DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

„Prototype-based Representation Learning with Deep Clustering“

verfasst von / submitted by

Dipl.-Ing. Lukas Miklautz BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 786 880
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ. Prof. Dr. Claudia Plant

Acknowledgements

I am grateful to all who contributed to the completion of this thesis.

First, I want to thank my supervisor Claudia Plant who gave me the chance to pursue a PhD and supported me during the whole endeavor. I am thankful for the freedom I had in conducting research and the many opportunities I was given. I would also like to thank Christian Böhm for the many helpful discussion sessions, feedback on papers and for building the bridge to the colleagues at LMU Munich.

Next, I want to thank Stratis Ioannidis and Matthias Renz who agreed to review my thesis and to serve as members on my committee.

I am grateful to all my co-authors, who have taught me a great deal. Especially, Dominik Mautz who supported me in my early days of my PhD and with whom I wrote my first paper. My thanks go also to Lena Bauer and Benjamin Schelling who listened patiently to my ramblings about clustering and representation learning and helped me to put all the pieces together. Further, I want to thank Can Altinigneli, Walid Durani, Collin Leiber, Andreas Stephan, Martin Teuffenbach and Pascal Weber for working with me on several exciting projects. I also want to thank Sebastian Tschiatschek for his advice and feedback during my PhD and our work together.

I am very thankful to Sepp Hochreiter for hosting me at the Institute for Machine Learning at JKU Linz and giving me the opportunity to meet great people there. My gratitude also goes to Johannes Lehner, who enjoys discussing research ideas as much as I do, and to Benedikt Alkin, who is by far the fastest at implementing new ideas that I know.

Being part of the Data Mining and Machine Learning group over the last few years and seeing it grow from a newly formed group with a few members to the large group we are today has made me really proud. I am happy for all the amazing colleagues I have met. Ylli who shares the office with me since I can remember and who is always up for talking about research and life. My past office mates Dr. Martin, Dr. Ben and Dr. Sahar who finished their PhD already. I also want to thank all the other members of the group for all the discussions, activities, lunches, hikes and fun together. I really enjoyed being a part of the team.

I want to thank our technician Ewald Hotop for keeping the infrastructure running, without him we would be helpless. I would also like to thank our administrative staff Tanja, Charlotte and Nurcan who were always supportive and helped me navigate the intricate university bureaucracy.

Finally, I am grateful to my friends, family and my partner Saskia for supporting me.

Bibliographic Note

Three papers of this thesis are already published in conference proceedings and one is currently under review. Therefore, the chapters of this thesis are based on the following publications:

- **Paper A:** Lukas Miklautz, Dominik Mautz, Muzaffer Can Altinigneli, Christian Böhm and Claudia Plant. “Deep Embedded Non-Redundant Clustering”. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI), 2020.
- **Paper B:** Lukas Miklautz, Lena G. M. Bauer, Dominik Mautz, Sebastian Tschiatschek, Christian Böhm and Claudia Plant. “Details (Don’t) Matter: Isolating Cluster Information in Deep Embedded Spaces”. In: The Thirtieth International Joint Conference on Artificial Intelligence (IJCAI), 2021.
- **Paper C:** Lukas Miklautz, Martin Teuffenbach, Pascal Weber, Rona Perjuci, Walid Durani, Christian Böhm and Claudia Plant. “Deep Clustering With Consensus Representations”. In: The Twenty-Second IEEE International Conference on Data Mining (ICDM), 2022.
- **Paper D:** Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch and Claudia Plant. “Non-Redundant Image Clustering of Early Medieval Glass Beads”. Submitted for review at: The ADS Track of the Twenty-Ninth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023.

Papers written during the time of the PhD which are not part of this thesis:

- **Paper E:** Benjamin Schelling, Lukas Miklautz and Claudia Plant. “Non-Linear Cluster Enhancement: Forcing Clusters into a Compact Shape”. In: The Twenty-Fourth European Conference on Artificial Intelligence (ECAI), 2020.
- **Paper F:** Muzaffer Can Altinigneli, Lukas Miklautz, Christian Böhm and Claudia Plant. “Hierarchical Quick Shift Guided Recurrent Clustering”. In: The Thirty-Sixth IEEE International Conference on Data Engineering (ICDE), 2020.
- **Paper G:** Pascal Weber, Lukas Miklautz, Akshey Kumar, Moritz Grosse-Wentrup and Claudia Plant. “CaFe DBSCAN: A Density-based Clustering Algorithm for Causal Feature Learning”. Submitted for review at: The Twenty-Ninth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023.

- **Paper H:** Johannes Lehner, Benedikt Alkin, Andreas Fürst, Elisabeth Rumetshofer, Lukas Miklautz, Sepp Hochreiter. "Contrastive Tuning: A Little Help to Make Masked Autoencoders Forget". Submitted for review at: The Twentieth IEEE/CVF International Conference on Computer Vision (ICCV), 2023.

Co-authored the open-source Python package ClustPy for recently introduced clustering algorithms that includes several of the methods proposed in this thesis. Available at `https://github.com/collinleiber/ClustPy`:

- **Project A:** Collin Leiber, Lukas Miklautz, Claudia Plant, Christian Böhm. "Benchmarking Deep Clustering Algorithms with ClustPy". Submitted for review as Lecture-Style Tutorial at: The Twenty-Ninth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023.

Abstract

Clustering is a fundamental task in data science. Clustering is concerned with finding groups of objects in data that share similar characteristics, while keeping dissimilar objects separated. In recent years, the increase in size, dimensionality and heterogeneity of data has led to a corresponding increase in clustering methods to address these novel challenges. For example, clustering high-dimensional objects such as images directly in the pixel space does not work satisfactorily. This has led to a growing interest in combining deep learning with clustering, known as deep clustering. Algorithms of this type combine the non-linear representation of a self-supervised or unsupervised deep learning algorithm, e.g., an autoencoder, with a clustering objective and optimize both simultaneously. In this thesis, we focus on learning representations for clustering and propose several methods that provide novel capabilities to deep clustering algorithms.

We introduce ENRC (Embedded Non-Redundant Clustering), the first deep clustering method that can find multiple, non-redundant clusterings of different dimensionalities in high-dimensional data, like images. For instance, the image of a red cube can be clustered once according to its shape and once according to its color. ENRC does this by learning to split the representation into useful features for each clustering. In cooperation with archaeologists, we use ENRC to find non-redundant clusterings of images of Early Medieval glass beads. This led to the first application of non-redundant clustering to images of archaeological artifacts.

Building on the idea of splitting the learned representation into further parts, we introduce ACe/DeC (Autoencoder Centroid-based Deep Clustering). ACe/DeC splits the autoencoder representation into features relevant to the clustering task and features relevant to the autoencoder reconstruction task. The separated representation leads to a more robust clustering performance with respect to the choice of learning rate, removes difficult to tune hyperparameters and makes the clustering result easier to interpret.

ENRC, ACe/DeC and many other existing deep clustering methods assume k-means-like clusters during representation learning. This has several limitations, such as the restriction to spherical clusters. With DECCS (Deep Embedded Clustering with Consensus representationS), we present a deep clustering algorithm that can combine several existing clustering methods in an ensemble to learn a single consensus representation on which all ensemble members achieve a consensus clustering. This enables DECCS to find non-spherical clusters as well.

The introduced methods are built around the notion of prototype-based representation learning for clustering, where prototypes are used to guide the deep clustering process. Further, the methods are characterized by incorporating concepts from subspace, non-redundant, and consensus clustering methodologies, leading to novel approaches in deep clustering research.

Kurzfassung

Clustering ist ein grundlegendes Gebiet der Datenwissenschaften. Beim Clustering handelt es sich um ein Verfahren das Gruppen von Objekten in Daten findet, die ähnliche Merkmale aufweisen, während unähnliche Objekte getrennt bleiben. In den letzten Jahren hat die zunehmende Größe, Dimensionalität und Heterogenität von Daten zu einer entsprechenden Zunahme von Clustering-Methoden geführt, um diesen neuen Herausforderungen zu begegnen. Beispielsweise funktioniert das Clustering hochdimensionaler Objekte wie Bilder im Pixelraum nicht zufriedenstellend. Dies hat zu einem wachsenden Interesse an der Kombination von Deep Learning mit Clustering, dem so genannten Deep Clustering, geführt. Algorithmen dieser Art kombinieren die nichtlineare Repräsentation eines selbstüberwachten oder unüberwachten Deep Learning Algorithmus, z. B. eines Autoencoders, mit einem Clustering-Ziel und optimieren beide gleichzeitig. In dieser Arbeit konzentrieren wir uns auf das Lernen von Repräsentationen (representation learning) für das Clustering und schlagen mehrere Methoden vor, die den Deep Clustering Algorithmen neue Möglichkeiten eröffnen.

Wir stellen ENRC (Embedded Non-Redundant Clustering) vor, die erste Deep Clustering Methode, die mehrere, nicht redundante Clusterings unterschiedlicher Dimensionalität in hochdimensionalen Daten wie Bildern finden kann. Zum Beispiel kann das Bild eines roten Würfels einmal nach seiner Form und einmal nach seiner Farbe gruppiert werden. ENRC erreicht das, indem es lernt, die Darstellung in nützliche Merkmale für jedes Clustering aufzuteilen. In Zusammenarbeit mit Archäologen haben wir ENRC eingesetzt, um nicht redundante Clusterings von Bildern frühmittelalterlicher Glasperlen zu finden. Dies führte zur ersten Anwendung des nicht-redundanten Clusterings auf Bildern archäologischer Artefakte.

Aufbauend auf der Idee, die gelernte Repräsentation in weitere Teile zu zerlegen, führen wir ACe/DeC (Autoencoder Centroid-based Deep Clustering) ein. ACe/DeC teilt die Autoencoder-Repräsentation in Merkmale, die für das Clustering relevant sind, und in Merkmale, die für die Autoencoder Rekonstruktion relevant sind. Die getrennte Repräsentation führt zu einem robusteren Clusteringresultat in Bezug auf die Wahl der Lernrate, beseitigt schwierig zu bestimmende Hyperparameter und macht das Clustering-Ergebnis einfacher zu interpretieren.

ENRC, ACe/DeC und viele andere bestehende Deep Clustering Methoden gehen beim Lernen der Repräsentation von k-Means-artigen Clustern aus. Dies führt zu mehreren Einschränkungen, wie zum Beispiel die Beschränkung auf sphärische Cluster. Mit DECCS (Deep Embedded Clustering with Consensus representationS) stellen wir einen Deep Clustering Algorithmus vor, der mehrere bestehende Clustering-Methoden in einem Ensemble kombinieren kann, um eine einzige Konsensusrepräsentation zu erlernen, auf der alle Ensemblemitglieder ein Konsensusclustering erreichen. Dadurch findet DECCS auch nicht sphärische Cluster.

Die vorgestellten Methoden basieren auf dem Konzept des prototypenbasierten Lernens von Repräsentationen für das Clustering, bei dem Prototypen zur Anleitung des Deep Clustering Prozesses verwendet werden. Darüber hinaus zeichnen sich die Methoden durch die Einbeziehung von Konzepten aus Unterraum-, nicht redundanten und Konsensus-Clustermethoden aus, was zu neuen Ansätzen in der Deep Clustering Forschung führt.

Contents

1. Introduction

"Features matter." Ross Girshick et al. [GDDM14]

Clustering is a fundamental task of human cognition, comprising concepts of similarity and dissimilarity. Already infants are able to group objects with respect to their common characteristics, like shape and color [GM11]. Clustering in data mining and machine learning is about formalizing these concepts such that a machine can automatically detect which objects belong together and which should be separate.

This has many applications in data science [Jai10]. First, the summarization of a large collection of data instances by their most informative prototypes. Second, to study the underlying structure of a data set to detect common and uncommon instances during exploratory data analysis, and third, as unsupervised classification, where we want to find a natural grouping that is not predetermined by (human) labels. These applications are not mutually exclusive and usually clustering is used to pursue multiple of these goals at once. In science more generally, cluster analysis is applied in many different fields, e.g., to study groups of genes in genome data [But02, TWSW19], group images of archaeological artifacts according to their common shape and color [MSL$^+$23] or cluster text data from news articles by grouping them according to their topic [IT95, Gro22].

All these examples have in common that we need to find clusters in complex, high-dimensional data like genetic microarrays, high resolution images or large corpora of text. Finding meaningful clusters in such data requires a representation that emphasizes object discriminability. Traditionally, this has been done using feature engineering/transformation [FPS96], but deep learning [Sch15, LBH15] has given rise to automatic feature engineering, i.e., representation learning [BCV13]. Feature engineering refers to the practice of processing and combining existing characteristics in the data to obtain more revealing features for a desired downstream task, like clustering or classification. For instance, histograms of oriented gradients [DT05] are commonly used features extracted from images.

Representation learning automates the practice of feature engineering. This has been successful in domains where we have large amounts of data and it is difficult to find good features manually. These domains include data sets of image, video, audio, text, but also biological data. It remains to be seen whether deep learning is necessary when the data is tabular [GOV22], as for many tabular data sets non-deep models can outperform more complex deep models [RR19, GRKB21, SA22, SRP22, GOV22]. Further, feature engineering and simple models are powerful when we have access to domain knowledge. Representation learning can help when this is not the case, e.g., when the data is so complicated that not even domain experts agree on which features to use. Representation learning can also help when domain knowledge is available, but the task is very complex as the recent interest in physics-informed neural networks shows [KKL$^+$21].

Many of the successes of representation learning in the form of deep learning have been in the supervised setting with large quantities of labeled data. Deep learning without labels has only recently taken off in the form of deep unsupervised learning using (masked) autoencoders [HS06, HCX$^+$22], variational autoencoders [KW14], generative adversarial networks [GPM$^+$14, GPM$^+$20] or self-supervised learning with contrastive methods [GH10], like SimCLR [CKNH20].

This plethora of unsupervised representation learning methods has led to an increased interest of the clustering community forming the sub-field of deep clustering [MGL$^+$18]. Deep clustering methods perform representation learning for clustering. They learn a representation that improves cluster performance by increasing the discriminability between clusters. For instance, the algorithm DEC (Deep Embedded Clustering) [XGF16] learns a representation for the k-means [Mac65, Llo82, Jai10] algorithm by increasing cluster separation and cluster compactness. DEC was one of the first deep clustering methods that sparked a large body of follow up work [MGL$^+$18, AGSC18, RPY$^+$22, ZXZ$^+$22] and has been cited more than 2,300 times[1]. The exchange between the representation learning and clustering community was not one-sided. There are now several methods that use implicit or explicit clustering objectives in self-supervised representation learning [CBJD18, CMM$^+$20, ARV20].

For large, high-dimensional data sets, deep clustering algorithms have several advantages. First, the deep learning literature offers a wide variety of neural network architectures. This makes deep clustering algorithms suitable for many data modalities. Second, the learned representations are often low-dimensional, which helps to alleviate the challenges of the "curse of dimensionality" in clustering. This phenomenon refers to the difficulty of accurately distinguishing objects in high-dimensional data spaces [BGRS99, HAK00, AHK01], which can lead to sub-optimal clustering results. Third, deep clustering algorithms optimize a common objective that includes both a clustering and a data dependent objective, e.g., the autoencoder [HS06] reconstruction loss. This allows the learned representations to be suitable for clustering while preserving important properties of the data. While deep clustering methods have several benefits, they face a circular dependency problem. To find a good clustering, the algorithm needs a good representation, and to update the representation, it already needs a good clustering. In practice, heuristics like alternating between clustering and representation update or joint optimization of both are used. Empirically, the alternating and joint optimization outperforms a sequential approach of first pre-training an autoencoder and then clustering its representation.

In this thesis, we focus on learning representations for clustering using autoencoders and prototypes. Prototypes are representatives of a cluster that we use as targets to transform the autoencoder representation. Autoencoders consist of three components: an encoder, a decoder, and a reconstruction objective. The encoder takes the input data and compresses it into a low-dimensional representation, while the decoder takes the compressed representation and aims to reconstruct the original input. The deviation

[1]According to a google scholar search in March, 2023.

between reconstruction and input is measured with the reconstruction loss, e.g., using the mean squared error. We focus on the autoencoder as it offers a general architecture that can be applied to many data modalities and is not limited to data that can be augmented. For instance, contrastive methods rely on extensive augmentation (e.g., random color changes of images) to avoid trivial solutions. Note, however the methods introduced in this thesis are not limited to autoencoders and can be used with other data dependent objectives from self-supervised learning.

1.1. Research Goal

This cumulative thesis contributes three methods for cluster analysis and presents the first empirical study of non-redundant clustering of high-resolution images of archaeological artifacts. All presented works are concerned with representation learning for clustering, removing the need for feature engineering during cluster analysis. This is achieved by jointly optimizing a prototype-based clustering objective and a data dependent objective, like the autoencoder reconstruction loss.

In developing the presented methods, the main goal was to make deep clustering methods more flexible and to allow the use of more advanced clustering methods. This research goal was motivated by the observation that deep clustering methods are designed around a particular clustering algorithm. This means that learning a representation for a specific clustering algorithm will inherit its assumptions. For instance, using k-Means as a base method will lead to learning a “k-Means friendly” [YFSH17] representation that consists of spherical clusters without consideration of non-convex shapes or alternative clustering solutions. Therefore, the choice of clustering method on which the deep clustering algorithm is based is essential.

There exist many non-deep clustering algorithms that would enhance the capabilities of deep clustering methods. We focused on incorporating ideas from non-redundant [MGFS12], subspace-centered [GHPB14, MYPB17], and consensus [SG02] clustering methods. These families of clustering methods enable deep clustering algorithms that find multiple, relevant clusterings (non-redundant clustering), that automatically determine the dimensions needed for clustering (subspace-centered clustering) or that combine multiple clustering algorithms into a single clustering solution (consensus clustering).

Following this research agenda led to many “firsts” during our work. We introduced the first deep Embedded Non-Redundant Clustering (ENRC) algorithm [MMA$^+$20], that finds multiple, non-redundant clusterings and conducted the first non-redundant clustering of images of archaeological artifacts with ENRC [MSL$^+$23]. With ACe/DeC (Autoencoder Centroid-based Deep Clustering) [MBM$^+$21] we proposed the first subspace-centered deep clustering algorithm. Using the ideas from subspace-centered clustering led to an algorithm that is more robust with respect to the choice of learning rate, removed difficult to tune hyperparameters and made the clustering result easier to interpret. DECCS (Deep Embedded Clustering with Consensus representationS) [MTW$^+$22a] is the first deep consensus clustering algorithm that can be used with many existing clustering methods to update clusterings and representation together. In [MTW$^+$22a], we also proposed the idea of a consensus representation, i.e., a representation on which an ensemble of clustering algorithms achieves the same consensus clustering.

In conclusion, the research goal and the methods presented form the basis for the subsequent chapters of this thesis. We provide an overview of these chapters below.

1.2. Thesis Structure

The thesis is structured in the following way. In Chapter 2 we provide some background that is directly related to the contributions made in this thesis. In Chapter 3 we present our detailed contributions and explain what the proposed methods have in common. In Chapter 4 we conclude the thesis by discussing the proposed methods with their limitations and point to exciting future work. The Appendix A contains the four contributed papers and the detailed listings of co-author contributions.

2. Background

The contributions of this work are based on the central fields of representation learning and cluster analysis. This chapter provides some necessary background information to help elaborate the contributions made in this thesis. We only briefly discuss each related area to the extent necessary, because the provided papers in Appendix A are already self-contained. Interested readers, will find detailed references in each section pointing to more related work.

2.1. Representation Learning

"Features matter" [GDDM14]. As the quote in Chapter 1 already implies, the success of data mining and machine learning methods depends on the features used to represent the data. Traditionally, this meant spending an extensive amount of time with engineering features that are suited for a specific downstream task. Representation learning [BCV13] can automate feature engineering and if done successfully will provide great results across domains and downstream tasks. Bengio, Courville, and Vincent [BCV13] defined it as "learning representations of the data that make it easier to extract useful information when building classifiers or other predictors".

There exist many algorithms for learning "rich representations" [GDDM14] that offer strong performance on several downstream tasks like classification, image segmentation or clustering [CKNH20, ZWW$^+$22, HCX$^+$22, CTM$^+$21]. We focus on the family of autoencoder methods [HS06, KW14, HCX$^+$22] that learn a representation by learning to reconstruct its input features as we explain next.

2.1.1. Autoencoders

The autoencoder is an unsupervised[1] algorithm that learns to predict its input data [HS06]. This is done by using an encoder to encode an input vector $x_i \in \mathbb{R}^D$ into a lower dimensional representation $z_i \in \mathbb{R}^d$ and then decoding it to reconstruct the original input as $\hat{x}_i \in \mathbb{R}^D$. The encoder and decoder are typically implemented as non-linear neural networks that minimize the reconstruction loss over the data set $X \in \mathbb{R}^{N \times D}$, e.g., using the mean squared error

$$\mathcal{L}_{rec} = \sum_{i}^{N} \|x_i - \hat{x}_i\|_2^2. \tag{2.1}$$

The autoencoder reconstruction loss serves as a proxy task to learn a non-linear dimensionality reduction of X to $Z \in \mathbb{R}^{N \times d}$, with $d < D$. In Figure 2.1 we show an illustration

[1]Whether it belongs to the self-supervised or unsupervised approaches is up to debate.

of a fully-connected autoencoder that was trained to reconstruct images of handwritten digits of zeros and ones[2]. The original 28×28 images are flattened to 784-dimensional

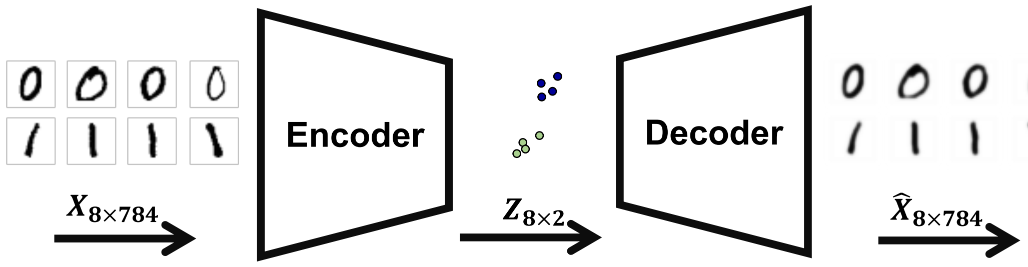


Figure 2.1.: Illustration of an autoencoder trained on a small sample of MNIST images of zeros and ones. The 2-dimensional representation Z shows two separated clusters of zeros and ones (blue and green).

vectors and encoded in a 2-dimensional representation. The encoded images are then transformed back to the original space using the decoder. The learned lower dimensional representation can be used for downstream tasks like clustering or classification.

For our methods, we chose the autoencoder framework, because of its generality. The reconstruction task can be applied for all data types and is not limited to data that can be augmented. Based on the data modality the encoder and decoder can be implemented with existing deep learning architectures like Convolutional Neural Networks [LB$^+$95], LSTMs [HS97] or Transformers [VSP$^+$17, DBK$^+$21]. Further, there exists a large variety of autoencoders, e.g., the Denoising Autoencoder [VLBM08], Contractive Autoencoder [RVM$^+$11], Stacked Autoencoder [BLPL06], Masked Autoencoder [HCX$^+$22], Variational Autoencoder [KW14] and many more.

2.2. Clustering

Clustering is an unsupervised learning task that, in its most general form, aims to group similar objects together and separates dissimilar objects. With such a definition we have many possible interpretations of what a cluster can be. Among other things it heavily depends on the notion of similarity. Therefore, every clustering algorithm has to make assumptions, either implicit or explicit, about the clusters it wants to find and what should be considered similar. The selection of a suitable clustering algorithm is dependent on the data and the goal of the analysis, which is why over the years many clustering algorithms have been proposed. For an overview of the broad literature on clustering see the surveys in [JMF99, KKZ09, Jai10, KKSZ11, BHR$^+$21, XSL23] or the books in [HKP11, LRU14, TSKK19]. In this thesis, we focus on a small selection of families of clustering methods, which we will address in the following sections.

[2]The images are part of the MNIST [LBB$^+$98] data set of handwritten digits and were taken from `https://en.wikipedia.org/wiki/MNIST_database`.

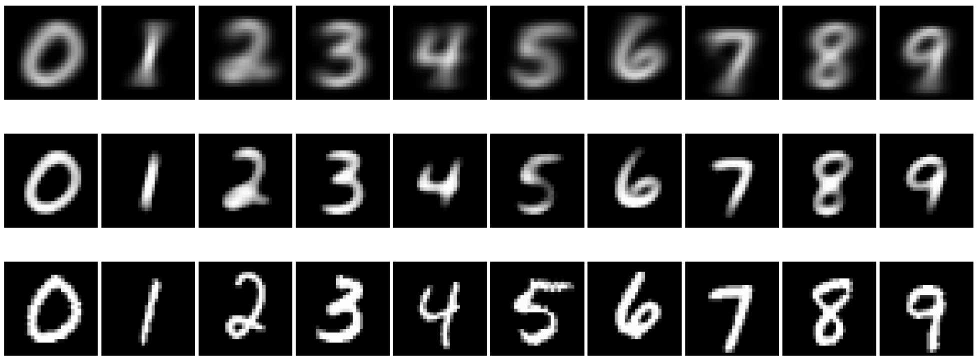


Figure 2.2.: Prototypes for MNIST w.r.t. the mean (upper), median (middle) and medoid (bottom) of each image per ground truth class (“0” to “9”).

2.2.1. Prototype-based Clustering

The most well-known prototype-based clustering algorithm is k-Means [Mac65, Llo82, Jai10]. In k-Means each cluster is represented by its centroid, which is simply the mean of all instances in the cluster. The centroid is a prototypical representation of its cluster and can be used to summarize the cluster members. Instead of the mean one can use the median along each dimension of all instances in a cluster, leading to the k-Medians [JD88] algorithm. Another prototype-based algorithm is k-Medoids [KR90], where clusters are represented by their medoids. A medoid is the most centrally located object in a cluster, meaning the prototype itself is an instance of the data. In Figure 2.2 we show mean, median and medoid prototypes for the classes of the MNIST [LBB$^+$98] data set for illustration. First, we see that the median prototypes for each digit are less blurry with sharper edges, whereas the mean prototypes are more blurred, and the medoids are representative instances of the data. Second, even though the medoids are data objects we see shared characteristics with the mean and median prototypes. For instance, the prototype of the digit “1” is right leaning for mean, median and medoid prototypes. There exist also prototype-based clustering algorithms that represent clusters with multiple prototypes, e.g., the k-Multiple-Means algorithm [NWL19]. For the development of the methods presented in this thesis we focused on centroid-based clustering methods, like k-Means, that use a single prototype per cluster which we explain next in more detail.

Centroid-based Clustering. As k-Means learns prototypes in the form of centroids it belongs to centroid-based clustering methods as well. k-Means has many available extensions [Jai10]. For deep clustering the stochastic gradient descent (SGD) k-Means [BB94] and mini-batch k-Means [Scu10] are most relevant, as they can be easily used with neural networks. Another benefit of using k-Means like objectives in deep clustering is that a cluster can be summarized by its prototype, which means that during optimization only the prototype and not the whole cluster needs to be processed. This is more efficient as the number of clusters is usually much smaller than the number of data points. For the sake of exposition, we will only briefly explain how k-Means works.

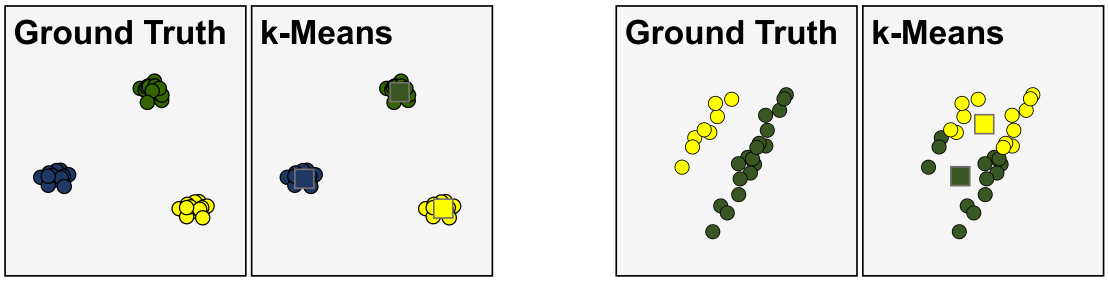


Figure 2.3.: Illustration of k-Means clustering for spherical (left) and non-spherical clusters (right). Corresponding centroids are plotted as squares.

k-Means learns centroids and cluster assignments in an alternating manner. First, centroids are initialized, e.g., randomly, from the data. Second, data objects are assigned to cluster centers based on a specified distance function. Third, based on the cluster assignments new centroids are calculated by averaging the objects inside each cluster. The second and third steps are repeated until convergence is reached. k-Means is fast and easy to apply, but it has several downsides. First, it can only find spherical-shaped clusters and it fails if this assumption is violated. In Figure 2.3 we show an example of spherical clusters and an example of non-spherical, but convex, clusters that k-Means clusters incorrectly. Note that the limitation of k-Means w.r.t. the non-spherical clusters is only important if the clusters are sufficiently close, and other algorithms such as expectation-maximization clustering [DLR77] can easily separate these two clusters. A second downside of k-Means is its sensitivity to noise and outlier points. Third, it is sensitive to imbalanced clusters, e.g., if one cluster contains many more samples than other clusters. Even though k-Means has these downsides it is still widely used, and usually one of the first algorithms applied during cluster analysis, due to its simplicity.

2.2.2. Subspace-centered Clustering

Clustering high-dimensional data is associated with the "curse of dimensionality", which refers to the problem that distances lose their meaning in high-dimensional spaces [BGRS99, HAK00, AHK01, KKZ09]. This can lead to sub-optimal clustering results as it becomes increasingly difficult to distinguish objects from each other. Subspace-centered clustering (a.k.a. common-subspace clustering) approaches, like Dip'n'Sub [BLBP23], SubKmeans [MYPB17] or FOSSCLU [GHPB14], avoid the "curse of dimensionality" by finding a lower dimensional optimal subspace for all clusters. Additionally, subspace-centered clustering methods assign each data object only to one cluster. This differentiates subspace-centered clustering from traditional subspace clustering [KKZ09], which assigns each cluster its own subspace and, depending on the algorithm, data objects can belong to multiple clusters. The main benefit of subspace-centered clustering methods over

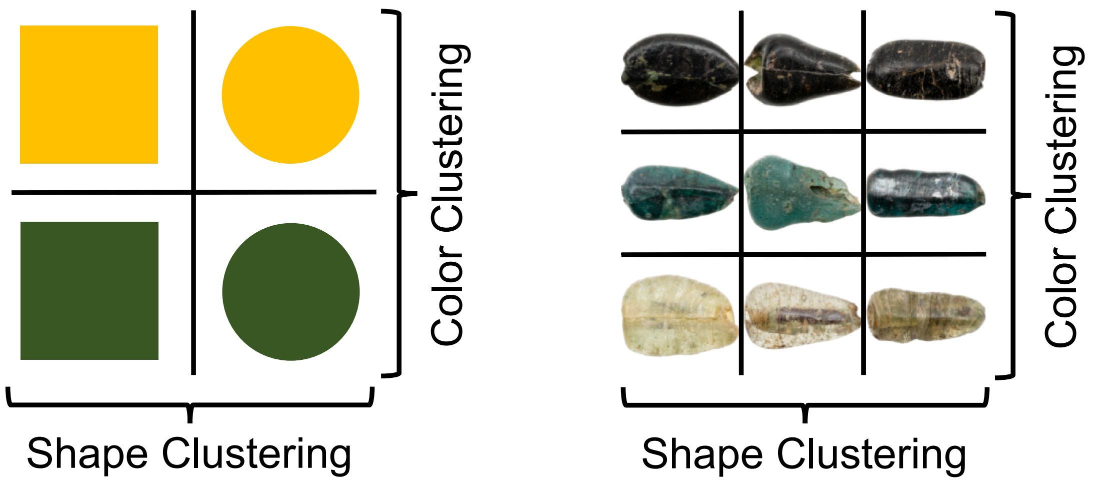


Figure 2.4.: Non-redundant clusterings according to shape and color for a synthetic (left) and a real world example of Medieval glass beads (right).

subspace clustering is that a single, common subspace for all clusters is found. This allows an easy comparison of objects in the single cluster subspace. Additionally, the cluster subspace is usually lower dimensional than the full data space, making visualizations easier as the cluster subspace already contains all information necessary for separating the clusters. For instance, the SubKmeans algorithm learns an optimal subspace for k-Means. It does so by splitting the data space into two arbitrarily oriented and orthogonal subspaces. A clustered subspace where the data objects are as close as possible to their respective cluster centers and a noise subspace where the data distribution is just a single Gaussian. For many real world data sets the found cluster subspace corresponds better to the discriminative structure of clustering than the variance preserving representation of Principal Component Analysis [Pea01], as shown in [MYPB17].

2.2.3. Non-Redundant Clustering

High-dimensional data can often be clustered in more than one meaningful way. Early subspace clustering algorithms, like CLIQUE [AGGR98] or SUBCLU [KKK04], approach the clustering of high-dimensional data by searching for all subspace clusters in all axis-parallel subspaces. A data object can then belong to multiple subspace clusters without considering whether the multiple clusterings are redundant. As a result, these approaches may lead to many clusters, making them difficult to analyze. Non-redundant clustering is motivated by this observation. Non-redundant clustering algorithms find multiple clustering solutions that are as different as possible, while still preserving relevant information from the data. This is also in contrast to classical clustering algorithms, like k-Means, which only find a single clustering solution. In Figure 2.4 we show two non-redundant clusterings. The left side depicts a toy example of geometric objects, and the right side shows a non-redundant clustering for Medieval glass beads from our cooperation with archaeologists [MSL$^+$23]. Here, both examples can be grouped according to their shape and color, two valid clusterings that provide different perspectives on the data.

Non-redundant clustering methods have been researched for about two decades. Several of the proposed methods extend existing clustering algorithms to handle multiple clusterings. The works in [CFD07, MYPB18, LMPB22] are based on k-Means, [YMHP16] uses expectation-maximization clustering [DLR77] and [NDJ10] extends spectral clustering [vL07]. After our deep non-redundant clustering algorithm ENRC [MMA$^+$20], the work of [FZW$^+$21] introduced a probabilistic deep non-redundant clustering algorithm using variational autoencoders [KW14]. See also [MGFS12] for an overview of non-redundant clustering algorithms.

For this thesis, mainly NrKMeans [MYPB18] is relevant. NrKMeans extends the SubKmeans [MYPB17] algorithm to handle multiple non-redundant k-Means clusterings. NrKMeans needs the number of clusterings and the number of clusters in each clustering as input parameters. Given these input parameters NrKMeans performs k-Means in each subspace and optimizes the subspaces using eigenvalue decompositions. Specifying the input parameters of NrKMeans is difficult in practice. AutoNR [LMPB22] extends NrKmeans to automatically estimate these input parameters using the Minimum Description Length Principle [Ris78, Grü05] making it more flexible to use.

2.2.4. Consensus Clustering

Consensus clustering combines multiple clustering results into a single robust clustering solution [SG02]. The underlying motivation is that the individual clustering algorithms all have limiting assumptions and may only partially discover relevant clusterings. Combining all the clustering algorithms into one ensemble can lead to better clustering solutions than considering each clustering individually.

Most consensus clustering algorithms consist of the following steps. First, a set of base clusterings is generated using individual clustering algorithms. Second, the base clusterings are combined using a consensus function to obtain a single robust clustering. The goal of the consensus function is to combine a set of base clusterings into one final consensus clustering π_{cc}, such that π_{cc} agrees as much as possible with the base clusterings. A common choice for measuring the agreement is the average normalized mutual information (NMI) [VEB10] between the consensus clustering and the base clusterings [SG02]:

$$\pi_{cc} = \mathrm{argmax}_{\bar{\pi}} \sum_{i=1}^{|\Pi|} \mathrm{NMI}(\pi_i, \bar{\pi}), \tag{2.2}$$

where Π is the set of base clusterings π_i. Using the NMI as a measurement of agreement has the benefit of being invariant to the permutation and absolute values of cluster labels. Further, the number of clusters can vary between each clustering. Additionally, the NMI is symmetric. According to [GAH$^+$21], a good consensus function should be robust (better average performance than the single partitionings), stable (outliers, i.e., poor partitionings, should not degrade the result) and scalable to large data sets.

Depending on the consensus function consensus clustering methods can be categorized into partition-based and object co-occurrence-based methods. Median partition methods find a partition that is most similar to all the base partitions, see [CS02, TJP05, LDJ07] for several methods from this family. Object co-occurrence based methods utilize the co-association matrix to find the ideal partitioning, where the entries of this matrix reflect how often every two instances are partitioned together, see [FJ05, LBR$^+$15, HWL18] for example methods. A common strategy for consensus clustering of high-dimensional data is to use random projections [Joh84, BM01] as done in [FB03, PKBZ15]. Using random projections is limited to linear transformations, which is why several non-linear consensus clustering methods using deep learning have been proposed [LSLF16, TLL$^+$19, RDMD21]. For instance, the deep consensus clustering method ConCURL [RDMD21] leverages image augmentations and random projections for learning a cluster ensemble using Softmax predictions for different augmented views.

2.2.5. Deep Clustering

Deep clustering combines deep learning methods with cluster objectives to learn representations that are more suited for clustering also called “cluster-friendly” [YFSH17]. For recent surveys see [AGSC18, MGL$^+$18, RPY$^+$22, ZXZ$^+$22]. Learning a “cluster-friendly” representation is only a loosely defined notion and it depends on which cluster assumptions we make. The general idea is that the deep clustering algorithm changes the representation such that it better represents the underlying assumptions of the cluster model. For that the usual procedure is to pre-train a model, e.g., an autoencoder, to obtain an initial representation and then cluster the representation with the algorithm of choice. The initial clustering serves as a target to fine-tune the pre-trained model. The fine-tuning can be done in an alternating or joint manner. The alternating variant works similar to Expectation-Maximization [DLR77] by alternating between clustering of the representation and updating (fine-tuning) the representation w.r.t. the obtained labels. For instance, the DCN [YFSH17] algorithm alternates between a k-Means update and a representation update given the k-Means centroids. The joint optimization variant uses either soft cluster labels or a form of mini-batch clustering, e.g., [Scu10], to optimize both the clustering and representation in a single update step. The algorithm DEC [XGF16] uses centroids and soft cluster labels to jointly optimize a soft auxiliary target distribution with the Kullback-Leibler divergence. In Figure 2.5 we illustrate how deep clustering works on a toy example with two exemplary cluster-friendly representations. Option A in Figure 2.5 shows a representation that has distinct correlation clusters [KKZ09] and separated outlier points[3]. Option B depicts a k-Means friendly representation, where clusters are spherical and outliers have been assigned to clusters. Option B is what many deep clustering algorithms focus on, e.g., DEC and DCN, because the centroids used in k-Means can be easily integrated in deep learning objectives. For instance, the centroids can serve as targets to which the embedded data objects are “moved” closer to

[3]Note, that to the best of our knowledge a deep clustering algorithm as in option A does not exist yet. The depicted example just serves as an illustration.

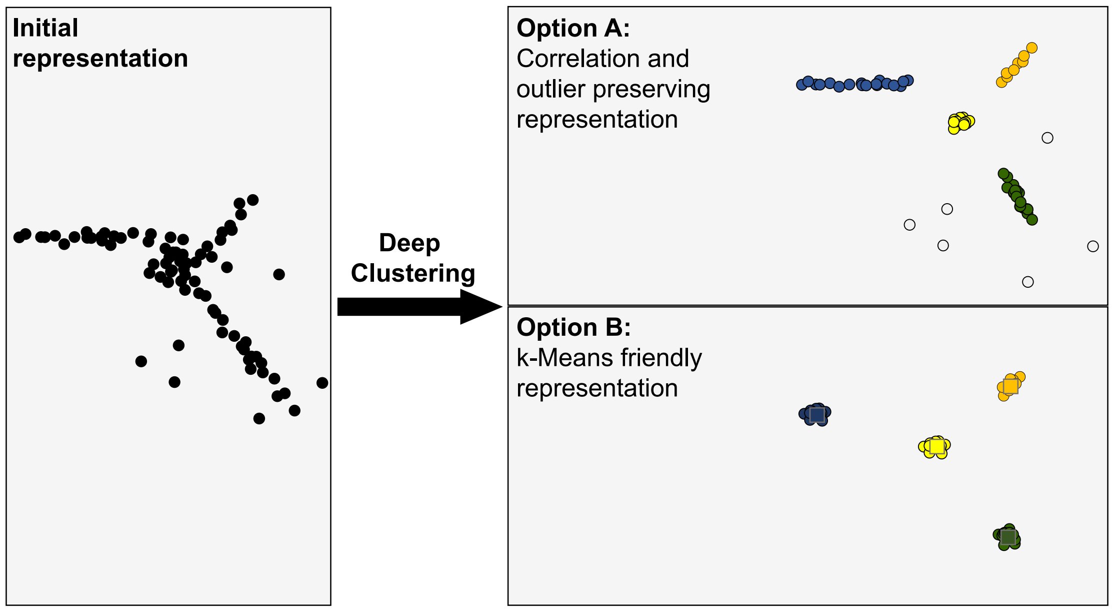


Figure 2.5.: Illustration of deep clustering to learn "cluster-friendly" representations. (Left) An initial representation learned from the data, e.g., using an autoencoder, with no clear cluster structure. (Right) Depending on the assumed cluster model the deep clustering approach moves towards two different representations and cluster assignments (in color). **Option A:** Correlations within clusters are preserved and clusters are better separated from each other. Outlier points are not assigned to any cluster. **Option B:** The representation is more k-Means friendly containing only spherical clusters (corresponding centroids are plotted as squares). Outliers are assigned to clusters.

by minimizing their distance.

In general, deep clustering methods optimize a cluster objective together with a data dependent regularizer. The latter is important to avoid trivial solutions, like mapping all embedded points to a single cluster. The typical [AGSC18] objective of deep clustering methods is

$$\mathcal{L} = \lambda_1 \mathcal{L}_{cluster} + \lambda_2 \mathcal{L}_{reg}, \tag{2.3}$$

where $\mathcal{L}_{cluster}$ is the cluster objective and $\mathcal{L}_{reg}$ is a data dependent regularizer, e.g., the autoencoder reconstruction loss (Equation 2.1). $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$ are hyperparameters that specify the trade-off between clustering and data dependent objective. In some methods they are also defined in relation to each other, e.g., $\lambda_2 = (1 - \lambda_1)$ with $0 \leq \lambda_1 \leq 1$. An instance of the general loss in Equation 2.3 is the loss of DCN [YFSH17], here shown for a single object i,

$$\mathcal{L}^i = \lambda_1 \|z_i - \mu_i\|_2^2 + \lambda_2 \|x_i - \hat{x}_i\|_2^2, \tag{2.4}$$

which consists of a k-Means like loss (with fixed centroid μ_i for embedded sample z_i during the representation update) and an autoencoder reconstruction loss in the form of the mean squared error. During the representation update of DCN the embedded data point z_i is positioned closer to its corresponding centroid μ_i. Note that without the autoencoder reconstruction loss, we could minimize Equation 2.4 by setting all z_i and correspondingly all centroids μ_i to a constant leading to a trivial solution.

To summarize, for learning cluster-friendly representation we need to make several design choices. First, we need an algorithm to learn an initial representation that can be modified, e.g., an autoencoder. Second, we need to select a clustering algorithm for which we want to learn a suitable representation. Third, we need to combine the clustering and the representation learning algorithm and optimize a common objective. The third step introduces a circular dependency problem. If we want to learn a good representation given a clustering solution, we already need to have a suitable representation for our clustering algorithm. In practice, heuristics like alternating optimization (updating cluster labels and representation separately) or joint optimization (updating cluster labels and representation together) are used. There exist many variations of this “recipe”, especially the optimization is difficult and requires a high engineering effort, but the general outline is similar for most methods.

3. Contributions

In this chapter, we explain each method briefly, discuss their contribution in detail and highlight their common properties. During the development of our deep clustering methods, we focused on adding new capabilities that are based on approaches from non-deep methods. We introduced the first deep non-redundant clustering algorithm ENRC (Embedded Non-Redundant Clustering) [MMA$^+$20] and conducted the first non-redundant clustering of images of archaeological artifacts [MSL$^+$23] with ENRC. With ACe/DeC (Autoencoder Centroid-based Deep Clustering) [MBM$^+$21] we proposed the first deep subspace-centered clustering algorithm that learns a separate cluster space in the autoencoder representation. Additionally, we introduced DECCS (Deep Embedded Clustering with Consensus representationS), a deep consensus clustering method [MTW$^+$22a] that can be used with existing clustering methods and is not limited to k-Means like clusters.

Inspired by the usage of visual abstracts[1] in other fields [RC20] we decided to construct one visual abstract for each work to explain it in a single figure. All further details are presented in the corresponding papers in Appendix A. Next, we explain the common properties of the introduced methods.

3.1. Common Properties of Our Proposed Methods

The contributed methods all learn different representations depending on the underlying clustering approach, but they have several common aspects. All methods are using the autoencoder framework for learning an initial representation and use it as a data dependent regularizer [LPW18] to avoid trivial solutions. The methods use a learned non-linear transformation to improve the performance of the clustering algorithm. They achieve this by learning prototypes that serve as targets for the non-linear optimization and therefore belong to prototype-based clustering methods. During training the representation is then jointly transformed by updating the cluster assignments and prototypes simultaneously. This leads to better separated and more compact clusters by "moving" embedded data instances closer to their prototypes. ENRC and ACe/DeC are both based on k-Means, but while DECCS uses prototypes as well it is not bound to the k-Means assumptions as it combines several clustering methods into a single consensus clustering. Overall, our methods can be summarized under the umbrella of *prototype-based representation learning with deep clustering* and we explain the detailed contribution of each method next.

[1]See `https://www.elsevier.com/authors/tools-and-resources/visual-abstract` for some examples.

