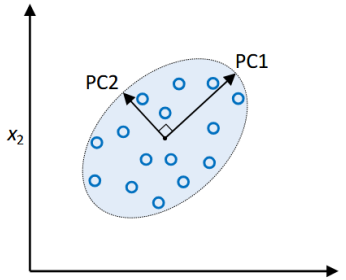
- Autoencoders are kind of feature extraction (where we don't keep the original features)
- It's an unsupervised learning (No target variable)
- Application and goals: • Finding hidden structures in data • Data compression • Clustering • Retrieving similar objects • Exploratory data analysis • Generating new examples

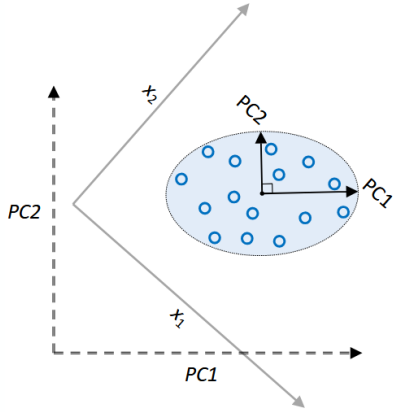
Principal Component Analysis (PCA)

1) Find directions of maximum variance

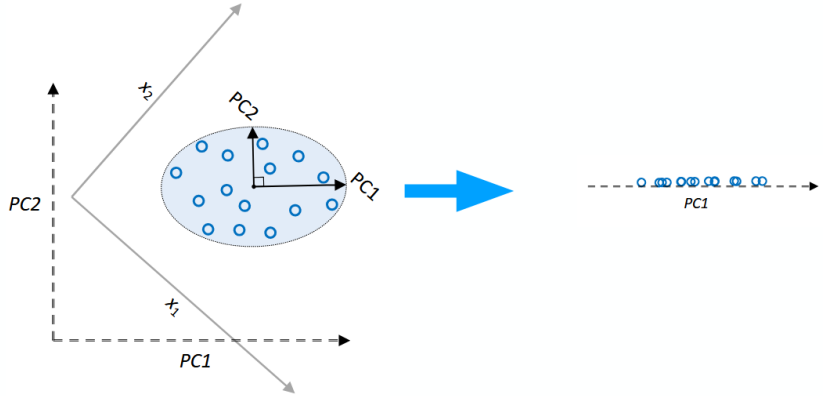


2) Transform features onto directions of maximum variance



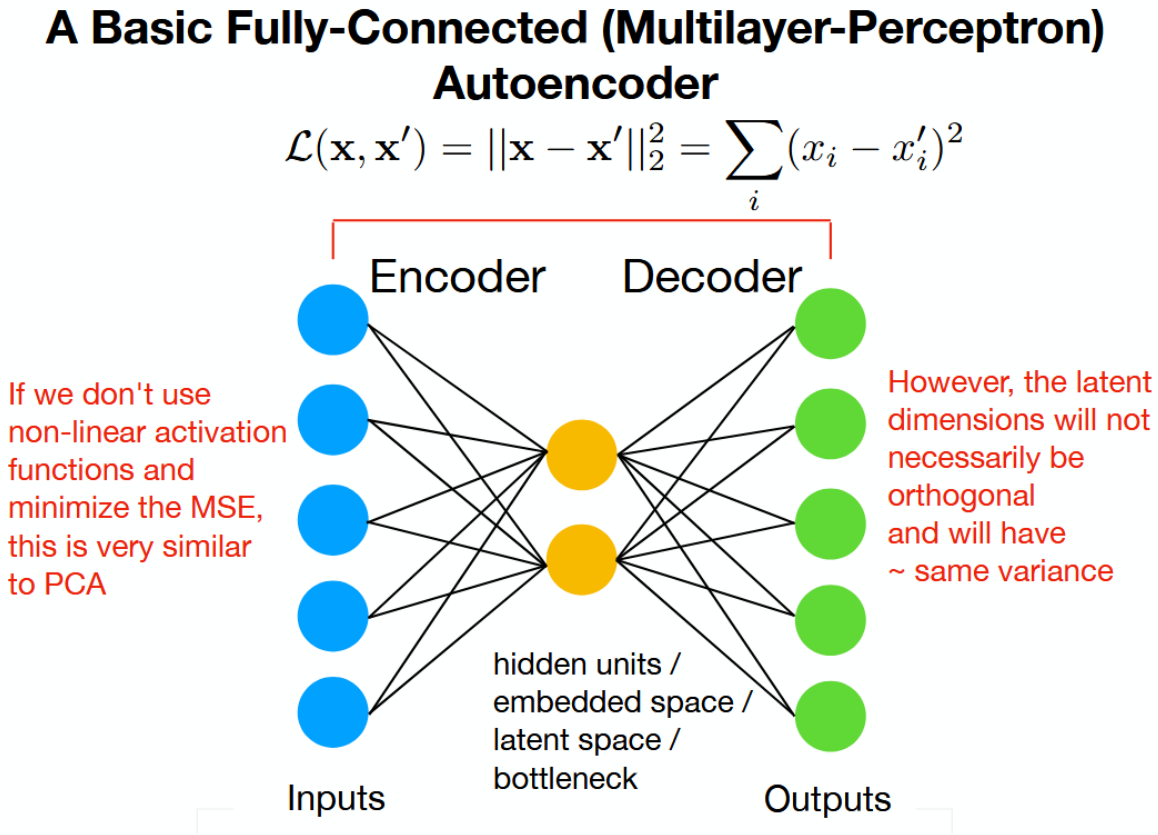


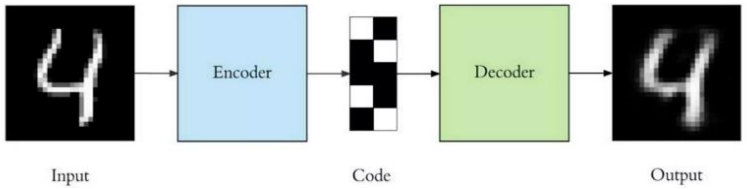
3) Usually consider a subset of vectors of most variance (dimensionality reduction)



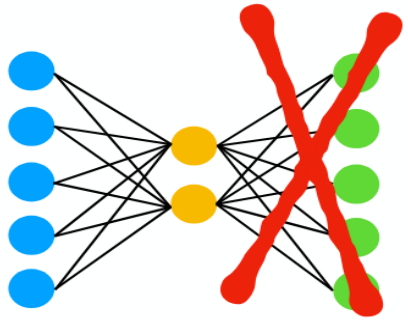
Reference: https://sebastianraschka.com/blog/2021/dl-course.html

