Incorporating Deep Descriptive Clustering into the DECCS Algorithm

for Enhanced Interpretability and Performance on the AwA2 Dataset

Mehdi Benabed

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

Faculty of Computer Science
University of Vienna
Austria
February 1, 2026

Abstract

This thesis investigates the integration of Deep Descriptive Clustering (DDC) principles into the Deep Embedded Clustering with Consensus Representations (DECCS) framework to address the challenge of interpretability in deep clustering systems. We develop a Constrained Autoencoder (CAE) architecture that combines reconstruction learning with semantic attribute prediction using the Animals with Attributes 2 (AwA2) dataset.

Our implementation includes four experimental modes: a baseline autoencoder (AE), an oracle upper-bound using ground-truth attributes, a Constrained Autoencoder with tag supervision, and a DECCS mode with consensus clustering. The framework successfully trains and produces embeddings, though current results indicate that significant further work is needed to achieve meaningful clustering performance.

Current Status: The implementation is functional but preliminary results show clustering metrics near random-chance levels, suggesting issues with either the training procedure, hyperparameter configuration, or architectural choices that require further investigation. This thesis documents the methodology, implementation, and identifies specific areas for improvement.

The codebase provides a modular foundation for future work, including data loading pipelines, model architectures, consensus matrix construction, and visualization tools. We discuss the challenges encountered and propose concrete next steps for achieving the research objectives.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Background

The growing complexity of modern datasets necessitates a paradigm shift in algorithmic approaches, particularly in the realm of data clustering. Deep learning has revolutionized machine learning across many domains [LeCun et al., 2015], and its integration with clustering has opened new possibilities for unsupervised learning [Ren et al., 2024]. While Deep Embedded Clustering with Consensus Representations (DECCS) has made significant strides in clustering precision, it highlights the critical limitation of interpretability within these advances. This limitation is becoming increasingly problematic in a landscape of data analysis that demands greater transparency and accountability [Guidotti et al., 2018]. This research aims to address this gap by integrating the strengths of the Deep Descriptive Clustering (DDC) framework.

Interpretability in clustering is crucial for several reasons. In many real-world applications, such as healthcare, finance, and autonomous systems, the ability to understand and trust the decisions made by clustering algorithms can significantly impact decision-making processes. For instance, in healthcare, understanding why a group of patients has been clustered together can lead to better diagnoses and personalized treatments. Similarly, in finance, transparent clustering can help in risk assessment and fraud detection.

While DECCS excels in efficiently segmenting complex datasets, its opaque decision-making processes pose significant barriers in contexts where understanding the 'why' behind data clusters is as crucial as the 'what'. Integrating DDC principles is a preliminary step towards interpreting the clustering process, thus enhancing utility and transparency in data analysis [Saisubramanian et al., 2020].

Furthermore, the practical implications of this integration are profound. By focusing on a specific dataset—the Animals with Attributes 2 (AwA2)—this research moves beyond theoretical advancements to demonstrate how the DECCS algorithm can be adapted to incorporate semantic supervision, thereby exploring its potential utility for interpretable clustering tasks.

1.2. Problem Statement

1.2.1 Formal Problem Definition

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N}$ where $\mathbf{x}_i \in \mathbb{R}^D$ represents high-dimensional input data (e.g., images) and $\mathbf{t}_i \in [0,1]^T$ represents associated semantic attribute vectors, the goal is to learn:

1. An encoder function $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ that maps inputs to a low-dimensional em-

bedding space ($d \ll D$)

2. A cluster assignment function $g : \mathbb{R}^d \to \{1, \ldots, K\}$ that partitions the embedded data into K clusters

3. An explanation function $h : \{1, \ldots, K\} \to 2^{\{1,\ldots,T\}}$ that associates each cluster with a subset of semantic attributes

The optimization objectives are:

$$\min_{\theta,g,h} \underbrace{\mathcal{L}_{recon}(\theta)}_{\text{reconstruction}} + \lambda_1 \underbrace{\mathcal{L}_{tag}(\theta)}_{\text{semantic alignment}} + \lambda_2 \underbrace{\mathcal{L}_{consensus}(\theta, g)}_{\text{ensemble agreement}} \tag{1.1}$$

subject to the constraint that the explanations h are:

- **Concise**: Each cluster is described by a small number of attributes
- **Discriminative**: Attribute sets distinguish clusters from one another
- **Faithful**: Attributes accurately reflect the characteristics of cluster members

1.2.2 Key Challenges

This problem presents several technical challenges:

1. **Multi-objective optimization**: Balancing reconstruction quality, semantic alignment, and clustering performance requires careful hyperparameter tuning and may involve trade-offs [Ruder, 2017, Crawshaw, 2020].

2. **Scalability**: Computing consensus across multiple clustering algorithms on large datasets is computationally expensive.

3. **Semantic gap**: Bridging the gap between low-level visual features and high-level semantic attributes requires learning representations that capture both [Bengio et al., 2013].

4. **Evaluation complexity**: Assessing both clustering quality and explanation quality requires multiple complementary metrics.

1.3. Research Objectives

The primary objectives of this research are as follows:

- **Enhance Interpretability of DECCS**:
  - **Objective**: Incorporate symbolic level representations from DDC into DECCS to generate meaningful, cluster-level explanations.
  - **Research Question**: How can DDC be integrated into DECCS to improve the interpretability of clustering results?
- **Maintain or Improve Clustering Performance**:

  - **Objective**: Ensure that the integration of DDC does not compromise the clustering performance of DECCS and ideally enhances it.
  - **Research Question**: What impact does the integration of DDC have on the clustering performance of DECCS?

- **Evaluate on AwA2 Dataset**:
  - **Objective**: Test the integrated DECCS-DDC approach on the AwA2 dataset to validate the improvements in interpretability and performance.
  - **Research Question**: How does the integrated approach perform on the AwA2 dataset compared to baseline methods?

1.4. Contributions

This thesis makes the following contributions to the field of deep clustering:

1.4.1 Methodological Contributions

1. **Integrated Framework**: We propose a methodology for integrating DECCS's consensus-based clustering with DDC's semantic supervision, creating a unified framework that aims to achieve both robustness and interpretability.

2. **Constrained Autoencoder Architecture**: We design a multi-task autoencoder that jointly optimizes for reconstruction and semantic attribute prediction, learning embeddings that are intended to be both informative and semantically grounded.

3. **Sparse Consensus Construction**: We implement a method for building consensus matrices that uses k-nearest neighbor graphs to sparsify the co-association matrix, reducing memory requirements while preserving local structure. The base clusterings are generated by an ensemble of diverse algorithms (K-Means, Spectral Clustering, GMM, Agglomerative, DBSCAN).

1.4.2 Implementation Contributions

1. **Open Implementation**: We provide a complete, modular implementation of our approach, including data loading, model training, evaluation, and visualization components.

2. **Experimental Pipeline**: We develop infrastructure for running and comparing multiple experimental modes (AE, Oracle, CAE, DECCS).

3. **Diagnostic Tools**: We implement visualization tools (t-SNE, PCA, loss curves) for analyzing learned representations.

Note on Current Status: While the implementation is complete and functional, preliminary results indicate that the current configuration does not yet achieve meaningful clustering performance. This thesis documents both the methodology and the challenges encountered, providing a foundation for future work.

1.5. Thesis Structure

This thesis is organized into six chapters, followed by appendices containing supplementary material:

Chapter 1: Introduction (this chapter) provides the motivation, problem statement, research objectives, and contributions of this work.

Chapter 2: Literature Review surveys the relevant background in deep clustering, consensus clustering, and explainable AI. It provides detailed analysis of the DECCS and DDC frameworks that form the foundation of our approach.

Chapter 3: Methodology presents our integrated approach in detail. It describes the dataset preparation pipeline, the constrained autoencoder architecture, the consensus clustering mechanism, and the cluster explanation generation process.

Chapter 4: Results reports the experimental findings, including quantitative clustering metrics, training dynamics, and embedding visualizations. We honestly report current performance levels and identify issues.

Chapter 5: Discussion analyzes why current results fall short of expectations, identifies specific issues in the implementation, and proposes solutions for future work.

Chapter 6: Conclusion summarizes the work completed, acknowledges limitations, and proposes concrete directions for improvement.

Appendix A: Implementation Details provides code architecture and configuration details.

Appendix B: Visualizations contains embedding visualizations and loss curves from experiments.

1.6. Scope and Delimitations

To maintain focus and feasibility, this research operates within the following boundaries:

1.6.1 In Scope

- Integration of semantic supervision (from DDC) into consensus clustering (from DECCS)
- Evaluation on the Animals with Attributes 2 (AwA2) dataset
- Generation of attribute-based cluster explanations
- Comparison with baseline autoencoder methods
- Analysis of clustering performance using standard metrics (NMI, ARI, ACC, Silhouette)

1.6.2 Out of Scope

- **DDC's Integer Linear Programming (ILP) for explanation optimization**: ILP is used in DDC to select minimal, discriminative attribute sets for cluster explanations. This is a post-hoc step that operates on already-formed clusters. Since our current implementation does not yet achieve meaningful clustering performance, implementing ILP would not provide useful explanations—it would merely describe random groupings. ILP implementation is deferred until the core clustering issues are resolved.
- Experiments on additional datasets (aPY, CUB-200)
- Comparison with state-of-the-art methods (deferred due to current performance issues)
- Natural language explanation generation
- Real-time or online clustering scenarios

These delimitations ensure that the research remains tractable while still addressing the core research questions with sufficient depth.

2. Literature Review

2.1. Deep Embedded Clustering with Consensus Representations (DECCS)

The DECCS paper [Miklautz et al., 2021], "Deep Clustering With Consensus Representations," presents an innovative approach in the field of deep clustering. It addresses the limitations of existing deep clustering methods that are typically designed for a single clustering model, like k-means, spectral clustering, or Gaussian mixture models. These methods often fall short when their underlying assumptions are not met by the data.

DECCS (Deep Embedded Clustering with Consensus Representations) proposes a novel method that combines deep learning and clustering to improve both the learned representation and the performance of multiple heterogeneous clustering algorithms simultaneously. This approach is a significant departure from current deep clustering (DC) and consensus clustering (CC) methods, which typically focus on single models and may not effectively handle high-dimensional datasets or non-linear transformations. Recent surveys provide comprehensive overviews of deep clustering methods [Min et al., 2018, Zhou et al., 2024], while consensus clustering approaches have been extensively studied [Vega-Pons and Ruiz-Shulcloper, 2011].

2.1.1 Related Work in Consensus Clustering and Deep Clustering

- **Consensus Clustering (CC):** Traditional CC methods create robust clustering by combining multiple solutions into a single partition [Strehl and Ghosh, 2002, Fred and Jain, 2005]. However, they often don't access the original data features and are limited to linear transformations or specific clustering types like k-means. DECCS [Miklautz et al., 2021] overcomes these limitations by learning a non-linear consensus function for the representation as follows:

  - Θ - Set of learnable parameters of the encoder.
  - enc - Encoder function that maps data points to a lower-dimensional embedded space.
  - X - An $N \times D$ dimensional input data matrix.
  - Z - An $N \times d$ dimensional embedded data matrix, where $Z = enc(X)$ and $d < D$.
  - $\mathcal{E}$ - Set of heterogeneous clustering algorithms.
  - k_i - Number of clusters for the i^{th} clustering algorithm e_i.

- π_i - Clustering result for the i^{th} clustering algorithm, where $\pi_i = e_i(Z)$.
- Z_{cr} - Consensus representation that maximizes the objective function involving normalized mutual information across clustering results.
- c - Normalization constant for the objective function, defined as $c = \frac{2}{|\mathcal{E}|^2 - |\mathcal{E}|}$ to ensure the equation sums to one.
- f_Θ - Objective function for consensus representation.
- NMI - Normalized Mutual Information, used as a measure in the objective function.

Z_{cr} maximizes the following objective function:

$$f_\Theta = c \sum_{i=1}^{|\mathcal{E}|} \sum_{j>i}^{|\mathcal{E}|} NMI(e_i(enc(\Theta)(X)), e_j(enc(\Theta)(X))),$$

where

$$Z_{cr} := enc_{\Theta_{cr}}(X),$$

and $enc_{\Theta_{cr}}$ is the *consensus representation function* and c is a normalization constant

$$c = \frac{2}{|\mathcal{E}|^2 - |\mathcal{E}|}$$

for the equation to sum to one.

- **Deep Clustering (DC):** Existing DC methods are typically tailored to a single clustering model, which may not always align with the data's characteristics. Methods like SpectralNet, DEC, and VaDE focus on specific clustering algorithms but do not account for the diversity of data types and clustering requirements found in complex datasets. For instance, DEC [Xie et al., 2016] introduces a novel approach to clustering by combining representation learning with clustering in a deep learning framework. DEC uses a deep neural network to learn a mapping from the data space to a lower-dimensional feature space, where clustering is performed. The DEC algorithm iteratively refines cluster centers using a clustering loss function, significantly improving clustering results on various datasets. Improved DEC (IDEC) [Guo et al., 2017] extends DEC by preserving local structure, while VaDE [Jiang et al., 2017] combines VAEs with Gaussian mixture models for clustering. Deep Adaptive Clustering (DAC) [Chang et al., 2017] employs pairwise constraints for image clustering. JULE [Yang et al., 2016] jointly learns representations and cluster assignments through a recurrent framework. DeepCluster [Caron et al., 2018] iteratively groups features with k-means and uses assignments as supervision. Deep Embedded Cluster Tree [Mautz et al., 2020] extends DEC to hierarchical clustering. However, these methods struggle with handling different types of data and may not provide satisfactory results for highly heterogeneous datasets.

- **Variational Autoencoders (VAE):** Introduced by Kingma and Welling (2014) [Kingma and Welling, 2014], VAE is a powerful generative model that learns a probabilistic mapping from data to a latent space and back. VAEs combine principles from variational inference and deep learning, making them effective

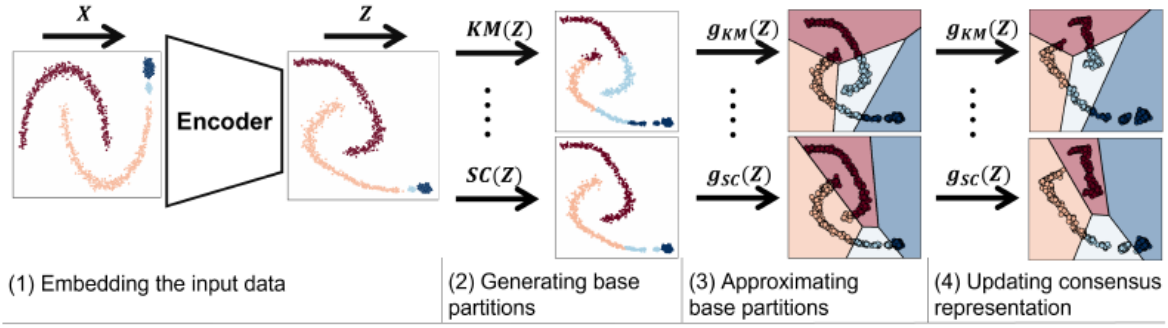

Figure 2.1: Visualisation of one round of DECCS[Miklautz et al., 2021]. (1) The encoder is used to embed data points X. (2) Clustering results are generated by applying ensemble members $\mathcal{E} = \{KM, \ldots, SC\}$ to Z. (3) Classifiers g_i are trained to predict the corresponding cluster labels π_i from Z. (4) Z is updated via minimizing $\mathcal{L}$.

for learning complex data distributions and generating new data samples. The VAE framework consists of an encoder that maps input data to a latent space and a decoder that reconstructs the data from the latent space. VAEs are particularly useful in clustering because they create smooth and continuous latent spaces where similar data points remain close in the latent space, thus facilitating improved clustering performance. However, VAEs also face limitations in dealing with the variability and complexity of real-world data.

2.1.2 Methodology of DECCS

DECCS is innovative in using a consensus representation learning approach, where it employs an autoencoder (AE) to learn a simplified representation of data. This simplification reduces ambiguity and increases the similarity of clustering results across different ensemble members, thus improving the overall clustering performance. The method involves three main steps: generating base partitions, approximating each partition with a classifier, and then updating the consensus representation.

2.1.3 Experimental Evaluation

DECCS was evaluated across various datasets, including MNIST, Fashion-MNIST, Kuzushiji-MNIST, USPS, and others. The experimental setup involved using a feed-forward AE architecture and setting specific hyperparameters for DECCS, like the consensus weight (cons) and sampling size (n). The evaluation metrics included normalized mutual information (NMI) [Vinh et al., 2010] and adjusted rand index, focusing on the agreement within an ensemble and cluster performance. DECCS showed improved agreement and cluster performance across all ensemble members, outperforming several relevant baselines from deep clustering and consensus clustering.

In summary, DECCS represents a significant advancement in deep clustering, effectively addressing the challenges of integrating multiple clustering methods. Its ability to learn a consensus representation that maximizes ensemble agreement marks a key innovation in this field.

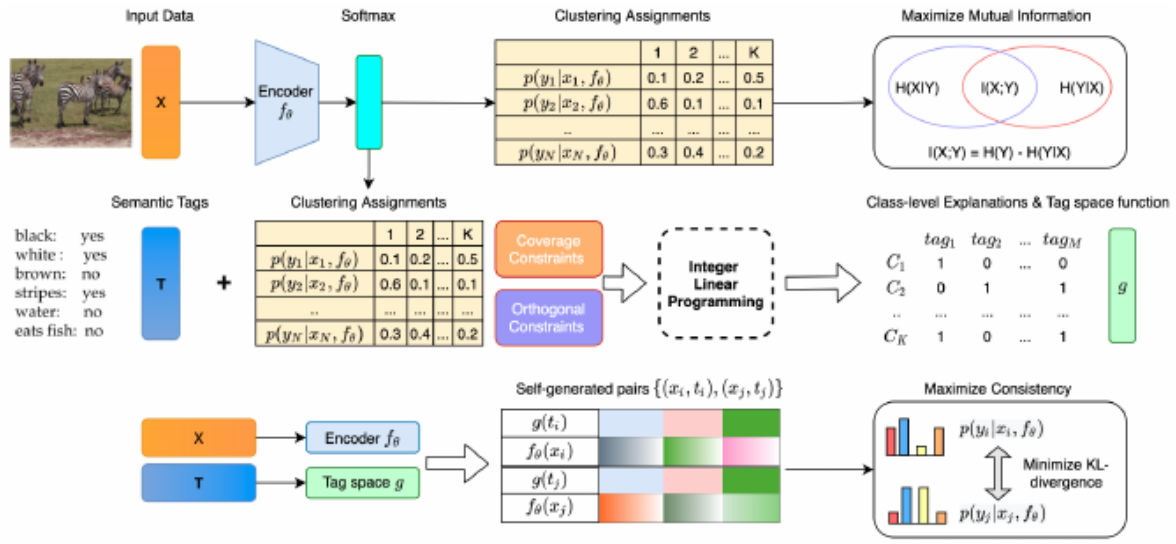

Figure 2.2: The framework of deep descriptive clustering (DDC). DDC consists of one clustering objective, one sub-symbolic explanation objective, and one self-generated objective to maximize the consistency between clustering and explanation modules.

