

MO433 - Unsupervised Learning

Unsupervised Feature Learning with CNNs

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

Why Unsupervised Feature Learning?

- ▶ Labels are expensive and time-consuming.
- ▶ Most real-world data is unlabeled.
- ▶ Manual annotation does not scale.

Why Unsupervised Feature Learning?

- ▶ Labels are expensive and time-consuming.
- ▶ Most real-world data is unlabeled.
- ▶ Manual annotation does not scale.

Solution: Learn meaningful representations from **data structure**.

Why Unsupervised Feature Learning?

- ▶ Labels are expensive and time-consuming.
- ▶ Most real-world data is unlabeled.
- ▶ Manual annotation does not scale.

Solution: Learn meaningful representations from **data structure**.

Statement of the problem

Transform high-dimensional input $x \in \mathbb{R}^n$ into compact, meaningful representations $z \in \mathbb{R}^d$ (where $d \ll n$) using an encoder function:

$$f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^d$$

Agenda

Generative approaches

Autoencoders

- ▶ Encoder: $x \rightarrow z$
- ▶ Decoder: $z \rightarrow \hat{x}$
- ▶ Loss: $\|x - \hat{x}\|^2$

Agenda

Generative approaches

Autoencoders

- ▶ Encoder: $x \rightarrow z$
- ▶ Decoder: $z \rightarrow \hat{x}$
- ▶ Loss: $\|x - \hat{x}\|^2$

Discriminative approaches

Self-supervised learning methods

- ▶ Create artificial tasks from data.
- ▶ Contrastive: SimCLR.
- ▶ Non-contrastive: BYOL, DINO.

Simple Autoencoder (dense layers only): Strategy

- ▶ **Encoder** $f_\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d$ (compression).
- ▶ **Latent space** $z \in \mathbb{R}^d$ where $d \ll n$.
- ▶ **Decoder** $g_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^n$ (reconstruction).

Minimize BYOL loss

$$\mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|^2$$

where $\hat{x}_i = g_\theta(f_\phi(x_i))$.

Simple Autoencoder (dense layers only): Strategy

- ▶ **Encoder** $f_\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d$ (compression).
- ▶ **Latent space** $z \in \mathbb{R}^d$ where $d \ll n$.
- ▶ **Decoder** $g_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^n$ (reconstruction).

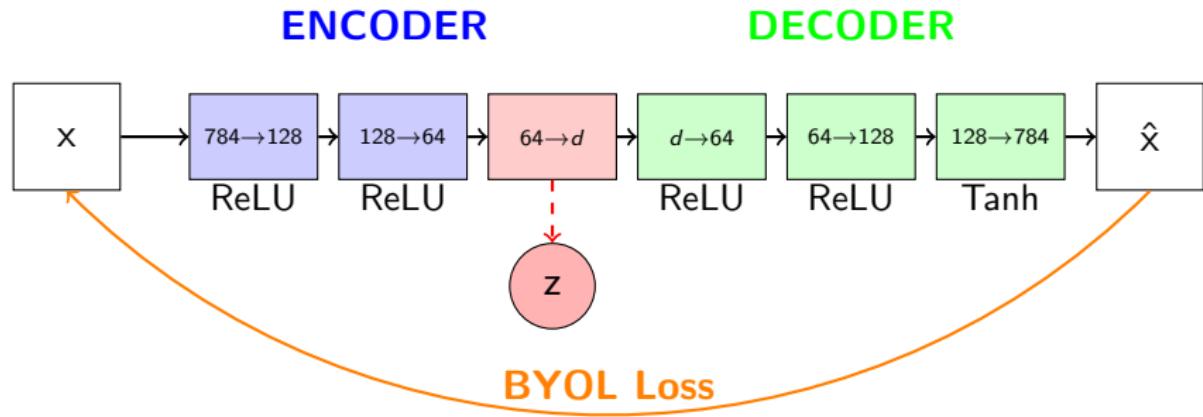
Minimize BYOL loss

$$\mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|^2$$

where $\hat{x}_i = g_\theta(f_\phi(x_i))$.

Encoder learns to store the most *essential features* in the latent z .

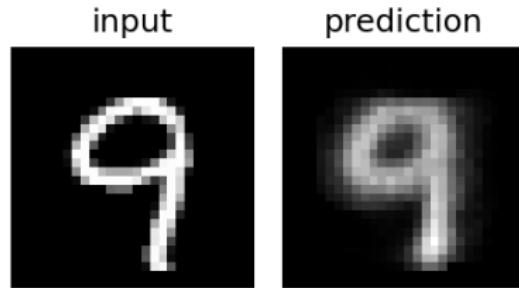
Simple Autoencoder: Architecture overview



Bottleneck design: Input dimension (784) \rightarrow Latent dimension (d) \rightarrow Output dimension (784). The latent space z contains compressed representations.

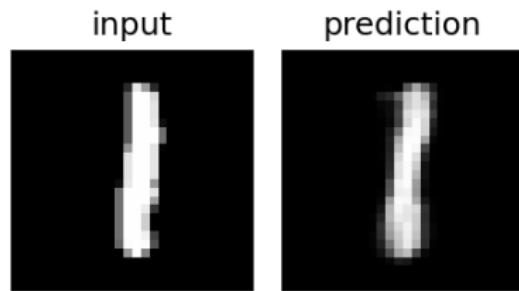
Simple Autoencoder: Reconstruction

Even with very low latent dimension (e.g., $d = 3$), the network can produce reasonable results (code1-simple_autoencoder.py).