3.2. Representations for Non-Redundant Clustering

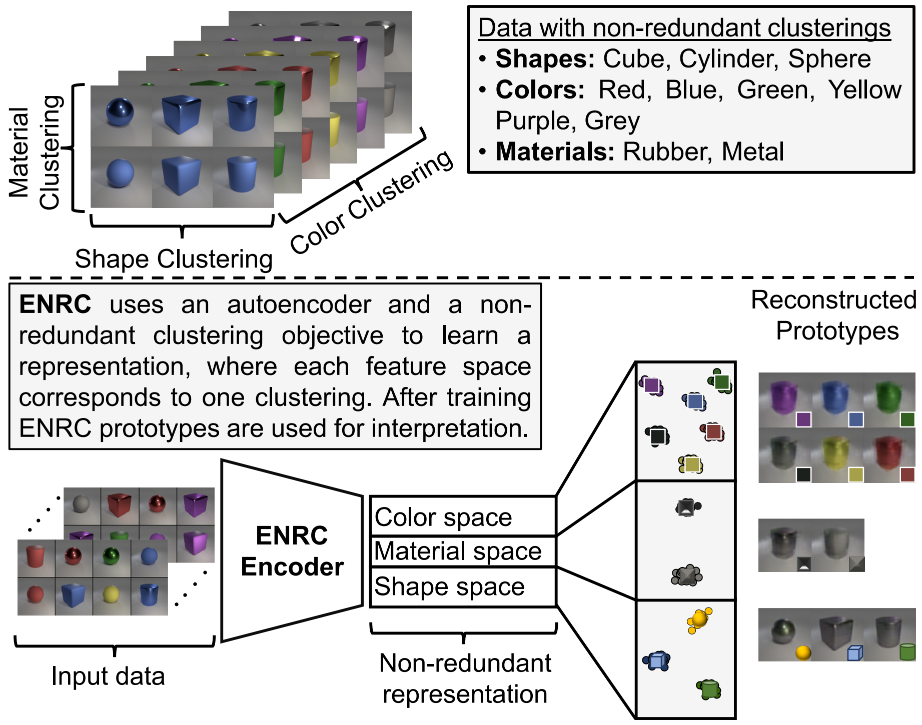

Data with non-redundant clusterings
• Shapes: Cube, Cylinder, Sphere
• Colors: Red, Blue, Green, Yellow Purple, Grey
• Materials: Rubber, Metal
Material Clustering
Shape Clustering
Color Clustering
ENRC uses an autoencoder and a non-redundant clustering objective to learn a representation, where each feature space corresponds to one clustering. After training ENRC prototypes are used for interpretation.
Input data
ENRC Encoder
Color space
Material space
Shape space
Non-redundant representation
Reconstructed Prototypes

Figure 3.1.: **Visual abstract for ENRC.** The upper part shows a data set of objects that can be clustered according to shapes, colors, and materials. Once ENRC is trained we can use the learned non-redundant representation to interpret the different clusterings (lower part). Each prototype in the non-redundant representation is reconstructed using the decoder (not shown) and can be interpreted in the following way: The reconstructed prototypes only represent the discriminative features to distinguish the clusters, and all other properties not relevant for this clustering are averaged. The averaging makes color prototypes blurry and shape prototypes sharp, but without colors.

We introduced in [MMA$^+$20] (Appendix A.1) the first deep Embedded Non-Redundant Clustering (ENRC) algorithm. In Figure 3.1 we explain the main ideas of ENRC. ENRC finds multiple, non-redundant clusterings using deep learning. This is in contrast to k-Means-based, deep clustering methods like DEC [XGF16], which can only find a single clustering. For instance, images can often be grouped w.r.t. their color and their shape. Here, DEC would find a single clustering that is a mixture of color and shape, while ENRC is able to extract both individual clusterings.

We showed in [MMA$^+$20] that ENRC can automatically learn which features are important for each clustering making the learned non-redundant representation interpretable. Additionally, ENRC outperformed non-redundant clustering algorithms which were applied to the same pre-trained autoencoders. This indicates the benefit of a joint deep clustering approach that fine-tunes the autoencoder further vs. a sequential approach of just pre-training an autoencoder and then clustering the representation.

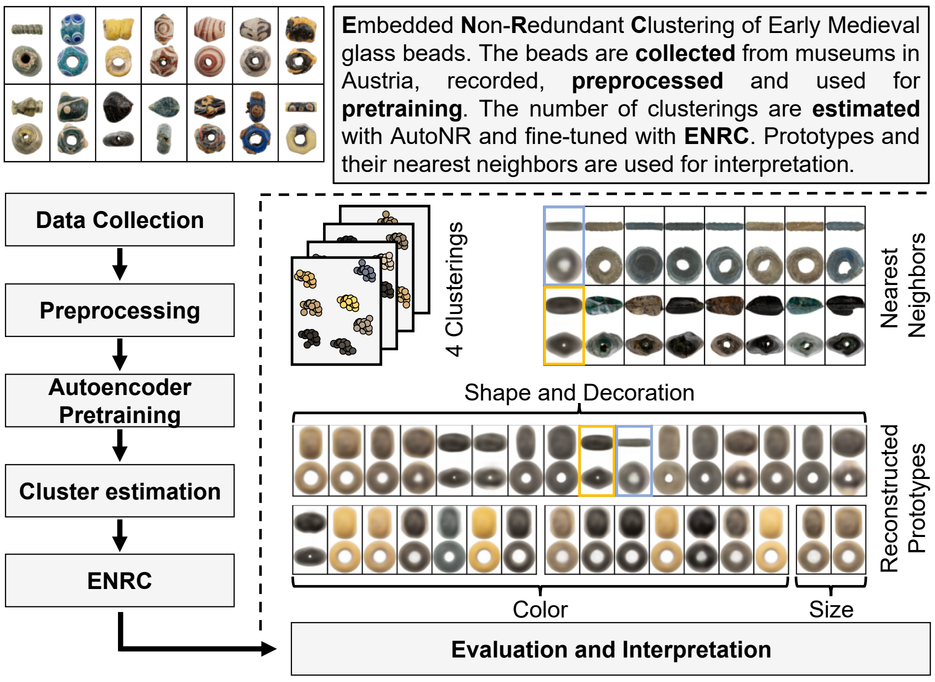

Embedded **N**on-**R**edundant **C**lustering of Early Medieval glass beads. The beads are **collected** from museums in Austria, recorded, **preprocessed** and used for **pretraining**. The number of clusterings are **estimated** with AutoNR and fine-tuned with **ENRC**. Prototypes and their nearest neighbors are used for interpretation.
Data Collection
Preprocessing
Autoencoder Pretraining
Cluster estimation
ENRC
4 Clusterings
Nearest Neighbors
Shape and Decoration
Reconstructed Prototypes
Color
Size
Evaluation and Interpretation

Figure 3.2.: **Visual abstract for non-redundant clustering of Early Medieval glass beads.** The upper, left part shows a diverse sample of glass beads varying in color, shape, and decoration (recorded from a side and top view). The processing pipeline from data collection to interpretation is shown on the lower, left part. ENRC found four non-redundant clusterings, from which we reconstructed the prototypes for interpretation. The upper row of the reconstructed prototypes shows a clustering w.r.t. shapes and decorations. The two sets of prototypes on the lower left depict clusterings according to color and the two prototypes in the lower right show a clustering according to longitudinal size. The seven nearest neighbors for the two shape prototypes show a high visual similarity indicating that the learned representation preserved meaningful characteristics of the data.

We also applied ENRC to a real world data set of Early Medieval glass beads that we collected together with collaboration partners from archaeology [MSL+23] (Appendix A.4). For this work we extended ENRC to learn augmentation invariances, handle multiple views and multiple modalities. In Figure 3.2 we explain our approach and show some qualitative findings. During this work we identified several challenges. First, the data is very imbalanced, containing mostly simple beads without decoration and only yellow, brown, or black colors. Second, the sample size of about 4,700 glass beads is relatively small for deep learning methods and considering the large variety of the beads. Third, even though in principle the color and shape of the glass beads can be varied independently of each other, the sample we have is biased. For instance, most yellow and black beads are barrel shaped, which leads to a correlation of shape and color features. This breaks an assumption of non-redundant clustering, namely that multiple clusterings are non-overlapping, e.g., orthogonal, or statistically independent. ENRC uses a soft assignment mechanism that allows a partial overlap of clusterings, mitigating this problem. We have further alleviated this problem by generating additional images using augmentations, but without a larger, unbiased sample, it is difficult to fully resolve.

This can be seen in the prototypes in Figure 3.2 as well, while the color prototypes contain mostly color information, there is still some variation in shape. The reconstructed shape prototypes show mostly shape information, but some variation for lighter and darker colors is still visible. Nonetheless, to the best of our knowledge this work was the first application of non-redundant clustering to archaeological image data. We showed that even in a real-world scenario where ground truth clusterings are not known, we successfully applied ENRC to find interesting clusterings that overlapped with domain experts knowledge.

3.3. Autoencoder Representations for Centroid-based Clustering

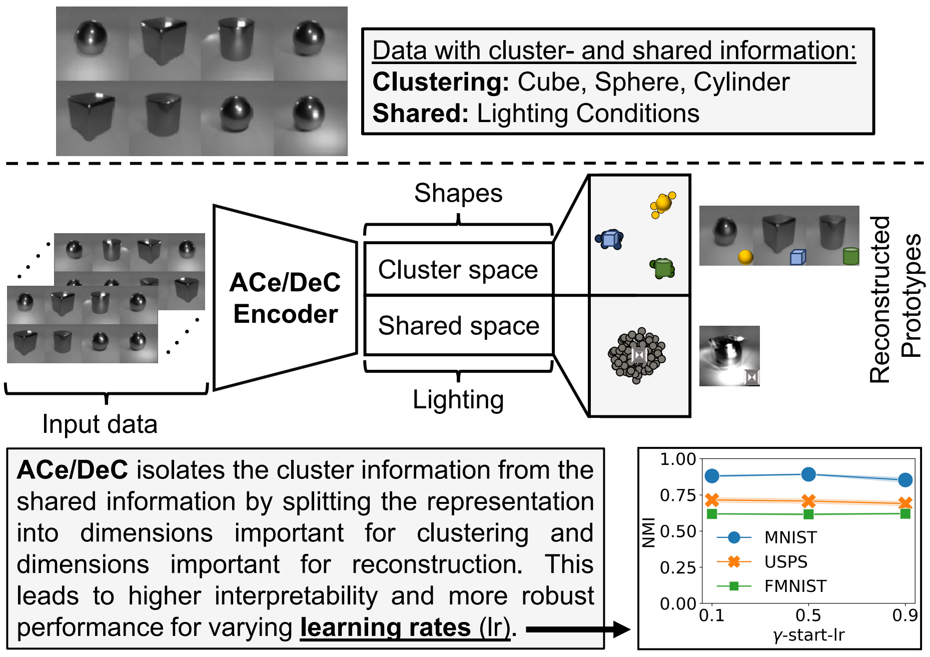
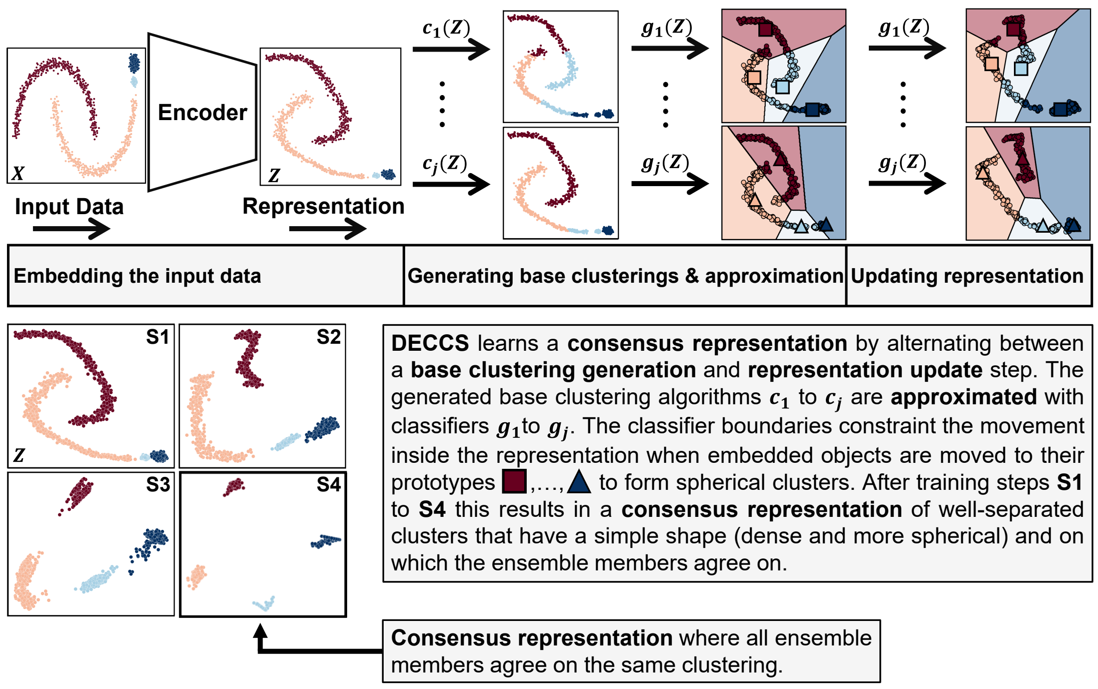

Data with cluster- and shared information:
Clustering: Cube, Sphere, Cylinder
Shared: Lighting Conditions
Shapes
ACe/DeC Encoder
Cluster space
Shared space
Lighting
Input data
Reconstructed Prototypes
ACe/DeC isolates the cluster information from the shared information by splitting the representation into dimensions important for clustering and dimensions important for reconstruction. This leads to higher interpretability and more robust performance for varying learning rates (lr).
NMI
1.00
0.75
0.50
0.25
0.00
0.1
0.5
0.9
γ-start-lr
MNIST
USPS
FMNIST

Figure 3.3.: **Visual abstract for ACe/DeC.** The upper part shows a data set of synthetic objects that can be clustered according to shapes, while the lighting is mostly relevant for reconstruction. Once ACe/DeC is trained we can use the learned representation to interpret which information is important for discriminating the clusters and which information is shared across objects. The lower part shows the reconstructed prototypes for the cluster and shared space. The reconstructed cluster prototypes correspond to the three shapes, while the lighting is less present. The shared space prototype is an average over the shapes but represents the light information. The split representation leads to stable performance for different learning rates for several benchmark data sets (plot on lower right), due to resolving the trade-off between reconstruction and clustering objectives (explanation in text below).

In [MBM$^+$21] (Appendix A.2) we introduced the ACe/DeC (Autoencoder Centroid-based Deep Clustering) framework which automatically learns to split the embedded space of an autoencoder. In Figure 3.3 we explain our approach and show some qualitative and quantitative results. ACe/DeC is inspired by the capability of ENRC to automatically learn which features are important for which clustering. We used this idea to learn to split the autoencoder representation into a set of features that are important for clustering (cluster space) and those which are not (shared space). ACe/DeC models the shared space with a single cluster to capture the features that are shared across all clusters and are mostly important for preserving additional details about the objects needed for reconstruction. ACe/DeC is related to subspace-centered clustering methods by learning a separate space for all clusters.

Clustering-Reconstruction Trade-off. Our motivation for ACe/DeC was the observation that the reconstruction and clustering objective in deep clustering methods have different goals that are not necessarily aligned. This has been observed in previous work for autoencoder-based semi-supervised learning [EM19] as well. For instance, the clustering objective learns which features are helpful for discriminating the clusters (e.g., the shapes in Figure 3.3), while the reconstruction objective preserves features important for reconstruction (e.g., lighting conditions). In most methods this trade-off is handled via the hyperparameters λ_1 and λ_2 that weight the cluster and reconstruction objective, as shown in Equation 2.4. This has the downside of requiring access to a form of supervision, e.g., ground truth labels, to tune the additional hyperparameters. ACe/DeC does not rely on these hyperparameters. Instead, ACe/DeC learns an individual weight for each dimension indicating whether the dimension belongs to the cluster or shared space. The individual weights create a more flexible way to handle the clustering-reconstruction trade-off than global (per loss) hyperparameters would allow. This idea can be used for existing centroid-based deep clustering methods like DCN [YFSH17]. We showed in experiments that the usage of ACe/DeC made DCN much more stable w.r.t. the choice of learning rate (lower, right plot in Figure 3.3), a crucial hyperparameter in deep learning. Additionally, it has the benefit of making the clustering more interpretable by removing information that is not needed for clustering.

3.4. Consensus Representations



DECCS learns a **consensus representation** by alternating between a **base clustering generation** and **representation update** step. The generated base clustering algorithms c_1 to c_j are **approximated** with classifiers $g_1$to g_j. The classifier boundaries constraint the movement inside the representation when embedded objects are moved to their prototypes ■,…,▲ to form spherical clusters. After training steps **S1** to **S4** this results in a **consensus representation** of well-separated clusters that have a simple shape (dense and more spherical) and on which the ensemble members agree on.

Consensus representation where all ensemble members agree on the same clustering.

Figure 3.4.: **Visual abstract for DECCS.** The upper part shows a synthetic input data set of two half-moon shaped clusters and two Gaussian clusters. DECCS uses the encoder of a pre-trained autoencoder to get the initial representation. Given the initial representation DECCS conducts its main steps using prototypes and constraints learned from the cluster ensemble. The lower part shows how DECCS transforms the initial representation (S1) in several update steps to the final consensus representation (S4).

In [MTW$^+$22a] (Appendix A.3) we proposed DECCS (Deep Embedded Clustering with Consensus representationS) and in Figure 3.4 we show an overview of its main steps. DECCS is the first deep clustering algorithm that can combine multiple, heterogeneous clustering algorithms into a single clustering by jointly updating the representation and clusterings. This is in contrast to many previously existing deep clustering methods, which are usually limited by the assumption of a single clustering algorithm. In difference to the deep consensus clustering method ConCURL [RDMD21], DECCS (depending on the used neural network) can be used with any data type and does not rely on augmentations to work well. Further, DECCS can include existing clustering algorithms, whereas ConCURL uses an ensemble of Softmax predictions for different augmented views. One key notion that we defined in [MTW$^+$22a] is the *consensus representation*, which is a learned representation for which all ensemble members reach a clustering consensus[2]. DECCS can learn an instance of a consensus representation that is motivated by the observation that most clustering algorithms can detect compact and well-separated clusters in a low-dimensional space (S4 in Figure 3.4). Learning a consensus representation allows DECCS to learn a consensus function that combines the base clusterings into a single consensus clustering. This is in contrast to other consensus clustering approaches, where a consensus function needs to be pre-specified.

[2]We provide more information on the consensus representation together with a detailed definition in Appendix A.3 (Definition 1).

4. Conclusion

In the following we split the conclusion of this thesis in a discussion, limitations and future work section and provide some final remarks. The Appendix contains all four papers with supplementary material and additional information, including the detailed listings of co-author contributions.

4.1. Discussion

Autoencoding vs. Contrastive Learning. All introduced methods (ENRC, ACe/DeC and DECCS) are based on the autoencoder framework to learn an initial representation and to regularize the embedded space during the deep clustering procedure. The autoencoder is not strictly necessary, in principle one could replace the autoencoder, with any other representation learning algorithm, e.g., contrastive methods [CKNH20, CMM$^+$20] for image data. One important design decision to consider is that the objective of the representation learning algorithm is aligned to the clustering objective. A negative example would be learning a color invariant representation using a contrastive method during pretraining but wanting to cluster objects according to their color afterwards. Unfortunately, many contrastive methods rely on extensive image augmentations, including random color transformations, to perform well. These extensive augmentations are needed to become invariant against "unnecessary" details for downstream tasks like classification. For instance, SimCLR ([CKNH20], Figure 5) reported a drop in ImageNet1K [DDS$^+$09] classification accuracy of approximately 28% when only minimal augmentations (random cropping and flipping) where used. This was also a reason why we chose the autoencoder framework for clustering the glass bead data in [MSL$^+$23], as color was an important property we wanted to preserve.

Domain Knowledge and Cluster-wise Invariances. Most deep clustering methods allow the inclusion of domain knowledge in the form of augmentations to steer the clustering solution in the desired direction. For instance, consider the task of clustering the MNIST [LBB$^+$98] data set of images of hand-written numerals from "0" to "9". In [MBM$^+$21] we found that the Normalized Mutual Information (NMI) [VEB10] w.r.t. the ground truth labels of MNIST increases by 6% in relative performance when cluster predictions do not change due to slight translations and rotations[1]. We refer to this desired property as *cluster-wise invariance* to distinguish it from the *instance-wise invariance* used in contrastive learning. Prototype-based deep clustering methods, like ours, can easily

[1]Larger random rotations of images in MNIST should be avoided as it could change the meaning of digits "6" and "9".

integrate cluster-wise invariances by “moving” both the embeddings of the original and augmented object to the same cluster prototype. The prototype determines the cluster assignments, so objects that are closely embedded to their corresponding prototype will share the same cluster label. We did not evaluate all our proposed methods with additional augmentations, as we benchmarked our methods also on data sets that do not have a commonly used set of augmentations. For instance, in [MTW$^+$22a] we used a data set based on protein expression levels, where it is not clear which augmentations are meaningful. We want to note that in principle all our methods can use additional augmentations and should profit from them if the augmentation invariances align with the desired clustering.

4.2. Limitations

Dependence on Hyperparameters. Deep clustering methods have many limitations that make them difficult to use in practice. Among other things, they require large data sets, specialized hardware, and a long training period. One of the most difficult problems to solve for deep clustering methods in practice is tuning the hyperparameters without access to ground truth labels. For deep clustering methods, this is particularly difficult because there are many possible hyperparameters to tune for the neural network, but also for the clustering algorithm. Furthermore, the objective used in training may not result in a learned representation that corresponds to the desired ground truth clustering. We also faced this issue during the development of our methods. We limited the dependence on hyperparameters by using the same hyperparameter setting across data sets for most experiments. In the hope that such a setting will more likely transfer to other data sets. Some positive evidence for that comes from the independent work of [YFB21]. They applied our ACe/DeC algorithm in another benchmark with our provided implementation and hyperparameters, where it performed in the top-5 among 18 compared methods. Additionally, in the real-world application of ENRC to images of Early Medieval Glass beads [MSL$^+$23], we also had no access to ground truth labels, but were able to find clusterings that overlapped with domain expert knowledge. We achieved this by tuning the autoencoder parameters w.r.t. the reconstruction error and selected the best clustering solutions with multiple unsupervised clustering metrics. Of course, this does not fix the underlying dependence on hyperparameters and the increased computational cost that arises during hyperparameter search. We believe that this is an important avenue for future research for deep clustering, but also for other self-supervised deep learning methods.

Limited Interpretability. Deep neural networks are black box algorithms that are difficult to interpret without specialized visualization methods [OSJ$^+$18]. There have been some deep learning approaches with built-in interpretability, like [CLT$^+$19], but they are not yet widely used and most research still focuses on improving benchmark performance. The limited interpretability of deep learning affected our methods as well, making the embedded space in which clusters are found difficult to understand. Autoencoder centroid-based deep clustering methods can decode the prototypes that are found during clustering using the decoder. For image data this can help with interpreting the found clustering, e.g., in [MSL+23] we found that the prototypes of the glass bead data correspond to known depictions of bead types (Figure 11 in [MSL+23]). Further, ACe/DeC and ENRC are splitting the embedded space according to the different clusterings such that the latent dimensions encode information that is important for each clustering. We explored this with ACe/DeC by interpolating the samples in the shared space and reconstructing the interpolated samples. We found that the interpolations in the shared space of ACe/DeC encoded information more important for the reconstruction task, like different styles of writing the digit "7" in MNIST, see Figure 3 in [MBM+21]. Using prototypes and autoencoder reconstructions can be challenging when trying to interpret high resolution natural image data, due to a large amount of variation in the background and low generation quality of autoencoder reconstructions. [BBV21] proposed a diffusion model that generates high resolution images from embedded spaces for interpretation, that could be used with our methods as well. Whether this works for other data modalities and is helpful in practice remains to be seen.

4.3. Future Work

Semi-Supervised Learning. ENRC [MMA+20] and ACe/DeC [MBM+21] would be suited for semi-supervised extensions in multi-label and single-label classification. This could be achieved by including the supervision with a classifier inside the autoencoder embedded space as in [LPW18], calculating prototypes based on a small set of labeled instances or a combination of both. We believe that this would lead to more data efficient and interpretable algorithms, by splitting the embedded space in separate parts for each clustering.

Self-Supervised Learning We would like to investigate the performance of our contributed algorithms when used with self-supervised or contrastive representation learning algorithms. We believe this might further improve the clustering performance on image data. Especially, since contrastive and deep clustering methods both share a discriminative objective and not a generative one, like the autoencoder reconstruction loss. Additionally, contrastive methods, showed that they can achieve a high classification performance on common benchmark data sets [CTM+21]. There, has also been some theoretical and empirical work that provides evidence that contrastive methods are implicitly learning a clustering of the input data [PCS+23, SPA+19, HM22, CFN+22, ABD+22]. Contrastive approaches are limited to data where augmentation is possible, but they are still a promising avenue of future deep clustering research. In fact, existing combinations of contrastive learning with deep clustering are already showing great results [LHL+21].

Vision Transformers and Masked Autoencoders. The recent work of [HCX+22] introduced the Masked Autoencoder (MAE). The MAE uses a Vision Transformer (ViT)

[DBK$^+$21] together with a patch masking strategy to learn a representation for downstream tasks like supervised finetuning or image segmentation. The MAE revived to some degree the interest in autoencoding images in pixel space, an approach that was neglected partially due to the success of other self-supervised methods. Our proposed methods are all based on autoencoders and subsequently improvements in autoencoding should lead to improvements for autoencoder-based deep clustering methods. We would like to do a unified benchmarking across multiple deep clustering algorithms with different backbone autoencoder architectures (fully connected, convolutional and ViT) to measure the performance gain from “stronger” architectures. Together with collaborators we created the open-source ClustPy package[2] that already includes several deep clustering algorithms and with which we plan to conduct the benchmarking study.

Overall, we are excited about research opportunities on the intersection of clustering and representation learning.

4.4. Final Remarks

For this thesis we developed three algorithms for cluster analysis and conducted the first empirical study of non-redundant clustering of archaeological artefacts. As described, each algorithm represents an innovation in the field of deep clustering and will hopefully stimulate further future work. Compared to the growing scientific literature, this work is only a small contribution, but we hope it will help practitioners and enable researchers to discover new patterns in data.

[2]https://github.com/collinleiber/ClustPy

Bibliography

[ABD$^+$22] Mahmoud Assran, Randall Balestriero, Quentin Duval, Florian Bordes, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael G. Rabbat, and Nicolas Ballas. The hidden uniform cluster prior in self-supervised learning. *CoRR*, abs/2210.07277, 2022.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105. ACM Press, 1998.

[AGSC18] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *CoRR*, abs/1801.07648, 2018.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

[ARV20] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *ICLR*. OpenReview.net, 2020.

[BB94] Léon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *NIPS*, pages 585–592. MIT Press, 1994.

[BBV21] Florian Bordes, Randall Balestriero, and Pascal Vincent. High fidelity visualization of what your self-supervised representation knows about. *CoRR*, abs/2112.09164, 2021.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[BGRS99] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is ”nearest neighbor” meaningful? In *ICDT*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

[BHR$^+$21] Juhee Bae, Tove Helldin, Maria Riveiro, Slawomir Nowaczyk, Mohamed-Rafik Bouguelia, and Göran Falkman. Interactive clustering: A comprehensive review. *ACM Comput. Surv.*, 53(1):1:1–1:39, 2021.

[BLBP23] Lena G. M. Bauer, Collin Leiber, Christian Böhm, and Claudia Plant. Extension of the dip-test repertoire-efficient and differentiable p-value calculation for clustering. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pages 109–117. SIAM, 2023.

[BLPL06] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, pages 153–160. MIT Press, 2006.

[BM01] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD*, pages 245–250. ACM, 2001.

[But02] Atul Butte. The use and analysis of microarray data. *Nature reviews drug discovery*, 1(12):951–960, 2002.

[CBJD18] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV (14)*, volume 11218 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2018.

[CFD07] Ying Cui, Xiaoli Z. Fern, and Jennifer G. Dy. Non-redundant multi-view clustering via orthogonalization. In *Proceedings of the 7th IEEE ICDM, 2007*, pages 133–142, 2007.

[CFN$^+$22] Mayee F. Chen, Daniel Y. Fu, Avanika Narayan, Michael Zhang, Zhao Song, Kayvon Fatahalian, and Christopher Ré. Perfectly balanced: Improving transfer and robustness of supervised contrastive learning. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 3090–3122. PMLR, 2022.

[CKNH20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020.

[CLT$^+$19] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan Su. This looks like that: Deep learning for interpretable image recognition. In *NeurIPS*, pages 8928–8939, 2019.

[CMM$^+$20] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

[CS02] Dana Cristofor and Dan A. Simovici. Finding median partitions using information-theoretical-based genetic algorithms. *J. Univers. Comput. Sci.*, 8(2):153–172, 2002.

[CTM$^+$21] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, pages 9630–9640. IEEE, 2021.

[DBK$^+$21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)*, pages 886–893. IEEE Computer Society, 2005.

[EM19] Baruch Epstein and Ron Meir. Generalization bounds for unsupervised and semi-supervised learning with autoencoders. *CoRR*, abs/1902.01449, 2019.

[FB03] Xiaoli Zhang Fern and Carla E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *ICML*, pages 186–193. AAAI Press, 2003.

[FJ05] Ana L. N. Fred and Anil K. Jain. Combining multiple clusterings using evidence accumulation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(6):835–850, 2005.

[FPS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The KDD process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, 1996.

[FZW$^+$21] Fabian Falck, Haoting Zhang, Matthew Willetts, George Nicholson, Christopher Yau, and Chris C. Holmes. Multi-facet clustering variational autoencoders. In *NeurIPS*, pages 8676–8690, 2021.

[GAH$^+$21] Keyvan Golalipour, Ebrahim Akbari, Seyed Saeed Hamidi, Malrey Lee, and Rasul Enayatifar. From clustering to clustering ensemble selection: A review. *Engineering Applications of Artificial Intelligence*, page 104388, 2021.

[GDDM14] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587. IEEE Computer Society, 2014.

[GH10] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 297–304. JMLR.org, 2010.

[GHPB14] Sebastian Goebl, Xiao He, Claudia Plant, and Christian Böhm. Finding the optimal subspace for clustering. In *ICDM*, pages 130–139. IEEE Computer Society, 2014.

[GM11] Susan A Gelman and Meredith Meyer. Child categorization. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(1):95–105, 2011.

[GOV22] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS 2022 Datasets and Benchmarks Track*, Advances in Neural Information Processing, New Orleans, United States, November 2022.

[GPM$^+$14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[GPM$^+$20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, 2020.

[GRKB21] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *NeurIPS*, pages 18932–18943, 2021.

[Gro22] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based TF-IDF procedure. *CoRR*, abs/2203.05794, 2022.

[Grü05] Peter Grünwald. Minimum description length tutorial. *Advances in minimum description length: Theory and applications*, 5:1–80, 2005.

[HAK00] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB*, pages 506–515. Morgan Kaufmann, 2000.

[HCX$^+$22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, pages 15979–15988. IEEE, 2022.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition.* Morgan Kaufmann, 2011.

[HM22] Jeff Z. HaoChen and Tengyu Ma. A theoretical study of inductive biases in contrastive learning. *CoRR*, abs/2211.14699, 2022.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[HWL18] Dong Huang, Chang-Dong Wang, and Jian-Huang Lai. Locally weighted ensemble clustering. *IEEE Trans. Cybern.*, 48(5):1460–1473, 2018.

[IT95] Makoto Iwayama and Takenobu Tokunaga. Cluster-based text categorization: A comparison of category search strategies. In *SIGIR*, pages 273–280. ACM Press, 1995.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognit. Lett.*, 31(8):651–666, 2010.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[Joh84] William B Johnson. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[KKK04] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *SDM*, pages 246–256. SIAM, 2004.

[KKL$^+$21] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[KKSZ11] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *WIREs Data Mining Knowl. Discov.*, 1(3):231–240, 2011.

[KKZ09] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, 2009.

[KR90] Leonard Kaufman and Peter J Rousseeuw. Partitioning around medoids (program pam). *Finding groups in data: an introduction to cluster analysis*, 344:68–125, 1990.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LBB$^+$98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LBR$^+$15] André Lourenço, Samuel Rota Bulò, Nicola Rebagliati, Ana L. N. Fred, Mário A. T. Figueiredo, and Marcello Pelillo. Probabilistic consensus clustering using evidence accumulation. *Mach. Learn.*, 98(1-2):331–357, 2015.

[LDJ07] Tao Li, Chris H. Q. Ding, and Michael I. Jordan. Solving consensus and semi-supervised clustering problems using nonnegative matrix factorization. In *ICDM*, pages 577–582. IEEE Computer Society, 2007.

[LHL$^+$21] Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. Contrastive clustering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):8547–8555, 2021.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LMPB22] Collin Leiber, Dominik Mautz, Claudia Plant, and Christian Böhm. Automatic parameter selection for non-redundant clustering. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 226–234. SIAM, 2022.

[LPW18] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In *NeurIPS*, pages 107–117, 2018.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[LSLF16] Hongfu Liu, Ming Shao, Sheng Li, and Yun Fu. Infinite ensemble for image clustering. In *KDD*, pages 1745–1754. ACM, 2016.

[Mac65] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math., Stat., and Prob*, page 281, 1965.

[MBM$^+$21] Lukas Miklautz, Lena G. M. Bauer, Dominik Mautz, Sebastian Tschiatschek, Christian Böhm, and Claudia Plant. Details (don't) matter: Isolating cluster information in deep embedded spaces. In *IJCAI*, pages 2826–2832. ijcai.org, 2021.

[MGFS12] Emmanuel Müller, Stephan Günnemann, Ines Färber, and Thomas Seidl. Discovering multiple clustering solutions: Grouping objects in different views of the data. In *Proceedings of the 28th ICDE, Washington, DC, USA, 1-5 April, 2012*, pages 1207–1210, 2012.

[MGL$^+$18] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[MMA$^+$20] Lukas Miklautz, Dominik Mautz, Muzaffer Can Altinigneli, Christian Böhm, and Claudia Plant. Deep embedded non-redundant clustering. In *AAAI*, pages 5174–5181. AAAI Press, 2020.

[MSL$^+$23] Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch, Christian Böhm, and Claudia Plant. Non-redundant image clustering of early medieval glass beads. *Under review at KDD*, 2023.

[MTW$^+$22a] Lukas Miklautz, Martin Teuffenbach, Pascal Weber, Rona Perjuci, Walid Durani, Christian Böhm, and Claudia Plant. Deep clustering with consensus representations. *CoRR*, abs/2210.07063, 2022.

[MTW$^+$22b] Lukas Miklautz, Martin Teuffenbach, Pascal Weber, Rona Perjuci, Walid Durani, Christian Böhm, and Claudia Plant. Deep clustering with consensus representations. In *ICDM*, pages 1119–1124. IEEE, 2022.

[MYPB17] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Towards an optimal subspace for k-means. In *KDD*, pages 365–373. ACM, 2017.

[MYPB18] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Discovering non-redundant k-means clusterings in optimal subspaces. In *Proceedings of the 24th ACM SIGKDD, 2018*, pages 1973–1982, 2018.

[NDJ10] Donglin Niu, Jennifer G. Dy, and Michael I. Jordan. Multiple non-redundant spectral clustering views. In *Proceedings of the 27th ICML*, pages 831–838, 2010.

[NWL19] Feiping Nie, Cheng-Long Wang, and Xuelong Li. K-multiple-means: A multiple-means clustering method with specified K clusters. In *KDD*, pages 959–967. ACM, 2019.

[OSJ+18] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[PCS+23] Advait Parulekar, Liam Collins, Karthikeyan Shanmugam, Aryan Mokhtari, and Sanjay Shakkottai. Infonce loss provably learns cluster-preserving representations. *CoRR*, abs/2302.07920, 2023.

[Pea01] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[PKBZ15] Mihail Popescu, James Keller, James C. Bezdek, and Alina Zare. Random projections fuzzy c-means (RPFCM) for big data clustering. In *FUZZ-IEEE*, pages 1–6. IEEE, 2015.

[RC20] Everly Ramos and Beatrice P. Concepcion. Visual abstracts: Redesigning the landscape of research dissemination. *Seminars in Nephrology*, 40(3):291–297, 2020. Nephrology and Social Media.

[RDMD21] Jayanth Reddy Regatti, Aniket Anand Deshmukh, Eren Manavoglu, and Ürün Dogan. Consensus clustering with unsupervised representation learning. In *IJCNN*, pages 1–9. IEEE, 2021.

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[RPY+22] Yazhou Ren, Jingyu Pu, Zhimeng Yang, Jie Xu, Guofeng Li, Xiaorong Pu, Philip S. Yu, and Lifang He. Deep clustering: A comprehensive survey. *CoRR*, abs/2210.04142, 2022.

[RR19] Cynthia Rudin and Joanna Radin. Why are we using black box models in ai when we don't need to? a lesson from an explainable ai competition. *Harvard Data Science Review*, 1(2):10–1162, 2019.

[RVM+11] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840. Omnipress, 2011.

[SA22] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Inf. Fusion*, 81:84–90, 2022.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[Scu10] D. Sculley. Web-scale k-means clustering. In *WWW*, pages 1177–1178. ACM, 2010.

[SG02] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — A knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, 2002.

[SPA+19] Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 5628–5637. PMLR, 2019.

[SRP22] Lesia Semenova, Cynthia Rudin, and Ronald Parr. On the existence of simpler machine learning models. In *FAccT*, pages 1827–1858. ACM, 2022.

[TJP05] Alexander P. Topchy, Anil K. Jain, and William F. Punch. Clustering ensembles: Models of consensus and weak partitions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(12):1866–1881, 2005.

[TLL+19] Zhiqiang Tao, Hongfu Liu, Jun Li, Zhaowen Wang, and Yun Fu. Adversarial graph embedding for ensemble clustering. In *IJCAI*, pages 3562–3568. ijcai.org, 2019.

[TSKK19] Pang-Ning Tan, Michael S. Steinbach, Anuj Karpatne, and Vipin Kumar. *Introduction to Data Mining (Second Edition).* Pearson, 2019.

[TWSW19] Tian Tian, Ji Wan, Qi Song, and Zhi Wei. Clustering single-cell rna-seq data with a model-based deep learning approach. *Nat. Mach. Intell.*, 1(4):191–198, 2019.

[VEB10] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 10 2010.

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 1096–1103. ACM, 2008.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[XGF16] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 478–487. JMLR.org, 2016.

[XSL23] Yiqun Xie, Shashi Shekhar, and Yan Li. Statistically-robust clustering techniques for mapping spatial hotspots: A survey. *ACM Comput. Surv.*, 55(2):36:1–36:38, 2023.

[YFB21] Lin Yang, Wentao Fan, and Nizar Bouguila. Deep clustering analysis via dual variational autoencoder with spherical latent embeddings. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[YFSH17] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*, volume 70, pages 3861–3870. PMLR, 2017.

[YMHP16] Wei Ye, Samuel Maurus, Nina Hubig, and Claudia Plant. Generalized independent subspace clustering. In *Proceedings of the 16th ICDM, 2016, Spain*, pages 569–578, 2016.

[ZWW$^+$22] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan L. Yuille, and Tao Kong. Image BERT pre-training with online tokenizer. In *ICLR*. OpenReview.net, 2022.

[ZXZ$^+$22] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao Li, Jiajun Bu, Jia Wu, Xin Wang, Wenwu Zhu, and Martin Ester. A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions. *CoRR*, abs/2206.07579, 2022.

A. Appended Papers

The remainder of this dissertation contains the contributed publications with their corresponding supplementary pages. Further, we provide essential information about the publication, such as the title, abstract, the conference it was published, all authors, the division of work among authors, and the assigned Digital Object Identifier (DOI).

A.1. Deep Embedded Non-Redundant Clustering

Title	Deep Embedded Non-Redundant Clustering
Authors	Lukas Miklautz, Dominik Mautz, Muzaffer Can Altinigneli, Christian Böhm and Claudia Plant
Publication Outlet	Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI). Note: CORE ranking: A*; Acceptance rate: 20.6%
DOI-URL	`https://doi.org/10.1609/aaai.v34i04.5961`

Division of Work This contribution was developed with intensive cooperation between the first two authors with daily meetings, pair programming sessions and fine grained distribution of work, resulting in equal contribution of the first two authors to this paper.
Lukas Miklautz: problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of variations of the method; experimental design; execution of experiments; writing and visualization of major parts of the article; review of relevant related work; creating the conference poster and presentation.
Dominik Mautz: conceiving the initial idea; problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of variations of the method; experimental design; execution of experiments; writing and visualization of major parts of the article; review of relevant related work.
Muzaffer Can Altinigneli: development and implementation of the autoencoders part; regular discussions of candidate methods and experiments; writing and visualizing parts of the experiments sections in the main paper and supplementary.
Christian Böhm: regular discussions of candidate methods and potential experiments; suggestions for enhancements and improvements; periodic review of drafts for the paper; mentoring and supervision.
Claudia Plant: regular discussions of candidate methods and potential experiments; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Abstract	Complex data types like images can be clustered in multiple valid ways. Non-redundant clustering aims at extracting those meaningful groupings by discouraging redundancy between clusterings. Unfortunately, clustering images in pixel space directly has been shown to work unsatisfactory. This has increased interest in combining the high representational power of deep learning with clustering, termed deep clustering. Algorithms of this type combine the non-linear embedding of an autoencoder with a clustering objective and optimize both simultaneously. None of these algorithms try to find multiple non-redundant clusterings. In this paper, we propose the novel Embedded Non-Redundant Clustering algorithm (ENRC). It is the first algorithm that combines neural-network-based representation learning with non-redundant clustering. ENRC can find multiple highly non-redundant clusterings of different dimensionalities within a data set. This is achieved by softly) assigning each dimension of the embedded space to the different clusterings. For instance, in image data sets it can group the objects by color, material and shape, without the need for explicit feature engineering. We show the viability of ENRC in extensive experiments and empirically demonstrate the advantage of combining non-linear representation learning with non-redundant clustering.
Thesis-Reference	[MMA^{+}20]

The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

Deep Embedded Non-Redundant Clustering

Lukas Miklautz,[*,1] **Dominik Mautz,**[*,2] **Muzaffer Can Altinigneli,**[1,2] **Christian Böhm,**[2,3]
Claudia Plant[1,4]
[1]Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Ludwig-Maximilians-Universität München, Munich, Germany
[3]MCML, [4]ds:UniVie
[1]{lukas.miklautz, claudia.plant}@univie.ac.at
[2]{altinigneli, boehm, mautz}@dbs.ifi.lmu.de

Abstract

Complex data types like images can be clustered in multiple valid ways. Non-redundant clustering aims at extracting those meaningful groupings by discouraging redundancy between clusterings. Unfortunately, clustering images in pixel space directly has been shown to work unsatisfactory. This has increased interest in combining the high representational power of deep learning with clustering, termed deep clustering. Algorithms of this type combine the non-linear embedding of an autoencoder with a clustering objective and optimize both simultaneously. None of these algorithms try to find multiple non-redundant clusterings. In this paper, we propose the novel Embedded Non-Redundant Clustering algorithm (ENRC). It is the first algorithm that combines neural-network-based representation learning with non-redundant clustering. ENRC can find multiple highly non-redundant clusterings of different dimensionalities within a data set. This is achieved by (softly) assigning each dimension of the embedded space to the different clusterings. For instance, in image data sets it can group the objects by color, material and shape, without the need for explicit feature engineering. We show the viability of ENRC in extensive experiments and empirically demonstrate the advantage of combining non-linear representation learning with non-redundant clustering.

1 Introduction

Every day massive amounts of complex data like images, texts, videos and audios are generated and most of them have no labels. This makes it nearly impossible to apply supervised methods, because it may be too expensive to label the data or there might not even be a labeling consensus. This calls for unsupervised classification techniques like clustering, which are unsupervised learning algorithms for partitioning data into similar groups. Unfortunately, these kinds of complex domains have been notoriously difficult to handle for classical clustering algorithms (Xie, Girshick, and Farhadi 2016). The deep learning 'revolution' of the last few years targets these data sets explicitly, which suggests that a combination of both approaches is promising. Several techniques have been proposed in this area under the terms 'deep clustering' or 'embedded clustering'. They combine a flat clustering objective—considering only one valid clustering— with unsupervised representation learning through neural networks (Bengio, Courville, and Vincent 2013; Aljalbout et al. 2018). These approaches make it possible to achieve a high-quality clustering even in the above described domains. The advantage of an integrated approach that performs clustering and representation learning simultaneously is that they benefit from each other. The clustering provides feedback to the transformation, and in return the non-linear transformation can alter the embedded space to improve the clustering. All these methods assume that the data can be partitioned into only a single valid clustering. Yet, we argue that most modern complex data sets can be partitioned in multiple valid ways. Techniques from subspace clustering address these issues, but they deliver a multitude of redundant clustering solutions, which might be valid, but are hard to interpret even for domain experts. Deep subspace clustering algorithms like (Ji et al. 2017; Zhang et al. 2018) can only find a single valid clustering, where a cluster can belong to a different subspace. We think it is necessary to constrain the solution space by only consid-

[*]First authors with equal contribution.

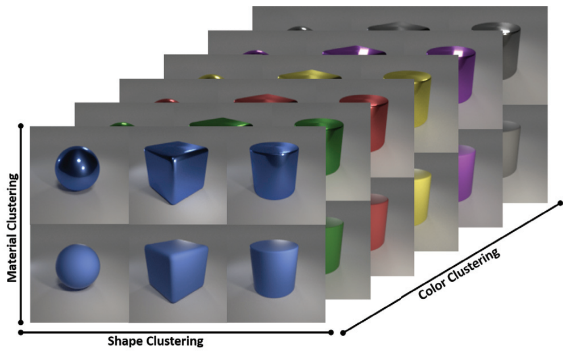


Figure 1: This data set can be meaningfully grouped either by shapes (cube, cylinder, sphere), by colors (red, blue, green, yellow, purple, gray), or by materials (rubber, metal). All of these clusterings are non-redundant.

ering clusterings, which are highly non-redundant, meaning that the found clusterings should be as different as possible. Techniques tackling these challenges are not new. Indeed, outside of deep clustering there is a whole range of algorithms for non-redundant clustering (Niu, Dy, and Jordan 2010; Ye et al. 2016; Mautz et al. 2018; Cui, Fern, and Dy 2007).

