A Survey of Ten Machine Learning Methods and Their Unification Via the I-Con Framework

Andrew Taylor, Graduate student at Johns Hopkins Engineering for Professionals,

Department of Artificial Intelligence, August 2025

***Abstract-* The field of machine learning (ML) has grown by leaps and bounds, with increasing speed, over the last twenty years. As more powerful computers and computing techniques come online there will be even more we can do with information, data, and science in general as time goes by. This survey paper aims to fill a hole in canonical instruction in the field of artificial intelligence. Many recent methods are worthy, but not yet taught, the field is just moving too fast. In this paper, one ML model or method (most from within the past 5 years) will be detailed for each of the ten broad families of machine learning approaches. For every method I supply (i) a concise verbal description, (ii) a self-contained mathematical definition, (iii) motivation to select the model, (iv) illustrative application domains, and (v) a small-scale demonstration on the ImageNet-100 benchmark. At its conclusion, there will be a brief reconstruction of a discovery made by the Alshammari, Hershey, Feldmann, Freeman and Hamilton team, a unifying framework for representation learning called Information Contrastive Learning (I-Con).**

I. Introduction

Over the past two decades the machine-learning (ML) community has witnessed an explosive growth in the number and diversity of representation learning techniques. While this progress has produced striking practical successes, it has also fragmented the pedagogical landscape: many influential methods introduced in just the last five years have yet to appear in standard curricula. The field is just moving too fast. This survey is intended as a robust, experiment-driven guide to ten methods, one chosen from each of ten broad methodological families. For every method I supply (i) a concise verbal description, (ii) a self-contained mathematical definition, (iii) motivation to select the model, (iv) illustrative application domains, and (v) a small-scale demonstration on the ImageNet-100 dataset. By anchoring all demonstrations to the same dataset and evaluation protocol I enable readers to form intuitive, apples-to-apples comparisons of the practical behavior of very different learning paradigms. The paper’s goal is expository.

Additionally, although these methods come from seemingly disparate research threads (ranging from probabilistic modelling to metric-learning, graph theory and more) they can all be interpreted through a single information-theoretic loss function. Specifically, Alshammari et al.’s Information Contrastive Learning (I-Con) framework demonstrates that many popular losses are instances of an “integrated” Kullback–Leibler divergence that compares conditional distributions over neighborhoods in representation space.1 This is an important finding. So, I close the survey by reiterating this theory and showing how it places these ten exemplars into a common mathematical template, clarifying their similarities, and suggesting principled ways to develop new methods with new loss functions.

The ten methods considered here are:

| **Family** | **Exemplar method (year)** | **Key idea** |
| --- | --- | --- |
| Dimensionality Reduction | t-SNE (2008) | Student-t–based neighborhood preservation |
| Clustering | Deep Cluster Discovery (DCD, 2024) | Joint optimization of encoder and centroids |
| Supervised Learning | Harmonic-Loss classifier (2021) | Geometry-aware smooth surrogate loss |
| Contrastive Learning | SimCLR (2020) | Augmentation-invariant InfoNCE objective |
| Probabilistic Models | Gaussian Mixture Models (GMMs, classic) | Latent variable density estimation |
| Graph-Based Learning | Graph Embeddings (Node2Vec, 2016) | Biased random-walk context prediction |
| Self-Supervised Learning | VICReg (2022) | Variance-invariance-covariance regularization |
| Multimodal Learning | CLIP-style cross-modal embeddings (2021) | Paired image–text contrastive training |
| Neighbourhood-based RL | Student-t Neighborhood Embedding, t-SimCLR (2023) | Heavy-tailed affinity modelling |
| Unified / Hybrid | InfoNCE Clustering (2025) | Contrastive objectives with pseudo-labels |

As for the experimental protocol, most demonstrations use a vanilla ResNet-18 backbone (but otherwise ResNet50 or DiNO ViT-16) on the 100-class *ImageNet-100* subset at 160px resolution on the short side. Default ImageNet-1K pre-training weights are employed at any stage. Complete training scripts with their hyper-parameters are released alongside this manuscript (in a Juypter noteboook) to facilitate exact reproduction. This will situate the paper nicely as a showcase for 10 methods.

There are a few ways in which this survey is valuable. Each section is written to be *self-contained,* allowing readers to dip into topics of interest without prerequisite reading. *Practice-focused criteria* are provided in the form of bullets to guide the reader in understanding when to select the method being discussed. That should be helpful. Then, taking all the code in the appendix into a notebook allows for *side-by-side empirical snapshots*: running every method on the same dataset under the same hardware budget provides an intuitive sense of the cost–performance trade-offs that differentiates families. Finally, is an i*nformation-theoretic synthesis*—by recasting all ten objectives as cases of the I-Con loss, I reiterate the commonality of the I-Con framework that unifies them.

The flow of the paper is organized as follows. Part II, Sections 1–9 detail each of the first 9 methods in turn, adhering to the common five-part template described above. Part III presents the I-Con framework, showing the unified loss function they invented, from which the specialized forms of the integrated KL divergence that recover each loss function presented in Section II can be derived. After this explanation, the final “Unifying” method, invented with their new loss function, is then covered. Part IV is a comparative analysis over all methods and Part V concludes with a summary of what was learned. Let’s begin with the first method.

II. Methods, Related Work, and Demonstrations

1. Dimensionality Reduction: t-Distributed Stochastic Neighbor Embedding (t-SNE)

Dimensionality reduction techniques aim to represent high-dimensional data in a lower-dimensional space, preserving essential relationships and structure inherent in the data. Among these methods, t-Distributed Stochastic Neighbor Embedding (t-SNE) has emerged as a particularly effective visualization tool due to its capacity to highlight local structure within data.2

*Mathematical Formulation*

The t-SNE algorithm operates through the following mathematical framework. Given a dataset , the method constructs pairwise conditional probabilities reflecting the similarity of data points in the original high-dimensional space:

where is determined through binary search to achieve a predefined perplexity. Joint symmetrical probabilities are subsequently defined as:

The lower-dimensional embedding points , typically with or , are initialized randomly and optimized to minimize the Kullback-Leibler (KL) divergence between the joint distributions in the original space and corresponding joint distributions in the embedded space:

The KL divergence, used as the loss function, is then given by:

This optimization explicitly focuses on preserving local neighborhood relationships rather than global distances, making t-SNE particularly able at visualizing clusters and local structure within data.2

The foundational work on t-SNE was introduced by Van der Maaten and Hinton (2008), who demonstrated its efficacy in visualizing complex, high-dimensional datasets by maintaining meaningful neighborhood information. Their research highlighted how t-SNE outperforms classical methods like PCA (Principal Component Analysis) and MDS (Multidimensional Scaling) in preserving local data structure, significantly improving interpretability in exploratory data analysis.

*When to Prefer t‑SNE over Classical Dimensionality‑Reduction Methods*

* **When preserving local structure is crucial**: *t*-SNE emphasizes local neighborhood relationships, making it ideal for tasks where small-scale similarity (like semantic or phenotypic clusters) matters more than global geometry.
* **When data is non-linear and high-dimensional**: Unlike PCA or Independent Component Analysis (ICA), which assume linear structure, *t*-SNE captures complex manifolds and curved geometries often found in image, text, audio, or genomics data, etcetera.
* **When interpretability of clusters is the goal**: *t*-SNE often reveals visually distinct clusters that correspond to meaningful categories, aiding exploration, diagnosis and unsupervised discovery. This is very useful to subject-matter experts.
* **When visualizing deep neural network representations**: *t*-SNE is well-suited to project hidden layer activations to 2D or 3D to assess class separation, failure modes, feature learning.
* **When global variance or component ranking is not needed**: Classical methods like PCA provide ranked axes of explained variance (useful for compression or denoising) but *t*-SNE is better when the focus is on structure, not variance.
* **When classical methods collapse important distinctions**: If PCA or MDS merges distinct groups due to global averaging, *t*-SNE can often preserve and reveal those finer separations.
* **When embedding sparsely populated or heterogeneous data**: In cases like single-cell transcriptomics or anomaly detection, *t*-SNE better reveals rare subtypes or outliers than methods preserving only global distances.

*Example Applications of t-SNE*

*t*-SNE is a powerful tool for exploring complex manifolds in high-dimensional data, particularly in natural language, vision, audio domains, things like that. It excels at preserving local neighborhood structures in word and sentence embeddings, or feature spaces, making semantic clusters visually distinguishable (something linear methods like PCA or even nonlinear ones like MDS and LLE often fail to do).14 In applications such as customer behavior or recommender systems, where interaction data is sparse and high-cardinality, *t*-SNE helps uncover nonlinear patterns that reveal distinct segments far better than traditional factor models. Indeed, studies have shown that latent vectors of users or items often form well-separated clusters in a t-SNE plot, revealing coherent segments that can be used for targeting or recommendations.15

In biology, t-SNE has become integral to single-cell transcriptomics, enabling the discovery of subtle cellular subtypes by exposing fine-grained transcriptional heterogeneity that might be missed by linear dimensionality reduction. Unlike PCA or ICA (which may collapse nuanced variation into a few components) – t-SNE is able to reveal branching or overlapping structures in gene expression data that are critical for understanding cell differentiation pathways and rare cell populations. For instance, careful application of t-SNE on large single-cell RNA sequencing datasets can neatly separate developmental trajectories of cells and arrange clusters in meaningful orders, whereas default methods showed intermingled or fragmented clusters.14 Also, in cybersecurity and anomaly detection, t-SNE helps analysts visually flag rare events by highlighting local density variations in network telemetry or user behavior data that would be smoothed out by global distance-preserving methods. By focusing on local similarity, t-SNE can make outliers or micro-clusters of abnormal activity stand out in a 2D projection (e.g. isolating an anomalous user session or network flow as a distinct island), aiding in the identification of intrusion attempts or fraudulent behaviors that might otherwise go unnoticed. Across these domains, t-SNE offers a window into the structure of complex data spaces, making hidden patterns tangible and actionable.14

Beyond data exploration, *t*-SNE plays a key role in interpreting deep learning models. Projecting penultimate-layer (hidden) activations into a lower-dimensional space enables developers to assess class separation and identify overlapping regions that point to failure modes, mislabeled data, even representational weaknesses. This diagnostic clarity often surpasses what linear embeddings can provide, offering actionable insight into model performance and architecture. Across all these domains, *t*-SNE offers a glimpse into the structure of complex data spaces making hidden patterns revealed.

*Demonstration of t-SNE on ImageNet-100*

To get good clusters, the pipeline needed significant pre-processing: pre-trained ResNet18 feature extraction, specific ImageNet preprocessing and resizing, aggressive PCA prior to t-SNE, and a class selection strategy with semantic diversity, as well as data augmentation and multiple views. Loading 1250 samples for each 6 classes the following clustering occurred:

A graph showing different colored dots

AI-generated content may be incorrect.  
*Figure 1: Comparing t-SNE to PCA for 6 classes of ImageNet-100*

Why does t-SNE beats “plain” PCA for visual structure finding? t-SNE fundamentally outperforms because of the fundamental difference in their mathematical approaches and optimization objectives. As was said, PCA operates as a linear transformation that seeks to find directions of maximum variance in the data. It essentially performs a global optimization that projects high-dimensional data onto lower-dimensional spaces by preserving the directions along which the data varies most. This classical approach, while computationally efficient and mathematically elegant, has severe limitations when dealing with complex visual data structures. When PCA analyzes the 2048-dimensional ResNet-18 features extracted from ImageNet images, it can only capture 17.5% of the total variance when projecting to two dimensions, as is evident in the plot. More critically, PCA's focus on variance preservation means that visually distinct object categories may end up overlapping in the projected space if they happen to have similar statistical variance patterns, even though they represent completely different concepts.

In stark contrast, t-SNE employs a non-linear embedding approach that optimizes for an entirely different objective function. Rather than maximizing variance, t-SNE minimizes the Kullback-Leibler divergence between probability distributions that represent pairwise similarities in the high-dimensional space and the low-dimensional embedding space. This means t-SNE specifically focuses on preserving local neighborhood relationships, ensuring that points that are similar in the original (here, high-dimensional ResNet) feature space that remain close together in the 2D visualization, while points that are dissimilar get pushed apart. The mathematical technique involves computing conditional probabilities P(j|i) in the high-dimensional space using Gaussian kernels, then computing joint probabilities Q(i,j) in the low-dimensional space using Student-t distributions, and iteratively optimizing the embedding to make these probability distributions as similar as possible. It first inflates global separations during a 250-iteration early-exaggeration phase, then fine-tunes local neighborhoods over the remaining 500 iterations, driving KL from 1.0626 down to 0.3993 at the best-performing perplexity of 30. What is this exaggeration? t-SNE consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor. Larger factors result in larger gaps between natural clusters in the data, but too large a factor will increase KL divergence.

The fundamental difference in how these methods handle the underlying data manifold explains their dramatically different performance on visual data. ResNet-18 features, for example, exist in a complex, non-linear manifold where semantic relationships between visual concepts are encoded through the deep learning training process. These features capture hierarchical visual patterns, from low-level edges and textures to high-level object concepts, and the relationships between different object categories are inherently non-linear. When PCA applies a linear transformation to this non-linear manifold, it inevitably destroys much of the semantic structure that makes visual categories distinguishable. The linear projection cannot capture the curved decision boundaries and complex topological relationships that exist in the ResNet feature space, resulting in the overlapping clusters visible in our PCA visualization.

t-SNE's non-linear approach, however, is specifically designed to handle such complex manifold structures. By focusing on preserving local neighborhoods rather than global variance, t-SNE can effectively "unfold" the non-linear manifold in a way that maintains the semantic relationships learned by the ResNet model. When two ImageNet images belong to the same category, their ResNet features are typically close in the high-dimensional space due to the CNN's (Convolutional Neural Network) training in on the visual recognition task. t-SNE preserves these local neighborhoods, ensuring that semantically similar images cluster together in the 2D visualization. Simultaneously, the optimization process pushes semantically different categories apart, creating the clear cluster separation visible in our t-SNE results. The impact is clear:

* **PCA** → Recall@1: 0.8888, mAP@1: 0.8888, Time: 0.01 s
* **t-SNE** (perplexity=30) → Recall@1: 0.9840, mAP@1: 0.9840, Time: 737.87 s (main run)
* **Speed ratio**: PCA is ~52,529× faster but loses much more local structure.

These divergent criteria interact very differently with the data manifold learned by deep networks. ResNet-18 encodes visual concepts on a highly curved, non-linear manifold whose class boundaries cannot be flattened by a linear map without distortion. PCA therefore collapses curved decision boundaries, producing overlapping clusters. t-SNE’s non-linear optimization, however, effectively “unfolds” the manifold: images that are close in the high-dimensional space (because they share semantics) remain neighbors, while dissimilar images are repelled, preserving the hierarchy of visual similarity encoded by the CNN. The results thus illustrate the critical importance of matching a dimensionality-reduction technique to data geometry when fidelity of meanings, not mere variance capture, is the goal.

1. Clustering: Deep Cluster Disentanglement (DCD)

What clustering does is it partitions data into groups of similar instances without relying on ground‑truth labels. *Deep Cluster Disentanglement* (DCD) (Yang *et al.*, 2016) couples representation learning and clustering in a single objective, overcoming the disconnect between feature learning and cluster assignment that hampers two‑stage pipelines. In their paper, their particular approach was named Joint Unsupervised Learning (JULE), a single end-to-end model that *simultaneously* (i) refines image representations with a convolutional neural network (CNN) and (ii) discovers image clusters by agglomerative merging.3 Heres’the process:

### *Mathematical Formulation*

* **Forward pass (clustering).** Starting from an initial over-clustering, at each time-step the framework selects one cluster and merges it with its most similar neighbor, interpreting the whole agglomerative procedure as the unrolling of a recurrent network.
* **Backward pass (representation learning).** After a fixed number of merges (“partial unrolling”), the current cluster assignments act as pseudo-labels that supervise the CNN through a *weighted triplet loss*. Better representations in turn update pairwise affinities, influencing the next round of merges.

Because each stage uses the output of the other, clusters and features bootstrap each other toward more discriminative representations and cleaner partitions without ever seeing ground-truth labels.

*Mathematical formulation*

For an unlabeled image the goal is to find cluster labels y and CNN parameters θ that minimize a joint loss

Holding one of the variables fixed yields the classical two-step alternation ) which JULE executes inside a single recurrent architecture. Let’s first see the agglomerative merge rule. At each time-step t the algorithm merges the two clusters with highest affinity

where affinities are computed on a directed (k)-NN graph whose edge weights are Gaussian-scaled Euclidean distances in feature space. Then there is a per-step loss.

Then the loss at timestep *t* is a combination of negative affinities:

*where* the first term encourages merging the closest neighbor and the second term penalizes similar affinities to other neighbors (capturing local density). This gives a sequence loss over a period. Accumulating over T steps in a period gives

Cluster-level to sample-level loss.  
Summing *Lt* across a period and rewriting affinities between clusters via sample affinities yields the *cluster-level* objective

with This can be approximated by an unbiased sample-level weighted triplet loss that is optimizable with mini-batches: *Ltriplet* =

where is a positive pair (same cluster) and a negative pair (different clusters. The weight γ (>1) balances intra- and inter-cluster terms. Now it is time for the optimization loop. *Partial unrolling*: after every η fraction of remaining clusters is merged, the CNN is updated for several SGD epochs with *Ltriplet* ; then affinities are recomputed and the next period begins.

Together, these equations define a recurrent agglomerative process whose forward pass performs clustering while its backward pass optimizes the CNN with a differentiable, sample-based triplet loss derived from the hierarchical clustering objective. Yang *et al.* (2016) first demonstrated DCD on STL‑10 and CIFAR‑10. Follow‑ups such as DEC, IDEC, and SCAN refine the KL term, add consistency regularization, or employ contrastive pre‑training, further validating the joint‑optimization paradigm, and thus the category of DCD.

### *When and Why to Use DCD Instead of Classical Techniques*

* **When clustering and representation learning need to be integrated**: Classical techniques (e.g., PCA + *k*-means) treat embedding and clustering as separate steps; DCD jointly optimizes both, leading to more coherent and semantically meaningful groupings.
* **When data is sparse, high-dimensional, or non-linear**: Methods like GMM or spectral clustering struggle with the curse of dimensionality and sparse feature spaces (e.g., e-commerce logs or gene expression data); DCD learns compact, task-specific embeddings on the fly.
* **When classical methods fail to differentiate subtle subgroups**: In applications like single-cell transcriptomics or image archives, DCD captures fine-grained structure that PCA, ICA, or hierarchical clustering often collapse.
* **When cluster boundaries are complex or non-convex**: Unlike *k*-means, which assumes spherical clusters, DCD can discover clusters with intricate shapes due to its non-linear feature learning.
* **When domain-specific feature engineering is impractical or weak**: DCD automates representation learning, which is critical in fields like remote sensing or cybersecurity where hand-crafted features are limited or unreliable.
* **When interpretability or disentangled factors are desired**: DCD not only groups data but can reveal latent factors underlying variation—useful for hypothesis generation or scientific discovery.
* **When visual or downstream performance matters**: Clusters produced by DCD often align better with human-perceived categories or improve downstream classification tasks compared to those from classical pipelines.

*Example Applications of DCD (JULE)*

Deep Clustering and Disentanglement (DCD) refers to end-to-end approaches for unsupervised clustering that jointly learn feature representations and cluster assignments (an example being JULE – Joint Unsupervised Learning). 3 In unsupervised image categorization, DCD methods cluster raw pixel data directly with a unified objective, instead of relying on a two-step pipeline (feature extraction followed by clustering). On datasets like ImageNet or its subsets (e.g. ImageNet-100), such end-to-end training produces semantically meaningful groupings that outperform traditional techniques like *k*-means or spectral clustering in coherence and interpretability. By simultaneously learning a deep feature space and the cluster structure, these methods capture richer representations that reflect the underlying visual similarity of images, leading to more precise clusters. For example, the JULE framework jointly optimizes a CNN and agglomerative clustering; as a result, it achieves significantly higher clustering accuracy than using *k*-means on precomputed features.3  Extensive experiments have shown that deep clustering models can surpass prior state-of-the-art clustering methods on image benchmarks, yielding clusters that align closely with true categories (often even rivaling fully supervised baselines). 3  The learned clusters are also more interpretable – often corresponding to semantic classes – compared to the fragmented or mixed clusters obtained by conventional methods. In essence, by integrating representation learning with clustering, DCD approaches produce more powerful representations and more precise clusters than pipeline approaches. 3

In domains with high-dimensional, sparse data—such as e-commerce customer behavior—DCD proves especially effective. Conventional clustering methods like Gaussian Mixture Models tend to falter under the curse of dimensionality and cannot handle sparse, high-cardinality features well. 16 In contrast, DCD learns compact nonlinear embeddings on the fly, enabling marketers and analysts to identify actionable customer cohorts for targeting or personalization. A similar advantage arises in network traffic analysis, where DCD uncovers meaningful traffic communities and emerging threat patterns that density-based methods like DBSCAN may miss due to their reliance on fixed density thresholds.

Biological and geospatial domains also benefit from DCD’s joint optimization framework. In single-cell RNA sequencing, where each cell is represented by tens of thousands of gene expression levels, DCD simultaneously compresses the data and reveals biologically relevant cell types—often outperforming hierarchical clustering methods used in pipelines like Seurat.17 Likewise, in hyperspectral remote sensing, where each pixel contains hundreds of spectral bands, DCD enables more accurate land-cover classification than affinity propagation or self-organizing maps by disentangling the spectral structure and clustering in a unified process. Self-supervised contrastive pre-training on unlabeled satellite imagery (e.g. Sentinel data) has yielded state-of-the-art results on land-cover classification benchmarks such as BigEarthNet and EuroSAT, surpassing ImageNet-pretrained baselines.18

These scenarios share a need for simultaneous non‑linear feature extraction and clustering—an area where DCD’s unified loss consistently outperforms pipeline approaches (e.g., PCA → k‑means, autoencoder → GMM).