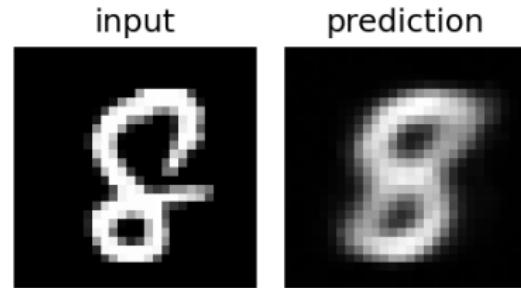
Simple Autoencoder: Reconstruction

Even with very low latent dimension (e.g., $d = 3$), the network can produce reasonable results (code1-simple_autoencoder.py).



Simple Autoencoder: Reconstruction

Even with very low latent dimension (e.g., $d = 3$), the network can produce reasonable results (code1-simple_autoencoder.py).



Simple Autoencoder: Problems

However, it presents the following problems:

- ▶ By flattening images to vectors, it destroys neighborhood relationships.
- ▶ It suffers from parameter explosion: $224 \times 224 \times 3 = 150,528$ inputs → millions of parameters in dense layers.
- ▶ Same object at different positions \neq similar encoding.

Simple Autoencoder: Problems

However, it presents the following problems:

- ▶ By flattening images to vectors, it destroys neighborhood relationships.
- ▶ It suffers from parameter explosion: $224 \times 224 \times 3 = 150,528$ inputs → millions of parameters in dense layers.
- ▶ Same object at different positions \neq similar encoding.

Convolutional autoencoders can address those problems.

Convolutional Autoencoder: Strategy

- ▶ Addresses limitations of simple autoencoders by using **convolutional layers** to preserve spatial relationships in images.
- ▶ **Encoder:** Convolutions + pooling progressively reduce spatial dimensions while extracting hierarchical features.
- ▶ **Decoder:** Transpose convolutions + upsampling restore spatial dimensions from compressed feature maps.
- ▶ Learns **translation-invariant** features through shared convolutional filters.
- ▶ Dramatically reduces parameters compared to dense layers for image data.

Convolutional Autoencoder: Strategy

- ▶ **Encoder** f_ϕ : Transforms input $x \in \mathbb{R}^{h \times w \times c}$ into a latent $z \in \mathbb{R}^{h' \times w' \times c'}, h' < h, w' < w, c' > c$.
- ▶ **Latent space** $z \in \mathbb{R}^{h' \times w' \times c'}$ preserves spatial structure.
- ▶ **Decoder** g_θ : Transforms $z \in \mathbb{R}^{h' \times w' \times c'}$ into $\hat{x} \in \mathbb{R}^{h \times w \times c}$.

Minimize pixel-wise reconstruction loss

$$\mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|_F^2$$

where $\hat{x}_i = g_\theta(f_\phi(x_i))$.

Convolutional Autoencoder: Strategy

- ▶ **Encoder** f_ϕ : Transforms input $x \in \mathbb{R}^{h \times w \times c}$ into a latent $z \in \mathbb{R}^{h' \times w' \times c'}, h' < h, w' < w, c' > c$.
- ▶ **Latent space** $z \in \mathbb{R}^{h' \times w' \times c'}$ preserves spatial structure.
- ▶ **Decoder** g_θ : Transforms $z \in \mathbb{R}^{h' \times w' \times c'}$ into $\hat{x} \in \mathbb{R}^{h \times w \times c}$.

Minimize pixel-wise reconstruction loss

$$\mathcal{L}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|_F^2$$

where $\hat{x}_i = g_\theta(f_\phi(x_i))$.

Encoder learns local patterns at multiple scales while preserving spatial relationships \Rightarrow meaningful latent representations.

Conv vs Simple Autoencoder: Key differences

Aspect	Simple Autoencoder	Conv Autoencoder
Input processing	Flatten to 784D vector	Keep $28 \times 28 \times 1$ structure
Feature extraction	Dense layers	Convolutional layers
Spatial awareness	Lost after flattening	Preserved throughout
Translation invariance	None	Built-in via convolutions
Parameter count	Millions for large images	Much fewer (shared filters)
Latent representation	1D vector	3D feature maps

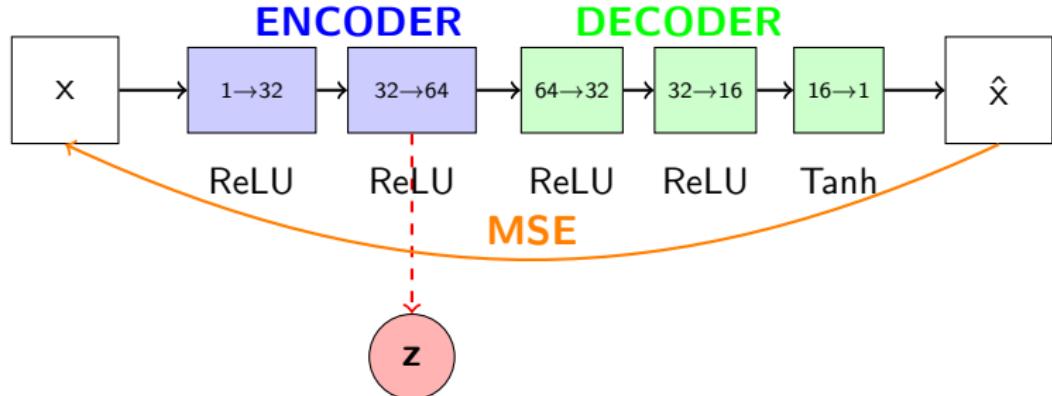
Conv vs Simple Autoencoder: Key differences

Aspect	Simple Autoencoder	Conv Autoencoder
Input processing	Flatten to 784D vector	Keep $28 \times 28 \times 1$ structure
Feature extraction	Dense layers	Convolutional layers
Spatial awareness	Lost after flattening	Preserved throughout
Translation invariance	None	Built-in via convolutions
Parameter count	Millions for large images	Much fewer (shared filters)
Latent representation	1D vector	3D feature maps

Convolutional advantages

- ▶ **Spatial structure:** 2D relationships preserved in feature maps.
- ▶ **Hierarchical learning:** Early layers detect edges, later layers detect objects.
- ▶ **Efficiency:** Shared convolution kernels dramatically reduce parameters.