Fully-connected Autoencoders



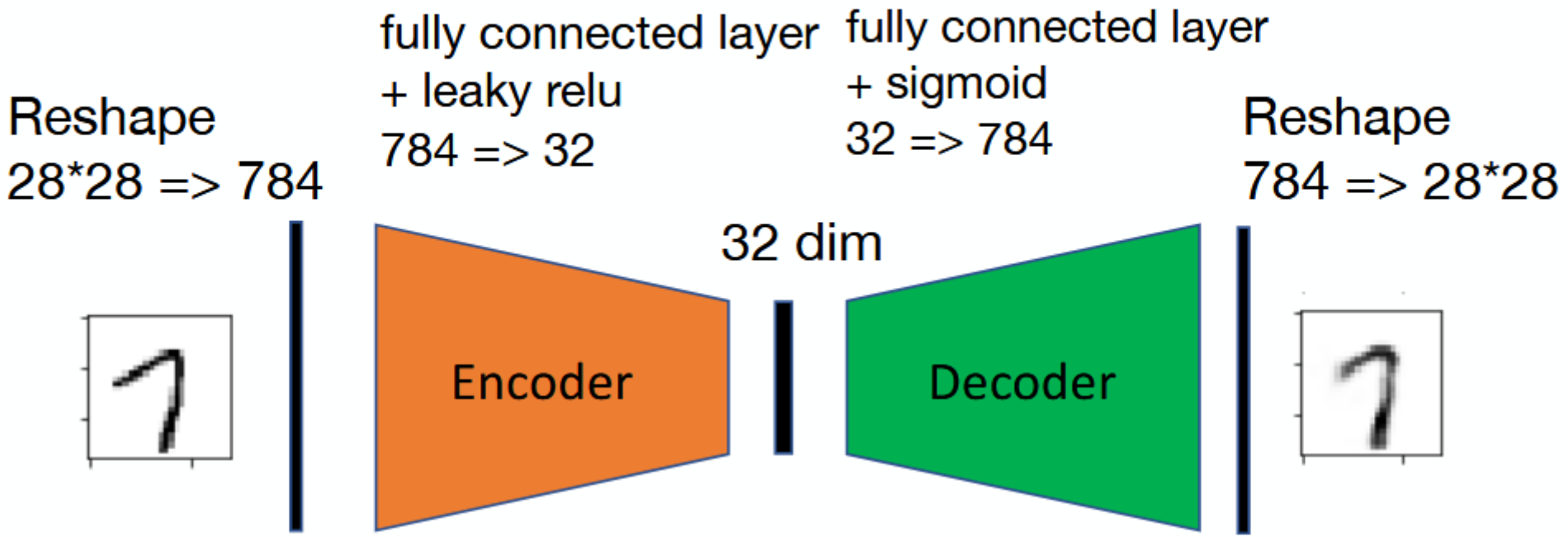


- The input is projected into the lower-dimensional latent space, and the output is a reconstructed version of that.
- No orthogonality constraints in this case.
- Potential Autoencoder Applications: After training, disregard this part:
  - Use embedding as input to classic machine learning methods (SVM, KNN, Random Forest, …)
  - Latent space can also be used for visualization (EDA, clustering)

