Clustering with Deep Learning:
Taxonomy and New Methods

Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel & Daniel Cremers
Computer Vision Group
Technical University of Munich
{firstname.lastname, cremers}@tum.de

September 17, 2018

ABSTRACT

Clustering methods based on deep neural networks have proven promising for clustering real-world data because of their high representational power. In this paper we propose a systematic taxonomy of clustering methods that utilize deep neural networks. We base our taxonomy on a comprehensive review of recent work and validate the taxonomy in a case study. In this case study, we show that the taxonomy enables researchers and practitioners to systematically create new clustering methods by selectively recombining and replacing distinct aspects of previous methods with the goal of overcoming their individual limitations. The experimental evaluation confirms this and shows that the method created for the case study achieves state-of-the-art clustering quality and surpasses it in some cases.

1 INTRODUCTION

The main objective of clustering is to separate data into groups of similar data points. Having a good separation of data points into clusters is fundamental for many applications in data analysis and data visualization.

The performance of current clustering methods is however highly dependent on the input data. Different datasets usually require different similarity measures and separation techniques. As a result, dimensionality reduction and representation learning have been extensively used alongside clustering in order to map the input data into a feature space where separation is easier. By utilizing deep neural networks (DNNs), it is possible to learn non-linear mappings that allow transforming data into more clustering-friendly representations without manual feature extraction/selection.

In the past, feature extraction and clustering were applied sequentially Ding and He (2004); Trigeorgis et al. (2014). However, recent work shows that jointly optimizing for both can yield better results Song et al. (2013); Xie et al. (2016); Yang et al. (2016a;b); Li et al. (2017).

The main contribution of this paper is the formulation of a taxonomy for clustering methods that rely on a deep neural network for representation learning. The proposed taxonomy enables researchers to create new methods in a structured and analytical way by selectively recombining or replacing distinct aspects of existing methods to improve their performance or mitigate limitations. The taxonomy is in particular also valuable for practitioners who want to create a method from existing building blocks that suits their task at hand. To illustrate the value of the proposed taxonomy, we conducted a case study in which we fuse a new method based on insights from the taxonomy.

In this case study, we use a fully convolutional autoencoder to learn clustering-friendly representations of the data by optimizing it with a two-phased training procedure. In the first phase, the
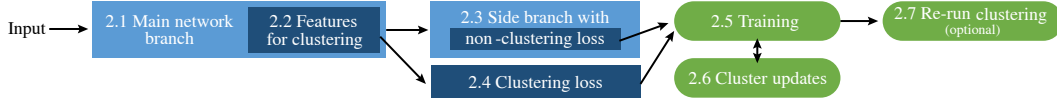
Figure 1: The general pipeline of most deep-learning-based clustering methods accoding to the proposed taxonomy, with building blocks as described in Sections 2.1–2.7.

autoencoder is trained with the standard mean squared error reconstruction loss. In the second phase, the autoencoder is then fine-tuned with a combined loss function consisting of the autoencoder reconstruction loss and a clustering-specific loss.

The rest of this paper is organized as follows: At first, we introduce the taxonomy and its building blocks in Section 2. In Section 3, we then present a comprehensive review of clustering methods and analyze them with respect to the proposed taxonomy. Subsequently, in Section 4, we propose a new method based on insights gained in a systematic way from the taxonomy. The evaluation of the proposed method is presented in Section 5, followed by the conclusions in Section 6.

2 TAXONOMY

The most successful methods for clustering with deep neural networks all work following the same principle: representation learning using DNNs and using these representations as input for a specific clustering method. As depicted in Figure 1, every method consists of the following parts, for each of which there are several options to choose from:

- Neural network training procedure, consisting of:
  - Main neural network branch and its usage
    * Architecture of main neural network branch, described in Section 2.1
    * Set of deep features used for clustering, described in Section 2.2
  - Neural network losses:
    * Non-clustering loss, described in Section 2.3
    * Clustering loss, described in Section 2.4
    * Method to combine the two losses, described in Section 2.5
  - Cluster updates, described in Section 2.6
- (Optional) Re-run clustering after network training, described in Section 2.7