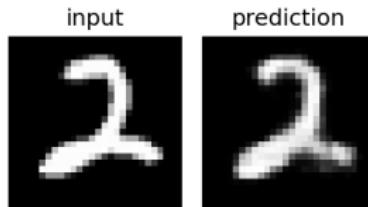
Convolutional Autoencoder: Architecture overview



Spatial preservation: Input x , output \hat{x} , and feature maps maintain 2D structure. Latent space z ($64 \times 2 \times 2$) contains spatial feature maps, not flattened vectors.

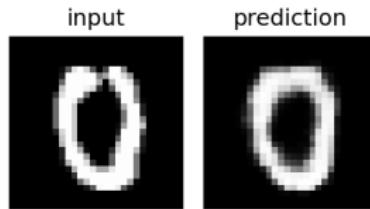
Conv Autoencoder (code2-conv_autoencoder.py)

For this simple problem, both autoencoders can create accurate predictions and improve data representation, depending on the latent dimension.



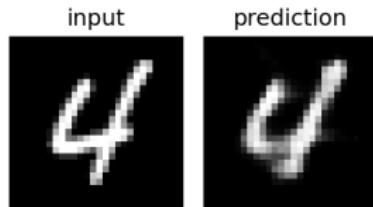
Conv Autoencoder (code2-conv_autoencoder.py)

For this simple problem, both autoencoders can create accurate predictions and improve data representation, depending on the latent dimension.



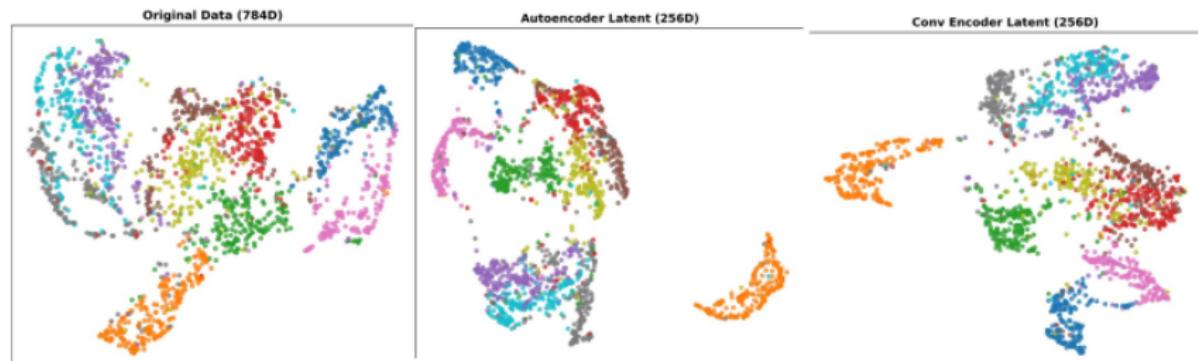
Conv Autoencoder (code2-conv_autoencoder.py)

For this simple problem, both autoencoders can create accurate predictions and improve data representation, depending on the latent dimension.



Conv Autoencoder (code2-conv_autoencoder.py)

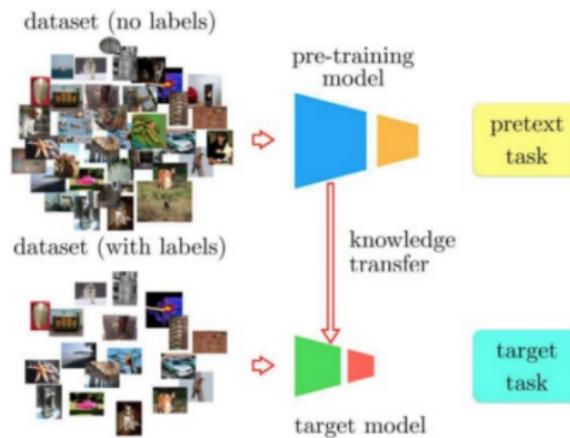
For this simple problem, both autoencoders can create accurate predictions and improve data representation, depending on the latent dimension.



While autoencoders learn useful representations through reconstruction, **can discriminative approaches do better?**

Self-supervised learning

- ▶ Creating a **pretext task** from unlabeled data enables learning meaningful data representations without manual annotations.
- ▶ The resulting encoder can then **transfer knowledge** to solve downstream **target tasks**.
- ▶ The network solving the target task requires **considerably less** labeled data for training.



Source: reference [1].

Self-supervised learning

- ▶ Creating a **pretext task** from unlabeled data enables learning meaningful data representations without manual annotations.
- ▶ The resulting encoder can then **transfer knowledge** to solve downstream **target tasks**.
- ▶ The network solving the target task requires **considerably less** labeled data for training.

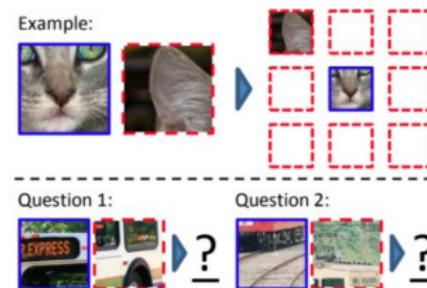


Figure 1. Our task for learning patch representations involves randomly sampling a patch (blue) and then one of eight possible neighbors (red). Can you guess the spatial configuration for the two pairs of patches? Note that the task is much easier once you have recognized the object!

Source: reference [2].