In Figure 1 we can see examples of rendered objects from a data set, which exhibit different aspects of non-redundancy, such as color, shape or material[1]. A non-redundant clustering objective targets the extraction of these multiple groupings. Applying a deep embedded flat clustering algorithm on the example in Figure 1 would result in only a single clustering that would be even—in a best case scenario—a mixture of shape, color and material. To address the challenge of finding non-redundant clusterings in complex high dimensional data, we propose our novel Embedded Non-Redundant Clustering algorithm (ENRC) that combines the benefits of a non-linear feature transformation with a non-redundant clustering objective. To the best of our knowledge ENRC is the first algorithm that combines these two aspects. The main contributions of the paper can be summarized as follows:

- **Non-redundant clustering layer:** We propose a novel clustering layer, that axis-aligns the different non-redundant structures, captured in the embedded space. Each clustering can have different dimensionalities that are automatically detected.
- **Feature importance:** Existing deep clustering methods have to resort to additional dimensionality reduction techniques like t-sne (Maaten and Hinton 2008) to visualize their results, which might not show a low dimensional representation faithful to the found clustering. In contrast for ENRC each axis-aligned embedded feature has a soft-assignment weight, where a high feature weight indicates that this feature is important for the clustering. This leads to a cluster aware dimensionality reduction for visualization.
- **Joint feature optimization:** Existing non-redundant clustering methods rely on hand engineered features. These features might already predetermine what structures one expects to find, e.g. engineering color features for detecting clusters based on color. We show that the joint optimization of ENRC and an autoencoder is able to to learn the relevant features directly from the data.
- **Preservation of non-redundancy:** To our knowledge we are the first to show empirically that non-redundant structure is preserved in autoencoders and can be recovered by non-redundant clustering algorithms.

2 Embedded Non-Redundant Clustering

2.1 Overview

An autoencoder is an unsupervised (self-supervised) neural network which learns to reconstruct its input. It consists of an encoder $\mathrm{enc}(\cdot)$ network, which embeds the input data in some latent space and a decoder $\mathrm{dec}(\cdot)$ network, which tries to reconstruct the original input from the embedding. To avoid a trivial identity mapping, i.e. simple copying, the latent space dimensionality is often chosen smaller than the input's dimensionality or some other regularization might be used. To explain our ENRC algorithm, we assume some generic pretrained encoder and decoder, which have the non-redundant clustering layer in the middle, as depicted in Figure 2. The novel algorithm finds multiple valid non-redundant clusterings in the embedded space of the autoencoder. To achieve this we introduce two new learnable parameters. A linear transformation matrix V that aligns the structures within each clustering with the axis, acting as an approximation of a rotation. The linearity is sufficient for this task, as the non-linear relationships are already learned by the autoencoder. Additionally, we use feature weights β_s that weigh the importance of each feature for each of the S clusterings acting as a soft separation mechanism of the space. Due to this the standard autoencoder reconstruction loss is adapted to:

$$\mathcal{L}_{\text{rec}} = ||\mathbf{x} - \mathrm{dec}(VV^{\mathsf{T}}\,\mathrm{enc}(\mathbf{x}))||_2^2, \qquad (1)$$

where the reconstruction acts as a "non-degeneracy control" (Le et al. 2011) to prevent the matrix V from degeneration.

In the following, we denote with $x[i]$ the i'th entry of vector x and $X[i,j]$ is the value of the i'th row and j'th column of matrix X. Further, we use the weighted squared euclidean distance, defined as

$$||a-b||_\tau^2 := \sum_{i=1}^{D} \tau[i](a[i]-b[i])^2, \qquad (2)$$

where $a, b \in \mathbb{R}^D$ are arbitrary vectors—for which we want to measure the distance—and $\tau \in \mathbb{R}^D$ is a weight vector containing weights for each dimension. An overview of other used notations can be found in the Supplementary at https://gitlab.cs.univie.ac.at/lukas/enrcpublic.

2.2 Feature Weighting

In classical non-redundant subspace clustering each clustering gets its own subspace orthogonal to the other subspaces. We relax this discrete assignment to a continuous weight $\beta_s[d]$ such that each dimension d belongs partially to a clustering s. This relaxation makes the loss function differentiable w.r.t. β. We require that these weights are positive

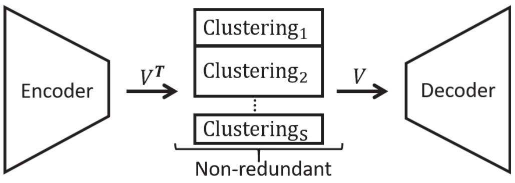


Figure 2: The architecture of ENRC. The linear layer V^{T} aligns the essential structures of the different clusterings along the axes. The clusterings can be of different dimensionality. V maps the clusterings back to the embedded space.

[1]All figures are best viewed in color.

and for a single dimension the fractions over all clusterings should sum to one:

$$\forall d \in \{1,2,..,D\} : \sum_{s=1}^{S} \beta_s[d] = 1, \tag{3}$$

where $\forall d \in \{1,2,..,D\}, \forall s \in S : \beta_s[d] \geq 0$. We can implement these constraints using a soft-max function on a trainable $S \times D$ parameter matrix B, s.t.:

$$\beta_s[d] := \frac{\exp(B[s,d])}{\sum_{i=1}^{S} \exp(B[i,d])}.$$

We can then optimize B without any further constraints. To express the feature importance during clustering, we utilize the βs for the weighted squared euclidean distance.

2.3 Compression Loss

The compression loss 'moves' embedded points $z := \mathrm{enc}(\mathbf{x})$ closer to their centers $\mu_{s,k}$, enhancing the separation between clusters. As we consider multiple non-redundant clusterings, we only want to move points, which are close to their respective centers in each weighted feature space. This is achieved by the following loss function:

$$\mathcal{L}_{\text{comp}} = \frac{1}{S} \sum_{s=1}^{S} \sum_{k=1}^{K_s} \frac{1}{|C_{s,k}|} \sum_{z \in C_{s,k}} ||V^{\mathsf{T}} z - \mu_{s,k}||^2_{\beta_s}. \tag{4}$$

Note that $\mu_{s,k}$ is the k'th cluster center of clustering s in the already V^{T}-transformed space. Here $\mu_{s,k}$ is considered a constant (details on its update below) and only V^{T}, β and z are updated.

2.4 Updating Data Assignment

For given cluster centers, we assign for each clustering s each embedded data point z to the closest center for the β_s weighted euclidean distance in the transformed embedded space:

$$\forall s \in S : \arg\min_{k \in [1;K_s]} ||V^{\mathsf{T}} z - \mu_{s,k}||^2_{\beta_s} \tag{5}$$

2.5 Updating Cluster Centers

Similar to (Yang et al. 2017) we use an adapted version of mini-batch k-means (Sculley 2010) for updating the cluster centers and assignments. For each incoming data batch, we iterate over the weighted feature spaces and update the clusterings in each space separately. First we update the cluster assignments with the previous cluster centers $\boldsymbol{\mu}^{t-1}_{s,k}$ with Eq. 5. From this new assignments we calculate a per center update for the current mini-batch in the following way:

$$\boldsymbol{\mu}^t_{s,k} = \boldsymbol{\mu}^{t-1}_{s,k}(1 - \eta_{s,k}) + \hat{\boldsymbol{\mu}}^t_{s,k}\eta_{s,k}. \tag{6}$$

where $\hat{\boldsymbol{\mu}}^t_{s,k}$ is the mean of the assigned points from the current mini-batch and $\eta_{s,k}$ is a per cluster learning rate. It is defined as one over the exponential weighted average of past assignments to the center, $\eta_{s,k} = {}^1/_{A^t_{s,k}}$. The denominator is given by:

$$A^t_{s,k} = (1 - \alpha)A^{t-1}_{s,k} + \alpha \hat{A}^t_{s,k} \tag{7}$$

with $0 < \alpha < 1$ as a discounting parameter and $\hat{A}^t_{s,k}$ as the number of assigned points from the current mini-batch. If a mini-batch does not contain any points for a cluster, the center will not be updated.

2.6 Initialization

At the beginning, we aim to find the initial clustering structures captured in the embedding. Therefore, we only optimize the linear transformation V, the cluster centers $\boldsymbol{\mu}$, and the βs (respectively B), but keep the autoencoder parameters fixed (i.e. $\mathrm{enc}(\cdot)$ and $\mathrm{dec}(\cdot)$). First, we initialize V as a random orthogonal matrix. Next, we soft-assign the first $\lfloor {}^D/_S \rfloor$ dimensions of the V-rotated space 'strongly' to the first clustering by setting the beta weights of these dimensions and the first clustering to 0.9: $\forall d \in [1 : \lfloor {}^D/_S \rfloor] : \beta_1[d] := 0.9$. The remaining weight of 0.1 for each of these dimensions is evenly distributed among the other clusterings—reflecting only a 'weak' soft-assignment. Then we assign the next $\lfloor {}^D/_S \rfloor$ dimensions strongly to the second clustering in the same manner—and so forth. Next, we initialize the cluster centers $\boldsymbol{\mu}$ of each clustering with the initialization procedure of k-means++ (Arthur and Vassilvitskii 2007) using the distance metric in Eq. 2. Finally, we optimize w.r.t. $\mathcal{L}_{\text{comp}}$ and update the cluster centers as described above. To ensure that V does not degenerate, but instead aligns the structural information of each clustering with the axis, we force V to be approximately orthogonal by adding to $\mathcal{L}_{\text{comp}}$ an adapted version of the reconstruction loss:

$$\mathcal{L}_{\text{orth}} = ||\,\mathrm{sg}[z] - VV^{\mathsf{T}}\,\mathrm{sg}[z]||^2_2, \tag{8}$$

where $\mathrm{sg}[\cdot]$ is the gradient stop iterator such that z is regarded as a constant in this loss term.[2] This differentiable loss avoids degeneration of V by creating an incentive for it to be approximately orthogonal (allowing only rotation and reflection). We only keep this loss term during the initialization phase and drop it in the refinement phase. The latter decision was made based on the results of (Le et al. 2011), which indicate that the reconstruction loss (Eq. (1)) is sufficient for non-degeneracy control. For smaller data sets, one could use alternatively the methods proposed in (Mautz et al. 2018; Cui, Fern, and Dy 2007). This selection of the initial μ's, β's and V is based on the minimal loss and can be performed several times.

2.7 Clustering Phase

As noted in previous work (Yang et al. 2017) the joint optimization of clusterings and embeddings can lead to improved results. The loss function of ENRC combines the reconstruction and compression loss

$$\mathcal{L} = \mathcal{L}_{\text{comp}} + \lambda \mathcal{L}_{\text{rec}} \tag{9}$$

This is motivated by the results of (Guo et al. 2017) that without the space preserving effect of the reconstruction loss, the found clusterings could become meaningless, because $\mathcal{L}_{\text{comp}}$ would be trivially fulfilled if it maps all points into one cluster. The hyperparameter $\lambda > 0$ defines this

[2]Note, that this is equivalent to $||VV^T - I||^2_F$, if z is whitened, see e.g. Lemma 3.1 in (Le et al. 2011).

trade-off, which we set for all experiments to one. During training ENRC updates all parameters of the clustering layer $\beta, V, \boldsymbol{\mu}_{s,i}$ and all weight parameters of encoder and decoder jointly together.

2.8 Cluster Center Reinitialization

Most other recently proposed deep clustering methods do not provide a mechanism for the case when a cluster does not get assigned any points after several mini-batch updates. This can lead to a degradation of performance, as the remaining clusters are merged. This scenario is comparable to the case where a k-means center does not get assigned any data points during the update step. A common strategy—e.g. in the scikit-learn package—is to reinitialize the lost center with a point that is badly represented by its assigned center in terms of euclidean distance. We adapt this strategy by first sampling n_s points from the whole data set and embed them in the rotated feature space. We select the cluster with the highest compression loss and choose the point which is farthest away from this cluster's centroid. After selecting the new centroid, we conduct l k-means update steps in the affected feature space. To avoid unnecessary reinitializations due to unbalanced clusters or small batch sizes we count how often a centroid lost all its points and compare it to a threshold value v, which increases during training by $\lfloor\sqrt{\text{iter}_t}\rfloor$, where iter_t is the current mini-batch iteration count. The latter is a simple heuristic, which accounts for the observation that in the beginning of training, cluster assignments may change more rapidly and become more stable towards the end. After reinitialization, the count for the reinitialized center is set to zero again. The parameters l and n_s should be chosen to fit the computational constraints.

3 Experiments

We evaluated ENRC with four different data sets. As we are the first to address the topic of non-redundant clustering with neural networks, we needed for our benchmark high-dimensional data sets, which are labeled and have enough data points. Note that in real world settings one would use ENRC for exploratory data analysis, without ground truth labels, as we show in Section 3.5. To quantify the ability of finding non-redundant structure in high dimensional data sets, we adapted three commonly used deep learning data sets. Additionally, we use the stickfigures data set, which is often used in the non-redundant clustering literature. Other commonly used data sets from the non-redundant clustering literature contain often less than 500 data points, here stickfigures is already one of the largest, see e.g. (Ye et al. 2016). A summary of the considered data sets is shown in Table 2. We implemented ENRC in Python and trained our networks on a single NVIDIA RTX 2080 Ti. We ran the comparison method on a machine with four Intel(R) Xeon(R) CPU E5-2650 Cores and 32 GB RAM. An implementation of ENRC and all experiments is available at https://gitlab.cs.univie.ac.at/lukas/enrcpublic.

3.1 Data Sets

Concatenated-MNIST We extend the well known MNIST (LeCun et al. 1998) data set by concatenating two digits side by side, resulting in 100 possible combinations, named C-MNIST. With this extension, we show the ability of ENRC to capture positional non-redundancy. This data set can be seen as containing two-digit numbers, from 00 to 99, where each digit (left and right) is independent from the other. To make it easier to use existing convolutional architectures, we added black row wise pixels, so the new format is 56×56 instead of 28×56.

NR-Objects We used a publicly available rendering software[3], which was used in (Johnson et al. 2017), to generate objects with three non-redundant clusterings, called (Non-Redundant) NR-Objects. Each object can be clustered by three shapes, two materials and six colors. Otherwise the default settings, which contained lighting jitter, were kept.

GTSRB The German Traffic Sign Benchmark (GTSRB) data set (Houben et al. 2013) contains real world images of traffic signs, it has been used in object detection literature for self-driving vehicles. We use a subset of 4 different traffic signs 'Speed limit (70km/h)', 'No passing', 'Ahead only', 'Keep right', which can be clustered w.r.t. to the four types of traffic sign and their two colors.

Stickfigures The stickfigures data set contains nine basic objects of dancing figures. It contains three clusterings for the upper and three for the lower body pose, a more detailed explanation can be found in (Ye et al. 2016).

We preprocessed all data sets by a channel-wise z-transformation, to get a zero mean and variance of one. For the GTSRB data set we used additionally histogram equalization, to improve the low contrast in some of the real world images. Examples of each data set can be found in the Supplementary.

3.2 Experimental Setup

For our experiments we use a convolutional autoencoder that utilizes several well-established architectural patterns. By exploiting skip (or identity) connections within convolutional *resnet* blocks as described in (He et al. 2015), we are able to train deeper networks which adapt their depth based on the data set. We applied some additional techniques from (He et al. 2018) to improve our autoencoder performance, e.g. we set the first convolutional layer's kernel size from 7×7 down to 3×3 and compensate the effective receptive field size with additional 3×3 convolutions. Additionally, we start with stride-one convolution instead of stride-two to avoid discarding $\frac{3}{4}$ of the input-image's pixels just within the first layers. We set the architecture parameters, such as the number of feature filters in a convolutional layer, based on the dimensionality and color channels of the data set and set the dropout rate based on the achieved reconstruction error. For all data sets we used the same embedding size of 16, which resulted in good reconstructions. We used this single architecture for all data sets.

For each data set we pretrain ten autoencoders and use them for ENRC and all baseline methods. With this setting we make sure that all methods have the same starting conditions. Similar to (Xie, Girshick, and Farhadi 2016) we pretrain the autoencoder first for 10,000 mini-batch iterations

[3]https://github.com/facebookresearch/clevr-dataset-gen

Table 1: The NMI averages and standard deviations for the ten pretrained autoencoders. Results marked with * had to be run on a subset of 10,000 data points due to memory constraints (>32 GB). Best value in bold.

Data Sets	**Clustering**	**ENRC**	**Orth1**	**Orth2**	**mSC**	**Nr-Kmeans**	**ISAAC**
NR-Objects	color	**1.00 ±0.00**	0.70 ±0.09	0.73 ±0.06	0.35 ±0.05	0.92 ±0.09	0.15 ±0.06
	material	**1.00 ±0.00**	0.46 ±0.16	0.11 ±0.12	0.03 ±0.07	0.95 ±0.14	0.53 ±0.08
	shape	**1.00 ±0.00**	0.39 ±0.20	0.20 ±0.08	0.03 ±0.03	0.92 ±0.16	0.60 ±0.07
GTSRB	type	**0.74 ±0.01**	0.57 ±0.07	0.73 ±0.15	0.04 ±0.04	0.72 ±0.01	0.60 ±0.07
	color	**0.67 ±0.00**	0.59 ±0.02	0.63 ±0.03	0.04 ±0.06	0.65 ±0.01	0.59 ±0.04
Stickfigures	upper	**1.00 ±0.00**	0.79 ±0.21	0.00 ±0.00	0.33 ±0.20	**1.00 ±0.00**	0.37 ±0.05
	lower	**1.00 ±0.00**	0.77 ±0.24	0.00 ±0.00	0.30 ±0.17	**1.00 ±0.00**	0.39 ±0.08
C-MNIST	left	**0.83 ±0.04**	0.33 ±0.02	0.35 ±0.03	0.07 ±0.02*	0.69 ±0.03	0.29 ±0.13*
	right	**0.82 ±0.01**	0.40 ±0.03	0.41 ±0.04	0.06 ±0.02*	0.70 ±0.03	0.19 ±0.13*

Table 2: Summary of used data sets. The last column shows the number of ground truth clusterings and the corresponding number of clusters.

Name	**# Points**	**# Dimensions**	**# Clusters**
C-MNIST	60,000	3,136	10; 10
NR-Objects	10,000	16,384	6; 3; 2
GTSRB	6,720	1,024	4; 2
Stickfigures	900	400	3; 3

Table 3: The VI averages and standard deviations for the ten pretrained autoencoders. Most methods, were able to find non-redundant clusterings. Results marked with * had to be run on a subset of 10,000 data points due to memory constraints (>32 GB).

Method	NR-Objects	GTSRB	Stickfigures	C-MNIST
ENRC	2.39 ±0.00	1.96 ±0.01	2.20 ±0.00	4.56 ±0.01
Orth1	2.31 ±0.03	1.98 ±0.03	2.06 ±0.17	4.41 ±0.06
Orth2	1.20 ±0.07	1.11 ±0.24	0.90 ±0.56	3.13 ±0.20
mSC	2.32 ±0.05	1.99 ±0.09	2.01 ±0.10	4.48 ±0.05*
NR-Kmeans	2.35 ±0.05	1.94 ±0.00	2.19 ±0.00	4.57 ±0.02
ISAAC	1.25 ±1.04	n.a.	1.14 ±0.25	n.a.*

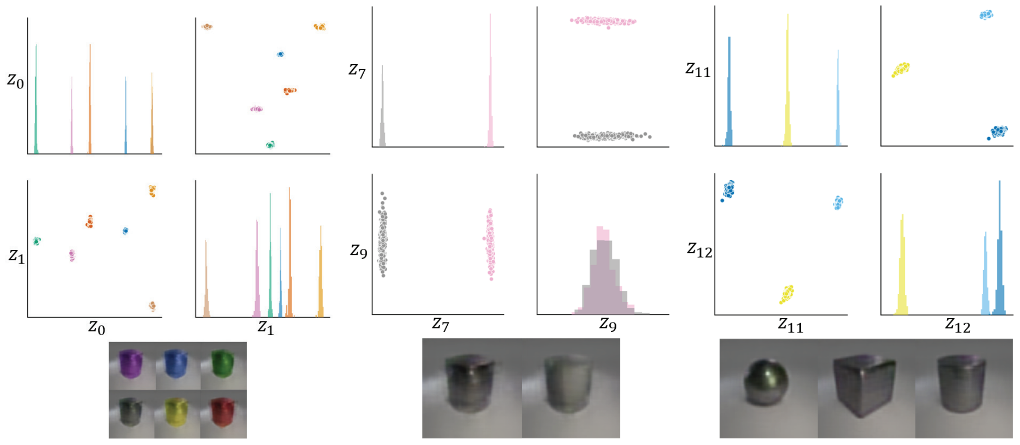
with dropout and then fine tune it for another 5,000 mini-batch iterations with deactivated dropout. For pretraining the autoencoder we use image augmentation (rotation, lighting, zooming), Adam (max-lr $= 0.01, \quad \beta_1 = 0.9, \quad \beta_2 = 0.99$) (Kingma and Ba 2014) with weight decay of 0.001. We boost the training speed of our network with the *super-convergence* learning rate schedule outlined in (Smith and Topin 2017). The intuition behind this idea is that we start with a low learning rate to warm-up and to slowly find the correct direction on the loss landscape. During the second halve of the training cycle, and close to the end of training process, we decrease the learning rate as we come closer to the optimum. Reaching the maximum learning rate max-lr separates the two training halves and a cosine annealing schedule provides a smooth transition between the two halves of the training cycle. For fine tuning we halve the max-lr to ensure smooth convergence and train for the remaining mini-batch iterations. Further information about the experimental setup and architectural details are presented in the Supplementary and our Python implementation. After pretraining we initialize μ, β and rotation matrix V. For the joint clustering optimization we train for another 20,000 mini-batch iterations, except for the simple stickfigures data set which converged already at 2,000 mini-batch iterations. Similar to (Xie, Girshick, and Farhadi 2016) we decrease the learning rate again and keep decreasing it every 2,000 iterations to ensure a smooth convergence for the joint optimization. We set the initial learning rate for β to max-lr $= 0.01$. This is motivated by the thought that the feature space soft-assignments should be updated faster; before the embedding is linearly transformed by V and clusters are compressed by Eq. 4. The discounting parameter α in Eq. 7, was set to 0.5 giving equal weight to new and old centers. The cluster reinitialization parameters were set to $l = 10$ and $n_s = 1,000$, which ensured a fast re-clustering procedure in case a center got lost. This setting worked well for all considered data sets and shifts most hyperparameter setting decisions of the neural network to the pretraining phase, which is self-supervised (reconstructing the input) for an autoencoder.

3.3 Evaluation Metrics

To evaluate the quality of the found non-redundant clusterings we use the normalized mutual information (NMI) (Vinh, Epps, and Bailey 2010) for each best matching clustering in the found feature spaces, where 1 is a perfect clustering and 0 indicates that no structure was captured. Additionally, we use the average variation of information (VI) (Meilă 2007) for measuring the redundancy of the found feature spaces. The VI calculates the similarity between two different clusterings, where higher values are better. Note that the VI cannot be computed for a single clustering and can only be used to compare different clusterings on the same data set, as the magnitude is data dependent.

3.4 General Results

We compare our ENRC with several state of the art non-redundant clustering algorithms. For each data set we pretrained ten autoencoders and use them for all methods. The



(a) Color (b) Material (c) Shape

Figure 3: The scatter plots show the dimensions z_s with the two highest feature weights β_s for each of the three clusterings. The images below are the reconstructed centers $\mathrm{dec}(V\boldsymbol{\mu}_{s,i})$ of each cluster. Best viewed in color.

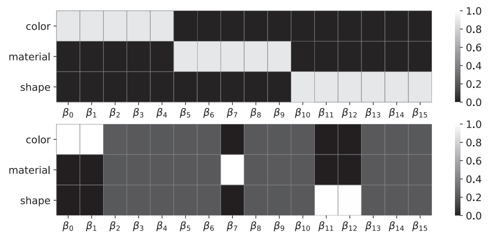


Figure 4: Sorted feature weights β_s for each clustering. **Upper** plot shows initial β_s. **Bottom** plot same feature weights after clustering.

considered comparison methods are Orth1 and 2 (Cui, Fern, and Dy 2007), Nr-Kmeans (Mautz et al. 2018), ISAAC (Ye et al. 2016) and mSC (Niu, Dy, and Jordan 2010). We use these algorithms, because in contrast to subspace clustering algorithms, they have an additional constraint that tries to reduce the redundancy between clusterings. A more detailed discussion can be found in Section 4. Additionally, all of these algorithms are able to find several clusterings in arbitrarily oriented subspaces. This is important, because we assume that the non-redundant clustering structure is preserved in the autoencoder, but the cluster structure does not have to be axis-aligned. For all considered data sets we know the exact number of clusterings and set the methods parameters accordingly. We ran Nr-Kmeans, Orth1 and 2 ten times for each of the ten autoencoders and averaged the best run in terms of their respective loss function. ISAAC and mSC were run once for each autoencoder, due to run time constraints (more than one day). Our experimental results in Table (3) show that, except for ISAAC, each method was able to achieve a quite similar VI. This indicates, that they were indeed able to find several non-redundant clusterings from the learned embeddings of the pretrained autoencoders. Algorithms that found only a single clustering are marked with 'n.a.'. The clustering performance in accordance to the ground truth labels is shown in Table 1. Again we see that all methods are able to recover the non-redundant clusterings from the embedded space of the autoencoder, but only ENRC is able to jointly transform the space, which results in even higher NMI values. For the quite simple stickfigures data set, Nr-Kmeans and ENRC are both able to find the two clusterings perfectly.

3.5 Case Study

In this section we highlight some of ENRC's key strengths by discussing the NR-Objects data set in more detail. In Figure 4 the feature weights β_s before and after training are shown, where higher values indicate stronger memberships. We can see that after training of the 16 dimensional feature space only two are needed for clustering the six colors, another two dimensions for the three shapes and only one for the two materials. The feature weights of the other eleven dimensions are evenly distributed between the three non-redundant clusterings, which indicates that these dimensions are not important for the specific clustering, but encode some shared information. The separation of features, into important and unimportant for the clusterings, allows ENRC to visualize the found embeddings without the need of an additional dimensionality reduction technique like t-sne (Maaten and Hinton 2008). In Figure 3 we can see scatter plots of the two highest β-weighted dimensions for each clustering, indicated by the matching subscripts of the β_s from the bottom plot in Figure 4. On the diagonal axis of the scatter plot are the histograms of each feature. As the material clustering has only one high feature weight β_7 we see that only z_7 contains clustering structure. Since all other feature weights for the material feature space are indifferent between dimensions, we can see here for the second dimension z_9 a unimodal distribution without clustering structure. For color and shape the feature weights β_s indicate that they need two features for their clustering, which can be seen in the multiple modes of the histograms. Another key benefit of using an autoencoder is that we can use the expressive power of a neural network to find non-redundant clusterings in the feature space, but still keep interpretability by utilizing the decoder. Below each scatter plot in Figure 3 we can see the decoded centers for each clustering as reconstructed images in the original pixel space. Each center of the color feature space is exhibiting only the color feature, but is an average of all other clusterings like shapes and materials, resulting in splodges of color. The same can be seen for material, but here the center for metallic objects is reflective and the one for rubber is not. The two centers appear to be grey because it is an average of all six colors. The latter is true for the three shape clusters as well. While the shape structure is crisp and detailed, the surfaces appear to be greyish and slightly reflective, because the information of the other—non-redundant—clusterings is averaged.

4 Discussion and Related Work

A centroid-based approach like we used in ENRC is quite common in embedded clustering. To the best of our knowledge, none of the proposed embedded clustering methods aim to find multiple non-redundant clusterings within a data set. Algorithms such as DEC (Xie, Girshick, and Farhadi 2016), IDEC (Guo et al. 2017), DMC (Chen, Lv, and Yi 2017) or DEPICT (Ghasedi Dizaji et al. 2017) utilize a pre-trained autoencoder to embed the data and a Gaussian or Student-t kernel to softly assign the data points to clusters. DMC directly penalizes the distance of data points embedded to each center weighted by the assignment. The other methods utilize a KL divergence loss between the soft-assignments and an auxiliary probability density function to harden the clusters. All five methods utilize a mini-batch gradient descent optimization scheme as ENRC does. DCN (Yang et al. 2017) is a k-means based method. In contrast to the above described methods, it performs a hard assignment like ENRC, however, it alternates between updating the embedding, the cluster assignments and the cluster centers iterating over the full data set in each step. Recent work in deep subspace clustering, e.g. (Ji et al. 2017; Zhang et al. 2018) find a single clustering, where each cluster can belong to a different subspace. With the work of (Zhang et al. 2018; Fard, Thonet, and Gaussier 2018) we share the idea that hard assignments in different situations can be relaxed with a softmax function. In contrast to these ENRC finds common non-redundant feature spaces for related clusterings. Other recently proposed methods like GMVAE (Dilokthanakul et al. 2016) and VaDE (Jiang et al. 2017) utilize variational approaches or generative adversarial networks, like ClusterGAN (Mukherjee et al. 2019). From all above described method, these last three approaches are the least similar to ENRC. Further algorithms are discussed in two recent survey papers (Aljalbout et al. 2018; Min et al. 2018) that provide a broader overview over proposed embedded clustering techniques. Multiple and non-redundant clustering methods are an active research field and in the classical clustering literature several different methods have been proposed. An overview and different variations can be found in (Müller et al. 2012). Methods like (Chang et al. 2017) assume that the subspaces or views are axis-parallel, however, we assume that the subspaces in the embedded space are arbitrarily-oriented. In the following, we only discuss algorithms that can find non-redundant clusterings in arbitrarily-oriented subspaces. Orth1 and Orth2 (Cui, Fern, and Dy 2007) are two clustering algorithms that extract multiple k-means clustering structures sequentially from spaces orthogonal to the space spanned by the previous cluster centers. The difference between the two versions is that the orthogonal projection can be w.r.t. all clusters or just a single cluster to which a data point is assigned. The main idea of Nr-Kmeans (Mautz et al. 2018) is that the data space can be split into several arbitrarily-oriented orthogonal subspaces and within each subspace the data follows a k-means like clustering. It also allows for an optional noise space without any clustering structure. Like in ENRC the method optimizes all subspaces and the clusterings within simultaneously. Further, it is shown that a parallel clustering extraction can be advantageous compared to a sequential approach used in Orth. ISAAC (Ye et al. 2016) utilizes a two step procedure. First, it uses Independent Subspace Analysis (ISA) to determine the subspaces. Then, it fits a Gaussian mixture model with hard assignments within each subspace. Thereby, all parameters are estimated based on the MDL principle. All four of the above described non-redundant clustering methods have in common that, for each clustering, they find a linear transformation for the respective subspace. In contrast, ENRC utilizes the autoencoder to perform a nonlinear transformation and jointly optimizes the clustering. mSC (Niu, Dy, and Jordan 2010) utilizes the Hilbert-Schmidt independence criterion to find multiple non-redundant views for the relaxed spectral clustering objective. Similar to the autoencoder based embedding, the spectral embedding of mSC can be seen as non-linear, however, its approach is quite different from the one of ENRC.

5 Conclusion

In this paper we proposed the Embedded Non-Redundant Clustering algorithm ENRC, to the best of our knowledge it is the first algorithm that combines an embedded and a non-redundant clustering objective. Its unique characteristics are the joint optimization of multiple non-redundant clusterings together with the non-linear embedding, the intrinsic cluster aware dimensionality reduction and automated feature extraction. Our experiments show that ENRC and its joint training procedure has an advantage over a two step process where the non-linear embedding and non-redundant clustering are separately optimized. This is in accordance with the results shown for flat embedded clustering algorithms. We highlighted some of the benefits of our algorithm ENRC in a case study showing its interpretable results. In future work we want to explore the possibility to incorporate different k-means extensions, such as estimating the number of cluster centers in each clustering. Another interesting direction would be to leverage the concept of non-redundancy for semi-supervised learning.

References

Aljalbout, E.; Golkov, V.; Siddiqui, Y.; and Cremers, D. 2018. Clustering with deep learning: Taxonomy and new methods. *CoRR* abs/1801.07648.

Arthur, D., and Vassilvitskii, S. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM SODA*, 1027–1035. Society for Industrial and Applied Mathematics.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

Chang, Y.; Chen, J.; Cho, M. H.; Castaldi, P. J.; Silverman, E. K.; and Dy, J. G. 2017. Multiple clustering views from multiple uncertain experts. In *Proceedings of the 34th ICML 2017*, 674–683.

Chen, D.; Lv, J.; and Yi, Z. 2017. Unsupervised multi-manifold clustering by learning deep representation. In *Workshops at the 31th AAAI 2017*, 385–391.

Cui, Y.; Fern, X. Z.; and Dy, J. G. 2007. Non-redundant multi-view clustering via orthogonalization. In *Proceedings of the 7th IEEE ICDM, 2007*, 133–142.

Dilokthanakul, N.; Mediano, P. A. M.; Garnelo, M.; Lee, M. C. H.; Salimbeni, H.; Arulkumaran, K.; and Shanahan, M. 2016. Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR* abs/1611.02648.

Fard, M. M.; Thonet, T.; and Gaussier, É. 2018. Deep k-means: Jointly clustering with k-means and learning representations. *CoRR* abs/1806.10069.

Ghasedi Dizaji, K.; Herandi, A.; Deng, C.; Cai, W.; and Huang, H. 2017. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE ICCV, 2017*, 5736–5745.

Guo, X.; Gao, L.; Liu, X.; and Yin, J. 2017. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th IJCAI, 2017*, 1753–1759.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *IEEE CVPR, 2015* 770–778.

He, T.; Zhang, Z.; Zhang, H.; Zhang, Z.; Xie, J.; and Li, M. 2018. Bag of tricks to train convolutional neural networks for image classification. In *The IEEE CVPR, 2018.*

Houben, S.; Stallkamp, J.; Salmen, J.; Schlipsing, M.; and Igel, C. 2013. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 IJCNN*, 1–8. IEEE.

Ji, P.; Zhang, T.; Li, H.; Salzmann, M.; and Reid, I. D. 2017. Deep subspace clustering networks. In *Proceedings of the 30th NeurIPS,2017*, 23–32.

Jiang, Z.; Zheng, Y.; Tan, H.; Tang, B.; and Zhou, H. 2017. Variational deep embedding: an unsupervised and generative approach to clustering. In *Proceedings of the 26th AAAI,19-25, 2017*, 1965–1972. AAAI Press.

Johnson, J.; Hariharan, B.; van der Maaten, L.; Fei-Fei, L.; Lawrence Zitnick, C.; and Girshick, R. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE CVPR, 2017*, 2901–2910.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Le, Q. V.; Karpenko, A.; Ngiam, J.; and Ng, A. Y. 2011. Ica with reconstruction cost for efficient overcomplete feature learning. In *Advances in NIPS, 2011*, 1017–1025.

LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-sne. *JMLR* 9(Nov):2579–2605.

Mautz, D.; Ye, W.; Plant, C.; and Böhm, C. 2018. Discovering non-redundant k-means clusterings in optimal subspaces. In *Proceedings of the 24th ACM SIGKDD, 2018*, 1973–1982.

Meilă, M. 2007. Comparing clusterings—an information based distance. *J. of multivariate analysis* 98(5):873–895.

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

Mukherjee, S.; Asnani, H.; Lin, E.; and Kannan, S. 2019. Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the 33rd AAAI*, 4610–4617.

Müller, E.; Günnemann, S.; Färber, I.; and Seidl, T. 2012. Discovering multiple clustering solutions: Grouping objects in different views of the data. In *Proceedings of the 28th ICDE, Washington, DC, USA, 1-5 April, 2012*, 1207–1210.

Niu, D.; Dy, J. G.; and Jordan, M. I. 2010. Multiple non-redundant spectral clustering views. In *Proceedings of the 27th ICML*, 831–838.

Sculley, D. 2010. Web-scale k-means clustering. In *Proceedings of the 19th WWW*, 1177–1178. ACM.

Smith, L. N., and Topin, N. 2017. Super-convergence: Very fast training of residual networks using large learning rates. *ArXiv* abs/1708.07120.

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

Xie, J.; Girshick, R. B.; and Farhadi, A. 2016. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd ICML, 2016*, 478–487.

Yang, B.; Fu, X.; Sidiropoulos, N. D.; and Hong, M. 2017. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th ICML, 2017*, volume 70, 3861–3870. PMLR.

Ye, W.; Maurus, S.; Hubig, N.; and Plant, C. 2016. Generalized independent subspace clustering. In *Proceedings of the 16th ICDM, 2016, Spain*, 569–578.

Zhang, T.; Ji, P.; Harandi, M.; Hartley, R. I.; and Reid, I. D. 2018. Scalable deep k-subspace clustering. In *ACCV (5)*, volume 11365 of *Lecture Notes in Computer Science*, 466–481. Springer.

Supplementary - Deep Embedded Non-Redundant Clustering

Lukas Miklautz,[*,1] **Dominik Mautz,**[*,2] **Muzaffer Can Altinigneli,**[1,2] **Christian Böhm,**[2,3] **Claudia Plant**[1,4]
[1]Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Ludwig-Maximilians-Universität München, Munich, Germany
[3]MCML, [4]ds:UniVie
[1]{lukas.miklautz,claudia.plant}@univie.ac.at
[2]{altinigneli,boehm,mautz}@dbs.ifi.lmu.de

Abstract

This document adds further information about the training setup and supplementary material like images for the data sets presented in the paper and a table with the used notation in Table 1. The accompanying Python code of ENRC and the used data sets can be found at https://gitlab.cs.univie.ac.at/lukas/enrcpublic.

1 Autoencoder Training

We utilize PyTorch (Paszke et al. 2017) machine learning framework and Fastai library (Howard and others 2018) for neural network training. We refactored PyTorch basic-blocks into more abstract ENRC sequential-building-blocks to improve code readability and to avoid coding mistakes. We illustrate the encoder's building blocks in Figure 1[1], in top-down increasing abstraction levels. We depict PyTorch's basic-blocks in white color and ENRC sequential-building-blocks in various other colors. The *Conv2d* applies two dimensional convolution over an image grid composed of several input channels and scales it down with a stride parameter that is greater than one. We employ the Rectified Linear Units *ReLU* introduced in (Nair and Hinton 2010) to mitigate the vanishing gradient problem and their leaky variants to prevent dying *ReLU*'s respectively. We use two dimensional batch norm blocks *BatchNorm2d* (Ioffe and Szegedy 2015) to accelerate training by using higher learning rates. The two dimensional dropout block *Dropout2d* in (Srivastava et al. 2014) prevents the layer's neurons overfitting on certain inputs. The encoder's convolutional layers contain $conv\_and\_res$ blocks with different depths or number of filters.

We design the decoder similarly as illustrated in Figure 2. The *ConvTranspose2d* block in the decoder acts like the inverse operation of the *Conv2d* block in the encoder. The *ConvTranspose2d* scales up the image grid by padding zeros in-between existing pixels and learns the convolutional weights for the up-scaled image. The decoder's inverse convolutional layers contain $conv\_trans\_and\_res$ blocks with different depths or number of filters. In order not to clutter the figures, we omit a few PyTorch blocks, such as the *Upsample* block, which is the first block of the decoder applied after the last *AdaptiveAvgPool2d* block of the encoder.

The optimal maximum learning rate is determined with Fastai's LR-Finder over a range of potential learning rates. Both the learning rate and the momentum of Adam optimization (Kingma and Ba 2014) are annealed based on (Smith 2015; Smith and Topin 2017; Smith 2018) during batch-wise training as illustrated in Figure 3. Our goal is to train our neural network faster than compared to the standard methods, so called super-convergence using cyclic learning in one-cycle. We present ENRC's network layer and training settings with corresponding data sets in Table 2. The number of input pixels are given in PyTorch [width x height x channels] order. The number of decoder filters at each layer is organized in the reverse order of number of encoder filters, therefore is left out. For all data sets the same hidden layer (hl) dimensionality of 16 was used.

Table 1: Most important symbols

Symbol	Interpretation
$D \in \mathbb{N}$	Dimensionality of the embedded space
$S \in \mathbb{N}$	Number of clusterings
$K_s \in \mathbb{N}$	Number of Clusters in the s'th clustering
$\mathcal{C}_{s,i}$	Objects of cluster i in clustering s
$\mathbf{x}$	A data point or object of the data set
z	A data point embedded by $\mathrm{enc}(\mathbf{x})$
$\boldsymbol{\mu}_{s,i} \in \mathbb{R}^d$	Cluster center of cluster i of clustering s
$V \in \mathbb{R}^{d\times d}$	Linear transformation initialized as an orthogonal matrix (rigid transformation)
$\beta_s \in \mathbb{R}^d$	Feature weights of clustering s

2 Datasets

2.1 NR-Objects

In this section we show a sample of images from the used data sets and their respective autoencoder reconstructions. In Figure 4 we show objects from the NR-Objects data set. To generate the images, we used the publicly available ren-

*First authors with equal contribution.

[1]All figures are best viewed in color.

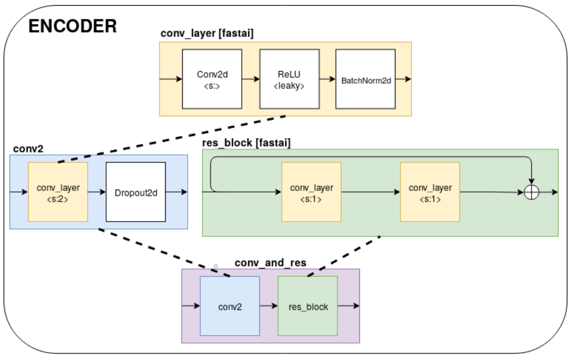


Figure 1: The architecture of Encoder. We depict PyTorch's basic-blocks in white color and ENRC sequential-building-blocks in various other colors. $< s >$ indicates stride and $< leaky >$ indicates leakage-slope respectively.

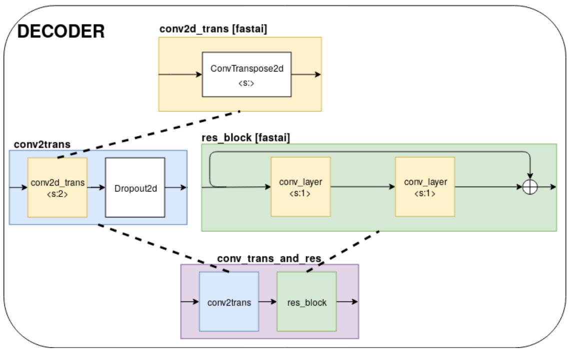


Figure 2: The architecture of Decoder is very similar to the architecture of Encoder.

Table 2: Summary of Network Layer and Training Settings. L2 Regularization with weight-decay= $1e-3$. Filters were chosen based on number of color channels. C-MNIST & Stickfigures are only greyscale, whereas NR-Objects & GT-SRB are RGB images.

Name	# Input Pixels	# Encoder Filters	hl;dropout rate;leaky
C-MNIST	56x56x1	[8,16,32,64,128]	16;0.1;0.1
NR-Objects	128x128x3	[16,32,64,128,256]	16;0.3;0.1
GTSRB	32x32x3	[16,32,64,128,256]	16;0.1;0.1
Stickfigures	32x32x1	[8,16,32,64,128]	16;0.2;0.1

dering software[2], which was used in (Johnson et al. 2017). Each object can be clustered by three shapes, two materials and six colors.

2.2 C-MNIST

In Figure 5 the images and reconstructions of the independently concatenated MNIST digits (LeCun et al. 1998) is depicted. With this extension we simulate positional non-redundancy. This data set can be seen as containing two-digit numbers, from 00 to 99, where each digit (left and right) is independent from the other. We added black row wise pixels, so the new format is 56×56 instead of 28×56, which would result from only concatenating the 28×28 MNIST images.

2.3 GTSRB

In Figure 6 a subset of the used images from the German Traffic Sign Benchmark (GTSRB) data set (Houben et al. 2013) are shown. The images have been preprocessed with histogram equalization to account for the low contrast of some of the images. We used a non-redundant subset of 4 different traffic signs: 'Speed limit (70km/h)', 'No passing', 'Ahead only', 'Keep right', which can be non-redundantly clustered w.r.t. to the four types of traffic sign and their two colors.

2.4 Stickfigures

The stickfigures data set contains nine basic objects of dancing figures. A subset of these objecst is depicted in Figure 7. It contains three clusterings for the upper and three for the lower body pose, a more detailed explanation can be found in (Ye et al. 2016).

[2]https://github.com/facebookresearch/clevr-dataset-gen

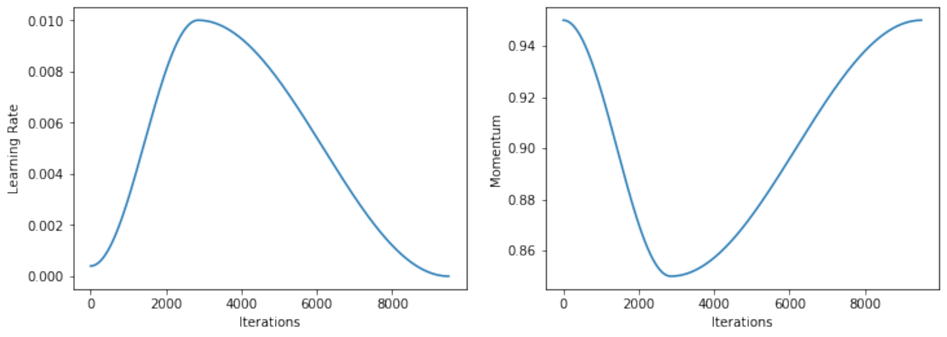
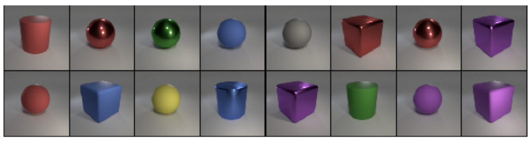
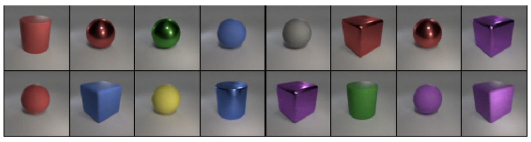


Figure 3: One-Cycle-Learning. Automated increasing and decreasing learning rates and corresponding momentum adaptation.

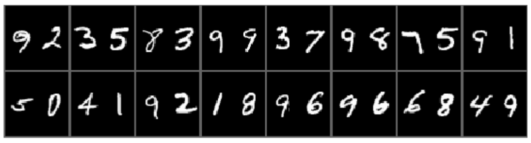


(a) Originals

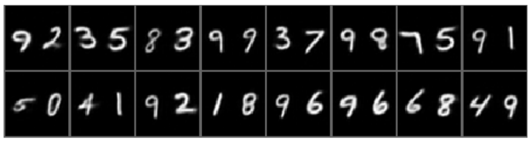


(b) Autoencoder reconstructions

Figure 4: A subset of images from the Non-Redundant-Objects data set.

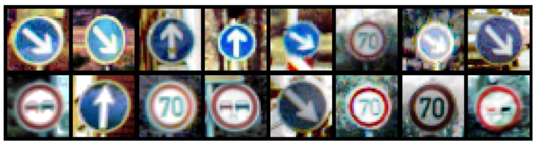


(a) Originals

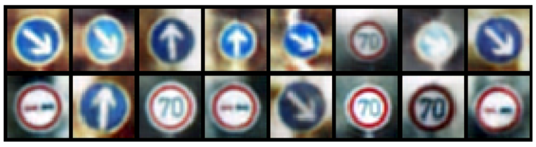


(b) Autoencoder reconstructions

Figure 5: A subset of images from the Concatenated-MNIST data set.