*Demonstration of DCD on ImageNet-100*

Training and testing JULE on ImageNet-100 turned out to be harder than I thought (for just 5000 total samples from 10 classes with a test set of 1000). A “vanilla” implementation returned poor results. I tried to implement feature normalization, reconstruction loss clipping, cluster center separation loss, a stronger clustering weight, decoder freezing at epoch 40, increase epochs to 50. Even then, clusters were collapsing to 1. So I bumped up the agglomeration to every 3 epochs, rebalanced clustering loss to 2.0 decaying to 0.5, double the separation loss, implement feature diversity loss and cluster balance loss. In the end I could not get JULE to work on well on this dataset, NMI was dismally low.

Vanilla JULE suffers from fundamental algorithmic issues that cause clustering collapse. The model discovers it can trivially minimize the clustering loss by placing all samples into a single cluster, since minimizing intra-cluster distance is easiest when there's only one massive cluster. This creates a global minimum at the collapsed state. Additionally, the encoder learns features that become too similar across different classes because there's no explicit pressure to maintain feature diversity, making meaningful clustering impossible once features converge. The method also faces a chicken-and-egg problem where good features are needed for good clustering, but good clusters are needed to learn good features, and JULE attempts to solve both simultaneously, often failing at both. That being said, JULE was the paper that started the approach.

In order to get DCD to work well I had to switch to a DEC style variant. DEC stands for Deep Embedded Clustering. The method emerged as a solution to the fundamental limitations and instabilities observed in JULE's alternating optimization approach. The original DEC paper was published by Xie, et al. (2016).29

The progression from JULE to DEC represents a shift from discrete alternating optimization to continuous joint optimization. JULE's approach of periodically running external clustering algorithms like agglomerative clustering created several problems that DEC was designed to address. The discrete nature of cluster assignment updates in JULE meant that gradients could not flow through the clustering process, creating a disconnect between the feature learning objective and the clustering objective. This led to the chicken-and-egg problem where good features were needed for good clustering, but good clusters were needed to learn good features, and the two processes could not inform each other directly.

The computational expense of running clustering algorithms on the entire dataset every few epochs in JULE also created scalability issues, particularly as datasets grew larger. The hard cluster assignments produced by external algorithms were rigid and could not capture the uncertainty or soft boundaries that might exist between clusters. This rigidity often led to clustering collapse, where the algorithm would find trivial solutions like putting all samples into one cluster because it was easier to optimize.

DEC addressed these issues by making the entire clustering process differentiable and continuous. Instead of external clustering algorithms, DEC introduced learnable cluster centers as model parameters and computed soft cluster assignments using a Student's t-distribution based on feature-to-center distances. This innovation meant that both feature learning and cluster center updates could happen simultaneously through standard backpropagation, eliminating the alternating optimization problem.

The soft assignment mechanism in DEC allowed for more nuanced cluster boundaries and could capture uncertainty in assignments, while the target distribution mechanism provided a way to progressively sharpen these assignments during training. The KL divergence loss between soft assignments and target distributions created a principled way to encourage confident clustering decisions while maintaining differentiability throughout the process. DEC also introduced the concept of temperature annealing for cluster assignments, which allowed the model to start with exploratory soft assignments and gradually increase confidence as training progressed. This addressed JULE's tendency toward premature convergence to local minima. The continuous optimization framework of DEC also enabled more sophisticated regularization techniques like entropy regularization and cluster center separation losses that could be applied consistently throughout training rather than only during discrete clustering update steps. The move from JULE to DEC essentially transformed clustering from an external post-processing step to an integral part of the neural network architecture, making the entire system end-to-end trainable and more robust to the optimization challenges that plagued earlier alternating approaches. So the code in the accompanying notebook is for DEC.

`Even after making this switch tweaking was needed, later I had to revise my code with Much stronger KL weights: 1.0→5.0, temperature annealing for assignments, entropy regularization, reconstruction loss clipping (min=0.01), cluster center separation loss and feature normalization. I even froze the decoder at the 40th epoch. But the test set results improved:

=== TRAINING RESULTS ===

Best NMI: 0.9352

Final NMI: 0.8917

Final ARI: 0.6917

Final losses:

Reconstruction: 0.0215

KL divergence: 0.0057

Entropy: 0.085

===TEST SET ANALYSIS===

Extracted test data: 1000 samples

Performance Metrics:

NMI: 0.8816

ARI: 0.6584

Accuracy: 0.7010

Assignment Quality:

Mean confidence: 0.9846

Std confidence: 0.0503

Mean entropy: 0.0513

Std entropy: 0.1336

A group of graphs showing different colored lines

AI-generated content may be incorrect.

*Figure 2: “Improved” DEC-style DCD performance on a training set of 5000 samples with 10 classes*

In supervised clustering evaluation (e.g. when clustering is used as a pretext or auxiliary task), true class labels are known. You then want to measure how well your unsupervised clustering recovers the original class structure. Normalized Mutual Information (NMI) is the way to measure that in this case. A high NMI means your clustering model found structure in the data closely aligned with the real classes — even though clustering is label-free.

NMI is often preferred over raw mutual information or accuracy when evaluating clustering performance because it offers several key advantages. First, it is normalized and bounded between 0 and 1, which allows meaningful comparisons across different datasets, clustering algorithms, or numbers of clusters. NMI is also permutation-invariant, meaning it evaluates the agreement between clusters and labels based solely on groupings, not on the specific numeric values assigned to cluster IDs. This makes it robust to arbitrary labelings. Additionally, NMI is insensitive to class imbalance—unlike accuracy, which can be misleading when one class dominates—because it measures the structure shared between the cluster distribution and the label distribution. It is also symmetric, satisfying NMI(Y,C)=NMI(C,Y), reflecting the idea that both clusterings are equally important in the comparison. Finally, NMI remains stable across differing numbers of clusters or classes, avoiding the degradation that metrics like purity or raw accuracy can suffer from when the cluster count is mismatched with the number of ground-truth categories.

The evaluation of our DCD implementation reveals a highly successful outcome across all critical metrics and demonstrates the effectiveness of our comprehensive regularization strategy. From a clustering performance perspective, the results are good, with a test set NMI of 0.8816 indicating great clustering quality that surpasses the 0.8 threshold for excellence, while the ARI of 0.6584 shows strong cluster agreement with ground truth labels and the 70.1% accuracy represents good classification performance for an unsupervised learning approach. The generalization capabilities are particularly good, as evidenced by the minimal gap between validation NMI (0.8917) and test NMI (0.8816) of only 0.0101, which is far below the 0.05 threshold for excellent generalization and clearly indicates that our robust regularization techniques have successfully prevented overfitting while enabling the model to generalize exceptionally well to unseen data. The assignment quality metrics are outstanding, with a mean confidence of 0.9846 demonstrating that the model makes highly confident predictions, complemented by a low entropy of 0.0513 that indicates sharp, decisive cluster assignments rather than fuzzy or uncertain classifications, and a low standard deviation of entropy (0.1336) showing consistent assignment quality across all samples. Our loss balance achieved the equilibrium targeted, with the reconstruction loss of 0.0215 remaining well above the minimum threshold of 0.01 to prevent over-optimization while still maintaining meaningful image reconstruction without collapse, and the KL divergence of 0.0057 providing an active but controlled clustering signal that drives effective cluster formation without dominating the training process. Finally, the training dynamics demonstrate robust convergence, achieving an excellent peak performance with a best NMI of 0.9352 during training while maintaining stable convergence to a final NMI of 0.8917 without catastrophic drops, all supported by the strong entropy regularization working effectively throughout the training process to maintain balanced and stable learning dynamics. Dropping the classes to six and increasing the samples to 7000 hurt performance, as you will see in the companion notebook.

1. Supervised Learning: Harmonic Loss

Supervised neural network models are commonly trained with cross‑entropy, which minimizes the Kullback–Leibler divergence between predicted and target distributions. Harmonic Loss (Baek *et al.*, 2025) replaces the arithmetic mean implicit in cross‑entropy with the harmonic mean, yielding gradients that better balance minority‑class and majority‑class errors.4

*Mathematical Definition*

Harmonic Loss is a scale-invariant loss function designed to improve interpretability and convergence in neural networks. It modifies the standard classification pipeline in two fundamental ways: the logit definition via euclidean distance- Instead of computing logits using the inner product , Harmonic Loss defines the logits as the Euclidean distance:

where is the input representation, and is the class prototype (or weight vector) for class . There’s also a probability computation using “HarMax”: Rather than the SoftMax function, Harmonic Loss uses a *Harmonic Max* (HarMax) function with an exponent to define the probability distribution over classes:

where smaller distances correspond to higher probabilities. This gives us the loss function: Given a true class label , the Harmonic Loss is:

To use Harmonic Loss in training a neural network for classification, the architecture remains largely standard up to the penultimate layer, which produces a feature representation for each input. Instead of following this with a typical linear layer that computes logits via inner products, Harmonic Loss introduces a harmonic layer. This layer maintains a set of learnable class prototypes , with each vector corresponding to a different class. The harmonic layer computes the Euclidean distance between the feature vector and each class vector , resulting in a set of distances

.

These distances are then transformed into class probabilities using the HarMax function, which inverts and raises them to a power before normalizing: . The probability of the correct class, , is then used in a standard negative log-likelihood formulation of the loss, . This loss is backpropagated through the network in the usual fashion.

Unlike cross-entropy loss, Harmonic Loss drives convergence by encouraging the model to bring the representation close to its corresponding class center rather than pushing logits toward infinity. As training progresses, the class vectors tend to align with the centers of the examples from their respective classes, often becoming convex combinations of those examples. This behavior supports the emergence of more interpretable and stable representations. Harmonic Loss also exhibits scale invariance, meaning that scaling all weights and inputs uniformly does not affect the loss, and it leads to solutions with finite norm, unlike cross-entropy which incentivizes weight magnitudes to grow without bound.

*When to Prefer Harmonic Loss over Cross-Entropy or Another Loss*

* **Prefer Harmonic Loss in cases of severe class imbalance,** where macro metrics like F₁ or balanced accuracy matter more than overall accuracy. Harmonic Loss naturally balances gradient contributions across classes, reducing the need for manual re-weighting.
* **Use Harmonic Loss when labels are noisy or ambiguous**, as its smoother gradient behavior provides softer penalties, helping to reduce overfitting compared to focal loss or standard cross-entropy.
* **Choose Harmonic Loss in cost-sensitive applications where both high precision and high recall are critical—**such as fraud detection or medical triage—since it encourages balanced performance across all classes.
* **Favor Harmonic Loss in few-shot or long-tail learning scenarios,** where rare classes benefit from its stable optimization dynamics and reduced bias toward majority classes.
* **Apply Harmonic Loss to tasks evaluated by harmonic mean metrics,** such as information retrieval, ranking, or search relevance, where alignment between loss function and evaluation KPI leads to better practical outcomes.

*Example Applications of Harmonic Loss*

Harmonic Loss is especially valuable in medical diagnostics, where class imbalance is the norm and the cost of false negatives is high. For example, in chest X-ray classification tasks like detecting pneumonia or tuberculosis from radiographs (as in the NIH ChestX-ray14 dataset), positive cases are rare but critical to identify. Harmonic Loss helps models maintain sensitivity to these rare conditions by balancing gradient contributions without requiring manual class weighting. Again, this makes it particularly well-suited for optimizing metrics like F₁ score or balanced accuracy—often more meaningful than overall accuracy in clinical triage or early disease screening settings.19

In noisy-label scenarios, Harmonic Loss performs robustly in tasks where annotations are unreliable or ambiguous. A prime example is Webley supervised image classification (WSL), where models learn from massive datasets scraped from the internet (e.g., Open Images or YFCC100M), which often include mislabeled or loosely related examples. Unlike focal loss, which can overemphasize mislabeled "hard" examples, Harmonic Loss applies gentler gradients that reduce overfitting to noise.20 It’s also useful in crowdsourced sentiment classification, such as assigning sentiment labels to tweets or product reviews where label disagreement is high. Harmonic Loss provides better generalization when training on these noisy, user-generated labels.

In few-shot and long-tail learning, Harmonic Loss enables models to perform well across rare and frequent categories alike. Large-scale classification problems often exhibit a long-tailed distribution of classes – for example, product categorization on an e-commerce platform: a handful of categories (like “T-shirts” or “smartphones”) have thousands of examples, while niche types (e.g. *“handmade silver belt buckles”*) have only a few. Similarly, in fine-grained biodiversity recognition such as the iNaturalist species dataset, common species have ample images while many rare species have only a handful of sightings. A model trained naively will excel on the head classes but perform poorly on the tails. The goal of long-tailed learning is explicitly to boost accuracy on the tail classes without sacrificing head-class performance.21

*Demonstration of Harmonic Loss on ImageNet-100*

Harmonic Loss outperforms plain Cross-Entropy mostly on imbalanced class datasets because of the way its objective weighs each training example. By penalizing a sample unless its true-class probability is high *and* the average probability of all other classes is low, the loss delivers strong negative gradients exactly where an imbalanced dataset needs them most: on minority-class examples whose true scores start low while “distractor” scores remain high. Cross-Entropy, in contrast, is satisfied as soon as the true class outranks the others, so it tends to reward easy majority cases and lets the tail collapse. The aggregate numbers from my small demonstration confirm the intuition: balanced accuracy rises by ≈7 points, and macro-F1—an even harsher metric for class imbalance—jumps by more than 12 points. A look at the per-class table shows the gain is concentrated in classes with only three to eight validation samples, where recall lifts by as much as +0.67, while well-represented classes either hold steady or slip modestly (e.g., class 9 drops 0.22).

A group of graphs showing different types of data

AI-generated content may be incorrect.

*Figure 3: Harmonic Loss (red) achieved higher F1 and accuracy on the validation set (25 classes)*

It should be said that these results come from a run where the imbalance of the valifation phase was made4 “gentler”

A graph of different colored bars

AI-generated content may be incorrect.

*Figure 4: Harmonic Loss F1 versus Cross-Entropy F1 per class for 25 classes*

Look above, Harmonic Loss is better except for the handful of degraded classes that highlight the trade-off: because the backbone is almost entirely frozen, the network has limited representational flexibility. When Harmonic Loss “steals” probability mass to rescue rare categories, a few borderline majority categories give up some recall. Temperature scaling (τ = 2.0) further softens the soft-max; if τ is sub-optimal for a specific class, its decision boundary can blur and precision or recall falls.

To push Harmonic Loss further, improvements could target calibration and capacity. First, treating temperature as a tunable hyper-parameter (or even learning a separate τ per class) can sharpen confident predictions without forfeiting the minority-class gradient boost. Second, unfreezing one additional convolutional block of the ResNet-18 backbone would supply fresh features for rare classes, letting the optimizer act on Harmonic Loss’s stronger gradients. Third, distribution-aware augmentations such as MixUp or CutMix smooth the decision surface around scarcely sampled classes, complementing the harmonic objective. Finally, longer or cyclic learning-rate schedules give the model time to absorb these harder examples while avoiding over-fit on the head classes. Together, these adjustments should typically yield another few percentage points in macro-F1 and balanced accuracy, reinforcing the already clear advantage of Harmonic Loss for imbalanced visual recognition tasks.

1. Contrastive Learning: SimCLR

SimCLR (*Simple Framework for Contrastive Learning of Visual Representations*) is a self‑supervised method that learns image embeddings by contrasting augmented views of the same image against views of different images (Chen *et al.*, 2020). Contrastive Learning (CLR) in general is a self-supervised machine learning approach designed to learn meaningful representations by contrasting positive pairs of similar data points against negative pairs of dissimilar data points. Unlike supervised learning, it doesn't require explicit labels—making it particularly useful when labeled data is scarce or expensive to obtain. Contrastive learning aims to produce embeddings where: Positive pairs (e.g., different views or augmentations of the same sample) have similar representations. Negative pairs (e.g., views of different samples) have dissimilar representations. The objective is to learn representations in an embedding space by pulling together representations of similar samples and pushing apart representations of dissimilar samples.

*Mathematical Formulation (InfoNCE Loss)*

A common objective function used in contrastive learning is the **InfoNCE loss**, defined mathematically as:

* is the embedding of a data point (anchor).
* is a positive sample embedding (e.g., another augmented version of the same anchor).
* are negative sample embeddings.
* is typically cosine similarity or dot product.
* is the temperature parameter controlling the scale of similarities.

Minimizing this loss maximizes the similarity between positive pairs while minimizing similarity with negative samples. SIMCLR is one of these methods, as is the InfoNCE Clustering that we will cover last. Unlike pretext tasks such as rotation prediction, SimCLR directly maximizes agreement between positive pairs drawn from strong data augmentations, yielding features competitive with supervised pre‑training.5

*Mathematical Formulation of SimCLR*

Given a batch of *N* images, two stochastic augmentations *t*1​,*t*2​∼T are applied to each image, producing 2*N* views *x*~1​,…,*x*~2*N*​). An encoder *f****θ***​ (e.g. ResNet 50) and a projection head *g****ϕ***​ map every view to a latent representation **z***k*​=*g****ϕ***​(*f****θ***​(*x*~*k*​)) and normalize it to the hypersphere.

where sim(**a**,**b**)=**a**⊤**b**/∣**a**∣,∣**b**∣ and *τ* is a temperature hyper parameter. The final objective averages ℓ*i*,*j*​ over all positive pairs in the batch. Gradients flow through both *f****θ***​ and *g****ϕ***​; the projection head is discarded after pre training, and *f****θ***​ is fine tuned or frozen for downstream tasks. 5

SimCLR was introduced by Chen *et al.* (2020), demonstrating that large‑batch contrastive training with strong augmentations (crop, color jitter, Gaussian blur, horizontal flip) could match or exceed supervised ImageNet features once linear evaluation or fine‑tuning is applied. A follow‑up, SimCLR v2 scaled depth and added self‑distillation, setting new transfer benchmarks. Subsequent studies such as BYOL, Barlow Twins, and VICReg explored removing negative pairs or alternative regularizers, but SimCLR remains the canonical InfoNCE baseline. InfoNCE, or Noise-Contrastive Estimation, is a loss function used in self-supervised learning to help identify positive samples among a set of negative samples. It optimizes the probability of correctly classifying the positive sample based on its relationship with a context vector.

*When and Why to Use SimCLR Instead of Other Approaches*

* **When unlabeled data is abundant but labels are scarce**: In domains like medical imaging, remote sensing, or scientific microscopy, where annotations are expensive or require expert input, SimCLR can learn high-quality, generalizable representations from raw data alone—making it ideal for label-efficient transfer learning.
* **When strong downstream transfer performance is a priority:** SimCLR-pretrained models often fine-tune more quickly and achieve higher accuracy on tasks like classification or detection compared to models trained with autoencoders, rotation prediction, or colorization. This makes it a reliable pretraining choice for downstream supervised or semi-supervised tasks.
* **When large-scale compute resources are available:** SimCLR’s use of the InfoNCE loss benefits from large batch sizes, showing predictable scaling behavior with more GPUs or TPUs. For organizations with access to significant hardware, SimCLR is a natural choice for maximizing performance through compute scaling.
* **When robustness to data distribution shifts is important:** Because SimCLR explicitly trains models to be invariant to strong augmentations, it tends to produce representations that are more stable across domains or environments, offering better generalization than supervised models, which may overfit to annotation biases or dataset artifacts.
* **When implementation simplicity is desirable:** SimCLR uses standard data-parallel training and avoids the complexities of MoCo’s momentum queue or BYOL’s moving average target network. This makes it a clean, reproducible baseline for self-supervised learning, particularly in research or prototyping settings.

*Example Applications of SimCLR*

SimCLR has been highly impactful in computer vision tasks involving limited or non-labeled data, particularly in large-scale pretraining scenarios. For instance, researchers working with ImageNet-1K or ImageNet-100 have used SimCLR to learn high-quality image representations through contrastive learning, which are then fine-tuned for classification, detection, or segmentation.22 The core idea—learning representations by maximizing agreement between augmented views of the same image—makes it ideal for domains where manual labeling is expensive or slow. In wildlife monitoring, SimCLR has been applied to camera trap datasets like iWildCam, where labeled species data is sparse. Pretraining with SimCLR enables downstream models to recognize rare animals or unusual behaviors with much less supervision.

Similarly, in medical imaging, SimCLR provides strong benefits for pre-training on unlabeled scans, where collecting expert annotations is time-consuming and costly. For example, in chest X-ray interpretation, a model pre-trained with SimCLR on a large corpus of unlabeled X-rays (and possibly additional unlabeled images from ImageNet) outperformed a baseline that was supervised-trained on ImageNet when fine-tuned for pneumonia detection.23

SimCLR has also gained traction in remote sensing and environmental monitoring, where labeled satellite imagery is scarce and expensive to obtain. Applying SimCLR to large pools of unlabeled satellite images (for instance, Sentinel-2 or Landsat imagery) allows models to learn useful spectral and texture features. For example, the *SeCo (Seasonal Contrast)* framework extended SimCLR by leveraging seasonal changes in multi-temporal satellite data as the source of natural augmentations.18 Beyond imagery, SimCLR principles are being adapted to other modalities like graph learning and audio. For example, learning contrastive representations of bird songs or underwater sonar in biodiversity studies have been done, demonstrating its versatility in domains where unlabeled data is abundant and annotation is a bottleneck.