2.1 ARCHITECTURE OF MAIN NEURAL NETWORK BRANCH

In most DNNs-based clustering methods, the "main branch" of the neural network (apart from side branches towards non-clustering losses, see Section 2.3) is used to transform the inputs into a latent representation that is used for clustering. The following neural network architectures have previously been used for this purpose:

- **Multilayer Perceptron (MLP)**: Feedforward network, consisting of several layers of neurons, such that the output of every hidden layer is the input to next one.
- **Convolutional Neural Network (CNN)**: Inspired by biology, more precisely by the organization of the animal visual cortex. Useful for applications to regular-grid data such as images, if locality and shift-equivariance/invariance of feature extraction is desired.
- **Deep Belief Network (DBN)**: Generative graphical model, consisting of several layers of latent variables. It is composed of several shallow networks such as restricted Boltzmann machines, such that the hidden layer of each sub-network serves as the visible layer of the next sub-network.
- **Generative Adversarial Network (GAN)**: A system of two competing neural network models G and D that engage in a zero-sum game. The generator G learns a distribution of interest to produce samples. The discriminator D learns to distinguish between real samples and generated ones (Goodfellow et al., 2014).

- **Variational Autoencoder (VAE)**: A Bayesian network with an autoencoder architecture that learns the data distribution (generative model).

2.2 SET OF DEEP FEATURES USED FOR CLUSTERING

After transforming the input to a more clustering-friendly representation, the features that are then used for clustering can be taken from one or more layers of the deep neural network:

- **One layer**: Refers to the case where only one layer of the network is used which is beneficial because of its low dimensionality. In most cases the output of the last layer is used.
- **Several layers**: Refers to the case where the representation is a combination of the outputs of several layers. Thus, the representation is richer and allows the embedded space to represent more complex semantic representations, which might enhance the separation process and help in the similarity computation (Saito and Tan, 2017).

2.3 NON-CLUSTERING LOSS

The non-clustering loss is independent of the clustering algorithm and usually enforces a desired constraint on the learned model. Possible options are as follows:

- **No non-clustering loss**: No additional non-clustering loss function is used and the network model is only constrained by the clustering loss. For most clustering losses, the absence of a non-clustering loss can have a danger of worse representations/results, or theoretically even collapsing clusters (Yang et al., 2016a), but the latter rarely occurs in practice.
- **Autoencoder reconstruction loss**: The autoencoder consists of two parts: an encoder and a decoder. The encoder maps its input x to a representation z in a latent space Z. During training, the decoder tries to reconstruct x from z, making sure that useful information has not been lost by the encoding phase. In the context of clustering methods, once the training is done the decoder part is no longer used, and the encoder is left for mapping its input to the latent space Z. By applying this procedure, autoencoders can successfully learn useful representations in the cases where the output's dimensionality is different from the input's or when random noise is injected to the input (Vincent et al., 2010). Additionally, they can also be used for dimensionality reduction goals (Hinton and Salakhutdinov, 2006). Generally the reconstruction loss is a distance measure $d_{\mathrm{AE}}(x_i, f(x_i))$ between the input x_i to the autoencoder and the corresponding reconstruction $f(x_i)$. One particular formulation of it is using the mean squared error of the two variables:

$$L = d_{\mathrm{AE}}(x_i, f(x_i)) = \sum_i \|x_i - f(x_i)\|^2, \tag{1}$$

  where x_i is the input and $f(x_i)$ is the autoencoder reconstruction. This loss function guarantees that the learned representation preserves important information from the initial one, which is why reconstruction is possible.
- **Self-Augmentation Loss** Hu et al. (2017) proposes a loss term that pushes together the representation of the original sample and their augmentations:

$$L = -\frac{1}{N} \sum_N s(f(x), f(T(x))), \tag{2}$$

  where x is the original sample, T is the augmentation function, $f(x)$ is the representation generated by the model, and s is some measure of similarity (for example cross-entropy if f has a softmax nonlinearity).
- **Other tasks**: Additional information about training samples that is available in the form of targets, even if not perfectly suitable to dictate clustering, can be used in a (multi-task) non-clustering loss to encourage meaningful feature extraction.