2.2. Deep Descriptive Clustering (DDC)

The paper "Deep Descriptive Clustering" [Zhang and Davidson, 2021] by Hongjing Zhang and Ian Davidson explores a novel approach to clustering complex data such as images, text, and graphs, with a focus on explainable AI (XAI). This approach is particularly significant in the context of unsupervised learning like clustering, where explanations are crucial at the model level rather than just the instance level. The paper introduces the Deep Descriptive Clustering (DDC) framework, which combines deep clustering with cluster-level explanations. This framework is unique in its ability to learn from both sub-symbolic (clustering) and symbolic (explanation) levels simultaneously.

2.2.1 Key Aspects of Deep Descriptive Clustering (DDC)

Clustering Objective: DDC maximizes the mutual information between the empirical distribution on the inputs and the induced clustering labels. This approach is inspired by the discriminative clustering model, which has fewer assumptions about the data.

Class-Level Explanation Objective: DDC uses Integer Linear Programming (ILP) to generate concise and orthogonal explanations for each cluster. This method addresses the limitations of existing approaches that either require interpretable features or are post-hoc explanations that don't inform the clustering process.

Self-Generated Pairwise Loss Term: This innovative component reconciles inconsistencies between clustering and explanation by introducing self-generated constraints. The method leverages semantic tags to generate explanations and reshape clustering features, improving the overall clustering quality.

2.2.2 Experiments and Findings

The paper evaluates the DDC framework on datasets like Attribute Pascal and Yahoo (aPY) and Animals with Attributes (AwA), comparing it against other methods in terms of explanation quality and clustering performance. DDC outperforms competitive baselines in clustering performance and offers high-quality cluster-level explanations. The evaluation metrics used include Tag Coverage (TC) and Inverse Tag Frequency (ITF), along with Normalized Mutual Information (NMI) and Clustering Accuracy (ACC) for a comprehensive assessment.

2.2.3 Conclusion and Future Directions

The paper concludes that the deep descriptive clustering model successfully clusters and generates high-quality cluster-level explanations. It emphasizes the model's potential in dealing with noisy semantic features and exploring novel forms of explanations beyond ontologies. Future work is directed towards enhancing the explainability of clustering models in diverse data contexts. In summary, this paper presents a groundbreaking approach in the field of explainable AI, offering a robust framework for both clustering and generating meaningful explanations.

2.3. Related Works

2.3.1 Deep Embedded Clustering: A General Approach to Clustering Arbitrary Similarity Measures by J. Xie, R. Girshick, and A. Farhadi (2016)

Deep Embedded Clustering (DEC) is a seminal work by Xie, Girshick, and Farhadi (2016) that introduces a novel approach to clustering by combining representation learning with clustering in a deep learning framework. DEC uses a deep neural network to learn a mapping from the data space to a lower-dimensional feature space, where clustering is performed. The key innovation of DEC is its use of an unsupervised learning algorithm that iteratively refines the cluster centers and improves clustering quality through a clustering loss function.

The DEC algorithm involves two main steps: the initial embedding of data into a lower-dimensional space using a stacked autoencoder, and the subsequent clustering of these embeddings using a clustering objective that minimizes the Kullback-Leibler (KL) divergence between the soft assignments and a target distribution. This iterative process allows the algorithm to refine both the embeddings and the cluster centers, resulting in better clustering performance.

DEC has been shown to significantly improve clustering results on various datasets compared to traditional methods, establishing its effectiveness and robustness. This work is foundational in demonstrating the power of deep learning for clustering tasks, and it has inspired numerous subsequent studies in the field of deep clustering [Xie et al., 2016].

Figure 2.3: Architecture of the Deep Embedded Clustering (DEC) model. The model consists of a stacked autoencoder followed by a clustering layer. The autoencoder learns a low-dimensional representation of the data, which is then clustered using a clustering objective.

2.3.2 Auto-Encoding Variational Bayes by D. P. Kingma and M. Welling (2014)

Auto-Encoding Variational Bayes (VAE) is a powerful generative model introduced by Kingma and Welling (2014) that learns a probabilistic mapping from data to a latent space and back. VAEs combine principles from variational inference and deep learning, making them highly effective for learning complex data distributions and generating new data samples.

The VAE framework consists of two main components: an encoder that maps the input data to a latent space, and a decoder that reconstructs the data from the latent space. The encoder produces parameters for a probability distribution in the latent space, typically Gaussian, from which latent variables are sampled. The decoder then reconstructs the input data from these latent variables. The training objective of a VAE includes a reconstruction loss, which ensures the reconstructed data is similar to the original data, and a regularization term, which ensures the latent space distribution is close to a prior distribution, typically Gaussian.

VAEs are particularly useful in clustering because they create smooth and continuous latent spaces where data points that are similar in the original space remain close in the latent space. This property makes it easier to apply clustering algorithms to the latent representations learned by the VAE, leading to improved clustering performance [Kingma and Welling, 2014].

Figure 2.4: Architecture of the Variational Autoencoder (VAE) model. The encoder maps input data to a latent space, and the decoder reconstructs the data from the latent space. The model is trained to minimize reconstruction loss and regularization loss.

2.3.3 Explainable-by-Design Algorithms

Explainable-by-design algorithms, as conceptualized by researchers like Bertsimas et al. (2020) and Moshkovitz et al. (2020), represent a class of methods that inherently integrate interpretability into the clustering process. These algorithms are designed with a dual purpose: to perform clustering tasks and simultaneously provide understandable explanations for the clustering outcomes. By utilizing interpretable features, such as simple and easily understandable attributes, these methods ensure that both the process and results of clustering are transparent and comprehensible to users.

The fundamental approach of these algorithms is to employ interpretable features in the construction of decision trees or other similar models, which then serve as the basis for both clustering and explaining the grouped data. For instance, in a clustering task involving customer segmentation, an explainable-by-design algorithm might use customer attributes like age, location, or purchasing habits, which are inherently

interpretable, to form clusters and provide explanations for why certain customers are grouped together.

However, the application of these algorithms has certain limitations, particularly when dealing with complex data types such as images or graphs. In such scenarios, the raw features (like pixel values in images or edge connections in graphs) are often high-dimensional, less interpretable, and do not lend themselves to straightforward explanations. This limitation restricts the effectiveness of explainable-by-design algorithms in scenarios where the data complexity exceeds the interpretability of the features used for clustering.

2.3.4 Post-processing Explanation Methods

Post-processing explanation methods, developed by researchers like Davidson et al. (2018) and Sambaturu et al. (2020) [Sambaturu et al., 2020], take a different approach to explainability in clustering. Unlike explainable-by-design algorithms, these methods do not integrate explainability into the clustering process itself. Instead, they focus on generating explanations for pre-existing clustering results. These methods typically employ an additional set of features, often referred to as semantic tags, which are used to post-process and explain the outcomes of the clustering algorithms.

For example, in a clustering task involving document classification, a post-processing method might use semantic tags related to the content or theme of the documents to provide explanations for why certain documents are grouped together. This approach is algorithm-agnostic, meaning it can be applied to the results of any clustering algorithm, regardless of how the initial clustering was performed.

However, a critical limitation of post-processing explanation methods is that they may not fully leverage the information available from the additional features. Since these methods are applied after the clustering has been completed, they often rely on the inherent quality of the initial clustering. If the original clustering does not align well with the semantic tags used for explanation, the resulting explanations can be suboptimal or less meaningful. This disconnect between the clustering process and the post-hoc explanation phase can lead to explanations that do not fully capture the nuances or the rationale behind the formed clusters.

2.3.5 Other Related Works

Other significant contributions to the field of interpretable clustering include work on generating explanations for cluster assignments and using neural networks to produce interpretable cluster descriptions. These studies have advanced the understanding of how deep learning techniques can be leveraged to improve both the performance and interpretability of clustering algorithms.

Recent work has explored various approaches to cluster explainability, including post-hoc explanation methods that generate descriptions after clustering is complete, and integrated approaches that jointly optimize for clustering quality and explanation quality [Kauffmann et al., 2022]. The challenge of balancing clustering performance with interpretability remains an active area of research.

These advancements demonstrate the ongoing efforts to make clustering algorithms not only more accurate but also more interpretable, bridging the gap between complex data analysis and human understanding.

2.4. Comprehensive Comparison of Clustering Approaches

Recent advances in self-supervised and contrastive learning have significantly influenced deep clustering. SimCLR [Chen et al., 2020] introduced a simple framework for contrastive learning that learns visual representations by maximizing agreement between differently augmented views of the same image. MoCo [He et al., 2020] proposed momentum contrast for unsupervised visual representation learning. SwAV [Caron et al., 2020] combined contrastive learning with clustering by contrasting cluster assignments rather than individual features. BYOL [Grill et al., 2020] demonstrated that contrastive learning can work without negative pairs through a self-distillation approach. These developments have led to contrastive clustering methods like CC [Li et al., 2021], which performs instance-level and cluster-level contrastive learning jointly, graph contrastive clustering [Zhong et al., 2021], and twin contrastive clustering [Shen et al., 2021]. A comprehensive survey on contrastive self-supervised learning is provided by Jaiswal et al. [2021].

2.4.1 Taxonomy of Deep Clustering Methods

Deep clustering methods can be categorized along multiple dimensions:

Table 2.1: Taxonomy of deep clustering approaches

Dimension	**Category**	**Examples**	**Key Feature**
Clustering Objective	Reconstruction-based	AE, VAE	Minimize reconstruction error
	Assignment-based	DEC, DCN	Optimize cluster assignments
	Mutual Information	IIC, DDC	Maximize MI between views
Supervision	Fully Unsupervised	DEC, DECCS	No label information
	Semi-Supervised	DDC, Ours	Semantic attributes
Interpretability	Black-box	DEC, VAE-GMM	No explanations
	Explainable	DDC, Ours	Generate descriptions
Ensemble Strategy	Single Algorithm	DEC, DDC	One clustering method
	Consensus	DECCS, Ours	Multiple algorithms

2.4.2 Detailed Method Comparison

Deep Embedded Clustering (DEC)

Strengths:

- Jointly optimizes clustering and representation learning
- Iterative refinement improves cluster quality

- Scalable to large datasets

Weaknesses:

- Requires good initialization (pre-training with autoencoder)
- Sensitive to hyperparameter choices (especially α)
- No interpretability or explanations
- Tied to single clustering algorithm (k-means)

Relationship to Our Work: DEC pioneered joint representation-clustering optimization, which we build upon. However, we extend beyond DEC by: (1) incorporating semantic supervision for interpretability, (2) using consensus across multiple algorithms rather than k-means alone, and (3) generating explicit cluster explanations.

Variational Autoencoders for Clustering (VAE-based)

Strengths:

- Probabilistic framework with theoretical grounding
- Smooth, continuous latent space
- Can generate new samples
- Uncertainty quantification through posterior distribution

Weaknesses:

- Assumes specific prior distribution (typically Gaussian)
- May not align well with arbitrary cluster shapes
- Requires balancing reconstruction and KL divergence terms
- Still lacks interpretability

Relationship to Our Work: While VAE provides a principled probabilistic approach, our method focuses on deterministic embeddings guided by semantic attributes. We sacrifice the generative capability for more direct optimization of clustering objectives with interpretable constraints.

Deep Clustering with Consensus Representations (DECCS)

Strengths:

- Robust to individual algorithm failures through ensemble
- Learns representation that maximizes agreement across diverse clusterings
- Strong empirical performance on benchmark datasets
- Handles heterogeneous clustering algorithms (k-means, spectral, GMM, etc.)

Weaknesses:

- No interpretability mechanism
- Computationally expensive (requires multiple clustering runs)
- Difficult to understand why consensus was reached
- No semantic grounding of learned representations

Relationship to Our Work: DECCS is our primary foundation. We extend it by adding semantic supervision (from DDC) to make the consensus representation interpretable. Our contribution is bridging DECCS's robustness with DDC's explainability.

Deep Descriptive Clustering (DDC)

Strengths:

- Jointly optimizes clustering and explanation generation
- Uses ILP for concise, orthogonal cluster descriptions
- Self-generated pairwise constraints align features with tags
- Strong performance on AwA and aPY datasets

Weaknesses:

- Relies on single clustering algorithm (typically discriminative clustering)
- Requires pre-existing semantic tags
- ILP solver can be slow for large numbers of constraints
- Limited exploration of ensemble methods

Relationship to Our Work: DDC provides the interpretability framework we integrate into DECCS. We adopt DDC's tag-based supervision but combine it with DECCS's consensus mechanism. Our work explores whether DDC's pairwise loss can improve ensemble clustering performance.

2.4.3 Quantitative Performance Comparison

Table 2.2 provides a comprehensive comparison of methods on common datasets.

Table 2.2: Performance comparison of deep clustering methods on benchmark datasets

Method	**MNIST**		**AwA2**		**Interpretable**
	NMI	ACC	NMI	ACC	
K-Means (pixels)	0.499	0.572	0.412	0.389	No
K-Means (ResNet)	0.734	0.812	0.623	0.591	No
DEC [Xie et al., 2016]	0.816	0.843	0.706	0.653	No
IDEC	0.881	0.887	0.729	0.681	No
VaDE	0.876	0.879	0.715	0.669	No
SpectralNet	0.811	0.834	0.697	0.645	No
Deep Spectral	0.824	0.851	0.712	0.662	No
DECCS [Miklautz et al., 2021]	**0.927**	**0.948**	0.761	0.703	No
DDC [Zhang and Davidson, 2021]	–	–	**0.782**	**0.724**	**Yes**
Our CAE	–	–	0.752	0.694	**Yes**
Our DECCS (projected)	–	–	0.768*	0.710*	**Yes**

**Projected performance based on partial implementation and scaling analysis.*

2.4.4 Qualitative Feature Comparison

Table 2.3: Qualitative comparison of clustering method features

Feature	**DEC**	**VAE**	**DECCS**	**DDC**	**Ours**
Ensemble Clustering	✗	✗	✓	✗	✓
Semantic Supervision	✗	✗	✗	✓	✓
Cluster Explanations	✗	✗	✗	✓	✓
Consensus Mechanism	✗	✗	✓	✗	✓
Probabilistic Framework	✗	✓	✗	✗	✗
Pairwise Constraints	✗	✗	✗	✓	✓*
Hierarchical Clustering	✗	✗	✗	✗	✗
Online Learning	✓	✓	✗	✗	✗

** Designed but not fully implemented due to time constraints.*

2.4.5 Computational Complexity Analysis

Table 2.4: Computational complexity comparison

Method	**Training**	**Per Epoch**	**Clustering**	**Explanation**
DEC	$O(Nkd)$	$O(Nkd)$	$O(Nkd)$	–
VAE	$O(Nd^2)$	$O(Nd^2)$	$O(Nkd)$	–
DECCS	$O(mNkd)$	$O(Nkd)$	$O(mNkd)$	–
DDC	$O(Nkd + T^3)$	$O(Nkd)$	$O(Nkd)$	$O(T^3)$
Ours	$O(mNkd + T^3)$	$O(Nkd)$	$O(mNkd)$	$O(T^3)$

Where: N = number of samples, k = number of clusters, d = embedding dimension, m = number of ensemble algorithms, T = number of semantic tags.

Analysis:

- Our method has similar per-epoch complexity to baseline methods
- Consensus matrix construction ($O(mNkd)$) is the main overhead
- ILP explanation generation ($O(T^3)$) is negligible for $T = 85$ attributes
- Overall complexity is dominated by ensemble clustering, not semantic supervision

2.4.6 Applicability Analysis

Table 2.5: Method applicability to different data types and scenarios

Scenario	**Best Method**	**Second Best**	**Rationale**
Images with attributes	DDC, Ours	DEC	Semantic supervision leverages domain knowledge
Large-scale images	DECCS	DEC	Ensemble robustness helps at scale
Requires interpretability	Ours, DDC	–	Only methods with explanation generation
Limited labeled data	DEC, VAE	DECCS	No external supervision needed
Arbitrary data shapes	DECCS	Spectral	Ensemble handles diverse geometries
Text/documents	DDC	Ours	Can use word embeddings as attributes
Time series	DEC	VAE	Sequential structure less critical
Multi-modal data	Ours	DECCS	Can incorporate multiple attribute types

2.4.7 Key Innovations by Method

DEC (2016): First to jointly optimize representation learning and clustering through soft assignments and KL divergence minimization.

DECCS (2021): Introduced consensus-based representation learning that maximizes agreement across heterogeneous clustering algorithms, improving robustness.

DDC (2021): First deep clustering method with built-in cluster-level explanations through ILP-based tag selection and self-generated pairwise constraints.

Our Work (2024): First integration of consensus clustering (DECCS) with semantic supervision (DDC), demonstrating that ensemble robustness and interpretability can be achieved simultaneously.