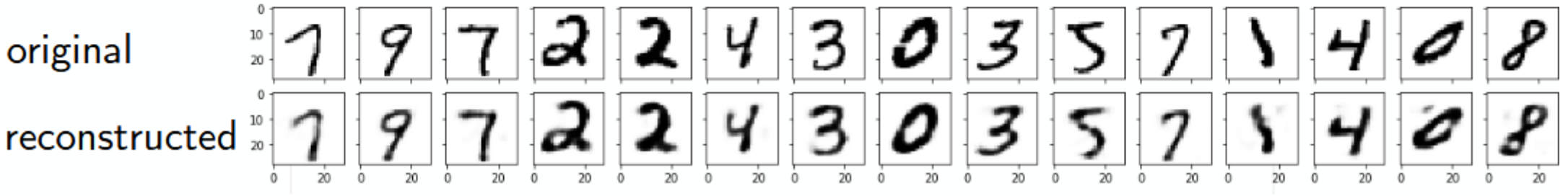


Reference: https://sebastianraschka.com/blog/2021/dl-course.html

A Simple Autoencoder

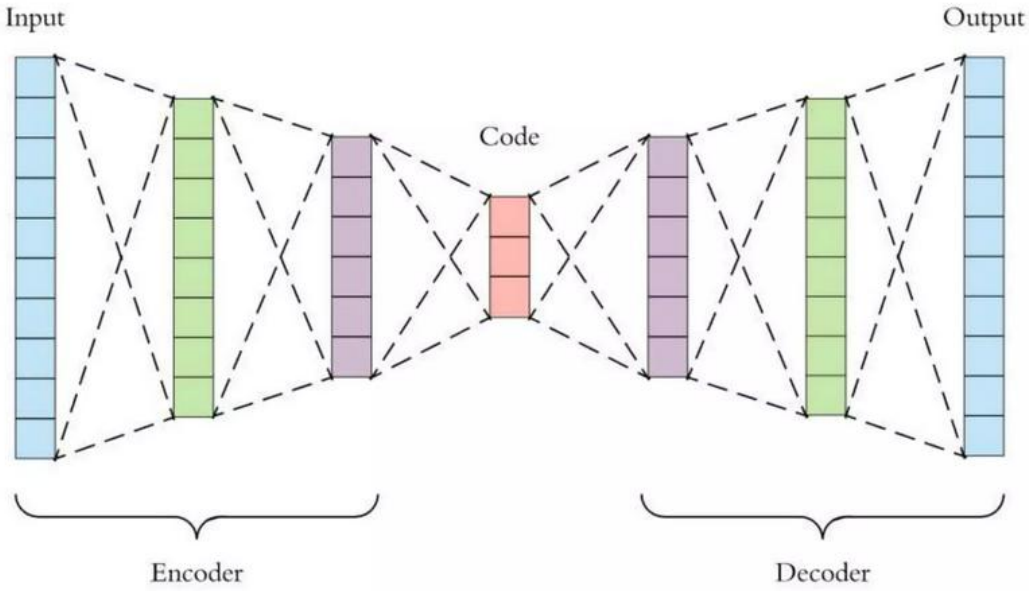


https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/autoencoder/ae-basic.ipynb



Reference: https://sebastianraschka.com/blog/2021/dl-course.html

More on Autoencoders



Hyperparameters:

Code Size: Number of nodes in the middle layer.
Number of layers: The encoder and decoder can be deep.
Number of nodes per layer
Loss Function: Either MSE or binary cross entropy. If the input values are in range [0,1] then we use cross entropy, otherwise MSE.

Types of Autoencoders:

- **Vanilla Autoencoder**
- **Multilayer Autoencoder**
- **Convolutional Autoencoder**
- **Denoising Autoencoder**
- **Regularized Autoencoder**
- **Sparse Autoencoder**
- **Variational Autoencoder**

Reference: https://www.slideshare.net/slideshow/autoencoder-141008858/141008858#6

Vanilla and Multilayer Autoencoders

Vanilla Autoencoder

• Vanilla Autoencoder are the simplest form used for unsupervised learning tasks. They consist of two main parts an encoder that compresses the input data into a smaller, dense representation and a decoder that reconstructs the original input from this compressed form.
• Training minimizes reconstruction error which measures the difference between input and output. This optimization is done via backpropagation which helps in updating the network weights to improve reconstruction accuracy.
• They are foundational models helps in serving as building blocks for more complex variants.

Applications

- **Data Compression:** They learn a compact version of the input data making storage and transmission more efficient.
- **Feature Learning:** It extract important patterns from data which is useful in image processing, natural language processing and sensor analysis.
- **Anomaly Detection:** If the reconstructed output is different from the original input, it can show an anomaly or outlier which makes autoencoders useful for fraud detection and system monitoring.

Implementation: https://github.com/mobinapourmoshir/Functional-Deep-Learning/blob/main/Autoencoders.ipynb

Multilayer Autoencoder:

If one hidden layer is not enough, we can add more hidden layers.

Reference: https://www.geeksforgeeks.org/numpy/types-of-autoencoders/

Convolution Autoencoders

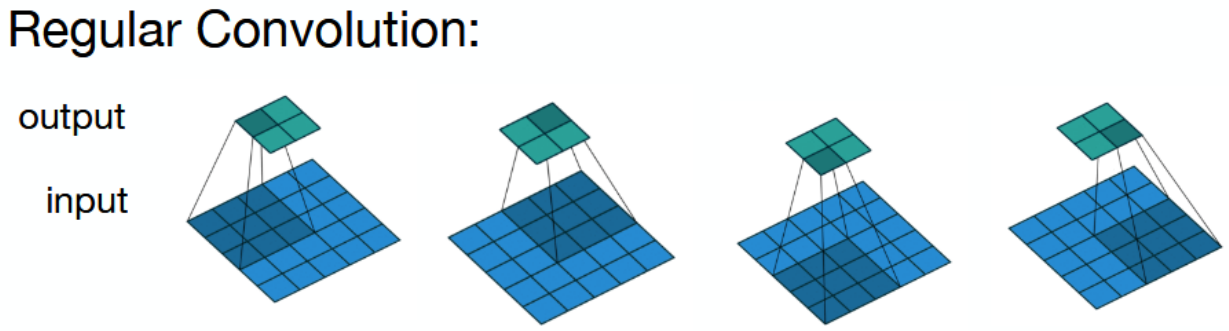
Convolution Autoencoder:

