

MO433 - Unsupervised Learning

Introduction to Image Generation

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

Introduction to image generation

- ▶ So far, we have focused on creating meaningful latent representations to cluster images.
- ▶ We can also use these representations to **generate new images**.
- ▶ However, the process counts on a decoder (e.g., PCA inverse transform, autoencoder decoder).

Agenda

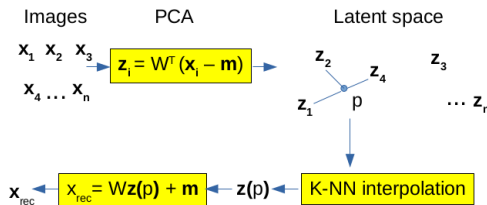
Image generation/reconstruction with

- ▶ PCA and autoencoders,
- ▶ Variational autoencoders,
- ▶ Application to deepfakes using UNet autoencoders.

Image generation with PCA

Given a PCA inverse transform as decoder, we can generate a new image by:

1. Selecting a query point p in the latent space,
2. Finding k nearest **encoded samples** $\{z_1, \dots, z_k\}$,
3. Interpolating the latent vector $z(p)$, and
4. Reconstructing the image: $x_{\text{rec}} = \text{Decoder}(z(p))$.



k-NN interpolation in latent space

Distance-based interpolation:

$$z(p) = \sum_{i=1}^k w_i z_i$$

where z_i are the k nearest real encoded samples to p , with distance $d_i = \|p - z_i\|$.

Inverse distance weighting:

$$w_i = \frac{d_i^{-1}}{\sum_{j=1}^k d_j^{-1}}$$

Why this works: By interpolating among real encoded samples at $z(p)$, we stay on/near the data manifold, avoiding empty locations p where the decoder produces unrealistic images.

Alternative interpolation methods

Different weighting schemes control the influence of nearby samples:

- ▶ **Gaussian kernel weighting:**

$$w_i = \frac{\exp\left(\frac{-d_i^2}{2\sigma^2}\right)}{\sum_{j=1}^k \exp\left(\frac{-d_j^2}{2\sigma^2}\right)}.$$

Smooth falloff, σ controls locality.

- ▶ **Shepard's method (power-inverse):**

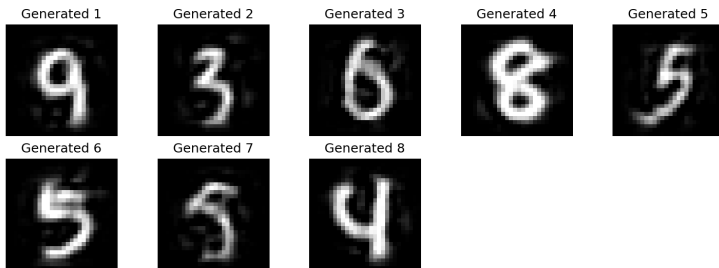
$$w_i = \frac{d_i^{-q}}{\sum_{j=1}^k d_j^{-q}}, \quad q \geq 1.$$

Larger q gives more weight to nearest neighbors.

PCA limitation

- ▶ PCA is limited to linear projections, performing well for simple images only.
- ▶ A Gaussian Mixture Model may fit the manifold to directly estimate the latent vector $z(p)$, but this requires very accurate fitting.

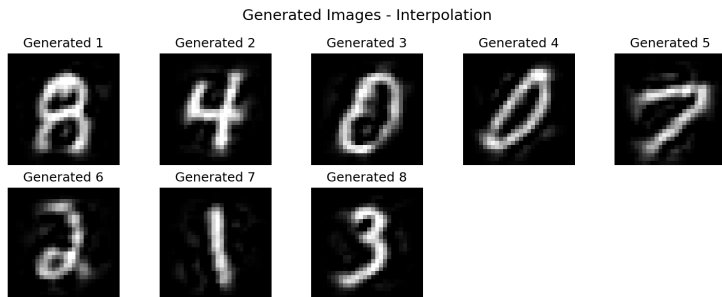
Generated Images - Weighted Neighbors



See [code1-gmm-image-generation.py](#).

PCA limitation

- ▶ PCA is limited to linear projections, performing well for simple images only.
- ▶ A Gaussian Mixture Model may fit the manifold to directly estimate the latent vector $z(p)$, but this requires very accurate fitting.

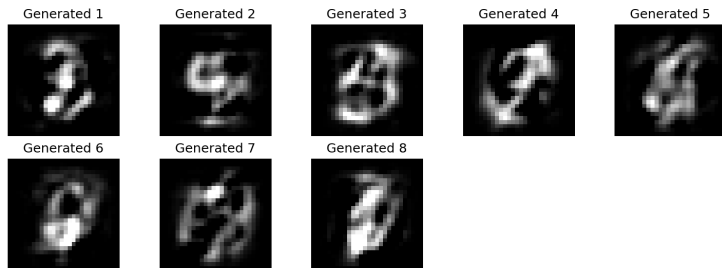


See [code1-gmm-image-generation.py](#).