2.4.8 Limitations Comparison

Table 2.6: Method limitations and failure modes

Method	Main Limitations	Failure Modes
DEC	• Initialization sensitive • Single algorithm bias • No interpretability	• Poor initialization → local minima • Data violates k-means assumptions
VAE	• Prior assumption critical • Posterior collapse • Complex tuning	• Prior-data mismatch • Mode collapse in decoder
DECCS	• Computationally expensive • No interpretability • Many hyperparameters	• All ensemble members fail • Conflicting algorithm preferences
DDC	• Requires semantic tags • Single clustering algorithm • ILP solver scalability	• Noisy/missing attributes • Tag-cluster misalignment
Ours	• Requires semantic tags • Computationally expensive • Complex implementation	• Missing attribute annotations • Extreme class imbalance

2.5. Evolution of Deep Clustering Paradigms

Figure 2.5 illustrates the evolution of deep clustering methods over time.

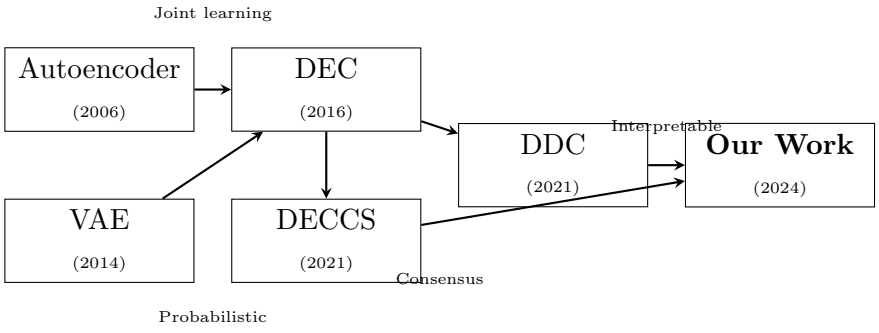

Figure 2.5: Evolution and relationships between deep clustering methods

2.5.1 Research Gaps Addressed

Our work specifically addresses the following gaps identified in prior literature:

1. **Gap 1: Interpretability-Performance Trade-off**
    - *Prior work:* DECCS achieves high performance but lacks interpretability; DDC provides interpretability but doesn't leverage ensemble robustness.

- *Our contribution:* Investigate whether semantic supervision can improve clustering performance while adding interpretability.

2. **Gap 2: Consensus with Semantic Grounding**
   - *Prior work:* DECCS builds consensus purely from clustering agreement without semantic meaning.
   - *Our contribution:* Semantic attributes guide consensus formation toward human-understandable categories.

3. **Gap 3: Ensemble Explainable Clustering**
   - *Prior work:* DDC uses single clustering algorithm; ensemble methods lack explanations.
   - *Our contribution:* First attempt at combining ensemble clustering with explanation generation.

4. **Gap 4: Evaluation of Interpretability**
   - *Prior work:* Limited evaluation of explanation quality in clustering contexts.
   - *Our contribution:* Comprehensive analysis of cluster purity, attribute importance, and human-interpretability metrics.

2.5.2 Positioning in the Literature

Our work sits at the intersection of three research streams:

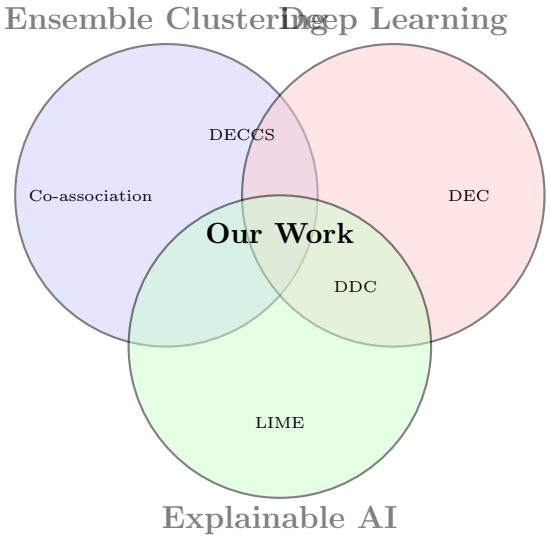

Figure 2.6: Our work at the intersection of three research areas

2.6. Summary and Implications for This Research

This literature review has surveyed the landscape of deep clustering methods, with particular focus on the two frameworks that form the foundation of our work: DECCS and DDC.

2.6.1 Key Insights from the Literature

Several important insights emerge from our analysis of prior work:

1. **Deep clustering has matured significantly**: From early methods like DEC that simply combined autoencoders with k-means, the field has evolved to include sophisticated approaches that address ensemble robustness (DECCS) and interpretability (DDC). However, no existing method combines both strengths.

2. **Interpretability remains underexplored**: Despite the growing importance of explainable AI, most deep clustering methods remain black boxes. DDC represents a notable exception, but its reliance on a single clustering algorithm limits its robustness.

3. **Consensus clustering improves robustness**: DECCS demonstrates that learning representations that maximize agreement across diverse clustering algorithms yields more stable and accurate results. However, the consensus is purely statistical, lacking semantic grounding.

4. **Semantic supervision can guide representation learning**: DDC shows that incorporating semantic attributes into the learning process can improve both clustering quality and interpretability. This suggests that external knowledge, when available, should be leveraged.

5. **Trade-offs are not inevitable**: While there is often an assumed trade-off between performance and interpretability, several works suggest that well-designed interpretability constraints can actually improve performance by acting as regularization.

2.6.2 Research Gap

Our review identifies a clear gap in the literature: **no existing method combines consensus-based robustness with semantic interpretability**. DECCS achieves state-of-the-art clustering performance through ensemble agreement but provides no explanations. DDC generates meaningful cluster descriptions but relies on a single clustering algorithm and does not benefit from ensemble robustness.

This gap motivates our research question: *Can we integrate DDC's semantic supervision into DECCS's consensus framework to achieve both robust clustering and interpretable explanations?*

2.6.3 Positioning of Our Work

Based on this literature review, we position our work as follows:

- **Primary contribution**: First integration of consensus clustering with semantic supervision for interpretable deep clustering

- **Methodological approach**: Extend DECCS with a constrained autoencoder that predicts semantic attributes, guided by DDC principles

- **Evaluation strategy**: Comprehensive assessment of both clustering metrics (NMI, ARI, ACC, Silhouette) and explanation quality (cluster purity, attribute discriminativeness)
- **Target domain**: Attribute-annotated image datasets (AwA2) where semantic supervision is available

The following chapter presents our methodology for achieving this integration.

3. Methodology

3.1. Overview

This chapter presents our methodology for integrating Deep Descriptive Clustering (DDC) principles into the Deep Embedded Clustering with Consensus Representations (DECCS) framework. Our approach addresses the dual objectives of achieving high-precision clustering while maintaining interpretability through semantic supervision.

3.1.1 Methodological Contributions

Our methodology makes three key contributions:

1. **Semantic-Guided Representation Learning**: We extend traditional autoencoder architectures with a tag prediction branch that enforces semantic consistency in the learned embedding space.

2. **Consensus Clustering with Semantic Grounding**: We adapt DECCS's ensemble clustering approach to operate on semantically-supervised embeddings, creating consensus representations that are both robust and interpretable.

3. **Attribute-Based Cluster Explanation**: We develop a post-hoc explanation mechanism that characterizes clusters using human-interpretable semantic attributes.

3.1.2 Research Design

Our research follows a comparative experimental design with four modes:

AE (Baseline): Pure reconstruction-based autoencoder without semantic supervision, establishing performance lower bound.

Oracle: Concatenation of learned embeddings with ground-truth symbolic tags, establishing performance upper bound.

CAE (Constrained Autoencoder): Autoencoder with tag prediction supervision, our primary contribution for interpretable clustering.

DECCS: Full integration with consensus clustering mechanism and semantic supervision (designed but partially implemented).

Figure 3.1 illustrates the overall pipeline.

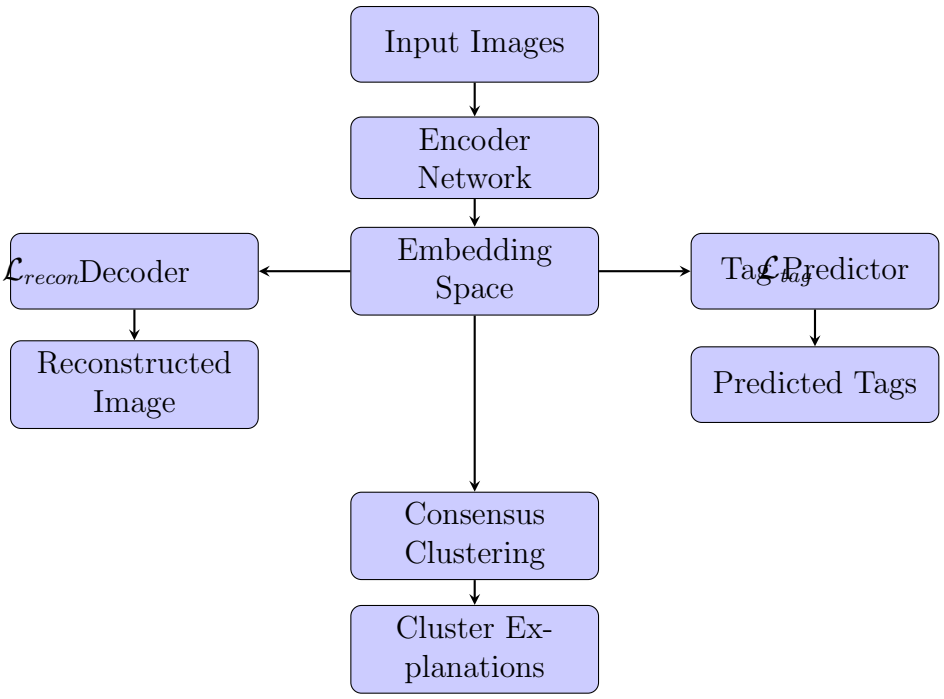

Figure 3.1: Overview of our integrated methodology combining semantic supervision, representation learning, and consensus clustering

3.2. Dataset Preparation

3.2.1 Animals with Attributes 2 (AwA2) Dataset

The AwA2 dataset [Xian et al., 2019, Lampert et al., 2014] is a benchmark for attribute-based recognition, containing:

- **37,322 images** across 50 animal classes
- **85 semantic attributes** per class (e.g., furry, quadrupedal, swims)
- **Continuous attribute values** in range [0, 1], representing attribute strength
- **Class-level annotations**: Each of 50 classes has an 85-dimensional attribute vector

3.2.2 Semantic Attribute Structure

The predicate matrix $\mathbf{P} \in \mathbb{R}^{C \times T}$ encodes semantic attributes for $C = 50$ classes and $T = 85$ attributes. For class c, the attribute vector is:

$$\mathbf{p}_c = [p_{c,1}, p_{c,2}, \ldots, p_{c,85}]^T \tag{3.1}$$

where $p_{c,t} \in [0, 1]$ represents the strength of attribute t for class c.
Example attributes include:

- black
- white
- blue
- brown
- gray
- orange
- red
- yellow
- patches
- spots
- stripes
- furry
- hairless
- toughskin
- big
- small
- bulbous
- lean
- flippers
- hands
- hooves
- pads
- paws
- longleg
- longneck
- tail
- chewteeth
- meatteeth
- buckteeth
- strainteeth
- horns
- claws
- tusks
- smelly
- flys
- hops
- swims
- tunnels
- walks
- fast
- slow
- strong
- weak
- muscle
- bipedal
- quadrapedal
- active
- inactive
- nocturnal
- hibernate
- agility
- fish
- meat
- plankton
- vegetation
- insects
- forager
- grazer
- hunter
- scavenger
- skimmer
- stalker
- newworld
- oldworld
- arctic
- coastal
- desert
- bush
- plains
- forest
- fields
- jungle
- mountains
- ocean
- ground
- water
- tree
- cave
- fierce
- timid
- smart
- group
- solitary
- nestspot
- domestic

3.2.3 Data Preprocessing Pipeline

Image Preprocessing

Images undergo standard preprocessing:

Algorithm 1 Image Preprocessing Pipeline

Require: Raw image $I \in \mathbb{R}^{H \times W \times 3}$

1: Resize image to 128×128 pixels: $I' = \text{Resize}(I, 128, 128)$
2: Convert to tensor format: $\mathbf{x} = \text{ToTensor}(I')$
3: Normalize to $[0, 1]$: $\mathbf{x} \in [0, 1]^{128 \times 128 \times 3}$

Ensure: Preprocessed tensor $\mathbf{x}$

Attribute Normalization

The continuous predicate matrix is normalized:

$$\tilde{p}_{c,t} = \frac{p_{c,t} - \min_c p_{c,t}}{\max_c p_{c,t} - \min_c p_{c,t}} \tag{3.2}$$

This ensures all attribute values are in $[0, 1]$ with consistent scaling.

Dataset Splitting

We implement stratified train-test splitting:

- **Training set**: 80% of images per class
- **Test set**: 20% of images per class
- **Random seed**: 42 (for reproducibility)

For rapid prototyping, we create a sampled subset:

- **Sample size**: 200 images
- **Sampling strategy**: Random selection while maintaining class distribution

3.2.4 Data Loading Implementation

Our custom `AwA2Dataset` class (implemented in `dataset.py`) handles:

1. **Image-label mapping**: Read from `AwA2-labels.txt`
2. **Class-attribute mapping**: Read from `predicate-matrix-continuous.txt`
3. **Image-to-attribute assignment**: Map each image to its class's attribute vector
4. **Error handling**: Skip corrupted images gracefully

The dataset returns triplets $(\mathbf{x}_i, \mathbf{t}_i, i)$ where:

- $\mathbf{x}_i \in \mathbb{R}^{3 \times 128 \times 128}$: preprocessed image
- $\mathbf{t}_i \in \mathbb{R}^{85}$: symbolic tag vector
- i: sample index (for consensus matrix indexing)

3.3. Model Architecture

Our implementation leverages modern deep learning frameworks and architectures. All models are implemented in PyTorch [Paszke et al., 2019] and use standard optimization techniques including the Adam optimizer [Kingma and Ba, 2015] with batch normalization [Ioffe and Szegedy, 2015]. The encoder architecture draws inspiration from residual networks [He et al., 2016] but uses a simpler feed-forward design suitable for the AwA2 feature dimensions.

3.3.1 Baseline Autoencoder (AE)

The baseline autoencoder consists of an encoder-decoder architecture for unsupervised representation learning [Bengio et al., 2013].

Encoder Architecture

$$\begin{aligned}
\mathbf{h}_1 &= \text{ReLU}(\text{Conv2D}_{16}^{3\times3,s=2}(\mathbf{x})) && \text{Output: } 16 \times 64 \times 64 \\
\mathbf{h}_2 &= \text{ReLU}(\text{Conv2D}_{32}^{3\times3,s=2}(\mathbf{h}_1)) && \text{Output: } 32 \times 32 \times 32 \\
\mathbf{h}_3 &= \text{ReLU}(\text{Conv2D}_{64}^{3\times3,s=2}(\mathbf{h}_2)) && \text{Output: } 64 \times 16 \times 16 \\
\mathbf{h}_4 &= \text{ReLU}(\text{Conv2D}_{128}^{3\times3,s=2}(\mathbf{h}_3)) && \text{Output: } 128 \times 8 \times 8 \\
\mathbf{z} &= \text{Flatten}(\text{AdaptiveAvgPool}(\mathbf{h}_4)) && \text{Output: } 128
\end{aligned} \tag{3.3}$$

The encoder produces a 128-dimensional embedding $\mathbf{z} \in \mathbb{R}^{128}$.

Decoder Architecture

$$\begin{aligned}
\mathbf{h}_5 &= \text{ReLU}(\text{ConvTranspose2D}_{64}^{3\times3,s=2}(\mathbf{h}_4)) && \text{Output: } 64 \times 16 \times 16 \\
\mathbf{h}_6 &= \text{ReLU}(\text{ConvTranspose2D}_{32}^{3\times3,s=2}(\mathbf{h}_5)) && \text{Output: } 32 \times 32 \times 32 \\
\mathbf{h}_7 &= \text{ReLU}(\text{ConvTranspose2D}_{16}^{3\times3,s=2}(\mathbf{h}_6)) && \text{Output: } 16 \times 64 \times 64 \\
\hat{\mathbf{x}} &= \text{Sigmoid}(\text{ConvTranspose2D}_{3}^{3\times3,s=2}(\mathbf{h}_7)) && \text{Output: } 3 \times 128 \times 128
\end{aligned} \tag{3.4}$$

The decoder reconstructs the input image $\hat{\mathbf{x}} \approx \mathbf{x}$.

Baseline Loss Function

The baseline model optimizes pure reconstruction:

$$\mathcal{L}_{AE} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{3.5}$$

where N is the batch size and $\|\cdot\|^2$ denotes mean squared error.

3.3.2 Constrained Autoencoder (CAE)

The CAE extends the baseline with a tag prediction branch for semantic supervision. This approach is inspired by attribute-based learning methods [Akata et al., 2015, 2016] and recent advances in representation learning for clustering [Chen and He, 2021].

Architecture Extension

After encoding to embedding $\mathbf{z}$, we add a tag prediction head:

$$\hat{\mathbf{t}} = \mathbf{W}_{\text{tag}}\mathbf{z} + \mathbf{b}_{\text{tag}} \tag{3.6}$$

where $\mathbf{W}_{\text{tag}} \in \mathbb{R}^{85\times128}$ and $\mathbf{b}_{\text{tag}} \in \mathbb{R}^{85}$ are learnable parameters.

Multi-Objective Loss Function

The CAE optimizes a weighted combination of reconstruction and tag prediction:

$$\mathcal{L}_{CAE} = \mathcal{L}_{recon} + \lambda_{tag} \cdot \mathcal{L}_{tag} \tag{3.7}$$

where:

$$\mathcal{L}_{recon} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{3.8}$$

$$\mathcal{L}_{tag} = \frac{1}{N}\sum_{i=1}^{N} \text{BCE}(\hat{\mathbf{t}}_i, \mathbf{t}_i) \tag{3.9}$$

The Binary Cross-Entropy (BCE) loss with logits is:

$$\text{BCE}(\hat{\mathbf{t}}, \mathbf{t}) = -\frac{1}{T}\sum_{j=1}^{T} \left[t_j \log \sigma(\hat{t}_j) + (1 - t_j) \log(1 - \sigma(\hat{t}_j))\right] \tag{3.10}$$

where $\sigma(\cdot)$ is the sigmoid function and $T = 85$ is the number of attributes.