Instead of fully connected layers, one may use convolutional layers. Allows us to increase the size of the output feature map compared to the input feature map.

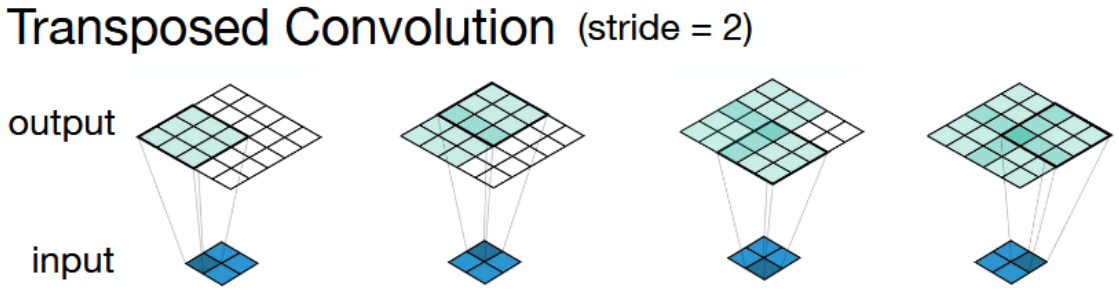
Image Reconstruction: Restores high-quality images from compressed latent codes.

Image Denoising: Removes noise while preserving spatial detail in images.

Feature Extraction: Captures hierarchical spatial features for tasks like classification and segmentation.



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).



A Conv2DTranspose with 3x3 kernel and stride of 2x2 applied to a 2x2 input to give a 5x5 output.
(https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8)

Reference: https://sebastianraschka.com/blog/2021/dl-course.html

Denoising Autoencoders

Denoising Autoencoder (DAE):

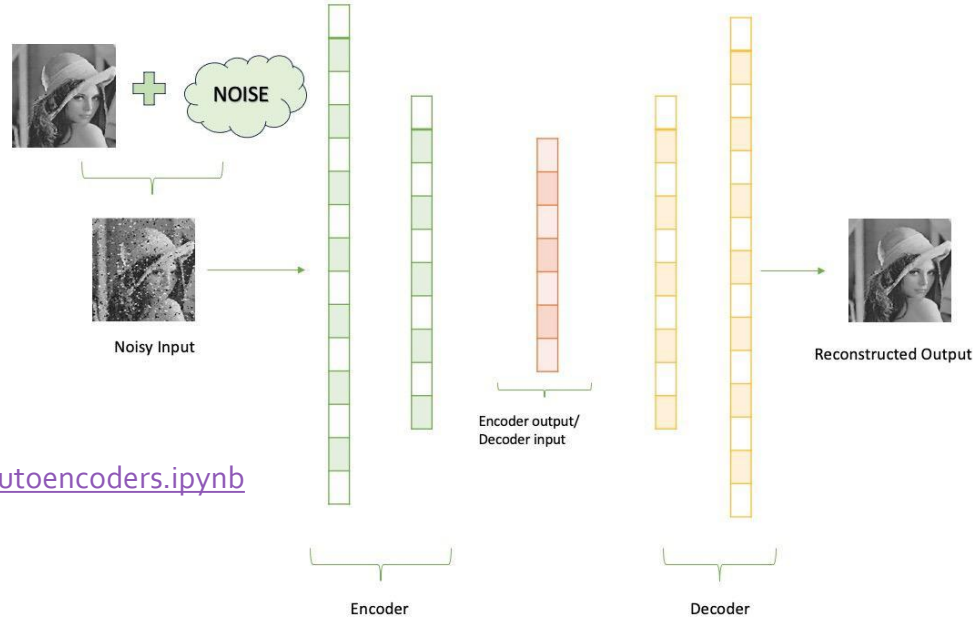
Denoising Autoencoders are designed to handle corrupted or noisy inputs by learning to reconstruct the clean, original data.
Training involves feeding intentionally corrupted inputs and minimizing the reconstruction error against the clean version.
This approach forces the model to capture robust features that are invariant to noise.