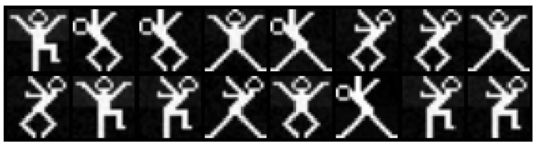


(a) Originals

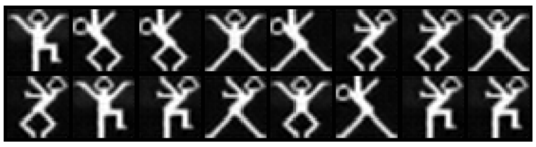


(b) Autoencoder reconstructions

Figure 6: A subset of images from the histogram equalized GTSRB data set.



(a) Originals



(b) Autoencoder reconstructions

Figure 7: A subset of images from the stickfigures data set.

References

Houben, S.; Stallkamp, J.; Salmen, J.; Schlipsing, M.; and Igel, C. 2013. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks IJCNN*, 1–8. IEEE.

Howard, J., et al. 2018. fastai. https://github.com/fastai/fastai.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, 448–456. JMLR.org.

Johnson, J.; Hariharan, B.; van der Maaten, L.; Fei-Fei, L.; Lawrence Zitnick, C.; and Girshick, R. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017*, 2901–2910.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, 807–814. USA: Omnipress.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Smith, L. N., and Topin, N. 2017. Super-convergence: Very fast training of residual networks using large learning rates. *ArXiv* abs/1708.07120.

Smith, L. N. 2015. Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)* 464–472.

Smith, L. N. 2018. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *ArXiv* abs/1803.09820.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958.

Ye, W.; Maurus, S.; Hubig, N.; and Plant, C. 2016. Generalized independent subspace clustering. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, 569–578.

A.2. Details (Don't) Matter: Isolating Cluster Information in Deep Embedded Spaces

Title	Details (Don't) Matter: Isolating Cluster Information in Deep Embedded Spaces
Authors	Lukas Miklautz, Lena G. M. Bauer, Dominik Mautz, Sebastian Tschiatschek, Christian Böhm and Claudia Plant
Publication Outlet	Proceedings of the 30th International Joint Conference on Artificial Intelligence, (IJCAI) 2021. Note: CORE ranking: A*; Acceptance rate: 19.3%
DOI-URL	`https://doi.org/10.24963/ijcai.2021/389`

Division of Work This contribution was developed with intensive cooperation between the first two authors with regular meetings resulting in equal contribution of the first two authors to this paper.

Lukas Miklautz: conceiving the initial idea; problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of variations of the method; experimental design; execution of experiments; writing and visualization of major parts of the article; review of relevant related work; creating the conference poster and presentation.

Lena G. M. Bauer: problem identification, definition and refinement; formulating the loss function and optimization strategy; formalization of algorithm and derivation of connection to existing methods; experimental design; writing and visualization of major parts of the article; creating the conference poster and presentation.

Dominik Mautz: problem identification, definition and refinement; review of relevant related work; regular discussions of candidate methods and potential experiments; suggestions for enhancements and improvements; periodic review of drafts for the paper;

Sebastian Tschiatschek: regular discussions of candidate methods and potential experiments; suggestions for enhancements and improvements; periodic review of drafts for the paper; mentoring and supervision.

Christian Böhm: regular discussions of candidate methods and potential experiments; suggestions for enhancements and improvements; periodic review of drafts for the paper; mentoring and supervision.

Claudia Plant: regular discussions of candidate methods and potential experiments; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Abstract	Deep clustering techniques combine representation learning with clustering objectives to improve their performance. Among existing deep clustering techniques, autoencoder-based methods are the most prevalent ones. While they achieve promising clustering results, they suffer from an inherent conflict between preserving details, as expressed by the reconstruction loss, and finding similar groups by ignoring details, as expressed by the clustering loss. This conflict leads to brittle training procedures, dependence on trade-off hyperparameters and less interpretable results. We propose our framework, ACe/DeC, that is compatible with **A**utoencoder **Ce**ntroid based **De**ep-**C**lustering methods and automatically learns a latent representation consisting of two separate spaces. The clustering space captures all cluster-specific information and the shared space explains general variation in the data. This separation resolves the above mentioned conflict and allows our method to learn both detailed reconstructions and cluster specific abstractions. We evaluate our framework with extensive experiments to show several benefits: (1) cluster performance – on various data sets we outperform relevant baselines; (2) no hyperparameter tuning – this improved performance is achieved without introducing new clustering-specific hyperparameters; (3) interpretability – isolating the cluster-specific information in a separate space is advantageous for data exploration and interpreting the clustering results; and (4) dimensionality of the embedded space – we automatically learn a low-dimensional space for clustering. Our ACe/DeC framework isolates cluster information, increases stability and interpretability, while improving cluster performance.
Thesis-Reference	[MBM$^+$21]

Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)

Details (Don't) Matter: Isolating Cluster Information in Deep Embedded Spaces

Lukas Miklautz[1,*] , **Lena G. M. Bauer**[3,*] , **Dominik Mautz**[2] , **Sebastian Tschiatschek**[1,3] , **Christian Böhm**[2,4] and **Claudia Plant**[1,3]
[1]Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Ludwig-Maximilians-Universität München, Munich, Germany
[3]ds:UniVie, Austria,
[4]MCML, Germany
{lukas.miklautz, lena.bauer}@univie.ac.at

Abstract

Deep clustering techniques combine representation learning with clustering objectives to improve their performance. Among existing deep clustering techniques, autoencoder-based methods are the most prevalent ones. While they achieve promising clustering results, they suffer from an inherent conflict between preserving details, as expressed by the reconstruction loss, and finding similar groups by ignoring details, as expressed by the clustering loss. This conflict leads to brittle training procedures, dependence on trade-off hyperparameters and less interpretable results. We propose our framework, ACe/DeC, that is compatible with **A**utoencoder **Ce**ntroid based **De**ep-**C**lustering methods and automatically learns a latent representation consisting of two separate spaces. The clustering space captures all cluster-specific information and the shared space explains general variation in the data. This separation resolves the above mentioned conflict and allows our method to learn both detailed reconstructions and cluster specific abstractions. We evaluate our framework with extensive experiments to show several benefits: (1) cluster performance – on various data sets we outperform relevant baselines; (2) no hyperparameter tuning – this improved performance is achieved without introducing new clustering-specific hyperparameters; (3) interpretability – isolating the cluster-specific information in a separate space is advantageous for data exploration and interpreting the clustering results; and (4) dimensionality of the embedded space – we automatically learn a low-dimensional space for clustering. Our ACe/DeC framework isolates cluster information, increases stability and interpretability, while improving cluster performance.

1 Introduction

The collection of massive amounts of complex data like images, text, video and audio gives rise to ever emerging challenges for data scientists trying to find patterns within data.

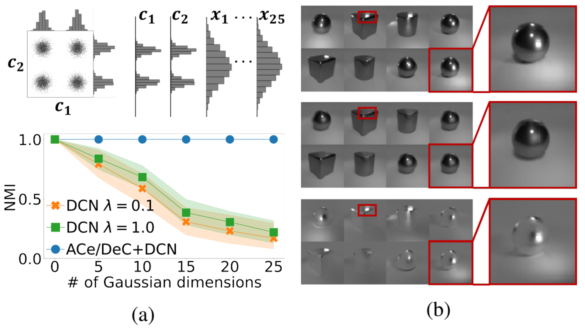


Figure 1: **(a)** Impact of shared information on cluster performance. [top left] A synthetic 2D data set with four clusters, that can be perfectly clustered by all methods (step 0 on x-axis). [top right] Adding dimensions, which do not contain cluster specific information (e.g., unimodal Gaussian distributed), considerably hurts the performance of DCN (measured in average NMI and 95% confidence intervals over 20 runs). In contrast, DCN combined with our ACe/DeC framework remains stable as it can isolate the cluster information. **(b)** Our ACe/DeC framework applied to the OBJECTS data set, which consists of images of cubes, spheres, and cylinders under different lighting conditions. Our framework allows DCN to isolate the cluster-specific information (the shape), from shared information mainly relevant for reconstruction (the lighting). [top] Input sample. [middle] Cluster information: Reconstructions using only dimensions of the cluster space show distinct shapes, but have less light information. [bottom] Shared information: Reconstructions from the shared space dimensions contain only light information.

Often they work in an unsupervised setting without access to labels, as these might be very costly or impossible to obtain. We can learn these labels with clustering methods, which partition the data into similar groups. Unfortunately, clustering high-dimensional data such as images directly works unsatisfactorily. In such situations it is beneficial to combine clustering with deep learning [Xie *et al.*, 2016] to automatically learn a high-quality, low-dimensional representation for clustering. This quite recent field of research is termed *deep clustering* (DC). Autoencoder (AE)-based DC approaches, such as DEC [Xie *et al.*, 2016], IDEC [Guo *et al.*, 2017] or DCN [Yang *et al.*, 2017], in particular, are well-known and serve as building blocks for many other methods, see, e.g.,

[*]Authors with equal contribution

the survey by [Aljalbout *et al.*, 2018]. AEs are a common approach to learn non-linear embeddings of high-dimensional data. They consist of an encoder network, which maps the input $\mathbf{x}$ to a lower-dimensional space and a decoder network, which produces a reconstruction $\hat{\mathbf{x}}$ optimized for minimizing a reconstruction loss $\mathcal{L}_{\text{rec}} = ||\mathbf{x} - \hat{\mathbf{x}}||$. DC methods learn a 'cluster friendly' embedding by adding a clustering objective to the reconstruction loss. DEC, IDEC or DCN, e.g. are *centroid-based* as they use a k-means-like objective for 'moving' points closer to their respective centroids. They learn the reconstruction and clustering in a single embedded space, which blends the features necessary for clustering and reconstruction, leading to brittle performance and less interpretable results.

These drawbacks are due to an inherent conflict between the reconstruction loss of the AE, that tries to preserve all details and the clustering loss, that tries to abstract from details. One might say, "that an AE wants to learn everything a [clustering algorithm] wants to forget" [Epstein and Meir, 2019]. An instance of this conflict can be seen in the synthetic data set in Figure 1a. Adding features that are irrelevant for the clustering, but important for the reconstruction, hurts the clustering performance of DCN drastically. DCN and other DC algorithms that consider all features equally important for clustering and reconstruction, attempt to solve this issue by weighing the trade-off between cluster loss and reconstruction loss with a new hyperparameter λ. Importantly, automatically tuning λ is very hard if not impossible in the unsupervised setting and existing work leaves open on how to tune this parameter without access to ground truth labels. However, as we can see in Figure 1a DCN performs poorly, for low and high λ values. This shows that a hyperparameter based approach to address the balancing of reconstruction and clustering loss in general is not sufficient. Unfortunately, simply removing $\mathcal{L}_{\text{rec}}$ can lead to arbitrary clustering solutions [Guo *et al.*, 2017], which is why the reconstruction - clustering - dilemma needs to be approached differently.

In this work we propose to resolve the mentioned drawbacks with our novel ACe/DeC framework, that is compatible with **A**utoencoder **Ce**ntroid-based **De**ep **C**lustering methods. We rephrase the DC objective to account for both, cluster and non-cluster information. We do this by learning an individual weight for each dimension of the embedded space separating cluster-specific information from shared information, allowing us to learn good clusterings while keeping all the details. In Figure 1a we can see how our ACe/DeC framework removes the impact of irrelevant dimensions and improves the performance of DCN considerably. Another instance of irrelevant information for clustering is shown in Figure 1b. The OBJECTS data set consists of images of spheres, cubes and cylinders, with random lighting, where the light source is mostly relevant for reconstruction, but not for distinguishing between objects as it is shared across all images. Our ACe/DeC framework allows DCN to isolate the cluster-specific features from the ones that are shared as shown in the middle and bottom row of Figure 1b, respectively.

In the following, we will first introduce the general framework and how the architecture is designed to achieve the space separation. Then we apply ACe/DeC to one popular AE

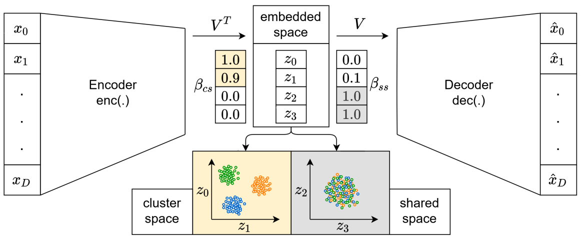


Figure 2: The overall architecture of ACe/DeC with the cluster space and the shared space. Our ACe/DeC framework extends the AE by introducing a learnable linear d $\times$ d transformation matrix $\mathbf{V}$ and two non-negative weight vectors β_{cs} and β_{ss} in the embedded space of the AE. The two weight vectors indicate which dimensions are important for clustering (β_{cs}) and which are not (β_{ss}). Here the β_{cs} weights indicate that the first two neurons z_0, z_1 have high cluster structure, while the last two neurons z_2, z_3 do not. In the plot below, one can see an example of a four dimensional embedded space split by ACe/DeC into two separate 2D spaces. We can see that the cluster space contains well-separated clusters, while the shared space contains only a single Gaussian without cluster structure. The matrix $\mathbf{V}$ aligns the clusters along the axes, based on the β-weights.

centroid-based DC procedure, namely DCN. We show experimentally the improvement of DCN with the framework compared to the original DCN algorithm as well as other competitors regarding

(1) **cluster performance** – on various data sets we outperform relevant baselines with increased stability regarding the choice of learning rate and cluster performance. Furthermore, we propose an optional augmentation procedure for image data, that can be used to further increase cluster performance and stability.

(2) **hyperparameter tuning** – most DC algorithms use cluster-specific hyperparameters that have to be tuned with ground truth labels. We remove such hyperparameters by rephrasing the DC objective to isolate cluster-specific information from shared information. This avoids having any data-specific hyperparameters to quantify the importance of cluster information making our approach more flexible to use in practice.

(3) **interpretability** – the isolation of cluster information leads to a natural separation of the embedded space into a cluster space and a shared space and enables a look into the AE blackbox. It is essential for visualizing and interpreting the clustering.

(4) **dimensionality of the embedded space** – ACe/DeC automatically determines the number of dimensions needed for clustering by removing irrelevant features.

2 Methodology

The AE is a key element of DC. It consists of an encoder network $\text{enc}(\cdot)$, which learns to project an input data point $\mathbf{x} \in \mathbb{R}^{\text{D}}$ from a data set X to an embedded (latent) vector $\mathbf{z} = \text{enc}(\mathbf{x})$ and a decoder network $\text{dec}(\cdot)$, which learns to project the embedded data point $\mathbf{z} \in \mathbb{R}^{\text{d}}$ back to the original data space, resulting in the reconstruction $\hat{\mathbf{x}} := \mathrm{dec}(\mathbf{z})$ of $\mathbf{x}$. The dimension d of $\mathbf{z}$ is often chosen to be smaller than the input dimension D. This bottleneck architecture avoids that the AE simply learns to copy the input to the output, which would be possible for d $\geq$ D, if no other regularization is added.

2.1 Method

To separate cluster information from non-cluster information we reformulate the commonly used DC objective $\mathcal{L} = \mathcal{L}_{\text{comp}} + \mathcal{L}_{\text{rec}}$. $\mathcal{L}_{\text{comp}}$ is the compression loss which is responsible for minimizing the distance of a point $\mathbf{z}$ to its closest centroid μ_k and $\mathcal{L}_{\text{rec}}$ is the AE reconstruction loss. Instead, we propose the loss function $\mathcal{L} = \mathcal{L}_{\text{cluster}} + \mathcal{L}_{\text{shared}} + \mathcal{L}_{\text{rec}}$, where the term $\mathcal{L}_{\text{shared}}$ is responsible for capturing the shared information by modelling the distance to the mean of the embedded data μ. In conjunction with the reconstruction loss this has the effect that we retain variance within the clusters, but are also able to model the common overall variance. $\mathcal{L}_{\text{cluster}}$ compresses the clusters similar to $\mathcal{L}_{\text{comp}}$. In particular we use $\mathcal{L} =$

$$\underbrace{\sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} \mathrm{dist}_{\beta_{cs}}(\mathbf{V^T z}, \mathbf{V^T}\mu_k)}_{\mathcal{L}_{\text{cluster}}} + \underbrace{\sum_{\mathbf{z}\in C} \mathrm{dist}_{\beta_{ss}}(\mathbf{V^T z}, \mathbf{V^T}\mu)}_{\mathcal{L}_{\text{shared}}} + \underbrace{\sum_{\mathbf{x}\in X} \mathrm{dist}_2(\mathbf{x}, \mathrm{dec}(\mathbf{V}\mathbf{V^T z}))}_{\mathcal{L}_{\text{rec}}}, \quad (1)$$

where $\mathrm{dist}_{\beta_s}(\cdot)$, $s \in \{cs, ss\}$ are generic distance functions, e.g. for the squared euclidean distance ($\mathrm{dist}_2(\cdot)$) and for some d-dimensional vectors $\mathbf{h}, \mathbf{g}$ we define

$$\mathrm{dist}_{\beta_s}(\mathbf{h}, \mathbf{g}) = ||\mathbf{h} - \mathbf{g}||^2_{\beta_s} = \sum_{i=1}^{d} \beta_s[i]^2 (\mathbf{h}[i] - \mathbf{g}[i])^2. \quad (2)$$

Additionally, we have K clusters, with C_k as the set of all embedded data points in the k^{th} cluster with corresponding centroid μ_k. C is the set of all embedded data points in the single shared space cluster with centroid $\mu = \frac{1}{|C|}\sum_{\mathbf{z}\in C}\mathbf{z}$. We can now compute the assignment of objects to the given cluster centers in the cluster space by assigning them to their closest center $\arg\min_{k\in[1;K]} ||\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu_k||^2_{\beta_{cs}}$.

We introduced new learnable parameters above. A linear d $\times$ d transformation matrix $\mathbf{V}$ and two non-negative weight vectors β_{cs} and β_{ss}. We explain the intuition behind them in Figure 2. To learn which dimensions are relevant for clustering, we need the β_s-weights to indicate which dimensions contain cluster structure, e.g. according to the k-means model, and which are not. Therefore, we use a trainable d-dimensional parameter $\mathbf{b}$ that is constrained by a sigmoid function $\beta_{cs}[i] = \mathrm{sigmoid}(\mathbf{b}[i]) := \frac{1}{1+\exp(-\mathbf{b}[i])}$ and $\beta_{ss}[i] = 1 - \beta_{cs}[i]$, where $[i]$ refers to the i^{th} component of a vector. Each component weighs one dimension of the embedded space. The two non-negative weight vectors β_{cs} and β_{ss} are constrained by $\beta_{cs} + \beta_{ss} = \vec{1}$, because the sigmoid function is between 0 and 1. These vectors represent a soft-assignment mechanism to the cluster space (cs) and shared space (ss), respectively.

The linear transformation matrix $\mathbf{V}$ can be seen as a linear layer which, 'guided' by the β-weights, axis-aligns the cluster structure along the most important cluster dimensions. With the above we can measure distances in two separated spaces using our weighted distances $\mathrm{dist}_{\beta_s}(\cdot)$, while still being differentiable. Altogether our ACe/DeC framework is lightweight requiring only d^2 + d trainable parameters for the matrix $\mathbf{V}$ and weights β_s, where d is usually much lower than the original data dimensionality D. Before we apply ACe/DeC to DCN we first state the loss function of DCN $\mathcal{L} = \frac{\lambda}{2}\mathcal{L}_{\text{comp}} + \mathcal{L}_{\text{rec}}$

$$= \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} \mathrm{dist}_2(\mathbf{z}, \mu_k) + \sum_{\mathbf{x}\in X} \mathrm{dist}_2(\mathbf{x}, \mathrm{dec}(\mathbf{z})) \quad (3)$$

$$= \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} ||\mathbf{z} - \mu_k||_2^2 + \sum_{\mathbf{x}\in X} ||\mathbf{x} - \hat{\mathbf{x}}||_2^2, \quad (4)$$

with the trade-off hyperparameter $\lambda > 0$. To integrate ACe/DeC into DCN, we adjust their loss function with our two new learnable parameters $\mathbf{V}$ and $\mathbf{b}$. We can then compute $\mathcal{L} = \mathcal{L}_{\text{cluster}} + \mathcal{L}_{\text{shared}} + \mathcal{L}_{\text{rec}}$ as

$$\sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} ||\mathbf{V^T z} - \mathbf{V^T}\mu_k||^2_{\beta_{cs}} + \sum_{\mathbf{z}\in C} ||\mathbf{V^T z} - \mathbf{V^T}\mu||^2_{\beta_{ss}} + \sum_{\mathbf{x}\in X} ||\mathbf{x} - \mathrm{dec}(\mathbf{V}\mathbf{V^T z})||_2^2, \quad (5)$$

where we can now learn which dimensions are important to the k-means clustering and which are not, without any potentially crucial hyperparameters. To further motivate this loss, one can think about the β_s-weighted euclidean distance from Eq.2 as rescaling the dimensions based on the cluster structure. If in the ith dimension the k-means centroids represent the embedded data points better than a single centroid, $\beta_{cs}[i]$ will increase and $\beta_{ss}[i]$ will decrease accordingly in the next update step in order to lower the loss. We show this with a mathematical analysis by considering the gradient of the loss function with respect to $\mathbf{b}[i]$, which determines the update of the $\beta_s[i]$-weights. We present this and further analysis in the Supplement (SP) [Miklautz *et al.*, 2021] (Sec. 1.1 - 1.5).

Another motivation for combining ACe/DeC with DCN comes from SubKmeans [Mautz *et al.*, 2017]. We show in the SP (Sec. 1.6) that for a linear AE both methods optimize the same objective, because the reconstruction error will then force $\mathbf{V}$ to be orthogonal. With the ACe/DeC+DCN loss function we only update $\mathbf{V}$ and the space weights β_s; the update of centroids and cluster assignments for DCN is done separately. The original update procedure for DCN is alternating between the update of the AE parameters with fixed centroids and updating the cluster assignments and centroids with fixed AE parameters. We implemented another strategy, which allows for an update without alternation using mini-batch k-means [Sculley, 2010], for details see SP (Sec. 1.7).

2.2 Initialization and Augmentation Procedure

We initialize $\mathbf{V}$ as a random orthogonal matrix and randomly assign each dimension to one of the two spaces. We then perform k-means in the cluster space to get the initial centroids

μ_k. As the shared space is modeled with a single cluster, we initialize its centroid with the mean of the embedded data. This adds only little overhead and is similar to other k-means based DC algorithms. We then train $\mathbf{V}$ and $\mathbf{b}$ while holding everything else constant to get an initial estimate.

Optionally, our ACe/DeC framework allows to leverage domain knowledge in the form of data augmentation to increase clustering performance. For this we use $\mathbf{x}_{aug} = f(\mathbf{x})$, where the function f augments the input data, e.g., by rotating it. We then minimize the distance of the embedded augmented data point $\mathbf{z}_{aug} = \text{enc}(\mathbf{x}_{aug})$ to the centroid of the original data point μ_k, with $\text{dist}_{\beta_{cs}}(\mathbf{V}^\mathbf{T}\mathbf{z}_{aug}, \mathbf{V}^\mathbf{T}\mu_k)$. This forces the clusters to be invariant to the augmentation. Additionally, we make use of our cluster and shared space. We reconstruct the augmented data point $\mathbf{x}_{aug}$ from the shared space and the original $\mathbf{x}$ from the clustered space. With this we 'move' information unnecessary for clustering to the shared space.

3 Related Work

There are two clustering approaches, which are conceptually related to ACe/DeC through the idea of separating features containing clustering structure from features without clustering structure. The algorithms FOSSCLU [Goebl *et al.*, 2014] and SubKmeans [Mautz *et al.*, 2017] search for a single optimal subspace in the k-means sense, but are bound to linear relationships and do not scale well to high-dimensional data. In addition, their objectives are non-differentiable due to discrete subspace assignments making them unsuitable for DC with gradient-based optimization. Furthermore, they cannot be optimized jointly together with the AE, limiting their representational power and clustering performance.

AE-based DC methods are arguably the most prevalent ones [Aljalbout *et al.*, 2018; Min *et al.*, 2018]—whether the AE is only used for an initial representation of the data as in the DEC algorithm [Xie *et al.*, 2016], or the reconstruction and clustering objective is jointly optimized as in methods like IDEC [Guo *et al.*, 2017], DCN [Yang *et al.*, 2017], JULE [Yang *et al.*, 2016], DualAE [Yang *et al.*, 2019] or DEPICT [Dizaji *et al.*, 2017]. DEPICT is one of the few methods that does not introduce a cluster-specific hyperparameter. It uses a convolutional AE and leverages noise augmentation for a more robust clustering. IDEC combines a reconstruction loss with an auxiliary target distribution to minimize the Kullback-Leibler divergence such as in the DEC algorithm, weighing the trade-off with a hyperparameter. For DCN, network parameters, (hard) cluster assignments and centroids are updated in alternation. In each step of DCN the full data set is passed and a carefully weighted k-means loss is optimized together with the reconstruction loss. Recently, more powerful techniques that combine data-specific architectures with several cluster-specific hyperparameters have been introduced. These hyperparameters have to be tuned with ground truth labels, leaving open how to set them in an unsupervised setting. JULE combines a recurrent convolutional AE with a joint clustering objective, leveraging a well-tuned triplet loss. DualAE combines AE-based deep spectral clustering with mutual information maximization and noise augmentation.

A recently proposed probabilistic DC method is ClusterGAN [Mukherjee *et al.*, 2019], which combines a generative adversarial network (GAN) [Goodfellow *et al.*, 2014] with a clustering prior. While not an AE-based method, it uses similar to other existing methods the same space for capturing cluster and non-cluster information, leading to a trade-off between losses and less interpretable results. In contrast to our method, ClusterGAN can not distinguish features shared between all clusters from the ones that are cluster-specific, as can be seen in Figure 4 of [Mukherjee *et al.*, 2019]. All of these methods need three cluster-specific hyperparameters tuned by ground truth labels to balance the different loss terms, but they do not account for the situation that different dimensions might be less important to the clustering as we motivated in Figure 1a. This is in contrast to our approach, as we learn the importance of each dimension and assume that we have no access to ground truth labels, which is usually the case for clustering in practice. ACe/DeC shares its idea of soft-assigned feature spaces with [Ji *et al.*, 2017; Zhang *et al.*, 2018; Zhang *et al.*, 2019; Miklautz *et al.*, 2020]. However, in contrast to these (deep) subspace clustering methods, which find a separate subspace for each cluster or multiple clustering subspaces, we find a single cluster subspace for all clusters and a single non-cluster space for shared information. To our knowledge we are the first to introduce this idea to DC for resolving the clustering/reconstruction trade-off. Our framework is developed for centroid-based DC methods, which is why we do not compare to algorithms outside of this family. There are, of course, powerful classical clustering methods, e.g. HDBSCAN [Campello *et al.*, 2013]. However, the clustering notion is very different (density-based vs centroid-based).

4 Experiments

We evaluate all algorithms with six different data sets focusing on common DC benchmarks like MNIST [LeCun *et al.*, 1998], Fashion-MNIST [Xiao *et al.*, 2017] and USPS. Additionally, we use a data set based on real world images of traffic signs (GTSRB) [Houben *et al.*, 2013], recorded under different camera angles and daylight conditions, and two synthetic data sets to show the impact of irrelevant information on DC performance. The OBJECTS data set consists of $10,000$ synthetically generated gray scale images of spheres, cubes and cylinders with $4,096$ dimensions. The SYNTH-25 data set consists of $12,000$ data points with four well-separated Gaussian clusters in two dimensions and 25 independent, unimodal Gaussian distributed dimensions without cluster structure (27 dimensions in total), corresponding to step 25 of the x-axis in Figure 1a. Further explanations about the experiments, data sets, information on our hardware setup, compared methods, and all additional experiments are in the SP (Sec. 2). We uploaded our code and supplement at https://gitlab.cs.univie.ac.at/lukas/acedec_public.

4.1 Quantitative Experiments

In this section we show that by using our ACe/DeC framework, we can achieve stable cluster performance across data sets with varying degree of non-cluster information, without tuning of the hyperparameter λ and using less dimensions for clustering. For the image data sets we use for all methods 10 pretrained

Methods	**SYNTH-25**	**FMNIST**	**MNIST-Full**	**MNIST-Test**	**OBJECTS**	**USPS**	**GTSRB**
ACe/DeC+DCN	**1.00** ±0.00	**0.62** ±0.00	**0.89** ±0.01	**0.87** ±0.02	**1.00** ±0.00	**0.71** ±0.02	**0.52** ±0.04
DCN ($\lambda = 0.1$)	0.17 ±0.24	**0.62** ±0.02	0.87 ±0.01	0.86 ±0.02	**1.00** ±0.00	**0.71** ±0.03	0.19 ±0.05
DCN ($\lambda = 1.0$)	0.22 ±0.21	0.55 ±0.03	0.86 ±0.03	0.83 ±0.03	0.99 ±0.01	0.65 ±0.03	0.16 ±0.07
DCN ($\lambda = 10.0$)	0.29 ±0.22	0.55 ±0.04	0.70 ±0.08	0.66 ±0.05	**1.00** ±0.00	0.56 ±0.04	0.24 ±0.19
k-means	0.90 ±0.16	0.51 ±0.01	0.50 ±0.00	0.50 ±0.00	**1.00** ±0.00	0.61 ±0.00	0.22 ±0.01
SubKmeans	0.84 ±0.13	0.52 ±0.00*	0.48 ±0.01*	0.48 ±0.02*	-	0.61 ±0.00	-
AE + SubKmeans	0.11 ±0.10	0.59 ±0.02	0.80 ±0.01	0.79±0.01	0.98 ±0.02	0.65 ±0.02	0.47 ±0.02
AE + k-means	0.11 ±0.10	0.59 ±0.01	0.80 ±0.01	0.79±0.01	0.98 ±0.02	0.65 ±0.02	0.46 ±0.03

Methods + Aug	**SYNTH-25**	**FMNIST**	**MNIST-Full**	**MNIST-Test**	**OBJECTS**	**USPS**	**GTSRB**
ACe/DeC+DCN	N/A	**0.64** ±0.01	**0.94** ±0.00	0.94 ±0.00	**1.00** ±0.00	**0.86** ±0.02	**0.66** ±0.02
DCN ($\lambda = 0.1$)	N/A	0.60 ±0.02	**0.94** ±0.00	**0.95** ±0.01	**1.00** ±0.00	0.83 ±0.02	0.64 ±0.02
DCN ($\lambda = 1.0$)	N/A	0.57 ±0.03	**0.94** ±0.00	**0.95** ±0.01	**1.00** ±0.00	0.84 ±0.03	0.65 ±0.06
DCN ($\lambda = 10.0$)	N/A	0.57 ±0.03	0.92 ±0.02	0.94 ±0.02	**1.00** ±0.00	0.78 ±0.05	0.65 ±0.06

Table 1: NMI averages and standard deviations over 10 (20 for SYNTH-25) pretrained AEs. We compare the results with and without image augmentation in the upper and lower tables, respectively. The **upper** table shows that the performance of DCN depends in general quite heavily on λ. For SYNTH-25 and GTSRB the performance of DCN does not improve considerably for any of the considered λ values. This is mainly due to the AE, which focuses too much on reconstructing the irrelevant dimensions instead of preserving the cluster information as can be seen when comparing k-means and AE+k-means for GTSRB and SYNTH-25. This can also be seen for the OBJECTS data set, but the effect is much smaller. In contrast, ACe/DeC allows DCN to perform stable across all data sets. The **lower** table shows the performance of ACe/DeC and DCN leveraging image augmentation (random rotation and shifts). Unsurprisingly, both methods improve through domain knowledge in the form of augmentation, but DCN still depends on λ for FMNIST and USPS, where the learned augmentation invariances might not be as relevant. Importantly, image augmentation can not solve this problem for non-image data sets like SYNTH-25.

Methods	**MNIST-Full**	**MNIST-Test**	**FMNIST**	**# HPs**
ACe/DeC+DCN	0.94	0.94	0.64	**0**
DEPICT	0.92	0.92	0.39	**0**
DEC	0.74	0.75	0.57	**0**
IDEC	0.80	0.77	0.61	1
JULE	0.91	0.92	0.61	3
C-GAN	0.89	0.89	0.64	3
DualAE	0.94	0.95	0.65	3

Table 2: Average NMI comparison among recently proposed techniques on common benchmarks. DEC and IDEC have been rerun based on our re-implementations and other results taken from the literature. Even though the comparison methods have an unrealistic advantage—hyperparameters (HP) tuned with access to labels—our performance is equally strong.

Method	**MNIST**	**FMNIST**	**USPS**	**GTSRB**
SubKM	10*	10*	10	-
AE + SubKM	9	9	9	9
ACe/DeC+DCN	**6**	**5**	**5**	**4**

Table 3: Average number of dimensions found for all ten pretrained AEs with $d = 10$. The joint non-linear optimization allows our method to reduce the dimensionality by a factor of two compared to the linear method SubKmeans (SubKM).

fully connected AEs with an embedding size of $d = 10$ according to the procedure in [Xie *et al.*, 2016]. We use this basic architecture, because all methods would profit from a more powerful AE. We benchmark two implementations of our ACe/DeC framework with DCN. One which leverages augmentation invariances (random rotation and shifts) and one which does not. We compare this to DCN with different values of λ. We used the same settings for the training of all DC methods (training budget, learning rate, optimizer, etc., see the SP (Sec. 2.4) for details). Additionally, we compare to SubKmeans [Mautz *et al.*, 2017] (AE + SubKmeans), k-means (AE + k-means) and included the results of k-means and SubKmeans on the raw data sets as well. Results marked with * were run on a subset of $10,000$ objects, empty results were stopped due to run time constraints. Additionally, we compare our results against a reimplemented DEC/IDEC using the same pretrained AEs and recently introduced state of the art DC methods that leverage augmentation and data-specific architectures, namely DEPICT, JULE, ClusterGAN (C-GAN) and DualAE where we report the results from the respective papers. For SYNTH-25 and the experiments in Figure 1a, we trained for each setting 20 single layer linear AEs with $d = D$.

Stable cluster performance without hyperparameter tuning. Table 1 shows the Normalized Mutual Information (NMI) [Vinh *et al.*, 2010] results of the considered methods and data sets, where an NMI close to 1 indicates perfect clustering and 0 a random one. Our method performs stable across different data sets, while DCN alone fails for a data set like GTSRB, that contains several features unrelated to clustering. In our synthetic SYNTH-25 data set, where the number of irrelevant dimensions overtakes the number of dimension with cluster information, we can see that DCN's performance is considerably worse independent of its λ value. Augmentation improves the performance of our method and of DCN, but DCN still depends on λ for data sets where the learned invariances might not be as important, like FMNIST and USPS. We show that the cluster performance of DCN highly depends on the choice of λ and its learning rate in Figure 4, while ACe/DeC stabilizes DCN across different learning rates and data sets. In Table 2 we compare our method against recently reported results in the literature that use augmentation and data-specific neural network architectures. We outperform DEC, IDEC, DEPICT, JULE and ClusterGAN (C-GAN) and perform close to DualAE, despite being at a disadvantage as

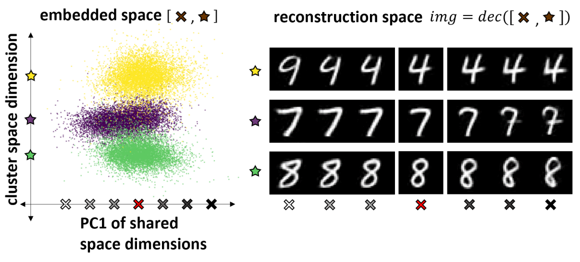


Figure 3: **Latent space traversal. (Left)** Plotting the dimension with most cluster structure (highest $\beta_{cs}[i]$) of ACe/DeC+DCN along the y-axis, we see that the digits 4,7 and 8 can be easily separated using a single dimension (colors indicate ground truth labels). The x-axis shows the direction of most variance in the shared space aka its first principal component (PC1). The equidistantly spaced markers ($\times$) along the PC1 axis indicate the traversal along this component. **(Right)** Each image shows the reconstructed data point with the corresponding coordinates in the embedded space plot on the left side, e.g., the top left image of the digit 4 is the reconstruction of the data point with the coordinates 'white cross' and 'yellow star'. Each row shows the traversal along PC1 for a fixed cluster space coordinate. Therefore, the cluster identity does not change, while the style varies as one moves from left to right, e.g., digit seven with and without horizontal bar. As style is a shared feature for digits, it is not contained in the cluster dimension, but can be captured in the shared space. These effects transfer to the other digits as well and other principal components reflect other features like rotation and thickness, showing that the shared space can capture multiple sources of variation at once, see SP (Sec. 2.1) for more details.

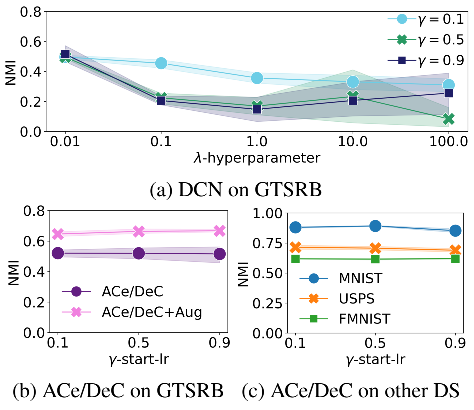


Figure 4: Figure 4a shows the average NMIs with 95% confidence intervals over 10 runs of DCN on GTSRB. We look at the stability of cluster performance w.r.t. different values of λ and different learning rates. We vary the learning rate with $10^{-3}\gamma$ for $\gamma \in \{0.1, 0.5, 0.9\}$, where the start value of 10^{-3} is based on the pretraining. We can observe, that the performance of DCN becomes very brittle. Its performance highly depends on different λ parameter and learning rate combinations. Furthermore, standard deviation is high for all parameter combinations. Using DCN with ACe/DeC does not need the λ hyperparameter, but we analyzed the performance for different learning rates in Figure 4b. The average NMIs are stable for all γ values with a standard deviation in a moderate range. The same holds for the other data sets (DS) as well, see Figure 4c.

we do not exploit ground truth labels.

Dimensionality of embedded space. Table 3 shows the average dimensions found with SubKmeans and ACe/DeC+DCN. The flexibility to learn a non-linear representation allows ACe/DeC to decrease the dimensionality needed for clustering by a factor of two. See SP (Sec. 1.4 and 2.2) on how to harden the soft assignments and more experiments.

4.2 Interpretability Experiments

In this section we show how we can use ACe/DeC to interpret which features are important for clustering and which are not. Using our framework, we gain three modes of understanding. First, we can separate the cluster and non-cluster information, as shown in Figure 1b. Second, we can traverse the latent space separately from the cluster space. With this we can view the change of varying information that is shared by all clusters. The analysis of the shared space of our algorithm when applied to MNIST can be seen in Figure 3. Here the direction of most variance of the shared space represents variation in style of MNIST digits. Note, that such an analysis is not possible for existing DC methods alone, which blend cluster and non-cluster information in a single space (see SP (Sec. 2.1) for examples). Third, by selecting the most discriminative dimensions based on β_{cs}, we can visualize the embedded space without the need of an additional dimensionality reduction technique like t-SNE [Maaten and Hinton, 2008], making our approach more faithful to the learned embedding as can be seen on the y-axis of Figure 3. We show another example for the OBJECTS data set in the SP (Sec. 2.1).

5 Conclusion

Sometimes details matter, sometimes they don't. Current AE-based DC methods need a priori knowledge about the data in the form of augmentation invariances or ground truth labels, to improve clustering results. However, this is an unrealistic setting for clustering and does not resolve the conflict between the autoencoder and clustering objective. We introduced our ACe/DeC framework that enables existing centroid-based DC algorithms to separate clustering information from shared information allowing the algorithm to preserve details for reconstruction and to abstract from details for clustering. Further, our framework improves interpretability, dimensionality reduction and performance stability, without a—in practice—impossible to tune hyperparameter. There are multiple promising directions to extend our framework, including but not limited to: integrating ACe/DeC into other centroid-based DC methods; replacing the reconstruction error with other self-supervised losses; or making our framework (semi-)supervised by combining it with a supervised AE [Le *et al.*, 2018].

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and the Federal Ministry of Education, Science and Research of Austria (project: Digitize! Computational Social Sciences). The authors of this work take full responsibility for its content.

References

[Aljalbout *et al.*, 2018] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *CoRR*, abs/1801.07648, 2018.

[Campello *et al.*, 2013] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[Dizaji *et al.*, 2017] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *ICCV*, pages 5747–5756, 2017.

[Epstein and Meir, 2019] Baruch Epstein and Ron Meir. Generalization bounds for unsupervised and semi-supervised learning with autoencoders. *CoRR*, abs/1902.01449, 2019.

[Goebl *et al.*, 2014] Sebastian Goebl, Xiao He, Claudia Plant, and Christian Böhm. Finding the optimal subspace for clustering. In *ICDM*, pages 130–139. IEEE Computer Society, 2014.

[Goodfellow *et al.*, 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[Guo *et al.*, 2017] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pages 1753–1759. ijcai.org, 2017.

[Houben *et al.*, 2013] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *IJCNN*, pages 1–8. IEEE, 2013.

[Ji *et al.*, 2017] Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian D. Reid. Deep subspace clustering networks. In *NeurIPS,2017*, pages 23–32, 2017.

[Le *et al.*, 2018] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In *NeurIPS*, pages 107–117, 2018.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Maaten and Hinton, 2008] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9:2579–2605, 2008.

[Mautz *et al.*, 2017] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Towards an optimal subspace for k-means. In *KDD*, pages 365–373. ACM, 2017.

[Miklautz *et al.*, 2020] Lukas Miklautz, Dominik Mautz, C Altinigneli, Christian Böhm, and Claudia Plant. Deep embedded non-redundant clustering. AAAI, 2020.

[Miklautz *et al.*, 2021] Lukas Miklautz, Lena Bauer, Dominik Mautz, Sebastian Tschiatschek, Christian Böhm, and Claudia Plant. Supplementary to: Details (don't) matter: Isolationg cluster information in deep embedded spaces. https://gitlab.cs.univie.ac.at/lukas/acedec_public, 2021.

[Min *et al.*, 2018] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[Mukherjee *et al.*, 2019] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan: Latent space clustering in generative adversarial networks. In *AAAI*, pages 4610–4617, 2019.

[Sculley, 2010] D. Sculley. Web-scale k-means clustering. In *WWW*, pages 1177–1178. ACM, 2010.

[Vinh *et al.*, 2010] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 10 2010.

[Xiao *et al.*, 2017] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[Xie *et al.*, 2016] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 478–487. JMLR.org, 2016.

[Yang *et al.*, 2016] Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *CVPR*, pages 5147–5156. IEEE Computer Society, 2016.

[Yang *et al.*, 2017] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*, volume 70, pages 3861–3870. PMLR, 2017.

[Yang *et al.*, 2019] Xu Yang, Cheng Deng, Feng Zheng, Junchi Yan, and Wei Liu. Deep spectral clustering using dual autoencoder network. In *CVPR*, pages 4066–4075. Computer Vision Foundation / IEEE, 2019.

[Zhang *et al.*, 2018] Tong Zhang, Pan Ji, Mehrtash Harandi, Richard I. Hartley, and Ian D. Reid. Scalable deep k-subspace clustering. In *ACCV (5)*, volume 11365 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2018.

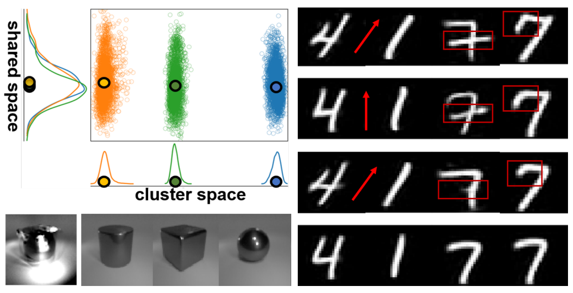
[Zhang *et al.*, 2019] Tong Zhang, Pan Ji, Mehrtash Harandi, Wenbing Huang, and Hongdong Li. Neural collaborative subspace clustering. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *ICML*, volume 97, pages 7384–7393. PMLR, 2019.

Supplementary to 'Details (Don't) Matter: Isolating Cluster Information in Deep Embedded Spaces'

Lukas Miklautz[1,*], **Lena G. M. Bauer**[3,*], **Dominik Mautz**[2], **Sebastian Tschiatschek**[1,3], **Christian Böhm**[2,4] and **Claudia Plant**[1,3]
[1]Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Ludwig-Maximilians-Universität München, Munich, Germany
[3]ds:UniVie, Austria, [4]MCML, Germany
{lukas.miklautz, lena.bauer}@univie.ac.at

Abstract

The supplementary material provides further details of our work. We explain derivations, relationships to other methods, our cluster update strategy, the used data sets and show the remaining experiments and visualizations as mentioned in the paper. Additionally, we describe our hard- and software setup, experiment settings, as well as the used evaluation metrics. For easier reproducibility of our results we uploaded the code, models and data sets at https://gitlab.cs.univie.ac.at/lukas/acedec_public.



(a) Embedding and centroids (b) Isolating cluster information

Figure 1: **(a)**ACe/DeC applied to the OBJECTS data set, which consists of images of cubes, spheres, and cylinders under different lighting conditions. [top] Embedded space: The plot shows two dimensions of the ACe/DeC embedded space. The dimension on the x-axis has a high cluster structure and is therefore assigned to the *cluster space*. On the y-axis, we show a dimension of the *shared space* that does not contain any discriminating features. [bottom] Cluster centroids: We show the three centroids from the cluster space on the right. They contain no varying light information, while the shared space centroid shown on the very left contains all the light information, but is an average of the shapes. **(b)** Isolating cluster information with our method for MNIST digits four, one and seven. [top] An input sample. [second from top] We highlighted in red how ACe/DeC separated the unique features, like cross bar and serif from the shared ones like rotation [second from bottom]. The last row shows the respective centroids for each input digit. Using our ACe/DeC framework we can gain further insights on how the algorithm arrived at its clustering.

1 Method

In this section, we provide further details about our proposed framework ACe/DeC and its application to DCN, referenced in the following as 'Ours'. We provide details about the joint clustering (Section 1.1) and how to prevent $\mathbf{V}$ from degeneration (Section 1.2). We show the derivation for the optimal $\mathbf{b}$ for our weights β_s (Section 1.3), derivatives for our loss function $\mathcal{L}$ w.r.t the linear transformation matrix $\mathbf{V}$ (Section 1.5) and the connections between our proposed algorithm and SubKmeans [Mautz *et al.*, 2017] (Section 1.6) and the mini-batch clustering procedure (Section 1.7).