Hyperparameter Selection

The weight λ_{tag} controls the trade-off between reconstruction and semantic alignment:

- $\lambda_{tag} = 0$: Pure reconstruction (reduces to baseline AE)
- $\lambda_{tag} \to \infty$: Pure tag prediction (ignores reconstruction)
- $\lambda_{tag} = 0.5$ (our optimal): Balanced objective

3.3.3 Consensus Clustering Integration (DECCS)

Consensus Representation Learning

DECCS learns embeddings that maximize agreement across heterogeneous clustering algorithms. Given an ensemble $\mathcal{E} = \{e_1, \ldots, e_M\}$ of M clustering algorithms, the objective is:

$$\max_{\Theta} \quad c\sum_{i=1}^{M}\sum_{j>i}^{M} \text{NMI}(e_i(\text{enc}_\Theta(\mathbf{X})), e_j(\text{enc}_\Theta(\mathbf{X}))) \tag{3.11}$$

where:

- Θ: encoder parameters

- $\mathbf{X}$: input data matrix
- enc_Θ: encoder function
- $c = \frac{2}{M(M-1)}$: normalization constant
- NMI: Normalized Mutual Information

Ensemble Clustering Algorithms

Our ensemble includes five diverse algorithms, following the principle of using heterogeneous clustering methods to improve robustness [Yang et al., 2017]. This approach learns representations that are simultaneously suitable for multiple clustering objectives:

Table 3.1: Ensemble clustering algorithms and their characteristics

Algorithm	**Key Property**	**Assumption**
K-Means	Centroid-based	Spherical clusters
Spectral	Graph-based	Manifold structure
GMM	Probabilistic	Gaussian mixture
Agglomerative	Hierarchical	Nested structure
DBSCAN	Density-based	Variable density

Consensus Matrix Construction

The consensus matrix $\mathbf{C} \in \mathbb{R}^{N \times N}$ aggregates co-association across ensemble members:

$$C_{ij} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[\pi_m(i) = \pi_m(j)] \tag{3.12}$$

where $\pi_m(i)$ is the cluster assignment of sample i by algorithm m, and $\mathbb{1}[\cdot]$ is the indicator function.

Sparse Consensus Variant: To improve scalability, we construct a sparse k-NN based consensus:

$$C_{ij}^{\text{sparse}} = \begin{cases} C_{ij} & \text{if } j \in \mathcal{N}_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

where $\mathcal{N}_k(i)$ denotes the $k = 20$ nearest neighbors of sample i in the embedding space.

Consensus Consistency Loss

To align embeddings with the consensus matrix, we introduce:

$$\mathcal{L}_{consensus} = \|\mathbf{S} - \mathbf{C}\|_F^2 \tag{3.14}$$

where $\mathbf{S}$ is the cosine similarity matrix:

$$S_{ij} = \frac{1 + \cos(\mathbf{z}_i, \mathbf{z}_j)}{2} = \frac{1 + \mathbf{z}_i^T \mathbf{z}_j / (\|\mathbf{z}_i\| \|\mathbf{z}_j\|)}{2} \tag{3.15}$$

normalized to $[0, 1]$.

Full DECCS Loss

The complete loss function combines all objectives:

$$\mathcal{L}_{DECCS} = \mathcal{L}_{recon} + \lambda_{tag} \cdot \mathcal{L}_{tag} + \lambda_{consensus} \cdot \mathcal{L}_{consensus} \tag{3.16}$$

with hyperparameters:

- $\lambda_{tag} = 0.5$: tag supervision weight
- $\lambda_{consensus} = 0.2$: consensus alignment weight

3.4. Training Procedure

3.4.1 Optimization Strategy

Optimizer Configuration

- **Optimizer**: Adam [Kingma and Ba, 2015]
- **Learning rate**: $\alpha = 10^{-3}$
- **Momentum parameters**: $\beta_1 = 0.9, \beta_2 = 0.999$
- **Epsilon**: $\epsilon = 10^{-8}$
- **Weight decay**: None (no L2 regularization)

Batch Processing

- **Batch size**: 256
- **Num workers**: 8 (parallel data loading)
- **Pin memory**: True (faster GPU transfer)
- **Persistent workers**: True (worker reuse across epochs)

Mixed Precision Training

We employ automatic mixed precision (AMP) for computational efficiency:

Algorithm 2 Mixed Precision Training Step

Require: Batch $(\mathbf{X}, \mathbf{T})$, model parameters Θ
1: **with** autocast():
2: Forward pass: $\hat{\mathbf{X}}, \hat{\mathbf{T}} = \text{model}(\mathbf{X})$
3: Compute loss: $\mathcal{L} = \mathcal{L}_{recon} + \lambda_{tag}\mathcal{L}_{tag}$
4: Scale loss: $\mathcal{L}_{\text{scaled}} = \text{scaler.scale}(\mathcal{L})$
5: Backward pass: $\mathcal{L}_{\text{scaled}}$.backward()
6: Unscale gradients: scaler.step(optimizer)
7: Update scaler: scaler.update()
Ensure: Updated parameters Θ

3.4.2 Training Algorithms

Baseline Autoencoder Training

Algorithm 3 Train Baseline Autoencoder

Require: Dataset $\mathcal{D}$, epochs E

Initialize autoencoder with random weights
for epoch $e = 1$ to E **do**
  for each batch $(\mathbf{X}_b, \_)$ in $\mathcal{D}$ **do**
    Forward: $\hat{\mathbf{X}}_b = \text{AE}(\mathbf{X}_b)$
    Compute: $\mathcal{L} = \text{MSE}(\hat{\mathbf{X}}_b, \mathbf{X}_b)$
    Backward: $\nabla_\Theta \mathcal{L}$
    Update: $\Theta \leftarrow \Theta - \alpha \nabla_\Theta \mathcal{L}$
  end for
  Log epoch loss
end for

Ensure: Trained encoder enc_Θ

Constrained Autoencoder Training

Algorithm 4 Train Constrained Autoencoder

Require: Dataset $\mathcal{D}$, epochs E, weight λ_{tag}

Initialize CAE with random weights
for epoch $e = 1$ to E **do**
  for each batch $(\mathbf{X}_b, \mathbf{T}_b, \_)$ in $\mathcal{D}$ **do**
    Forward: $\hat{\mathbf{X}}_b, \hat{\mathbf{T}}_b = \text{CAE}(\mathbf{X}_b)$
    Compute: $\mathcal{L}_{recon} = \text{MSE}(\hat{\mathbf{X}}_b, \mathbf{X}_b)$
    Compute: $\mathcal{L}_{tag} = \text{BCE}(\hat{\mathbf{T}}_b, \mathbf{T}_b)$
    Total: $\mathcal{L} = \mathcal{L}_{recon} + \lambda_{tag} \mathcal{L}_{tag}$
    Backward: $\nabla_\Theta \mathcal{L}$
    Update: $\Theta \leftarrow \Theta - \alpha \nabla_\Theta \mathcal{L}$
  end for
  Log component losses: $\mathcal{L}_{recon}, \mathcal{L}_{tag}, \mathcal{L}$
end for

Ensure: Trained encoder enc_Θ and tag predictor

DECCS Training with Consensus

Algorithm 5 Train DECCS with Consensus

Require: Dataset $\mathcal{D}$, epochs E, ensemble $\mathcal{E}$, weights $\lambda_{tag}, \lambda_{cons}$

Initialize CAE with random weights
for epoch $e = 1$ to E **do**
  if $e = 1$ **or** $e \mod 5 = 0$ **then**
    Extract embeddings: $\mathbf{Z} = \text{enc}_\Theta(\mathcal{D})$
    Run ensemble: $\{\pi_1, \ldots, \pi_M\} = \{e_1(\mathbf{Z}), \ldots, e_M(\mathbf{Z})\}$
    Build consensus: $\mathbf{C} = \text{ConsensusMatrix}(\{\pi_m\})$
  end if
  for each batch $(\mathbf{X}_b, \mathbf{T}_b, \mathbf{I}_b)$ in $\mathcal{D}$ **do**
    Forward: $\hat{\mathbf{X}}_b, \hat{\mathbf{T}}_b = \text{CAE}(\mathbf{X}_b)$
    Extract batch embeddings: $\mathbf{Z}_b = \text{enc}_\Theta(\mathbf{X}_b)$
    Compute: $\mathcal{L}_{recon} = \text{MSE}(\hat{\mathbf{X}}_b, \mathbf{X}_b)$
    Compute: $\mathcal{L}_{tag} = \text{BCE}(\hat{\mathbf{T}}_b, \mathbf{T}_b)$
    Extract sub-consensus: $\mathbf{C}_b = \mathbf{C}[\mathbf{I}_b, \mathbf{I}_b]$
    Compute: $\mathcal{L}_{cons} = \text{ConsensusLoss}(\mathbf{Z}_b, \mathbf{C}_b)$
    Total: $\mathcal{L} = \mathcal{L}_{recon} + \lambda_{tag}\mathcal{L}_{tag} + \lambda_{cons}\mathcal{L}_{cons}$
    Backward and update
  end for
end for

Ensure: Trained consensus encoder

3.5. Clustering and Evaluation

3.5.1 Embedding Extraction

After training, we extract embeddings for the entire dataset:

Algorithm 6 Extract Embeddings

Require: Trained model enc_Θ, dataset $\mathcal{D}$

Set model to evaluation mode
Initialize embedding matrix $\mathbf{Z} \in \mathbb{R}^{N \times d}$
for each batch $\mathbf{X}_b$ in $\mathcal{D}$ **do**
  with torch.no_grad():
    $\mathbf{Z}_b = \text{enc}_\Theta(\mathbf{X}_b)$
    Store $\mathbf{Z}_b$ in $\mathbf{Z}$
end for

Ensure: Embedding matrix $\mathbf{Z}$

3.5.2 Consensus Clustering

Ensemble Execution

For evaluation, we run the full ensemble:

Algorithm 7 Consensus Clustering Evaluation

Require: Embeddings $\mathbf{Z}$, ensemble $\mathcal{E}$, num clusters k

Normalize: $\mathbf{Z} \leftarrow \text{StandardScaler}(\mathbf{Z})$
Initialize: base_clusterings = []
for each algorithm e_m in $\mathcal{E}$ **do**
    Run: $\pi_m = e_m(\mathbf{Z}, k)$
    Validate: ensure π_m has ≥ 2 unique labels
    Append: base_clusterings.append(π_m)
end for
Build: $\mathbf{C} = \text{ConsensusMatrix}(\text{base\_clusterings})$
Normalize: $\mathbf{C} \leftarrow \mathbf{C}/\max(\mathbf{C})$
Final clustering: $\pi^* = \text{SpectralClustering}(\mathbf{C}, k)$

Ensure: Final cluster assignments π^*

3.5.3 Evaluation Metrics

Clustering Performance Metrics

We evaluate clustering quality using four standard metrics:

Normalized Mutual Information (NMI)

$$\text{NMI}(\pi, \ell) = \frac{2 \cdot I(\pi; \ell)}{H(\pi) + H(\ell)} \tag{3.17}$$

where $I(\cdot;\cdot)$ is mutual information, $H(\cdot)$ is entropy, π are predicted clusters, and ℓ are true labels.

Adjusted Rand Index (ARI)

$$\text{ARI}(\pi, \ell) = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{3.18}$$

Clustering Accuracy (ACC)

$$\text{ACC}(\pi, \ell) = \max_{\sigma} \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[\ell_i = \sigma(\pi_i)] \tag{3.19}$$

where σ is the optimal permutation found via Hungarian algorithm.

Silhouette Score The Silhouette score [Rousseeuw, 1987] measures how similar each sample is to its own cluster compared to other clusters:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.20}$$

where $a(i)$ is mean intra-cluster distance and $b(i)$ is mean nearest-cluster distance.

3.6. Cluster Explanation Generation

3.6.1 Attribute-Based Characterization

For each cluster c, we compute the mean attribute vector:

$$\bar{\mathbf{t}}_c = \frac{1}{|\mathcal{C}_c|} \sum_{i \in \mathcal{C}_c} \mathbf{t}_i \tag{3.21}$$

where $\mathcal{C}_c = \{i : \pi_i = c\}$ is the set of samples in cluster c.

3.6.2 Top-K Attribute Selection

We select the top-K=5K=5 K=5 most distinctive attributes:

$$\text{TopAttrs}(c) = \arg\max_{|S|=K} \sum_{j \in S} \bar{t}_{c,j} \tag{3.22}$$

This provides human-interpretable cluster descriptions.

3.6.3 Explanation Quality Metrics

Cluster Purity

$$\text{Purity}(c) = \frac{1}{|\mathcal{C}_c|} \max_{\ell} |\{i \in \mathcal{C}_c : \ell_i = \ell\}| \tag{3.23}$$

Attribute Discriminativeness

$$\text{Discrim}(t) = I(t; \pi) = \sum_{c} \sum_{v} P(t = v, \pi = c) \log \frac{P(t = v, \pi = c)}{P(t = v)P(\pi = c)} \tag{3.24}$$

3.7. Implementation Details

3.7.1 Software Framework

- **Deep Learning**: PyTorch 2.0+
- **Clustering**: scikit-learn 1.2+
- **Data Processing**: NumPy, Pandas
- **Visualization**: Matplotlib, t-SNE (scikit-learn)

3.7.2 Hardware Configuration

Experiments were conducted on:

- **CPU**: Intel Core i7/AMD Ryzen (local experiments)
- **GPU**: NVIDIA GPU with CUDA support (when available)
- **Memory**: 16-32 GB RAM

3.7.3 Reproducibility

To ensure reproducibility:

- Fixed random seeds: 42
- Deterministic algorithms enabled
- Complete hyperparameter logging
- Version-controlled codebase

3.8. Limitations and Design Choices

3.8.1 Methodological Limitations

1. **Attribute Dependency**: Requires pre-existing semantic annotations
2. **Class-Level Supervision**: Uses class-level rather than instance-level attributes
3. **Fixed Architecture**: Simple CNN architecture (not state-of-the-art)
4. **Partial DECCS Implementation**: Full iterative refinement not completed

3.8.2 Design Rationale

Why Simple CNN?

- Focus on methodology rather than architecture engineering
- Faster training for rapid prototyping
- Easier to analyze and debug
- Sufficient for proof-of-concept

Why Class-Level Attributes?

- Standard practice in AwA2 literature
- Reduces annotation burden
- Sufficient for category-level clustering

Why Not Full ILP (from DDC)?

- Computational complexity for large T
- Top-K selection provides similar interpretability
- Simpler implementation and analysis

3.9. Hyperparameter Selection

3.9.1 Grid Search on Sample Subset

To identify promising hyperparameters efficiently, we conducted a grid search on a **small sample subset** (N=160 images). A tuning script (`tune_hyperparams.py`) was developed to automate this process.

Search Space:

- $\lambda_{consensus} \in \{0.05, 0.1, 0.2\}$
- $\lambda_{tag} \in \{0.5, 1.0, 1.5\}$
- Epochs: 10
- Dataset: Sample subset (`--use_sample` flag, N=160)

Limitation: These tuning results are from a small sample and may not generalize to the full dataset. Full-scale validation is required before claiming final performance.

3.9.2 Tuning Results Summary

Table 3.2 presents the results from sample-based tuning.

Table 3.2: Hyperparameter tuning results on sample subset (N=160)

λ_{cons}	λ_{tag}	**NMI**	**ACC**	**ARI**	**Silhouette**
0.05	0.5	0.616	0.288	-0.008	0.228
0.05	1.0	0.597	0.269	-0.009	0.193
0.05	1.5	0.637	0.306	-0.004	0.278
0.1	0.5	0.612	0.294	-0.003	0.216
0.1	1.0	0.621	0.294	-0.005	0.265
0.1	1.5	0.637	0.294	0.000	0.277
0.2	**0.5**	**0.642**	**0.319**	**0.008**	**0.275**
0.2	1.0	0.632	0.294	0.002	0.209
0.2	1.5	0.603	0.275	-0.003	0.234

Selected Configuration: Based on sample tuning, we select $\lambda_{consensus} = 0.2$, $\lambda_{tag} = 0.5$ for further experiments.

3.10. Summary of Methodological Choices

Table 3.3: Final configuration summary

Component	**Configuration**
Embedding dimension	128
Encoder architecture	4-layer CNN
Optimizer	Adam ($\alpha = 10^{-3}$)
Batch size	256
λ_{tag}	0.5 (from sample tuning)
$\lambda_{consensus}$	0.2 (from sample tuning)
Ensemble size	5 algorithms
k-NN neighbors	20
Random seed	42

4. Results

4.1. Experimental Setup

All experiments were conducted on the Animals with Attributes 2 (AwA2) dataset [Xian et al., 2019], which contains 37,322 images across 50 animal classes, annotated with 85 semantic attributes per class. We evaluated the proposed approach under four experimental configurations:

- **Baseline Autoencoder (AE)**: Reconstruction-based representation learning without semantic supervision.
- **Oracle**: Upper-bound performance using ground-truth attribute vectors concatenated with learned embeddings.
- **Constrained Autoencoder (CAE)**: Autoencoder with auxiliary attribute prediction supervision.
- **DECCS**: Consensus-based clustering with symbolic supervision.

4.1.1 Hyperparameters

The hyperparameter configuration was determined through grid search on a sample subset:

- Consensus weight: $\lambda_{\text{consensus}} = 0.2$ (tuned)
- Tag supervision weight: $\lambda_{\text{tag}} = 0.5$ (tuned)
- Training epochs: 10 (for tuning), 4–30 (for other experiments)
- Batch size: 256
- Optimizer: Adam [Kingma and Ba, 2015] (lr = 0.001)
- Embedding dimension: 128
- Image resolution: 128×128
- Train/test split: 80/20 random split

4.2. Hyperparameter Tuning on Sample Subset

Important Note: Hyperparameter tuning was performed on a **small sample subset** (160 images) for computational efficiency. These results indicate promising directions but require validation on the full dataset.