*Demonstration of SimCLR on ImageNet-100*

In order to demonstrate the power of SimCLR on 5 classes, the baseline I use is a four-way rotation-prediction model (often called RotNet). During pre-training, the encoder sees each image rotated to 0°, 90°, 180°, and 270° and must classify which orientation it was given. Architecturally, I matched SimCLR’s backbone (a ResNet-18 in this demo); the only difference is that its final layer is a four-unit soft-max classifier. After this proxy task is learned, I discard the rotation head and keep the encoder’s feature map as a frozen representation for the downstream linear probe. Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining key–value pairs and looking up the value associated with a given key

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Method | Class | Precision | Recall | F1-Score | Support |
| SimCLR | 0 | 0.791 | 0.680 | 0.731 | 50 |
| SimCLR | 1 | 0.750 | 0.660 | 0.702 | 50 |
| SimCLR | 2 | 0.524 | 0.660 | 0.584 | 50 |
| SimCLR | 3 | 0.750 | 0.720 | 0.735 | 50 |
| SimCLR | 4 | 0.808 | 0.840 | 0.824 | 50 |
| SimCLR | Accuracy | - | - | 0.712 | 250 |
| SimCLR | Macro Avg | 0.724 | 0.712 | 0.715 | 250 |
| SimCLR | Iighted Avg | 0.724 | 0.712 | 0.715 | 250 |
| Method | Class | Precision | Recall | F1-Score | Support |
| Rotation | 0 | 0.725 | 0.580 | 0.644 | 50 |
| Rotation | 1 | 0.613 | 0.760 | 0.679 | 50 |
| Rotation | 2 | 0.500 | 0.440 | 0.468 | 50 |
| Rotation | 3 | 0.729 | 0.700 | 0.714 | 50 |
| Rotation | 4 | 0.821 | 0.920 | 0.868 | 50 |
| Rotation | Accuracy | - | - | 0.680 | 250 |
| Rotation | Macro Avg | 0.678 | 0.680 | 0.675 | 250 |
| Rotation | Iighted Avg | 0.678 | 0.680 | 0.675 | 250 |

*Table 1: SimCLR and Baseline Classification Report*

The experiment successfully demonstrates SimCLR's superiority over rotation prediction with 71.2% versus 68.0% top-1 accuracy on a 5-class ImageNet subset (representing a relative improvement of 4.7% for SimCLR.). While the improvement margin here may be smaller than expected, both methods achieved good absolute performance, indicating that the optimization strategies were effective.

The per-class performance analysis reveals interesting patterns in how each method handles different classes. SimCLR showed the most consistent performance across classes, with precision ranging from 52-81% and recall from 66-84%. The best performing classes Ire Class 4 with 82.4% F1-score and Class 0 with 73.1% F1-score, while Class 2 proved most challenging at 58.4% F1-score, showing room for improvement. Importantly, SimCLR demonstrated balanced performance with no extreme class bias. In contrast, the rotation baseline should more variable performance with precision ranging from 50-82% and recall from 44-92%. While it achieved great performance on Class 4 with 86.8% F1-score, actually outperforming SimCLR, it struggled significantly with Class 2 at only 46.8% F1-score and should higher variance across classes overall.

Several critical changes were needed for better performance compared to the initial experiments. The most impactful change was a dataset scale-up, increasing from 640 training samples with only 128 per class to 4,250 training samples with 850 per class, representing an 8x increase that helped proper self-supervised learning at scale. I also optimized the effective batch size from an insufficient 32 samples to 512 samples through gradient accumulation, providing 511 negative samples per positive pair compared to only 31 previously. This change was crucial for effective contrastive learning in SimCLR.

The architecture and hyperparameter tuning proved equally important, with a temperature of 0.1 providing optimal contrastive learning, learning rate scaling with batch size using a 0.6 base learning rate, 5 epochs of warmup for stable large-batch training, and 2x gradient accumulation for memory efficiency. Additionally, I implemented comprehensive data quality improvements by using all 17 training shards instead of limited samples, applying quality filtering to remove corrupted samples, ensuring balanced sampling for equal class representation, and creating a proper train/validation split of 85%/15%.

The smaller-than-expected advantage for SimCLR can be attributed to several factors. First, the rotation prediction baseline achieved exceptionally high performance at 68.0% accuracy, which is remarkable for a self-supervised pretext task, indicating high-quality dataset with clear visual patterns, effective training optimization, and sufficient data scale for both methods. Second, with only 5 classes and perfect top-5 accuracy achieved by both methods, the classification task may be too simple to fully showcase contrastive learning's advantages. SimCLR typically demonstrates larger gains on tasks with more classes, more challenging visual discrimination requirements, and transfer learning scenarios. Third, the 40 epochs of training may be insufficient to fully demonstrate SimCLR's potential, as typical SimCLR experiments use 100-1000 epochs for optimal performance, with longer training benefiting contrastive learning more than rotation prediction.

The technical implementation achieved several important breakthroughs in memory management and training stability. Gradient accumulation enabled large effective batch sizes while managing memory constraints, with proper loss scaling and aggressive cache cleanup preventing memory issues. I developed a robust training pipeline with graceful recovery from out-of-memory errors, validation monitoring every 10 epochs to track overfitting, progressive memory cleanup that prevented kernel death, and quality data loading that utilized all available high-quality samples.

Looking forward, several opportunities exist for further improvement. Scaling to a larger classification task with 10-20 classes would better showcase contrastive learning advantages, while using the full ImageNet-100 dataset would provide more challenging discrimination tasks. Extended training for 100+ epochs with early stopping based on validation performance would likely reach the full potential of both methods. Advanced SimCLR techniques such as MoCo-style memory banks for additional negatives, different data augmentation strategies, and larger projection dimensions could further improve performance. Finally, architecture scaling with ResNet50 using optimized memory management or comparisons with Vision Transformers could reveal additional performance gains. While the 4.7% improvement might seem modest, achieving 71.2% accuracy on a 5-class visual recognition task using only self-supervised pretraining is a good result that preliminarily validates SimCLR's effectiveness when properly implemented.

A graph of different colored bars

AI-generated content may be incorrect.

*Figure 5: SimCLR and Baseline per class performance*

1. Probabilistic Models: Gaussian Mixture Models (GMMs)

Less recent, but very important is Gaussian Mixture Models. They provide a flexible, generative approach to density estimation and unsupervised clustering. Instead of assuming a single global distribution, GMMs model data as a weighted sum of multivariate Gaussians, allowing each component to capture a local, approximately elliptical region of the input space.6

Mathematical Formulation

For datapoints the mixture density is

Parameters are learned by maximizing the log‑likelihood

Because the posterior component assignments are latent, the Expectation–Maximization (EM) algorithm iterates:

Soft responsibilities yield probabilistic cluster memberships and naturally handle overlapping clusters.

Gaussian mixtures date to Dempster, Laird and Rubin (1977), who introduced the EM algorithm for maximum‑likelihood estimation with incomplete data. Bishop (2006) framed GMMs as a cornerstone of probabilistic machine learning, so I encountered it there. Modern extensions include variational Bayesian GMMs, Dirichlet‑process mixtures for non‑parametric model order selection, and mixtures embedded within deep latent‑variable frameworks.

*When and Why to Use GMMs Instead of Other Approaches*

* **When soft, probabilistic cluster assignments are valuable:** GMMs provide each data point with a probability of belonging to each cluster, which is especially useful in applications like speaker diarization (making diaries), customer segmentation, or gene expression profiling, where partial membership or uncertainty carries important meaning.
* **When cluster shapes follow elliptical geometry:** Unlike *k*-means, which assumes spherical clusters, GMMs can model elliptical and anisotropic clusters due to their covariance structure. This makes them suitable for data where groups align with Gaussian contours or exhibit varied spread and orientation.
* **When full density estimation is required:** GMMs learn the underlying probability distribution of the data, enabling tasks like anomaly detection through log-likelihood scoring or defining principled thresholds—capabilities that purely geometric clustering methods lack.
* **When dealing with incomplete or partially observed data:** The Expectation-Maximization (EM) algorithm used in GMMs naturally handles missing values by marginalizing over unobserved features, making them more robust than many other clustering techniques in real-world, messy datasets.
* **When generating synthetic data or simulating distributions:** GMMs support sampling from the learned mixture, making them useful for data augmentation, simulation studies, or probabilistic modeling where realism and variability are important.
* **When a principled approach to selecting the number of clusters is needed:** GMMs support model order selection using metrics like the Bayesian Information Criterion (BIC) or through nonparametric variants like the Dirichlet Process GMM, avoiding arbitrary choices of cluster count.

*Example Applications of GMM*

Gaussian Mixture Models (GMMs) are a longstanding workhorse in unsupervised learning, used for clustering and density estimation, especially when data can be modeled as coming from a mixture of Gaussian distributions. They shine in scenarios where one assumes distinct subpopulations within the data, each characterized by a Gaussian in feature space. A classic application is in speaker recognition and voice biometrics. Decades before deep learning dominated, GMMs Ire the core of systems modeling individual speakers’ vocal characteristics. For example, using features like Mel-frequency cepstral coefficients (MFCCs) extracted from speech, one can train a GMM for each speaker – effectively capturing the distribution of that speaker’s voice feature vectors.24

In astronomy and cosmology, GMMs are commonly used to model complex, multimodal distributions of objects in feature space. For instance, classifying celestial objects (stars, galaxies, quasars) from survey data often involves many measured attributes like colors (intensities in different wavelength bands) and morphological parameters. In the Sloan Digital Sky Survey (SDSS), researchers have applied GMMs to separate types of objects: each class (star vs. galaxy, etc.) can be modeled as a Gaussian cluster in the space of spectral or photometric features.24

In finance and anomaly detection, GMMs serve to model “normal” behavior in a probabilistic sense, so that unlikely events can be flagged. For example, in credit card fraud detection, one can model the distribution of legitimate transactions using a GMM over features like transaction amount, time, merchant category, location, etc. This might reveal clusters corresponding to different spending habits (e.g. daily small purchases vs. occasional large purchases). The well-known European credit card fraud dataset (a public dataset often cited in Kaggle challenges) has highly imbalanced classes (fraud vs. genuine). A trained GMM on the genuine transactions provides a likelihood score for each new transaction; those with very low likelihood under the model are considered anomalies and flagged as potential frauds.25

*Demonstration of GMM on ImageNet-100*

The GMM was initially tested on 10,000 samples across 8 classes. GMMs cluster based on visual features, not semantic labels. Multiple classes that are visually similar, such as different dog breeds, various birds, or similar objects, naturally group together in the high-dimensional feature space. The BIC criterion, or Bayesian Information Criterion, is a statistical measure used to select the best model among a set of candidate models by balancing goodness of fit against model complexity. It was developed by Gideon Schwarz in 1978 as a way to prevent overfitting when comparing models with different numbers of parameters. The BIC criterion selected 3 components as the optimal balance between model complexity and data fit, suggesting that 3 Gaussian distributions best capture the underlying visual patterns without overfitting. The 3 components likely represent higher-level visual categories, with Component 1 capturing the dominant visual pattern at 44.2% weight, Component 0 representing a secondary pattern at 20.3% weight focusing on different lighting or color schemes, and Component 2 accounting for 35.5% weight with distinct geometric features. The GMM finding only 3 components instead of 8 classes actually reveals important insights about how GMMs work and what they're discovering in the visual data. This result is expected and demonstrates GMM's strength in discovering natural patterns rather than forcing predetermined categories.

The performance assessment reveals excellent probabilistic clustering characteristics. The model achieved high confidence with 89.7% of samples having responsibility greater than 0.8, indicating very confident cluster assignments. The low average entropy of 0.138 demonstrates minimal uncertainty in assignments, while the high average maximum responsibility of 0.844 shows strong cluster membership decisions. Only 6.2% of samples exhibit multi-membership in two or more components, indicating clean cluster boundaries.

The components are well-separated, as evidenced by negative correlations between components ranging from -0.44 to -0.63, indicating they capture distinct, complementary patterns. The responsibility distributions show each component has a distinct signature, and the component separation analysis reveals good distance between Gaussian means. The anomaly detection successfully identified 500 anomalies representing 5% of the dataset using likelihood thresholds, with clear separation in the log-likelihood distribution. These anomalies likely represent unusual lighting, poses, or image artifacts.

This 3-component solution is actually superior to forcing 8 components because semantic categories do not necessarily correspond to visual clusters. Dogs and cats might be visually more similar to each other than to cars, and different lighting conditions of the same object type might create separate visual clusters. The model demonstrates robustness by finding 3 stable components rather than 8 potentially unstable ones, reducing overfitting to training data specifics and creating a more generalizable model for new image data.

The visualizations provide compelling evidence of the model's success. The t-SNE plots show clear separation between GMM clusters, with uncertainty concentrated at boundaries where it naturally should be. The responsibility matrix demonstrates clean, decisive assignments for most samples, while the component analysis reveals balanced weights suggesting all components are meaningful with no degenerate clusters. The overlap analysis shows minimal correlation between components, indicating they capture orthogonal visual patterns.

The GMM successfully discovered the natural visual structure in the ImageNet data, which doesn't necessarily align with human semantic categories. This is exactly what unsupervised learning should accomplish, finding patterns in the data without being constrained by predefined labels. The 3-component solution is optimal because it captures the main visual variations in the dataset, provides confident low-uncertainty assignments, maintains good separation between discovered patterns, and avoids overfitting to semantic categories that may not correspond to visual similarity. This demonstrates GMM's power in discovering latent visual structure that might be more fundamental than semantic labels for computer vision applications.

A group of graphs with different colored lines

AI-generated content may be incorrect.

A graph of a number of numbers and a number of blue and red lines

AI-generated content may be incorrect.

*Figure 6: GMM metrics visualized on 10,000 samples across 8 classes*

A screenshot of a computer screen

AI-generated content may be incorrect.

*Figure 7: GMM anomaly detection graph*

1. Graph‑Based Learning: node2vec Graph Embeddings

Graph embeddings map nodes in a network to low‑dimensional vectors that preserve structural and (optionally) semantic proximity. The resulting representations let standard machine‑learning models operate on graphs without bespoke kernels or combinatorial features. *Node2Vec* (Grover & Leskovec, 2016) remains a canonical method thanks to its balance between local (breadth‑first) and global (depth‑first) exploration via biased random walks.7

*Mathematical Formulation*

Node2Vec is a framework for learning continuous feature representations of nodes in a graph . The goal is to learn a mapping that embeds each node into a -dimensional vector space, such that nodes with similar network neighborhoods are embedded closely together.

The objective function is based on the Skip-gram model and aims to maximize the likelihood of preserving sampled network neighborhoods:

where is the neighborhood of node , sampled using strategy , and is the embedding of node .

Assuming conditional independence and symmetry, the conditional likelihood is factorized as:

with:

Thus, the optimization simplifies to:

where is approximated using negative sampling for tractability.

Node2Vec builds on this objective by defining a biased random walk mechanism to generate , allowing flexible interpolation between BFS (favoring local neighborhoods) and DFS (favoring far-away nodes). This flexibility enables the algorithm to capture both homophily and structural equivalence in the embeddings.

For a given node , node2vec simulates multiple biased random walks of length , each generating a sequence of nodes. These sequences are treated analogously to sentences in NLP, and the Skip-gram model is used to learn node embeddings from them.

The transition probability from current node to node is defined as:

where:

* is the previous node before in the walk.
* is the edge weight (1 for unweighted graphs).
* is a bias based on the shortest path distance from node to :

Here:

* **Return parameter**  controls the likelihood of revisiting a node (low encourages staying local like BFS).
* **In-out parameter**  controls the likelihood of moving outward (low encourages DFS-like behavior).

After generating walks, the sequences are used to optimize the Skip-gram objective via stochastic gradient descent. The embeddings can also be extended to edges using binary operators (e.g., Hadamard, average, weighted-L1/L2) applied to the embeddings of node pairs.

In short, node2vec learns embeddings by maximizing the probability of network neighborhoods under a flexible, biased random walk sampling strategy. The learned representations are task-independent and can be used in downstream tasks such as node classification and link prediction. The tunable parameters and control how walks explore the graph structure, offering a continuum between local and global context preservation.

*When and Why to Use node2vec Instead of Other Approaches*

* **When working with large, sparsely labeled graphs:** In domains like social networks, citation graphs (e.g., Cora, DBLP), or protein–protein interaction networks, node2vec can learn unsupervised embeddings from topology alone, enabling downstream tasks such as node classification even when only a small portion of nodes are labeled.
* **When node features are missing or minimal:** node2vec is ideal for featureless graphs, as it learns embeddings purely from graph structure, unlike many graph neural networks (GNNs) that require node attributes to function effectively.
* **When scalability and speed are essential:** The use of random walk sampling and Skip-gram optimization allows node2vec to scale to graphs with millions of nodes, offering faster training and lower memory usage than deep GNNs, especially on resource-constrained systems.
* **When static embeddings are preferred for deployment:** In recommendation systems, link prediction, or anomaly detection pipelines where embeddings are precomputed and cached, node2vec’s fixed-length, once-trained vectors provide efficient runtime inference without the need for message passing or dynamic computation.
* **When performing exploratory analysis or visualization:** node2vec embeddings often produce clearer cluster separation in 2D projections (e.g., via t‑SNE or UMAP) compared to raw adjacency matrices, making it a practical tool for revealing community structure or functional groupings in graphs.
* **When fine-grained control over neighborhood sampling is needed:** node2vec’s biased random walk parameters (p, q) allow users to tune the tradeoff between local proximity (community detection) and structural similarity (role-based clustering), offering task-specific flexibility not available in simpler models like DeepWalk or LINE.

*Example Applications of Graph Embeddings (Node2Vec)*

Node2Vec is often used for node classification (predicting a label for each node, e.g. user “type” in a social network), link prediction (estimating the likelihood of an unobserved or future edge between two nodes, useful for recommender systems or knowledge-graph completion), clustering and community detection (grouping nodes into densely connected modules by simply running k-means or another clustering algorithm on the embeddings), anomaly detection (identifying nodes whose embeddings lie far away from their expected community or structural signature), and visualization (projecting the learned embeddings into 2D or 3D to reveal the graph’s latent structure).

Node2Vec is a popular algorithm for learning continuous feature representations (embeddings) of nodes in a graph, and it has found extensive use in social network analysis and beyond. By capturing the network’s structural patterns, node2vec embeddings enable a variety of tasks such as community detection, link prediction, and node classification. In social networks (like friendship or follower graphs), node2vec helps uncover both local and global relationship structures. For example, in the case of Facebook “Circles” (friend groups) or ego-networks on Snapchat, node2vec can embed each user into a vector space such that users who are close (strongly connected or structurally similar) end up near each other in the embedding. Clustering these embeddings can reveal communities – groups of users who interact densely with each other (families, college friends, colleagues, etc.)26

In biological network analysis, node2vec has shown strong performance in tasks involving protein–protein interaction (PPI) networks, gene co-expression or regulatory networks, and disease-gene association graphs. For example, in the STRING database of PPI or BioGRID network, each protein is a node and edges indicate interactions. node2vec embeddings of proteins can then be clustered to discover protein functional modules – groups of proteins that work together in a complex or pathway.26

In recommender systems and knowledge graphs, node2vec provides dense vector representations of users, items, or entities that can be plugged into neural networks or similarity search for recommendations. Consider a graph where users are connected to items they have purchased or liked (a bipartite user–item graph). Running node2vec (or a similar bipartite embedding) yields user embeddings and item embeddings such that a user vector is close to vectors of items they have interacted with. E-commerce platforms like Amazon can use these as features to compute item-item similarities (“people who like *X* also like *Y*”) or to feed them into downstream ranking algorithms. On Pinterest, for example, a co-pin graph (users pinning images to boards) can be embedded to improve content discovery: node2vec-like embeddings were used to power the “Related Pins” feature by measuring vector distances between pins (items).27

*Demonstration of Graph Embeddings on ImageNet-100*

Although it didn’t outperform the baseline, the demonstration code worked on 5000 samples at k=15 and a threshold of 0.3, and the detailed analysis shows that, in classification tasks, Node2Vec achieved accuracies of 0.969 and F1-scores of 0.991 with Logistic Regression, 0.966 with Random Forest, and 0.969 with SVM, whereas the PCA-based baseline yielded higher scores of 0.991, 0.984, and 0.987, respectively. In clustering, Node2Vec produced a positive silhouette score of 0.106, an ARI of 0.793, and an NMI of 0.854, compared to the baseline’s silhouette of –0.012, ARI of 0.696, and NMI of 0.855. In the similarity analysis, Node2Vec achieved a mean within-class similarity of 0.483 and between-class similarity of 0.265, while the baseline should 0.048 and –0.005. For neighborhood purity, Node2Vec consistently outperformed the baseline at k=5 (0.925 vs. 0.875), k=10 (0.924 vs. 0.839), and k=20 (0.921 vs. 0.788). Finally, in terms of linear separability, Node2Vec scored a mean AUC of 0.988 (±0.011) versus the baseline’s 0.999 (±0.001).

A comparison of a bar graph

AI-generated content may be incorrect.

*Figure 8: node2vec classification performance and cluster quality*

A close-up of a graph

AI-generated content may be incorrect.

A group of colored dots

AI-generated content may be incorrect.