Applications

Image Denoising: Removes noise from images to increase quality and improve downstream processing.
Signal Cleaning: Filters noise from audio and sensor signals helps in boosting detection accuracy.
Data Preprocessing: Cleans corrupted data before input to other models helps in increasing robustness and performance.

Encoder: $h = f_\theta(\tilde{x})$ → compresses noisy input to latent representation
Decoder: $\hat{x} = g_\phi(h)$ → reconstructs clean data from latent code

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - g(f(\hat{x}))\|^2$$



Implementation: https://github.com/mobinapourmoshir/Functional-Deep-Learning/blob/main/Autoencoders.ipynb

Reference: https://www.slideshare.net/slideshow/autoencoder-141008858/141008858#6

Regularized Autoencoders

A **regularized autoencoder** adds **extra constraints** (regularization terms) to the loss function so that the learned latent representation h has desirable properties. e.g., being sparse, smooth, disentangled, or close to a prior distribution.
This helps the model learn **more meaningful and generalizable** features rather than just memorizing data.
A regular autoencoder minimizes the **reconstruction loss**:

$$\mathcal{L}_{Rec} = \|x - \hat{x}\|^2$$

In a **regularized autoencoder,** we modify the total loss as:

$$\mathcal{L} = \mathcal{L}_{Rec} + \lambda\mathcal{R}(h)$$

Where $\mathcal{R}(h)$ is a regularization term,λ>o is a tuning parameter that controls its strength.

Reference: https://www.slideshare.net/slideshow/autoencoder-141008858/141008858#6

Common Types of Regularization

Type	Input Setup	Loss Function	Regularization / Constraint	Goal / Effect
Basic Autoencoder	Clean input x	Reconstruction loss: $L = \|\|x - \hat{x}\|\|^2$	None	Learn compressed representation that can reconstruct input.
Denoising Autoencoder (DAE)	Noisy input $\tilde{x}$ = x + noise	$L = \|\|x - g(f(\hat{x}))\|\|^2$	Implicit (via noise corruption)	Learn robust features; denoise corrupted data.
Sparse Autoencoder (SAE)	Clean input x	$\mathcal{R}(h) = \sum_j \mathrm{KL}(\rho \,\|\, \hat{\rho}_j)$ or $\mathcal{R}(h) = \|h\|_1$	Enforce sparsity on hidden activations	Learn interpretable, sparse latent features.
Contractive Autoencoder (CAE)	Clean input x	$L = \|x - \hat{x}\|^2 + \lambda \|\nabla_x f_\theta(x)\|_F^2$	Penalize encoder's sensitivity (Jacobian norm)	Make representation invariant to small input changes (smooth manifold).
Variational Autoencoder (VAE)	Clean input x, probabilistic latent h	$L = \mathbb{E}_{q_\phi(h\|x)}[\|x - g_\phi(h)\|^2] + \beta\, D_{\mathrm{KL}}(q_\theta(h\|x) \,\|\, p(h))$	KL divergence between encoder q(h|x) and prior p(h); enforces latent space to follow a standard normal distribution.	Learn a smooth, continuous, and structured latent space that allows data generation, interpolation, and sampling.
Weight-Regularized Autoencoder	Clean input x	$L = \|x - \hat{x}\|^2 + \lambda \|W\|_2^2$	Penalize network weights (L2/L1)	Prevent overfitting; improve generalization.