2.4 CLUSTERING LOSS

The second type of functions is specific to the clustering method and the clustering-friendliness of the learned representations, therefore such functions are called clustering loss functions. The following are options for clustering loss functions:

- **No clustering loss**: Even if a neural network has only non-clustering losses (Section 2.3), the features it extracts can be used for clustering after training (Sections 2.6–2.7). The neural network serves in this case for changing the representation of the input, for instance changing its dimensionality. Such a transformation could be beneficial for the clustering sometimes, but using a clustering loss usually yields better results (Xie et al., 2016; Yang et al., 2016a).

- **k-Means loss**: Assures that the new representation is k-means-friendly (Yang et al., 2016a), i.e. data points are evenly distributed around the cluster centers. In order to obtain such a distribution a neural network is trained with the following loss function:

$$L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} s_{ik} \|z_i - \mu_k\|^2, \tag{3}$$

where z_i is an embedded data point, μ_k is a cluster center and s_{ik} is a boolean variable for assigning z_i with μ_k. Minimizing this loss with respect to the network parameters assures that the distance between each data point and its assigned cluster center is small. Having that, applying k-means would result in better clustering quality.

- **Cluster assignment hardening**:
Requires using soft assignments of data points to clusters. For instance, Student's t-distribution can be used as the kernel to measure the similarity (van der Maaten and Hinton, 2008) between points and centroids. This distribution Q is formulated as follows:

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\nu)^{-\frac{\nu+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\nu)^{-\frac{\nu+1}{2}}}, \tag{4}$$

where z_i is an embedded data point, μ_j is the jth cluster centroid, and ν is a constant, e.g. $\nu = 1$. These normalized similarities between points and centroids can be considered as soft cluster assignments. The *cluster assignment hardening* loss then enforces making these soft assignment probabilities stricter. It does so by letting cluster assignment probability distribution Q approach an auxiliary (target) distribution P which guarantees this constraint. Xie et al. (2016) propose the following auxiliary distribution:

$$p_{ij} = \frac{q_{ij}^2/\Sigma_i q_{ij}}{\Sigma_{j'}(q_{ij'}^2/\Sigma_i q_{ij'})}. \tag{5}$$

By squaring the original distribution and then normalizing it, the auxiliary distribution P forces assignments to have stricter probabilities (closer to 0 and 1). It aims to improve cluster purity, put emphasis on data points assigned with high confidence and to prevent large clusters from distorting the hidden feature space (Xie et al., 2016). One way to formulate the divergence between the two probability distributions is using the Kullback–Leibler divergence (Kullback and Leibler, 1951). It is formulated as follows:

$$L = \mathrm{KL}(P\|Q) = \sum_{i} \sum_{j} p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{6}$$

which is minimized for the aforementioned Q and P via neural network training.

- **Balanced assignments loss**: This loss has been used alongside other losses such as the previous one (Dizaji et al., 2017). Its goal is to enforce having balanced cluster assignments. It is formulated as follows:

$$L_{ba} = \mathrm{KL}(G\|U) \tag{7}$$

where U is the uniform distribution and G is the probability distribution of assigning a point to each cluster:

$$g_k = P(y = k) = \frac{1}{N} \sum_{i} q_{ik} \tag{8}$$

By minimizing equation 7, the probability of assigning each data point to a certain cluster is uniform across all possible clusters (Dizaji et al., 2017). It is important to note that this property (uniform assignment) is not always desired. Thus, in case any prior is known it is still possible to replace the uniform distribution by the known prior one.

- **Locality-preserving loss**: This loss aims to preserve the locality of the clusters by pushing nearby data points together (Huang et al., 2014). Mathematically, it is formulated as follows:

$$L_{lp} = \sum_i \sum_{j \in N_k(i)} s(x_i, x_j) \|z_i - z_j\|^2 \tag{9}$$

where $N_k(i)$ is the set of k nearest neighbors of the data point x_i, and $s(x_i, x_j)$ is a similarity measure between the points x_i and x_j.

- **Group sparsity loss**: It is inspired by spectral clustering where block diagonal similarity matrix is exploited for representation learning (Ng et al., 2002). Group sparsity is itself an effective feature selection method. In Huang et al. (2014), the hidden units were divided into G groups, where G is the assumed number of clusters. When given a data point x_i the obtained representation has the form $\{\phi^g(x_i)\}_{g=1}^G$. Thus the loss can be defined as follows:

$$L_{gs} = \sum_{i=1}^N \sum_{g=1}^G \lambda_g \|\phi^g(x_i)\|, \tag{10}$$

where $\{\lambda_g\}_{g=1}^G$ are the weights to sparsity groups, defined as

$$\lambda_g = \lambda \sqrt{n_g}, \tag{11}$$

where n_g is the group size and λ is a constant.