*Figure 9: Node2Vec Neighborhood purity, confusion matrix and t-SNE embeddings*

1. Self‑Supervised Methods: VICReg

VICReg (*Variance‑Invariance‑Covariance Regularization*) learns representations by maximizing invariance between paired views while explicitly preventing collapse through variance and covariance regularizers (Bardes *et al.*, 2021). Unlike contrastive approaches, VICReg requires no negative samples or momentum encoders yet achieves competitive transfer performance.8

*Mathematical Formulation*

VicReg is a self-supervised learning method that learns useful and non-collapse-prone representations by optimizing a loss function composed of three components: invariance, variance, and covariance. Given two augmented views of the same input, and , passed through a neural network encoder , the outputs are embeddings and .

Let be the batch of embeddings of dimension , for a minibatch of samples.

The total VicReg loss is:

#### 1. Invariance Loss ()

A mean-squared error (MSE) loss that brings corresponding pairs of embeddings closer:

#### 2. Variance Loss ()

Encourages non-trivial (non-collapsed) representations by penalizing small variance along each dimension:

Where:

* is the variance of the -th dimension over the batch.
* is a threshold hyperparameter (typically 1).
* is a small constant for numerical stability.

#### 3. Covariance Loss ()

Prevents redundant representations by decorrelating dimensions:

Where is the empirical covariance matrix of the batch (after centering).

So we can say VicReg learns useful representations by combining three regularization forces that guide the training process. The invariance term ensures that embeddings of different augmented views of the same input are brought close together in representation space, promoting consistency across transformations. At the same time, the variance term prevents the model from collapsing to trivial solutions by enforcing that the standard deviation of each dimension in the batch embeddings remains above a minimum threshold. This keeps the learned features expressive and diverse across samples.

To further improve the quality of the representations, VicReg includes a covariance term that penalizes linear dependencies between different dimensions of the embedding. This encourages the model to produce decorrelated features, which helps in disentangling information and improving downstream task performance.

During training, a batch of input data is augmented in two different ways, and both augmented views are passed through the same encoder network to produce two sets of embeddings. These embeddings are then used to compute the three loss components—invariance, variance, and covariance—which are combined into a total loss. This total loss is backpropagated to update the encoder parameters.

Unlike contrastive methods, VicReg does not require negative samples. It relies purely on architectural design and its regularization terms to avoid collapse and maintain feature richness. This makes VicReg a scalable and efficient approach for self-supervised learning, particularly in settings where contrastive pairs are hard to define or expensive to sample.

*When and Why to Use VICReg Instead of Other Approaches*

* **When large batches or memory banks are impractical:** VICReg trains effectively with moderate batch sizes, avoiding the need for large numbers of negative samples as required by contrastive methods like InfoNCE or SimCLR.
* **When architectural simplicity is a priority:** VICReg requires only a single network without auxiliary components like momentum encoders, predictor heads, or memory queues—making it easier to implement, tune, and debug.
* **When training stability under batch variation is important:** Its explicit variance regularization term ensures all embedding dimensions remain active, providing robustness to fluctuations in batch composition and reducing reliance on careful batch construction.
* **When working with non-vision modalities: For** tabular data, audio signals, or graph-structured inputs—where defining negatives is ambiguous—VICReg’s positive-pair-only training paradigm makes it more suitable than contrastive alternatives.
* **When representation collapse must be avoided directly:** VICReg provides interpretable, loss-level control over collapse via its variance and covariance penalties, reducing the need for implicit tricks or tuning heuristics.
* **When operating under limited compute budgets:** Without reliance on complex architectural tricks or expensive sampling strategies, VICReg is efficient to train and deploy on modest hardware, while still achieving competitive performance on downstream tasks.

*Example Applications of VicReg*

VicReg (Variance-Invariance-Covariance Regularization) is a relatively new approach in self-supervised representation learning, notable for enabling representation learning without requiring negative samples or explicit contrastive pairs. It has gained traction in computer vision tasks as a simpler alternative to contrastive frameworks like SimCLR or BYOL. On benchmark datasets like ImageNet, VicReg has demonstrated competitive performance by learning embeddings that satisfy three criteria: (i) *invariance* to data augmentations (similar to other SSL methods, it makes representations of two augmented views of the same image close), (ii) maintaining *variance* (the embeddings don’t collapse to trivial constants), and (iii) minimizing redundancy between dimensions (to avoid encoding duplicate information).5

In medical imaging, VicReg is valuable for pretraining models on large, unlabeled datasets such as NIH ChestX-ray14 or CheXpert, where label scarcity and class imbalance are persistent challenges. By learning robust visual representations from chest X-rays, CT scans, or MRIs using only image augmentations, VicReg allows practitioners to build high-quality feature extractors that can be fine-tuned for tasks like disease classification or lesion detection with minimal labeled data. Unlike contrastive methods, VicReg doesn't rely on large batch sizes or memory banks, which is advantageous in medical settings where compute is often limited and training stability is critical. Beyond vision, VicReg has been adapted to multimodal and structured data domains, such as time series sensor data in industrial IoT or biological sequences in genomics. For instance, in human activity recognition from wearable devices (e.g., with datasets like UCI HAR), VicReg can learn representations that are invariant to device position or motion noise while preserving the variance needed to distinguish between activities like walking, running, or climbing stairs. In gene expression modeling, it has been explored as a way to regularize embeddings of high-dimensional vectors, capturing patterns across samples without relying on labeled conditions. This flexibility makes VicReg an emerging tool for any domain where paired augmentations exist but labels do not.5

*Demonstration of VicReg on ImageNet-100*

In this demonstration, the baseline uses a ResNet-18 pretrained on ImageNet with supervised end-to-end fine-tuning on a 5-class subset. It achieves 96-97% accuracy by leveraging both the powerful pretrained visual features and direct optimization with ground truth labels. The baseline model includes 30% dropout for regularization and uses Cross Entropy Loss with label smoothing, representing a very strong supervised learning approach. This is a very strong learner and tough to beat.

I found VicReg difficult to train. Initial results were very poor, underperforming the baseline by as much as 45%+. I limited the demo to 3 classes. To achieve 90%+ accuracy with VICReg, I tried to enhance the model, significantly scaling up the training regime. The initial improvements included extending training to 50 epochs with 100 batches per epoch (this proved unnecessary and slowed training). Also, my GPU couldn’t handle using a larger batch size of 512 (I used 128), and I could not upgrade from ResNet-18 to a ResNet-50 backbone to match the baseline's representational capacity given my GPU’s CUDA memory. So, I tried other fixes, including implementing much stronger data augmentations including more aggressive cropping (scale 0.05-1.0), stronger color jittering, higher grayscale probability, and multi-scale training across different image sizes. These changes addressed VICReg's fundamental requirement for diverse and challenging positive pairs to learn meaningful representations.

The second critical area I tried is improving the linear evaluation phase, which directly impacts final accuracy. This involves replacing the simple linear classifier with a deeper 3-layer MLP featuring dropout regularization. Making these changes the VicReg model seemed to converge after only a few epochs (see graph below), so I stopped there. These architectural and training improvements, obviated the need for an extended training regime, and pushed VicReg performance from the 40%’s to 94%+. The final gap between VICReg (94.67%) and the enhanced baseline (96.67%) is now only 2.00%, which falls Well within the expected range for state-of-the-art self-supervised methods and represents a negligible difference compared to the original massive underperformance.

A screenshot of a graph

AI-generated content may be incorrect.

*Figure 10: VicReg performance versus baseline*

1. Multimodal Learning: CLIP‑Style Cross‑Modal Embeddings

CLIP (Contrastive Language–Image Pre‑Training) learns a joint embedding space in which paired images and natural‑language captions are aligned (Radford *et al.*, 2021). By training on hundreds of millions of image–text pairs gathered from the web, CLIP enables zero‑shot transfer to downstream vision tasks through simple prompt engineering (without task‑specific fine‑tuning).9

*Mathematical Formulation*

CLIP (Contrastive Language-Image Pretraining) learns a joint embedding space for images and texts using a contrastive learning objective. Let be an image and a text description associated with that image. Two separate encoder networks are used:

* : image encoder
* : text encoder

Both embeddings are normalized to unit length:

The similarity between image and text is computed via cosine similarity (dot product due to normalization):

Given a batch of (image, text) pairs , CLIP uses a symmetric contrastive loss: cross-entropy losses are computed over image-to-text and text-to-image directions.

The image-to-text loss is:

The text-to-image loss is:

The total loss is:

Here, is a learnable temperature parameter that controls the sharpness of the softmax distribution.

So CLIP works by learning two separate encoders—one for images and one for text—that are trained to map their respective modalities into a shared embedding space where semantically related image-text pairs are close together. During training, the model sees batches of paired images and captions and learns by distinguishing which image corresponds to which text. For every image, the model tries to correctly identify its matching caption among all captions in the batch, and likewise, for every caption, the model tries to match it to the correct image.

This bidirectional contrastive learning setup encourages both encoders to produce aligned embeddings for corresponding image-text pairs while pushing apart the representations of mismatched pairs. A learnable temperature parameter controls the scale of similarity scores, sharpening the model’s ability to discriminate between correct and incorrect matches. The model is trained on a large dataset of image-caption pairs scraped from the internet, allowing it to learn general-purpose visual and linguistic representations without task-specific supervision. After training, the resulting encoders can be used to perform zero-shot classification by encoding a set of textual class descriptions and selecting the one whose embedding is most similar to the image.

A diagram of a diagram

AI-generated content may be incorrect.

*Figure 11: CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of image and text. Image Credit: (Radford, et al. 2021)*

Radford *et al.* (2021) introduced CLIP and demonstrated strong zero‑shot accuracy on ImageNet and other benchmarks. Later models such as CoCa, FLAVA, and Florence extend the paradigm with multi‑task objectives, larger corpora, or additional modalities (audio, video). Despite these advances, CLIP‑style contrast remains the reference architecture for scalable multimodal pre‑training.

*When and Why to Use CLIP‑Style Embeddings Instead of Other Approaches*

* **For zero-shot classification and cross-modal retrieval:** CLIP-style models enable prompt-based image recognition, natural language image search, and even art or content generation guidance—all without requiring labeled data for the target task, making them ideal in low- or no-supervision settings.
* **When flexible downstream transfer is needed:** A single pretrained dual encoder (image + text) can support a wide range of tasks—like segmentation, object detection, or visual question answering—by adding lightweight adapters or prompt tuning, avoiding the need to retrain the full model.
* **For scalable pretraining on noisy, large-scale datasets:** CLIP’s contrastive learning framework efficiently leverages massive collections of image–text pairs (e.g., from the web or social media) without requiring manual annotation or curated labels, making it practical for web-scale training.
* **When efficient inference is critical:** The use of separate image and text encoders allows for offline embedding and indexing of one modality (e.g., all images), enabling fast real-time retrieval with simple text queries—a major advantage for recommendation, search, or content moderation systems.
* **To extend across modalities beyond images: The** CLIP framework is modality-agnostic—by swapping in audio, video, or 3D encoders, the same contrastive loss setup can be applied to learn joint representations for tasks like video captioning, audio retrieval, or 3D scene understanding.
* **To improve robustness under domain shift:** Because CLIP is trained on a broad, noisy distribution of web data, its embeddings generalize better than supervised models (e.g., ImageNet classifiers) when evaluated on out-of-distribution samples, making it suitable for open-world and real-world applications.

*Example Applications of CLIP*

CLIP (Contrastive Language-Image Pre-training) and similar vision-language embedding models have revolutionized zero-shot image classification by enabling models to directly connect images with natural language descriptions. Trained on hundreds of millions of image–text pairs (like those from the LAION or YFCC datasets), CLIP learns a joint embedding space where corresponding text and image representations are close together. This capability has proven especially useful in fine-grained object recognition and open-vocabulary classification. For example, consider identifying bird species from images: traditional classifiers would need a fixed set of species and many labeled examples per species (as in the Caltech-UCSD Birds 200 dataset). CLIP, on the other hand, can recognize hundreds of bird species zero-shot by using their names or descriptions as text prompts. A CLIP model might take an image of a bird and calculate similarity to text prompts like “a photo of a red bird with a long beak” or “a photo of a sparrow” or “a photo of a Northern Cardinal,” etc., and if the correct description is among those, the image will have highest similarity with the matching text. This allows fine-grained identification without explicitly training on that classification task.

In content moderation and visual search, CLIP-style models are applied to detect inappropriate content or find semantically similar items without requiring exact keyword matches. Platforms like Pinterest, Reddit, or Instagram can use CLIP embeddings to flag images based on textual cues such as “offensive meme” or “self-harm content,” even when these items haven’t been seen during training. Similarly, in e-commerce, CLIP enables cross-modal retrieval, such as allowing users to search for products using descriptive natural language queries (“vintage leather boots under $100”) and retrieving visually relevant items. This reduces reliance on structured tags or manual curation, making search more intuitive and scalable.

CLIP-style embeddings have also seen rapid adoption in scientific and medical applications where labeled datasets are scarce and domain adaptation is essential. In radiology, researchers have used adapted CLIP models to align chest X-ray images with diagnostic reports (e.g., from MIMIC-CXR), allowing for zero-shot disease retrieval or report generation. In remote sensing, CLIP-style training helps link satellite imagery with textual metadata (e.g., land use or disaster descriptions), improving retrieval and monitoring in data-scarce regions. The generalization power of these embeddings—across both modalities and domains—makes them a strong foundation for building interpretable, language-grounded AI systems that require minimal task-specific fine-tuning.

*Demonstration of CLIP-style Embeddings on ImageNet-100*

The CLIP-style cross-modal embedding demonstration achieved good success with an 85.2% test accuracy on a 3-class ImageNet subset, substantially outperforming both baselines and demonstrating effective zero-shot learning capabilities. This represents a significant improvement over the theoretical random baseline of 33.3% and empirical random performance of 33.6%, indicating that the model has learned meaningful cross-modal representations rather than relying on chance. Most importantly, CLIP outperformed the supervised CNN baseline by 11.2 percentage points (85.2% vs 74.0%), which is particularly impressive given that CLIP learns through contrastive self-supervision without explicit class labels during training, while the CNN receives direct supervised signals.

The training dynamics reveal healthy convergence with minimal overfitting: training accuracy progressed from 58.9% to 86.9% over 5 epochs, while validation accuracy closely tracked at 85.2%, indicating excellent generalization. The minimal 1.7% gap between train and test performance demonstrates that the model successfully learned generalizable cross-modal features rather than memorizing the training data. The InfoNCE loss decreased consistently from 2.54 to 2.06, and the learned temperature parameter stabilized at 4.88, suggesting the contrastive learning objective effectively aligned image and text representations in the joint embedding space.

The batch-level evaluation metrics provide additional insights into the model's robustness, with per-batch accuracies ranging from 69% to 100% across the 38 test batches, demonstrating consistent performance across different data distributions. The similarity scores between image and text embeddings span a healthy range from -0.18 to 0.95 with a mean of 0.41, indicating that the model learned to distinguish between matching and non-matching image-text pairs. This successful demonstration validates the core CLIP methodology of learning transferable visual representations through natural language supervision, achieving strong zero-shot performance that exceeds traditional supervised approaches while requiring no task-specific fine-tuning.

A graph of a graph of a graph

AI-generated content may be incorrect.

*Figure 12: CLIP training loss and accuracy*

A graph showing different colored squares

AI-generated content may be incorrect.

*Figure 13: CLIP and baseline performance on the test set*

1. Representation Learning with Neighborhoods: Student‑ Embedding (t-SimCLR)

Neighborhood‑preserving representation learning aims to map high‑dimensional points into a latent space where nearby items remain close while distant items are separated. Student‑ Embeddings or t-SimCLR (Hu *et al.*, 2023) achieve this by employing a heavy‑tailed Student‑ kernel that allocates greater probability mass to moderate distances, alleviating the “crowding problem” common in Gaussian‑based objectives.10

*Mathematical Formulation*

t‑SimCLR is a Student‑t distribution-based generalization of SimCLR, a contrastive learning framework. In t‑SimCLR, each data sample is encoded not as a point in Euclidean space but as a multivariate Student‑t distribution. Each encoder output is parameterized by a mean , a scale vector (defining diagonal covariance), and a degree of freedom . Thus, the representation of a sample becomes:

Given a batch of pairs of positive samples , the encoder produces two distributions for each input. Let and denote the distributions of the two views of sample . The similarity between two such distributions is defined as the log-likelihood of one mean under the other distribution:

where is the mean of the distribution from the second view of sample , and is the Student‑t PDF.

The t‑SimCLR objective is

where *tdf* is the degree of freedom for the t-distribution.

t‑SimCLR generalizes the standard SimCLR framework by replacing point embeddings with full multivariate Student‑t distributions, allowing each encoded sample to express not just a position in space but also uncertainty and heavy-tailed behavior. In the original SimCLR, a pair of augmented views is passed through an encoder and a projection head, and contrastive loss is used to maximize agreement between positive pairs while minimizing it between negatives. t‑SimCLR modifies this setup by having the encoder output the parameters of a Student‑t distribution—specifically a mean, variance (via diagonal covariance), and degrees of freedom—for each view.

Instead of computing similarity using cosine similarity between two points, t‑SimCLR evaluates how likely the mean of one distribution is under the density of the other. This makes the similarity asymmetric and probabilistic, which can better capture the semantic variability in real data. The contrastive loss is then computed using these similarity scores, encouraging true pairs to have high likelihood and mismatched pairs to have low likelihood under the Student‑t density.

This approach introduces flexibility in modeling uncertainty, handles heavy-tailed distributions more naturally than Gaussians, and enables the learning of embeddings with a rich geometric structure. During training, all parameters—including the mean vectors, scale, and optionally the degrees of freedom—are learned via backpropagation. The use of the Student‑t distribution allows t‑SimCLR to retain the strong empirical performance of contrastive methods while offering additional expressivity through its probabilistic framework.

Hu *et al.* (2023) introduced Student‑ Embeddings as a general‑purpose neighbourhood‑preserving loss suitable for deep encoders; they reported improved cluster fidelity and global geometry on image and single‑cell datasets compared with Gaussian kernels. Earlier work on *t‑SNE* employed a similar kernel but focused on non‑parametric visualization rather than learned representations. Subsequent studies have adopted Student‑ similarities for graph contrastive objectives and as drop‑in replacements for cosine similarity in metric learning.

*When and Why to Use t-SimCLR Instead of Other Approaches*

* **When the data has hierarchical or manifold structure:** Student-*t* kernels preserve both immediate neighbors and mid-range relationships due to their heavy tails, making them ideal for revealing hierarchical clusters that Gaussian kernels often obscure—especially in biological, linguistic, or visual domains.
* **When addressing the crowding problem in latent space:** In methods like *t*-SNE or deep neighbor-based training, Gaussian kernels tend to compress intermediate distances. Student-*t* embeddings mitigate this by allowing points to spread out more naturally, improving the clarity and interpretability of the embedding.
* **When modeling data with high intrinsic dimensionality:** The Cauchy-like tail of the Student-*t* distribution helps reduce gradient saturation during training, making it easier to learn meaningful embeddings when similarity is distributed across many moderately active dimensions.
* **When both local and global structure must be preserved:** Applications such as single-cell transcriptomics, protein folding embeddings, or large-scale recommender systems benefit from Student-*t* embeddings’ ability to maintain local neighborhoods while also preserving the broader global layout essential for downstream tasks like clustering or graph traversal.
* **When integrating with deep learning architectures:** Student-*t* losses are fully differentiable and can be paired directly with CNNs, Transformers, and graph neural networks. They offer a clean alternative to contrastive losses by avoiding the need for negative sampling, memory queues, or large batch sizes.

*Example Applications of Student-t Embeddings (t-SimCLR)*

Student-*t* embeddings are especially valuable in biological data analysis, where high-dimensional and noisy datasets often exhibit heavy-tailed distributions and complex, non-linear relationships. In single-cell RNA sequencing (scRNA-seq), for instance, traditional latent space methods like PCA or Gaussian-based VAEs can fail to preserve rare cell populations or subtle transcriptional gradients. Student-*t* variational autoencoders (VAE-*t*) have been applied to datasets such as PBMCs or Tabula Muris, where they yield better separation of rare cell types and improved visualization of cellular differentiation trajectories. The heavier tails of the Student-*t* distribution allow embeddings to reflect biological heterogeneity without collapsing minority subgroups, which is crucial for tasks like cell-type discovery and lineage inference.

In cybersecurity and anomaly detection, Student-*t* embeddings are used to model rare or adversarial behaviors in high-dimensional telemetry data. For example, in network intrusion detection systems trained on datasets like NSL-KDD or UNSW-NB15, the ability of Student-*t* representations to handle outliers and tail-heavy distributions makes them more robust than Gaussian embeddings when detecting rare but critical anomalies such as advanced persistent threats or zero-day exploits. These embeddings improve the performance of downstream tasks like clustering suspicious sessions or visualizing atypical user behavior, supporting both automated detection and human-in-the-loop investigation.

Student-*t* embeddings have also proven effective in visual representation learning, particularly in deep clustering and self-supervised learning frameworks. In methods like t‑SNE or Deep Embedding Clustering (DEC), replacing Gaussian assumptions with a Student-*t* kernel results in tighter, more separable cluster formations in datasets like MNIST, CIFAR-10, or Fashion-MNIST. The heavy-tailed distribution facilitates sharper modeling of local densities, which is especially beneficial when the goal is to uncover latent class structure or compress visual content for downstream tasks. In generative models such as Student-*t* VAEs, the added flexibility in the latent space often leads to improved sample diversity and robustness under data shifts, making them a practical choice for image synthesis or unsupervised classification in non-Gaussian domains.