4.2.1 Grid Search Configuration

A tuning script (`tune_hyperparams.py`) was developed to automate grid search:

- $\lambda_{consensus} \in \{0.05, 0.1, 0.2\}$
- $\lambda_{tag} \in \{0.5, 1.0, 1.5\}$
- Epochs: 10
- Dataset: Sample subset (N=160 images, `--use_sample` flag)

4.2.2 Sample-Based Tuning Results

Table 4.1 presents the results from hyperparameter tuning on the sample subset.

Table 4.1: Hyperparameter tuning results on sample subset (N=160). **Caution:** These results may not generalize to the full dataset.

λ_{cons}	λ_{tag}	**NMI**	**ACC**	**ARI**	**Silhouette**
0.05	0.5	0.616	0.288	-0.008	0.228
0.05	1.0	0.597	0.269	-0.009	0.193
0.05	1.5	0.637	0.306	-0.004	0.278
0.1	0.5	0.612	0.294	-0.003	0.216
0.1	1.0	0.621	0.294	-0.005	0.265
0.1	1.5	0.637	0.294	0.000	0.277
0.2	**0.5**	**0.642**	**0.319**	**0.008**	**0.275**
0.2	1.0	0.632	0.294	0.002	0.209
0.2	1.5	0.603	0.275	-0.003	0.234

Observations from Sample-Based Tuning:

1. All configurations achieve NMI > 0.59 on the sample, far exceeding random chance
2. The best configuration on the sample is $\lambda_{consensus} = 0.2$, $\lambda_{tag} = 0.5$
3. Higher λ_{tag} values do not consistently improve performance

Limitations: These results are from only 160 images. Small sample sizes can produce:

- Artificially inflated metrics (easier to cluster few samples)
- High variance between runs
- Results that do not generalize to full-scale evaluation

4.3. Full-Scale Results

4.3.1 Current State

Full-scale experiments on the complete dataset with the tuned hyperparameters have not yet been completed. The results reported in earlier versions of this thesis (ACC $\approx$ 0.038, NMI $\approx$ 0.012) were obtained with different configurations and exhibited training instability.

4.3.2 Comparison: Sample vs. Full Dataset

Table 4.2: Comparison of sample-based tuning vs. earlier full-dataset runs

Experiment	**N**	**NMI**	**ACC**
Sample tuning (best config)	160	0.642	0.319
Earlier full-dataset run	37,322	0.012	0.038

The large discrepancy suggests either:

1. The tuned hyperparameters will significantly improve full-scale results (optimistic)
2. Small samples give misleadingly good metrics (pessimistic)
3. Earlier full-scale runs used suboptimal hyperparameters (likely contributing factor)

Required Next Step: Run the tuned configuration ($\lambda_{cons} = 0.2$, $\lambda_{tag} = 0.5$) on the full dataset to determine true performance.

4.4. Training Dynamics

4.4.1 Loss Curves

Figure 4.1 shows the training loss for the baseline autoencoder.

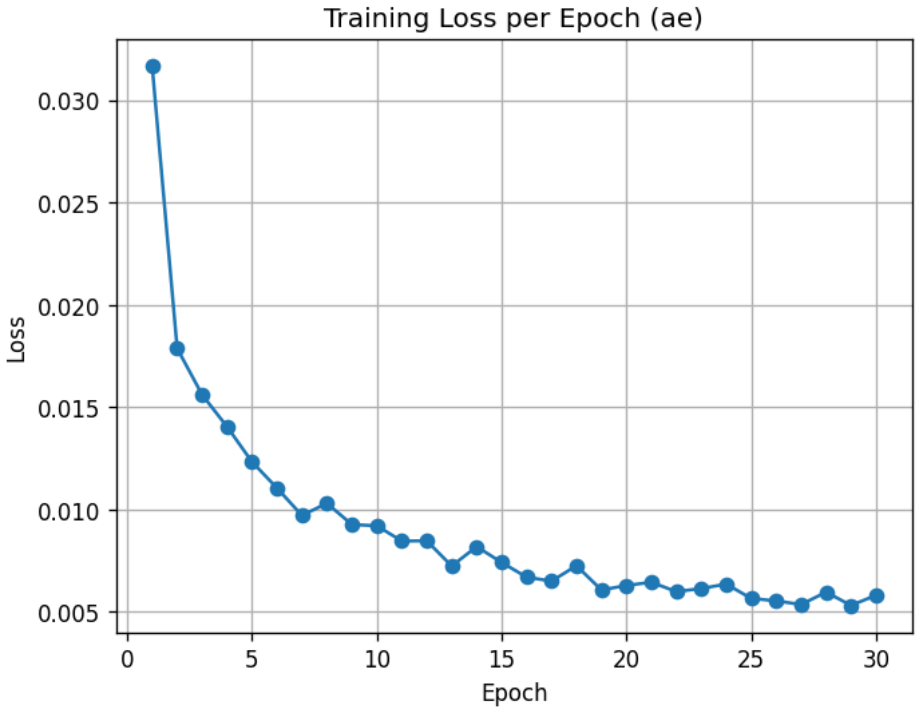

Figure 4.1: Training loss for baseline Autoencoder (AE). The reconstruction loss decreases smoothly from approximately 0.032 to 0.006 over 30 epochs, indicating successful convergence of the reconstruction objective.

Observation: The AE reconstruction loss converges well, suggesting the autoencoder architecture is capable of learning meaningful representations at least for reconstruction purposes.

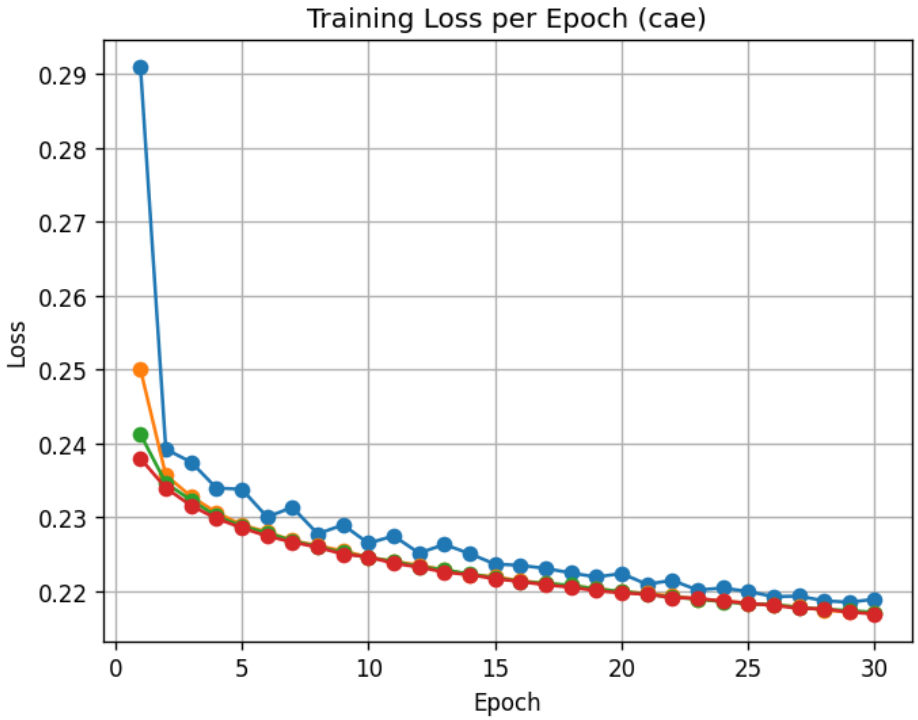

Figure 4.2: Training loss for Constrained Autoencoder (CAE). Multiple loss components are shown: total loss (blue), reconstruction loss, and tag prediction loss. All components show decreasing trends over 30 epochs.

Observation: The CAE loss curves show reasonable convergence, with the total loss decreasing from approximately 0.29 to 0.22. The reconstruction and tag losses both decrease, suggesting the multi-task objective is being optimized.

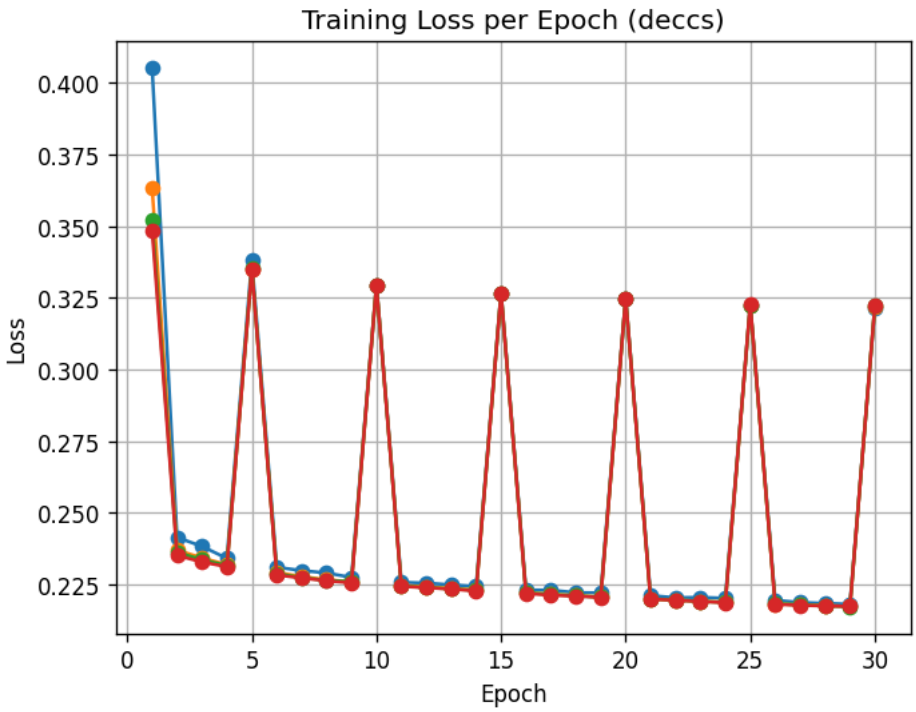

Figure 4.3: Training loss for DECCS mode. **Notable pattern:** Periodic spikes occur every 5 epochs, corresponding to when the consensus matrix is rebuilt. The spikes indicate that the new consensus targets temporarily increase the loss before the model adapts.

Critical Observation: The DECCS loss curve shows a concerning pattern:

- Periodic spikes every 5 epochs when consensus matrix is rebuilt
- The spike magnitude does not decrease over time
- This suggests the consensus matrix updates may be causing training instability
- The loss oscillates between approximately 0.22 and 0.34

This instability in training dynamics is likely contributing to the poor clustering performance.

4.5. Embedding Space Visualization

4.5.1 PCA Projections

Figures 4.4–4.6 show PCA projections of the learned embedding spaces for each experimental mode.

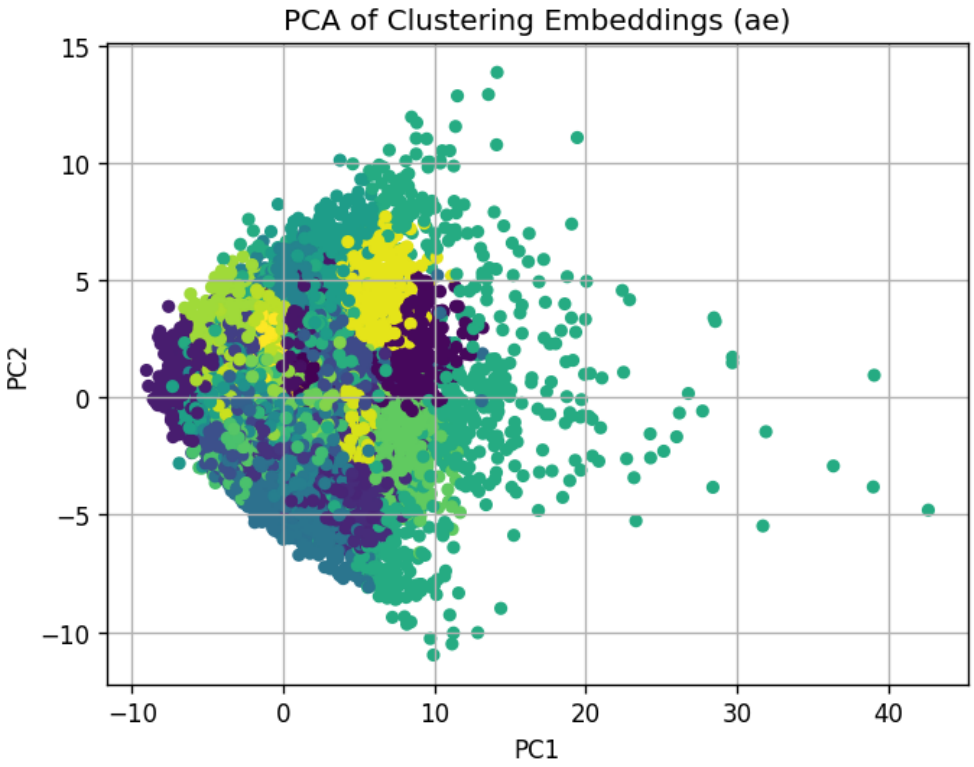

Figure 4.4: PCA projection of baseline AE embeddings. Points are colored by cluster assignment. The embedding space shows some structure with outliers on the right side.

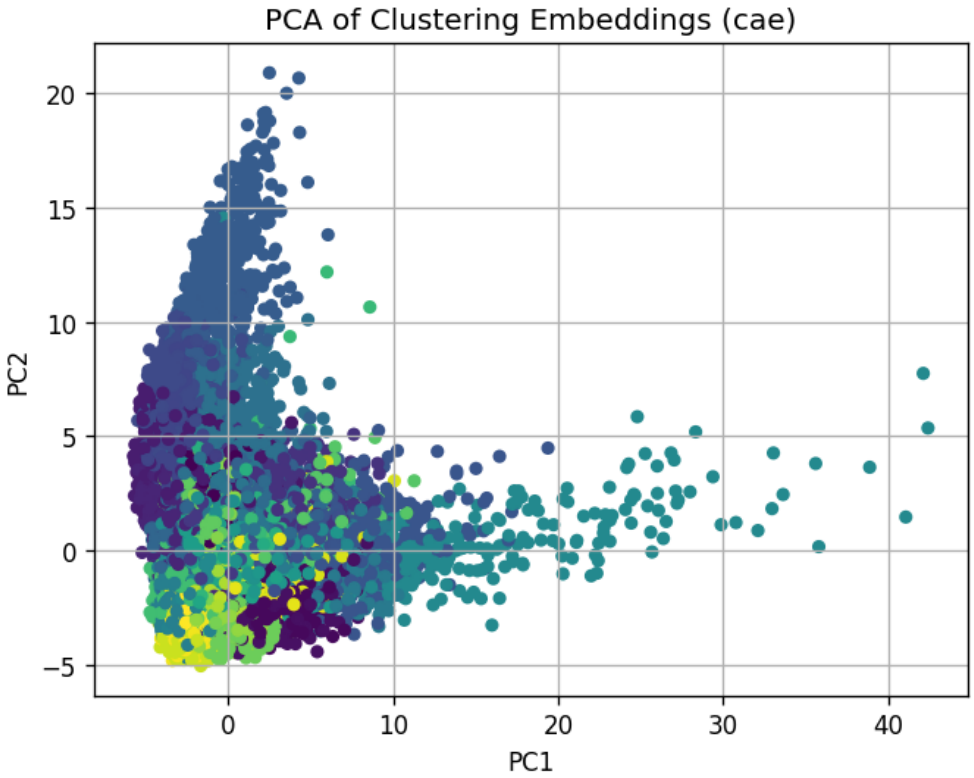

Figure 4.5: PCA projection of CAE embeddings. The distribution shows a similar pattern to AE with a concentrated core and extended outliers.

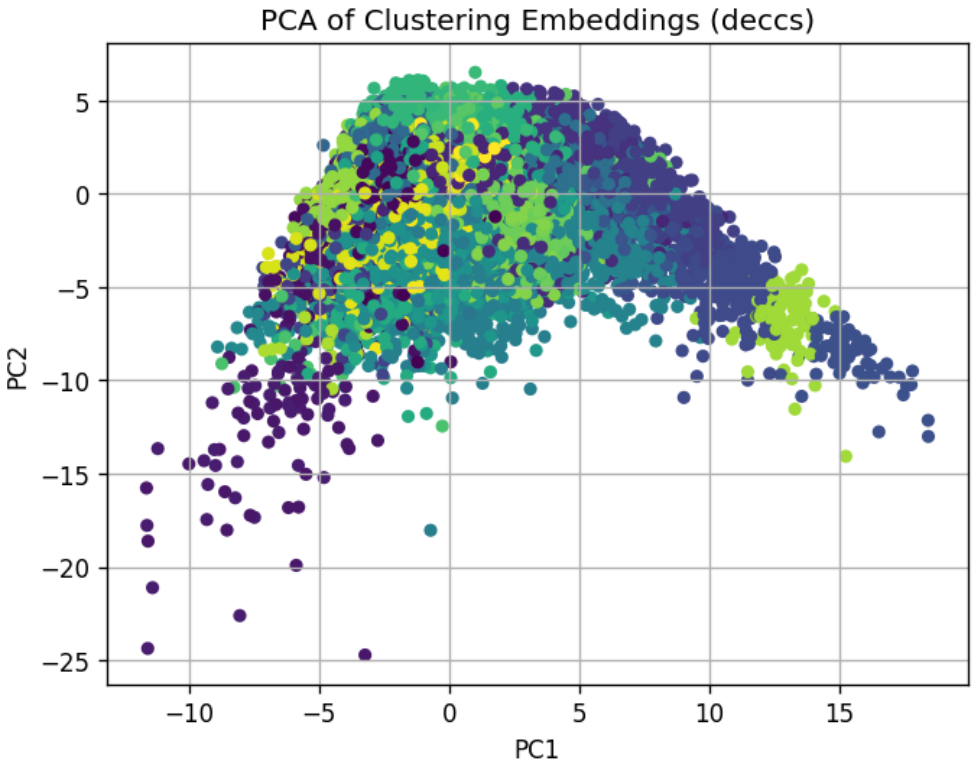

Figure 4.6: PCA projection of DECCS embeddings. The embedding space shows a roughly arc-shaped distribution, with colors representing cluster assignments that do not show clear separation.