1.1 Joint Clustering

The core motivation of deep clustering is to learn functions that map the input data to a cluster-friendly embedded space. The usual deep clustering pipeline is structured as follows. First, pretrain an autoencoder by minimizing the reconstruction loss $\mathcal{L}_{\text{rec}}$ to get an initial low dimensional embedding. Second, to improve the cluster structure, add a cluster objective $\mathcal{L}_{\text{comp}}$, with a carefully selected hyperparameter $\lambda > 0$ and jointly minimize $\mathcal{L}_{\text{rec}} + \lambda\mathcal{L}_{\text{comp}}$. Third, rerun k-means in case a centroid lost its cluster. For ACe/DeC all of this stays the same except that we use a different initialization procedure, that we do not need a weighting with a hyperparameter and that we harden the weights for dimensionality reduction.

1.2 Degeneration Prevention for V

To prevent the degenerate solution for $\mathbf{V}$, e.g., collapsing all points into a single centroid, we include—similar to [Le *et al.*, 2011]—our parameters in the autoencoder reconstruction loss $\mathcal{L}_{\text{rec}} = \text{dist}_2(\mathbf{x}, \text{dec}(\mathbf{V}\mathbf{V}^{\mathbf{T}}\mathbf{z})) = ||\mathbf{x} - \text{dec}(\mathbf{V}\mathbf{V}^{\mathbf{T}}(\beta_{cs}\mathbf{z} + \beta_{ss}\mathbf{z}))||_2^2 = ||\mathbf{x} - \text{dec}(\mathbf{V}\mathbf{V}^{\mathbf{T}}\mathbf{z})||_2^2$. Since $\beta_{cs} + \beta_{ss} = \vec{1}$, the weights drop out of $\mathcal{L}_{\text{rec}}$. To prevent degeneration during initialization of $\mathbf{V}$ and β we fix $\mathcal{L}_{\text{rec}}$ to enforce $\mathbf{V}\mathbf{V}^{\mathbf{T}} \approx \mathbf{I}$.

1.3 β_s-weights Analysis

In the main paper we claimed that if in the ith dimension the k-means centroids represent the embedded data points $\mathbf{V}^{\mathbf{T}}\mathbf{z}[i]$ better than a single centroid $\mathbf{V}^{\mathbf{T}}\mu[i]$, $\beta_{cs}[i]$ will increase and vice versa $\beta_{ss}[i]$ will decrease in the next update step in order to lower the loss. We argued that this can also be shown with a

*Authors with equal contribution

mathematical analysis by considering the gradient of the loss function with respect to $\mathbf{b}[i]$ as this determines the update of the $\beta_s[i]$-weights. In this section we provide the according mathematical derivations.

The ACe/DeC-adapted DCN loss as stated in the main paper is given by $\mathcal{L} = \frac{\lambda}{2}\left[\mathcal{L}_{\text{cluster}} + \mathcal{L}_{\text{shared}}\right] + \mathcal{L}_{\text{rec}} =$

$$\frac{\lambda}{2}\left[\sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} \|\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu_k\|^2_{\beta_{cs}} + \sum_{\mathbf{z}\in C} \|\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu\|^2_{\beta_{ss}}\right] + \sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x} - \operatorname{dec}(\mathbf{V}\mathbf{V}^T\mathbf{z})||^2_2, \tag{1}$$

where, for some vectors $\mathbf{h}, \mathbf{g} \in \mathbb{R}^d$, we define

$$\begin{aligned}\operatorname{dist}_{\beta_s}(\mathbf{h},\mathbf{g}) &= ||\mathbf{h} - \mathbf{g}||^2_{\beta_s} \\ &= \sum_{i=1}^{d} \left(\beta_s[i](\mathbf{h}[i] - \mathbf{g}[i])\right)^2 \\ &= \sum_{i=1}^{d} \beta_s[i]^2(\mathbf{h}[i] - \mathbf{g}[i])^2,\end{aligned} \tag{2}$$

with

$$\beta_{cs}[i] := \frac{1}{1+\exp(-\mathbf{b}[i])}, \quad \beta_{ss}[i] := 1 - \beta_{cs}[i]. \tag{3}$$

Note, that for completeness, we here keep the hyperparameter λ for now, even though it is not included in our optimization and learning procedure. As discussed in the main paper, we can think about the β_s-weighted euclidean distance as rescaling the dimensions based on the cluster structure, as for dimensions i where $\mathcal{L}_{\text{cluster}} < \mathcal{L}_{\text{shared}}$, $\beta_{cs}[i]$ will increase. The rescaling becomes also evident with another interpretation of our β_s distances. We can consider it as the squared Mahalanobis distance,

$$\operatorname{dist}^2_M(\mathbf{x},\mu) = \sum_{i=1}^{d}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu), \tag{4}$$

where the covariance matrix Σ is a diagonal matrix with the inverse of $\beta_s[i]^2$ as entries.

For the following analysis, we state the k-means loss $\mathcal{L}_{\text{KM}}$ and variance (scatter matrix) loss $\mathcal{L}_{\text{Var}}$

$$\begin{aligned}\mathcal{L}_{\text{K}} := \mathcal{L}_{\text{KM}} &:= \sum_{k=1}^{K}\sum_{\mathbf{z}\in C_k} ||\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu_k||^2_2 \\ \mathcal{L}_{\text{V}} := \mathcal{L}_{\text{Var}} &:= \sum_{\mathbf{z}\in C} ||\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu||^2_2.\end{aligned} \tag{5}$$

While $\mathcal{L}_{\text{Var}}$ models the variance of the data, $\mathcal{L}_{\text{K}}$ is simply the k-means loss. Rewriting equation (1) with equations (2) and (5) reads as

$$\frac{\lambda}{2}\left[\sum_{i=1}^{d} \beta_{\text{cs}}[i]^2\mathcal{L}_{\text{K}}[i] + \beta_{\text{ss}}[i]^2\mathcal{L}_{\text{V}}[i]\right] + \sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x} - \operatorname{dec}(\mathbf{V}\mathbf{V}^T\mathbf{z})||^2_2, \tag{6}$$

where $\mathcal{L}_{\text{K}} = \sum_{i=1}^{d} \mathcal{L}_{\text{K}}[i]$ and $\mathcal{L}_{\text{V}} = \sum_{i=1}^{d} \mathcal{L}_{\text{V}}[i]$, with $\mathcal{L}_{\text{K}}[i]$ and $\mathcal{L}_{\text{V}}[i]$ as the k-means loss and variance loss in a single dimension, respectively. We now take the derivative $\frac{\partial}{\partial \mathbf{b}[j]}\mathcal{L}$

$$\begin{aligned}&= \frac{\lambda}{2}\left(\frac{\partial}{\partial\mathbf{b}[j]}\beta_{\text{cs}}[j]^2\mathcal{L}_{\text{K}}[j] + \frac{\partial}{\partial\mathbf{b}[j]}\beta_{\text{ss}}[j]^2\mathcal{L}_{\text{V}}[j]\right) \\ &+ \frac{\partial}{\partial\mathbf{b}[j]}\sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x} - \operatorname{dec}(\mathbf{V}\mathbf{V}^T\mathbf{z})||^2_2 \\ &= \frac{\lambda}{2}\left[\frac{\partial}{\partial\mathbf{b}[j]}\left(\frac{1}{1+\exp(-\mathbf{b}[j])}\right)^2\mathcal{L}_{\text{K}}[j]\right. \\ &\left.+\frac{\partial}{\partial\mathbf{b}[j]}\left(1-\frac{1}{1+\exp(-\mathbf{b}[j])}\right)^2\mathcal{L}_{\text{V}}[j]\right] + 0 \\ &= \frac{\lambda}{2}\left[\frac{2\exp(-\mathbf{b}[j])}{(\exp(-\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{K}}[j] - \frac{2\exp(\mathbf{b}[j])}{(\exp(\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{V}}[j]\right] \\ &= \lambda\left[\underbrace{\frac{\exp(-\mathbf{b}[j])}{(\exp(-\mathbf{b}[j])+1)^3}}_{u}\mathcal{L}_{\text{K}}[j] - \underbrace{\frac{\exp(\mathbf{b}[j])}{(\exp(\mathbf{b}[j])+1)^3}}_{w}\mathcal{L}_{\text{V}}[j]\right].\end{aligned}$$

When this gradient is negative $\mathbf{b}[i]$, and hence $\beta_{cs}[i]$, will increase. This is the case, when $\mathcal{L}_{\text{K}}[i] < \frac{w}{u}\cdot\mathcal{L}_{\text{V}}[i]$. Before the update step, $\mathbf{b}[i]$ is fixed, and so $\frac{w}{u}$ is a positive constant. This means, that $\mathbf{b}[i]$, and hence $\beta_{cs}[i]$, will increase in dimensions, where $\mathcal{L}_{\text{K}}[i] \lesssim \mathcal{L}_{\text{V}}[i]$. This is precisely the case, when the embedded data points are well represented by their respective centroids. Note, that $\frac{w}{u}$ is monotonically decreasing. Thus, with increasing $\mathbf{b}[i]$, the representation with the k-means centroids needs to become more 'convincing' (i.e. $\mathcal{L}_{\text{K}}[i]$ needs to get smaller) in order that $\mathbf{b}[i]$ is further increased and $\beta_{cs}[i]$ is further pushed closer to the cluster space. After optimization these $\beta_{cs}[i]$ will then be close to 1.

This is the upper bound imposed by the asymptotic maximum of the sigmoid function. In the reverse case $\beta_{cs}[j]$ will decrease, but it is lower bounded by the minimum of the loss function with respect to $\mathbf{b}[j]$. We will derive this minimum by setting the derivative $\frac{\partial}{\partial\mathbf{b}[j]}\mathcal{L}$ to zero:

$$\lambda\left[\frac{\exp(-\mathbf{b}[j])}{(\exp(-\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{K}}[j] - \frac{\exp(\mathbf{b}[j])}{(\exp(\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{V}}[j]\right] \overset{!}{=} 0$$

$$\begin{aligned}\frac{\exp(2\mathbf{b}[j])}{(\exp(\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{K}}[j] &= \frac{\exp(\mathbf{b}[j])}{(\exp(\mathbf{b}[j])+1)^3}\mathcal{L}_{\text{V}}[j] \\ \exp(2\mathbf{b}[j])\mathcal{L}_{\text{K}}[j] &= \exp(\mathbf{b}[j])\mathcal{L}_{\text{V}}[j] \\ \frac{\exp(2\mathbf{b}[j])}{\exp(\mathbf{b}[j])} &= \frac{\mathcal{L}_{\text{V}}[j]}{\mathcal{L}_{\text{K}}[j]} \\ 2\mathbf{b}[j] - \mathbf{b}[j] &= \ln(\frac{\mathcal{L}_{\text{V}}[j]}{\mathcal{L}_{\text{K}}[j]}) \\ \mathbf{b}[j]^* &= \ln(\frac{\mathcal{L}_{\text{V}}[j]}{\mathcal{L}_{\text{K}}[j]}) = -\ln(\frac{\mathcal{L}_{\text{K}}[j]}{\mathcal{L}_{\text{V}}[j]})\end{aligned}$$

for $\lambda \neq 0$. Updates that are decreasing $\mathbf{b}[i]$ occur when $\mathcal{L}_{\text{K}}[i] > \mathcal{L}_{\text{V}}[i]$. In this case ACe/DeC will try to map all points in the dimension to a single center $\mu[i]$, including all other centroids $\mu_k[i]$. (The amount of retained variance in this dimension is only influenced by $\mathcal{L}_{\text{rec}}$; see the shared space of Figure 1a). In this situation we will arrive at $\mu[i] = \mu_k[i]$ and, therefore, $\mathcal{L}_{\text{V}}[i] = \mathcal{L}_{\text{K}}[i]$ and $\mathbf{b}^*[i] = 0$. Plugging this back into the sigmoid function we get $\beta^*_{cs}[i] = \text{sigmoid}(0) = 0.5 = \beta^*_{ss}[i]$. We call dimensions where their corresponding weights are close to 0.5 to be *indifferent* between the shared and cluster space. This prevents the degenerate solution of assigning all dimensions to the shared space. Note, that the reverse 'trivial solution' of assigning all dimensions to the cluster space ($\beta_{cs} = \vec{1}, \beta_{ss} = \vec{0}$) is possible, if all dimensions are important for clustering. If there is no cluster structure at all in the data set, all dimensions will be indifferent. Note, that if all β_{ss} are zero and consequently all β_{cs} are one, we arrive at the compression loss of DCN with an additional linear layer $\mathbf{V}$, but the algorithm has now the freedom to learn this trade-off. For simplicity we remove the constant term $\frac{\lambda}{2}$ in Eq. (6) in the following sections and call this version of our loss $\mathcal{L}_{\text{Ours}}$.

1.4 Dimensionality Reduction - β-Hardening

After the weights reached a stable solution, the indifferent weights can be completely assigned to the shared space (i.e., $\beta_{ss}[i] = 1$) for interpretation by including a penalty term. The penalty depends on the set $I = \{i : \beta_{cs}[i] \leq T\}$ of indifferent dimensions w.r.t. a threshold value T, where $T = 0.5 + \varepsilon$, $\varepsilon > 0$, small. The penalty is then defined as $\mathcal{L}_p = \sum_{i\in I} \beta_{cs}[i] + \sum_{j\notin I}^{d} \beta_{ss}[j]$. The ε should account for the random variation due to stochastic gradient descent and we set it to 0.1 for all experiments (ε could also be set automatically, e.g. by the variance of past updates of β_s). Adding the term $\mathcal{L}_p$ to our loss forces the network to slowly push the indifferent β_{ss} into the shared space and the β_{cs} are stronger assigned to the cluster space. We show in Section 2.2 experimental results regarding the parameter T.

1.5 Linear Transformation V

In this section, we take a closer look at the derivative of $\mathcal{L}_{\text{Ours}}$ with respect to the linear transformation matrix $\mathbf{V}$. Throughout this section, we will use denominator convention for the differentiation of a scalar with respect to a matrix. Since the update of $\mathbf{V}$ is independent of the effect of $\mathbf{V}$ on the centroids μ_k and μ, we have to consider the following loss term, where $\mathcal{X}$ is the set of all data points and C_k is the set of all data points $\mathbf{x}$ assigned to the k^{th} cluster, i.e, $C_k = \{\mathbf{x} \in \mathcal{X} | \text{argmin}(||\mathbf{V}\mathbf{x} - \mathbf{V}\mu_k||^2_{\beta_{cs}})\}$:

$$\begin{aligned}
\mathcal{L}_{\text{Ours}} = & \underbrace{\sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} ||\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}}||^2_{\beta_{cs}}}_{\mathcal{L}_{\text{cluster}}} \\
& + \underbrace{\sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}^T\mathbf{x} - \mu||^2_{\beta_{ss}}}_{\mathcal{L}_{\text{shared}}} \\
& + \underbrace{\sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2}_{\mathcal{L}_{\text{rec}}}
\end{aligned}$$

Let us look at the three terms individually. We start with $\mathcal{L}_{\text{cluster}}$ and rewrite the β-weighted Euclidean norm as a vector multiplication and diagonal matrix multiplication with the squares of the entries of the weight vector β_{cs} in the diagonal. For simplicity of notation, we will define $D = \text{diag}(\beta^2_{cs})$.

$$\mathcal{L}_{\text{cluster}} = \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} (\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})^T D (\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})$$

By use of the formula $\frac{\partial}{\partial X^T}((X\mathbf{a}+\mathbf{b})^T C(X\mathbf{a}+\mathbf{b})) = ((C + C^T)(X\mathbf{a} + \mathbf{b})\mathbf{a}^T)^T$ and the identities $X = \mathbf{V}^T$, $\mathbf{a} = \mathbf{x}$, $\mathbf{b} = -\mu_{\mathbf{k}}$ and $C = D$, the derivative is given by the following expression: (Note, that $D^T = D$)

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{V}}\mathcal{L}_{\text{cluster}} &= \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} \frac{\partial}{\partial \mathbf{V}} (\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})^T D (\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}}) \\
&= \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} (2D(\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})\mathbf{x}^T)^T \\
&= \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} 2\mathbf{x}(\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})^T D \qquad (7)
\end{aligned}$$

We get the same for $\mathcal{L}_{\text{shared}}$, except that $D = \text{diag}(\beta^2_{ss})$ and $\mu_k = \mu$.

$$\frac{\partial}{\partial \mathbf{V}}\mathcal{L}_{\text{shared}} = \sum_{\mathbf{x}\in\mathcal{X}} 2\mathbf{x}(\mathbf{V}^T\mathbf{x} - \mu)^T D \qquad (8)$$

For the reconstruction loss, we get

$$\begin{aligned}
\mathcal{L}_{\text{rec}} &= \sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2 \\
&= \sum_{\mathbf{x}\in\mathcal{X}} (\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x})^T(\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}) \\
&= \sum_{\mathbf{x}\in\mathcal{X}} \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} + \mathbf{x}^T\mathbf{x}
\end{aligned}$$

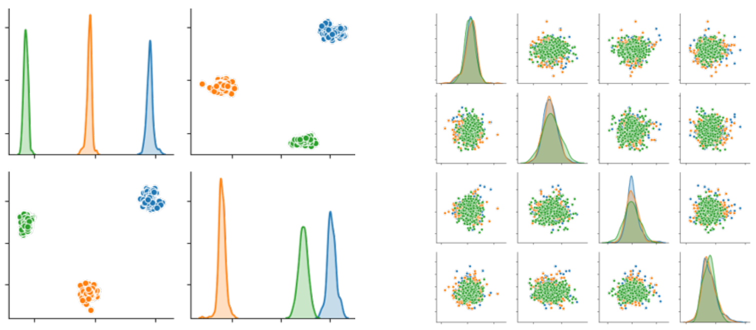

(a) Cluster space (b) Shared space

Figure 2: Scatter plot matrices for our method applied to OBJECTS for an autoencoder with a six dimensional embedded space (colors represent ground truth labels). The cluster space only needs two dimensions to represent the cluster structure (Figure 2a), while the remaining dimensions in the shared space (Figure 2b) do not contain any cluster structure (mixed colors). Note, that all dimensions in the shared space are unimodal Gaussians (diagonal in Figure 2b).

The derivative is then:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{V}}\mathcal{L}_{\text{rec}} = \sum_{\mathbf{x}\in\mathcal{X}} & \underbrace{\frac{\partial}{\partial \mathbf{V}}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x}}_{I} \\ & -2\underbrace{\frac{\partial}{\partial \mathbf{V}}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}}_{II} \\ & +\underbrace{\frac{\partial}{\partial \mathbf{V}}\mathbf{x}^T\mathbf{x}}_{III=\mathbf{0}}\end{aligned}$$

I) First, we can rewrite the expression as

$$\begin{aligned}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x} &= tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) \\ &= tr(\mathbf{V}\mathbf{V}^T\mathbf{x}\mathbf{x}^T\mathbf{V}\mathbf{V}^T) \\ &= tr(\mathbf{V}\mathbf{V}^T X\mathbf{V}\mathbf{V}^T),\end{aligned}$$

with $\mathbf{x}\mathbf{x}^T =: X = X^T$, i.e. , $X \in \mathbb{R}^{D\times D}$ is a symmetric matrix. We then get

$$\begin{aligned} & d(tr(\mathbf{V}\mathbf{V}^T X\mathbf{V}\mathbf{V}^T)) \\ &= tr(d(\mathbf{V}\mathbf{V}^T X\mathbf{V}\mathbf{V}^T)) \\ &= tr(d(\mathbf{V}\mathbf{V}^T)X\mathbf{V}\mathbf{V}^T) + tr(\mathbf{V}\mathbf{V}^T d(X\mathbf{V}\mathbf{V}^T)) \\ &= tr([d(\mathbf{V})\mathbf{V}^T + \mathbf{V}d(\mathbf{V}^T)]X\mathbf{V}\mathbf{V}^T) \\ &\quad + tr(\mathbf{V}\mathbf{V}^T[d(X\mathbf{V})\mathbf{V}^T + X\mathbf{V}d(\mathbf{V}^T)]) \\ &= tr(d(\mathbf{V})\mathbf{V}^T X\mathbf{V}\mathbf{V}^T) + tr(\mathbf{V}\underbrace{d(\mathbf{V}^T)}_{=d(\mathbf{V})^T}X\mathbf{V}\mathbf{V}^T) \\ &\quad + tr(\mathbf{V}\mathbf{V}^T\underbrace{d(X\mathbf{V})}_{=Xd(\mathbf{V})}\mathbf{V}^T) + tr(\mathbf{V}\mathbf{V}^T X\mathbf{V}\underbrace{d(\mathbf{V}^T)}_{d(\mathbf{V})^T}) \\ &= tr(\mathbf{V}^T X\mathbf{V}\mathbf{V}^T d(\mathbf{V})) + \underbrace{tr(X\mathbf{V}\mathbf{V}^T\mathbf{V}d(\mathbf{V})^T)}_{\substack{=tr(d(\mathbf{V})\mathbf{V}^T\mathbf{V}\mathbf{V}^T X^T) \\ =tr(\mathbf{V}^T\mathbf{V}\mathbf{V}^T X^T d(\mathbf{V}))}} \\ &\quad + tr(\mathbf{V}^T\mathbf{V}\mathbf{V}^T X d(\mathbf{V})) + \underbrace{tr(\mathbf{V}\mathbf{V}^T X\mathbf{V}d(\mathbf{V})^T)}_{\substack{=tr(d(\mathbf{V})\mathbf{V}^T X^T\mathbf{V}\mathbf{V}^T) \\ =tr(\mathbf{V}^T X^T\mathbf{V}\mathbf{V}^T d(\mathbf{V}))}} \\ &= tr([\mathbf{V}^T X\mathbf{V}\mathbf{V}^T + \mathbf{V}^T\mathbf{V}\mathbf{V}^T X^T \\ &\quad + \mathbf{V}^T\mathbf{V}\mathbf{V}^T X + \mathbf{V}^T X^T\mathbf{V}\mathbf{V}^T]d(\mathbf{V}))\end{aligned}$$

Now it follows from $y = tr(Xd(\mathbf{V})) \Rightarrow \frac{dy}{d\mathbf{V}} = X^T$, where X is independent from $\mathbf{V}$, that

$$\begin{aligned}&\tfrac{\partial}{\partial \mathbf{V}} tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) \\ &\quad = \mathbf{V}\mathbf{V}^T X^T\mathbf{V} + X\mathbf{V}\mathbf{V}^T\mathbf{V} + X^T\mathbf{V}\mathbf{V}^T\mathbf{V} + \mathbf{V}\mathbf{V}^T X\mathbf{V}\end{aligned}$$

and since $X = X^T$

$$\begin{aligned}&= 2\mathbf{V}\mathbf{V}^T X\mathbf{V} + 2X\mathbf{V}\mathbf{V}^T\mathbf{V} \\ &= 2(\mathbf{V}\mathbf{V}^T\mathbf{x}\mathbf{x}^T\mathbf{V} + \mathbf{x}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}).\end{aligned}$$

Summarising, we get that

$$\begin{aligned}\frac{\partial}{\partial \mathbf{V}}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x} &= \frac{\partial}{\partial \mathbf{V}} tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) \qquad (9) \\ &= 2(\mathbf{V}\mathbf{V}^T\mathbf{x}\mathbf{x}^T\mathbf{V} + \mathbf{x}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V})\end{aligned}$$

II) For the second term we proceed in the same way

$$\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} = tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) = tr(\mathbf{V}^T X\mathbf{V})$$

Then we get

$$\begin{aligned} d(tr(\mathbf{V}^T X\mathbf{V}^T)) &= tr(d(\mathbf{V})^T X\mathbf{V}^T + \mathbf{V}^T d(X\mathbf{V})) \\ &= tr(\mathbf{V}X^T d(\mathbf{V})) + tr(\mathbf{V}^T X d(\mathbf{V})) \\ &= tr([\mathbf{V}X^T + \mathbf{V}^T X]d(\mathbf{V})) \end{aligned}$$

As before we then get

$$\frac{\partial}{\partial\mathbf{V}} tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) = X\mathbf{V} + X^T\mathbf{V} = 2X\mathbf{V}$$

which leads to

$$\frac{\partial}{\partial\mathbf{V}}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} = \frac{\partial}{\partial\mathbf{V}} tr(\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}) = 2\mathbf{x}\mathbf{x}^T\mathbf{V} \quad (10)$$

Term III) vanishes, since it is independent of $\mathbf{V}$. With formulas (9) and (10), we finally get for the reconstruction loss

$$\frac{\partial}{\partial\mathbf{V}}\mathcal{L}_{\text{rec}} = \sum_{\mathbf{x}\in\mathcal{X}} 2(\mathbf{V}\mathbf{V}^T\mathbf{x}\mathbf{x}^T\mathbf{V} + \mathbf{x}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}) - 4\mathbf{x}\mathbf{x}^T\mathbf{V}$$

Note, that in the case of an orthogonal matrix, i.e., $\mathbf{V}\mathbf{V}^T = \mathbf{I}_D$, all addends vanish and, therefore, the derivative is $\mathbf{0}$, which means that we reach a minimum of this loss term. (Note, that the reconstruction loss is convex, which means, we cannot arrive at a maximum.) This is in accordance with the fact that the reconstruction loss is minimal, when it is zero, which is precisely the case, when $\mathbf{V}\mathbf{V}^T = \mathbf{I}_D$.
To sum up, we get:

$$\begin{aligned} \frac{\partial}{\partial\mathbf{V}}\mathcal{L}_{\text{Ours}} &= \frac{\partial}{\partial\mathbf{V}}\mathcal{L}_{\text{cluster}} + \frac{\partial}{\partial\mathbf{V}}\mathcal{L}_{\text{shared}} + \frac{\partial}{\partial\mathbf{V}}L_{\text{rec}} \\ &= \sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} 2\mathbf{x}(\mathbf{V}^T\mathbf{x} - \mu_{\mathbf{k}})^T D \\ &+ \sum_{\mathbf{x}\in\mathcal{X}} 2\mathbf{x}(\mathbf{V}^T\mathbf{x} - \mu)^T D \\ &+ \sum_{\mathbf{x}\in\mathcal{X}} 2(\mathbf{V}\mathbf{V}^T\mathbf{x}\mathbf{x}^T\mathbf{V} + \mathbf{x}\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{V}) - 4\mathbf{x}\mathbf{x}^T\mathbf{V} \end{aligned}$$

1.6 ACe/DeC +DCN and SubKmeans

In the following, we will derive that if we exclude the non-linearity of ACe/DeC, and further assume hard assignments for the β-weights ($\beta_s = \beta_s^2,\ s \in \{cs, ss\}$), we effectively solve the same problem as SubKmeans [Mautz *et al.*, 2017]. Thereby we denote with a center dot $\cdot$ the Euclidean scalar vector product, while no dot is—depending on the objects—either a matrix-matrix or a matrix-vector multiplication.
We first have a look at the loss function of SubKmeans:

$$\begin{aligned} \mathcal{L}_{SUBK} &= \sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} ||\bar{P}_C^T(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_{\mathbf{k}})||^2 \\ &+ \sum_{\mathbf{x}\in\mathcal{X}} ||\bar{P}_N^T(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)||^2 \end{aligned} \quad (11)$$

subject to

$$\mathbf{V}^T\mathbf{V} = \mathbf{I}. \quad (12)$$

The matrices

$$\bar{P}_C = \begin{pmatrix} \mathbf{I}_m \\ \mathbf{0}_{d-m,m} \end{pmatrix} \quad \bar{P}_N = \begin{pmatrix} \mathbf{0}_{m,d-m} \\ \mathbf{I}_{d-m} \end{pmatrix}$$

describe the (linear) projections on the m dimensions of the *cluster space* and the $d - m$ dimensions of the *noise space*. $\mathbf{I}_m$ is the m dimensional identity matrix and $\mathbf{0}_{k,l}$ is a zero matrix with k rows and l columns. Without loss of generality, the dimensions are ordered, such that the first m dimensions belong to the cluster space. Note, that we can extend these matrices to squared $d \times d$ matrices by simply adding zeros columns, without changing the meaning—and value—of the loss. More specifically, with

$$P_C = \begin{pmatrix} \mathbf{I}_m & \mathbf{O}_{m,d-m} \\ \mathbf{0}_{d-m,m} & \mathbf{O}_{d-m,d-m} \end{pmatrix}$$

$$P_N = \begin{pmatrix} \mathbf{O}_{m,d-m} & \mathbf{0}_{m,d-m} \\ \mathbf{O}_{d-m,d-m} & \mathbf{I}_{d-m} \end{pmatrix}$$

the SubKmeans loss can be written as

$$\begin{aligned} \mathcal{L}_{SUBK} &= \underbrace{\sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} ||P_C(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_{\mathbf{k}})||^2}_{\mathcal{L}_C} \\ &+ \underbrace{\sum_{\mathbf{x}\in\mathcal{X}} ||P_N(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)||^2}_{\mathcal{L}_N}, \end{aligned} \quad (13)$$

with the respective losses $\mathcal{L}_C$ and $\mathcal{L}_N$ for cluster and noise space.
Next, we consider the loss function of ACe/DeC + DCN omitting normalisation factors and the non-linearities:

$$\begin{aligned} &\mathcal{L}_{Ours} \\ &= \underbrace{\sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} ||\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_{\mathbf{k}}||_{\beta_{cs}}^2 + \sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu||_{\beta_{ss}}^2}_{\mathcal{L}_{comp}} \\ &+ \underbrace{\sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2}_{\mathcal{L}_{rec}}. \end{aligned} \quad (14)$$

We can then rewrite the β-weighted Euclidean norm as a vector multiplication and a point wise multiplication $\otimes$ with the weight vector

$$\begin{aligned} &= \sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} (\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_{\mathbf{k}})^T \cdot (\beta_{cs} \otimes (\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_{\mathbf{k}})) \\ &+ \sum_{\mathbf{x}\in\mathcal{X}} \left(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu\right)^T \cdot (\beta_{ss} \otimes (\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)\right) \\ &+ \sum_{\mathbf{x}\in\mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2 \end{aligned}$$

We can further rewrite the point wise multiplication as a matrix multiplication with the β_s weights in the diagonal

$$= \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} (\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_\mathbf{k})^T \cdot (\text{diag}(\beta_{cs})(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_\mathbf{k})) + \sum_{\mathbf{x} \in \mathcal{X}} (\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)^T \cdot (\text{diag}(\beta_{ss})(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)) + \sum_{\mathbf{x} \in \mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2$$

Rewriting the scalar product of vectors as Euclidean norms again yields

$$= \underbrace{\sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} ||\text{diag}(\beta_{cs})(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu_\mathbf{k})||^2}_{\hat{=}\mathcal{L}_C} + \underbrace{\sum_{\mathbf{x} \in \mathcal{X}} ||\text{diag}(\beta_{ss})(\mathbf{V}^T\mathbf{x} - \mathbf{V}^T\mu)||^2}_{\hat{=}\mathcal{L}_N} + \sum_{\mathbf{x} \in \mathcal{X}} ||\mathbf{V}\mathbf{V}^T\mathbf{x} - \mathbf{x}||^2. \tag{15}$$

Assuming hard assignments $\{0, 1\}$ for the β weights, the diagonal matrices diag(β_{cs}) and diag(β_{ss}) are effectively the linear projection matrices P_C and P_N onto the cluster and noise space, respectively. Finally, the third sum approximates the condition given by equation (12).

1.7 Clustering Procedure

In contrast to the original DCN [Yang *et al.*, 2017] clustering procedure, we proposed an adapted version that is based on mini-batch k-means [Sculley, 2010]. The original DCN algorithm uses an alternating update, which holds either the autoencoder fixed and updates the centroids or fixes the centroids and updates the autoencoder parameters. We implemented a mini-batch strategy, which allows an update without alternating between full updates, similar to [Miklautz *et al.*, 2020]. The assignment of objects to their given cluster centers in the cluster space is done by assigning them to their closest center, e.g., when integrated in our ACe/DeC framework we compute the β_{cs}-weighted euclidean distance and $\arg\min_{k \in [1;K]} ||\mathbf{V}^T\mathbf{z} - \mathbf{V}^T\mu_k||^2_{\beta_{cs}}$ (For the shared space, we have only a single center to which all points are assigned). We update the cluster centroids by calculating a running mean for each centroid per mini-batch. We weight the updates with a per cluster learning rate $\alpha = \frac{1}{a_k}$ that is calculated with a discounted weighted average over the number of past assignments for a cluster, where we set the discounting parameter to 0.5 for all experiments. With this, we get the following centroid update rule $\mu_k = (1-\alpha)\mu_k^{t-1} + \alpha \frac{\sum_{z \in C_k^{MB}} \mathbf{V}^T\mathbf{z}}{|C_k^{MB}|}$, where $|C_k^{MB}|$ is the number of samples in a mini-batch assigned to the k-th cluster. Similar to [Yang *et al.*, 2017], we detect for our DCN and its integration with ACe/DeC whether a cluster has lost all its points. If this is the case we reassign its centroid to a random point with a small perturbation. After a cluster was reinitialized, we reset the past assignment count. This reinitialization makes DCN much more stable and helps it recover from lost clusters.

2 Experiments

This section contains additional information about the experiments in the main paper including further qualitative (Section 2.1) and quantitative experiments (Section 2.2), the used data sets (Section 2.3), the experiment setup (Section 2.4) and the Hard- and Software setup (Section 2.5).

2.1 Qualitative Experiments

In addition to the experiments in the main paper, we isolated the cluster information for MNIST in Figure 1b and conducted further latent space traversals for MNIST in Figure 3 and OBJECTS in Figure 4. We perform the latent space traversal along the first three principal components of the shared space. The reasoning behind this, is that PCA [Pearson, 1901] applied to a multivariate Gaussian will make the components asymptotically independent. Thus, we can visualize the independent effects of the shared space, which is an approximation of a multivariate Gaussian (Figure 2b). In the case of MNIST in Figure 3 and OBJECTS in Figure 4, these effects correspond to human interpretable concepts. While our main focus is to separate cluster structure from non-cluster structure, it is interesting to see that the non-cluster structure can be further decomposed and interpreted.

We created the cluster space reconstructions of a single data point with $\hat{\mathbf{x}}_{cs} = \text{dec}(\mathbf{V}\mathbf{V}^\mathbf{T}(\mathbf{z} \odot \beta_{cs} + \mu_k \odot \beta_{ss}))$ and for the shared space $\hat{\mathbf{x}}_{ss} = \text{dec}(\mathbf{V}\mathbf{V}^\mathbf{T}(\mathbf{z} \odot \beta_{ss} + \mu_k \odot \beta_{cs}))$, where μ_k is the closest centroid of $\mathbf{z}$ and $\odot$ is the element wise product. This can be interpreted as applying the cluster or shared style of an embedded object to the centroid.

2.2 Quantitative Experiments

Evaluation Metric

To quantify the quality of the found clustering, we use the normalized mutual information (NMI) [Vinh *et al.*, 2010]. This measure ranges from 0 to 1, where a value close to 0 implies that almost no structure was found, while a value of 1 indicates a perfect clustering. The NMI measures the mutual information between ground-truth labels and cluster assignments normalized by both their entropies. Note, that a random clustering leads to an NMI of 0.

Dimensionality Reduction

We show in Table 3 the average dimensionality reduction for different threshold values and the corresponding NMI values after reduction in Table 2. For hardening β_s, we added the penalty $\mathcal{L}_p = \sum_{i \in I} \beta_{cs}[i] + \sum_{j \notin I}^{d} \beta_{ss}[j]$, where $I = \{i : \beta_{cs}[i] \leq T\}$, with $T = 0.5 + \varepsilon$, $\varepsilon > 0$, small. The additional term $\mathcal{L}_p$ forces the network to push the indifferent β_{ss} into the shared space and the β_{cs} to the cluster space, effectively separating the two subspaces. We trained ACe/DeC +DCN for additional 20,000 mini-batch iterations for all results in Table 2. This can also be automated by stopping training after β_s have converged.

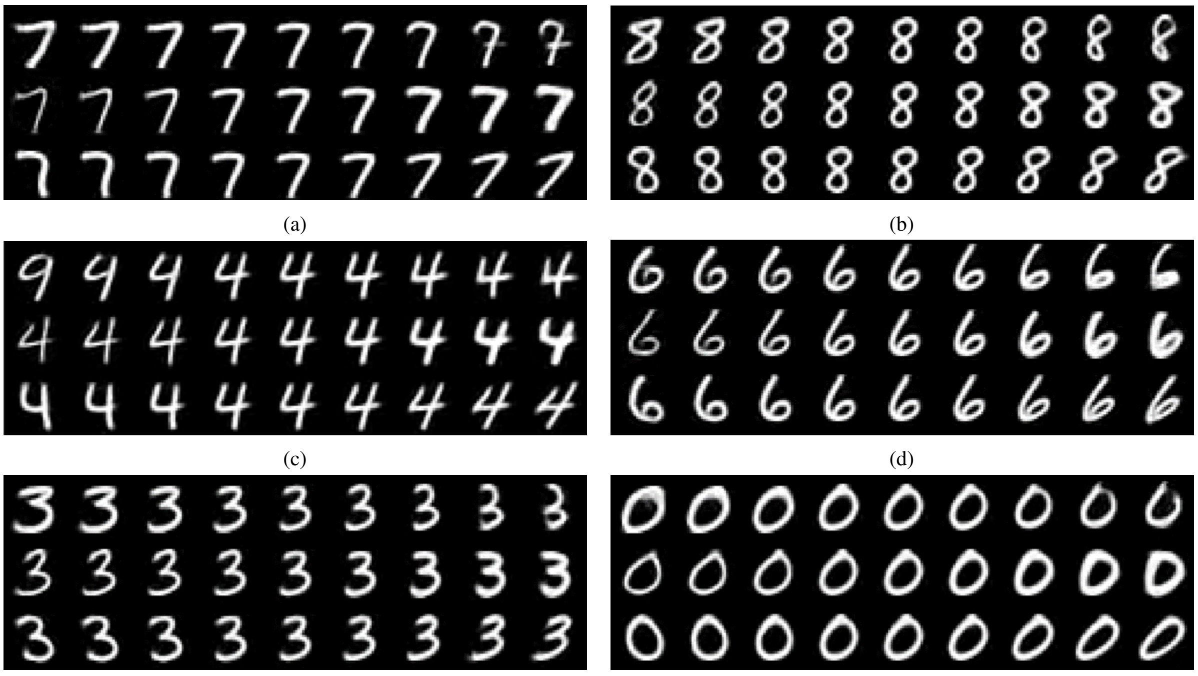

(a) (b) (c) (d) (e) (f)

Figure 3: The latent space traversal of ACe/DeC +DCN along the first three principal components of the shared space varies different properties of MNIST digits, while the cluster identities remain unchanged. Each row of each cluster $(0, 3, 4, 6, 7, 8)$ corresponds to one principal component, e.g., PC 1 is the first row, PC 2 the second and PC 3 the third. In Figure 3a, we can see that the first row (PC 1) changes the style of the digit seven (horizontal bar and serif). The second row (PC 2) changes the thickness and (PC 3) the orientation aka azimuth of the seven (left leaning vs. straight line). Another example is the digit four in Figure 3c. Here we see again that the first row (PC 1) changes the style between a four that is closed (left side) and one that is open (right) side. The second and third rows are again changing the thickness and orientation of the four. These effects transfer to the other digits as well.

Name	# Points	# Dimensions	# Clusters
MNIST	70,000	784	10
FMNIST	70,000	784	10
USPS	9,298	256	10
GTSRB	15,540	3,072	10
OBJECTS	10,000	4,096	3
SYNTH-25	12,000	27	4

Table 1: Summary of used data sets.

2.3 Data Sets

We reported our results on six different data sets. We focused on common deep clustering benchmarks like MNIST [LeCun *et al.*, 1998], Fashion-MNIST [Xiao *et al.*, 2017] and USPS. We describe these three data sets in more detail below. Additionally, as explained in the paper, we used two synthetically generated data sets called OBJECTS and SYNTH-25 and the German Traffic Sign Recognition Benchmark (GTSRB). A summary of the used data sets can be found in Table 1.

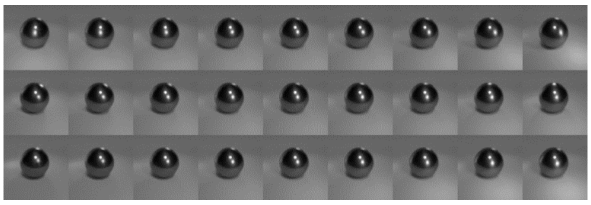
MNIST: The MNIST data set contains $70,000$ images of handwritten digits, containing ten clusters from 0 to 9. In the deep clustering literature it has become common to use two variations of MNIST splits for evaluation. MNIST-Test refers to a train/test split, where the models have been trained on the $60,000$ training images of MNIST and then evaluated on the remaining $10,000$ test images. MNIST-Full means that we train and test on the full data set ($70,000$ images) and report those results.

Fashion-MNIST: Fashion-MNIST (FMNIST) consists of $70,000$ images of different fashion items, containing the ten clusters T-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot.

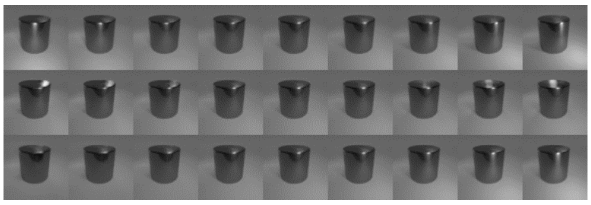
USPS: The USPS data set contains images of handwritten digits similar to MNIST, but it is more challenging due to the lower size of only $9,298$ samples.

GTSRB: The German Traffic Sign Recognition Benchmark (GTSRB) data set contains different images of traffic signs under varying light conditions and viewing-angles. We use a subset of ten classes consisting of $15,540$ data points.

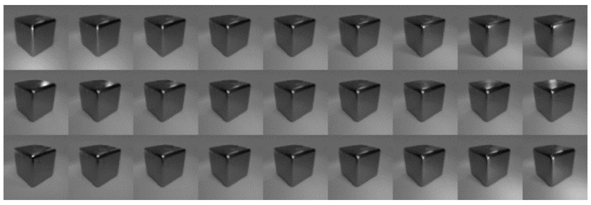
OBJECTS: For the OBJECTS data set, we generated $10,000$ gray scale 3D objects with three different shapes (cube, sphere and cylinder) using the publicly available rendering



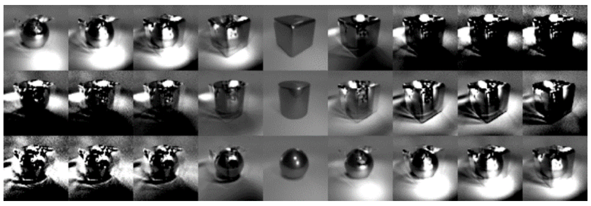
(a)



(b)



(c)



(d)

Figure 4: **(a-c)** The latent space traversal of ACe/DeC +DCN along the first three principal components of the shared space varies different light properties of OBJECTS, while the cluster identities (sphere, cylinder and cube) remain unchanged. Each row of each image block corresponds to one principal component (PC). In Figure 4a, we see that the first row (PC 1) changes the front light source from left to right. The second row (PC 2) changes the back light source from left to right and the third row (PC 3) changes the front light source up and down. Note, that on the left side the light source is weaker. This matches with how the data was generated (see the shared space centroid in Figure 1a, where most front light comes from the lower right corner). The same can be seen in Figures 4b and 4c. **(d)** The latent space traversal for DCN without ACe/DeC along the first principal component yields results that mix several light sources and distorts the three centroids.

	MNIST-Full	**FMNIST**	**USPS**	**GTSRB**	**OBJECTS**
Soft	0.89 $\pm$0.01	0.62 $\pm$0.00	0.71 $\pm$0.02	0.52 $\pm$0.04	1.00 $\pm$0.00
$T = 0.6$	0.88 $\pm$0.01	0.62 $\pm$0.01	0.68 $\pm$0.03	0.50 $\pm$0.05	1.00 $\pm$0.00
$T = 0.7$	0.87 $\pm$0.01	0.60 $\pm$0.02	0.64 $\pm$0.03	0.42 $\pm$0.10	1.00 $\pm$0.00
$T = 0.8$	0.80 $\pm$0.04	0.55 $\pm$0.03	0.34 $\pm$0.03	0.25 $\pm$0.13	1.00 $\pm$0.00

Table 2: Average NMIs (mean $\pm$ standard deviation) before (Soft Assignment) and after the dimensions with indifferent β_s have been assigned to the shared space at the threshold levels $T \in \{0.6, 0.7, 0.8\}$. The values of T have been selected according to Table 3, where at least a single dimension on average should be kept for each data set. First, we can observe that there is no significant change in clustering performance for $T = 0.6$, indicating that the dimensions with indifferent β_s close to 0.5 are indeed not important for clustering. As we increase the threshold we can see that the cluster performance drops, because we are removing dimensions that are important for clustering from the cluster space. We used here the same learning rate for each T. In practice, one might want to use lower learning rates for higher T to avoid disruption of the learned embedding. Note, that one can tune the learning rate or the optimal dimensionality based on the change from the initial cluster label distribution and the reduced ones for different thresholds T.

T	**MNIST-F**	**FMNIST**	**USPS**	**GTSRB**	**OBJ**
0.5	10	10	10	10	10
0.6	6	5	5	4	2
0.7	5	4	4	3	2
0.8	4	2	1	1	2
0.9	1	2	0	0	2

Table 3: Average number of dimensions found for different thresholds $T \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. Abbreviations: OBJ = OBJECTS, MNIST-F = MNIST-Full

software used in [Johnson *et al.*, 2017][1]. Each object has some additional random lighting jitter coming from three directions and has 4,096 dimensions in the pixel space. The OBJECTS data set can be found in the online folder.

SYNTH-25: For the SYNTH-25 data set, we generate $12,000$ data points that form four well separated spherical clusters. Additionally, we added 25 independent, unimodal Gaussian distributions that contain no information about the four clusters. The code for generating this data set can be found in the code folder.

We pre-process all image data sets by scaling the pixels from $[0, 255]$ to $[0, 1]$ and applying a z-transformation. For the GTSRB images we used additionally histogram equalization, due to the low contrast in most of the images. The SYNTH-25 data set was only z-transformed. The preprocessing scripts can be found in our supplementary file.