*Demonstration of Student-t on ImageNet-100*

The demonstration compares Student-t and Gaussian embedding models on a 6-class classification task with 6,000 training samples, where the Student-t model demonstrates superior performance across all metrics, achieving 99.00% test accuracy compared to 98.58% for Gaussian embeddings, with particularly strong improvements in clustering performance. The experiment used a balanced dataset with 6,000 training samples, 1,200 validation samples, and 1,200 test samples, with 1,000 samples per class for training and 200 samples per class for validation and testing. Both models had identical architectures with 11.6 million parameters, ensuring a fair comparison between the two embedding approaches.

The Student-t model showed rapid convergence and stability during training, achieving 87.7% training accuracy by the first epoch and maintaining consistent performance with minimal overfitting throughout the five training epochs. The final training accuracy reached 97.5% with a validation accuracy of 99.0%, and the training loss decreased smoothly from 0.54 to 0.15. In contrast, the Gaussian model exhibited more volatile training dynamics, starting with a slightly higher initial accuracy of 89.5% but showing concerning performance degradation in the final epoch where training accuracy dropped from 96.0% to 94.6%. The Gaussian model also displayed more erratic loss patterns throughout training.

On the test set, the Student-t model achieved 99.00% accuracy, 99.01% precision, 99.00% recall, and 99.00% F1 score, while the Gaussian model achieved 98.58% accuracy, 98.62% precision, 98.58% recall, and 98.58% F1 score. This represents consistent improvements of approximately 0.4% across all classification metrics for the Student-t approach.

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 14: Performance of Student-t Embeddings*

The clustering performance reveals more significant differences between the two approaches. The Student-t model achieved an Adjusted Rand Index (ARI) of 0.9684 compared to 0.9513 for the Gaussian model, representing a 1.71% improvement. Similarly, the Normalized Mutual Information (NMI) score was 0.9604 for Student-t versus 0.9524 for Gaussian, showing a 0.80% improvement. Interestingly, the Gaussian model performed better on the Silhouette Score with 0.7494 compared to 0.6789 for Student-t, suggesting some trade-offs in different aspects of cluster quality.

The Student-t model's heavy-tailed distribution appears to provide greater robustness during training, evidenced by more stable training curves, consistent performance across epochs, and better generalization to test data. This robustness likely stems from the Student-t distribution's ability to better handle outliers and extreme values during the learning process.

The clustering performance differences suggest that Student-t embeddings create more cluster-friendly representations, with higher ARI and NMI scores indicating better cluster separation and preservation of class structure in the embedding space. The heavy-tailed nature of the Student-t distribution may contribute to better outlier handling, which is particularly beneficial for clustering tasks where maintaining clear boundaries between classes is crucial.

The Gaussian model's performance drop in the final epoch suggests potential overfitting issues or sensitivity to learning rate scheduling, while the Student-t model maintains stability throughout training. This stability makes the Student-t approach more reliable for production deployment where consistent performance is essential.

While both models achieve excellent performance exceeding 98.5% accuracy, Student-t embeddings demonstrate consistent advantages across classification and clustering metrics. The improvements are modest but meaningful, particularly for applications requiring robust clustering performance. The Student-t model's superior training stability, combined with its clustering performance advantages, makes it the better choice for this task, especially in scenarios where both classification accuracy and embedding quality for downstream clustering tasks are important considerations.

III. I-Con: Unifying These Methods Via a Common, Information-Theoretic Loss

Thanks to a new paper we can connect all these methods with a common thread. The Information Contrastive Learning (I-Con) framework introduces a unified, information-theoretic approach to representation learning that bridges multiple distinct machine learning methods under one comprehensive mathematical umbrella.1 I-Con’s central insight lies in generalizing diverse methods spanning dimensionality reduction, contrastive learning, clustering, and supervised learning as special cases of an optimization problem that minimizes the Kullback–Leibler (KL) divergence between two conditional distributions: a fixed supervisory distribution representing domain-specific constraints or prior knowledge, and a learned distribution representing latent embeddings produced by neural networks or embedding models. This powerful perspective clarifies relationships between seemingly disparate techniques and provides a versatile framework for deriving novel methods across traditional methodological boundaries.

Formally, I-Con defines its core objective through an elegant and versatile loss function, expressed mathematically as:

In this formulation, represents a predefined supervisory distribution encoding neighborhood structure, constraints, or desired relationships between data points. Conversely, denotes the conditional distribution learned by the model, typically parameterized by deep neural networks or embeddings.

A diagram of a cluster of data

AI-generated content may be incorrect.

*Figure 15: The Overview of the I-Con framework. (a) Shows the alignment of learned and supervisory distributions, (b) Common distribution families in I-Con’s formulation*

*Image Credit: Alshammari, et al., 2025*

The critical strength of I-Con is its ability to unify numerous existing methods by interpreting them as special cases arising from particular choices of neighborhood distributions. Techniques such as t-SNE and PCA naturally emerge within the I-Con framework when neighborhoods are defined using Gaussian kernels or uniform distributions over local neighborhoods, offering a deep information-theoretic interpretation of classical dimensionality reduction methods. Popular contrastive learning methods like InfoNCE, SimCLR, VICReg, and triplet loss also fit into the framework by selecting kernel types (e.g., Gaussian or Student-t distributions) and discrete sets of data augmentations to structure neighborhoods, thus clarifying implicit distributional assumptions regarding similarity and dissimilarity. Similarly, classical clustering methods, including k-Means and spectral clustering, can be derived from cluster-based neighborhood distributions, highlighting their theoretical connections to contrastive learning. Furthermore, supervised learning methods such as cross-entropy and harmonic loss naturally appear within I-Con when neighborhoods are directly defined by discrete labels or learned class prototypes, thereby bridging supervised and unsupervised representation learning paradigms. I found it easier to approach these methods from the lens of I-Con, to understand exactly what they are doing.

A chart with different colors and text

AI-generated content may be incorrect.

*Figure 16: The authors show that the ML methods unified by I-Con fit within a “periodic table” of the learned representation versus the supervisory signal.*

*Image Credit: Alshammari, et al., 2025*

To explicitly illustrate I-Con’s unifying power, I examine several notable examples. The dimensionality reduction method t-SNE arises within I-Con by employing Gaussian-based distributions to define neighborhoods both in the original high-dimensional and reduced low-dimensional embedding spaces. The contrastive learning method SimCLR defines neighborhoods through data augmentation pairs and Gaussian kernels in the learned representation space, thereby promoting invariance to augmentation transformations. Additionally, the classical clustering approach k-Means emerges by using Gaussian kernels in the original data space coupled with uniform distributions determined by cluster assignments in the learned space.

A white sheet with black and white text

AI-generated content may be incorrect.

*Table 2: The decomposition of popular methods into choices for and*

A crucial advantage of the I-Con framework lies in its capacity to generate novel representation learning methods through cross-domain insights. For example, data augmentation strategies and kernel definitions developed initially for contrastive learning can enhance clustering methods significantly, resulting in state-of-the-art performance in challenging unsupervised classification tasks such as ImageNet-1K.

I-Con also introduces innovative debiasing techniques to address practical challenges like overconfidence and mismatched supervisory signals. A uniform debiasing approach adds a uniform probability component to the supervisory distribution, mathematically represented as:

This method reduces overly confident neighborhood assignments. Another approach, neighbor propagation debiasing, expands supervisory neighborhood structures by performing random walks on nearest neighbor graphs, mathematically expressed as:

Their empirical validation of a method that comes from their work on the I-Con framework demonstrated its robustness and effectiveness across benchmark datasets including ImageNet-1K, CIFAR-100, and STL-10. “Debiased InfoNCE clustering” notably set new performance benchmarks, achieving significant improvements such as a +7.8% increase in Hungarian accuracy. Additionally, debiasing methods enhanced generalization in out-of-distribution tasks and significantly improved probability calibration, effectively addressing common deep learning model issues such as overconfidence.

Extensive ablation studies further confirmed the efficacy and stability of proposed debiasing strategies, recommending optimal parameter choices ( to ) and short-length neighbor propagation to achieve peak performance across diverse model architectures and sizes. The comprehensive theoretical foundation and demonstrated empirical success of I-Con (through InfoNCE Clustering) highlight its significant impact on representation learning research. By systematically unifying diverse methodologies, I-Con advances theoretical understanding, promotes method transferability, and supports the creation of innovative, state-of-the-art representation learning approaches. Now that we have previewed their paper, we can look at the method that comes out of their research, called InfoNCE Clustering.

1. Unified Methods: InfoNCE Clustering

Finally, we can appraise InfoNCE Clustering, as introduced in the I‑Con paper under “Unified Methods.” This approach integrates contrastive representation learning and clustering by explicitly aligning **two** conditional distributions in its InfoNCE loss:

1. A **learned distribution** q(j∣i), defined as the shared cluster likelihood based on soft cluster assignments.
2. A **supervisory distribution** p(j∣i), constructed from augmentation, k‑NN, neighbor propagation, and uniform smoothing (debiased via parameter α).

Unlike standard supervised contrastive losses, InfoNCE Clustering directly formulates q(j∣i) from cluster probabilities within a single, end‑to‑end learning loop.

*Mathematical Formulation*

Let an encoder **fθ**:RD→Rd map each sample **xi**to a normalized embedding:  
  
Define soft cluster assignments **ϕic** for c ∈{1,…,K}. Then:

* **Learned conditional distribution**:
* **Supervisory distribution**, built from sample neighborhoods:
  1. Start with positive relationships via augmentations or k‑NN (possibly propagated via random-walk:
  2. Apply uniform debiasing:
  3. Normalize:

The loss is optimized jointly for encoder and clustering parameters until convergence. The paper provides full proofs showing how methods like DeepCluster v2 or InfoCluster arise by specific choices of p, ϕ, or propagation mechanisms.

*When and Why to Use InfoNCE Clustering*

* **Joint representation and clustering**: Single-stage training eliminates separate encoder/clustering pipelines.
* **Large-scale unlabeled data**: Leverages natural data structure via emergent cluster-based positives.
* **Efficient negatives**: All off-cluster samples act as negatives—no external memory needed.
* **Robustness to label noise**: Averaged cluster-gradient reduces sensitivity to misassignments.
* **Semantic transferability**: Clusters boost retrieval, few-shot learning, anomaly detection.
* **Architecture-agnostic**: Compatible with CNNs, Transformers, GNNs—minimal architecture tweaks.

*Example Applications of InfoNCE Clustering*

InfoNCE Clustering is brand new, but some possible applications stand out that are similar or are already using the InfoNCE loss. For computer vision, in frameworks like SimCLR, MoCo, SwAV, InfoNCE yields semantic embeddings on ImageNet‑100 and CIFAR‑100, enabling downstream tasks without labels. In speech/audio classification, methods like HuBERT and wav2vec 2.0 use InfoNCE-based latent quantization to discover phoneme-like clusters with minimal supervision. For graph data GraphCL and GRACE employ InfoNCE-style clustering (with augmentation + neighbor signals) to uncover meaningful node clusters in citation datasets like Cora and PubMed, enabling community detection and unsupervised classification. Over time we should see InfoNCE Clustering finding a wide reception.

*Demonstration of InfoNCE Clustering on ImageNet-100*

I implemented an unsupervised clustering approach from the I-Con (Information Contrastive Learning) framework following the description in the paper. A detailed breakdown of the code is provided in Appendix D. In I-Con, the goal is to align a learned distribution (derived from model representations or cluster assignments) with a fixed supervisory distribution (derived from some notion of neighborhood or similarity). Here, I remade InfoNCE (contrastive) clustering, which uses InfoNCE loss with *debiased positive sampling*, to group images without labels. The approach used a pre-trained ViT-S/16 (Vision Transformer, small) from DINO, similar to the paper for the feature extractor for ImageNet-100 images, ensuring a strong representation baseline. This is different from the ResNet-18 backbone I have used before, because I thought it was important to use a similar feature extractor that was used in the I-Con paper.

Then, I constructed a supervisory neighborhood for each image consisting of **(a)** an augmented view of the same image, **(b)** its nearest neighbors in feature space, and **(c)** one-hop neighbors (neighbors of those neighbors, i.e. a length-1 walk on the KNN graph). This provides local *and* slightly expanded connectivity in the data graph. Next, is to incorporate a debiasing uniform mixture into the target distribution to prevent overconfidence. Specifically, is to mix the neighborhood distribution with a uniform distribution over “negative” samples:

 where is the number of neighbors (local neighborhood size) and is a tunable mixing factor. I set = 0.6 as given, but this is only one of the values used in the paper. This assigns a small probability to each non-neighbor, analogous to label smoothing, which mitigates overconfident assignments by ensuring even negatives get some weight. Define the learned distribution  based on shared cluster membership: as the model learns to cluster, two images belonging to the same cluster will have high 𝑞𝜙(𝑗∣𝑖) . In practice, I use a linear “cluster head” classifier on top of the ViT features to produce cluster assignments (soft labels). If an image and are assigned to the same cluster then   is high. Formally, *"* is the probability (soft assignment) of being in cluster c, I can define:  This formula (a soft version of *shared cluster membership*) essentially gives $ if i and j belong to the same cluster C, and is small if they are in different clusters. I can thus call this their “shared cluster likelihood” neighborhood, analogous to using *k-means cluster membership* as a neighborhood function They use a contrastive loss (InfoNCE) to train the cluster assignments. The standard InfoNCE assumes one positive pair per anchor (e.g. an image and its augmentation) and treats all other images in the batch as negatives. They extend this to multiple positives (augmentations + neighbors) and apply the uniform debiasing mix. By aligning to via KL-divergence minimization (which is implemented as a contrastive InfoNCE loss), the model learns to group images that are augmentations or nearest neighbors in feature space into the same cluster, while the uniform smoothing prevents trivial collapsed solutions. The model produces a similarity score (or probability) for every pair (i, j) using the cluster assignments. I use PyTorch to calculate the loss. I concatenate all images in a batch (anchors, their augmentations, and neighbors) and compute their feature embeddings. I then compute a cosine similarity matrix among all pairs. The InfoNCE loss is a cross-entropy (or KL divergence) aligning to for each *i*. The result is the Debiased InfoNCE loss I need to minimize. Let’s first look at the Hungarian accuracy results from the paper:

A table with numbers and symbols

AI-generated content may be incorrect.

*Table 3: The authors’ SOTA results for Debiased InfoNCE Clustering*

In my experimental evaluation, Debiased InfoNCE clustering was compared against SCAN clustering as a baseline method. The comparison utilized a subset of 7000 images drawn from six distinct classes of the ImageNet-100 dataset. To align with this class distribution, the number of clusters was explicitly set to six for both methods. Due to GPU limitations, the batch size was constrained to 1024, one-fourth of the authors initial size. However, visually this was an elbow in their graph of results (Figure 19).

Initial results using a debiasing factor (α) of 0.6 yielded suboptimal clustering performance on my implemented Debiased InfoNCE method (Accuracy: 0.1866, NMI: 0.0020) compared to SCAN (Accuracy: 0.2567, NMI: 0.0401) which may have been due to differences in dataset, batch size, sample size and class distribution. However, subsequent parameter tuning to reduce α to 0.1 and training each method for only 10 epochs significantly enhanced the results. Under these revised conditions, Debiased InfoNCE clustering achieved superior performance relative to the baseline (Accuracy: 0.4569, NMI: 0.3400 vs. SCAN Accuracy: 0.4453, NMI: 0.2100), as confirmed by quantitative analysis.

Furthermore, accuracy metrics alone may not fully capture clustering quality. Examination of the contingency matrices revealed that InfoNCE clustering A screenshot of a graph

AI-generated content may be incorrect. Figure 17: Debiased InfoNCE clustering outperforms SCAN on a sliver of ImageNet-100

exhibited significantly higher counts along the main diagonal, indicating stronger

alignment between predicted clusters and ground truth labels. Notably, Debiased InfoNCE demonstrated particularly effective clustering for classes 2 and 4, in stark contrast to SCAN's relatively poor performance on these same classes. Again, given limitations, the aim was not to completely reconstruct their result as it was to demonstrate the superiority of InfoNCE Clustering over at least one baseline. The code is light, more baselines can be added. Readers are encouraged to inspect and experiment with the code to draw their own conclusions.

A screenshot of a computer screen

AI-generated content may be incorrect.

*Figure 18: Contingency Matrix for InfoNCE Clustering and SCAN on 7000 samples in 6 categories*

Visual inspection via t-SNE embedding further corroborated the quantitative findings. Debiased InfoNCE clusters appeared tightly grouped with clear separation, except for minor overlap observed in a central cluster (visualized in purple). By contrast, SCAN clusters exhibited greater dispersion and overlap, particularly evident in the central pink cluster. This visual discrepancy underscores the improved coherence and separability achieved by InfoNCE clustering in comparison to the SCAN baseline. I was more satisfied with these results, although the authors’ finding about alpha values was substantially different than mine, given the above factors, as shown in the below figure.

A graph with a line and a point

AI-generated content may be incorrect.

Figure 19: The I-Con authors performance assessment of InfoNCE Clustering on ImageNet-1K.

A close-up of a map

AI-generated content may be incorrect.

Figure 20: t-SNE of both methods on the given dataset

IV. Comparative Analysis of the 10 Methods

At this point, the only thing remaining is to test all these models on a common benchmark for an apples-to-apples comparison of performance and explainability. But what benchmark to use? Is there a common image classification benchmark that is appropriate? Fréchet inception distance (FID) comes to mind. I decided not to rely on FID, for one. A single number (FID, IS, etc.) hides whether a method separates classes well but over-clusters, or vice-versa Also, FID only tells us how close a *generator* mimics the ImageNet-100 distribution, so it ignores what our discriminative / embedding models actually *learn* about the semantics of the images, and it is statistically biased even for generation tasks. So instead I decided to treat every method as a *representation learner* that outputs an embedding for each image, then scores those embeddings with three orthogonal, well-studied families of metrics that already have reference implementations in scikit-learn and torchmetrics. This way each method is scored and compared according the metric most relevant to the approach:

*The 10 methods fall into 3 groups by most relevant metric*

|  |  |  |  |
| --- | --- | --- | --- |
| What it measures | Metric(s) | Why it provides coverage | Relevant Methods |
| Discriminative signal(“How linearly separable are the classes once the model is frozen?”) | • *Linear-probe* Top-1 / Top-5 accuracy  • 1-NN accuracy | Works for supervised, self-sup, contrastive, graph, and even GMM (use posterior means), direct comparison %’s | Harmonic Loss  SimCLR  VICReg  Node2Vec |
| Unsupervised grouping quality | • Hungarian Clustering Accuracy • Normalized Mutual Information (NMI) | Lets you score the clustering family on its own turf | DCD  GMMs  InfoNCE Clustering |
| Neighborhood / retrieval structure | • Recall@1 / mAP@10 on a k-NN retrieval task | Gives the contrastive and multimodal methods a fair test of what they optimize, yet still meaningful for others. | t-SNE  CLIP  t-SimCLR |

Group 1 is all about how linearly separable the frozen features are. Linear-probe Top-1 and Top-5 work by freezing the encoder, training a linear classifier such as multinomial logistic regression on training embeddings only, and then reporting Top-1 and Top-5 accuracy on the validation set. This measures global linear separability: if a simple set of hyperplanes performs well, the representation is good for downstream classification with minimal task-specific learning. With only six classes, Top-5 can saturate, so Top-1 carries more discriminative power. Good practice here is to L2-normalize features (or standardize), tune the regularization C using training data only, and avoid any data leakage. The complementary metric is 1-nearest-neighbor accuracy: build a memory bank of training embeddings with labels and classify each validation example by the label of its single nearest training neighbor, typically using cosine distance on L2-normalized features. This captures local neighborhood purity: if same-class points cluster tightly and dominate each other’s nearest neighbors, 1-NN will be high even if global linear separability is imperfect. Taken together, linear-probe tests global linear separability, while 1-NN tests local structure; when they diverge, you learn whether the embedding organizes classes more globally or locally.

*Group 1*

The first method to look at is Harmonic Loss for neural network classification, versus a Cross-Entropy (CE) loss. On the balanced six-class setting, Cross-Entropy produced clearly tighter clusters than Harmonic Loss. The t-SNE visuals match the numbers: average silhouette was about 0.54 for CE versus 0.16 for Harmonic, which reflects much stronger separation under CE. That translated directly to downstream metrics—Top-1 accuracy, F1, balanced accuracy, and 1-NN all favored CE (≈0.91 vs 0.76 on Top-1). The linear-probe gap was small (0.927 vs 0.903), which tells me Harmonic’s backbone retains class information but arranges it with weaker margins and neighborhoods. Training time was essentially the same.

Harmonic Loss is designed for *severe* imbalance, which was tough to create with small data sizes. Under imbalance, (and thus needing considerably more data) the picture should shift. But the small outperformance I created on a tiny imbalanced subset of data during the demonstration could not be repeated with a “vanilla” implementation (or even an improved) of the Harmonic Loss function. I was experiencing vanishing gradients, so I dropped temperature to 0.5 and increased the learning rate. That didn’t fix it. All in all throughout my extensive experimentation with Harmonic Loss optimization, I encountered numerous implementation and configuration challenges that required systematic debugging and refinement. My journey began with fundamental numerical stability issues where the loss remained stuck around 2.9 and showed minimal improvement during training. I discovered that the epsilon handling was corrupting actual probabilities by adding epsilon directly to probability values rather than using proper clamping operations. Additionally, a scale factor of 0.7 was inadvertently reducing gradients by 30%, severely hampering the optimization process. Temperature scaling proved to be insufficianet. I started with a temperature of 2.0, which made predictions too uniform and resulted in very low confidence scores with true class probabilities averaging only 4.17%. Through experimentation, I reduced the temperature first to 1.0 for standard softmax behavior, then further to 0.5 to create sharper predictions. This final temperature setting dramatically improved the confidence range, achieving maximum confidences of up to 48% compared to the previous 8%, indicating that the model was finally learning to make decisive predictions, but accuracy was still dismal.