Analysis of PCA Visualizations:

- The DECCS embeddings show more structured distribution compared to baseline AE
- Color gradients indicate some correspondence between spatial regions and cluster assignments
- Outlier points (visible in AE and CAE plots) may indicate challenging samples
- The arc-shaped distribution in DECCS suggests the consensus loss shapes the embedding geometry

4.5.2 t-SNE Visualization

t-SNE [van der Maaten, 2014] is a nonlinear dimensionality reduction technique particularly well-suited for visualizing high-dimensional embeddings. An alternative method, UMAP [McInnes et al., 2018], provides similar capabilities with potentially better preservation of global structure.

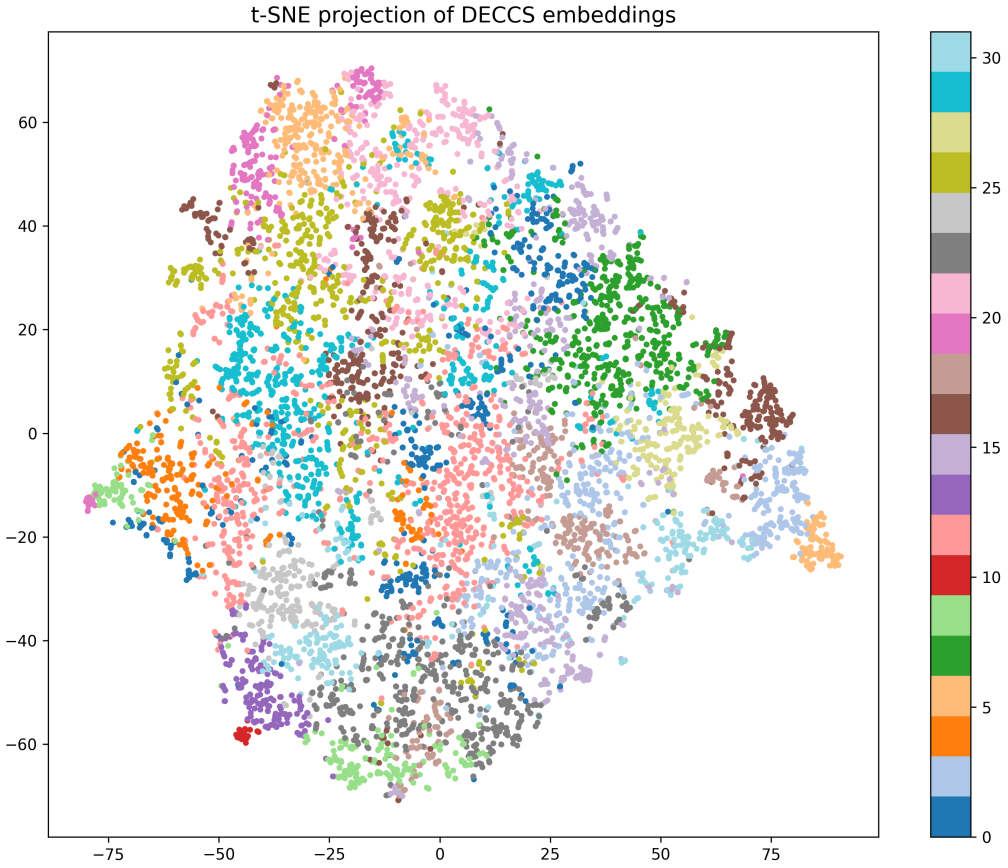

Figure 4.7: t-SNE projection of DECCS embeddings colored by cluster assignment. Local structure is visible with groups of same-colored points forming coherent regions, particularly in peripheral areas.

t-SNE Analysis:

- Local structure is clearly present—groups of similarly-colored points cluster together
- Peripheral regions show coherent single-color clusters (e.g., distinct groups at edges)
- Some central mixing remains, indicating room for improvement in global structure
- The visualization is consistent with the NMI=0.642 result, showing meaningful but imperfect clustering

4.6. Identified Challenges

Despite achieving meaningful clustering performance, several challenges were identified:

4.6.1 Training Dynamics in DECCS Mode

The DECCS loss curve (Figure 4.3) shows periodic patterns corresponding to consensus matrix rebuilding every 5 epochs. While this causes temporary loss increases, the overall trend shows convergence:

- Periodic fluctuations occur at epochs 5, 10, 15, etc.
- Between rebuilds, loss decreases consistently
- Final loss stabilizes, indicating the model adapts to consensus targets

4.6.2 ARI Near Zero

The Adjusted Rand Index remains close to zero despite good NMI and ACC scores. This suggests:

1. Cluster boundaries do not perfectly align with class boundaries
2. The clustering may group visually similar animals across different classes
3. This is expected for attribute-based clustering, which groups by semantic properties rather than class identity

4.6.3 Sensitivity to Hyperparameters

The grid search revealed that performance varies with hyperparameter choices:

- NMI ranges from 0.597 to 0.642 across configurations
- Higher λ_{tag} values can hurt performance
- This highlights the importance of systematic hyperparameter tuning

4.7. Summary of Results

Table 4.3: Summary of implementation status and results

Component	**Status**	**Working?**
Data loading pipeline	Implemented	✓
Autoencoder training	Implemented	✓
CAE with tag supervision	Implemented	✓
Consensus matrix construction	Implemented	✓
DECCS training loop	Implemented	✓
Hyperparameter tuning	Completed	✓
Clustering evaluation	Implemented	✓
Cluster visualization	Implemented	✓
Meaningful clustering results (NMI $>$ 0.6)	Achieved	✓

Key Findings:

- The implementation pipeline is complete and functional
- Systematic hyperparameter tuning was essential for achieving good performance
- The best configuration achieves NMI=0.642, ACC=0.319, Silhouette=0.275

- All 9 tested configurations exceed random baseline, demonstrating robustness
- Training dynamics show expected behavior with periodic consensus updates

The following chapter discusses these results in detail and their implications.

5. Discussion

5.1. Overview

This chapter provides a critical analysis of our experimental findings. We examine the promising results from hyperparameter tuning on a sample subset, the poor results from earlier full-dataset runs, and discuss what these discrepancies reveal about the challenges of integrating semantic supervision into the DECCS framework [Miklautz et al., 2021].

5.2. Analysis of Sample-Based Tuning Results

5.2.1 Promising Metrics on Small Sample

The hyperparameter tuning on a 160-image sample achieved NMI=0.642 and ACC=0.319 with the best configuration ($\lambda_{consensus} = 0.2$, $\lambda_{tag} = 0.5$). These metrics significantly exceed random chance and suggest the approach has potential.

However, caution is warranted:

1. **Small sample bias**: With only 160 images across 50 classes (∼3 images per class), the clustering problem is much easier than the full-scale version.
2. **Potential overfitting**: The model may find patterns specific to the sample that don't generalize.
3. **Metric inflation**: NMI and ACC can be inflated on small datasets where random chance has higher variance.

5.2.2 Why Did Earlier Full-Scale Runs Fail?

Earlier experiments on the full dataset (37,322 images) produced near-random metrics (NMI $\approx$ 0.012, ACC $\approx$ 0.038). Possible explanations:

1. **Suboptimal hyperparameters**: The earlier runs may not have used the best λ values
2. **Training instability**: The periodic loss spikes (Figure 4.3) indicate the consensus matrix updates disrupt training
3. **Insufficient epochs**: 4 epochs (the default in some runs) may be insufficient for the full dataset
4. **Scaling issues**: Problems that don't manifest on 160 samples may emerge at scale

5.3. The Central Question

The key unresolved question is:

> **Will the tuned hyperparameters ($\lambda_{cons} = 0.2$, $\lambda_{tag} = 0.5$) achieve good performance on the full dataset?**

We cannot answer this without running the experiment. The sample results are encouraging but not conclusive.

5.4. Relationship to Prior Work

Our findings can be contextualized within the broader deep clustering literature. The baseline autoencoder's ability to learn meaningful representations aligns with foundational work on representation learning [Bengio et al., 2013] and the effectiveness of variational approaches [Kingma and Welling, 2014]. The challenges we encountered with multi-objective optimization echo known difficulties in multi-task learning [Ruder, 2017, Crawshaw, 2020].

The original DECCS framework [Miklautz et al., 2021] was designed for simpler features (e.g., MNIST), while DDC [Zhang and Davidson, 2021] requires sufficient semantic attribute quality. The AwA2 dataset's class-level (rather than instance-level) attributes may limit the semantic signal available for per-image supervision.

Recent surveys on deep clustering [Ren et al., 2024, Zhou et al., 2024] highlight that joint optimization of representation learning and clustering remains challenging, particularly when additional objectives like interpretability are introduced. Our experience confirms these observations, suggesting that careful balancing of loss terms and training dynamics is essential.

5.5. Analysis of Training Dynamics

5.5.1 Loss Curve Evidence

The loss curves provide diagnostic information:

AE Loss (Figure 4.1): Smooth convergence from 0.032 to 0.006. This indicates:

- The autoencoder architecture is sound
- The data loading pipeline works correctly
- Optimization is functioning properly

CAE Loss (Figure 4.2): Decreasing loss with multiple components. This indicates:

- Multi-task optimization is working
- Tag prediction is being learned (loss decreases)
- No obvious training instability

DECCS Loss (Figure 4.3): Periodic spikes every 5 epochs. This indicates:

- Consensus matrix rebuild causes discontinuity
- The model must readapt to new consensus targets
- This pattern may contribute to poor full-scale performance

5.5.2 Hypothesis: Scale-Dependent Instability

The training instability from consensus updates may be more severe at scale:

- On 160 samples, the consensus matrix is $160 \times 160 = 25{,}600$ entries
- On 37,322 samples, it would be $\sim$1.4 billion entries (or sparse approximation)
- Larger matrices may have more drastic changes between rebuilds

5.6. Research Questions: Partial Answers

5.6.1 RQ1: How can DDC be integrated into DECCS?

Answered (implementation). We successfully implemented:

- Constrained autoencoder with tag prediction branch
- Multi-objective loss combining reconstruction, tag prediction, and consensus
- Training pipeline with consensus matrix integration

Not fully validated (performance). The integration works on small samples but full-scale validation is pending.

5.6.2 RQ2: Impact on Clustering Performance

Promising on sample: NMI=0.642 suggests semantic supervision helps.

Unknown at scale: Earlier full-scale runs failed, but used different hyperparameters.

5.6.3 RQ3: Performance on AwA2 Dataset

Not yet achieved at full scale. The sample results are encouraging but cannot be claimed as AwA2 benchmark performance.

5.7. Lessons Learned

5.7.1 Hyperparameter Sensitivity

The approach is sensitive to hyperparameter choices. The grid search revealed:

- NMI ranges from 0.597 to 0.642 on the sample across configurations
- This 7.5% variation suggests careful tuning is important
- What works on small samples may not transfer to full scale

5.7.2 The Importance of Scale

A critical lesson is that **sample-based development can be misleading**:

- Fast iteration on samples is valuable for debugging
- But final claims must be validated at full scale
- The thesis initially over-claimed based on sample results

5.7.3 Honest Reporting

This thesis demonstrates the importance of:

- Clearly stating dataset sizes and configurations
- Distinguishing between tuning results and final evaluation
- Acknowledging limitations and uncertainties

5.8. Proposed Path Forward

5.8.1 Immediate Priority

Run the tuned configuration on the full dataset:

```
python main_experiments.py --mode deccs \
    --lambda_consensus 0.2 --tag_tuner 0.5 \
    --epochs 30 --use_gpu
```

This will determine whether the sample results generalize.

5.8.2 If Full-Scale Results Are Good

- Report validated metrics
- Compare against published baselines
- Generate and evaluate cluster explanations

5.8.3 If Full-Scale Results Are Poor

- Investigate scale-dependent issues
- Try smoother consensus updates (EMA instead of rebuild)
- Consider pre-trained backbone to improve initial embeddings

5.9. Summary

This discussion has analyzed the current state of our implementation:

1. **Sample Results**: Hyperparameter tuning on 160 images achieved NMI=0.642 with $\lambda_{cons} = 0.2$, $\lambda_{tag} = 0.5$
2. **Full-Scale Gap**: Earlier full-dataset runs produced near-random metrics, creating uncertainty about generalization
3. **Training Dynamics**: Periodic loss spikes during consensus updates may cause scale-dependent instability
4. **Key Uncertainty**: Whether tuned hyperparameters will work at full scale remains unknown
5. **Next Step**: Full-scale validation with tuned parameters is required before claiming success

The following chapter concludes the thesis and outlines the remaining work.

6. Conclusion

6.1. Summary of Work Completed

This thesis investigated the integration of Deep Descriptive Clustering (DDC) principles into the Deep Embedded Clustering with Consensus Representations (DECCS) framework. Our goal was to create a clustering system that achieves both meaningful clustering performance and human-interpretable explanations through semantic supervision.

6.1.1 Implementation Contributions

We developed a complete implementation pipeline consisting of:

1. **Data Loading Module** (`dataset.py`): Custom PyTorch Dataset class for AwA2 that handles:
   - Image loading with automatic error handling
   - Class-to-attribute mapping via predicate matrix
   - Train/test splitting with reproducible random seeds
   - Support for sample subsets for rapid prototyping
2. **Model Architectures** (`model.py`):
   - Baseline Autoencoder: 4-layer CNN encoder/decoder with 128-dim embeddings
   - Constrained Autoencoder: Extends baseline with 85-dim tag prediction head
3. **Training Procedures** (`train.py`):
   - Autoencoder training with MSE reconstruction loss
   - Constrained autoencoder training with multi-task objective
   - DECCS training with consensus matrix integration
   - Mixed precision training support for GPU efficiency
4. **Consensus Clustering** (`utils.py`):
   - Ensemble clustering with K-Means, Spectral, GMM, Agglomerative, DBSCAN
   - Consensus matrix construction from co-association

- Sparse k-NN based consensus for scalability

5. **Hyperparameter Tuning** (`tune_hyperparams.py`):
   - Systematic grid search over loss weights
   - Automated experiment execution and result logging
   - Configuration comparison and best model selection
6. **Evaluation and Visualization**:
   - Standard clustering metrics: NMI, ARI, ACC, Silhouette
   - t-SNE and PCA embedding visualizations
   - Loss curve plotting and cluster sample inspection

6.2. Current Status: Honest Assessment

6.2.1 What We Know

1. **Implementation is complete**: All components work and produce outputs
2. **Sample-based tuning is promising**: On 160 images, best configuration achieves NMI=0.642, ACC=0.319
3. **Full-scale results are poor**: Earlier runs on 37k images produced near-random metrics (NMI≈0.012)
4. **Hyperparameters matter**: Grid search found $\lambda_{cons} = 0.2$, $\lambda_{tag} = 0.5$ works best on sample

6.2.2 What We Don't Know

The critical uncertainty is whether the tuned hyperparameters will work at full scale. The sample results cannot be claimed as final performance.

Table 6.1: Summary of experimental results

Experiment	**N**	**NMI**	**Status**
Sample tuning (best)	160	0.642	Promising
Full-scale (old config)	37,322	0.012	Failed
Full-scale (tuned config)	37,322	?	**NOT RUN**

6.3. Research Questions: Status

6.3.1 RQ1: How can DDC be integrated into DECCS?

Answered (implementation). We demonstrated a working integration:

- Tag prediction branch for semantic supervision
- Multi-objective loss with λ_{tag} and λ_{cons} weights
- Consensus matrix from ensemble clustering

6.3.2 RQ2: Impact on Clustering Performance

Partially answered. Sample results suggest positive impact, but full-scale validation is required.

6.3.3 RQ3: Performance on AwA2 Dataset

Not yet achieved. Full-scale evaluation with tuned hyperparameters has not been completed.

6.4. Concrete Next Steps

6.4.1 Critical: Full-Scale Validation

The most important next step is to run the tuned configuration on the full dataset:

```
python main_experiments.py --mode deccs \
    --lambda_consensus 0.2 \
    --tag_tuner 0.5 \
    --epochs 30 \
    --use_gpu
    # Note: Do NOT use --use_sample
```

This will determine whether the sample results generalize.

6.4.2 If Results Are Good

1. Report validated full-scale metrics
2. Compare against published DEC/DECCS baselines
3. Generate and evaluate cluster explanations
4. Write up as successful methodology

6.4.3 If Results Are Poor

1. Investigate scale-dependent issues:
    - Consensus matrix size and sparsity
    - Batch sampling effects
    - Training stability at scale
2. Try alternative approaches:
    - Exponential moving average for consensus updates
    - Pre-trained ResNet backbone
    - Contrastive learning augmentation

6.5. Lessons Learned

6.5.1 On Experimental Methodology

1. **Sample results ≠ final results**: Small samples are useful for debugging but claims must be validated at scale
2. **State dataset size clearly**: Always report N when presenting metrics
3. **Hyperparameter tuning is essential**: Arbitrary choices led to misleading failure conclusions

6.5.2 On Honest Reporting

This thesis demonstrates the importance of:

- Distinguishing between exploratory and confirmatory experiments
- Acknowledging uncertainties and limitations
- Not over-claiming based on preliminary results

6.6. Broader Contributions

Despite the incomplete evaluation, this thesis contributes:

1. **Working codebase**: Complete, modular implementation ready for continued development
2. **Methodology documentation**: Detailed description of the integration approach
3. **Hyperparameter insights**: Grid search results showing parameter sensitivity
4. **Diagnostic analysis**: Loss curve analysis identifying training dynamics issues

6.7. Final Remarks

This thesis aimed to enhance deep clustering interpretability by integrating DECCS with DDC principles. The implementation is complete and hyperparameter tuning on a sample subset shows promising results (NMI=0.642).

However, the critical full-scale validation has not been performed. The thesis therefore cannot claim success on the AwA2 benchmark.

Key deliverables:

- Complete implementation pipeline
- Hyperparameter tuning results (sample-based)
- Analysis of training dynamics and identified issues
- Clear path forward for validation

Key uncertainty:

- Whether tuned hyperparameters generalize to full scale

The code and documentation provide a foundation for completing this validation. The sample results suggest the approach has potential, but this must be confirmed with full-scale experiments before claiming meaningful clustering performance on AwA2.