- **Cluster classification loss**: Cluster assignments obtained during cluster updates (Section 2.6) can be used as "mock" class labels for a classification loss in an additional network branch, in order to encourage meaningful feature extraction in all network layers (Hsu and Lin, 2017).

- **Agglomerative clustering loss**: Agglomerative clustering merges two clusters with maximum affinity (or similarity) in each step until some stopping criterion is fulfilled. A neural network loss inspired by agglomerative clustering (Yang et al., 2016b) is computed in several steps. First, the cluster update step (Section 2.6) merges several pairs of clusters by selecting the pairs with the best affinity (some predefined measure of similarity between clusters). Then network training retrospectively even further optimizes the affinity of the already merged clusters (it can do so because the affinity is measured in the latent space to which the network maps). After the next cluster update step, the network training switches to retrospectively optimizing the affinity of the newest set of newly merged cluster pairs. In this way, cluster merging and retrospective latent space adjustments go hand in hand. Optimizing the network parameters with this loss function would result in a clustering space more suitable for (agglomerative) clustering.

2.5 Method to Combine the Losses

In the case where a clustering and a non-clustering loss function are used, they are combined as follows:

$$L(\theta) = \alpha L_c(\theta) + (1 - \alpha)L_n(\theta), \tag{12}$$

where $L_c(\theta)$ is the clustering loss, $L_n(\theta)$ is the non-clustering loss, and $\alpha \in [0; 1]$ is a constant specifying the weighting between both functions. It is an additional hyperparameter for the network training. It can also be changed during training following some schedule. The following are methods to assign and schedule the values of α:

- **Pre-training, fine-tuning**: First, α is set to 0, i.e. the network is trained using the non-clustering loss only. Subsequently, α is set to 1, i.e. the non-clustering network branches (e.g. autoencoder's decoder) are removed and the clustering loss is used to train (fine-tune) the obtained network. The constraint forced by the reconstruction loss could be lost after training the network long enough for clustering only. In some cases, losing such constraints may lead to worse results (see Table 1).

- **Joint training**: $0 < \alpha < 1$, for example $\alpha = 0.5$, i.e. the network training is affected by both loss functions.

- **Variable schedule**: α is varied during the training dependent on a chosen schedule. For instance, start with a low value for α and gradually increase it in every phase of the training.

In phases with $\alpha = 1$, no non-clustering loss is imposed, with potential disadvantages (see *No non-clustering loss* in Section 2.3). Similarly, in phases with $\alpha = 0$, no clustering loss is imposed, with potential disadvantages (see *No clustering loss* in Section 2.4).

2.6 CLUSTER UPDATES

Clustering methods can be broadly categorized into hierarchical and partitional (centroid-based) approaches (Jain et al., 1999). Hierarchical clustering combines methods which aim to build a hierarchy of clusters and data points. On the other hand, partitional (centroid-based) clustering groups methods which create cluster centers and use metric relations to assign each of the data points into the cluster with the most similar center.

In the context of deep learning for clustering, the two most dominant methods of each of these categories have been used. **Agglomerative clustering**, which is a hierarchical clustering method, has been used with deep learning (Yang et al., 2016b). The algorithm has been briefly discussed in Section 2.4. In addition, **k-means**, which falls into the category of centroid-based clustering, was extensively used (Xie et al., 2016; Yang et al., 2016a; Li et al., 2017; Hsu and Lin, 2017).

During the network training, cluster assignments and centers (if a centroid-based method is used) are updated. Updating cluster assignments can have one of the two following forms:

- **Jointly updated with the network model**: Cluster assignments are formulated as probabilities, therefore have continuous values between 0 and 1. In this case, they can be included as parameters of the network and optimized via back-propagation.
- **Alternatingly updated with the network model**: Clustering assignments are strict and updated in a different step than the one where the network model is updated. In this case, several scenarios are possible, dependent on two main factors:
  - **Number of iterations**: Number of iterations of the chosen clustering algorithm, that are executed at every cluster update step. For instance, in Xie et al. (2016), at each cluster update step, the algorithm runs until a fixed percentage of points change assignments between two consecutive iterations.
  - **Frequency of updates**: How often are cluster updates started. For instance in Yang et al. (2016b), for every P network model update steps, one cluster updates step happens.