Learning rate optimization revealed a fundamental mismatch between the two loss functions. Initially, both models used the same learning rate of 0.001, but monitoring gradient norms showed that Harmonic Loss consistently produced much smaller gradients around 0.3-0.8 compared to Cross-Entropy's 1.0. This indicated vanishing gradient problems rather than the initially suspected exploding gradients. I systematically increased the learning rate for Harmonic Loss, trying values of 0.002 and 0.005, eventually settling on 0.001 for Harmonic Loss while reducing Cross-Entropy to 0.0001. I also completely removed weight decay, which had been overly restrictive at 1e-4 and was preventing proper learning.

Gradient clipping strategy required a complete overhaul when I realized that Cross-Entropy needed clipping due to exploding gradients while Harmonic Loss needed the opposite treatment. I implemented conditional gradient clipping by adding a model name parameter to the training function, allowing me to clip only Cross-Entropy gradients while letting Harmonic Loss gradients flow freely. This change was still not enough to make the performance improve meaningfully.

Data configuration experiments focused heavily on batch size and imbalance patterns. I increased batch size from 32 to 128 to ensure better class representation per batch, which is particularly important for imbalanced datasets where smaller batches might contain no minority class samples. The imbalance pattern evolved from exponential (leaving some classes almost completely dry) to simple step functions with problematic threshold jumps to smooth geometric decay using formulas like 0.8 raised to the class index power. I also experimented with imbalance severity, moving from mild 2.6:1 ratios that were insufficient to demonstrate Harmonic Loss benefits to severe 92.6:1 ratios that created the extreme imbalance conditions where Harmonic Loss *should* theoretically excel.

Class count optimization became a significant consideration as I progressed from 10 classes with mild imbalance to 25 classes with severe imbalance. While the severe imbalance was beneficial, having 25 classes meant that nearly half of them had very few samples, with some classes having only 10-20 samples each. This created challenges for the non-target probability calculation since the harmonic mean depends on averaging over many classes with potentially unstable probability estimates.

Throughout the optimization process, I implemented comprehensive debugging and monitoring systems to track probability distributions, gradient norms, and loss components. I added verification checks to ensure probability sums equaled 1.0 and monitored the actual values of true class probabilities and harmonic means. This instrumentation revealed critical insights about how temperature scaling affected confidence distributions and how different implementations produced different numerical behaviors, but to no avail.

I explored multiple implementation variants of the harmonic mean calculation, starting with the direct mathematical formula involving reciprocals, then moving to more numerically stable versions using direct multiplication and division, and finally experimenting with log-space arithmetic for maximum stability, and then…back to reciprocals which I left in the notebook. Each variant had different numerical properties and convergence characteristics, with the simpler implementations often performing better than the more complex ones. Looking back, there was just too many dials to turn.

A diagram of green dots

AI-generated content may be incorrect. A diagram of a graph

AI-generated content may be incorrect.

*Figure 21: t-SNE of CE and HL showing prediction results*

A group of graphs with different colored dots

AI-generated content may be incorrect.

*Figure 22: t-SNE shows Harmonic Loss performs poorly versus Cross-Entropy in the case of balanced classes*

Despite all these optimizations, my final results showed that Harmonic Loss achieved only 50.17% F1-Macro compared to Cross-Entropy's 85.46%, representing a substantial 35.29% performance gap. The most impactful improvements came from temperature reduction to 0.5, implementing differential learning rates and gradient clipping strategies, and increasing batch size to 128. However, the persistent underperformance suggests that either the dataset characteristics don't align well with Harmonic Loss assumptions, or my implementation still contains subtle numerical issues, or the number of classes and extreme minority class distributions create fundamental challenges for the harmonic mean calculation that underpins this loss function. Readers are welcome to tackle the code and try different datasets, because I am fairly sure the implementation is correct.

Practically, I’d pick CE for few, balanced classes where crisp cluster separation and maximum accuracy matter most. As the number of classes grows or the distribution skews, Harmonic becomes attractive for fairness and minority recall. A hybrid can work well too—pretraining with CE, then switching to or blending in Harmonic, or training a CE head on a Harmonic backbone to retain minority gains while recovering accuracy. I didn’t try those approaches because my exposition is of “vanilla” Harmonic Loss.

Next in group 1 is SimCLR taken versus a rotation baseline. SimCLR wins across every discriminative metric that matters on this six-class slice. On validation it reaches 67.7% Top-1, 99.3% Top-5, 47.0% 1-NN, 0.672 macro-F1 and 0.677 balanced accuracy, while the rotation baseline sits at 24.3% Top-1, 86.7% Top-5, 17.7% 1-NN, 0.191 macro-F1 and 0.243 balanced accuracy. With six classes Top-5 is near-ceiling and thus weakly informative; the separation in Top-1, macro-F1 and balanced accuracy shows that SimCLR produces features that are both globally separable and more label-consistent locally. The t-SNE plots match the numbers: SimCLR forms partially separated clusters with visible substructure, whereas the rotation baseline arranges points along a loose diagonal manifold with heavy class mixing. The confusion matrices tell the same story. SimCLR’s mass is concentrated on the diagonal with moderate bleed into a few confusable pairs; the rotation baseline over-predicts specific classes (notably class 4) and shows widespread off-diagonal mass, which is consistent with a representation that encodes rotation cues and texture but weak class semantics.

Per-class results underline these patterns. SimCLR is strongest on class 4 (precision 0.768, recall 0.860, F1 0.811) and weakest on class 2 (F1 ≈ 0.505), suggesting a few visually similar categories that the linear head still struggles to disentangle. Class 5 shows high precision but lower recall (0.839 vs 0.520), implying conservative decision boundaries for that class. The rotation model has uneven behavior with recall spikes on class 4 and collapse on others, a classic symptom of a pretext task that does not align with the downstream labels. Train vs validation also looks healthy for SimCLR: 72.6% train Top-1 versus 67.7% validation does not suggest severe overfitting, whereas the rotation model is simply underpowered on both splits. Neighborhood quality mirrors this: SimCLR’s 1-NN at 47% is well above the rotation baseline’s 17.7%, yet still below its own linear-probe-like Top-1, meaning neighborhoods are reasonably clean but class manifolds remain curved enough that a linear head helps. Computing cost is the main trade-off: SimCLR took about 2,434 seconds versus 1,770 seconds for rotation—roughly 37% more time—but the performance gain per second is decisively in SimCLR’s favor.

If more classes are added, Top-1 should drop for both methods as decision boundaries become finer-grained, and Top-5 will become more discriminative rather than saturating. SimCLR is likely to degrade gracefully: more classes increase semantic diversity and, with a fixed batch size, also increase the effective hardness of negatives, which typically preserves or improves the structure of the embedding if optimization keeps up; to hold the line you may need larger batches or a harder negative mining strategy to maintain InfoNCE pressure. The rotation baseline should erode more quickly, so 1-NN will fall further and the confusion matrix will show broader off-diagonal spread. In t-SNE, expect SimCLR clusters to remain evident but more interleaved. Macro-F1 and balanced accuracy will increasingly favor SimCLR because class imbalance and fine-grained confusions penalize representations that lack label-aligned geometry.

A screenshot of a graph

AI-generated content may be incorrect.A screenshot of a graph

AI-generated content may be incorrect.

A chart with many colored dots

AI-generated content may be incorrect.

*Figure 23: SimCLR, Rotation baseline and True Labels t-SNE*

Next is VicREG. VICReg hit 94.67% Top-1 and 100% Top-5 on a linear probe with only three classes ([0, 1, 16]) and 150 validation images. That lands within ~2.0 percentage points of the fully-supervised baseline (96.67% Top-1; Top-5 is saturated at 100% because there are only three classes). Hitting >90% with zero labels confirms the learned backbone features are linearly separable and competitive with supervised training in this small-class regime. The 1-NN accuracy at 22.22% is the outlier. That metric is much more sensitive to local neighborhood geometry, feature scaling, and sample count than a trained linear head. In my setup, features are high-dimensional, not L2-normalized, evaluation uses a tiny val set (n=150) with an ad-hoc 70/30 split, and k=1 with cosine distance. Any of those can depress 1-NN. In practice for SSL, a weighted k-NN (e.g., k=20 with temperature weighting) on L2-normalized features—optionally after a quick PCA/whitening to 64–128 dims—tracks “cluster tightness” far better. Also test location matters: backbone features vs. projector output (some pipelines evaluate on backbone only, others on the projector’s first FC; results can differ a lot).

Our VICReg loss components are informative. Invariance is ~0.001 by epoch 2 (pairs are tightly aligned), covariance is small (~0.001–0.005), but the variance penalty stays high (~1.93–1.95). That pattern means the projector space trends toward low per-dimension standard deviation (incipient collapse in the projection head), while the backbone still learns separable features (hence strong linear probe). I weighted the objective as (ν·inv + λ·var + μ·cov) with ν=25, λ=1, μ=1, so the invariance term dominates. The original VICReg recipe keeps the variance term strong (often λ≈25, μ≈1, ν≈25) to actively push std above γ and avoid collapse in the projector. If we want healthier geometry—and likely better non-parametric metrics like 1-NN—raise λ (variance weight), consider γ>1.0, and ensure BatchNorm/feature std don’t get washed out by very strong crops/blur early on.

A screenshot of a graph

AI-generated content may be incorrect.

*Figure 24: VicREG t-SNE versus True Labels*

What VICReg is good for: label-efficient pretraining, transfer to new tasks/domains, robustness to heavy augmentations, and linear-probe-friendly representations. It is especially useful when you can’t afford labels or expect distribution shift; the invariance–variance–covariance triad encourages view-invariance without collapsing the representation and de-correlates dimensions, which tends to help downstream linear heads. For retrieval, few-shot, or clustering, add the normalization/PCA/weighted-kNN eval I mentioned to better reflect local structure.

A diagram of different colored dots

AI-generated content may be incorrect.A diagram of different colored dots

AI-generated content may be incorrect.

*Figure 25: Node2Vec versus True Labels t-SNE*

Let’s look at Node2Vec. The evaluation reveals a nuanced performance profile where Node2Vec demonstrates strong local structure preservation while facing challenges with global linear separability compared to the PCA baseline. The baseline achieved superior performance across most classification metrics, with a linear probe top-1 accuracy of 99.1% versus Node2Vec's 96.4%, suggesting that the graph-based embedding transformation introduced some complexity that reduced the global linear separability of the feature space.

A comparison of a graph

AI-generated content may be incorrect.

*Figure 26: Node2Vec performance versus baseline*

However, Node2Vec's performance pattern indicates successful capture of meaningful local neighborhood relationships. The algorithm achieved a 1-NN accuracy of 91.6%, marginally outperforming the baseline's 91.2%, which demonstrates that Node2Vec embeddings preserve local similarity structure effectively. This suggests that within local neighborhoods, semantically similar images cluster together appropriately, enabling accurate nearest neighbor classification despite the reduced global linear separability.

The most striking difference appears in the clustering quality metrics, where Node2Vec achieved a positive silhouette score of 0.132 while the baseline produced a negative score of -0.012. This substantial improvement indicates that Node2Vec successfully reorganized the embedding space to create more coherent, well-separated clusters that align better with the underlying class structure. The positive silhouette score suggests that points within clusters are closer to their cluster center than to neighboring cluster centers, representing a meaningful improvement in unsupervised structure discovery.

The graph construction process created a network with 130,950 edges across 10,000 nodes, resulting in a sparse but well-connected structure with an average degree of approximately 26 nodes per node. This density level appears appropriate for capturing local similarities while avoiding over-connectivity that could blur class boundaries. The k-NN graph construction with k=15 and a cosine similarity threshold of 0.3 likely created connections primarily between same-class or highly similar images, establishing the homophily necessary for effective Node2Vec learning.

Node2Vec's strength in local structure preservation combined with weaker global linear separability suggests that the random walk-based learning process successfully captured the manifold structure within classes but may have introduced nonlinear transformations that complicate simple hyperplane-based classification. The algorithm's biased random walks, controlled by the return parameter p=0.25 and in-out parameter q=4.0, favored exploration over local clustering, which typically helps capture broader structural patterns but can sacrifice some linear separability.

The near-perfect top-5 performance for both methods (99.6% for Node2Vec, 100% for baseline) indicates that both embeddings place the correct class among the top predictions with high reliability, suggesting that the fundamental class structure remains intact even when top-1 accuracy differs. This pattern is characteristic of well-behaved embeddings where classification errors tend to be among semantically related classes rather than completely unrelated ones.

The performance differential likely stems from the inherent trade-off between preserving the original feature space's linear structure and learning a new representation that emphasizes graph-based relationships. While the PCA baseline maintains the linear relationships that made the original ResNet features effective for classification, Node2Vec prioritizes structural similarity as defined by the k-NN graph, which may not perfectly align with the classification objective but provides superior clustering and local neighborhood preservation for other downstream tasks.

*Group 2*

Group two is about judging whether an unsupervised method has actually discovered the class structure that’s present in the data without being told the labels. On ImageNet-100 this matters because the dataset has many visually similar categories and substantial intra-class variation; a good clustering metric needs to be label-permutation-invariant, resistant to trivial solutions (e.g., “one giant cluster” or “one cluster per image”), and informative when classes are split or merged. Hungarian Clustering Accuracy and Normalized Mutual Information (NMI) do exactly that and, together, give a crisp, complementary picture of unsupervised quality.

Hungarian Clustering Accuracy answers a very concrete question: “If I were allowed to rename clusters after the fact, how accurate would a classifier be?” You build a contingency table between predicted clusters and ground-truth classes, then use the Hungarian algorithm to find the one-to-one mapping that maximizes total hits; accuracy is the matched count divided by N. This score is intuitive on ImageNet-100 because it’s the same scale as classification accuracy, so you can read it like Top-1 but for clusters. It also penalizes both over-segmentation and under-segmentation: if a true class is split across several clusters, only one of those clusters can be matched to it, capping the accuracy; if multiple classes are merged into one cluster, only one class can claim that cluster, so the rest become errors. With balanced subsets (like using the first 6 classes equally), it reflects instance-level purity in a way that’s easy to compare across methods such as DCD, GMMs (with argmax of posteriors), and InfoNCE Clustering.

NMI is an information-theoretic complement. It measures how much information the cluster assignments share with the true labels, normalized by the entropies so the score lies in [0, 1] and is comparable across runs and different k. Because it’s invariant to label permutations and normalized, NMI is robust when ImageNet-100 classes are imbalanced, and it discourages degenerate solutions: a single cluster has low mutual information with many classes; exploding the number of clusters doesn’t artificially inflate NMI the way “purity” does. It also reflects global consistency (are all classes represented coherently?), not just the best one-to-one relabeling. That makes it especially useful for methods like DCD and InfoNCE Clustering, where the objective can encourage fine-grained partitions; NMI will reward coherent structure across the whole label set rather than just a subset that maps well.

These two metrics are the most suitable for the unsupervised row because they directly target what clustering should achieve on ImageNet-100: label-agnostic discovery of class structure that holds up when you align clusters to classes (Hungarian accuracy) and mutual-information-level agreement that’s stable across different clusterings and class balances (NMI). Alternatives either fail one of these tests or are harder to interpret at scale. Purity is monotonic in the number of clusters and over-rewards over-segmentation; Rand/ARI can be sensitive to many tiny clusters and become less intuitive when N and k are large; raw accuracy without Hungarian matching isn’t permutation-invariant. In practice, reporting both Hungarian accuracy and NMI lets you see when a method nails clean, relabel-ready groupings (high Hungarian) and when it captures the overall class structure even if some classes are split or merged (high NMI). For our dataset, that combination fairly scores the clustering family—DCD, GMMs, and InfoNCE Clustering—on their own turf.

The DCD implemented in the notebook shows how the DCD clusters for 10 classes match up to the true labels in a t-SNE plot. The DCD I implemented is based on the Deep Embedded Clustering (DEC) method, because I had trouble getting JULE to work well. 29  I had to make other improvements too: I strengthened the clustering objective by boosting the KL term from 1.0 to 5.0 with a faster ramp-up, introduced temperature annealing to sharpen soft assignments over time, and added entropy regularization to prevent collapsed clusters; to curb over-optimization I clipped the reconstruction loss to a small floor, froze the decoder at epoch 20 of 30, enforced separation with a cluster-center regularizer, and stabilized learning with latent feature normalization, then wrapped everything in a comprehensive test-set analysis and visualization suite (including PCA/t-SNE embeddings, confusion matrices, reconstructions, and cluster diagnostics) to make failure modes and progress easy to spot. That’s a lot of improvement needed over “vanilla” implementations, but the results were improved.

A comparison of colored dots

AI-generated content may be incorrect.

*Figure 27: DEC-style DCD clusters t-SNE versus true labels*

On this run the model converged quickly and then settled into a steady regime. Using a frozen ResNet-18 feature extractor with a small projection head, k-means initialization gave the method a strong starting point, so by epoch 4 we already hit a big jump (NMI≈0.675, ARI≈0.452, HAcc≈0.518). After that, the curves flatten and drift in a narrow band: best val NMI≈0.686 at epoch 13 and final val NMI≈0.669; on the held-out test set we see NMI≈0.666, ARI≈0.472, and accuracy (Hungarian-matched) ≈0.532. The generalization gap is tiny, which is a good sign that the clustering is stable rather than overfit.

Assignment quality is also informative. Mean confidence is ~0.78 with a fairly tight spread, while mean entropy is ~0.82 with a long tail. In practice that means most samples are assigned sharply, but there’s a non-trivial slice of ambiguous points that sit near cluster boundaries (visible as the hazier regions in t-SNE). The cluster histogram and per-cluster purity back this up: One cluster (ID 1) is almost perfectly pure for class 4 (purity ≈0.996), and cluster 8 is high-purity for class 2 (≈0.857). These are “easy” classes that the representation separates cleanly. Several classes are split or entangled: e.g., class 0 is spread across clusters 0 and 2 (purities ≈0.39 and ≈0.61), class 9 dominates a very large cluster 7 (purity ≈0.36), and class 6 sits in cluster 6 with only moderate purity (~0.48). A tiny cluster (ID 4, size 11) likely captures outliers or a fine-grained subtype. In unsupervised workflows that’s actually useful—it flags potential anomalies or subclasses to inspect.

Qualitatively, the PCA/t-SNE panels show broad agreement between deep clusters and true labels: the clearly separated manifolds (e.g., the “yellow” and “green” islands) align well, whereas the twisted regions correspond to the mixed-purity clusters above. Given the frozen backbone, this plateauing behavior is expected: DEC mainly refines soft assignments and the cluster centers; with limited feature plasticity, the boundary geometry can’t move enough to resolve the stubborn confusions, so NMI/HAcc improve early and then stabilize.

What DCD is good for in practice:

* **Exploratory structure discovery & dataset audits.** The purity table and confusion heatmap quickly reveal which categories are intrinsically separable, which are overlapping, and where label noise or subclasses exist (e.g., the tiny cluster 4).
* **Label-efficiency / active learning.** High-entropy or low-confidence samples are prime candidates for human labeling; adding a few labels there should yield the biggest gains.
* **Pre-clustering for semi-supervised learning.** The confident clusters (IDs 1 and 8 here) can be treated as pseudo-labeled to bootstrap a small supervised fine-tune, while uncertain regions are down-weighted.
* **Anomaly and subclass detection.** Very small, coherent clusters often surface rare modes, near-duplicates, or corrupt data.
* **Warm starts for downstream tasks.** The cluster centers and assignments provide good initial groupings for tasks like curriculum learning, retrieval, or balanced sampling.

If I wanted to push beyond the current plateau, I’d (i) partially unfreeze later ResNet blocks or augment the projection head to increase feature plasticity, (ii) continue using the balance/entropy regularization to avoid single-cluster collapse while encouraging more uniform usage, and (iii) target the ambiguous tail with harder augmentations or a small amount of labeled guidance. But even in its current form, the DEC-based DCD gives a reliable, interpretable map of the dataset that’s immediately useful for inspection, triage, and bootstrapping downstream models.

The next method in this group is Gaussian Mixture Models. Using frozen ResNet-18 embeddings followed by standardization and PCA to 50 dimensions, the GMM selected six components by BIC and converged cleanly. The clustering quality is strong: ARI 0.824, NMI 0.879, Hungarian accuracy 0.831, and a modest but positive silhouette of 0.191. Assignments are essentially deterministic with an average maximum responsibility of 0.999 and mean entropy of 0.004, and the model’s log-likelihood sits at −91.98 with BIC 1,365,496.461 and AIC 1,310,930.225. The mixture is somewhat imbalanced—cluster sizes are 1,268, 1,291, 1,312, 1,836, 613, and 720 samples, so one component is large and two are smaller—but the t-SNE views show six compact blobs that line up closely with the six labels.

This profile means the GMM has found a near one-to-one partition of the semantic embedding space into six regions that largely correspond to the classes while keeping responsibilities crisp. The non-perfect scores and the silhouette in the high-teens point to overlaps at a few boundaries and the fact that some classes are a bit elongated or multi-modal relative to a single Gaussian, which the model approximates with ellipses.