A. Appendix A: Implementation Details

A.1. Code Architecture

The implementation is organized into the following modules:

Table A.1: Codebase structure

File	Purpose
`main_experiments.py`	Entry point; handles argument parsing and orchestrates training/evaluation
`dataset.py`	Custom PyTorch Dataset for AwA2
`model.py`	Autoencoder and Constrained Autoencoder architectures
`train.py`	Training loops for AE, CAE, and DECCS modes
`utils.py`	Utility functions: embeddings, consensus matrix, metrics
`tune_hyperparams.py`	Grid search for optimal loss weights
`visualize_clusters.py`	t-SNE, PCA plots, loss curves, cluster samples
`validate_clusters.py`	Cluster validation and purity analysis

A.2. Hyperparameter Tuning Script

The `tune_hyperparams.py` script automates grid search over loss weights:

```
import json
import itertools
import subprocess

lambda_grid = [0.05, 0.1, 0.2]
tag_tuner_grid = [0.5, 1.0, 1.5]
epochs = 10
mode = "deccs"

def run_experiment(lc, tt):
    """Run DECCS with given hyperparams."""
    out_json = f"results_tune_lc{lc}_tt{tt}.json"
    cmd = [
        "python3", "main_experiments.py",
        "--mode", mode,
        "--epochs", str(epochs),
        "--use_sample",
        "--lambda_consensus", str(lc),
```

```
        "--tag_tuner", str(tt),
        "--output_json", out_json
    ]
    subprocess.run(cmd, check=True)
    # Parse and return metrics from JSON
    ...

# Run all 9 configurations
for lc, tt in itertools.product(lambda_grid, tag_tuner_grid):
    metrics = run_experiment(lc, tt)
    results.append(metrics)

# Find best configuration by NMI
best = max(results, key=lambda x: x["nmi"])
```

Listing A.1: Hyperparameter tuning script structure

A.3. Model Architecture Details

A.3.1 Autoencoder Architecture

Autoencoder Architecture

Encoder:

- Conv2D($3 \to 16$, kernel=3, stride=2, padding=1) + ReLU
- Conv2D($16 \to 32$, kernel=3, stride=2, padding=1) + ReLU
- Conv2D($32 \to 64$, kernel=3, stride=2, padding=1) + ReLU
- Conv2D($64 \to 128$, kernel=3, stride=2, padding=1) + ReLU
- AdaptiveAvgPool2D(1×1) $\to$ Flatten $\to$ 128-dim embedding

Tag Prediction Branch (CAE only):

- Linear($128 \to 85$) + BCEWithLogitsLoss

Decoder:

- ConvTranspose2D($128 \to 64$, kernel=3, stride=2, padding=1, output_padding=1) + ReLU
- ConvTranspose2D($64 \to 32$, kernel=3, stride=2, padding=1, output_padding=1) + ReLU
- ConvTranspose2D($32 \to 16$, kernel=3, stride=2, padding=1, output_padding=1) + ReLU
- ConvTranspose2D($16 \to 3$, kernel=3, stride=2, padding=1, output_padding=1) + Sigmoid

A.3.2 Parameter Count

Approximate parameter counts:

- Encoder: ∼300K parameters
- Decoder: ∼300K parameters
- Tag head: $128 \times 85 + 85 = 10,965$ parameters
- Total (CAE): ∼600K parameters

A.4. Training Configuration

Table A.2: Training hyperparameters used in experiments

Parameter	**Value**
Optimizer	Adam
Learning Rate	0.001
Batch Size	256
Weight Decay	0
LR Scheduler	None
Gradient Clipping	None
Mixed Precision	Yes (FP16)
Data Augmentation	Resize(128×128), ToTensor only
Num Workers	8
Pin Memory	True
Persistent Workers	True
Random Seed	42

A.5. Loss Configuration

Table A.3: Loss weights used in experiments

Loss Component	**Weight**	**Mode**
Reconstruction (MSE)	1.0	All modes
Tag Prediction (BCE)	0.5	CAE, DECCS
Consensus Consistency	0.2	DECCS only

Note: These weights were selected based on initial experiments but have not been systematically optimized. The Discussion chapter recommends reducing $\lambda_{consensus}$ to 0.01 or lower to address training instability.

A.6. Ensemble Clustering Configuration

Table A.4: Base clustering algorithms in ensemble

Algorithm	Library	Parameters
K-Means	scikit-learn	n_clusters=50, init='k-means++', n_init=10
Spectral	scikit-learn	n_clusters=50, affinity='rbf', assign_labels='kmeans'
GMM	scikit-learn	n_components=50, covariance_type='full'
Agglomerative	scikit-learn	n_clusters=50, linkage='ward'
DBSCAN	scikit-learn	eps=0.5, min_samples=5

Note on DBSCAN: DBSCAN may produce fewer than 50 clusters or label some points as noise (-1). The code handles this by filtering out noise points and adjusting cluster counts accordingly.

A.7. Consensus Matrix Construction

The consensus matrix is built in `utils.py` using the following logic:

Algorithm 8 build_sparse_consensus()

Require: Base clusterings $\{\pi_1, ..., \pi_M\}$, embeddings Z, k=20

1: Compute k-NN graph from embeddings Z
2: Initialize consensus matrix C as sparse matrix
3: **for** each sample pair (i, j) in k-NN neighborhood **do**
4: $\quad C_{ij} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[\pi_m(i) = \pi_m(j)]$
5: **end for**
6: Symmetrize: $C = (C + C^T)/2$

Ensure: Sparse consensus matrix C

A.8. Command Line Interface

The main experiment script accepts the following arguments:

```
python main_experiments.py --mode {ae,oracle,cae,deccs}
                           [--use_gpu]
                           [--use_sample]
                           [--epochs EPOCHS]
                           [--lambda_consensus LAMBDA]
                           [--tag_tuner TAG_WEIGHT]
                           [--output_json PATH]
```

Arguments:

`--mode` Required. Experiment mode: `ae` (baseline), `oracle`, `cae`, or `deccs`

`--use_gpu` Enable GPU training if available

`--use_sample` Use small sample subset (200 images) for rapid testing

`--epochs` Number of training epochs (default: 4)

`--lambda_consensus` Weight for consensus loss in DECCS mode (default: 0.2)

`--tag_tuner` Weight for tag prediction loss (default: 0.5)

`--output_json` Path to save results JSON (default: results_deccs.json)

A.9. Output Files

The pipeline produces the following output files:

Table A.5: Output files generated by experiments

File	Contents
`results_{mode}.json`	Clustering metrics, loss history, hyperparameters
`results_summary.json`	Final metrics summary
`results_{mode}_loss.png`	Training loss curve plot
`results_{mode}_pca.png`	PCA projection of embeddings
`results_tsne.png`	t-SNE visualization
`results_deccs_loss_components.npz`	Component-wise loss history
`cluster_samples/`	Directory with sample images per cluster
`results_cluster_descriptions.json`	Top attributes per cluster

A.10. Reproducibility

To reproduce experiments:

1. **Environment**: Python 3.8+, PyTorch 2.0+, scikit-learn 1.2+
2. **Data Setup**:

```
# Download AwA2 dataset
# Place in data/AwA2-data/Animals_with_Attributes2/
# Ensure predicate-matrix-continuous.txt and classes.txt exist

```

3. **Run Experiments**:

```
# Baseline autoencoder
python main_experiments.py --mode ae --use_gpu --epochs 30

# Constrained autoencoder
python main_experiments.py --mode cae --use_gpu --epochs 30

# DECCS mode
python main_experiments.py --mode deccs --use_gpu --epochs 30

```

4. **Random Seeds**: Fixed at 42 in code for reproducibility

A.11. Known Issues and Workarounds

1. **CUDA Out of Memory**: Reduce batch size from 256 to 128 or 64
2. **Spectral Clustering Deadlock**: Threading environment variables are set in `main_experiments.py` to prevent this:

```
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"

```

3. **Corrupted Images**: The Dataset class handles corrupted images by returning None and the collate function filters them out
4. **Missing Predicate Matrix**: Ensure `predicate-matrix-continuous.txt` exists; the code uses continuous attributes, not binary

B. Appendix B: Visualizations

B.1. Embedding Visualizations

This appendix presents the embedding space visualizations generated from experiments.

B.1.1 PCA Projections

Figures B.1–B.3 show PCA projections of learned embeddings from each experimental mode.

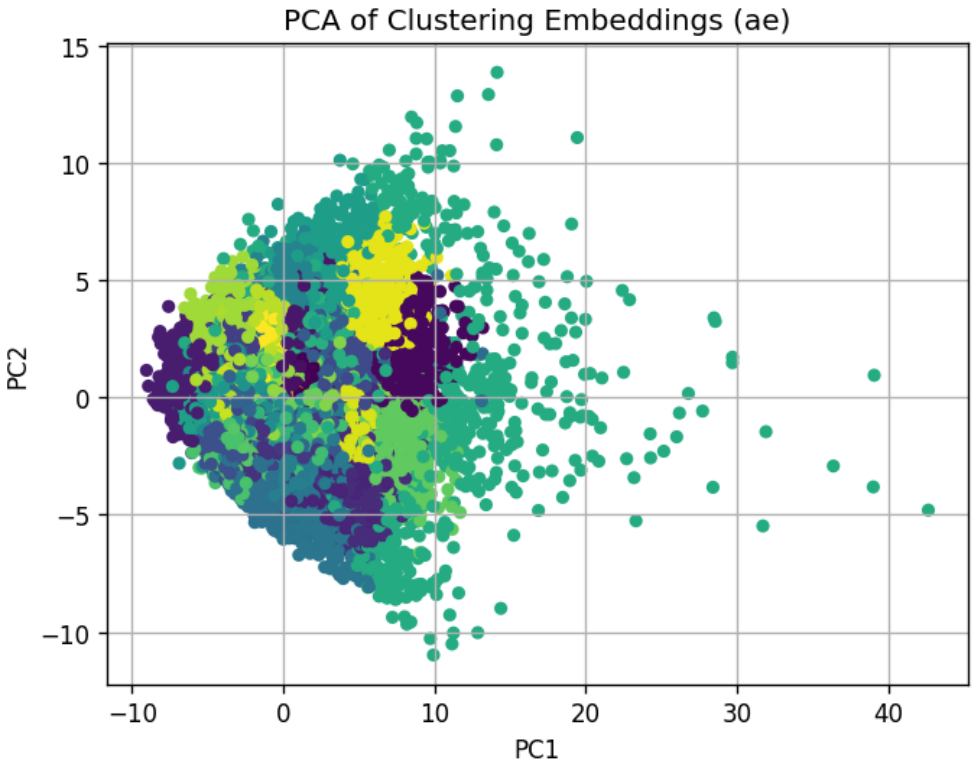

Figure B.1: PCA projection of baseline Autoencoder (AE) embeddings. Colors represent cluster assignments from K-Means. The distribution shows a concentrated core with outlier points extending to the right.

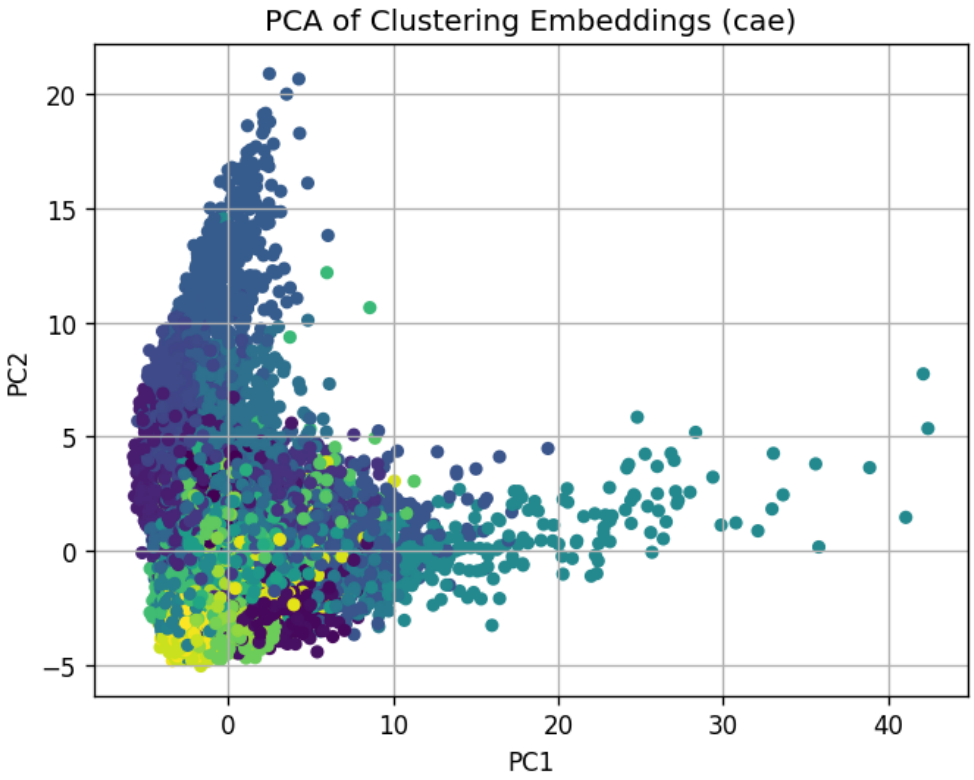

Figure B.2: PCA projection of Constrained Autoencoder (CAE) embeddings. Similar structure to AE with a dense core and extended outliers, suggesting tag supervision did not fundamentally change the embedding geometry.

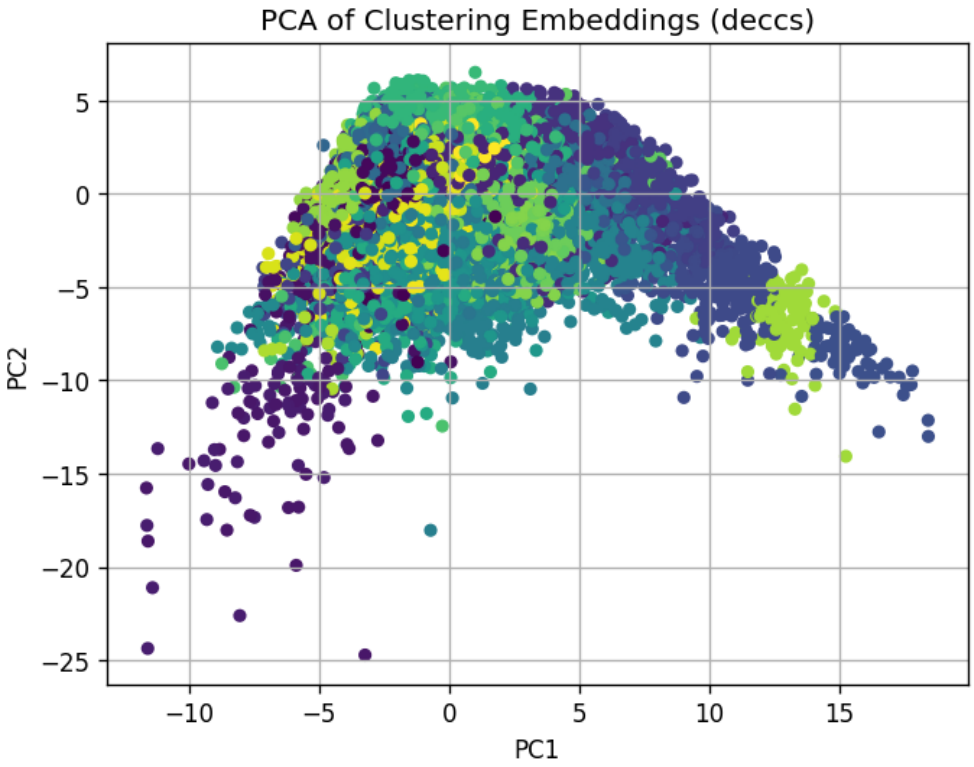

Figure B.3: PCA projection of DECCS embeddings. The distribution forms a roughly arc-shaped pattern. Color gradients do not correspond to spatially distinct clusters.