2.7 AFTER NETWORK TRAINING

After the training is finished, even if a clustering result was produced in the process, it can make sense to re-run a clustering algorithm from scratch using the learned features, for one of several reasons:

- **Clustering a similar dataset**: The general and the most trivial case is to reuse the learned features representation mapping on another dataset which is similar to the one that has been used but has different data.
- **Obtaining better results**: Under certain circumstances, it is possible that the results of clustering after the training are better than the ones obtained during the learning procedure. For instance, in Yang et al. (2016b), such a behavior is reported. One possible reason for this to happen is that the cluster update step during the training doesn't go all the way till the end (see *Number of iterations* in Section 2.6).

3 RELATED METHODS

In this section we give a systematic overview of related work and analyze it within the framework of the proposed taxonomy. In Table 1 the methods are decomposed with regard to the following building blocks: Section 2.1 (network architecture), 2.2 (used features), 2.3 (non-clustering loss),

2.4 (clustering loss), 2.5 (loss combination), clustering algorithm, as well as their performance. Additional explanations to some of the methods are given in the following.

METHOD	ARCH	FEATURES FOR CLUSTERING	NON-CLUSTERING LOSS	CLUSTERING LOSS	COMBINING THE LOSS TERMS	CLUSTERING ALGORITHM	NMI MNIST	ACC MNIST	NMI COIL20	ACC COIL20
JULE Yang et al. (2016b)	CNN	CNN output	-	Agglomerative loss	-	Agglomerative clustering	0.915	-	**1**	-
CCNN Hsu and Lin (2017)	CNN	Internal CNN layer	-	Cluster classification loss	-	k-Means	0.876	-	-	-
SCCNN Lukic et al. (2016)	CNN	CNN output	Classification loss	-	-	"Standard Clustering Method"	-	-	-	-
DEC Xie et al. (2016)	MLP	Encoder output	Autoencoder reconstruction loss	Cluster assignment hardening	Pretraining and fine-tuning	Network estimates centroids (as trainable parameters); soft assignments done based on distance to centroids	0.800[1]	0.843	-	-
DBC Li et al. (2017)	CNN	Encoder output	Autoencoder reconstruction loss	Cluster assignment hardening	Pretraining and fine-tuning	k-Means	0.917	0.964	0.895	0.793
DEPICT Dizaji et al. (2017)	CNN	Encoder output	Autoencoder reconstruction loss	Balanced-assignment	Pretraining followed by joint training	Network estimates centroids (as trainable parameters) and predicts assignments	0.916	0.965	-	-
DCN Yang et al. (2016a)	MLP	Encoder output	Autoencoder reconstruction loss	k-Means loss	Alternating between joint training and cluster updates	Network estimates centroids; soft assignments based on distance	0.810	0.830	-	-
DEN Huang et al. (2014)	MLP	Encoder output	Autoencoder reconstruction loss	Locality-preserving + Group sparsity	Joint training	k-Means	-	-	0.870	0.724
Neural Clustering Saito and Tan (2017)	MLP	Concat. of encoder layers output	Autoencoder reconstruction loss	-	-	k-Nearest Neighbors	-	0.966	-	-
NMMC Chen (2015)	DBN	Deepest DBN layer	Usual DBN loss	Maximum margin for mixture components	Pre-train DBN, estimate number of clusters, fine-tune only the deepest layers	-	-	-	-	-
UMMC Chen et al. (2017)	DBN	Deepest DBN layer	Usual DBN loss	Locality-preserving loss + Cluster assignment hardening	Pretraining followed by joint training	k-Means	0.864	-	0.891	-
UGATC (Premachandran & Yuille 2016)	GAN	Penultimate discriminator layer	Usual GAN loss	-	-	k-Means++	-	-	-	-
VaDE Zheng et al. (2016)	VAE	Encoder output	VAE loss with mixture model for clusters		-	Network estimates centroids	-	0.944	-	-
TAGnet Wang et al. (2016)		Sparse-coding iterations time-unfolded (reformulated) into a deep network	Sparse coding (implicit in architecture)	- Similarity graph regularization (implicit) - Cluster classification or maximum margin	Training on explicit loss; implicit losses are part of the architecture	Network predicts soft-assignments to clusters	0.651	0.692	0.927	**0.899**
IMSAT Hu et al. (2017)	MLP	MLP output	Information maximization + Self-Augmentation loss	-	Joint training	Network predicts soft-assignments to clusters	-	**0.984**	-	-
Proposed Method	CNN	Encoder output	Autoencoder reconstruction loss	Cluster assignment hardening	Pretraining followed by joint training	Network estimates centroids; soft assignments based on distance	**0.923**	0.961	0.853	0.781