PCA limitation

- ▶ PCA is limited to linear projections, performing well for simple images only.
- ▶ A Gaussian Mixture Model may fit the manifold to directly estimate the latent vector $z(p)$, but this requires very accurate fitting.

Generated Images - Gaussian Mixture

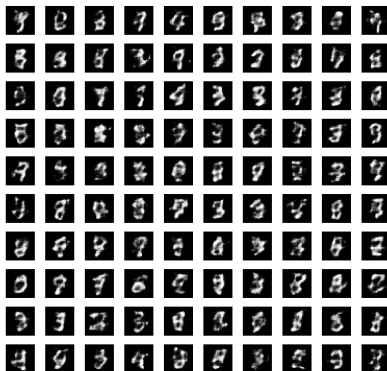


See [code1-gmm-image-generation.py](#).

Autoencoders and their limitation

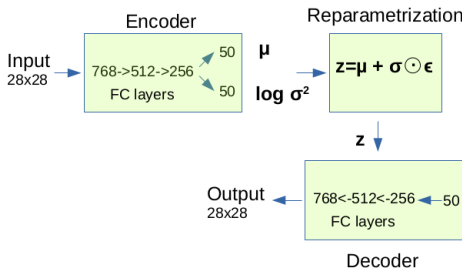
- ▶ Autoencoders provide non-linear projections.
- ▶ However, their latent space is **sparse**, requiring k-NN interpolation to decode $z(p)$.
- ▶ No interpolation leads to unrealistic reconstructions from p , as shown in code2-conv_autoencoder.py of a previous lecture.

Random Generated Images from Latent Space



Variational autoencoders (VAE)

Variational autoencoders can provide a **dense** latent space to dismiss interpolation while sampling new z .



Key differences from standard AE:

- ▶ Encoder outputs two vectors $\mu \in \mathbb{R}^{50}$, $\log \sigma^2 \in \mathbb{R}^{50}$.
- ▶ Sampling step: $z = \mu + \sigma \odot \epsilon$, where the noise $\epsilon \sim \mathcal{N}(0, I)$.
- ▶ Loss = reconstruction + β KL divergence, where $\beta > 0$.

Reparametrization and VAE loss function

Reparametrization Trick:

Sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, then compute:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon$$

This allows backpropagation through sampling:

$$\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$$

VAE Loss Function:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\|\mathbf{x} - \hat{\mathbf{x}}\|^2}_{\mathcal{L}_{\text{recon}}} + \beta \underbrace{\text{KL}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))}_{\mathcal{L}_{\text{KL}}}$$

Since $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ and $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$, the KL term has closed form:

$$\mathcal{L}_{\text{KL}} = \frac{1}{2} \sum_{j=1}^d (\mu_j^2 + \sigma_j^2 - 1 - \log \sigma_j^2)$$

The KL term pushes $\boldsymbol{\mu} \rightarrow 0$ and $\boldsymbol{\sigma}^2 \rightarrow 1$, regularizing the latent space to be standard normal!

Understanding β and KL divergence

► **Standard Autoencoder:**

- Encoder can place z anywhere.
- Latent space is sparse with holes.
- Random sampling fails!

► **Variational Autoencoder (β -VAE):**

- KL regularizes encodings toward $\mathcal{N}(\mu, \sigma^2)$.
- Latent codes cluster around origin.
- Smooth, continuous latent space.
- Random sampling works!

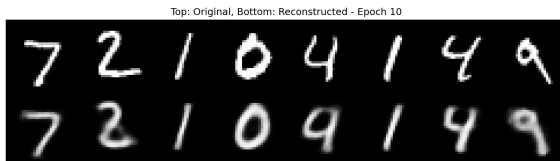
► **Role of β :** Controls reconstruction vs. regularization tradeoff.

- $\beta = 1$: Standard VAE.
- $\beta > 1$: Disentangled representations (each z_i captures one factor: position, scale, rotation in object generation with conv-VAE)

See [code2-vae.py](#)

Generation and reconstruction qualities

- **Blurrier reconstruction (second row):**



- **Better random generation (all random digits):**



Two different goals for autoencoders

Image Generation:

- ▶ Sample from latent space.
- ▶ Create new, diverse images.
- ▶ Use case: Data augmentation.
- ▶ **Solution is VAE.**

Image Reconstruction:

- ▶ Accurate pixel-level detail.
- ▶ Preserve spatial information.
- ▶ Use cases: Denoising, super-resolution, deepfakes.
- ▶ **Solution is U-Net autoencoders** – skip connections improve details.