Self-supervised learning

The pretext tasks became more abstract (e.g., maximize agreement between augmented views) after 2020:

- ▶ SimCLR: Simple Framework for Contrastive Learning of Visual Representations [3].
- ▶ BYOL: Bootstrap Your Own Latent [4].
- ▶ DINO: Self-Distillation with No Labels [5].

Self-supervised learning

The pretext tasks became more abstract (e.g., maximize agreement between augmented views) after 2020:

- ▶ SimCLR: Simple Framework for Contrastive Learning of Visual Representations [3].
- ▶ BYOL: Bootstrap Your Own Latent [4].
- ▶ DINO: Self-Distillation with No Labels [5].

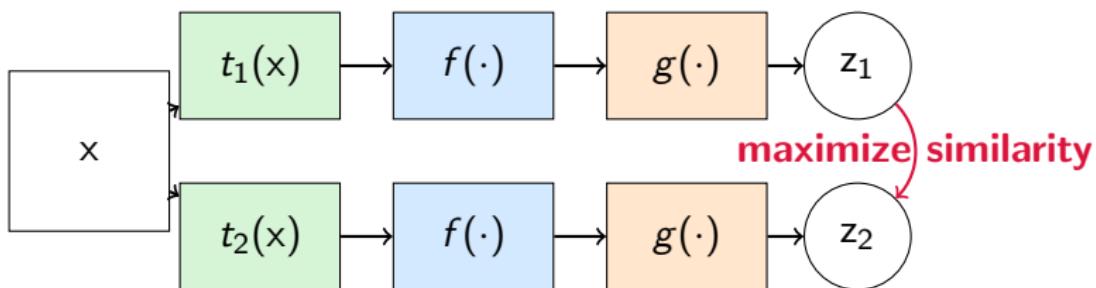
Focus is shifted to learning good general representations rather than solving auxiliary tasks.

SimCLR: Strategy

- ▶ The method trains an **encoder** followed by a **projector** by **contrastive learning**.
- ▶ It creates **augmented pairs** from unlabeled images and learns to distinguish between similar and dissimilar pairs by using a contrastive loss.
- ▶ Representations of augmented versions of a same image should be **similar**, while different images should have **dissimilar** representations.

SimCLR: Architecture overview

Augment. Encoder Projection



Shared weights: Both augmented views use the same encoder f and projection head g .

SimCLR: Data augmentation

Diverse views of the same image are created through random transformations.

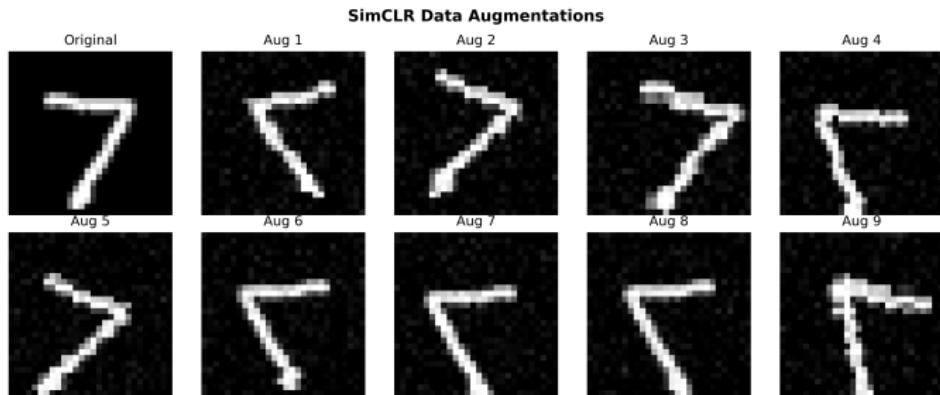
- ▶ RandomRotation(20°) - Rotational invariance.
- ▶ RandomAffine with translation ($\pm 10\%$) and scaling ($0.9-1.1 \times$).
- ▶ RandomHorizontalFlip - Spatial invariance.
- ▶ Gaussian noise addition - Robustness to noise.

Critical insight.

Strong augmentation is essential for SimCLR success. Weak augmentations lead to trivial solutions where the model learns shortcuts rather than meaningful representations.

SimCLR: Data augmentation

Example of data augmentations.



SimCLR: Encoder architecture

CNN-based feature extractor:

$$\mathbf{h} = f_{\theta}(\mathbf{x}) \in \mathbb{R}^d$$

Implementation details for MNIST:

- ▶ **Convolutional layers:** Extract spatial features.
 - ▶ Conv2d($1 \rightarrow 32$) + BatchNorm + ReLU + MaxPool.
 - ▶ Conv2d($32 \rightarrow 64$) + BatchNorm + ReLU + MaxPool.
- ▶ **Feature dimension:** $64 \times 2 \times 2 = 256$ features.
- ▶ **Parameter sharing:** Same encoder processes both augmented views.

Key principle: The encoder learns to extract features that are **invariant** to the augmentations but **discriminative** between different images.

SimCLR: Projection head

Nonlinear projection for contrastive learning

$$z = g_{\phi}(h) = W_2 \sigma(W_1 h) \in \mathbb{R}^{128}$$

- ▶ **Two-layer MLP:** $256 \rightarrow 256 \rightarrow 128$.
- ▶ **ReLU activation:** Nonlinear transformation.
- ▶ **Lower dimensionality:** Projects to contrastive learning space.

Why a projection head?

- ▶ Contrastive learning works better in **lower-dimensional** spaces.
- ▶ Separates representation learning from contrastive optimization.
- ▶ **Key insight:** Use h for downstream tasks, not z .

SimCLR: InfoNCE loss (NT-Xent)

Contrastive learning objective For a batch of N images, create $2N$ augmented views. The loss for positive pair (i, j) is:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} 1_{k \neq i} \exp(\text{sim}(z_i, z_k)/\tau)}$$

where:

- ▶ $\text{sim}(z_i, z_j) = z_i^T z_j / (\|z_i\| \|z_j\|)$ (cosine similarity).
- ▶ τ = temperature parameter (0.5 in implementation).
- ▶ $1_{k \neq i}$ excludes self-similarity.