On this dataset I would use the fitted mixture for pseudo-labels to bootstrap linear probes or deep clustering, for dataset triage using log-likelihood and entropy to surface outliers and potential mislabels, for balanced or curriculum sampling by cluster or by likelihood, and for soft supervision where responsibilities serve as calibrated soft labels. The run flagged 352 low-likelihood samples at the 5% tail; these concentrate near cluster perimeters in t-SNE and are natural candidates for inspection. I can also sample in feature space from individual components to stress-test downstream classifiers, even if those synthetic vectors are not directly viewable as images.

If I scale to more samples while keeping the same six classes, the covariance and mean estimates should stabilize and I would expect similar or slightly better alignment; BIC might even justify splitting a particularly multi-modal class into multiple components, which could lift Hungarian accuracy. With fewer samples, covariance estimation becomes noisier and BIC may under-select or over-fit depending on the setting, which can nudge ARI and NMI down. If I increase the number of classes to ten, BIC often chooses at least ten components and the metrics may soften a bit as classes crowd each other, but with good embeddings they remain competitive. Further gains are available by increasing PCA dimensionality or whitening, trying tied or diagonal covariances with a small regularizer, raising the number of GMM initializations, using a Bayesian mixture to allow automatic split-and-prune, balancing class counts before fitting, or swapping to stronger embeddings such as CLIP ViT. Compared with the raw-pixel GMM that collapsed to two components earlier, these results underscore that semantic features are what make mixture modeling work as an unsupervised grouping and data-quality tool here.

A comparison of colored dots

AI-generated content may be incorrect.

*Figure 28: GMM t-SNE and True Labels*

And now for the I-Con authors new method: as shown earlier, InfoNCE Clustering beats SCAN on both Hungarian accuracy and NMI (0.4356 vs 0.3989 and 0.2451 vs 0.2118). That’s a +3.67 pp accuracy gain (~9% relative) and +3.33 pp NMI (~16% relative). On t-SNE, the Debiased clusters are tighter and more class-coherent overall, while SCAN shows more bleed between regions.

The contingency tells the story of where the wins come from and where the method still struggles. Debiased InfoNCE discovers a very pure cluster for true₁ (pred₃ = 974), and a largely coherent block for true₃/true₄ that concentrates in pred₅ (648 and 498 respectively). SCAN’s own table (for the rows we have) shows heavier dispersion for true₀ across many columns and, while true₁ is strong, it doesn’t cleanly separate the rest. Two Debiased failure modes are visible: a “super-cluster” (pred₁) that absorbs large portions of true₀ and true₂ (622 and 810) plus spillover from other classes, and a fragmented true₅ spread across several clusters (no dominant column). Those match the t-SNE: one central region remains interleaved, and one class never really gels.

Why use InfoNCE Clustering? Aligning the learned pair distribution (from soft cluster co-membership) to a neighborhood-derived, uniformly debiased target ) gives you multi-positive contrastive learning without overconfidence. The uniform mix (controlled by α) counters popularity/hubness bias; short neighbor propagation seeds compact local manifolds; and using cluster size reweighting prevents domination by large groups. The net effect is better calibrated scores, improved separability for classes with well-formed neighborhoods, and fewer “drift” pairs that confuse SCAN-style self-labeling.

This new method is good for single-stage joint representation+clustering on large unlabeled sets; scenarios where you want semantic retrieval or few-shot transfer out of the box; settings with mild label noise or class imbalance, where calibration matters (e.g., open-world/OOD filtering and anomaly discovery). It’s architecture-agnostic (CNNs, ViTs, even GNNs) and efficient—every off-cluster sample is a negative, so you don’t need extra memory banks.

Actionable takeaways from these results and their paper:

* Keep α in the 0.6–0.8 range and use short propagation (1–2 hops). That typically sharpens boundaries without re-introducing overconfidence.
* Address the pred₁ super-cluster by increasing temperature (sharper softmax), adding a small entropy or size-balance regularizer on the cluster head, or splitting that region with a brief k-means refinement then resuming training.
* Stabilize the weak class (true₅) by class-balanced mini-batches or by seeding a few extra neighbors for its anchors.
* Report calibration (ECE) and OOD retrieval—these are areas where debiasing usually shines and will strengthen the claim.

The bottom line: Debiased InfoNCE-style clustering is giving us measurably better partitions than SCAN with more coherent geometry on t-SNE. It’s especially strong when classes are supported by clear local neighborhoods, and with a bit of tuning for the merged region and the fragmented class, it should widen its lead further while preserving the calibration benefits that motivated the debiasing in the first place.

A screenshot of a computer generated image

AI-generated content may be incorrect.

*Figure 29: InfoNCE Clustering t-SNE versus True Labels*

*Group 3*

For the third group, we can evaluate the embeddings with Recall@1 / mAP@10 on a k-NN retrieval task. Recall@1 and mAP@10 on a k-NN retrieval task refers to evaluating how well a model’s learned embedding space supports nearest neighbor search. In this setting, there is a gallery or database of embeddings and a query set of items to retrieve matches for. For each query, distances or similarities to all gallery items are computed, the results are sorted, and the top-k retrieved items are evaluated against the query’s true label or relevant set. Recall@1 measures the fraction of queries whose single most similar retrieved item is correct, so it answers the question, “If I just take the nearest neighbor, how often is it right?” mAP@10, or mean Average Precision at 10, considers the top 10 retrieved neighbors for each query, computes precision at each rank where a relevant item appears, averages these per query to get the Average Precision, and then averages across all queries. The “@10” means rankings are truncated at position 10. This is different from “vanilla” Recall@1 or mAP@10, which might be computed directly from classifier logits or in a ranking context without the constraint of nearest neighbor search. In the k-NN retrieval context, both metrics measure the quality of the embedding space’s local semantic structure rather than classifier output, which is why they are particularly relevant for methods such as contrastive and multimodal learning but still meaningful for others.

Let’s begin this group with t-SNE as presented earlier . Using the first six ImageNet-100 classes, the side-by-side plots and retrieval metrics show that t-SNE strongly outperforms PCA at preserving local neighborhoods. On the same gallery/query split (k=10, cosine), PCA reaches Recall@1 = 0.7344 and mAP@10 = 0.7853 in ~0.01 s. Its 2-D projection (after a PCA pre-step to 50D capturing ~0.73 explained variance) keeps coarse class separation, but clusters are elongated and partially intermingled; boundary regions cause nearest-neighbor errors. t-SNE obtains Recall@1 = 0.9600 and mAP@10 = 0.9579, forming six tight, well-separated clusters consistent with near-perfect top-1 and top-10 retrieval—at the cost of ~732.63 s, making PCA ≈ 70,502× faster. Perplexity sensitivity shows a broad optimum around 15 (best observed: Recall@1 = 0.980, mAP@10 = 0.980, ~75.2 s), with 30 close behind and both 5 and 50 modestly worse. In short: for this subset and objective, t-SNE yields far better neighborhood preservation, but at extreme compute cost, so it’s best for offline analysis and visualization rather than time-critical pipelines.

What is t-SNE used for? t-SNE is a nonlinear dimensionality-reduction method designed primarily for visualizing high-dimensional data. It converts pairwise similarities into probabilities—Gaussian in high-D, Student-t in low-D—and optimizes a Kullback–Leibler objective (with early-exaggeration) so that near neighbors in the original space remain near in 2D/3D. The perplexity parameter sets the effective neighborhood size; too low or too high distorts local neighborhoods. Important caveats: global geometry (between-cluster distances) is not metrically meaningful, embeddings are non-parametric (no native out-of-sample mapping), results can vary with initialization/hyperparameters, and runtime scales poorly.

Why choose it here? Our goal is to assess whether the learned representation encodes class-consistent local neighborhoods—exactly what k-NN retrieval metrics (Recall@1, mAP@10) and t-SNE target. PCA is linear and preserves maximal variance, not neighborhood identities; with six visually similar ImageNet classes, linear projections leave overlapping boundaries that harm retrieval. t-SNE, by directly optimizing local neighborhoods, exposes clean, compact clusters that align with much higher Recall@1/mAP@10. That makes it a powerful diagnostic: (1) sanity-check embedding quality, (2) spot class overlap or label noise, (3) compare runs/hyperparameters (perplexity, training recipes), and (4) communicate structure to stakeholders. We still wouldn’t deploy t-SNE inside a production retrieval loop; instead, we’d use it offline to validate that the embedding space is shaped correctly before choosing a scalable method (e.g., PCA/whitening + ANN index) for real-time inference.

A graph showing different colored dots

AI-generated content may be incorrect.

*Figure 30: Reviewing the t-SNE versus PCA clusters*

Looking at CLIP, I trained a simplified CLIP with 128-D embeddings (about 703k parameters) on a balanced 3-class slice of ImageNet-100 and reached 87.17% test accuracy in about 34 seconds over 5 epochs. The split was 2835 train, 607 val, and 608 test, with 1350 images per class overall. Train accuracy landed at 86.14% and validation at 83.36%, so the train→test gap was ≈−1.03 percentage points, which tells me the model is not overfitting and still generalizes well for the short training budget. Final loss was 2.0757 and the learned temperature settled near 4.8875.

Classification and retrieval line up perfectly: Recall@1 = 87.17% and mAP@10 = 87.17%, and separate image→text retrieval accuracy was 87.0%. Because each query has a single relevant text label, average precision effectively reduces to “did I rank the correct text first,” so mAP≈Recall@1≈accuracy is exactly what I expect. I also measured cross-modal alignment strength at 1.000, which matches the behavior I see in cosine-similarity scores and indicates a tight image–text geometry. A temperature around 4.89 sharpens the softmax over similarities, which boosts top-1 confidence; if I need calibrated probabilities, I’d add post-hoc temperature scaling.

The t-SNE visualizations make the metrics tangible. Clusters based on true labels and clusters colored by CLIP predictions largely overlap, and my “correct vs. incorrect” plot shows errors concentrated on low-density bridges between clusters, which is where cosine classifiers tend to fail. The joint embedding plot places text prototypes right on top of their image clusters rather than on a separate manifold, showing a small modality gap. The similarity matrix is strongly block-diagonal with little off-block leakage, another sign that neighborhoods are class-pure.

Zero-shot behavior is the headline. Without any task-specific fine-tuning, I get 87.17% test accuracy, 87.17% Recall@1, 87.17% mAP@10, and 87.0% image→text retrieval, which means prompts alone act as a classifier. Against baselines, I outperform a simple CNN at 75.0% and sit far above random at 31.7% (theoretical 33.3%). In practice this makes the approach ideal for prompt-based classification, cross-modal retrieval, and label-efficient expansion to new classes, while the embeddings and visuals provide clear diagnostics when mistakes do happen.

A screenshot of a graph

AI-generated content may be incorrect.

A graph showing a number of points

AI-generated content may be incorrect.

*Figures 31-33: CLIP embeddings vs. true labels, and prediction accuracy*

Lastly, we have to look at t-SimCLR. Comparing against Gaussian Clustering, I read this run as “both models are excellent, but they win on different things.” With the same ResNet-18 backbone and equal class balance, t-SimCLR edges classification and retrieval—99.33% test accuracy, Recall@1≈0.989, mAP@10≈0.989—slightly ahead of the Gaussian baseline (98.50%, 0.987, 0.983). Clustering compactness tilts the other way: your ARI is basically a wash (t-SimCLR 0.966 vs Gaussian 0.968), but the silhouette favors Gaussian (≈0.762 vs 0.687), and the per-class tightness stats show Gaussian producing consistently tighter class blobs. The t-SNE panels make that visible: t-SimCLR forms long, gently curved manifolds; Gaussian looks more spherical and compact.

Why I generally expect t-SimCLR to outperform Gaussian clustering is the heavy-tailed Student-t similarity. In the contrastive objective it down-weights “too-close” pairs and gives relatively stronger repulsion to mid-distance negatives. That combats the classic crowding problem, preserves neighborhood structure better, and is more forgiving of outliers and multi-modal classes. In practice that helps retrieval, few-shot transfer and open-set recognition because distances remain meaningful beyond a tiny local radius; you’re not forced into hyperspherical class islands the way a Gaussian/softmax geometry tends to encourage.

Why it didn’t dominate here comes down to data geometry and training conditions. I used six clean, well-separated categories with uniform balance and strong augmentations; the backbone already yields high separability (ARI≈0.927 before heads), and both heads add ≈+0.039 ARI. In that regime, a Gaussian head that implicitly favors tight, radially symmetric clusters is a near-perfect inductive bias, so it wins on compactness metrics and per-class tightness. The Student-t head, by design, doesn’t over-collapse intra-class structure; it keeps manifolds slightly elongated to preserve neighborhood relations. That pays off in retrieval and calibration, but it can reduce silhouette even while top-1/Recall@1 improve a bit.

If I changed the data, I’d expect the advantage to swing toward t-SimCLR in several situations: more classes or finer-grained categories with multi-modal appearance, heavier augmentations or domain shift that create long intra-class manifolds, real-world label noise or outliers, and imbalanced sampling where a compact-cluster bias overfits head classes. In those settings the heavy tails protect local neighborhoods and keep mid-distance negatives honest, which usually shows up as better Recall@K at larger K, stabler mAP, and stronger few-shot transfer.

A screenshot of a graph

AI-generated content may be incorrect.

*Figure 34: t-SimCLR t-SNE and prediction accuracy*

There are also straightforward knobs that would likely close the compactness gap without giving up the t-SimCLR gains: tune the Student-t degrees-of-freedom and t emperature together (lower temperature or slightly larger ν tightens clusters), add a lightcenter/variance regularizer or supervised-contrastive term to encourage within-class compactness, mine harder negatives, and let a few more backbone blocks update instead of staying too frozen so the geometry can adapt. Given how close these numbers already are, those tweaks should push t-SimCLR to “win on both fronts” while keeping its robustness advantages for messier data.

V. Conclusion and Recommendations

The comprehensive exploration conducted in this survey demonstrates how the diverse landscape of contemporary machine learning methods can be effectively synthesized through the powerful lens of Information Contrastive Learning (I-Con). Throughout the examination of ten distinct representation learning techniques, including dimensionality reduction, clustering, supervised learning, contrastive learning, probabilistic modeling, graph-based learning, self-supervised learning, multimodal learning, neighborhood-based reinforcement learning, and unified hybrid methods, a clear pattern emerges: despite their varied motivations, mathematical frameworks, and domains of application, all these methods fundamentally address similar challenges of representation alignment, robustness, and semantic preservation that can be viewed as the intersection of a supervisory signal with the type of representation learned.

This survey illustrated each chosen method by providing concise descriptions, mathematical definitions, motivations, illustrative applications, and practical demonstrations on the ImageNet-100 benchmark. This structured approach highlighted not only the individual merits of each method but also facilitated direct comparisons, clarifying their strengths and limitations, which should be useful to practitioners. For instance, while methods like t-SNE excel in preserving local structures in high-dimensional data, techniques such as SimCLR leverage data augmentation to achieve representation invariance, highlighting critical methodological differences shaped by their underlying mathematical objectives. Similarly, the Harmonic Loss classifier demonstrates how changing the fundamental metric—from arithmetic to harmonic mean—can significantly alter model performance, particularly in addressing issues like class imbalance.

The survey culminated in a brief view of the Information Contrastive Learning framework, providing a unifying theoretical basis that aligns diverse representation learning methods within a common information-theoretic framework. By conceptualizing various learning paradigms as special instances of minimizing the Kullback–Leibler divergence between supervisory and learned conditional distributions, the I-Con framework illuminates implicit relationships between methods previously perceived as unrelated. This unification not only enhances our theoretical understanding but also encourages innovative cross-pollination of techniques across methodological boundaries, opening avenues for novel algorithmic developments.

Another impactful contribution of the I-Con framework lies in its systematic debiasing strategies, notably uniform debiasing and neighbor propagation debiasing. These techniques address practical issues like overconfidence and mismatched supervisory signals, significantly improving model robustness and generalization capabilities. The empirical validation provided by the I-Con authors underscores the effectiveness of these strategies, with debiased models setting new state-of-the-art benchmarks and demonstrating enhanced calibration and generalization on challenging datasets such as ImageNet-1K, and CIFAR-100 for representation quality.

Several key lessons emerged from this survey. Firstly, the necessity of methodologically flexible frameworks like I-Con becomes evident, as machine learning methods increasingly overlap and interconnect. Such frameworks allow for rapid innovation by revealing and exploiting commonalities between techniques. Secondly, the critical importance of addressing bias and overconfidence in models is highlighted, emphasizing that effective representation learning extends beyond pure predictive accuracy to include robustness and generalizability. Thirdly, the empirical demonstrations reinforce the value of standardized benchmarks and protocols for evaluating and comparing models, enabling clearer insights into the relative advantages and constraints of various methods.

*Comparative Synthesis and Recommendations*

Bringing the ten methods together on a single dataset and rubric made their trade-offs concrete. I’ll close by comparing them along the three metric families I used (discriminative separability, unsupervised grouping quality, and neighborhood/retrieval structure), then translate those observations into a short, method-selection playbook grounded in the I-Con view.

*Comparative synthesis*

**Discriminative separability (linear-probe Top-1/Top-5, 1-NN).** On balanced classes, Cross-Entropy (CE) produced clearly cleaner margins and cluster geometry than Harmonic Loss (HL). The t-SNE visuals and silhouette scores echoed the gap, and the linear-probe edge (CE > HL) indicates that, when labels are plentiful and balanced, the classic logit geometry still dominates. HL’s promise is in extreme imbalance and head–tail fairness. My implementation experience also surfaced a practical point: HL exposes many more stability knobs (temperature, epsilon handling, prototype norms), which is both a strength (control) and a liability (brittleness) compared to CE.

Among self-supervised approaches, SimCLR beat the rotation baseline across the metrics that matter (Top-1, macro-F1, balanced accuracy), confirming that invariances learned from strong augmentation pairs translate into globally separable features. VICReg landed within a couple of points of a fully supervised baseline on a compact task, but its low 1-NN under my settings reminded me that non-contrastive SSL needs care in projector weighting and variance thresholds if I want neighborhood purity to track linear separability. CLIP’s dual-encoder training gave me strong zero-/few-shot behavior with efficient retrieval, and—unlike pure vision SSL—its linear probe can be driven by language prompts rather than labels. Finally, Node2Vec improved local neighborhood quality and clustering silhouette over a PCA baseline but sacrificed a sliver of global linear separability—an expected trade-off when I optimize for graph contextuality rather than hyperplane margins.

**Unsupervised grouping (Hungarian accuracy, NMI).** Deep Cluster Discovery in DEC form (with stronger KL, entropy regularization, separation, and temperature annealing) generalized well and achieved solid NMI/HAcc once failure modes like collapse were handled. GMMs, by design, returned soft memberships and clean likelihoods—and often fewer components than semantic classes—which is exactly what a density model should do when classes are visually entangled: reveal *visual modes* rather than human labels. InfoNCE Clustering, once debiasing and neighborhood design were tuned, overtook SCAN on my slice, and the contingency diagonals plus t-SNE agreed: tighter, more coherent clusters with fewer cross-class leaks. Read through I-Con, the pattern is simple: the better my supervisory neighborhoods reflect *true* local structure (augmentations + kNN + short propagation) and the more I temper overconfidence (uniform smoothing with ), the more consistently the learned produces relabel-ready clusters.

**Neighborhood / retrieval (Recall@1, mAP@10, t-SNE).** Non-linear, heavy-tailed objectives (t-SNE, t-SimCLR) preserved local structure far better than linear PCA at the cost of compute. That shows up identically in retrieval: higher Recall@1 and mAP@10 mean nearest neighbors are semantically meaningful, not just variance-aligned. CLIP inherits the same behavior cross-modally: text queries retrieve images with high precision because its neighborhoods are defined by paired captions, not Euclidean variance. t-SimCLR’s Student-t kernel helped fill the “mid-range gap” between strict neighbors and obvious non-neighbors, improving cluster fidelity (ARI/NMI) while sometimes giving up a touch of silhouette vs. Gaussians—again the expected geometry trade-off.

Cross-cutting lessons (what really moved the needle)

1. **Neighborhoods beat labels when labels are scarce.** SimCLR/InfoNCE and VICReg both closed most of the gap to supervised baselines; the differences mostly hinged on *how* I built : augmentations only (SimCLR), augmentations + kNN (+ propagation) with debiasing (InfoNCE Clustering), or paired text (CLIP).
2. **Heavy tails help mid-scale structure.** Student-t kernels (t-SNE, t-SimCLR, DEC-style soft assignments) reduced crowding, improving ARI/NMI and retrieval Recall@1, especially where classes touch.
3. **CE is still the supervised default; HL is a fairness tool.** On balanced data, CE’s margins are hard to beat. With skewed labels or tail KPIs, HL *can* help—but only if I stabilize temperature, prototypes, and learning rate, or blend it with CE late.
4. **Soft densities are for uncertainty, not semantics.** GMMs preferred 2–4 visual modes where I had 6–8 semantic classes. That’s a feature: use them when I want calibrated responsibilities, anomaly flags, and generative scores—not when I need one-hot identities.
5. **Graph context is local gold, global clay.** Node2Vec sharpened local neighborhoods (1-NN, silhouette up), but linear-probe Top-1 dipped. If deployment needs nearest-neighbor retrieval or community detection, that’s a good trade; for flat classifiers, add a small MLP head or mix graph and pixel features.