Two different goals for autoencoders

Image Generation:

- ▶ Sample from latent space.
- ▶ Create new, diverse images.
- ▶ Use case: Data augmentation.
- ▶ **Solution is VAE.**

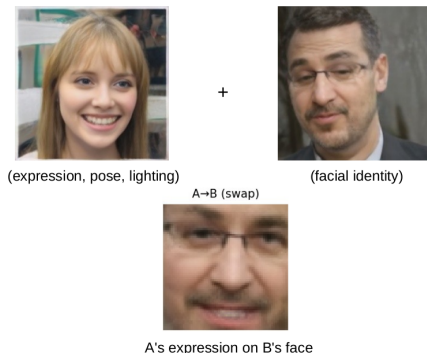
Image Reconstruction:

- ▶ Accurate pixel-level detail.
- ▶ Preserve spatial information.
- ▶ Use cases: Denoising, super-resolution, deepfakes.
- ▶ **Solution is U-Net autoencoders** – skip connections improve details.

Convolutional VAE is not the solution for more complex image generation, but it is useful for stable diffusion.

Application: Face swapping (deepfake)

Goal: Transfer facial identity while preserving expression and pose.



Key challenges:

- ▶ Separate identity from expression/pose/lighting.
- ▶ Preserve fine details (wrinkles, skin texture).
- ▶ Maintain color consistency and realism.

Face swapping

Goal: Given two people A and B, learn mappings:

$$f_A : \mathcal{X} \rightarrow \mathcal{X}_A \quad \text{and} \quad f_B : \mathcal{X} \rightarrow \mathcal{X}_B$$

where \mathcal{X} is the space of face images, and $\mathcal{X}_A, \mathcal{X}_B$ are identity-specific subspaces.

Decompose each mapping as:

$$f_A = D_A \circ E \quad \text{and} \quad f_B = D_B \circ E$$

where:

- ▶ $E : \mathcal{X} \rightarrow \mathcal{Z}$ is a **shared encoder** that extracts identity-agnostic features (expression, pose, lighting).
- ▶ $D_A, D_B : \mathcal{Z} \rightarrow \mathcal{X}$ are **person-specific decoders** that add facial identity.

Face swap: $x_A \xrightarrow{E} z \xrightarrow{D_B} x'_B$ (A's expression, B's face).

Training objective

Learn reconstruction mappings:

For person A:

$$\min_{E, D_A} \mathbb{E}_{x_A \sim p_A} [\mathcal{L}(x_A, D_A(E(x_A)))]$$

For person B:

$$\min_{E, D_B} \mathbb{E}_{x_B \sim p_B} [\mathcal{L}(x_B, D_B(E(x_B)))]$$

Multi-component loss:

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|_1 + \lambda_p \mathcal{L}_{\text{perceptual}}(x, \hat{x}) + \lambda_c \mathcal{L}_{\text{color}}(x, \hat{x})$$

where:

- ▶ $\mathcal{L}_{\text{perceptual}}(x, \hat{x}) = \|\phi(x) - \phi(\hat{x})\|_1$ with VGG16 features ϕ .
- ▶ $\mathcal{L}_{\text{color}}(x, \hat{x}) = \|\bar{x} - \bar{\hat{x}}\|_2^2$ (mean color preservation).

Architecture: U-Net with Attention

Shared Encoder $E : \mathbb{R}^{256 \times 256 \times 3} \rightarrow \mathbb{R}^{4 \times 4 \times 1024}$

$$z, \{s_1, s_2, s_3, s_4\} = E(x)$$

where z is the bottleneck and s_i are skip connections at different resolutions.

Person-Specific Decoders $D_A, D_B : \mathbb{R}^{4 \times 4 \times 1024} \rightarrow \mathbb{R}^{256 \times 256 \times 3}$

$$\hat{x} = D(z, \{s_i\})$$

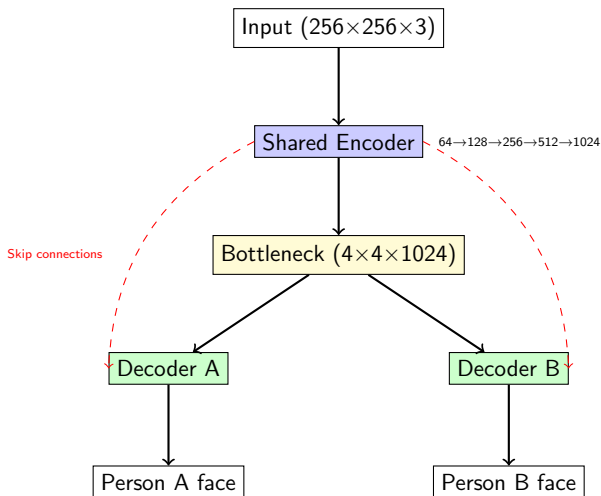
with attention-gated skip connections:

$$s'_i = s_i \alpha_i(g_i, s_i)$$

where $s_i \alpha_i(g_i, s_i)$ is a element-wise multiplication, g_i is the decoder's feature map at level i , and α_i is the learned attention weight map with values in $[0, 1]$.