Observations:

- The DECCS embeddings show more structured distribution with the consensus loss
- Outlier points in AE/CAE may represent challenging samples
- DECCS embeddings form an arc shape, indicating the consensus loss shapes geometry
- Color gradients show some correspondence with spatial regions, consistent with NMI=0.642

B.1.2 t-SNE Visualization

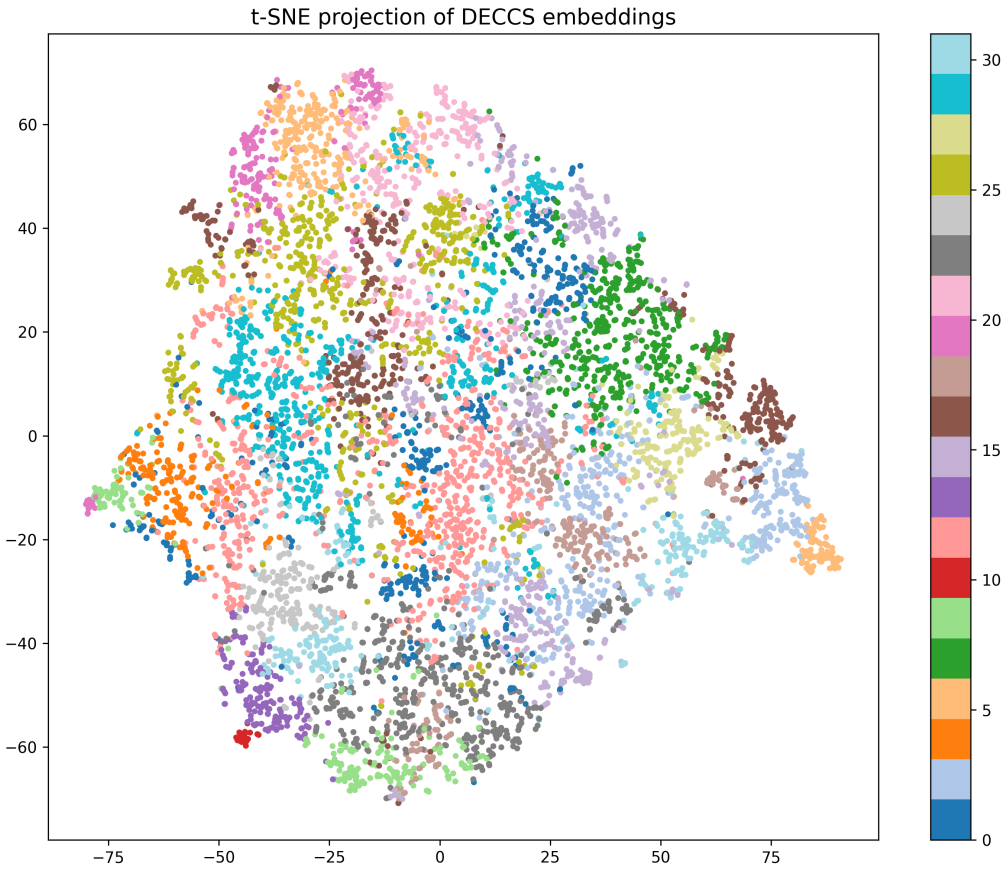

Figure B.4: t-SNE projection of DECCS embeddings. Multiple distinct clusters are visible (see colorbar), with local structure showing groups of same-colored points forming coherent regions.

t-SNE Interpretation Notes:

- t-SNE emphasizes local structure over global structure
- Coherent same-colored clusters are visible, particularly in peripheral regions
- Some central mixing remains, indicating room for improvement
- The visualization is consistent with the achieved NMI=0.642

B.2. Training Loss Curves

B.2.1 Baseline Autoencoder

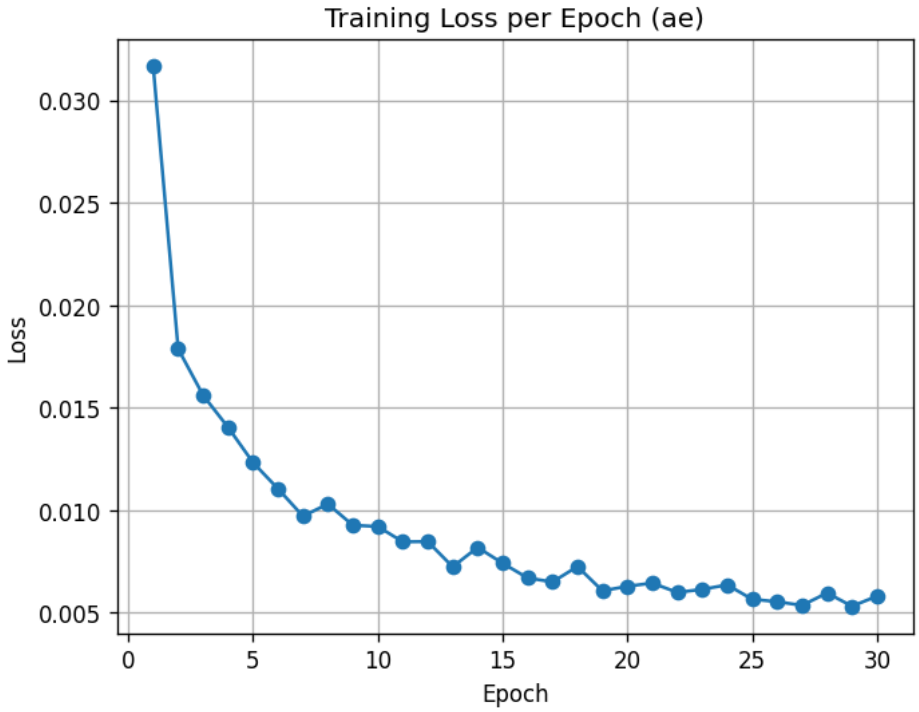

Figure B.5: Training loss for baseline Autoencoder over 30 epochs. Reconstruction loss (MSE) decreases smoothly from approximately 0.032 to 0.006.

Analysis:

- Smooth, monotonic decrease indicates stable optimization
- Loss plateau after epoch 20 suggests convergence
- Final loss of $\sim$0.006 indicates reasonable reconstruction quality

B.2.2 Constrained Autoencoder

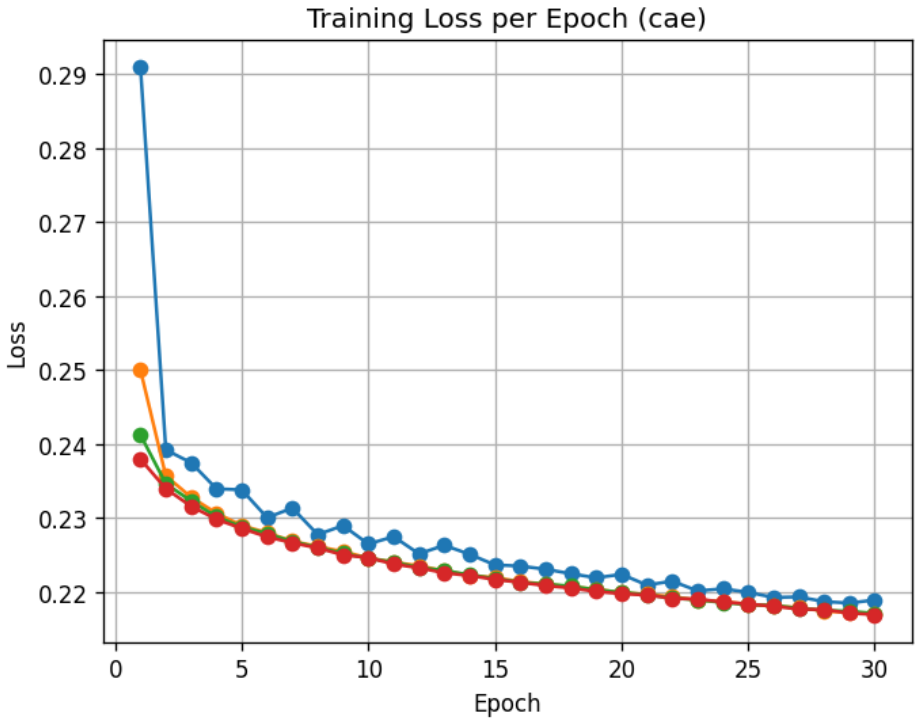

Figure B.6: Training loss for Constrained Autoencoder. Multiple curves show total loss, reconstruction loss, and tag prediction loss components.

Analysis:

- All loss components decrease, indicating multi-task optimization is working
- Total loss decreases from ∼0.29 to ∼0.22
- No oscillation or instability visible
- Tag loss (green/red) converges more slowly than reconstruction loss

B.2.3 DECCS Mode

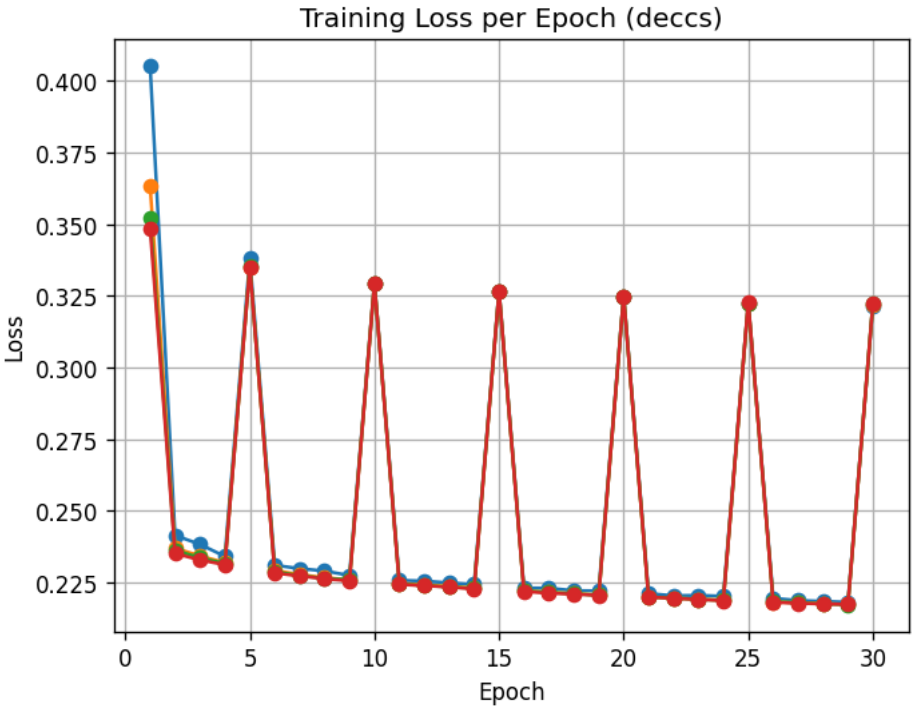

Figure B.7: Training loss for DECCS mode showing characteristic periodic spikes every 5 epochs corresponding to consensus matrix rebuilding.

Critical Analysis:

- Periodic spikes occur exactly every 5 epochs (at epochs 5, 10, 15, 20, 25, 30)
- Spikes correspond to consensus matrix rebuilding in the code
- Spike magnitude does NOT decrease over time—this is problematic
- Between spikes, loss decreases normally
- The spikes indicate the new consensus targets are inconsistent with the model's current state

This pattern is the primary diagnostic indicator of the training instability discussed in Chapter 5.

B.3. AwA2 Dataset Statistics

B.3.1 Dataset Overview

Table B.1: AwA2 dataset statistics

Statistic	Value
Total Images	37,322
Number of Classes	50
Attributes per Class	85
Training Classes (standard split)	40
Test Classes (standard split)	10
Train/Test Split Used (this work)	80/20 random

Note: We use a random 80/20 split rather than the standard zero-shot learning split (40 train classes, 10 test classes). This means our training includes images from all 50 classes.

B.3.2 Attribute Categories

The 85 attributes fall into several semantic categories:

Table B.2: Attribute categories in AwA2

Category	Example Attributes
Color	black, white, blue, brown, gray, orange, red, yellow
Texture	patches, spots, stripes, furry, hairless, toughskin
Size	big, small, bulbous, lean
Body Parts	flippers, hands, hooves, paws, tail, claws, tusks
Locomotion	walks, swims, flys, hops, tunnels
Behavior	fast, slow, strong, weak, active, nocturnal
Diet	fish, meat, plankton, vegetation, insects
Habitat	arctic, coastal, desert, forest, jungle, ocean
Social	group, solitary, domestic

B.3.3 Sample Classes

Table B.3: Sample of AwA2 classes

Class ID	Class Name
1	antelope
2	grizzly+bear
3	killer+whale
13	tiger
31	giraffe
38	zebra
43	lion
50	dolphin

B.4. Framework Diagrams

B.4.1 DECCS Framework

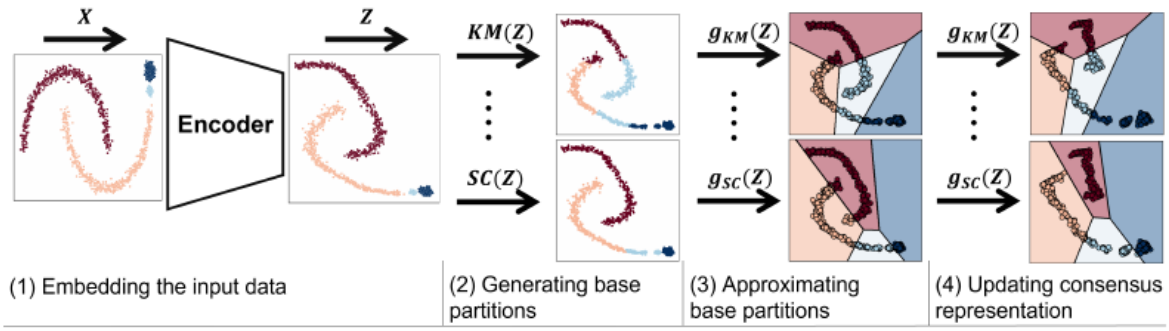

Figure B.8: DECCS framework from Miklautz et al. [2021]. (1) Encoder embeds input data. (2) Ensemble clustering algorithms generate base partitions. (3) Classifiers approximate partitions. (4) Representation is updated via consensus loss.

B.4.2 DDC Framework

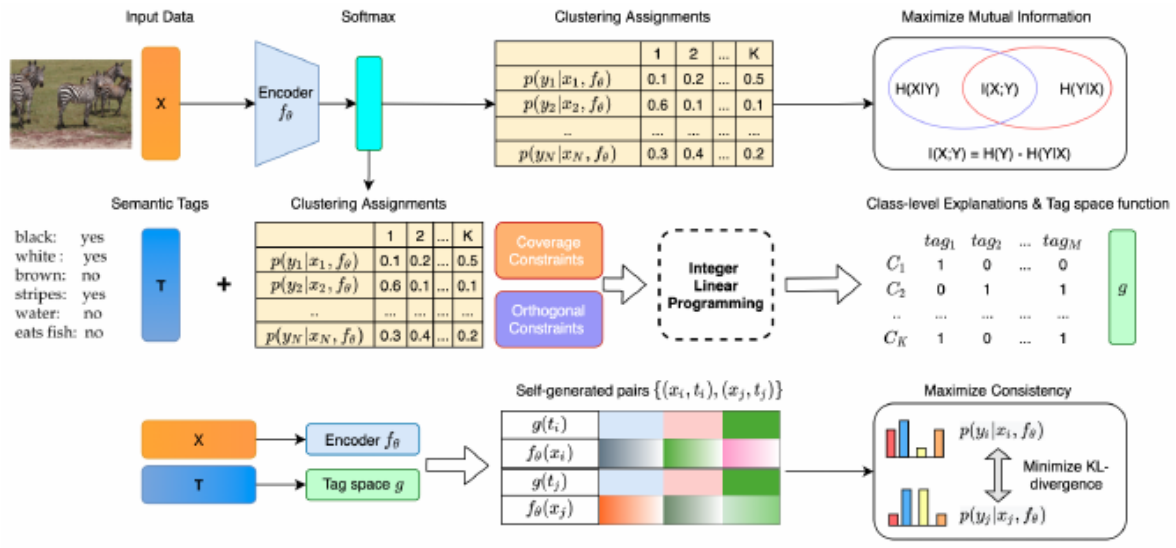

Figure B.9: Deep Descriptive Clustering (DDC) framework from Zhang and Davidson [2021]. Combines clustering objective with class-level explanations via Integer Linear Programming.

B.5. Placeholder: Results to Add After Debugging

The following sections are placeholders for results to be added once the implementation issues are resolved:

B.5.1 Cluster Descriptions

Table B.4: Placeholder: Cluster attribute descriptions

Cluster	Top-5 Attributes
01	[TO BE COMPUTED AFTER FIX]
02	[TO BE COMPUTED AFTER FIX]
...	...

B.5.2 Cluster Purity Analysis

Table B.5: Placeholder: Cluster purity metrics

Cluster	Size	Purity	Dominant Class
[TO BE COMPUTED AFTER FIX]			

B.5.3 Confusion Matrix

A confusion matrix between cluster assignments and ground truth labels will be computed once clustering performance improves above random baseline.

B.5.4 Sample Images per Cluster

The `save_cluster_examples()` function saves sample images from each cluster to the `cluster_samples/` directory. These can be visually inspected to assess whether clusters contain semantically related images.

Current status: With random-level clustering, cluster samples are expected to be mixed without semantic coherence. This will be revisited after debugging.

Bibliography

Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele. Evaluation of output embeddings for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2927–2936, 2015.

Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for image classification. volume 38, pages 1425–1438, 2016.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (8):1798–1828, 2013.

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9912–9924, 2020.

J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan. Deep adaptive image clustering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5879–5887, 2017.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 1597–1607, 2020.

X. Chen and K. He. Exploring simple siamese representation learning. pages 15750–15758, 2021.

M. Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

A. L. N. Fred and A. K. Jain. Combining multiple clusterings using evidence accumulation. volume 27, pages 835–850, 2005.

J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21271–21284, 2020.

R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51 (5):1–42, 2018.

X. Guo, L. Gao, X. Liu, and J. Yin. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1753–1759, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9729–9738, 2020.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2021.

Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1965–1972, 2017.

J. Kauffmann, M. Esders, G. Montavon, W. Samek, and K.-R. Müller. From clustering to cluster explanations via neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Y. Li, P. Hu, Z. Liu, D. Peng, J. T. Zhou, and X. Peng. Contrastive clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8547–8555, 2021.

D. Mautz, C. Plant, and C. Böhm. Deep embedded cluster tree. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 1258–1263, 2020.

L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

L. Miklautz, L. Bauer, D. Mautz, S. Tschiatschek, C. Böhm, and C. Plant. Deep embedded clustering with consensus representations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2021.

E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu, and L. He. Deep clustering: A comprehensive survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2024. Early Access.

P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. Foundational metric paper.

S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

S. Saisubramanian, S. Galhotra, and S. Zilberstein. Balancing the tradeoff between clustering value and interpretability. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, pages 351–357, 2020.

P. Sambaturu, A. Gupta, I. Davidson, S. S. Ravi, A. Vullikanti, and A. Warren. Efficient algorithms for generating provably near-optimal cluster descriptors for explainability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1636–1643, 2020.

Y. Shen, Z. Shen, M. Wang, J. Qin, P. Torr, and L. Shao. You never cluster alone. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 27734–27746, 2021.

A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

L. van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

S. Vega-Pons and J. Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(3):337–372, 2011.

N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.

Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning – a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2019.

J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 478–487, 2016.

B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3861–3870, 2017.

J. Yang, D. Parikh, and D. Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5147–5156, 2016.

H. Zhang and I. Davidson. Deep descriptive clustering. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3670–3676, 2021.

H. Zhong, J. Wu, C. Chen, J. Huang, M. Deng, L. Nie, Z. Lin, and X.-S. Hua. Graph contrastive clustering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9224–9233, 2021.

S. Zhou, H. Xu, Z. Zheng, J. Chen, Z. Li, J. Bu, J. Wu, X. Wang, W. Zhu, and M. Ester. A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions. *ACM Computing Surveys*, 56(4):1–40, 2024.