Intuition:

- ▶ **Numerator:** Similarity to positive pair (augmented version).
- ▶ **Denominator:** Similarities to all negatives (other images).

Why SimCLR works?

Contrastive objective interpretation The InfoNCE loss can be viewed as:

$$\mathcal{L} = -\mathbb{E} \left[\log \frac{\exp(z_i^T z_j / \tau)}{\exp(z_i^T z_j / \tau) + \sum_{k \neq i, j} \exp(z_i^T z_k / \tau)} \right]$$

This encourages:

- ▶ **Alignment:** Positive pairs have high cosine similarity

$$z_i^T z_j \rightarrow \|z_i\| \|z_j\| \cos(0) = 1$$

- ▶ **Uniformity:** Features spread uniformly on unit hypersphere

$$\mathbb{E}_{i \neq j} [z_i^T z_j] \rightarrow 0$$

Result: Learned representations capture semantic similarity while preserving discriminative power (code3-simclr_unsupfeatlearn.py).

BYOL: Strategy

- ▶ The method trains **online** and **target** networks.
 - ▶ **Online:** Encoder + Projector + Predictor.
 - ▶ **Target:** Encoder + Projector.
- ▶ Two **augmented views** of the same image are passed through **both networks** (4 forward passes total).
- ▶ The online **predicts** target network's representations.
- ▶ While the online's parameters are updated using **BYOL loss**, the target uses **momentum** updates to prevent collapse by creating a **moving target** that evolves slowly.

$$\theta_{\text{target}} \leftarrow \tau \theta_{\text{target}} + (1 - \tau) \theta_{\text{online}}$$

BYOL vs SimCLR: Fundamental differences

Aspect	SimCLR	BYOL
Negative samples	Required (large batches)	Not needed
Loss function	InfoNCE (contrastive)	Cosine-based
Architecture	Symmetric	Asymmetric
Batch dependency	Strong (more negatives = better)	Weak
Key mechanism	Contrast positive/negative	Momentum target updates

BYOL may use the same data augmentation operations as SimCLR.

BYOL vs SimCLR: Fundamental differences

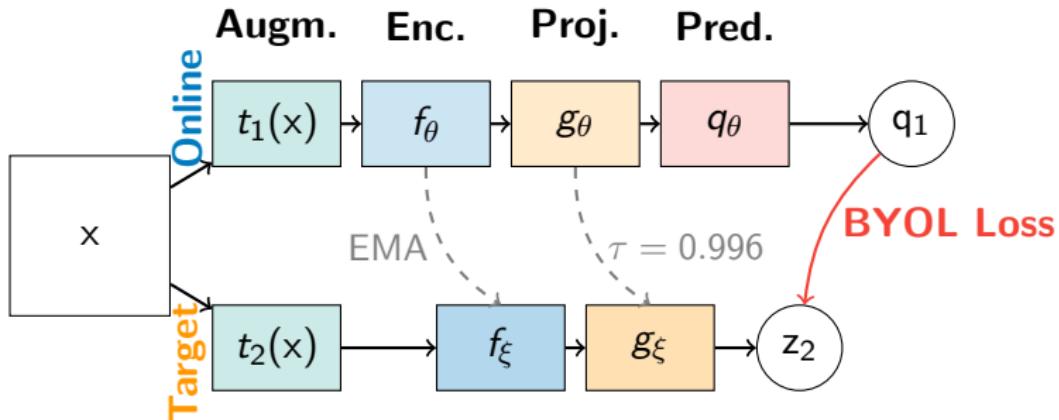
Aspect	SimCLR	BYOL
Negative samples	Required (large batches)	Not needed
Loss function	InfoNCE (contrastive)	Cosine-based
Architecture	Symmetric	Asymmetric
Batch dependency	Strong (more negatives = better)	Weak
Key mechanism	Contrast positive/negative	Momentum target updates

BYOL may use the same data augmentation operations as SimCLR.

BYOL's advantages.

- ▶ **Simpler training:** No need to manage negative samples.
- ▶ **Batch size flexibility:** Performance less sensitive to batch size.
- ▶ **Stable training:** Momentum updates provide stability.

BYOL: Architecture overview



Key Asymmetry: Only online network has predictor q_θ . Target network ξ updated via exponential moving average (EMA). Disable target gradients to prevent backpropagation.

BYOL: Asymmetric network design

Online network (trainable):

$$q_1 = q_\theta(g_\theta(f_\theta(x_1)))$$

- ▶ **Encoder:** f_θ - extracts features.
- ▶ **Projector:** g_θ - maps to representation space.
- ▶ **Predictor:** q_θ - predicts target representations.

Target network (momentum updated):

$$z_2 = g_\xi(f_\xi(x_2))$$

- ▶ **Encoder:** f_ξ - same architecture as online.
- ▶ **Projector:** g_ξ - same architecture as online.
- ▶ **No predictor** - key architectural difference.

BYOL: Implementation

For MNIST: **Encoders:**

- ▶ Conv2d($1 \rightarrow 32$) + BatchNorm + ReLU + MaxPool.
- ▶ Conv2d($32 \rightarrow 64$) + BatchNorm + ReLU + MaxPool.
- ▶ Feature dimension: 256 ($64 \times 2 \times 2$).

Projectors: MLP($256 \rightarrow 256 \rightarrow 64$) + BatchNorm + ReLU.

Predictor: MLP($64 \rightarrow 32 \rightarrow 64$) + BatchNorm + ReLU.