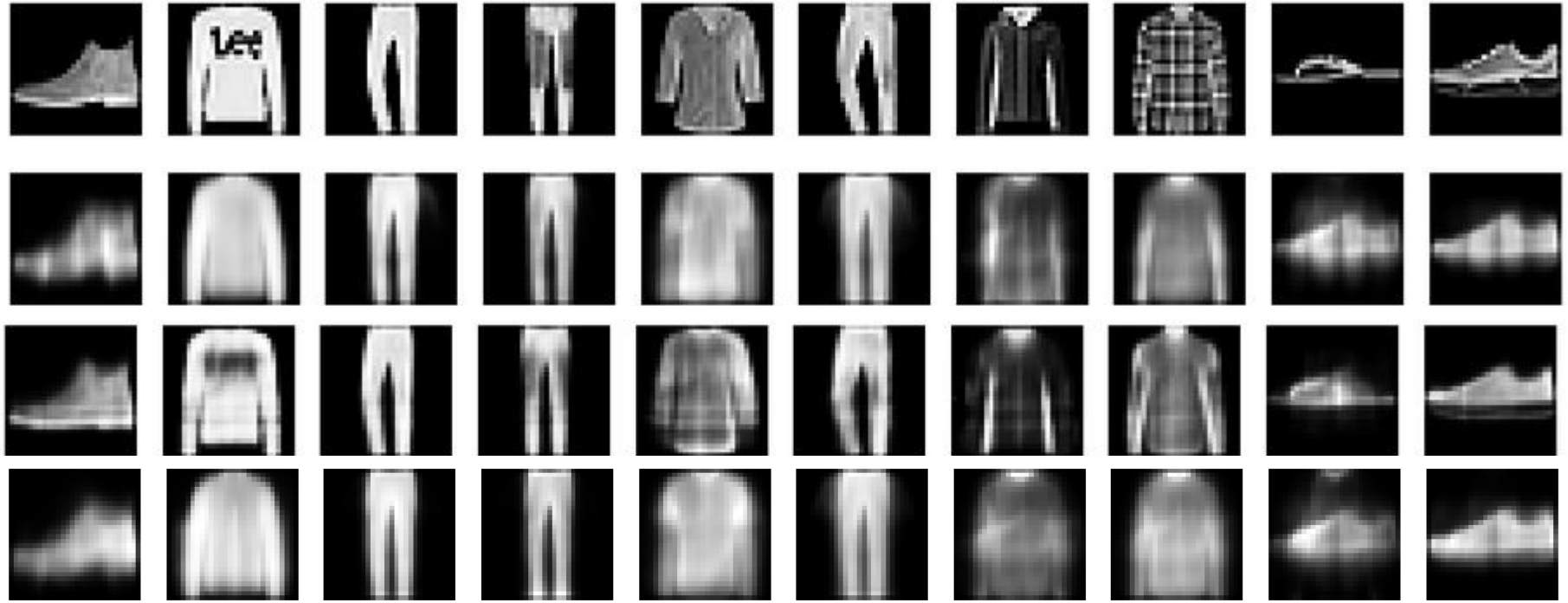
Implementation: https://github.com/mobinapourmoshir/Functional-Deep-Learning/blob/main/Autoencoders.ipynb

Question?

Can we combine these regularization terms to have a more comprehensive and accurate autoencoder mode?

The answer is YES! As an example combining smoothness and sparsity is both valid and powerful. It helps learn structured, robust, and interpretable representations by blending local stability (smoothness) with parsimony (sparsity).
It can be done by combining L1 and contractive (Jacobian) penalties as below:

$$\mathcal{L} = \|x - \hat{x}\|^2 + \lambda_1 \|h\|_1 + \lambda_2 \|\nabla_x f_\theta(x)\|_F^2$$



THANK YOU!