2.4 Experiment Setup

For all our experiments with image data, we use the same stacked denoising autoencoder [Vincent *et al.*, 2010] architecture as in [Xie *et al.*, 2016; Guo *et al.*, 2017; Yang *et al.*, 2017], except for the activation function. We use the Leaky ReLu [Maas *et al.*, 2013] instead of the ReLU function, which resulted in better performances for all methods. The used architecture is designed as a feed forward autoencoder with dimensionalities $\mathrm{D} - 500 - 500 - 2000 - \mathrm{d}$ and a mirrored decoder, were D is the data dimensionality and d is the embedding size. We set $\mathrm{d} = 10$ for all image experiments, which is equal to the maximum number of clusters $\mathrm{k} = 10$ for all considered data sets. For each image data set we pretrain

[1]https://github.com/facebookresearch/clevr-dataset-gen

ten stacked denoising autoencoders and use them for all algorithms. For the augmentation procedure we used random rotations (20 degrees) and random shifting. During the pretraining for the augmented version we applied the random augmentations with a probability of 0.5 to the input data.
Note, that we do not focus here on current state of the art architectural improvements for autoencoders for deep clustering, as we expect that all approaches benefit from those, which is why we use the fully connected autoencoders. We use $25,000$ mini-batch iterations per layer resulting in $100,000$ mini-batch iterations in total for the stacked layer-wise pretraining. Similar to [Xie *et al.*, 2016] we use dropout [Srivastava *et al.*, 2014] with a rate of 0.2 and a noise level of 0.2 during pretraining. The fine-tuning scheme is then performed on the full autoencoder for additional $50,000$ mini-batch iterations without dropout and noise. For all data sets we set the batch size to 256 and for optimization we use the Adam optimizer (pretraining-lr $= 0.001, m_1 = 0.9, m_2 = 0.99$) without weight decay [Kingma and Ba, 2015]. For pretraining and fine-tuning we set the learning rate to pretraining-lr $= 0.0001$. We initialize our algorithm by running 20 times the initialization procedure and choose the one with the smallest cost. For DCN, DEC and IDEC we initialize their centers with 20 runs of k-means using again the run with the smallest k-means loss. We use additional $100,000$ mini-batch iterations for the joint clustering procedure. Similar to [Xie *et al.*, 2016], we start for all methods with a higher learning rate than was used during pretraining, namely 0.001. To ensure stable convergence we reduce the learning rate by a factor of 2 every $20,000$ iterations, which equals the learning rate schedule $\gamma = 0.5$, we compared other schedules in the main paper. K-means and SubKmeans are run ten times on each of the ten autoencoder embeddings, resulting in 100 runs per data set. We selected for each of the ten runs the best one in terms of their respective objective function and average them. For our algorithm the β_s act as a 'guide' or scaling factor for $\mathbf{V}$ and therefore we update them ten times faster than $\mathbf{V}$ in all experiments. Results marked with $*$ where run on a subset of $10,000$ samples and empty results where stopped due to run time constraints. For the SYNTH-25 data set we used a fully connected autoencoder with a single hidden layer and no dimensionality reduction (because all Gaussian dimensions are independent) resulting in an autoencoder with dimensionalities $27 - 27 - 27$. For the synthetic data we ran all methods twenty times to get better estimates for the mean, because the standard deviations were quite high for some methods.

2.5 Hard- and Software Setup

We implemented all experiments in Python, using Pytorch [Paszke *et al.*, 2019] for all deep learning based methods. We used scikit-learn [Pedregosa *et al.*, 2011] and SciPy [Virtanen *et al.*, 2019] for implementing k-means and SubKmeans. We trained all deep clustering algorithms on a machine with a single NVIDIA RTX 2080 TI GPU (11 GB on-board memory), 96 GB RAM and an Intel(R) Xeon(R) Gold 6130 CPU. K-means and SubKmeans were run on the same machine, but on the CPU. The code can be found in the supplementary files. A single run for DCN took about 90 minutes and for DCN with our ACe/DeC framework and mini-batch update about 60 minutes (Note, we always trained for the same minibatch iterations and not epochs for each data set).

References

[Guo *et al.*, 2017] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pages 1753–1759. ijcai.org, 2017.

[Johnson *et al.*, 2017] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE CVPR, 2017*, pages 2901–2910, 2017.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[Le *et al.*, 2011] Quoc V Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Y Ng. Ica with reconstruction cost for efficient overcomplete feature learning. In *Advances in NIPS, 2011*, pages 1017–1025, 2011.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Maas *et al.*, 2013] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, volume 30, page 3, 2013.

[Mautz *et al.*, 2017] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Towards an optimal subspace for k-means. In *KDD*, pages 365–373. ACM, 2017.

[Miklautz *et al.*, 2020] Lukas Miklautz, Dominik Mautz, C Altinigneli, Christian Böhm, and Claudia Plant. Deep embedded non-redundant clustering. AAAI, 2020.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[Pearson, 1901] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and

E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Sculley, 2010] D. Sculley. Web-scale k-means clustering. In *WWW*, pages 1177–1178. ACM, 2010.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[Vincent *et al.*, 2010] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[Vinh *et al.*, 2010] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 10 2010.

[Virtanen *et al.*, 2019] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, 2019.

[Xiao *et al.*, 2017] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[Xie *et al.*, 2016] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 478–487. JMLR.org, 2016.

[Yang *et al.*, 2017] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*, volume 70, pages 3861–3870. PMLR, 2017.

A.3. Deep Clustering With Consensus Representations

Title	Deep Clustering With Consensus Representations
Authors	Lukas Miklautz, Martin Teuffenbach, Pascal Weber, Rona Perjuci, Walid Durani, Christian Böhm and Claudia Plant
Publication Outlet	Proceedings of the 22nd IEEE International Conference on Data Mining (ICDM), 2022. Note: CORE ranking: A*; Acceptance rate: 20%
DOI-URL	`https://doi.org/10.1109/ICDM54844.2022.00141`

Division of Work Lukas Miklautz: conceiving the initial idea; problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of variations of the method; experimental design; execution of experiments; writing and visualization of major parts of the article review of relevant related work; creating the conference poster and presentation.

Martin Teuffenbach: problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of comparison methods from consensus clustering; experimental design; execution of experiments; writing and visualization of parts of the article; review of relevant related work.

Pascal Weber: problem identification, definition and refinement; formulating the loss function and optimization strategy; implementation and testing of comparison methods from consensus clustering; experimental design; execution of experiments; writing and visualization of parts of the article; review of relevant related work.

Rona Perjuci: implementation and testing of comparison methods from deep clustering; experimental design; execution of experiments; writing of parts of the article; review of relevant related work.

Walid Durani: implementation and testing of comparison methods from deep clustering; experimental design; execution of experiments; writing of parts of the article; review of relevant related work.

Christian Böhm: problem identification; regular discussions of candidate methods and potential experiments; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Claudia Plant: regular discussions of candidate methods and potential experiments; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Abstract The field of deep clustering combines deep learning and clustering to learn representations that improve both the learned representation and the performance of the considered clustering method. Most existing deep clustering methods are designed for a single clustering method, e.g., k-means, spectral clustering, or Gaussian mixture models, but it is well known that no clustering algorithm works best in all circumstances. Consensus clustering tries to alleviate the individual weaknesses of clustering algorithms by building a consensus between members of a clustering ensemble. Currently, there is no deep clustering method that can include multiple heterogeneous clustering algorithms in an ensemble to update representations and clusterings together. To close this gap, we introduce the idea of a consensus representation that maximizes the agreement between ensemble members. Further, we propose DECCS (Deep Embedded Clustering with Consensus representationS), a deep consensus clustering method that learns a consensus representation by enhancing the embedded space to such a degree that all ensemble members agree on a common clustering result. Our contributions are the following: (1) We introduce the idea of learning consensus representations for heterogeneous clusterings, a novel notion to approach consensus clustering. (2) We propose DECCS, the first deep clustering method that jointly improves the representation and clustering results of multiple heterogeneous clustering algorithms. (3) We show in experiments that learning a consensus representation with DECCS is outperforming several relevant baselines from deep clustering and consensus clustering.

Thesis-Reference [MTW+22a]

Note We show here the long version of the paper. For the short paper version, see [MTW+22b].

Deep Clustering With Consensus Representations

Lukas Miklautz[1,2], Martin Teuffenbach[1,2], Pascal Weber[1,2], Rona Perjuci[1],
Walid Durani[3], Christian Böhm[1], Claudia Plant[1,4]
[1]*Faculty of Computer Science*, *University of Vienna*, Vienna, Austria
[2]*UniVie Doctoral School Computer Science*, Vienna, Austria
[3]*Institute of Informatics, Ludwig Maximilian University of Munich*, Munich, Germany
[4]*ds:UniVie*, Vienna, Austria
{firstname.lastname}@univie.ac.at, durani@dbs.ifi.lmu.de

***Abstract*—The field of deep clustering combines deep learning and clustering to learn representations that improve both the learned representation and the performance of the considered clustering method. Most existing deep clustering methods are designed for a single clustering method, e.g., k-means, spectral clustering, or Gaussian mixture models, but it is well known that no clustering algorithm works best in all circumstances. Consensus clustering tries to alleviate the individual weaknesses of clustering algorithms by building a consensus between members of a clustering ensemble. Currently, there is no deep clustering method that can include multiple heterogeneous clustering algorithms in an ensemble to update representations and clusterings together. To close this gap, we introduce the idea of a consensus representation that maximizes the agreement between ensemble members. Further, we propose DECCS (Deep Embedded Clustering with Consensus representationS), a deep consensus clustering method that learns a consensus representation by enhancing the embedded space to such a degree that all ensemble members agree on a common clustering result. Our contributions are the following: (1) We introduce the idea of learning consensus representations for heterogeneous clusterings, a novel notion to approach consensus clustering. (2) We propose DECCS, the first deep clustering method that jointly improves the representation and clustering results of multiple heterogeneous clustering algorithms. (3) We show in experiments that learning a consensus representation with DECCS is outperforming several relevant baselines from deep clustering and consensus clustering.**

***Index Terms*—Deep Clustering, Representation Learning, Consensus Clustering**

I. Introduction

Clustering is the task of unsupervised classification, where we infer cluster labels from the data.[1] Deep clustering (DC) combines unsupervised deep learning and clustering to learn representations (embeddings) that improve clustering performance. Current DC methods are designed with only a single clustering model in mind, e.g., DEC [1] which improves the representation for k-means [2], VaDE [3] for Gaussian mixture models [4], DeepECT [5] for hierarchical clustering [6], and SpectralNet [7] for spectral clustering [8]. Relying on the assumptions of a single clustering model leads to poor results if the assumptions are not met by the data.

Consensus clustering (CC) can alleviate the limitations of individual clusterings by combining a clustering ensemble into a single robust clustering [9]. Unfortunately, applying

[1]Our code is available at https://gitlab.cs.univie.ac.at/lukas/deccs.

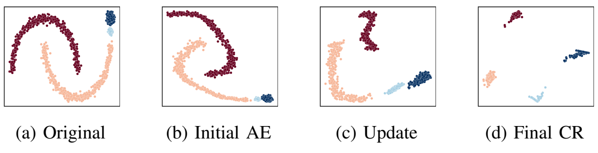


Fig. 1. A synthetic data set **(a)** containing four clusters is embedded **(b)** with an autoencoder (AE). DECCS transforms the initial AE embedding via several updates **(c)** to the final consensus representation (CR) in which clusters are compact and well separated **(d)**.

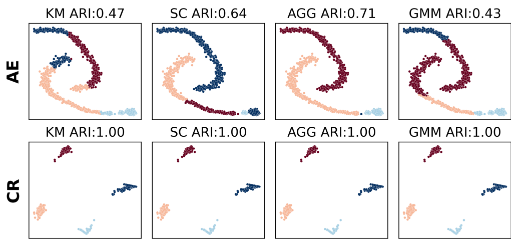


Fig. 2. Cluster performance on initial AE embedding and learned CR for an ensemble of k-means (KM), Spectral clustering (SC), Agglomerative clustering (AGG), and Gaussian mixture model (GMM).

current CC methods to high-dimensional data sets leads to unsatisfactory results, because they are either limited to linear transformations [10]–[12], only work for k-means like clusterings [13], or only use CC information as input features for DC without updating the CC in response to improved data representations [14], [15].

In contrast to that, we propose our novel **D**eep **E**mbedded **C**lustering with **C**onsensus representation**S** (DECCS) method, which is a DC method that can be applied to high-dimensional data, finds non-linearly hidden clusters and works with many existing clustering algorithms. DECCS learns a *consensus representation* (CR) that maximizes the agreement between ensemble members. The key idea we use for consensus representation learning with DECCS is that most clustering methods can find well-separated clusters in a low-dimensional space that have a simple shape, e.g., dense, spherical clusters. Using this idea, DECCS learns a consensus representation by transforming the embedded space such that it is trivial to cluster and, therefore, all ensemble members naturally agree on one partitioning into clusters. Fig. 1 illustrates on a synthetic data

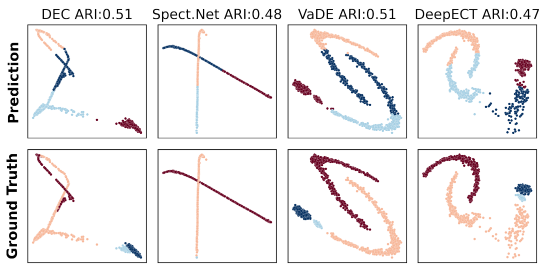


Fig. 3. Existing DC methods are limited by their assumed cluster model. Here they fail, because the data contains clusters of differing shapes.

set how DECCS transforms an initial embedding that contains clusters of different shapes to a consensus representation that consists only of dense, spherical, and well separated clusters. In Fig. 2, multiple, heterogeneous algorithms with different assumptions about the cluster structure are applied to the initial autoencoder (AE) embedding (upper row). Initially, the clustering algorithms perform poorly, but applied to the consensus representation learned with DECCS all algorithms in the ensemble reach the same, perfect clustering (bottom row) as measured with the adjusted rand index (ARI) [16]. In Fig. 3, we apply existing DC methods to the same synthetic data set using the same AE and plot their learned embeddings. While this data set can be clustered by classical clustering techniques, we see that DC methods fail, because their assumptions are not met. For example, DEC is performing poorly because the data set contains non-spherical clusters, which is not suited for k-means. As a consequence DEC is producing a distorted embedding (first column).

In this work, we tackle the shortcomings of existing DC and CC techniques and present the following contributions:

(1) We introduce the idea of a consensus representation, which is a representation that maximizes the agreement of the applied clustering algorithms by producing similar clustering results for all clustering methods included in the ensemble.

(2) We propose DECCS, the first DC algorithm that can include multiple heterogeneous clustering methods to jointly improve the learned embedding and clustering results by simplifying the representation.

(3) Our method is outperforming several relevant baselines in terms of cluster performance.

II. Background - Consensus Clustering

CC can overcome the limitations of individual clusterings by combining multiple clustering solutions into a single robust partitioning [9]. In general, CC algorithms consist of two stages:

1) Generate a set of base partitions using single clustering algorithms (e.g., k-means, Spectral Clustering, etc.)
2) Combine the base partitions using a consensus function to obtain a final partition.

Traditionally, the two stages are independent of each other. The consensus function does not access the original features of the data set to find the optimal combination of base partitions.

During the design of the consensus function the goal is to combine a set Π of $|\Pi|$ partitions π_i into one final clustering π_{cc}, such that π_{cc} agrees as much as possible with the base partitions. In their framework, [9] suggested to use the average pairwise normalized mutual information (ANMI) between the CC and the base clusterings as an objective function to measure the agreement:

$$\pi_{cc} = \text{argmax}_{\bar{\pi}} \sum_{i=1}^{|\Pi|} \text{NMI}(\pi_i, \bar{\pi}) \tag{1}$$

Using the normalized mutual information (NMI) has the benefit that it is invariant to the permutation and absolute values of cluster labels and allows for a different number of clusters k_i in each partition π_i. Further, the NMI is symmetric and is 1 if two clusterings match perfectly and 0 if they are independent of each other.

Instead of the need to design a consensus function to optimize Eq. 1 our DECCS algorithm learns a (non-linear) consensus function to learn the *consensus representation* as we explain in the following.

III. Objective Function for Consensus Representation Learning

For our novel problem setting, we use an encoder enc_Θ that maps a data point $\mathbf{x} \in \mathbb{R}^D$ to a typically lower-dimensional embedded vector $\mathbf{z} \in \mathbb{R}^d$, where Θ are the learnable parameters of the encoder. Then, let $\mathbf{X}$ be an $N \times D$ dimensional input data matrix and $\mathbf{Z} = \text{enc}_\Theta(\mathbf{X})$ be an $N \times d$ dimensional embedded data matrix with $d < D$. Further, let $\mathcal{E}$ be a set of heterogeneous clustering algorithms with potentially different number of clusters k_i, where each i^{th} member e_i produces a clustering result $\pi_i = e_i(\mathbf{Z})$. We define the consensus representation in the following.

Definition 1 (Consensus representation $\mathbf{Z}_{cr}$)**.** Let Θ, enc_Θ, $\mathbf{X}$, $\mathbf{Z}$, and $\mathcal{E}$ be defined as above. The *consensus representation* $\mathbf{Z}_{cr}$ maximizes the following objective function:

$$f_\Theta = c \sum_{i=1}^{|\mathcal{E}|} \sum_{j>i}^{|\mathcal{E}|} \text{NMI}(e_i(\text{enc}_\Theta(\mathbf{X})), e_j(\text{enc}_\Theta(\mathbf{X}))), \tag{2}$$

with $\mathbf{Z}_{cr} := \text{enc}_{\Theta_{cr}}(\mathbf{X})$, where $\text{enc}_{\Theta_{cr}}$ 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.

The consensus representation maximizes the agreement of all partitions with each other, where the agreement is measured using the pairwise NMI [9]. The optimal encoder parameters for the consensus representation $\mathbf{Z}_{cr}$ are then learned with

$$\Theta_{cr} = \text{argmax}_\Theta f_\Theta. \tag{3}$$

Note that Eq. 2 allows for degenerate solutions, like setting $\mathbf{Z}_{cr}$ to a constant if enc_Θ is non-linear. To avoid degenerate solutions in practice we include regularizers in the objective, like enforcing the invertibility of $\mathbf{Z}_{cr}$ back to $\mathbf{X}$ by using the AE reconstruction loss. In the following, we introduce our DECCS method and illustrate how it approaches the consensus representation learning problem.

IV. Method - DECCS

We motivate our consensus representation learning approach using the observation that most clustering methods are able to detect compact and well-separated clusters in low-dimensional spaces. DECCS uses the cluster information from all ensemble members to learn such a simplified representation. Decreasing the ambiguity of the representation during training will increase the similarity of the clusterings that ensemble members will produce, which subsequently increases the pairwise NMI (Eq. 2) of clustering results. DECCS works by alternating between representation and clustering update steps until an agreement is reached through the consensus representation. In the following, we explain our approach in more detail.

A. Overview

We use a (non-linear) autoencoder (AE) to learn enc by reconstructing the original input data $\mathbf{x}$ from $\mathbf{z}$ using the decoder dec resulting in $\hat{\mathbf{x}} := \mathrm{dec}(\mathrm{enc}(\mathbf{x}))$.[2] The AE reconstruction $\hat{\mathbf{x}}$ is learned by minimizing a reconstruction loss $\mathcal{L}_{\text{rec}} = \|\mathbf{x}-\hat{\mathbf{x}}\|$, e.g., using the mean squared error. Given the AE, our DECCS algorithm consists of three main steps that we explain in the following sections. First, we illustrate how we generate a set of base partitions by applying the cluster ensemble to a subsample of the embedding in Section IV-B. Second, we show how to approximate each partition with a classifier to label the remaining data points in Section IV-C. Third, we state our consensus objective in Section IV-D and in Section IV-E, we show how the consensus representation is updated. The algorithm is presented in Section IV-F.

B. Generating base partitions

At the beginning of each round t of our algorithm, we draw a small random sample $\mathbf{X}_t$ of size $n < N$ from $\mathbf{X}$, because some clustering algorithms are impractical to be applied to large data sets and re-sampling can make the CC more robust [17]. Next, we embed the sample using $\mathrm{enc}_\Theta(\mathbf{X}_t) = \mathbf{Z}_t$ and generate a set of base partitions Π_t by applying all ensemble members to the embedding $\pi_i = e_i(\mathbf{Z}_t)$. The sampling procedure and the low-dimensional embedded space allow us to use more run-time and memory expensive algorithms, such as spectral clustering, in our ensembles. Further, using the sampling and heterogeneous ensembles, we can achieve a sufficiently diverse set of base partitions Π_t.

C. Approximating base partitions

Since we only have cluster labels for $n < N$ data points due to the random sampling, but require cluster labels for all N data points, we use a classifier to approximate the clustering for the remaining $N-n$ points. We approximate the set of base partitions Π_t using a set of classifiers $\mathcal{G}_t$, where classifier g_i is trained to predict the corresponding clustering π_i. We train each classifier by minimizing the cross-entropy loss of cluster labels π_i and its prediction, i.e.,

$$\mathcal{L}_{\text{CE}} = \sum_{i=1}^{|\Pi_t|} \mathcal{L}_{CE}^i = -\frac{1}{n}\sum_{j=1}^{n}\sum_{l=1}^{k_i} I[l = \pi_{i,j}] \log g_i(\mathbf{z}_j), \quad (4)$$

where $\pi_{i,j}$ is the cluster label corresponding to the jth data point and I is the indicator function. While in principle one can use any classifier for g_i we chose linear classifiers with the Softmax function as output, i.e., $g_i(\mathbf{x}) = \mathrm{softmax}(\mathbf{W}_i\mathbf{x}+\mathbf{b}_i)$ with $\mathbf{W}_i$ and $\mathbf{b}_i$ as weights and bias terms respectively. The linear classifiers can be trained with little overhead, having only $d \cdot k_i + k_i$ trainable parameters. Updating the linear classifiers together with the non-linear encoder allows us then to approximate non-linear clusterings as well.

D. Consensus Objective

Optimizing Eq. 2 from Definition 1 directly is not possible, because the cluster ensemble members are not differentiable. Thus, we learn a low-dimensional representation in which all clusters are spherical, dense, and well separated, such that the ensemble members trivially agree on one partition. To transform non-spherical-shaped clusters into spherical clusters we "move" cluster points closer to their cluster representatives. We choose the mean center of a cluster as the representative because it is stable across update steps, but other choices like the median are also possible. In the following, we use the terms representative and center interchangeably.

Let $\mathcal{C}_{\pi_i}^j$ be the set of data points in the jth cluster of partition π_i, then we can calculate its center μ_j using $\mu_j = \frac{1}{|\mathcal{C}_{\pi_i}^j|}\sum_{\mathbf{x}\in\mathcal{C}_{\pi_i}^j} \mathrm{enc}_\Theta(\mathbf{x})$, and subsequently can construct the $k_i \times d$ matrix M_i containing k_i centers μ_j as row vectors. Next, we define our differentiable consensus objective as

$$\begin{aligned}\mathcal{L}_{\text{cons}} &= \sum_{i=1}^{|\Pi_t|} \mathcal{L}_{\text{cons}}^i = \sum_{i=1}^{|\Pi_t|} \|\mathbf{A}_i\mathbf{M}_i - \mathrm{enc}_\Theta(\mathbf{X}_t)\|_F^2 \quad (5)\\ &= \sum_{i=1}^{|\Pi_t|}\sum_{l=1}^{k_i}\sum_{\mathbf{x}\in\mathcal{C}_{\pi_i}^l} \|\mu_l - \mathrm{enc}_\Theta(\mathbf{x})\|_2^2,\end{aligned}$$

where $\mathbf{A}_i$ is the $n \times k_i$ one hot encoded cluster assignment matrix of partition π_i, $\|\cdot\|_2^2$ the squared Euclidean norm, and $\|\cdot\|_F^2$ the squared Frobenius norm. Here $\mathbf{M}_i$ and $\mathbf{A}_i$ are fixed, so the encoder enc_Θ has to learn parameters Θ that map embedded data points $\mathbf{z} = \mathrm{enc}_\Theta(\mathbf{x})$ as close as possible to their assigned centers across all partitions. Note, that data points that are close to similar centers across partitions will receive a higher gradient update due to the summation and are thus gathering faster than data points that have conflicting assignments. The centers alone can not capture complex cluster structures properly, which is why we include the cross-entropy loss (Eq. 4) in our objective, as we explain in the next Section.

[2]We reuse enc here, whether it is vector or matrix-valued should be clear from the context.

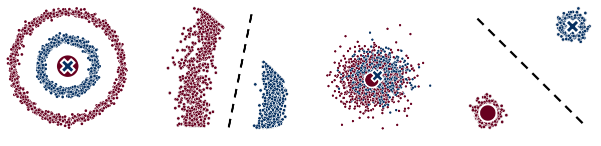


Fig. 4. Example of how optimizing $\mathcal{L}_{\text{CE}}$ together with $\mathcal{L}_{\text{cons}}$ is able to enhance the representation even if cluster centers overlap. **(a)** A data set with two clusters (red and blue circle) that have the same centers (red dot and blue cross). **(b)** Optimizing DECCS only with $\mathcal{L}_{\text{CE}}$ learns a representation in which both clusters are separated by the decision boundary (dashed black line), but cluster shapes are distorted. **(c)** Only using $\mathcal{L}_{\text{cons}}$ merges both clusters. **(d)** Optimizing $\mathcal{L}_{\text{CE}}$ together with $\mathcal{L}_{\text{cons}}$ separates the clusters and transforms them from circles to spheres, a representation that can be easily clustered.

E. Updating consensus representation

Putting everything together the objective of our DECCS algorithm is

$$\mathcal{L} = \sum_{i}^{|\Pi_t|} \lambda_i (\mathcal{L}_{CE}^i + \mathcal{L}_{\text{cons}}^i) + \lambda_{rec}\mathcal{L}_{\text{rec}}, \tag{6}$$

where $\lambda_i = \frac{1}{|\Pi_t|-1}\sum_{j=1}^{|\Pi_t|}\text{NMI}(\pi_i, \pi_j)$ is a weighting parameter based on the agreement of partition π_i with all other partitions measured as average pairwise NMI to exclude random partitions and downscale outlier partitions. We choose the commonly used AE reconstruction loss $\mathcal{L}_{\text{rec}}$ as data dependent regularizer to avoid degenerate solutions by keeping $\mathbf{Z}$ approximately invertible. The hyperparameter λ_{rec} weights the importance of $\mathcal{L}_{\text{rec}}$.

The cross-entropy loss $\mathcal{L}_{CE}^i$ of each classifier is included to make sure that the updated representation is still predictive for each partition, e.g., by avoiding the merging of clusters if centers of different clusters are the same, as illustrated in Fig. 4. Additionally, we show in the Appendix in Fig. 9 that this still works even if half of the ensemble members are no better than chance.

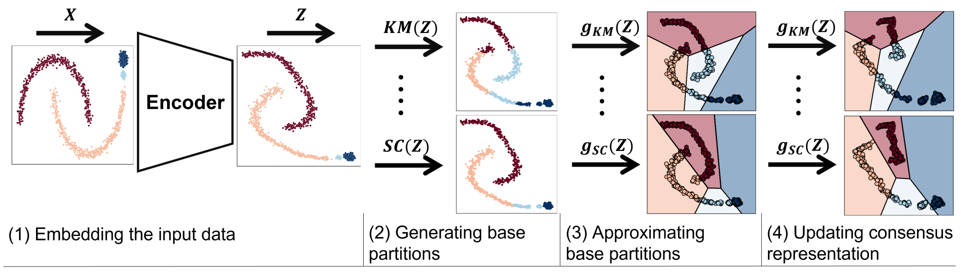
Eq. 6 can be optimized using stochastic gradient descent for a fixed amount of update steps. Once the training has stopped, our algorithm starts again with a new round by applying the clustering algorithms, which will adjust their clustering results to the updated embedding. Then it again pretrains classifiers to approximate them and minimizes $\mathcal{L}$. These steps are repeated for several rounds until a stable agreement is achieved or a maximum number of rounds T has been reached. In the following section, we explain our algorithm in more detail.

F. Algorithm

Given a data set $\mathbf{X}$, the pretrained encoder enc_Θ, and a parameterized ensemble of clustering methods $\mathcal{E}$, we learn a consensus representation $\mathbf{Z}_{cr}$ and subsequently a consensus clustering π_{cc} with our DECCS algorithm in the following way. We encode the input data using enc_Θ, generate the base partitions by applying cluster ensemble members on $\mathbf{Z}_t$, approximate the base partitions using classifiers g_i and update the representation by minimizing $\mathcal{L}$. We repeat these steps for several rounds until a stable agreement is achieved or we reached a maximum number of rounds T. As agreement function $a(\Pi_t)$ we use the average pairwise NMI between all clusterings in the set of partitions Π_t. We measure the stability of the agreement by calculating $\|a(\Pi_t) - a(\Pi_{t-1})\|_1 < \tau$, where τ is the cluster agreement threshold, a user-specified parameter, and $\|\cdot\|_1$ is the absolute distance between the agreement of two subsequent sets of partitions. After the algorithm stops, it returns the estimated consensus representation $\hat{\mathbf{Z}}_{cr}$ and it's corresponding estimated consensus clustering $\hat{\pi}_{cc}$. The consensus clustering $\hat{\pi}_{cc}$ is obtained by applying a clustering algorithm from the ensemble, e.g., k-means with the desired k to $\hat{\mathbf{Z}}_{cr}$. If the number of clusters is the same in all ensemble members ($k_i = k_j, \forall e_i, e_j \in \mathcal{E}$), we choose for $\hat{\pi}_{cc}$ just the result of one of the ensemble members. The pseudocode of our algorithm is depicted in Algorithm 1. In Fig. 5, we have a visual illustration of one round of our DECCS algorithm applied to a synthetic data set, and Fig. 6 shows its optimization over several rounds.

Algorithm 1: DECCS

```
Param : Agreement function $a(\cdot)$, agreement threshold $\tau$,
        subsample size $n$, regularization weight $\lambda_{rec}$,
        rampup function $w(\cdot)$, maximum number of rounds
        $T$, number of mini-batch training iterations ITER
Input : data set $\mathbf{X}$
        initial representation function $\text{enc}_\Theta$
        ensemble of clustering algorithms $\mathcal{E}$
Output: Estimated consensus representation $\hat{\mathbf{Z}}_{cr}$ and
        corresponding consensus clustering $\hat{\pi}_{cc}$
1  $t = 0$;
2  while $t \leq T$ do
3      draw sample $\mathbf{X}_t$ of size $n$ from $\mathbf{X}$ and create
         corresponding embedding $\mathbf{Z}_t = \text{enc}_\Theta(\mathbf{X}_t)$;
       // Generate new base partitions
4      generate new empty list for cluster partitions $\Pi_t$;
5      foreach ensemble member $e_i \in \mathcal{E}$ do
6          insert cluster prediction $\pi_i = e_i(\mathbf{Z}_t)$ into $\Pi_t$;
       // Approximate base partitions
7      initialize list of classifiers $\mathcal{G}_t$;
8      foreach cluster prediction $\pi_i \in \Pi_t$ do
9          pretrain classifier $g_i$ by minimizing
             $\mathcal{L}_{CE}^i(g_i(\mathbf{Z}_t), \pi_i)$;
10         insert pretrained classifier $g_i$ into $\mathcal{G}_t$;
       // Stop if stable agreement or $T$ is
          reached
11     if $(t > 0) \wedge ((\|a(\Pi_t) - a(\Pi_{t-1})\|_1 < \tau) \vee (t == T))$
         then
12         $\hat{\mathbf{Z}}_{cr} := \text{enc}_\Theta(\mathbf{X})$, $\hat{\pi}_{cc} := g_0(\hat{\mathbf{Z}}_{cr})$;
13         break;
14     else
           // Update consensus representation
15         while $j < ITER$ do
16             foreach mini-batch $\mathcal{B} \in X$ do
17                 calculate for $\mathcal{B}$ loss $\mathcal{L}$:
18                   $\sum_i^{|\Pi_t|} \lambda_i(\mathcal{L}_{CE}^i + w(t)\mathcal{L}_{\text{cons}}^i) + \lambda_{rec}\mathcal{L}_{\text{rec}}$;
19                 update $\text{enc}_\Theta$ and $\mathcal{G}_t$ using $\mathcal{L}$;
20                 $j = j + 1$;
21     $t = t + 1$;
22 return $\hat{\mathbf{Z}}_{cr}$, $\hat{\pi}_{cc}$;
```



X
Encoder
Z
$KM(Z)$
$SC(Z)$
$g_{KM}(Z)$
$g_{SC}(Z)$
$g_{KM}(Z)$
$g_{SC}(Z)$
(1) Embedding the input data
(2) Generating base partitions
(3) Approximating base partitions
(4) Updating consensus representation

Fig. 5. Visualisation of one round of DECCS. (1) The encoder is used to embed data points $\mathbf{X}$. (2) Clustering results are generated by applying ensemble members $\mathcal{E} = \{\text{KM}, \ldots, \text{SC}\}$ to $\mathbf{Z}$. (3) Classifiers g_i are trained to predict the corresponding cluster labels π_i from $\mathbf{Z}$. (4) $\mathbf{Z}$ is updated via minimizing $\mathcal{L}$.

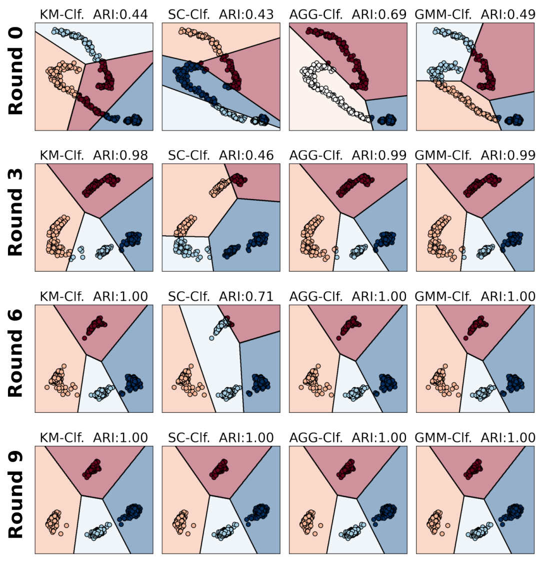


Fig. 6. Consensus representation learning with DECCS over several rounds for the synthetic data set and $\mathcal{E} = \{\text{KM}, \text{SC}, \text{AGG}, \text{GMM}\}$. Each plot shows the classification boundaries for each classifier (Clf.) trained on the cluster partitions. Over several rounds the clusters get better separated and more compact, leading to the same clustering for all ensemble members.

We use three heuristics for the optimization of DECCS. First, to speed up convergence we include the predicted cluster labels of the $N - n$ unclustered data points for each classifier g_i during the computation of $\mathcal{L}^i_{\text{cons}}$. These predictions are updated during each mini-batch iteration for unclustered data points in the mini-batch $\mathcal{B}$. Second, to account for the classifiers' uncertainty we weight each distance computation in $\mathcal{L}^i_{\text{cons}}$ with $\alpha_{i,l} = g_{i,l}(\text{enc}_\Theta(\mathbf{x}))$, which is the l^{th} entry of the prediction probability vector of classifier g_i. Third, to enforce the consensus over time t, we increase the weight of the consensus loss until a maximum weight λ_{cons} is reached. We use the sigmoid schedule as rampup function w, like [18], to increase the weight $w(t)$ from 0 to λ_{cons} over time. In total, our algorithm needs the following user-specified parameters, an agreement threshold τ that indicates how small the agreement gap between two subsequent sets of partitions should be. The data sampling size n, which should be chosen w.r.t. computational constraints and demands of clustering algorithms to have a sufficient number of samples. The maximum consensus weight λ_{cons} is a hyperparameter that together with the regularization weight λ_{rec} trades-off the confidence in the chosen ensemble with the structure of the underlying data. The maximum number of rounds T and the maximum number of mini-batch iterations ITER for the consensus representation update can be set based on computational constraints. We speed up the training of classifiers and encoders using early stopping, a heuristic that stops training once the loss on a held-out evaluation set starts to increase due to overfitting.

V. Related Work

A. Consensus Clustering

Based on the consensus function (CF) consensus clustering methods can be broadly categorized into median partition- and object co-occurrence based methods. Median partition methods find a partition that is most similar to all the base partitions. Object co-occurrence based methods utilize the co-association (CA) matrix to find the ideal partitioning, where the entries of this matrix reflect how often every two instances are partitioned together. Fred *et al.* [19] introduced Evidence Accumulation (EAC), a hierarchical clustering algorithm that uses entries of the CA matrix as a similarity measure that is used to produce the final clustering. More recently, [20] extended this idea by proposing Locally Weighted Evidence Accumulation (LWEA), introducing an entropy-based weighting schema, which makes it more robust to outlier partitions. Strehl *et al.* [9], and later Fern *et al.* [21], utilized the CA matrix to formulate graph-based algorithms as a consensus function. Li *et al.* [22] proposed a more efficient Nonnegative Matrix Factorization (NMF) based algorithm to factorize the CA matrix as an alternative.

To generate base partitions for high dimensional data, like images, a line of research follows the idea of random projections (RP). Inspired by the Johnson–Lindenstrauss (JS) lemma [23], [10] introduced with Random Projection Expectation Minimization (RP+EM) the first RP-based CC algorithm, where the data is projected onto various lower-dimensional subspaces using random matrices. The entries of the resulting

CA matrix are then used for a hierarchical clustering approach. Similar to this idea, [11] proposed Random Projection Fuzzy c-Means (RP+FCM), where each subspace is clustered with a Fuzzy c-Means (FCM) algorithm. Those partitions are then combined with an agreement function. However, contrary to DECCS, RP methods are limited to linear transformations.

B. Deep Clustering

Most, current DC methods are designed with only a single clustering model in mind, e.g., SpectralNet [7] for spectral clustering, DEC [1], IDEC [24], DCN [25] for k-means like clustering, VaDe [3] for Gaussian mixture models, or Deep-ECT [5] for hierarchical clustering or ENRC [26] for non-redundant clustering, see [27] for an overview. SpectralNet is a deep extension of spectral clustering for large data and out-of-sample generalization. DEC minimizes a soft auxiliary target distribution using the Kullback-Leibler divergence, which is related to soft k-means [28]. Improved DEC (IDEC) includes the AE reconstruction loss in the DEC objective to avoid arbitrary clustering results. In contrast to the soft clustering objective of DEC, the DCN algorithm uses hard cluster assignments together with an alternating optimization scheme. It alternates between k-means clustering and representation update to achieve a k-means friendly embedding. VaDE combines a Gaussian mixture model prior with a variational autoencoder (VAE) [29] to learn a deep generative clustering. DeepECT [5] introduced a deep embedded cluster tree to learn a hierarchical embedding.

The ConCURL [13] algorithm leverages image augmentation and RPs to learn a cluster ensemble of Softmax predictions to improve the overall clustering performance. A difference between their approach and ours is that they are limited to data that can be augmented, e.g., images or text. Further, they create a k-means like ensemble by using the Softmax, see [30] for the connection between the Softmax and k-means. In contrast to that, our DECCS algorithm can be used with a wide range of existing clustering methods and is not limited to k-means. Liu *et al.* [14] proposed the IEC algorithm which embeds multiple clustering results with a marginalized Denoising AE [31] and clusters the learned embedding with k-means, without considering the original data. Tao *et al.* [15] extended the idea of [14] and proposed the AGAE method. Instead of embedding the clustering results, AGAE uses a consensus graph constructed from the CA matrix of the base partitions as an input to a DC method, which together with the original data produces an enriched embedding. In contrast to our approach, AGAE does not learn a consensus with the neural network but uses initial clusterings to construct a consensus graph as input for their DC algorithm, without updating the graph during training. Importantly and in contrast to ConCURL [13] and DECCS, both IEC and AGAE are not jointly updating the consensus clusterings and representation, a key feature of DC [27] to improve cluster performance.

VI. Experiments

We evaluate our DECCS algorithm with respect to several aspects. In Section VI-A, we evaluate DECCS w.r.t. its most important hyperparameters for MNIST [32] as it is usually done in DC [1], [3], [24], [25] and show that our objective increases the agreement and cluster performance for all ensemble members across data sets. Additionally, we perform an ablation study across data sets. In Section VI-B, we compare DECCS to several CC and DC methods.

Evaluation Metrics: We evaluate the performance using normalized mutual information (NMI) [33] and adjusted rand index (ARI) [16]. Both range between 0 and 1, where 0 indicates no match and 1 a perfect match with the ground-truth. We evaluate the agreement within an ensemble by calculating the average pairwise NMI [9] between all clusterings.

Data sets: The synthetic data set (SYNTH) consists of four clusters and is depicted in Fig. 1a. The real-world data sets consist of commonly used DC image data sets like MNIST, Fashion-MNIST (FMNIST) [34], Kuzushiji-MNIST (KMNIST) [35], and USPS [36] and three UCI [37] data sets PENDIGITS, HAR and MICE. We provide a detailed description of the data sets in Appendix A. All data sets are preprocessed using a z-transformation.

Experimental Setup: For all data sets that have more than 2,000 data points, we use a feed-forward AE architecture with layers D-500-500-2000-10 for the encoder and a corresponding mirrored decoder, which is the same setting as used in [1]. For the MICE and SYNTH data sets, we have used smaller networks, with D-256-128-64 and D-20-20-2 for the encoders and mirrored decoders respectively. We use these architectures for DECCS, DEC, IDEC, DCN, and VaDE. For SpectralNet[3] and ConCURL[4], we used the settings that are available in their public implementations. IEC and AGAE have no publicly available code, which is why we only show the NMI results reported in their papers[5].

For hyperparameters that are specific to DECCS, we set λ_{cons} to 0.1 for the image data sets and to 10.0 for the UCI and SYNTH data sets, where the higher weight leads to better results for all data sets. The sampling size n is set to $0.08 \cdot N$ for data sets with $N > 11{,}000$ and to $0.5 \cdot N$ for the others. We let our algorithm run for $T = 10$ rounds and report the result with the highest agreement between ensemble members, thus not needing to specify τ. We train the classifiers and encoder of DECCS with mini-batch SGD and momentum [38] set to 0.9 for all data sets and $|\mathcal{B}| = 256$. The classifiers are pretrained with a learning rate of 0.01 and the representation updates are done with a learning rate of 0.001, which is reduced by a factor of 0.9 after each round t. We set the number of maximum mini-batch iterations to ITER $= 20{,}000$ for all data sets. We used the early stopping heuristic during the classifier pretraining and consensus representation learning

[3]https://github.com/KlugerLab/SpectralNet

[4]https://github.com/JayanthRR/ConCURL_NCE

[5]Symbol † indicates results are taken from [14] and ‡ from [15]

TABLE I
ABLATION STUDY FOR COMBINATIONS OF LOSS TERMS OF DECCS. BEST RESULTS ARE MARKED AS BOLD AND RUNNER-UP IS UNDERLINED. ALL RESULTS ARE GIVEN IN NMI AS MEAN ± STD OVER 10 RUNS.

$\mathcal{L}_{\text{cons}}$	$\mathcal{L}_{\text{CE}}$	$\mathcal{L}_{\text{rec}}$	**SYNTH**	**MICE**	**PENDIGITS**	**HAR**	**MNIST**	**FMNIST**	**KMNIST**	**USPS**
X			0.17 ± 0.21	0.44 ± 0.05	0.12 ± 0.07	0.01 ± 0.01	0.45 ± 0.03	0.42 ± 0.02	0.27 ± 0.01	0.52 ± 0.04
	X		0.79 ± 0.08	0.42 ± 0.02	0.72 ± 0.02	0.68 ± 0.04	0.81 ± 0.04	0.58 ± 0.01	0.54 ± 0.01	0.77 ± 0.01
		X	0.53 ± 0.03	0.43 ± 0.02	0.68 ± 0.01	0.55 ± 0.05	0.75 ± 0.01	0.61 ± 0.01	0.49 ± 0.01	0.67 ± 0.02
X		X	0.41 ± 0.02	0.50 ± 0.04	$\underline{0.74} \pm 0.02$	0.60 ± 0.03	0.79 ± 0.01	$\underline{0.64} \pm 0.01$	0.52 ± 0.02	0.71 ± 0.01
	X	X	$\underline{0.82} \pm 0.09$	0.43 ± 0.04	0.73 ± 0.02	0.65 ± 0.06	0.83 ± 0.02	0.63 ± 0.02	0.56 ± 0.01	0.79 ± 0.01
X	X		$\mathbf{0.99} \pm 0.01$	$\underline{0.55} \pm 0.03$	$\mathbf{0.82} \pm 0.02$	$\underline{0.73} \pm 0.03$	$\underline{0.87} \pm 0.02$	$\underline{0.64} \pm 0.01$	$\underline{0.60} \pm 0.01$	$\underline{0.84} \pm 0.02$
X	X	X	$\mathbf{0.99} \pm 0.02$	$\mathbf{0.57} \pm 0.03$	$\mathbf{0.82} \pm 0.02$	$\mathbf{0.75} \pm 0.02$	$\mathbf{0.88} \pm 0.02$	$\mathbf{0.65} \pm 0.01$	$\mathbf{0.61} \pm 0.01$	$\mathbf{0.85} \pm 0.01$

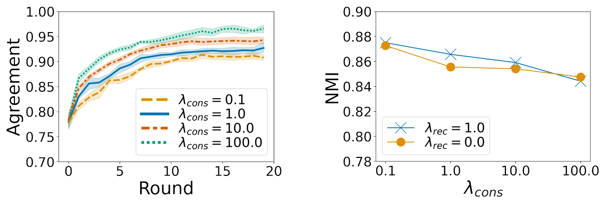


Fig. 7. DECCS parameter analysis for MNIST. (Left) Average agreement (thick lines) and 95% confidence intervals for ten runs of DECCS show that increasing the consensus weight λ_{cons} leads to an increased agreement between ensemble members during training. (Right) Average cluster performance for different values of λ_{cons} and λ_{rec} over ten runs.

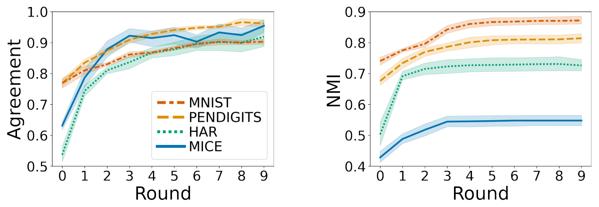


Fig. 8. Average agreement (thick lines) and 95% confidence intervals for ten runs of DECCS show the increase in agreement between ensemble members over training across data sets on the left side and the corresponding increase in cluster performance of DECCS on the right side.

and decreased the learning rate by 0.9 when a loss plateau was reached.