BYOL: Momentum update mechanism

Exponential Moving Average (EMA) Target network parameters are updated as:

$$\xi \leftarrow \tau \xi + (1 - \tau) \theta$$

where $\tau \in [0, 1]$ is the momentum coefficient.

Implementation details:

- ▶ **Initialization:** $\xi_0 = \theta_0$ (target = online at start).
- ▶ **Momentum schedule:** τ increases from 0.99 to 0.999 during training.
- ▶ **Update frequency:** Every training step (after gradient update of the online network).

BYOL: Loss function

Symmetric cosine-based loss: For augmented pair (x_1, x_2) ,

$$\mathcal{L}_{BYOL} = 2 - \frac{\mathbf{q}_1 \cdot \mathbf{z}_2}{\|\mathbf{q}_1\| \|\mathbf{z}_2\|} - \frac{\mathbf{q}_2 \cdot \mathbf{z}_1}{\|\mathbf{q}_2\| \|\mathbf{z}_1\|}$$

where:

- ▶ $\mathbf{q}_1 = q_\theta(g_\theta(f_\theta(x_1)))$ - online prediction for view 1.
- ▶ $\mathbf{q}_2 = q_\theta(g_\theta(f_\theta(x_2)))$ - online prediction for view 2.
- ▶ $\mathbf{z}_1 = g_\xi(f_\xi(x_1))$ - target projection for view 1.
- ▶ $\mathbf{z}_2 = g_\xi(f_\xi(x_2))$ - target projection for view 2.

Why BYOL works?

Without negatives, why does not the model learn trivial constant representations (i.e., does not collapse)?

1. **Asymmetric architecture:** Target has no predictor.
 - ▶ Online network learns to predict target representations.
 - ▶ Creates learning pressure without direct target optimization.
2. **Momentum updates:** $\xi \leftarrow \tau\xi + (1 - \tau)\theta$
 - ▶ Target evolves slowly, preventing immediate trivial alignment.
 - ▶ Provides stable learning targets that gradually improve.
3. **Data augmentation:** Forces invariance learning.
 - ▶ Different augmentations create non-trivial prediction task.
 - ▶ Must learn meaningful features to predict across transforms.

see code4-byol_unsupfeatlearn.py

DINO: Strategy

- ▶ The method trains **student** and **teacher** networks.
 - ▶ **Student:** Encoder + Projector + **Softmax**.
 - ▶ **Teacher:** Encoder + Projector + **Softmax** + **Centering**.
- ▶ The networks receive **augmented pairs** of the same image (or multi-crop: local crops for student, global crops for teacher).
- ▶ The student learns to predict the teacher's **softmax outputs** via *self-distillation*.
- ▶ The teacher provides **stable targets** via centering mechanism and temperature scaling.
- ▶ While the student's parameters are updated using **cross-entropy loss**, the teacher uses **EMA updates** from student parameters.

DINO vs SimCLR vs BYOL: Key differences

Aspect	SimCLR	BYOL	DINO
Negative samples	Required	Not needed	Not needed
Loss function	InfoNCE	Cosine-based	Cross-entropy
Architecture	Symmetric	Asymmetric	Asymmetric
Networks	Single	Online + Target	Student + Teacher
Key mechanism	Contrastive	Momentum + Predictor	Centering + EMA
Collapse prevention	Negatives	Asymmetric predictor	Centering
Multi-crop support	No	No	Yes (core innovation)

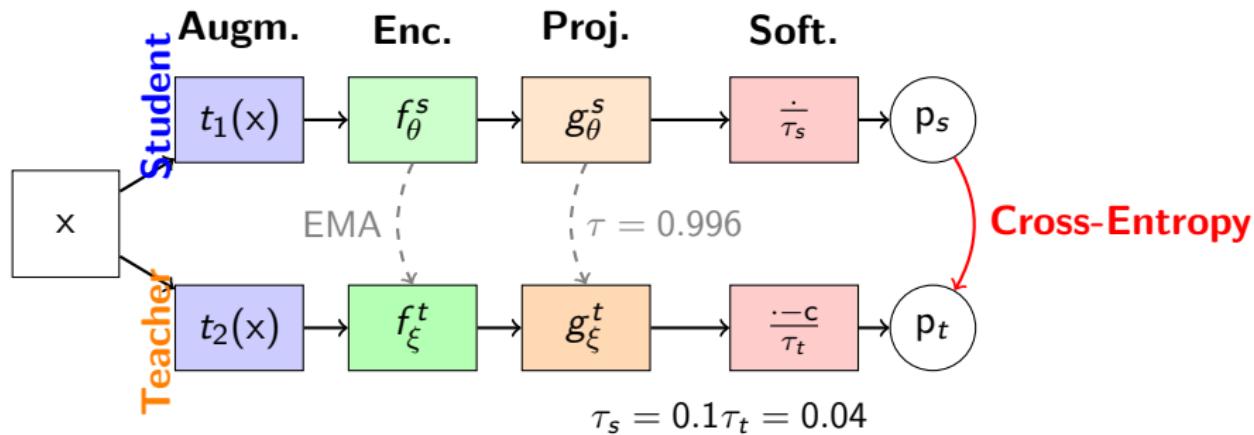
DINO vs SimCLR vs BYOL: Key differences

Aspect	SimCLR	BYOL	DINO
Negative samples	Required	Not needed	Not needed
Loss function	InfoNCE	Cosine-based	Cross-entropy
Architecture	Symmetric	Asymmetric	Asymmetric
Networks	Single	Online + Target	Student + Teacher
Key mechanism	Contrastive	Momentum + Predictor	Centering + EMA
Collapse prevention	Negatives	Asymmetric predictor	Centering
Multi-crop support	No	No	Yes (core innovation)

DINO's advantages

- ▶ **Stable training:** Centering prevents collapse without negatives.
- ▶ **Multi-crop learning:** Student learns global context from local patches.
- ▶ **Better representations:** Self-distillation creates rich features.