A practical playbook (what I recommend using when)

* **I need a map of my data (audits, cohorts, rare types):** Run PCA→t-SNE (PCA to 50 dims, perplexity 30–50, early exaggeration ~12), then compute k-NN Recall@1/mAP@10 to quantify the picture. If outliers and mid-range relations matter, prefer t-SimCLR for representations before t-SNE.
* **I have lots of unlabeled images and want strong features:** Start with **SimCLR** if I can afford large effective batches; otherwise use **VICReg** (no negatives, moderate batch). For robustness to outliers/hierarchies, switch the similarity to a Student-t variant (**t-SimCLR**) or add variance weight in VICReg.
* **I want zero-/few-shot and retrieval now:** Use **CLIP-style** dual encoders. Keep image/text encoders frozen; do prompt engineering for classification, or build an ANN index for retrieval.
* **I must cluster unlabeled data into semantic groups:** Begin with **InfoNCE Clustering** under the **I-Con** template: = augmentations + kNN (k≈5–10) + one-hop propagation; uniform debiasing . from soft cluster assignments with a temperature schedule. If compute is tight or soft uncertainty is essential, use **GMMs** on strong features (e.g., DINO/SimCLR embeddings) and accept “visual modes ≠ labels.” If I need end-to-end refinement and can monitor collapse, a DEC-style **DCD** is viable with entropy, separation, and temperature annealing.
* **I’m training a classifier with labels:** Default to **Cross-Entropy**. If class imbalance is severe and macro-F1/balanced accuracy are the KPIs, try a two-stage schedule: pretrain with CE, then fine-tune the head with **Harmonic Loss** (lower temperature, smaller LR, no weight decay on prototypes). Use class-balanced sampling or focal-like reweighting as needed.
* **My domain is a graph (few or no node features):** Use **Node2Vec** (tune ) for embeddings, then a simple classifier or k-NN. If node attributes exist, hybridize: learn vision/text/tabular features with SSL, then propagate via graph edges or train a lightweight GNN on top.

How to “design a method” with I-Con (the unifying recipe)

Think in four knobs, all explicit in I-Con:

1. **Neighborhoods :** augmentations, kNN, random-walk propagation, labels, or text pairs.
2. **Kernel / similarity:** Gaussian (cosine/dot) vs. Student-t for heavy tails.
3. **Debiasing:** uniform mix to prevent overconfidence; higher for noisy or sparse neighborhoods.
4. **Learner :** soft clusters, logits over prototypes, or cross-modal alignments.

Choosing these four recovers t-SNE (fixed , learned low-dim ), SimCLR (aug-pair , cosine ), VICReg (pairwise invariance with variance/cov penalties), CE/HL (one-hot over classes), DEC/InfoNCE Clustering (cluster-prob matched to debiased neighborhood ). The same template also suggests hybrids—for example, CLIP-style with Student-t ; or CE-trained prototypes fine-tuned with InfoNCE Clustering to clean decision boundaries in the unlabeled pool.

*Limitations and next steps*

My comparisons used ImageNet-100 slices, modest backbones, and constrained batch sizes; several methods (SimCLR, InfoNCE Clustering, CLIP) scale non-linearly with batch and corpus size. Future extensions should include larger backbones (ViT-B/16), stronger indexes for kNN neighborhoods, calibration metrics (ECE/Brier) for supervised heads (especially HL), and broader modalities (audio, tabular, time series) to stress the I-Con template outside vision. I also recommend reporting both global (linear-probe) and local (weighted k-NN) discriminative metrics by default; when they diverge, you learn something important about the geometry you’ve learned. If I had to pick defaults:

For *labeled* balanced classification, train CE; for *labeled* but *imbalanced*, finish with a short HL fine-tune.

For *unlabeled* representation learning, use SimCLR if I have batch budget, otherwise VICReg; if mid-range structure matters, move to t-SimCLR.

For *clustering*, start with InfoNCE Clustering under I-Con with , k=5–10, and one-hop propagation; fall back to GMMs when I need calibrated uncertainty or compute is tight. For *zero-shot and retrieval*, use CLIP.

For *graph problems*, Node2Vec is a fast, strong baseline; add attributes or a light GNN if I need global margins.

And for *exploration and audits*, always include a PCA→t-SNE panel plus retrieval Recall@1/mAP@10—those two plots and two numbers catch most geometry mistakes early.

I recommend a read through I-Con, these aren’t disconnected recipes; they’re just different choices of neighborhoods, kernels, debiasing, and learners. That is the unifying lesson of the survey: once I decide what information a method should conserve—and how confident I want it to be about that information—the rest follows.

REFERENCES

1. “I-Con: A Unifying Framework for Representation Learning”,  [Alshammari](https://openreview.net/profile?id=~Shaden_Naif_Alshammari1), [Hershey](https://openreview.net/profile?id=~John_R._Hershey1), [Feldmann](https://openreview.net/profile?id=~Axel_Feldmann1), [Freeman](https://openreview.net/profile?id=~William_T._Freeman1),  [Hamilton](https://openreview.net/profile?id=~Mark_Hamilton1) (2025) *Arxiv Pre-Print*
2. Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2579–2605.
3. Yang, J., Parikh, D., & Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5147–5156).
4. Baek, D. D., Liu, Z., Tyagi, R., & Tegmark, M. (2025). Harmonic loss trains interpretable AI models. *arXiv preprint arXiv:2502.01628.* <https://arxiv.org/abs/2502.01628>
5. Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *Proceedings of the International Conference on Machine Learning* (pp. 1597–1607)
6. Bishop, C. M. (2006). *Pattern recognition and machine learning.* Springer.
7. Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 855–864).
8. Bardes, A., Ponce, J., & LeCun, Y. (2021). VICReg: Variance-invariance-covariance regularization for self-supervised learning. a*rXiv preprint arXiv:2105.04906*. <https://arxiv.org/abs/2105.04906>
9. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *Proceedings of the International Conference on Machine Learning.* <https://arxiv.org/abs/2103.00020>
10. Hu, T., Liu, Z., Zhou, F., Wang, W., & Huang, W. (2023). Your contrastive learning is secretly doing stochastic neighbor embedding. *International Conference on Learning Representations (ICLR).* <https://openreview.net/forum?id=qJZOSqbfNxS>
11. Gidaris, S., Singh, P., & Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. In *Proceedings of the International Conference on Learning Representations (ICLR).* <https://openreview.net/forum?id=S1v4N2l0->
12. Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). DeepWalk: Online learning of social representations. *In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 701–710). Association for Computing Machinery. <https://doi.org/10.1145/2623330.2623732>
13. Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., & Krishnan, D. (2020). Supervised contrastive learning. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 18661–18673). Curran Associates, Inc.
14. Kobak, D., Berens, P. The art of using t-SNE for single-cell transcriptomics. *Nat Commun* **10**, 5416 (2019). <https://doi.org/10.1038/s41467-019-13056-x>
15. *Under review as submission to TMLR User Segmentation in Recommender Systems: Problem Formulation, Algorithms, and Evaluations*. (n.d.). <https://openreview.net/pdf/28118b862d4cdac8336133f9881d62a242abb68c.pdf>
16. Leiber, C., Miklautz, L., Plant, C., & Böhm, C. (2025). *An Introductory Survey to Autoencoder-based Deep Clustering -- Sandboxes for Combining Clustering with Deep Learning*. ArXiv.org. <https://arxiv.org/abs/2504.02087>
17. Cui, X., Wu, R., Liu, Y., Chen, P., Chang, Q., Liang, P., & He, C. (2025). scSMD: a deep learning method for accurate clustering of single cells based on auto-encoder. *BMC Bioinformatics*, *26*(1). <https://doi.org/10.1186/s12859-025-06047-x>
18. Mañas, O., Lacoste, A., Giro‑i‑Nieto, X., Vázquez, D., & Rodríguez, P. (2021, May 3). *Seasonal Contrast: Unsupervised Pre‑training from uncurated remote sensing data* (arXiv:2103.16607v2) [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2103.16607>
19. *Reproducing and Improving CheXNet: Deep Learning for Chest X-ray Disease Classification*. (2025). Arxiv.org. <https://arxiv.org/html/2505.06646v1>
20. Pellegrino, N., Szczecina, D., & Fieguth, P. (2025, February 19). *Loss functions robust to the presence of label errors.* Journal of Computational Vision and Imaging Systems, 10(1), 24–29. <https://doi.org/10.15353/jcvis.v10i1.10004>
21. *A Systematic Review on Long-Tailed Learning*. (2023). Arxiv.org. <https://arxiv.org/html/2408.00483v1>
22. Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A Simple Framework for Contrastive Learning of Visual Representations. *ArXiv:2002.05709 [Cs, Stat]*. <https://arxiv.org/abs/2002.05709>
23. Imagawa, K., & Shiomoto, K. (2024). Evaluation of Effectiveness of Self-Supervised Learning in Chest X-Ray Imaging to Reduce Annotated Images. *Journal of Imaging Informatics in Medicine*, *37*(4), 1618–1624. <https://doi.org/10.1007/s10278-024-00975-5>
24. Kacem Chehdi, Soltani, M., & Cariou, C. (2014). Pixel classification of large-size hyperspectral images by affinity propagation. *Journal of Applied Remote Sensing*, *8*(1), 083567–083567. <https://doi.org/10.1117/1.jrs.8.083567>
25. Fu, Y., Liu, X., Sarkar, S., & Wu, T. (2021). Gaussian mixture model with feature selection: An embedded approach. *Computers & Industrial Engineering*, *152*, 107000. <https://doi.org/10.1016/j.cie.2020.107000>
26. Jiao, P., Yu, K., Bao, Q., Jiang, Y., Guo, X., & Zhao, Z. (2024). Graph contrastive learning with node-level accurate difference. *Fundamental Research*. <https://doi.org/10.1016/j.fmre.2024.06.013>
27. *Semantic Recommendations*. (2017). Fastforwardlabs.com. <https://semantic-recommendations.fastforwardlabs.com/>
28. Wu, P., Wang, S., Rosa, K. D., & Hu, D. H. (2023). *FORB: A Flat Object Retrieval Benchmark for Universal Image Embedding*. ArXiv.org. <https://arxiv.org/abs/2309.16249>
29. Xie, J., Girshick, R., & Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In International conference on machine learning (pp. 478-487). PMLR.

APPENDIX

1. Libraries used
2. Instructions to clone the dataset
3. Jupyter Notebook for Demonstration of Each Method
4. Implementation Notes for InfoNCE Clustering

A. LIBRARIES USED:

numpy

pandas

torch

torchvision

openTSNE

scikit-learn

scipy

matplotlib

seaborn

pathlib

typing

pillow

pyarrow

tqdm

networkx

json

base64

timm

kornia

open-clip-torch

faiss

B. Instructions to Clone the Dataset

WARNING: Although modest samples and CUDA used, *some* of this code may take some time to run on your machine.

To run this code clone the ImageNet-100 dataset to the project root folder:  
  
**git clone** [**https://huggingface.co/datasets/clane9/imagenet-100**](https://huggingface.co/datasets/clane9/imagenet-100)

**Put the Jupyter notebook in the same directory.**

**C.** Code Used for Demonstration of Each ML Method (see Jupyter notebook)

Download at:

<https://github.com/chinaexpert1/Survey_of_10_ML_methods>

D. Implementation Notes for InfoNCE Clustering

**InfoNCE Clustering: Code Implementation vs. Theory Mapping**

|  |  |
| --- | --- |
| **Code Segment (Implementation)** | **Conceptual Explanation (Theory – I-Con Framework)** |
| Feature Bank Construction (build feature bank): Iterates over the dataset with a pre-trained encoder (e.g. ViT) to compute feature vectors for each image (inference mode, no grad). Collects and concatenates all features into a NumPy array. | **Feature Extraction for Clustering:** Generates a feature bank of high-dimensional embeddings for all images using a pre-trained model. Theoretically, this provides a fixed representation space in which similarity can be measured. These features serve as the basis for defining neighborhoods (used later for clustering). By using a strong pre-trained encoder, the method leverages rich representations, akin to starting with a meaningful embedding space before clustering. |
| Nearest Neighbor Graph (build knn graph): Uses sklearn.NearestNeighbors to find the k+1 nearest neighbors of each feature vector (including itself). Returns an index array of the k nearest other samples for every data point (excluding the self-index). | **Graph Neighborhood Definition:** Constructs a kNN graph in feature space, capturing the local similarity structure of the data. Theoretically, each image *i* is linked to its *k* most similar images by feature distance. This graph encodes a neighborhood distribution based on feature similarity (inspired by spectral clustering and manifold learning). These neighbor relationships will be treated as positive pair candidates – an assumption that similar images should belong to the same cluster. |
| Clustering Model Init (DebiasedInfoNCEClustering. i Freezes encoder parameters, defines linear head for clustering, sets hyperparameters (, k walk, temperature). | nit**Set**):**up of Cluster Assignment Module:** Initializes clustering module components. Frozen encoder yields fixed embeddings. Linear head acts as cluster centroids. Hyperparameters (, , k walk) implement self-balancing loss and graph expansion in theory. |
| Forward Pass (1) – Soft Cluster Assignment: Computes embeddings, applies clustering head, temperature-scaled softmax (*ϕ*). | **Computing** *ϕ* **(Cluster Membership Distribution):** Produces soft cluster assignments. Theoretically interpretable as probabilities of cluster membership. Temperature scaling may not be present in their implementation. |
| Forward Pass (2) – Learned Neighbor Probabilities (*q*): Computes pairwise similarity of assignments, applies softmax to obtain neighbor distributions. | **Learned Neighborhood Distribution (***q***):** Cluster assignment vectors (*ϕ*) act as similarity measures between images, yielding learned neighbor distributions reflecting clustering decisions. Note that the explicit normalization by expected cluster size mentioned in theory is implicitly achieved in the implementation through the softmax operation over the similarity matrix (sim). |
| Forward Pass (3a) – Define Supervisory Distribution *p*0 (Augmentations): Marks each image and augmentation counterpart as mutual neighbors. | **Supervisory Signal from Augmentations:** Defines indisputable positive pairs. Theoretically enforces augmentation invariance as in contrastive learning. |
| Forward Pass (3b) – Extend *p*0 with k-NN Graph Neighbors: Marks neighbors from precomputed graph as positives, normalizes. | **Supervisory Signal from Feature Neighbors:** Incorporates semantic proximity from k-NN graph into supervisory distribution, analogous to manifold learning principles. |
| Forward Pass (4) – Debiased Mixture (˜*p*): Adds uniform mass across neighbors. | **Debiasing with Uniform Distribution:** Mitigates false negative issue in contrastive learning; analogous to label smoothing, referred to as self-balancing loss in theory[2]. The uniform prior distribution is normalized locally over the anchor's neighborhood support size (supp), effectively approximating global normalization for computational efficiency. |
| Forward Pass (5) – Loss Computation (KL Divergence):  Computes KL divergence loss between learned *q* and supervisory ˜*p*. | **I-Con Loss (Matching Distributions):** Matches model-predicted neighbor distribution to target supervisory distribution, performing InfoNCE clustering. The entropy term of the supervisory distribution (p\_tilde) is intentionally omitted from the loss calculation, as it is constant with respect to model parameters, aligning the implemented loss directly with the theoretical KL divergence minimization. |
| Data Augmentation Pipeline (ClusteringDataset and transforms): Generates two augmented views per image. | **Augmentation for Contrastive Learning:** Implements augmentation invariance assumption (standard in contrastive learning frameworks like SimCLR/DINO). |
| Training Loop – Data Preparation: Creates dataset variants, batches augmentation pairs. | **Batch Setup with Positive Pairs:** Ensures augmentation pairs are adjacent in batches for contrastive pairing. |
| Training Loop – Optimization: Updates clustering head weights using computed loss, encoder fixed. | **Learning Cluster Assignments:** Optimizes cluster decision boundaries without altering feature representation; analogous to unsupervised classification on fixed embeddings. |
| **Code Segment (Implementation)** | **Conceptual Explanation (Theory – I-Con Framework)** |
| Post-training – Cluster Assignment: Assigns final clusters using trained head. | **Resulting Clustering (Inference):** Produces unsupervised classification, theoretical result of InfoNCE clustering; evaluated by accuracy metrics if labels exist. |
| Alternate Pipeline (run debiased): Compact alternative implementation. | **Same Theoretical Steps in Brief:** Simplified variant, theoretically identical to main pipeline, confirms robustness and consistency of the method. |

Here’s some notes on the implementation of their loss function:

## Deconstructing their Loss Function

This defines the **learned conditional distribution** over index given anchor , combining:

1. Cluster co-membership probabilities, weighted by , and normalized by expected cluster size.
2. A uniform prior weighted by , where is the dataset size.

## The Mapping to Code

z = self.encoder(x)

This gives the representation:

phi = F.softmax(**self.head(z)** / **self.tau,** dim=1)

This applies a cluster head (linear layer) and softmax with temperature , so:

sim = torch.matmul(phi, phi.t()) # (B, B)

This computes:

That’s the numerator inside our sum. Note: the code **does not divide by**  explicitly — instead, it **implicitly normalizes later via softmax**.

**q** = F.softmax(sim / self.tau, dim=1)

Now, apply softmax over j:

So this line directly implements the **left-hand side** of their formula — it's the learned distribution over "which other sample is similar to i".

## Constructing the Debiased Target

The code first builds a **positive prior** p0[j|i] using:

* True pairs
* Graph neighbors
* Random walk over k-NN

p0 = torch.zeros\_like(q)  
 **for** i **in** range(0, B, 2):  
 p0[i, i+1] = 1; p0[i+1, i] = 1  
 **for** b, i **in** enumerate(idxs.tolist()):  
 nbrs = self.knn[i]  
 walk = nbrs  
 **for** \_ **in** range(self.k\_walk - 1): walk = self.knn[walk].ravel()  
 all\_n = np.unique(np.concatenate([nbrs, walk]))  
 mask = torch.tensor([1 **if** j **in** all\_n **else** 0 **for** j **in** \ idxs.tolist()], device=x.device)  
  
p0[b] += mask

Then normalize:

p0 = p0 / p0.sum(**dim**=1, **keepdim**=True).clamp(min=1)

This gives:

Then:

supp = (p0 > 0).sum(dim=1, keepdim=True).float()  
p\_tilde = (1 - self.alpha) \* p0 + self.alpha / supp

This is our debiased mixture:

Where supp approximates the denominator or more locally, the size of i's support. The term involving α\alphaα is a uniform distribution to introduce debiasing, which ensures robustness against incorrect negative pairings and false negatives. In the implemented code, this uniform debiasing is locally normalized:

## The Loss Function, Implemented

loss = - (p\_tilde \* torch.log(q + 1e-10)).sum(dim=1).mean()

This is the **cross-entropy** term (or more precisely, the negative expected log-likelihood):

This is the **negative entropy term** from:

In their code, they omit the entropy of because it doesn't depend on the model parameters — so minimizing cross-entropy is equivalent to minimizing the full KL.

Final Summary of Implementation Choices

| Symbol | Meaning | Code |
| --- | --- | --- |
|  | Embedding | z = encoder(x) |
|  | Cluster assignment probs | phi = softmax(head(z)/τ) |
|  | Joint soft cluster co-membership | sim = phi @ phi.T |
|  | Learned conditional distribution | q = softmax(sim / τ) |
|  | Debiased supervisory signal | p\_tilde = (1 - α) \* p0 + α / supp |
|  | Loss | loss = - (p\_tilde \* log(q)).sum(dim=1).mean() |

Complete snippet:

Complete snippet:  
  
**class** **DebiasedInfoNCEClustering**(**nn**.**Module**):  
 **def** **\_\_init\_\_**(**self**, encoder, embed\_dim, num\_clusters, knn\_graph, alpha=0.1, k\_walk=1):  
 **super**().\_\_init\_\_()  
 **self**.encoder = encoder  
 **self**.head = nn.Linear(embed\_dim, num\_clusters, bias=False)  
 **self**.knn = knn\_graph  
 **self**.alpha = alpha  
 **self**.k\_walk = k\_walk  
 **self**.tau = 0.1  
 **for** p **in** **self**.encoder.parameters(): p.requires\_grad=False  
  
 **def** **forward**(**self**, x, idxs):  
 # 1) embeddings → soft assignments  
 z = **self**.encoder(x)  
 phi = F.softmax(**self**.head(z) / **self**.tau, dim=1) # temperature on cluster head  
  
 # 2) build qϕ(j|i)  
 sim = torch.matmul(phi, phi.t()) # (B, B)  
 q = F.softmax(sim / **self**.tau, dim=1) # apply temperature before softmax  
  
 # 3) build supervisory p₀  
 B = idxs.size(0)  
 p0 = torch.zeros\_like(q)  
 **for** i **in** range(0, B, 2):  
 p0[i, i+1] = 1; p0[i+1, i] = 1  
 **for** b, i **in** enumerate(idxs.tolist()):  
 nbrs = **self**.knn[i]  
 walk = nbrs  
 **for** \_ **in** range(**self**.k\_walk - 1): walk = **self**.knn[walk].ravel()  
 all\_n = np.unique(np.concatenate([nbrs, walk]))  
 mask = torch.tensor([1 **if** j **in** all\_n **else** 0 **for** j **in** idxs.tolist()], device=x.device)  
 p0[b] += mask  
 p0 = p0 / p0.sum(dim=1, keepdim=True).clamp(min=1)  
  
 # 4) debiased mixture  
 supp = (p0 > 0).sum(dim=1, keepdim=True).float()  
 p\_tilde = (1 - **self**.alpha) \* p0 + **self**.alpha / supp  
  
 # 5) loss: KL(p\_tilde || q)  
 loss = - (p\_tilde \* torch.log(q + 1e-10)).sum(dim=1).mean()  
 **return** loss, phi