DEC, IDEC, and DCN are learning k-means friendly embeddings, SpectralNet extends spectral clustering, DeepECT learns hierarchical embeddings, and VaDE is a deep version of Gaussian mixture models. We, therefore, choose a heterogeneous ensemble of k-means (KM), spectral clustering (SC), agglomerative clustering (AGG), and Gaussian mixture models (GMM), based on the correspondence of the chosen DC methods, i.e., $\mathcal{E} = \{\text{KM}, \text{SC}, \text{AGG}, \text{GMM}\}$.

For the CC approaches, we compare against eight methods (six classical methods, two utilizing RPs). We evaluate the CC methods on the raw and the AE embedded data sets using the same ensemble $\mathcal{E}$ as DECCS. For all methods, we assume the number of clusters k to be known. We provide hyperparameter settings and further details for all methods in Appendix B. We uploaded the used data sets, our code and further results at https://gitlab.cs.univie.ac.at/lukas/deccs.

A. Algorithm Evaluation

Ablation study: We evaluate the impact of the individual components of DECCS' loss function in Table I. We see that the combination of consensus loss ($\mathcal{L}_{\text{cons}}$) and cross-entropy loss ($\mathcal{L}_{\text{CE}}$), with and without reconstruction loss ($\mathcal{L}_{\text{rec}}$) perform best (last two rows) for all data sets. Using only $\mathcal{L}_{\text{cons}}$ without $\mathcal{L}_{\text{CE}}$ leads to worse results because the classifiers are not preventing the merging of clusters (first row), as we have discussed in Fig. 4.

Impact of ensemble size: We evaluated the impact of the ensemble size by increasing its original size $|\mathcal{E}| = 4$ by doubling and tripling each member in the ensemble $\mathcal{E}$, leading to $|\mathcal{E}_{\times 2}| = 8$ and $|\mathcal{E}_{\times 3}| = 16$. We evaluated the results by averaging the cluster performance of DECCS over ten runs on MNIST, where we achieved the same average NMI of 0.87 for each ensemble. We believe that we cannot see a benefit here because we use strong clustering methods, which do not add more diversity to the ensemble.

Impact of sampling size n: To evaluate the effect of the sampling size n, we varied it for ratios $\{0.02, 0.04, 0.06, 0.08\}$ of MNIST ($N = 70{,}000$) and averaged the performance over ten runs. DECCS achieved the same performance (NMI $= 0.87$) for all ratios. We chose 0.08 for the remaining experiments as this was also stable for the other large data sets.

Impact of λ_{cons} and λ_{rec}: To demonstrate that the objective of DECCS increases the agreement between cluster ensemble members, we vary the consensus weight λ_{cons} for values in $\{0.1, 1.0, 10.0, 100.0\}$ for the MNIST data set while keeping $\lambda_{rec} = 0$. We see on the left side of Fig. 7 that a higher value for λ_{cons} leads to a corresponding higher agreement. This is expected because we enforce the consensus with a higher weight. The right side of Fig. 7 shows the corresponding average cluster performance for $\lambda_{rec} \in \{0.0, 1.0\}$. The trend with and without the reconstruction loss is similarly downwards trending for very high values because a very highly weighted consensus loss disregards the underlying structure of the data.

Increase of agreement and NMI during training: On the left side of Fig. 8, we show how DECCS increases the ensemble agreement over training for three UCI data sets respectively, and for MNIST as the behavior for the image data sets was very similar. This experiment gives additional evidence that our algorithm can effectively maximize the pairwise NMI

TABLE II
CLUSTER PERFORMANCE RESULTS MEASURED IN NMI. CORRESPONDING ARI VALUES ARE SHOWN IN TABLEIII.

Method	**SYNTH**	**MICE**	**PENDIGITS**	**HAR**	**MNIST**	**FMNIST**	**KMNIST**	**USPS**
DECCS	**0.99** $\pm$ 0.02	**0.57** $\pm$ 0.03	**0.82** $\pm$ 0.02	**0.75** $\pm$ 0.02	0.88 $\pm$ 0.02	**0.65** $\pm$ 0.01	**0.61** $\pm$ 0.01	**0.85** $\pm$ 0.01
CSPA [9]	0.75 $\pm$ 0.00	0.35 $\pm$ 0.02	0.73 $\pm$ 0.02	0.49 $\pm$ 0.02	0.59 $\pm$ 0.04	0.53 $\pm$ 0.03	0.46 $\pm$ 0.02	0.65 $\pm$ 0.02
HGPA [9]	0.75 $\pm$ 0.00	0.37 $\pm$ 0.01	0.61 $\pm$ 0.05	0.49 $\pm$ 0.04	0.47 $\pm$ 0.02	0.46 $\pm$ 0.02	0.35 $\pm$ 0.02	0.56 $\pm$ 0.03
MCLA [9]	0.57 $\pm$ 0.08	0.34 $\pm$ 0.01	0.73 $\pm$ 0.04	0.59 $\pm$ 0.03	0.55 $\pm$ 0.05	0.53 $\pm$ 0.02	0.49 $\pm$ 0.02	0.60 $\pm$ 0.08
HBGF [21]	0.62 $\pm$ 0.02	0.33 $\pm$ 0.03	0.72 $\pm$ 0.03	0.48 $\pm$ 0.03	0.58 $\pm$ 0.03	0.51 $\pm$ 0.04	0.46 $\pm$ 0.02	0.64 $\pm$ 0.01
NMF [22]	0.46 $\pm$ 0.05	0.34 $\pm$ 0.01	0.75 $\pm$ 0.03	0.59 $\pm$ 0.03	0.59 $\pm$ 0.03	0.52 $\pm$ 0.03	0.49 $\pm$ 0.03	0.72 $\pm$ 0.03
LWEA [20]	0.60 $\pm$ 0.02	0.37 $\pm$ 0.03	0.76 $\pm$ 0.02	0.59 $\pm$ 0.00	0.62 $\pm$ 0.03	0.56 $\pm$ 0.01	0.51 $\pm$ 0.01	0.72 $\pm$ 0.01
RP+EM [10]	0.61 $\pm$ 0.00	0.46 $\pm$ 0.04	0.67 $\pm$ 0.05	0.46 $\pm$ 0.06	0.48 $\pm$ 0.05	0.50 $\pm$ 0.05	0.44 $\pm$ 0.04	0.64 $\pm$ 0.04
RP+FCM [11]	0.64 $\pm$ 0.05	0.28 $\pm$ 0.08	0.63 $\pm$ 0.01	0.51 $\pm$ 0.01	0.22 $\pm$ 0.02	0.40 $\pm$ 0.01	0.25 $\pm$ 0.01	0.38 $\pm$ 0.02
AE+CSPA [9]	0.76 $\pm$ 0.05	0.41 $\pm$ 0.03	0.73 $\pm$ 0.02	0.53 $\pm$ 0.01	0.84 $\pm$ 0.02	0.59 $\pm$ 0.02	0.54 $\pm$ 0.02	0.74 $\pm$ 0.03
AE+HGPA [9]	0.75 $\pm$ 0.00	0.41 $\pm$ 0.03	0.64 $\pm$ 0.07	0.49 $\pm$ 0.05	0.61 $\pm$ 0.02	0.50 $\pm$ 0.03	0.43 $\pm$ 0.03	0.60 $\pm$ 0.02
AE+MCLA [9]	0.53 $\pm$ 0.11	0.43 $\pm$ 0.02	0.73 $\pm$ 0.05	0.59 $\pm$ 0.06	0.83 $\pm$ 0.01	0.62 $\pm$ 0.03	0.59 $\pm$ 0.02	0.75 $\pm$ 0.06
AE+HBGF [21]	0.65 $\pm$ 0.08	0.40 $\pm$ 0.03	0.71 $\pm$ 0.04	0.53 $\pm$ 0.03	0.83 $\pm$ 0.02	0.58 $\pm$ 0.02	0.54 $\pm$ 0.02	0.72 $\pm$ 0.02
AE+NMF [22]	0.50 $\pm$ 0.13	0.44 $\pm$ 0.04	0.76 $\pm$ 0.04	0.60 $\pm$ 0.01	0.82 $\pm$ 0.04	0.61 $\pm$ 0.02	0.59 $\pm$ 0.03	0.82 $\pm$ 0.04
AE+LWEA [20]	0.61 $\pm$ 0.04	0.46 $\pm$ 0.04	0.75 $\pm$ 0.03	0.58 $\pm$ 0.06	0.86 $\pm$ 0.02	**0.65** $\pm$ 0.01	**0.61** $\pm$ 0.03	0.83 $\pm$ 0.03
AE+RP+EM [10]	0.62 $\pm$ 0.03	0.51 $\pm$ 0.04	0.67 $\pm$ 0.04	0.48 $\pm$ 0.04	0.77 $\pm$ 0.04	0.59 $\pm$ 0.02	0.58 $\pm$ 0.03	0.65 $\pm$ 0.03
AE+RP+FCM [11]	0.63 $\pm$ 0.10	0.41 $\pm$ 0.03	0.54 $\pm$ 0.05	0.48 $\pm$ 0.03	0.45 $\pm$ 0.07	0.49 $\pm$ 0.03	0.31 $\pm$ 0.04	0.37 $\pm$ 0.03
SpectralNet [7]	0.72 $\pm$ 0.06	0.27 $\pm$ 0.06	**0.82** $\pm$ 0.04	0.61 $\pm$ 0.06	**0.92** $\pm$ 0.00	0.64 $\pm$ 0.01	**0.61** $\pm$ 0.02	0.83 $\pm$ 0.02
DEC [1]	0.65 $\pm$ 0.03	0.49 $\pm$ 0.02	0.75 $\pm$ 0.02	0.54 $\pm$ 0.09	0.84 $\pm$ 0.01	0.60 $\pm$ 0.01	0.52 $\pm$ 0.01	0.80 $\pm$ 0.01
IDEC [24]	0.64 $\pm$ 0.03	0.50 $\pm$ 0.03	0.76 $\pm$ 0.02	0.53 $\pm$ 0.09	0.85 $\pm$ 0.02	0.62 $\pm$ 0.02	0.55 $\pm$ 0.03	0.81 $\pm$ 0.01
DCN [25]	0.59 $\pm$ 0.08	0.48 $\pm$ 0.04	0.75 $\pm$ 0.02	0.51 $\pm$ 0.08	0.84 $\pm$ 0.03	0.62 $\pm$ 0.02	0.54 $\pm$ 0.04	0.78 $\pm$ 0.04
VaDE [29]	0.62 $\pm$ 0.10	0.45 $\pm$ 0.06	0.75 $\pm$ 0.02	0.54 $\pm$ 0.09	0.83 $\pm$ 0.03	**0.65** $\pm$ 0.01	0.56 $\pm$ 0.01	0.79 $\pm$ 0.03
DeepECT [5]	0.61 $\pm$ 0.10	0.47 $\pm$ 0.06	0.74 $\pm$ 0.02	0.56 $\pm$ 0.10	0.82 $\pm$ 0.03	0.62 $\pm$ 0.03	0.52 $\pm$ 0.04	0.76 $\pm$ 0.06
ConCURL [13]	n.a.	n.a.	n.a.	n.a.	0.60 $\pm$ 0.04	0.48 $\pm$ 0.02	0.30 $\pm$ 0.03	0.49 $\pm$ 0.02
IEC [14]	-	-	0.72‡	-	0.54†	-	-	0.64†
AGAE [15]	-	-	0.74‡	-	-	-	-	0.74‡

between ensemble members (Eq. 2) by learning a consensus representation. The agreement is stabilizing for all data sets at round 8, except for MICE which fluctuates at a high agreement level due to the smaller data set size. The right side of Fig. 8 shows the corresponding increase in cluster performance. We see that DECCS reaches stable cluster performance already after round five for all data sets.

B. Cluster performance

In Table II, we show the clustering results of all methods w.r.t. NMI over ten runs. We see in Table II that for the SYNTH, MICE and HAR data set DECCS clearly outperforms the next best method. For the PENDIGITS data set, we perform similar to SpectralNet in NMI and outperform it w.r.t. ARI (0.73 vs 0.67). For the MICE data set, we see that all CC methods improve when applied to the AE embedded space, but DECCS is still outperforming them, showing that updating the representation can increase the cluster performance even further. The highest improvement for the real-world data sets can be seen for the HAR data set, where we outperform the next best clustering method (NMI=0.61) by 0.14. The results on the image data sets show that DECCS outperforms all comparison methods on USPS. DECCS performs similar to the DC methods for MNIST, FMNIST, and KMNIST. For MNIST, we are only outperformed by SpectralNet. Interestingly, the CC algorithms that are applied to the embedded space for the image data sets serve as strong baselines, e.g., reaching 0.65 and 0.61 NMI for FMNIST and KMNIST respectively. DECCS outperforms the deep consensus clustering method ConCURL for all image data sets. ConCURL heavily relies on image augmentation and for small, greyscale images there are fewer augmentation invariances available, which might be the reason for ConCurl's poor performance. Further, ConCURL cannot be applied to non-augmentable data, which is why these results are marked as not applicable (n.a.).

VII. DISCUSSION AND CONCLUSION

Noise and outlier points: Currently, we have not considered noise-aware clustering methods, like DBSCAN [39], in our ensembles. DECCS could be extended to include methods like DBSCAN, e.g., by excluding noise and outlier points during the representation update, such that a consensus representation is learned only for inlier clusters.

Consensus representation learning: With DECCS we have introduced the first algorithm to learn consensus representations for CC. In future work, we would like to explore alternative approaches for optimizing the proposed objective in Eq. 2, which could lead to novel approaches to CC.

We have proposed the idea of consensus representations, a novel way of learning a CC by maximizing the agreement between ensemble members using representation learning. Additionally, we have introduced the DECCS algorithm, to the best of our knowledge, it is the first DC algorithm that can use multiple heterogeneous clustering methods to jointly improve the learned representation and clustering results.

References

[1] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 478–487.

[2] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–136, 1982.

[3] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: An unsupervised and generative approach to clustering," in *IJCAI*. ijcai.org, 2017, pp. 1965–1972.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B*, vol. 39, no. 1, pp. 1–22, 1977.

[5] D. Mautz, C. Plant, and C. Böhm, "Deep embedded cluster tree," in *ICDM*. IEEE, 2019, pp. 1258–1263.

[6] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *WIREs Data Mining Knowl. Discov.*, vol. 2, no. 1, pp. 86–97, 2012.

[7] U. Shaham, K. P. Stanton, H. Li, R. Basri, B. Nadler, and Y. Kluger, "Spectralnet: Spectral clustering using deep neural networks," in *ICLR (Poster)*. OpenReview.net, 2018.

[8] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.

[9] A. Strehl and J. Ghosh, "Cluster Ensembles - A Knowledge Reuse Framework for Combining Multiple Partitions," *Journal of Machine Learning Research*, pp. 583–617, 2002.

[10] X. Z. Fern and C. E. Brodley, "Random projection for high dimensional data clustering: A cluster ensemble approach," in *ICML*. AAAI Press, 2003, pp. 186–193.

[11] M. Popescu, J. Keller, J. Bezdek, and A. Zare, "Random projections fuzzy c-means (RPFCM) for big data clustering," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015, pp. 1–6.

[12] J. Bezdek, X. Ye, M. Popescu, J. Keller, and A. Zare, "Random projection below the JL limit," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 2414–2423.

[13] J. R. Regatti, A. A. Deshmukh, E. Manavoglu, and Ü. Dogan, "Consensus clustering with unsupervised representation learning," in *IJCNN*. IEEE, 2021, pp. 1–9.

[14] H. Liu, M. Shao, S. Li, and Y. Fu, "Infinite ensemble for image clustering," in *KDD*. ACM, 2016, pp. 1745–1754.

[15] Z. Tao, H. Liu, J. Li, Z. Wang, and Y. Fu, "Adversarial graph embedding for ensemble clustering," in *IJCAI*. ijcai.org, 2019, pp. 3562–3568.

[16] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[17] S. Monti, P. Tamayo, J. P. Mesirov, and T. R. Golub, "Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data," *Mach. Learn.*, vol. 52, no. 1-2, pp. 91–118, 2003.

[18] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *NIPS*, 2017, pp. 1195–1204.

[19] A. L. Fred and A. K. Jain, "Combining multiple clusterings using evidence accumulation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, 2005.

[20] D. Huang, C.-D. Wang, and J.-H. Lai, "Locally Weighted Ensemble Clustering," *IEEE Transactions on Cybernetics*, pp. 1460–1473, 2018.

[21] X. Z. Fern and C. E. Brodley, "Solving cluster ensemble problems by bipartite graph partitioning," in *Twenty-first international conference on Machine learning - ICML '04*. ACM Press, 2004, p. 36.

[22] T. Li, C. Ding, and M. I. Jordan, "Solving Consensus and Semi-supervised Clustering Problems Using Nonnegative Matrix Factorization," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 577–582.

[23] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," in *Contemporary Mathematics*, 1984.

[24] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," in *IJCAI*. ijcai.org, 2017, pp. 1753–1759.

[25] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3861–3870.

[26] L. Miklautz, D. Mautz, M. C. Altinigneli, C. Böhm, and C. Plant, "Deep embedded non-redundant clustering," in *AAAI*. AAAI Press, 2020, pp. 5174–5181.

[27] E. Aljalbout, V. Golkov, Y. Siddiqui, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *CoRR*, vol. abs/1801.07648, 2018.

[28] M. Jabi, M. Pedersoli, A. Mitiche, and I. B. Ayed, "Deep clustering: On the link between discriminative models and k-means," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 6, pp. 1887–1896, 2021.

[29] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[30] S. Hess, W. Duivesteijn, and D. Mocanu, "Softmax-based classification is k-means clustering: Formal proof, consequences for adversarial attacks, and improvement through centroid based tailoring," *CoRR*, vol. abs/2001.01987, 2020.

[31] M. Chen, K. Q. Weinberger, F. Sha, and Y. Bengio, "Marginalized denoising auto-encoders for nonlinear representations," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014, pp. 1476–1484.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] X. V. Nguyen, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *J. Mach. Learn. Res.*, vol. 11, pp. 2837–2854, 2010.

[34] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[35] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *CoRR*, vol. abs/1812.01718, 2018.

[36] J. J. Hull, "A database for handwritten text recognition research," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, 1994.

[37] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[38] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML (3)*, ser. JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org, 2013, pp. 1139–1147.

[39] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*. AAAI Press, 1996, pp. 226–231.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015.

APPENDIX

A. DATA SETS

Mice Protein Expression (MICE) [37]: Data set consisting of 552 vectors with 77 dimensions and 8 ground-truth clusters. Each vector represents the expression levels of 77 proteins of the mice's cortex.
Pendigits [37]: Data set consisting of 10,992 vectors with 16 dimensions, representing 8 coordinates. The coordinates were gathered during the writing of digits (0 to 9) on a tablet.
Human Activity Recognition (HAR) [37]: Data set consisting of 10,299 vectors with 561 dimensions with records from smartphones and smartwatches. The data set contains six clusters corresponding to different human activities.
MNIST [32]: Data set consisting of 70,000 hand-written digits (0 to 9) with a size of 28×28 pixels.
FMNIST [34]: Data set consisting of 70,000 goods from the Zalando online store. Each sample belongs to one of 10 products and has a size of 28×28 pixels.
KMNIST [35]: Data set consisting of 70,000 Kanji characters (10 different characters) with a size of 28×28 pixels.
USPS [36]: Data set consisting of 9,298 hand-written digits (0 to 9) with a size of 16×16 pixels.

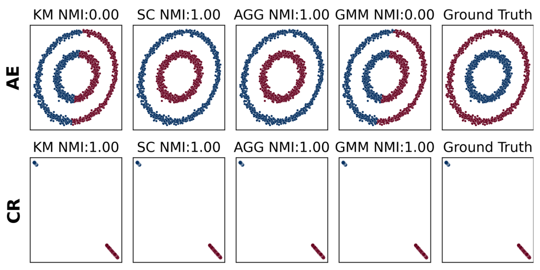


Fig. 9. Consensus representation (CR) learned with DECCS for two not linearly separable clusters. *Upper row*: Initial clustering results on an autoencoder (AE) embedding. Two ensemble members (SC, AGG) can perfectly discover the ground-truth clustering, but the two others (KM, GMM) do not perform better than chance. *Lower row*: Clustering results on a CR learned with DECCS leads to the same, perfect performance for all clustering algorithms

B. EXPERIMENT SETUP

Hardware setup: We trained all DC algorithms on a machine with a single NVIDIA RTX 2080TI GPU (11GB onboard memory), 96 GB RAM, and an Intel(R) Xeon(R) Gold 6130 CPU. All other comparison methods were run on the same machine using only the CPU.
Implementation: We implemented DECCS in PyTorch (https://pytorch.org/). Currently, the generation and approximation of base partitions is done sequentially, but could be further optimized using parallelization. One run of DECCS for MNIST took about one hour, which is in the same order of magnitude as other DC methods, e.g. DCN needed about 40 minutes.
Parameters of cluster ensemble: For the clustering algorithms in the ensemble $\mathcal{E} = \{\text{KM, SC, AGG, GMM}\}$, we used the implementations of the *sklearn* [40] package. For the real-world data sets, we parameterized them using this setting:

```
KMeans(n_clusters=k)

SpectralClustering(n_clusters=k,
                   affinity='nearest_neighbors',
                   n_neighbors=10,
                   assign_labels='kmeans')

AgglomerativeClustering(n_clusters=k,
                        linkage='ward')

GaussianMixture(n_components=k,
                covariance_type='full',
                reg_covar=1e-5)
```

For the synthetic data sets, we set linkage = 'single' for agglomerative clustering.
Consensus Clustering: For the classical methods, we used the same parameterization as suggested by the authors in the original papers ([9], [22], [21], [20]). For the RP-based methods, we determined the subspace dimension with a grid search and report the value with the best average NMI over 10 runs. We evaluated the RP algorithms with an ensemble size of four (size of our ensemble) and 30 (ensemble size used in [11]) and again picked the run with the highest NMI. The parameters of the Expectation Maximization (EM) and FCM algorithms were set according to [10] and [11] respectively. [11] proposed two versions of their RP-based algorithm, we run our experiments with both versions and only report the run with the highest NMI. Aligned with [11], we performed a grid search for q in [5, 100] for both RP-based methods and picked the run with the highest NMI. For LWEA, we set the hyperparameter θ to 0.4, as [20] do for all their experiments. Except for LWEA, other classical methods are implemented using the cluster ensemble package.[6]
Deep Clustering: For the UCI data sets, we determined the learning rate and other AE parameters, like dropout rate [41], using a grid search during the AE pretraining taking the parameters with lowest reconstruction loss. All other AEs were pretrained with a learning rate of 0.001. All AEs were trained using early stopping and the learning rate was reduced by 0.5 if the reconstruction loss reached a plateau. We use for DECCS, VaDE, DEC, IDEC, and DCN a batch size of $|\mathcal{B}| = 256$. For VaDE, DEC, IDEC, and DCN, we use a constant learning rate of 0.0001 for the joint clustering. We train the DC algorithms for 100,000 mini-batch iterations for all data sets using the Adam optimizer [42] as was done in the original papers. For SpectralNet [7], we used for the SYNTH and MICE data set the same parameters as suggested for small data sets. For the other data sets (PENDIGITS, HAR, and image data), we used the same parameters as [7] used for MNIST.
ConCURL [13]: For ConCURL, we used the author's repository. For all data sets, we performed ten runs each. In each run, we trained the algorithm for 300 epochs. To change the basic architecture as little as possible, we transformed gray images into three dimensions by copying the gray color channel. In our experiment, we used PyTorch's resnet18 with a hidden MLP of 2048. We chose SGD as the optimizer with a

[6]https://github.com/827916600/ClusterEnsembles

TABLE III

CLUSTER PERFORMANCE RESULTS MEASURED IN ARI. BEST RESULTS ARE MARKED AS BOLD AND RUNNER-UP IS UNDERLINED. ALL RESULTS ARE GIVEN IN ARI AS MEAN $\pm$ STD OVER 10 RUNS. DECCS $\lambda_{\text{REC}} = 1$ INDICATES THAT $\mathcal{L}_{\text{REC}}$ WAS USED, WHILE $\lambda_{\text{REC}} = 0$ ARE THE RESULTS WITHOUT $\mathcal{L}_{\text{REC}}$.

Method	**SYNTH**	**MICE**	**PENDIGITS**	**HAR**	**MNIST**	**FMNIST**	**KMNIST**	**USPS**
DECCS ($\lambda_{rec} = 1$)	$\underline{0.99} \pm 0.02$	$\mathbf{0.36} \pm 0.04$	$\mathbf{0.73} \pm 0.03$	$\mathbf{0.65} \pm 0.03$	$\underline{0.85} \pm 0.05$	$\underline{0.47} \pm 0.01$	$\mathbf{0.45} \pm 0.01$	$\mathbf{0.78} \pm 0.01$
DECCS ($\lambda_{rec} = 0$)	$\mathbf{1.00} \pm 0.01$	$\underline{0.33} \pm 0.03$	$\underline{0.72} \pm 0.04$	$\underline{0.62} \pm 0.03$	$\underline{0.85} \pm 0.04$	$\underline{0.47} \pm 0.01$	$\underline{0.44} \pm 0.02$	$\underline{0.77} \pm 0.02$
CSPA [9]	0.67 ± 0.00	0.19 ± 0.02	0.64 ± 0.04	0.37 ± 0.02	0.49 ± 0.05	0.40 ± 0.04	0.36 ± 0.02	0.54 ± 0.02
HGPA [9]	0.67 ± 0.00	0.20 ± 0.01	0.47 ± 0.08	0.36 ± 0.05	0.34 ± 0.03	0.30 ± 0.03	0.23 ± 0.03	0.42 ± 0.04
MCLA [9]	0.43 ± 0.08	0.17 ± 0.01	0.61 ± 0.06	0.46 ± 0.03	0.42 ± 0.04	0.36 ± 0.02	0.35 ± 0.02	0.52 ± 0.08
HBGF [21]	0.56 ± 0.02	0.17 ± 0.03	0.64 ± 0.03	0.37 ± 0.04	0.49 ± 0.03	0.39 ± 0.05	0.37 ± 0.03	0.54 ± 0.02
NMF [22]	0.37 ± 0.04	0.19 ± 0.02	0.63 ± 0.05	0.46 ± 0.06	0.47 ± 0.05	0.37 ± 0.03	0.37 ± 0.02	0.59 ± 0.05
LWEA [20]	0.44 ± 0.02	0.18 ± 0.02	0.62 ± 0.03	0.46 ± 0.00	0.50 ± 0.05	0.40 ± 0.02	0.35 ± 0.03	0.63 ± 0.01
RP+EM [10]	0.44 ± 0.00	0.26 ± 0.04	0.42 ± 0.11	0.31 ± 0.07	0.22 ± 0.04	0.29 ± 0.07	0.24 ± 0.05	0.45 ± 0.07
RP+FCM [11]	0.53 ± 0.10	0.15 ± 0.05	0.47 ± 0.02	0.32 ± 0.00	0.12 ± 0.02	0.25 ± 0.01	0.14 ± 0.01	0.20 ± 0.06
AE+CSPA [9]	0.69 ± 0.07	0.23 ± 0.03	0.65 ± 0.02	0.40 ± 0.02	0.82 ± 0.03	0.46 ± 0.02	0.43 ± 0.02	0.64 ± 0.04
AE+HGPA [9]	0.67 ± 0.01	0.23 ± 0.02	0.51 ± 0.08	0.36 ± 0.05	0.47 ± 0.04	0.34 ± 0.04	0.29 ± 0.03	0.45 ± 0.03
AE+MCLA [9]	0.42 ± 0.10	0.23 ± 0.03	0.62 ± 0.07	0.43 ± 0.08	0.78 ± 0.03	0.46 ± 0.02	$\mathbf{0.45} \pm 0.03$	0.68 ± 0.07
AE+HBGF [21]	0.59 ± 0.10	0.22 ± 0.03	0.62 ± 0.05	0.41 ± 0.03	0.80 ± 0.03	0.44 ± 0.02	$\underline{0.44} \pm 0.02$	0.64 ± 0.03
AE+NMF [22]	0.41 ± 0.15	0.25 ± 0.04	0.66 ± 0.07	0.41 ± 0.03	0.75 ± 0.08	$\mathbf{0.48} \pm 0.03$	$\underline{0.44} \pm 0.05$	0.74 ± 0.09
AE+LWEA [20]	0.45 ± 0.04	0.25 ± 0.04	0.62 ± 0.05	0.41 ± 0.09	0.81 ± 0.02	$\underline{0.47} \pm 0.01$	$\mathbf{0.45} \pm 0.05$	$\underline{0.77} \pm 0.06$
AE+RP+EM [10]	0.46 ± 0.04	0.31 ± 0.05	0.38 ± 0.08	0.38 ± 0.03	0.68 ± 0.09	0.42 ± 0.03	0.40 ± 0.04	0.48 ± 0.03
AE+RP+FCM [11]	0.54 ± 0.16	0.22 ± 0.03	0.37 ± 0.04	0.35 ± 0.04	0.29 ± 0.08	0.30 ± 0.03	0.18 ± 0.04	0.21 ± 0.05
SpectralNet [7]	0.53 ± 0.08	0.15 ± 0.04	0.67 ± 0.08	0.46 ± 0.09	$\mathbf{0.93} \pm 0.00$	$\underline{0.47} \pm 0.00$	0.42 ± 0.04	0.67 ± 0.05
DEC [1]	0.50 ± 0.03	0.27 ± 0.03	0.61 ± 0.04	0.39 ± 0.11	0.81 ± 0.02	0.44 ± 0.02	0.39 ± 0.01	0.73 ± 0.01
IDEC [24]	0.49 ± 0.03	0.29 ± 0.03	0.62 ± 0.04	0.40 ± 0.11	0.82 ± 0.03	0.46 ± 0.03	0.41 ± 0.03	0.74 ± 0.01
DCN [25]	0.44 ± 0.08	0.27 ± 0.04	0.60 ± 0.04	0.36 ± 0.11	0.79 ± 0.06	0.45 ± 0.03	0.38 ± 0.05	0.70 ± 0.07
VaDE [29]	0.49 ± 0.13	0.25 ± 0.05	0.61 ± 0.04	0.38 ± 0.10	0.78 ± 0.06	$\mathbf{0.48} \pm 0.02$	0.40 ± 0.02	0.70 ± 0.06
DeepECT [5]	0.47 ± 0.12	0.27 ± 0.06	0.60 ± 0.04	0.41 ± 0.12	0.76 ± 0.06	0.44 ± 0.05	0.36 ± 0.05	0.67 ± 0.09
ConCURL [13]	n.a.	n.a.	n.a.	n.a.	0.48 ± 0.05	0.34 ± 0.02	0.20 ± 0.03	0.33 ± 0.02

learning rate of 0.015 and set the batch size to 128. The alpha parameter was set to 0, and the beta and gamma parameters were set to 1. In the experiment, we set NCE-temp to 0.085 and NCE-k to 4096. For the hyperparameters, we followed the hyperparameters of CIFAR-10 available in their repository. For image augmentation, we used random rotations between -10 and 10 degrees, translations between 0 and 0.1, scaling between 0.6 and 1.2, and shearing between -10 and 10.

A.4. Non-Redundant Image Clustering of Early Medieval Glass Beads

Title	Non-Redundant Image Clustering of Early Medieval Glass Beads
Authors	Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch, Christian Böhm and Claudia Plant.
Submitted for review at:	The ADS Track of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023. Submitted on 03.02.2023.

Division of Work

Lukas Miklautz: problem identification, definition and refinement; formulating the loss function and optimization strategy; investigation, preprocessing and cleaning of the glass bead data and integration into the pipeline; setting up the pretraining and cluster analysis pipeline; experiment design; conducting (deep) non-redundant clustering experiments and evaluation; writing the main parts of the paper;

Andrii Shkabrii: Investigation, preprocessing and cleaning of the glass bead data and integration into the pipeline; Implementing and conducting the pretraining experiments on the meluxina compute node; documentation of the experiments in the paper;

Collin Leiber: Writing of methods and related work sections on non-redundant clustering; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft.

Bendeguz Tobias: Conceiving the initial idea; identifying relevant glass beads; project management and setting up data collection pipeline; providing domain expertise in archaeology and discussion of clustering results; writing of the related work section; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Benedict Seidl: Setting up data collection pipeline; documenting, measuring and photographing the beads; providing domain expertise in archaeology and discussion of clustering results.

Elisabeth Weissensteiner: Setting up data collection pipeline; documenting, measuring and photographing the beads; providing domain expertise in archaeology and discussion of clustering results.

Andreas Rausch: Setting up data collection pipeline; documenting, measuring and photographing the beads; providing domain expertise in archaeology and discussion of clustering results.

Christian Böhm: Suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Claudia Plant: Conceiving the initial idea; regular discussions of candidate methods and potential experiments; suggestions for enhancement and improvements; periodic review of drafts for the paper; refining the final draft; mentoring and supervision.

Abstract	Glass beads were among the most common grave goods in the Early Middle Ages, with an estimated number in the millions. The color, size, shape and decoration of the beads are diverse. Accordingly, archaeological classification systems are often subjective, complex and usually limited to one burial field. An automated, objective, extensible and reproducible classification system is highly desirable to facilitate the analysis of the archaeological artefacts. Using such a system would enable archaeologists to get an objective overview of the different types of glass beads from the early Medieval Ages. From a data mining perspective this task can be tackled using cluster analysis. As we want to find multiple clusterings (e.g., color, size, shape and decoration) from high-dimensional image data, we make use of deep non-redundant clustering to identify multiple, meaningful clusterings of glass bead images and learn a classification system in a data-driven way. During the cluster analysis we address several challenges associated with the data and as a result identify high-quality clusterings that show great overlap with archaeological domain expertise. To the best of our knowledge this is the first application of non-redundant image clustering for archaeological data.
Thesis-Reference	[MSL+23]

Non-Redundant Image Clustering of Early Medieval Glass Beads

Lukas Miklautz
lukas.miklautz@univie.ac.at
Faculty of Computer Science,
University of Vienna,
UniVie Doctoral School Computer Science
Vienna, Austria

Andrii Shkabrii
andrii.shkabrii@univie.ac.at
Faculty of Computer Science,
University of Vienna
Vienna, Austria

Collin Leiber
leiber@dbs.ifi.lmu.de
Institute for Informatics,
LMU Munich,
MCML
Munich, Germany

Bendeguz Tobias
Bendeguz.Tobias@oeaw.ac.a
Austrian Academy of Sciences,
Institute for Medieval Research
Vienna, Austria

Benedict Seidl
Austrian Academy of Sciences,
Institute for Medieval Research
Vienna, Austria

Elisabeth Weissensteiner
e.weissensteiner@univie.ac.at
Austrian Academy of Sciences,
Institute for Medieval Research
Vienna, Austria

Andreas Rausch
Austrian Academy of Sciences,
Institute for Medieval Research
Vienna, Austria

Christian Böhm
christian.boehm@univie.ac.at
Faculty of Computer Science,
University of Vienna
Vienna, Austria

Claudia Plant
claudia.plant@univie.ac.at
Faculty of Computer Science,
University of Vienna, ds:UniVie
Vienna, Austria

ABSTRACT

Glass beads were among the most common grave goods in the Early Middle Ages, with an estimated number in the millions. The color, size, shape and decoration of the beads are diverse. Accordingly, archaeological classification systems are often subjective, complex and usually limited to one burial field. An automated, objective, extensible and reproducible classification system is highly desirable to facilitate the analysis of the archaeological artefacts. Using such a system would enable archaeologists to get an objective overview of the different types of glass beads from the early Medieval Ages. From a data mining perspective this task can be tackled using cluster analysis. As we want to find multiple clusterings (e.g., color, size, shape and decoration) from high-dimensional image data, we make use of deep non-redundant clustering to identify multiple, meaningful clusterings of glass bead images and learn a classification system in a data-driven way. During the cluster analysis we address several challenges associated with the data and as a result identify high-quality clusterings that show great overlap with archaeological domain expertise. To the best of our knowledge this is the first application of non-redundant image clustering for archaeological data.

CCS CONCEPTS

• **Theory of computation → Unsupervised learning and clustering**.

KEYWORDS

clustering, deep learning, digital humanities, archaeological data

ACM Reference Format:
Lukas Miklautz, Andrii Shkabrii, Collin Leiber, Bendeguz Tobias, Benedict Seidl, Elisabeth Weissensteiner, Andreas Rausch, Christian Böhm, and Claudia Plant. 2023. Non-Redundant Image Clustering of Early Medieval Glass Beads. In *Proceedings of (Submitted to KDD 2023).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

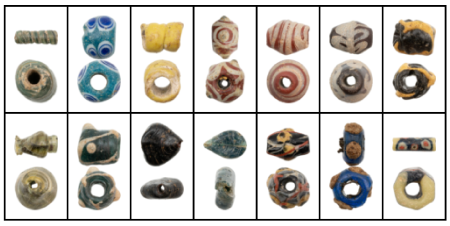


Figure 1: Diverse sample of glass beads varying in color, shape and decoration. Recorded from a side view and a top view.

1 INTRODUCTION

The field of archaeology is currently going through a phase of digitalization, with an increasing amount of archaeological data every year. The growth in data calls for interdisciplinary research between archaeologist and data scientists. There have already been successful interdisciplinary collaborations in the restoration of ancient texts [2], the rejoining of oracle bones [38] or the analysis of petroglyphs [40]. In this work, we want to contribute to the analyzes of glass beads of the early Middle Ages (400-900 AD) using modern data mining techniques. The largest production areas of glass beads at that time were in the Middle East. From there, the beads reached even the most remote areas of Europe. During that time glass beads were among the most common grave goods and their number is estimated to be in the millions [34]. This large amount of potential data makes it impossible for archaeologists to analyze it by hand and shows the need for data mining and machine learning techniques. Another issue is that the glass beads can vary highly in regards of color, size, shape and decoration. See Figure 1 for such a diverse sample of glass beads.[1] The high variety of different beads leads to the problem of subjectiveness in many of the used archaeological classification systems, i.e., depending on the used classification system the same glass bead could be assigned to different groups. An example is shown in Figure 2, where two different sources describe the same bead either as *speckled* [12] or as *confetti*[2]. Both descriptions imply a focus on different attributes. The adjective *speckled* would attribute more importance to the dotted pattern, whereas *confetti* emphasizes both the dotted pattern and the varying color of each dot.

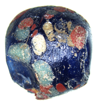


Figure 2: Example of subjectiveness in existing classification systems for glass beads. The image shows a glass bead that is described in the literature either as *speckled* or as *confetti.*

We avoid the issue of subjectivity by learning the classification system directly from the data. Such a classification system has many benefits. It enables scientists to get an objective overview of the different types of glass beads from the early Middle Ages and is applicable to large amounts of data.

We use *non-redundant clustering* to learn a data-driven classification system. Non-redundant clustering algorithms can find multiple clusterings that are mutually non-redundant, i.e., share as little information as possible, while still preserving important cluster information [28]. In Figure 3, we show a non-redundant clustering of *almond* (a.k.a. *melon-seed*) glass beads. These can be clustered according to their colors (black, turquoise or light gray) or their shapes (leaf-, arrow-, cylindrical-shaped). Both of these clusterings are equally meaningful. In the context of glass beads, clusterings according to color, shape, decoration and size can be expected [32]. The assumption of non-redundant clustering algorithms that different clusterings share almost no information is consistent with the manufacturing process where, e.g., color and shape can be varied independently from each other.

We collected and photographed more than 4,500 beads of early medieval cemeteries in Austria. The beads are mostly made of glass, but some are also made of other materials. Each object was photographed once from the side and once from the top, resulting in more than 9000 images. Examples are shown in Figure 1. We face several challenges while analyzing these images. The data is very unbalanced in terms of different attributes, e.g. we have many

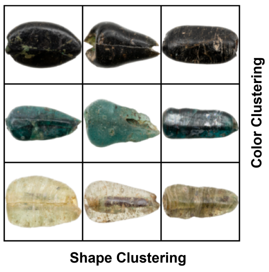


Figure 3: Example of non-redundant clusterings of *almond* (a.k.a. *melon-seed*) glass beads according to shape and color.

visually simple beads and only few beads with elaborate decorations. During the recording of the beads, various inaccuracies occur, such as differing rotations, recording angles and reflections. In addition, the collected beads themselves contain outliers. For example, we have instances that are damaged or are not made of glass, but of rolled metal. Common damages are strong corrosion or broken off parts. In Appendix B in Figure 13 we show different versions of *barrel* glass beads, which are the most common ones in the data. While they do not have any elaborate decorations, they can still vary significantly. In Figure 13 we also show some corruptions in images (4)-(7) , like restoration artifacts, broken parts and corrosion. These corruptions are also present for other glass bead types.

Our novel pipeline enables us to overcome most of these difficulties and thus provide a high-quality analysis of early medieval glass beads. The pipeline consist of data selection, representation learning [3] with deep non-redundant clustering [25] and evaluation of the results. Overall we present the following contributions:

- First application of non-redundant image clustering for archaeological data.
- Presentation of a novel and high-quality data set for the data mining and machine learning community.
- Extension of the deep Embedded Non-Redundant Clustering (ENRC) algorithm [25] to learn augmentation invariances, handle multiple views and multiple modalities.
- Step-by-step guideline showing how to apply deep non-redundant clustering techniques in realistic scenarios with outliers and absence of ground truth labels.
- Extraction of interesting and high-quality clusters that are easy to interpret and confirm existing knowledge of experts.

2 OVERVIEW

Before we explain the data and our approach in detail we give a high level overview in Figure 4. Similar to the KDD process [9] our pipeline comprises data selection, data preprocessing, representation learning with deep non-redundant clustering (combined transformation and data mining steps) and the evaluation of the results. We explain the data selection in Section 3.1, discuss some identified challenges for mining the data in Section 3.2 and break down our preprocessing steps in Section 3.3. In Section 4 we introduce some necessary background before we explain the cluster analysis in Section 5. The cluster analysis comprises the autoencoder pretraining

[1]All shown images are resized to the same scale for illustration, if not otherwise indicated.
[2]See the entry at the online archaeological data base https://earlymedievaleurope.org/image.php?i=Confetti.

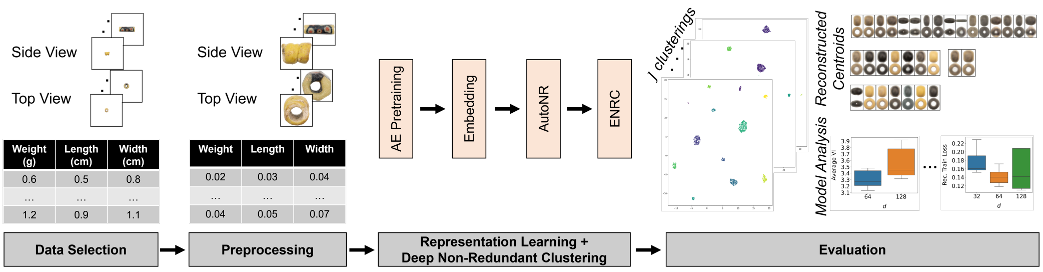


Figure 4: Our pipeline comprises data selection (gathering and recording of glass beads), data preprocessing (resizing images and scaling tabular features), representation learning with deep non-redundant clustering and the evaluation of results (model analysis and interpretation of clusterings).

(Sections 5.1, 5.3) with image augmentations (Section 5.2) for the deep non-redundant clustering (Sections 5.4 - 5.5). We evaluate our results in Section 6 with quantitative and qualitative experiments in Sections 6.2 and 6.3 respectively. We make all data, clustering results and our code available at https://tinyurl.com/cluster-glassbeads.

3 DATA

3.1 Data Selection

We collected 4,669 beads in the region of present-day eastern Austria south of the Danube river. The largest portion comes from the two cemeteries of *Mödling-An der Goldenen Stiege* (550 inhumations) and *Vienna-Csokorgasse* (720 inhumations). Others were recorded from various smaller cemeteries, dating from the 4th to the 9th century AD. We use beads from multiple sites to ensure a regional and diachronic cross-section of bead forms. Therefore, we collected the beads from various museums in Austria. Most of them are glass beads, in rare cases also non-glass beads occur that were used as pendants. These pendants are quite varied in the material, as it was common to reuse and recycle objects, e.g. shells, parts of glass vessels or metal objects. For each glass bead two pictures with a resolution of 4000 × 4000 pixels were taken; one from the top view aligned with the hole of the bead and one from a side view that is orthogonal to the hole, as shown in Figure 1, leading to 9,338 images. Additionally, the weight, length and width of each bead was measured.

3.2 Challenges

From analyzing the data and the recording process, we identified the following challenges for mining the data.

Cluster imbalances: The data is very unbalanced in terms of various aspects. There are many *barrel* shaped glass beads (shown in Figure 13), which have a small size and weight. On the left side of Figure 5 we see that the distribution of the area in cm^2 (width × length) and weight in grams (g) is highly skewed as indicated by the density plots on each axis. We also see a strong positive correlation between the area and weight, which makes sense as almost all beads are made of the same material. The right side of Figure 5 shows the two smallest and two largest objects, revealing the diversity of the samples. Additionally, there are many visually simple beads and only some glass beads with elaborate decoration. This has an impact on the color distribution of the data set, as shown in Figure 6, where we plot the main colors present in the data. Figure 6 clearly shows that yellow and black are overall the dominating colors. In the Appendix C.1, we explain how this plot was created and show, additionally in Figure 17 a two dimensional embedding of the data according to the main colors.

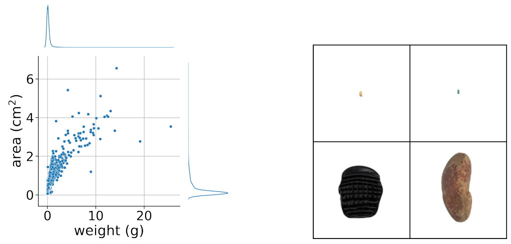


Figure 5: (Left) Scatter plot of size measured as area and weight of beads. The density plots on each axis indicate that the data is highly imbalanced, mostly consisting of small and light beads. (Right) The two smallest beads in the top row vs. the two largest ones in the bottom row. Here, the images are not resized to the same scale to show the large relative difference in size.

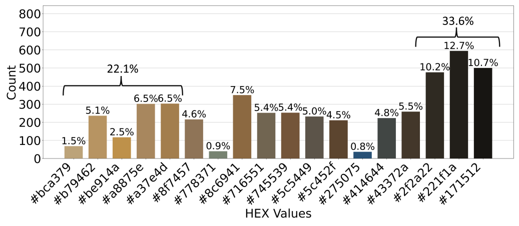


Figure 6: Distribution of the main colors in all images as HEX values, ordered from brighter to darker colors. It shows a high number of yellow and black beads, summing up to 22.1% and 33.6% of total images respectively.