DINO: Architecture overview



Encoder and Projector with the same architecture (required for EMA updates).

DINO: Symmetric network design

Student network (trainable):

$$p_s = \text{softmax} \left(\frac{g_\theta^s(f_\theta^s(x_1))}{\tau_s} \right)$$

- ▶ **Encoder:** f_θ^s - extracts features.
- ▶ **Projector:** g_θ^s - maps to representation space.
- ▶ **Temperature:** $\tau_s = 0.1$ - controls sharpness.

Teacher network (EMA updated):

$$p_t = \text{softmax} \left(\frac{g_\xi^t(f_\xi^t(x_2)) - c}{\tau_t} \right)$$

- ▶ **Encoder:** f_ξ^t - same architecture as student.
- ▶ **Projector:** g_ξ^t - same architecture as student.
- ▶ **Centering:** c - prevents collapse.
- ▶ **Temperature:** $\tau_t = 0.04$ - sharper than student.

DINO: Implementation for MNIST

Student encoder:

- ▶ Conv2d($1 \rightarrow 32$) + BatchNorm + ReLU + MaxPool
- ▶ Conv2d($32 \rightarrow 64$) + BatchNorm + ReLU + MaxPool
- ▶ Feature dimension: 256

Student projection head:

- ▶ MLP($256 \rightarrow 512 \rightarrow 512 \rightarrow 256$) + BatchNorm + GELU
- ▶ Output dimension: 256 (for softmax)

Teacher network:

- ▶ Identical architecture to student (encoder + projector).
- ▶ No gradients - updated only via EMA.

DINO: EMA update mechanism

Exponential Moving Average for entire network: Teacher parameters updated as:

$$\xi \leftarrow \tau \xi + (1 - \tau) \theta$$

where $\tau \in [0.996, 0.999]$ is the momentum coefficient.

Both encoder and projector updated:

$$f_\xi^t \leftarrow \tau f_\xi^t + (1 - \tau) f_\theta^s$$

$$g_\xi^t \leftarrow \tau g_\xi^t + (1 - \tau) g_\theta^s$$

Implementation details:

- ▶ **Initialization:** $\xi_0 = \theta_0$ (teacher = student at start).
- ▶ **Momentum schedule:** τ increases from 0.996 to 0.999 during training.
- ▶ **Update frequency:** Every training step (after student gradient update).

DINO: Loss function

Cross-entropy loss with centering: For augmented pair (x_1, x_2) ,

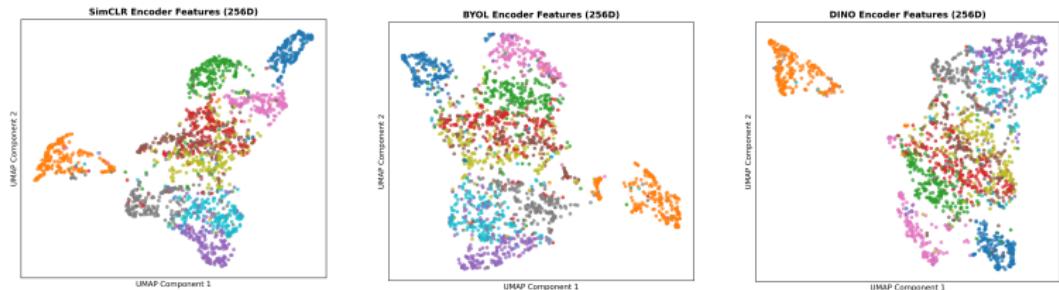
$$\mathcal{L}_{DINO} = \frac{1}{2} \left[H(P_t^{(2)}, P_s^{(1)}) + H(P_t^{(1)}, P_s^{(2)}) \right]$$

where $H(P, Q) = -\sum_k P_k \log Q_k$ is the cross-entropy between distributions P and Q .

- ▶ $P_s^{(1)} = \text{softmax} \left(\frac{g_\theta^s(f_\theta^s(x_1))}{\tau_s} \right)$ - student distribution for view 1.
- ▶ $P_t^{(2)} = \text{softmax} \left(\frac{g_\xi^t(f_\xi^t(x_2)) - c}{\tau_t} \right)$ - teacher distribution for view 2.
- ▶ **Centering:** $c \leftarrow 0.9c + 0.1 \frac{1}{B} \sum_{b=1}^B g_\xi^t(f_\theta^t(x_b))$, where x_b is a batch image and c starts at 0.

Intuition: Student distribution should match teacher distribution for different views (`code5-dino_unsupfeatlearn.py`).

Comparison among representations using projections



Comparison among representations using projections may be misleading; it is better to evaluate them on a target task.

DINO: Multi-crop strategy (Advanced)

True DINO innovation: Asymmetric multi-crop processing

Student receives local crops:

- ▶ 6-8 small crops (96×96 patches)
- ▶ Learns from **partial information**
- ▶ Forced to understand global context from local details

Teacher receives global crops:

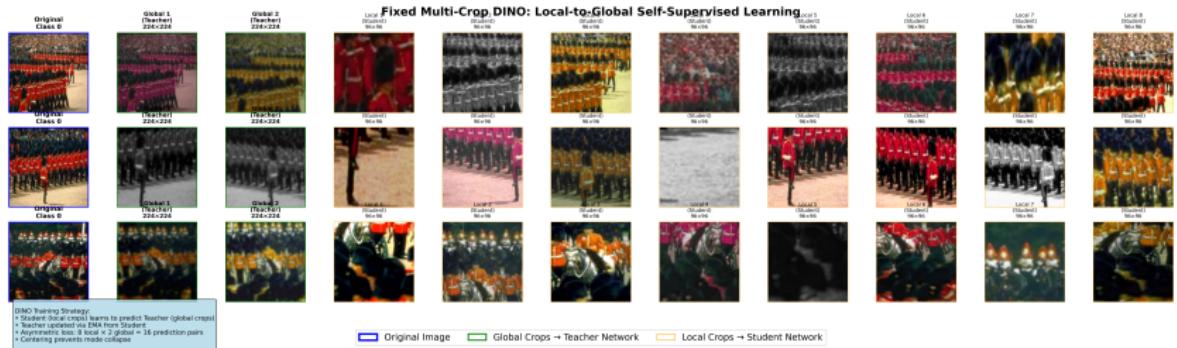
- ▶ 2 large crops (224×224 views)
- ▶ Provides **global semantic context**
- ▶ Full image understanding

Loss: Student (local) \rightarrow Teacher (global)

$$\mathcal{L} = \sum_{\text{local}} \sum_{\text{global}} \text{CrossEntropy}(\text{student}_{\text{local}}, \text{teacher}_{\text{global}})$$

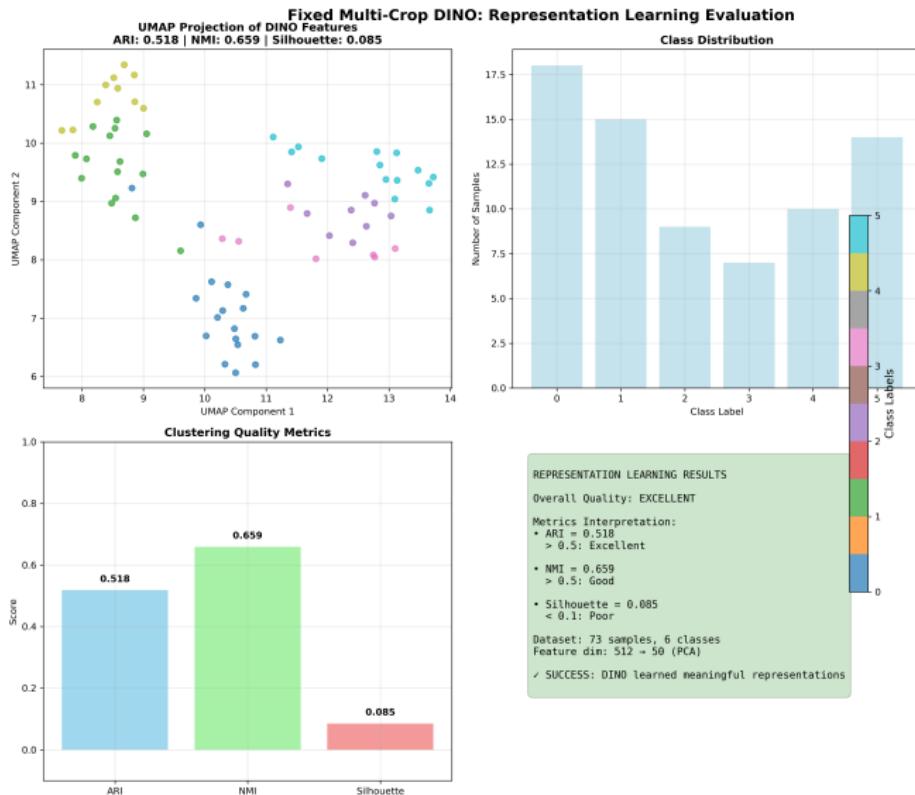
This local-to-global learning is DINO's key advantage for natural images!

DINO: Multicrop examples



See code6-dino_multicrop_unsupfeatlearn.py.

DINO: Final data representation



Why DINO works?

Without negatives, why does the model not collapse to trivial representations?

1. Centering mechanism:

- ▶ Prevents teacher from outputting uniform distributions.
- ▶ Forces teacher to maintain meaningful output diversity.

2. Temperature scaling: Different τ_s and τ_t

- ▶ Teacher outputs ($\tau_t = 0.04$) are sharper than student ($\tau_s = 0.1$).
- ▶ Creates informative, peaked target distributions.

3. EMA updates: Teacher evolves slowly

- ▶ Provides stable targets that gradually improve.
- ▶ Student cannot immediately "game" the teacher.

Centering + temperature scaling + EMA = stable self-distillation!

DINO: Why better than BYOL?

Theoretical advantages:

- ▶ **Probabilistic targets:** Cross-entropy loss more principled than MSE.
- ▶ **No predictor asymmetry:** Softmax is cleaner.
- ▶ **Multi-crop capability:** Local-to-global learning impossible in BYOL.
- ▶ **Better collapse prevention:** Centering more reliable than predictor asymmetry.

Empirical benefits:

- ▶ **Better downstream performance:** Especially on dense prediction tasks.
- ▶ **More stable training:** Less sensitive to hyperparameters.
- ▶ **Richer representations:** Self-distillation creates more informative features.

DINO represents the evolution from contrastive (SimCLR) → asymmetric (BYOL) → self-distillation approaches.

References

1. Noroozi et al. Boosting Self-Supervised Learning via Knowledge Transfer, https://openaccess.thecvf.com/content_cvpr_2018/papers/Noroozi_Boosting_Self-Supervised_Learning_CVPR_2018_paper.pdf.
2. Doersch et al. Unsupervised Visual Representation Learning by Context Prediction, <https://arxiv.org/pdf/1505.05192>.
3. Chen et al. A simple framework for contrastive learning of visual representations. ICML 2020.
4. Grill et al. Bootstrap your own latent: A new approach to self-supervised learning. NeurIPS 2020.
5. Caron et al. Emerging properties in self-supervised vision transformers. ICCV 2021.
6. He et al. Masked autoencoders are scalable vision learners. CVPR 2022.