Key: Skip connections preserve spatial details; attention selects relevant features.

UNet autoencoder



Training: $A \rightarrow \text{Encoder} \rightarrow \text{Decoder A} \rightarrow A$ (reconstruct).

Inference: $A \rightarrow \text{Encoder} \rightarrow \text{Decoder B} \rightarrow B'$ (swap).

Training and Inference

Training phase: Learn to reconstruct each person

$$\text{Sample } x_A \sim p_A : \quad x_A \rightarrow E(x_A) \rightarrow D_A(E(x_A)) \approx x_A$$

$$\text{Sample } x_B \sim p_B : \quad x_B \rightarrow E(x_B) \rightarrow D_B(E(x_B)) \approx x_B$$

Inference phase: Cross-identity reconstruction (face swap)

$$\text{A to B: } \quad x_A \rightarrow E(x_A) \rightarrow D_B(E(x_A)) = x_{A \rightarrow B}$$

$$\text{B to A: } \quad x_B \rightarrow E(x_B) \rightarrow D_A(E(x_B)) = x_{B \rightarrow A}$$

Why this works:

- ▶ E learns to extract z containing expression, pose, lighting (identity-free).
- ▶ D_A always renders with person A's facial features.
- ▶ D_B always renders with person B's facial features.

Theoretical Justification

Assumption: Face images admit a factorization:

$$x = g(z_{\text{id}}, z_{\text{attr}})$$

where z_{id} represents identity and z_{attr} represents attributes (expression, pose, lighting).

Learning dynamics:

- ▶ The shared encoder E is trained on both p_A and p_B .
- ▶ To minimize reconstruction loss for both identities, E must learn features common to both distributions.
- ▶ These common features are attributes, not identity (identity is person-specific).
- ▶ Each decoder D_A, D_B learns to map $z_{\text{attr}} \rightarrow x$ for their respective identity.

Result: E implicitly learns z_{attr} , enabling identity transfer at inference.

Face swapping pipeline

1. **Data preprocessing:** Face detection and cropping.
2. **Dataset:** Add random warping augmentation.
3. **Autoencoder:** U-Net (shared encoder + dual decoders).
4. **Loss function:** Multi-component loss for quality.
5. **Joint optimization:** The shared encoder is updated with gradients from both decoders simultaneously.

Files needed:

- ▶ personA/ and personB/: Input face images.
- ▶ random_warp.py: Data augmentation (provided separately).

Implementation Details

Data augmentation: Random warping $x \mapsto W(x)$

- ▶ Training input: warped face $W(x)$.
- ▶ Training target: original face x .
- ▶ Forces encoder to learn deformation-invariant features.

Normalization: GroupNorm divides C channels into G groups

$$\hat{x}_{n,c} = \frac{x_{n,c} - \mu_{n,g}}{\sqrt{\sigma_{n,g}^2 + \epsilon}} \cdot \gamma_c + \beta_c$$

Preserves color relationships between channels.

Optimization: Two optimizers updating shared encoder

- ▶ Optimizer 1: $\{E, D_A\}$ with samples from p_A .
- ▶ Optimizer 2: $\{E, D_B\}$ with samples from p_B .

Results

Qualitative evaluation:

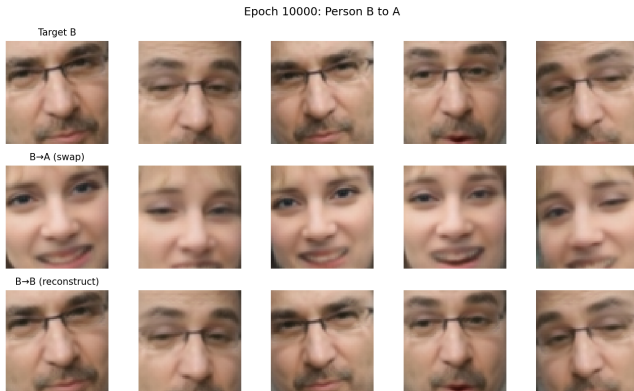


- ▶ Expression and pose preserved during swap.
- ▶ Facial identity successfully transferred.
- ▶ Color consistency maintained.

See [code3-deepfakes.py](https://github.com/code3-deepfakes).

Results

Qualitative evaluation:



- ▶ Expression and pose preserved during swap.
- ▶ Facial identity successfully transferred.
- ▶ Color consistency maintained.

See [code3-deepfakes.py](https://github.com/code3-deepfakes).