Table 1: Decomposition and comparison of existing methods based on the building blocks of the taxonomy. Quality values are taken from the respective original publications, unless stated otherwise. The method created as part of the case study outperforms other methods in NMI on MNIST and achieves balanced quality results across all metrics.

Convolutional neural networks as feature extractors Some methods utilize convolutional neural networks as feature extractors to improve clustering of image data. JULE (Yang et al., 2016b) follows a hierarchical clustering approach and uses the agglomerative clustering loss as the sole loss function.

In CCNN (Hsu and Lin, 2017), initialization of the k-means centroids is performed based on the softmax output layer of a convolutional neural network and the output of internal network layers are used as features. IMSAT (Hu et al., 2017) uses the information maximization loss and a self-augmentation loss. SCCNN (Lukic et al., 2016) uses a convolutional network that was pretrained on a related, domain-specific task (speaker identification) as a feature extractor.

---

[1]Values are taken from (Yang et al., 2016a) since the DEC paper did not report NMI results.

Autoencoders as feature extractors In contrast to the aforementioned methods, some of the most promising methods utilize autoencoders as feature extractors. Training is performed in two phases. In the first phase, the autoencoder is pretrained using a standard reconstruction loss. The second phase differs between the methods: In the second phase of DEC (Xie et al., 2016), the network is fine-tuned using the cluster assignment hardening loss. DEC is often used as a baseline for new publications. DBC (Li et al., 2017) and DEPICT Dizaji et al. (2017) are similar to DEC except for one aspect each: DBC utilizes a convolutional autoencoder in order to improve clustering of image datasets and DEPICT adds a balanced assignments loss to the clustering loss to alleviate the danger of obtaining degenerate solutions.

In contrast to DEC and similar methods, DCN (Yang et al., 2016a) jointly trains the network with both the autoencoder reconstruction loss and the k-means clustering loss in the second phase. In, DEN Huang et al. (2014), the second phase is comprised of joint training with a combination of the reconstruction loss, a locality-preserving loss, and a group sparsity loss. In Neural Clustering Saito and Tan (2017), the second training phase does not exist and no additional clustering loss is used. The good results can possibly be attributed to the fact that multiple network layers are concatenated and used as the clustering features.

Other deep architectures as feature extractors NMMC (Chen, 2015) and UMMC (Chen et al., 2017) each use a deep belief network for feature extraction. UGTAC (Premachandran and Yuille, 2016) uses the penultimate layer of the discriminator in the DCGAN (Radford et al., 2015) architecture as features for k-means++ clustering. VaDE (Zheng et al., 2016) uses a variational autoencoder in combination with a mixture of Gaussians.

Other methods Instead of directly using a neural network to extract features, infinite ensemble clustering (Liu et al., 2016) uses neural networks to generate infinite ensemble partitions and to fuse them into a consensus partition to obtain the final clustering.
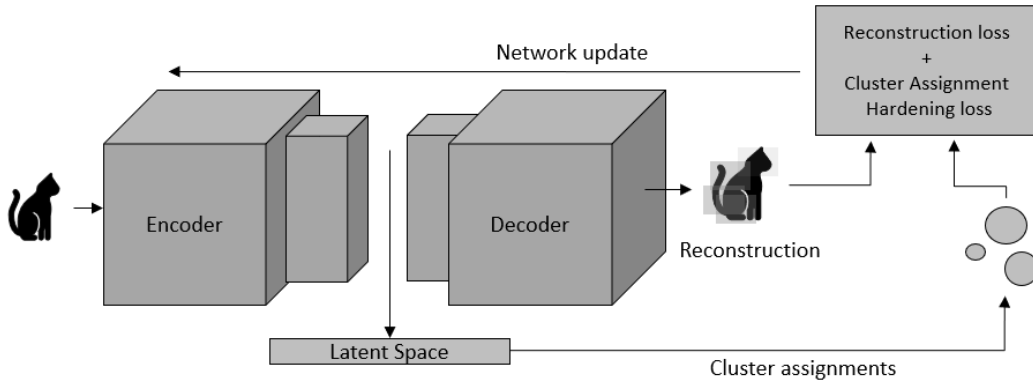