Corrupted instances due to aging and restoration: Many beads show signs of aging due to being in the earth for over 1,000 years. This leads to issues with the appearance of the beads. Many have broken parts that influence not only the shape, but also the color as the surface color might be different as the color of the subsurface material. In Appendix B in Figure 14 we show a sample of broken glass beads. Some are not properly recognizable anymore (last two images on the right), while others have only small missing parts. Another age related issue is corrosion, which also changes the overall color of the glass beads (see bead (7) in Figure 13). Some beads have been restored, e.g. by using varnish to polish the surface. Unfortunately, in some cases the remaining varnish dried between the holes, see bead (4) in Figure 13.

Recording artifacts: Issues from data recordings are present as it is not possible to position diversely shaped beads in the same angle. Additionally, the orientation of each glass bead is not unique and varies between glass beads of the same type, e.g. they might be flipped horizontally or vertically or are rotated in some way.

Small sample size: The data set is quite small ($N = 4,669$), given the diversity of shapes and decorations. While the main glass bead types are well covered, there are still a lot of combinations missing. Additionally, using deep learning requires a large amount of data to learn good features.

Image and tabular data: For each glass bead we have two images, a top view and a side view. Additionally, we have measurements about the weight, length and width. In order to learn a good representation we need to combine these features into a single meaningful representation for clustering.

Outliers: Many necklaces in the early medieval period contained other objects beside glass beads as pendants. Recycling and reusing objects as part of necklaces was quite common, which introduces some beads made from other materials. In Figure 15 in the Appendix we show a set of outlier objects. They are mainly outliers in shape and size, but not necessarily in color, an information that is useful, e.g. for studying peoples' color preferences. We decided to keep them in the data set as they provide additional information.

Unknown number of clusterings: The exact number of ground truth clusterings and number of clusters in each clustering is not known. Based on domain knowledge we have some idea on the number of clusterings as some meaningful partitions like color, shape, decoration or size are known. However, it is not clear how many different decorations there are or how many shapes, as it can be quite subjective to decide on the differences.

Partially overlapping clusterings: Due to the already mentioned cluster imbalances, the different clusterings are not completely independent of each other. Almost all barrel shaped glass beads are either black or yellow, correlating their color and shape clustering. This is an issue for non-redundant clustering algorithms, which assume that the clusterings are independent of each other.

3.3 Preprocessing

Given a raw image $\mathbf{x}$ of our data set $\mathbf{X}$ we perform the following preprocessing steps. We crop and resize all images to a size of 64×64 to account for the large relative differences in size between the beads as illustrated in Figure 5. After resizing all images to the same size we cannot infer the relative size anymore, but we have the weight, length and width measurements to account for that. The image pixels are then rescaled from 0 - 255 to 0 - 1 intervals and scaled using ImageNet [7] mean and standard deviation statistics. After preprocessing, all images of our data set $\mathbf{X}$ are of shape 64×64, resulting in a dimensionality of $D = 4096$. The tabular attributes $\mathbf{A}$ include weight, length and width measurements. We use min-max normalization to rescale the tabular features to an interval of 0 to 1, because they are of different metrics (centimeters vs. grams).

4 BACKGROUND AND RELATED WORK

In the following sections we give an overview of existing work in non-redundant clustering (Section 4.1), deep clustering (Section 4.2) and classification systems for glass beads (Section 4.3), relate them to the challenges identified in Section 3.2 and provide necessary background to understand our analysis pipeline.

4.1 Non-Redundant Clustering

Alternative and non-redundant clustering methods have been actively researched for about two decades, an overview can be found in [28]. Non-redundant clustering extracts multiple clusterings that are as different as possible, while still being informative. Typically, this involves assigning an individual subspace to each clustering that contains the features relevant for the respective clusters.

Although we have some theories about which meaningful groupings might be present in the glass beads data, such as a clustering solution each for color and shape, we do not have ground truth knowledge overall. Therefore, we would like to limit ourselves to methods that are able to determine a suitable number of clusterings and clusters per clustering automatically. In this way, we hope to gain insights that we might have missed in a manual analysis. Potential algorithms are ISAAC [37], MISC [35], NrDipmeans [22] and AutoNR [17]. ISAAC and MISC both use a combination of Independent Subspace Analysis (ISA) [33] and the Minimum Description Length (MDL) [11, 31] to find subspaces suitable for clustering. Within these subspaces, ISAAC then uses a combination of EM-clustering [6] and MDL to automatically identify high-quality clusters. MISC, on the other hand, uses graph regularized semi-negative matrix factorization [8] and bayesian k-Means [36] for this purpose. The problem with both methods is that identifying the subspaces through ISA and MDL has a complexity of $O(Nd^3)$ [37], where N is the number of data points and d the dimensionality of the data. Therefore, these methods are not efficient for large, high-dimensional data sets. NrDipmeans and AutoNR both utilize the non-redundant clustering algorithm NrKmeans [21], which, given the total number of clusters, iteratively executes k-Means in each subspace and then optimizes those subspaces using eigenvalue decompositions. The eigenvalue decomposition has a theoretical complexity of $O(J^2d^3)$[21], where J equals the number of subspaces, but since we compute eigenvalue decompositions not for the entire feature space but only for each pair of subspaces i and j, the actual dimensionality $(m_i + m_j)$ is usually much smaller than d. Furthermore, for larger data sets, $J^2 \ll N$ usually applies. [17] has shown in a runtime analysis that NrDipmeans and AutoNR are better suited for high-dimensional data than ISAAC and MISC. Further, in contrast to using ISA, the subspaces and cluster assignments are able to influence each other and are not discovered one after the other. Here, the definition of a noise space plays a crucial role. This is a subspace that only contains a single cluster and accordingly includes no cluster-relevant structures. Given the total number of subspaces, NrDipmeans uses the Dip-test of unimodality [13] to decide whether or not to split a cluster. AutoNR does not even need to know the number of subspaces a priori, but can use a maximum threshold to speed up the search. Therefore, subspaces and clusters within subspaces are repeatedly split and merged. After each operation, the resulting clusterings are evaluated based on their MDL costs and the process is repeated with the best intermediate result.

Another feature that we would like to support is the identification of outliers. These are samples that do not fit any of the existing clusters. In the case of non-redundant clustering, it is important to note that samples declared as outliers in one clustering may be inliers in another. Therefore, one cannot simply run a common outlier detection algorithm (see e.g. [4]) as a preprocessing step. While it is possible to perform outlier detection separately in each identified subspace, we find it more advantageous if the outlier detection has a direct impact on the definition of the clusters. To the best of our knowledge, AutoNR is the only non-redundant clustering algorithm capable of doing this. For this purpose, it again uses the MDL costs. In summary, AutoNR best meets our requirements regarding the automatic identification of clusterings, runtime and outlier identification.

4.2 Deep Clustering

Although AutoNR has shown a better runtime performance than other non-redundant clustering methods, it still has problems with very complex data. In particular, it cannot feasibly process high-dimensional images directly. In the analysis of images, the research field of deep clustering has achieved very good results in recent years. Deep clustering is the combination of deep learning and clustering to learn representations that are improving clustering performance. For recent surveys see [1, 27, 39]. Many deep clustering algorithms make use of an autoencoder that is first pre-trained using a reconstruction loss. In the resulting embedding an arbitrary non-deep clustering algorithm is executed. This initial clustering result is then improved by the deep clustering algorithm by updating both the embedding and the cluster labels. Due to this procedure, deep clustering algorithms can be combined with many existing algorithms or even ensembles of them [26]. Although there are deep clustering algorithms that can work with multiple clusterings (e.g. [25]) as well as those that are able to determine the number of clusters (e.g. [16]), the combination of these two techniques has not yet been extensively studied. Most existing deep clustering methods, are based on k-means and return only a single clustering solution, which is not suited to our data, where we have multiple valid clusterings. The deep **E**mbedded **N**on-**R**edundant **C**lustering algorithm ENRC [25] can find multiple clustering solutions. ENRC has shown superior performance for common image benchmark data sets against its non-deep competitors like NrKmeans or ISAAC. ENRC learns multiple non-redundant feature spaces inside an embedded space, e.g., an autoencoder, using gradient descent. Similar to NrKmeans, ENRC can determine the dimensionality of the relevant clustered spaces automatically. Unfortunately, ENRC has, up until now, only been applied to benchmark data sets where the number of ground truth clusterings is known a priori. We circumvent this issue by using AutoNR instead of NrKmeans as an initialization to ENRC. Since AutoNR is based on NrKmeans, this can be done in a natural way. While NrKmeans and AutoNR use hard subspace assignments for each non-redundant clustering, ENRC uses soft feature assignment weights. This allows for a partial overlap between the extracted clusterings, which is important as our data is biased.

4.3 Glass Beads Taxonomy and Classification

Existing work in archaeology relies on handcrafted and highly subjective characterizations for classification systems of beads. Considering the number and potential informative value of glass beads, previous approaches for classification are astonishingly short-sighted and, above all, regionally and chronologically conceived. In most cases, beads from individual cemeteries were taken as examples and examined. The main classification criteria are the size, shape, decoration and color [30, 32]. How strongly each criteria influences the determined bead type depends on the individual examiner. Categorising the shapes is quite subjective as the boundaries between different shape types like *cylindrical* and *barrel-shaped* or *melon seed-shaped* or *almond-shaped* are fluid, which can also be seen in the variety of shapes present for the same bead type in Figure 3.

Surprisingly, this also applies to the way color is addressed. In many cases, basic colors such as *red*, *blue* or *green* are used subjectively. For color values in between, expressions such as *green-black* or *black-green* are used. From a scientific point of view, this is incomprehensible, since color values in particular are perceived differently by everyone. Sometimes different color scales are used, such as Munsell's Soil Color Charts, Munsell Bead Color Book or other color-coding systems [5]. Here, however, no care is taken to ensure that the color determination is carried out under constant conditions (e.g. type of light source). Further, criteria such as ornamentation (color overlays) play a decisive role in the classification. Differences or nuances that could have arisen from the manufacturing process are not taken into consideration. Even in this case, there is no explicit approach, so sometimes completely different terminology is used although the same decor is described [12]. A researcher may come up with an attempt to distinguish *wavy lines* from *zigzag lines* or beads from each other considering the number of dots. Thus, it is left to the individual processor to decide whether, for example, a cylindrical bead with a wavy line and three dots in between and one with only two dots form two different types. In the following Section we explain how to automatically learn a classification system for glass beads using cluster analysis.

5 CLUSTER ANALYSIS

We account for the issues described in Section 4.3 by using deep non-redundant clustering to extract several meaningful clusterings from the data, which makes it easily adaptable if new data arrives, e.g., from new archaeological sites. In the following sections we describe how we perform the cluster analysis, which corresponds to the representation learning and deep non-redundant clustering steps in Figure 4. An overview of the used notation can be found in Table 2.

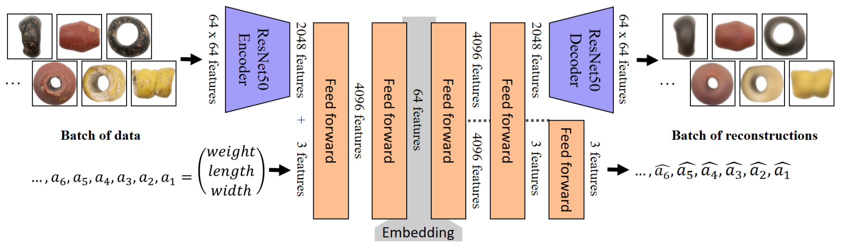


Figure 7: Illustration of the Mixed Convolutional Autoencoder (MCAE) architecture. MCAE takes the preprocessed images and tabular features as input, combines them in a common embedding and reconstructs each modality (image and tabular) with separate decoders.

5.1 Autoencoder Architecture

Similar to other deep clustering approaches [1, 27] and the pretraining strategy of ENRC we use an autoencoder to learn good initial features for clustering. The autoencoder consists of an encoder enc, which learns to embed the input vector $\mathbf{x}$ such that $\mathbf{z} = \text{enc}(\mathbf{x})$ and a decoder dec, which learns a reconstruction $\hat{\mathbf{x}}$ from $\mathbf{z}$ by minimizing the reconstruction loss w.r.t. the input. A common choice for the reconstruction loss is the mean squared error between the reconstruction and the input vector, i.e., $\|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$. In order to not simply copy the input, the autoencoder has a bottleneck layer of dimensionality d which is smaller than the original data dimensionality D. A vanilla autoencoder usually just takes a single instance as input. We, on the other hand, need to combine the image of each bead with its corresponding numerical measurement values to learn a good representation for clustering.

As we are dealing with image data we chose to design a Mixed Convolutional Autoencoder (MCAE) based on the ResNet [14] architecture to also use the tabular features $\mathbf{A}$. The MCAE consists of a ResNet50 encoder and decoder with feed forward autoencoder bottleneck. The feed forward encoder takes the features extracted from the images via the ResNet encoder and concatenates them with the numerical measurements of weight, length and width, denoted as $\mathbf{a}$. The feed forward encoder projects the combined features to a lower dimensional embedded space $\mathbf{z}$ and passes these features to a feed forward decoder to reconstruct the numerical features and to a separate feed forward decoder that passes its features to the ResNet50 decoder to reconstruct the image. This leads to an encoder $\mathbf{z} = \text{enc}(\mathbf{x}, \mathbf{a})$ that takes the image $\mathbf{x}$ and its attributes $\mathbf{a}$ as input and produces their reconstructions via $\hat{\mathbf{x}}, \hat{\mathbf{a}} = \text{dec}(\mathbf{z})$. The reconstruction loss for all images and its attributes is then simply

$$\mathcal{L}_{\text{rec}} = \sum_{\mathbf{x},\mathbf{a} \in \mathbf{X},\mathbf{A}} \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2 + \|\hat{\mathbf{a}} - \mathbf{a}\|_2^2. \quad (1)$$

Figure 7 shows an illustration of MCAE and its encoder and decoder. For pretraining we reconstruct each view per bead separately and only later when we cluster the data we combine the views by averaging them.

5.2 Image augmentations

We make use of color and non-color image augmentations to account for several of the challenges mentioned in Section 3.2. We apply random horizontal and vertical flipping together with random affine transformations like rotations, translations and shearing to account for biases that were introduced during the data recording. We use random resized crops in a small interval of 90% to 100% of the image size to account for beads that are broken or have missing parts around their edges. We do not use smaller crops as we still want to preserve the shape information of each bead. The non-color augmentations are independently applied to each view $x_{\text{top}}, x_{\text{side}}$ (top and side view image respectively) for a bead $\mathbf{x}$. We denote these non-color augmentations as aug_{x_t} for $t \in \{\text{top}, \text{side}\}$. To account for the color bias we use random inversion combined with random color jittering, changing the contrast, saturation, brightness and hue of each image. The random color augmentations are applied with a probability of 50%, to also keep the original colors present. Importantly, we use the same color augmentation for each bead across views to keep the color information consistent. We denote the color augmentation for a bead $\mathbf{x}$ as color_x. In Figure 16 we show an example of the discussed augmentations on a sample of beads. We see that we receive more diverse colors and beads with slightly missing parts (e.g., first image from top left). The detailed augmentation setting can be found in Table 3 in the Appendix. We use the image augmentations during the pretraining and clustering steps, as we explain next.

5.3 Autoencoder Pretraining and Embedding

We pretrain the MCAE on the data with image augmentations using cosine annealing [18] with linear warmup [10, 20] and the Adam optimizer [19] with weight decay. We minimize the mean squared error between the augmented images and their reconstructions. From the pretrained autoencoders we select the one with the lowest reconstruction loss on a held out test set and embed the views per bead with $\mathbf{z}_{\text{top}} = \text{enc}(\mathbf{x}_{\text{top}}, \mathbf{a})$ and $\mathbf{z}_{\text{side}} = \text{enc}(\mathbf{x}_{\text{side}}, \mathbf{a})$. After that we average the embedded views to get one combined embedding per instance, hereby referred to as $\mathbf{z} := (z_{\text{top}} + z_{\text{side}})/2$. We apply AutoNR on the averaged embedding of the full data set $\mathbf{Z}$ to estimate the number of clusters and get the initial parameters for ENRC.

5.4 Estimating the number of clusterings

We briefly discuss some details of NrKmeans and AutoNR to explain how the initial clusterings are obtained. NrKmeans is able to identify multiple clusterings with an arbitrary orientation in the feature space. For this purpose, it uses an orthonormal matrix $V \in \mathbb{R}^{d \times d}$

that rotates the embedded data. Each subspace j is assigned a projection matrix $P_j \in \mathbb{N}^{d \times m_j}$, where m_j defines the dimensionality of the subspace, that specifies the relevant features of the rotated space. Within each subspace defined as $Z_j = \{zVP_j | z \in Z\}$, the labels and cluster centers of the corresponding clustering are updated. The updated cluster properties can then in turn be used to optimize V and P_j such that the sum of the distances between data points and cluster centers is globally minimized. Therefore, an eigenvalue decomposition is performed for each pair of subspaces. Structures that do not fit any clustering are assigned to the noise space. Updating the cluster structures as well as V and all P_j is repeated until the procedure converges. AutoNR uses a greedy heuristic to find a NrKmeans result that minimizes the MDL costs, here different parameterizations are explored. MDL [11, 31] is an unsupervised metric that specifies how many bits are required to encode a certain data model. Fewer bits indicate that more structures have been identified in the data that can be used for compression. This typically corresponds to a better clustering quality. The exact details of the encoding are described in [17]. More details on the cluster estimation steps are explained in Appendix D.

5.4.1 Identify Outliers. The ability of AutoNR to evaluate clustering results by their MDL costs also allows outliers to be identified without requiring additional input parameters. For this purpose, the encoding strategy is used to check for the points farthest from their respective cluster center whether it is cheaper to encode them separately or as part of the cluster. If we achieve a higher compression by considering them individually, they are declared as outliers. Importantly, the definition of outliers can change after each update of the cluster structures.

5.5 Deep Embedded Non-Redundant Clustering

Given the learned cluster centers μ_j for each subspace j, cluster assignments and projection matrices V and P_j from AutoNR, we can initialize ENRC. First, we convert the discrete subspace projection matrices P_j to the differentiable feature importance vectors β_j of ENRC. The feature importance vectors weigh each dimension p continuously with a scalar weight $\beta_j[p]$ such that p belongs partially to clustering j. The β_j weights are all positive and sum over all J clusterings in a single dimension to one, such that $\sum_j^J \beta_j[p] = 1$. They are defined via the soft-max function on a trainable $J \times d$ parameter matrix B, s.t.:

$$\beta_j[p] := \frac{\exp(B[j,p])}{\sum_{i=1}^{J} \exp(B[i,p])}.$$

We convert the discrete projection matrices of one clustering P_j to the soft feature weights by initializing the weights of the assigned subspace dimensions to 0.9 and distribute the remaining weights equally to the other clusterings with $0.1/(J-1)$. We use the β_j-weights to define the β_j-weighted squared euclidean distance $||u-v||^2_{\beta_j} := \sum_{i=1}^{d} \beta_j[i](u[i]-v[i])^2$ for two real vectors u, v. Using this adapted distance we can learn soft feature space weights and weigh the importance of each dimension for each clustering separately. The initial matrix V and cluster centers μ_j are then passed from AutoNR to ENRC directly. Given the initial parameters, ENRC optimizes the autoencoder parameters together with V, μ_j and B to learn a deep non-redundant clustering.

During the optimization of ENRC we want to become invariant between the two views $\mathbf{x}_{\text{top}}, \mathbf{x}_{\text{side}}$ of each bead $\mathbf{x}$ and their augmentations. The invariance is only desired against the non-color transformations specified in Section 5.2, like rotation, flipping and cropping. The color information for clustering should be preserved, which is why we apply the same color augmentation to each view per bead. Note that the color augmentation is only applied with a probability of 0.5, so we also have beads in their original color. We adapt the original ENRC objective to account for the desired invariances. The new ENRC loss is then defined as $\mathcal{L}_{\text{enrc}} =$

$$\sum_{j=1}^{J} \sum_{k=1}^{K_j} \sum_{z_{\text{top}}, z_{\text{side}} \in C_{j,k}} ||V^T z_{\text{top}} - V^T \mu_{j,k}||^2_{\beta_j} + ||V^T z_{\text{side}} - V^T \mu_{j,k}||^2_{\beta_j}, \tag{2}$$

where $z_{\text{top}}, z_{\text{side}}$ are the encoded views with augmentations, i.e., $z_t = \text{enc}(\text{aug}_{x_t}(\text{color}_x(x_t)), a), t \in \{\text{top}, \text{side}\}$ with the same color augmentation for bead $\mathbf{x}$ and different non-color augmentations per view. We use the average embedding $z = (z_{\text{top}} + z_{\text{side}})/2$ to calculate the cluster centers $\mu_{j,k}$ and corresponding cluster assignments for each bead. Given the cluster assignments we construct $C_{j,k}$, which is the set of tuples of all embedded views across beads in cluster k of feature space j. To regularize the embedded space and avoid arbitrary solutions we include the reconstruction loss into the objective as well, leading to our final objective $\mathcal{L} = \mathcal{L}_{\text{enrc}} + \mathcal{L}_{\text{rec}}$.

During the training of ENRC the objective in Equation (2) "moves" both embedded views per bead closer to their shared center in each feature space, effectively compressing the embedded space. This objective makes the cluster assignments invariant to each view and the applied augmentations while preserving important color information.

6 EXPERIMENTS

We evaluate our pipeline with quantitative and qualitative experiments. A key challenge is that we do not have access to ground truth labels and depend on unsupervised metrics for selecting the best clustering. We show how we can use those to find a suitable clustering solution. In the qualitative experiments we further investigate this solution. The detailed software and hardware settings used for our experiments can be found in Appendix C.3.

6.1 Metrics

We make use of several unsupervised metrics to find the best non-redundant clustering. First, we use the reconstruction loss $\mathcal{L}_{\text{rec}}$ as defined in Equation 1 to select the best pretrained Mixed Convolutional Autoencoder (MCAE) on a held out test set. Further, among the best performing autoencoders we select the one that has the lowest bottleneck dimensionality to avoid the curse of dimensionality for clustering. To select the best, most informative non-redundant clusterings we use the average Variation of Information (VI) [24] for measuring the redundancy of the found clusterings. The VI calculates the dissimilarity between different clusterings, where higher values are better. Additionally, we make use of the MDL costs for AutoNR and ENRC to assess the quality of the clusterings and their fit to the data. A good non-redundant clustering should balance the two metrics. The clustering should be non-redundant as measured via the VI and it should fit to the data without being overly complex as measured with the MDL costs. As we are also interested to learn a view invariant representation with ENRC we measure the average cosine similarity between the embeddings of both views for all beads, where a similarity of 0 indicates no similarity and a value of 1 indicates a perfect overlap.

6.2 Quantitative Experiments

6.2.1 Autoencoder Experiments. We conduct hyperparameter tuning of MCAE using a holdout test set of 10% of the total amount of data to determine the best values for the learning rate, weight decay, latent space dimensionality and model type (ResNet18 vs. ResNet50). We trained all models for 4000 epochs to ensure a proper reconstruction. The detailed setting and considered hyperparameters can be found in Table 4 of the Appendix. In total, we performed 144 runs with different hyperparameters over two random seeds.

Latent Space Dimensionality: The dimensionality of the embedded space d is one of the most crucial parameters to select in deep clustering. If the dimensionality is too small we would lose too much information making the subsequent clustering meaningless. If the dimensionality is chosen too high we face the curse of dimensionality. Therefore, we conduct experiments for $d \in \{32, 64, 128\}$. In Figure 8 we show a box plot of the test and train reconstruction losses for all models across the different values for d. Overall we

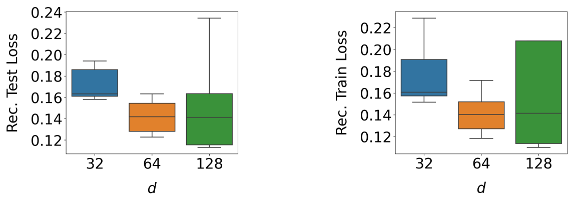


Figure 8: Box plots of reconstruction loss vs. latent space dimensionality d for all pretrained MCAE models.

see that the median test and train reconstruction losses are similar indicating no severe overfitting. We see that the models with $d = 32$ have a higher median reconstruction loss than the ones with 64 or 128 dimensions. The median loss for the models with 64 and 128 is similar, but we see a much higher variance for the runs with 128 dimensions. The higher variance might be due to the small amount of data that we have in comparison to the large ResNet50 model. To account for the model capacity we conducted additional experiments with the smaller ResNet18 model next.

Model Capacity: Our MCAE model with ResNet50 encoder and decoder has about 110,000,000 parameters, begging the question whether we need so many trainable parameters for such a small data set. We compare the ResNet50 model with a version of MCAE with only ResNet18 encoder and decoder, which has only 23,000,000 parameters. In Figure 9 we show a box plot of train and test reconstruction losses with respect to the two different architectures. We clearly see that the ResNet50 architecture achieves a better performance than its ResNet18 counter part without overfitting.

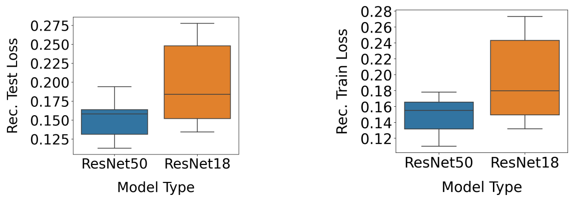


Figure 9: Box plots of reconstruction loss vs. model type for all pretrained MCAE models.

We believe that we do not overfit to the data due to several reasons. First, the beads are complex to reconstruct and have a high variety. Second, we make use of image augmentation which randomly changes the images increasing the effective training data size. Third, we use a low dimensionality for the bottleneck dimension $d \in \{32, 64, 128\}$, which acts as a regularizer against overfitting. Overall, the ResNet50 version of MCAE with a latent space of 64 performed best.

6.2.2 AutoNR Experiments. For AutoNR we consider three different metrics to select the best performing model. We use the MDL costs, the Average VI and the number of extracted clusterings J. First, we know that we should have at most four clusterings using the domain knowledge that we want to find clusterings for color, shape, decoration and size. AutoNR uses a *maximum number of subspaces* parameter that limits the search for non-redundant clusterings, which we set to five (four clusterings and one noise space clustering). The remaining parameter settings are shown in Table 5. We applied AutoNR with this setting to all pretrained ResNet50 models with a 32, 64 and 128 latent space dimensionality respectively. On average we found two clusterings with $d = 32$ and four clusterings with $d \in \{64, 128\}$. It seems that a dimensionality of only 32 is losing too much information relevant for clustering, underestimating the potential number of clusterings. We selected the best AutoNR run from the models with $d = 64$ based on the MDL costs, as these where the best for the MCAE pretraining. This run also achieved an VI of 3.22, which is close to the median VI value of 3.37, balancing the non-redundancy with the model fit.

6.2.3 ENRC Experiments. Based on the previous analysis we selected the AutoNR run with the lowest MDL costs from the models with a latent space of $d = 64$. We used the paramaters learned with AutoNR to initialize ENRC and remove the found outliers in their respective subspaces during the optimization with ENRC. Performing ENRC on top of AutoNR can be seen as a form of fine tuning the found clustering results to learn the desired invariances and improve the clusterings. We performed a hyperparameter search with respect to the learning rate and selected the best run w.r.t. the MDL costs. The detailed hyperparameter setting can be found in Table 6. In Table 1 we show how the fine tuning with ENRC changed the performance metrics against its previous AutoNR results. The first metric, that changed the most is the average *cosine similarity*. After pretraining with the MCAE the cosine similarity is low at around 0.37 and after fine tuning with ENRC we reached a perfect view invariant representation with a maximum cosine similarity of 1. The decoder cannot reconstruct the input views as good as

Metric	Before	After
Cosine Similarity	0.37	1.00
Reconstruction Loss	0.14	0.44
Average VI	3.22	2.75
Nr of clusters per subspace j	[27, 6, 6, 2]	[16, 7, 7, 2]
log MDL Costs	6.87	6.84

Table 1: Performance metrics before and after applying ENRC to the best AutoNR run

before as the invariance in the embedded space increases. This is why we see an increase of the reconstruction loss from 0.13 to 0.44. This is not a problem as we are mainly interested in the clustering performance at this stage. As a result, the Average VI decreased and the total number of clusters was reduced (41 vs. 32), e.g., due to the augmentation invariance flipped images are not assigned their own cluster. The MDL costs improved, shown here on a log scale. While the change from 6.87 to 6.84 might seem small in the original scale this corresponds to an improvement of 1e5 or 6%. Next, we investigate the found clustering of the best ENRC model in more detail using visualizations.

6.3 Qualitative Experiments

For interpreting the found clusterings we look at the reconstructed cluster centers and their corresponding nearest neighbors. The nearest neighbor analysis gives us a tool for interpreting a cluster based on its most prototypical members. In total the best ENRC run found four clusterings with $K_j \in \{16, 7, 7, 2\}$ number of clusters in each clustering. In Figure 10 we show the reconstructed cluster centers for each clustering. The top rows shows the $K_1 = 16$ cluster centroids, which correspond to different shapes and applied decorations, while the color information is averaged out. The second row

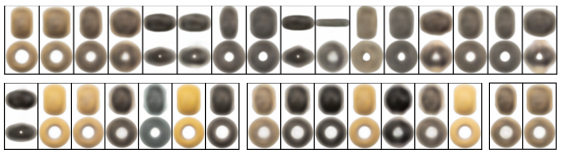


Figure 10: Reconstructed cluster centers per clustering.

shows two clusterings according to color ($K_2 = 7, K_3 = 7$) and one according to longitudinal size ($K_4 = 2$). The extracted centers can be used by archaeologists to find the most prototypical bead types, which offers a more intuitive and quantifiable way of describing a bead class. To explore the clusterings further we look at the nearest neighbors of some of the most interesting clusters. In Figure 11 we show three clusterings according to shape that were extracted. The first two rows are cylindrical and leaf shaped glass beads. Note, that we have objects of the same shape but with different alignment (horizontal and vertical flips) or partially broken pieces inside the same clusters. The clustering in the last row shows *tubular* shaped glass beads. The top view of the tubular beads shows also the difficulty of recording them always from the same angle through the

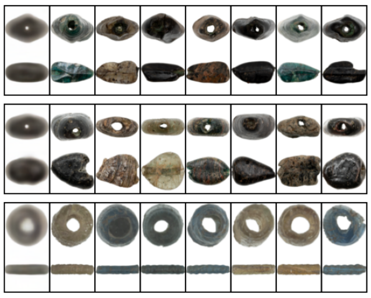


Figure 11: Nearest neighbors per cluster center of clustering $K_1 = 16$. The first image (from left to right) is the reconstructed cluster center and the other seven images are the corresponding nearest neighbors.

hole. Despite these issues ENRC could find the cluster due to the learned shearing, rotation and translation invariances.

In Figure 12 we show a sample of outliers that were found using AutoNR and ENRC. We see that these instances are either rare beads or no beads at all. While the last glass bead on the right is a

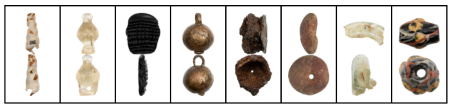


Figure 12: Sample of found outliers. From left to right: mollusc shell, rare glass bead type, large glass bead, metal pendant, piece of metal, large bead made of amber, piece of glass vessel, rare decoration type.

valid bead type it is flagged as an outlier as it is the only bead with that specific form and decoration in the data set.

7 FUTURE WORK AND CONCLUSION

As archaeological data bases will grow, larger data sets of glass beads will also become available. In future work we want to evaluate the transferability of our pipeline to larger data sets of other grave sites. The goal is to have an automatic system that gets better with more diverse data and is reproducible worldwide. Currently, some rare bead types are flagged as outliers, which is valid given their unusual decoration, shape or color, but with growing data it might be discovered that these bead types are more common and form their own cluster.

We have proposed the first application of non-redundant clustering to archaeological image data. We showed how unsupervised model selection can be done for deep embedded non-redundant clustering algorithms in the absence of ground truth labels. Further, we introduced a novel, high quality data set of diverse beads that poses new challenges for non-redundant clustering. We showed in quantitative and qualitative evaluations that we could find non-redundant clusterings that align with domain knowledge of experts, like different subtypes of the *melon-seed*-shaped glass beads (cylindrical and leaf shaped glass beads).

REFERENCES

[1] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. 2018. Clustering with Deep Learning: Taxonomy and New Methods. *CoRR* abs/1801.07648 (2018).

[2] Yannis M. Assael, Thea Sommerschield, Brendan Shillingford, Mahyar Bordbar, John Pavlopoulos, Marita Chatzipanagiotou, Ion Androutsopoulos, Jonathan Prag, and Nando de Freitas. 2022. Restoring and attributing ancient texts using deep neural networks. *Nat.* 603, 7900 (2022), 280–283.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[4] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–37.

[5] Matthew C Delvaux. 2018. Colors of the Viking Age. *Journal of Glass Studies* 60 (2018), 41–68.

[6] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*. IEEE Computer Society, 248–255.

[8] Chris HQ Ding, Tao Li, and Michael I Jordan. 2008. Convex and semi-nonnegative matrix factorizations. *IEEE transactions on pattern analysis and machine intelligence* 32, 1 (2008), 45–55.

[9] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Commun. ACM* 39, 11 (1996), 27–34.

[10] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2019. A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation. In *ICLR (Poster)*. OpenReview.net.

[11] Peter Grünwald. 2005. Minimum description length tutorial. *Advances in minimum description length: Theory and applications* 5 (2005), 1–80.

[12] Margaret Guido. 1999. The Glass Beads of Anglo-Saxon England.

[13] John A Hartigan and Pamela M Hartigan. 1985. The dip test of unimodality. *The annals of Statistics* (1985), 70–84.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. IEEE Computer Society, 770–778.

[15] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons.

[16] Collin Leiber, Lena GM Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. 2021. Dip-based deep embedded clustering with k-estimation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 903–913.

[17] Collin Leiber, Dominik Mautz, Claudia Plant, and Christian Böhm. 2022. Automatic Parameter Selection for Non-Redundant Clustering. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 226–234.

[18] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR (Poster)*. OpenReview.net.

[19] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR (Poster)*. OpenReview.net.

[20] Jerry Ma and Denis Yarats. 2021. On the Adequacy of Untuned Warmup for Adaptive Optimization. In *AAAI*. AAAI Press, 8828–8836.

[21] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. 2018. Discovering Non-Redundant K-means Clusterings in Optimal Subspaces. In *Proceedings of the 24th ACM SIGKDD, 2018*. 1973–1982.

[22] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. 2020. Non-redundant subspace clusterings with nr-kmeans and nr-dipmeans. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14, 5 (2020), 1–24.

[23] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software* 3, 29 (2018), 861.

[24] Marina Meilă. 2007. Comparing clusterings—an information based distance. *J. of multivariate analysis* 98, 5 (2007), 873–895.

[25] Lukas Miklautz, Dominik Mautz, Muzaffer Can Altinigneli, Christian Böhm, and Claudia Plant. 2020. Deep Embedded Non-Redundant Clustering. In *AAAI*. AAAI Press, 5174–5181.

[26] Lukas Miklautz, Martin Teuffenbach, Pascal Weber, Rona Perjuci, Walid Durani, Christian Böhm, and Claudia Plant. 2022. Deep Clustering With Consensus Representations. *CoRR* abs/2210.07063 (2022).

[27] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. 2018. A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture. *IEEE Access* 6 (2018), 39501–39514.

[28] Emmanuel Müller, Stephan Günnemann, Ines Färber, and Thomas Seidl. 2012. Discovering Multiple Clustering Solutions: Grouping Objects in Different Views of the Data. In *Proceedings of the 28th ICDE, Washington, DC, USA, 1-5 April, 2012*. 1207–1210. https://doi.org/10.1109/ICDE.2012.142

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[30] Adrien Pásztor. 1997. Typologische Untersuchung der früh-und mittelawarenzeitlichen Perlen aus Ungarn.(A typological analysis of beads of the Early and Middle Avar period from Hungary). *Perlen. Archäologie, Techniken, Analysen. Akten des Internationalen Perlensymposiums in Mannheim vom 11. bis 14. November 1994. Kolloquien zur Vor- und Frühgeschichte 1 (Bonn 1997).* (1997), 213–230.

[31] Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465–471.

[32] Barbara Sasse and Claudia Theune. 1996. Perlen als Leittypen der Merowingerzeit. *Germania* 074/1 (01 1996), 187ff.

[33] Zoltán Szabó, Barnabás Póczos, and András Lőrincz. 2012. Separation theorem for independent subspace analysis and its consequences. *Pattern Recognition* 45, 4 (2012), 1782–1791.

[34] Barbara Theune-Großkopf. 2015. Perlenglanz: Modeströmungen und Produktionshinweise im frühen Mittelalter. In *Glasklar - Archäologie eines kostbaren Werkstoffes (Friedberg 2015)*. 44–49.

[35] Xing Wang, Jun Wang, Carlotta Domeniconi, Guoxian Yu, Guoqiang Xiao, and Maozu Guo. 2019. Multiple independent subspace clusterings. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5353–5360.

[36] Max Welling and Kenichi Kurihara. 2006. Bayesian K-means as a "maximization-expectation" algorithm. In *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 474–478.

[37] Wei Ye, Samuel Maurus, Nina Hubig, and Claudia Plant. 2016. Generalized independent subspace clustering. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 569–578.

[38] Chongsheng Zhang, Bin Wang, Ke Chen, Ruixing Zong, Bofeng Mo, Yi Men, George Almpanidis, Shanxiong Chen, and Xiangliang Zhang. 2022. Data-Driven Oracle Bone Rejoining: A Dataset and Practical Self-Supervised Learning Scheme. In *KDD*. ACM, 4482–4492.

[39] Sheng Zhou, Hongjia Xu, Zhuonan Zheng, Jiawei Chen, Zhao Li, Jiajun Bu, Jia Wu, Xin Wang, Wenwu Zhu, and Martin Ester. 2022. A Comprehensive Survey on Deep Clustering: Taxonomy, Challenges, and Future Directions. *CoRR* abs/2206.07579 (2022).

[40] Qiang Zhu, Xiaoyue Wang, Eamonn J. Keogh, and Sang-Hee Lee. 2011. An efficient and effective similarity measure to enable data mining of petroglyphs. *Data Min. Knowl. Discov.* 23, 1 (2011), 91–127.

A SYMBOLS

All symbols used in the paper are briefly explained in Table 2.

Table 2: Overview of the used symbols.

Symbol	Description
$N \in \mathbb{N}$	Number of samples
$D \in \mathbb{N}$	Dimensionality of the images
$d \in \mathbb{N}$	Dimensionality of the embedded space
$J \in \mathbb{N}$	Number of subspaces
$V \in \mathbb{R}^{d\times d}$	Orthogonal (rotation) matrix
$m_j \in \mathbb{N}$	Dimensionality of subspace j
$P_j \in \mathbb{N}^{d\times m_j}$	Projection matrix of subspace j
$\beta_j \in \mathbb{R}^d$	ENRC feature importance vector of subspace j
$X \subseteq \mathbb{R}^D$	Image data set
$\mathbf{x} \in X$	Single sample of X
$A \subseteq \mathbb{R}^3$	Set of attributes
$a \in A$	Single sample of A
$Z \subseteq \mathbb{R}^d$	Embedded data set
$z \in Z$	Embedded version of ($\mathbf{x}$, a)
$\hat{\mathbf{x}} \in \mathbb{R}^D$	Image reconstruction
$\hat{a} \in \mathbb{R}^3$	Attribute reconstruction

B DATA

In this section we show some additional plots for a better overview of the data. Figure 13 shows a diverse sample of *barrel*-shaped beads. In Figure 14 we show some beads that are partially or severely broken. Some atypical beads are shown in Figure 15. In Figure 16 we show the random augmentations that we applied to get more robust results.

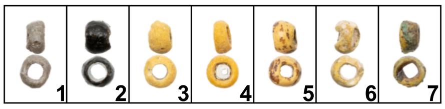


Figure 13: From left to right variations of *barrel* glass beads according to color (1) - (3); according to restoration artifacts like remains of varnish partially closing the hole (4) or remains of glue that causes brown speckles on the beads' surface (5); according to age like partially missing parts (6) or changes to the surface color of a former yellow bead due to corrosion (7).

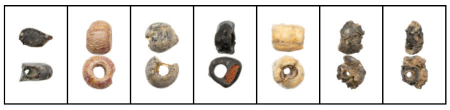


Figure 14: Sample of partially broken beads.

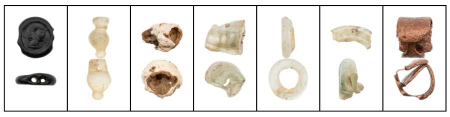


Figure 15: Outlier sample: The shown objects are either atypical or no glass beads at all. From left to right: glass bead with coin imprint, rare bead type, mollusc shell, piece of glass, lip of a glass vessel, piece of glass vessel, rolled metal sheet.

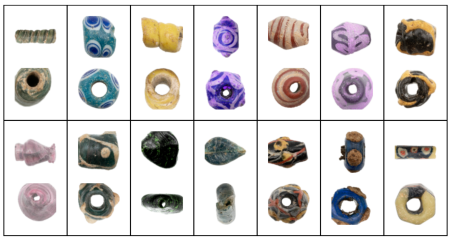


Figure 16: Random augmentations applied to images from Figure 1. Color augmentations are shared across views of the same bead and increase color diversity.

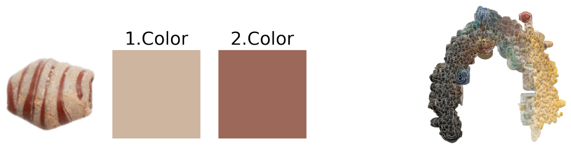


Figure 17: (Left) The two main colors extracted via k-means for a glass bead with *wound spiral* decoration. (Right) Images embedded in two dimensions according to the main colors show large areas of yellow and black beads.

C EXPERIMENT DETAILS

C.1 Colour Plot

To create the distribution in Figure 6, we extracted the two main colors of each bead by concatenating both views and running k-Means with $k = 2$ using the pixels as input. The extracted features resulted in a six dimensional RGB vector per bead. An example of two extracted colors are shown in the left part of Figure 17. The most common extracted color per image was then binned using k-Medoids [15] to get a robust clustering. Each bar in the plot represents one medoid extracted with k-Medoids and the count represents the size of the cluster. The right side of Figure 17 shows a UMAP [23] embedding of the glass beads using the two extracted color features reducing the dimensionality from six dimensions (two extracted RGB colors) to two dimensions.

C.2 Parameter settings

We provide in Tables 3, 4, 5 and 6 the considered hyperparameter settings for augmentation, pretraining, AutoNR clustering and ENRC clustering respectively.

Augmentation	Parameter
RandomAffine	degrees=(-16, +16), translate=(0.1, 0.1) shear=(-8, 8), p=0.5
RandomResizedCrop	size=64, scale=(0.9, 1.0)
RandomResizedFlip	p=0.5
RandomHorizontalFlip	p=0.5
RandomInvert	p=0.25
ColorJitter	brightness=0.3, contrast=0.5 saturation=0.5, hue=[-0.5,0.5], p=0.5

Table 3: Augmentation parameters for PyTorch augmentation transforms.

Config	Parameters
optimizer	AdamW
scheduler	Cosine annealing
linear warmup ratio	0.1
base learning rate	1e-3, 5e-4, 1e-4, 8e-5
weight decay	0, 1e-2, 1e-3
latent space	32, 64, 128
ResNet18 feed forward layers	[512, 1024, latent space]
ResNet50 feed forward layers	[2048, 4096, latent space]
activation function	ReLU
loss function	$\mathcal{L}_{\text{rec}}$
epochs	4000
batch size	256

Table 4: Pretraining parameters

Config	Parameter
Number of NrKmeans repetitions	10
Maximum number of clusters per subspace	30
Maximum number of subspaces	5

Table 5: AutoNR setting.

C.3 Software and Hardware Settings

We implemented our pipeline in Python using PyTorch [29]. To conduct our experiments, we used one node of the EuroHPC supercomputer MeluXina (https://docs.lxp.lu). The utilized GPU node consists of 2 AMD Rome CPUs (32 cores @ 2.35 GHz), 4 NVIDIA A100-40 GPUs, 512 GB of RAM, and a local SSD of 1.92 TB.

D ESTIMATING THE NUMBER OF CLUSTERINGS CONTINUED

To deepen our understanding of how our initial clusterings are obtained we explain here the clustering procedure of AutoNR in more detail. Initially, AutoNR creates a single subspace containing all the features of the data set and a single cluster containing all the data points. Accordingly, we are dealing with a noise space. Next, an attempt is made to split off a cluster space, which is an additional subspace with more than one cluster. This operation is also called *noise space split*. Here NrKmeans is executed with $k = [k_1, 1]$, where k_1 is successively increased. The result producing the lowest MDL costs is used for the following steps. If cluster spaces are present, AutoNR first tries to split those before executing another *noise space split*. For this *cluster space split*, the existing cluster space is split into two cluster spaces and NrKmeans is executed with different parameterizations, where the parameters are bounded by $\max(k_{\text{new}_1}, k_{\text{new}_2}) \leq k_{\text{old}} \leq k_{\text{new}_1} \cdot k_{\text{new}_2}$. If no split was able to lower the MDL costs, an *cluster space merge* will attempt to merge existing cluster spaces. This operation can be understood as a reversed *cluster space split*, starting with the highest possible number of clusters and combining the two closest clusters one after the other. The operations *noise space split*, *cluster space split* and *cluster space merge* are repeated as long as NrKmeans results with lower MDL costs are identified. AutoNR is then performed on a resampled version of the data to account for the color and size bias as we explain next.

Config	Parameter
optimizer	Adam
scheduler	Cosine annealing
linear warmup ratio	0.2
base learning rate	5e-4
weight decay	0
latent space	64
loss function	$\mathcal{L}$
epochs	30
batch size	256

Table 6: Best ENRC setting.

D.1 Handling the color and size bias

To account for the color and size bias we resample the data using the extracted color distribution and tabular features A, before applying AutoNR. We obtain the resampling weights by clustering the color and tabular features A using k-means and inversely weighing each sample with its cluster count. We then randomly sample 10,000 beads without replacement to have more rare beads in the data. To increase the diversity of this resampled data set w.r.t. to the colors, we apply the random color augmentations for each bead with a probability of 0.5, giving us a data set where color and shape information is more independent from each other. We use this data set for AutoNR to learn the initial parameters for ENRC. ENRC is then using the resampling weights for adjusting the sampling probabilities during the mini-batch training. After ENRC has been trained we predict the labels for the clean, original data. Using this we can circumvent some of the color bias in the data. Note, that the performance metrics in the paper are always reported on the original data without resampling.