Figure 2: Our proposed method is based on a fully convolutional autoencoder trained with reconstruction and cluster hardening loss as described in Section 2.3 and 2.4. It results in clustering-friendly feature space with no risk of collapsing.

4 CASE STUDY: NEW METHOD

We argue that the proposed taxonomy (Section 2) enables a more systematic and analytical approach to creating new methods. By decomposing existing methods according to building blocks of the taxonomy, as shown in Table 1, researchers and practitioners can analyze commonalities amongst good methods and attribute limitations to specific building blocks. To show that the taxonomy makes this approach easier, we conduct a case study, in which we create a method that overcomes the limitations of the previous approaches and achieves better results for the task at hand.

A high-level overview of our method is provided in Figure 2. We will make the code publicly available.
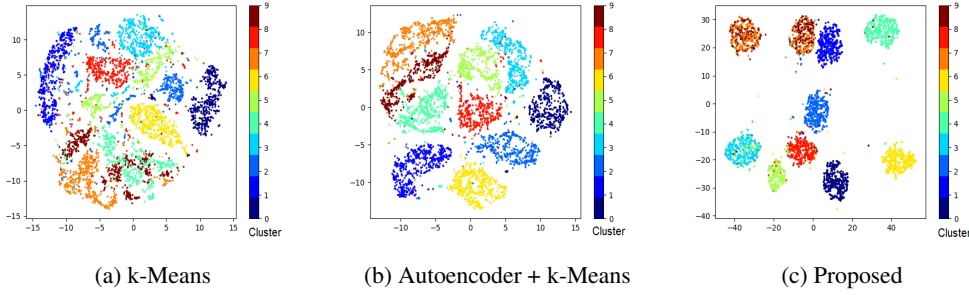
| (a) k-Means | (b) Autoencoder + k-Means | (c) Proposed |

Figure 3: t-SNE visualizations for clustering on MNIST dataset in (a) Original pixel space, (b) Autoencoder hidden layer space and (c) Autoencoder hidden layer space with the proposed method.
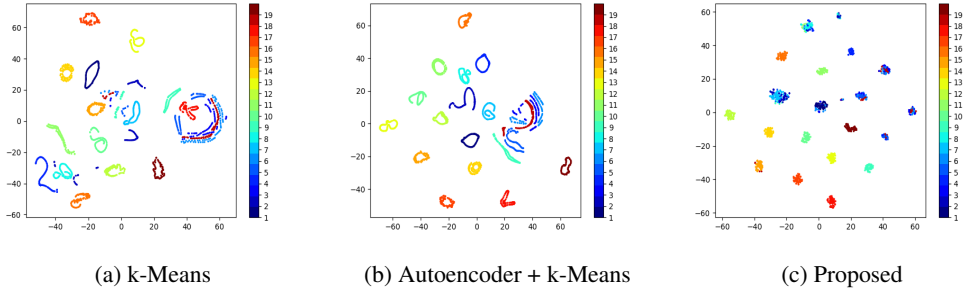

| (a) k-Means | (b) Autoencoder + k-Means | (c) Proposed |

Figure 4: t-SNE visualizations for clustering on COIL20 dataset in (a) Original pixel space, (b) Autoencoder hidden layer space and (c) Autoencoder hidden layer space with the proposed method.

Since the dataset that is to be clustered consists of images, we chose a convolutional architecture for the architecture building block (Section 2.1) and an autoencoder reconstruction loss as the non-clustering loss building block (Section 2.3).

After pretraining the network with this configuration, we chose to keep the non-clustering autoencoder reconstruction loss and add the cluster hardening loss as the clustering loss building block (Section 2.4). We decided to keep the non-clustering loss in this phase, because other methods (e.g. DEC and DBC) which omit it, have a theoretical risk of collapsing clusters and have also empirically shown to yield worse clustering quality. This is because valuable feature detectors that were previously learned during pretraining can be destroyed. Keeping the non-clustering loss helps to keep those detectors stable. For a similar reason, we decided to avoid a training procedure with alternating losses as for instance used by DBC, DEN, and UMMC. The necessity for this alternation originates in the hard assignments as enforced by the k-means loss, which is why we chose soft cluster assignments and cluster hardening loss instead. This loss is achieved via introducing a clustering layer, which aims at estimating the cluster assignments based on equation (4).

Once both training phases (pretraining on non-clustering loss and fine-tuning on both losses) are completed, the network's encoder has learned to map its input to a clustering-friendly space. Additionally, the resulting network is capable of estimating the cluster assignments. However, based on the previous success of re-running clustering from scratch after training (see Section 2.7), we run the k-means algorithm on the network's output representation.

5 EXPERIMENTAL RESULTS

In this section we evaluate our proposed method and compare it to the methods previously introduced in Section 3.

Validation Metrics As evaluation metric we use clustering accuracy (ACC) and normalized mutual information (NMI) (Strehl and Ghosh, 2002; Vinh et al., 2010; Cai et al., 2011). Both metrics have

a range of $[0, 1]$ with 1 being the perfect clustering. Experiments are performed on datasets for which meaningful cluster assignments are known (hence ACC and NMI can be computed) but these assignments are not provided to the methods.

Experimental Setup A structured hyper-parameter search led us to training our model with a learning rate of 0.01 and a momentum of 0.9. We used batch normalization (Ioffe and Szegedy, 2015) and L2 regularization.

Datasets The experiments were performed on two publicly available datasets:

- **MNIST**: Consists of 70000 images of hand-written digits of 28×28 pixel size. The digits are centered and size is normalized (LeCun, 1998).

- **COIL20**: Contains 1440, 32×32 gray scale images of 20 objects. For each object, 72 images were taken with a 5 degrees distance (Nene et al., 1996).

Performance In Table 1, the clustering performance is denoted with respect to the two metrics and two datasets. The results for the other methods are taken from the respective publications. In comparison to those other methods, our method is able to surpasses the state of the art for NMI on the MNIST dataset with a value of 0.923 and performs comparably on others. It should also be noted, that compared to other state of the art methods, our results are more balanced with respect to the datasets. For instance, while TAGNet (Wang et al., 2016) defines the state of the art on the COIL20 dataset, it performs quite bad on MNIST.

Our results show that the taxonomy enables the creation of high-performing methods in a structured and analytical way. Researches are now able to selectively recombine or replace distinct aspects of existing methods instead of relying on a experiment and discovery based approach.

Figure 3 and 4 visualize the clustering spaces at different training stages of the proposed network. The clustering spaces are 120-dimensional and 320-dimensional for MNIST and COIL20, respectively. The true cluster labels are indicated by the different colors. The visualizations show that the proposed method results in a more clustering-friendly space compared to the original image space and the space before jointly training with the clustering loss.

6 CONCLUSION

In this paper, we proposed a universal taxonomy for clustering methods that utilize deep neural networks. The taxonomy was created by decomposing existing methods and analyzing the building blocks for properties that are indicative of desirable or undesirable results. Our case study shows that by using this taxonomy, researches and practitioners can derive new clustering methods in a systematic and analytical way to suit their task. The method we constructed as part of the case study was able to outperform the previous state of the art method on the MNIST dataset. Following the same approach, the proposed taxonomy can thus inspire a myriad of new methods.

REFERENCES

Cai, D., He, X., and Han, J. (2011). Locally consistent concept factorization for document clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):902–913.

Chen, D., Lv, J., and Yi, Z. (2017). Unsupervised multi-manifold clustering by learning deep representation. In *Workshops at the AAAI Conference on Artificial Intelligence*, pages 385–391.

Chen, G. (2015). Deep learning with nonparametric clustering. *arXiv preprint arXiv:1501.03084*.

Ding, C. and He, X. (2004). k-Means clustering via principal component analysis. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 29. ACM.

Dizaji, K. G., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5747–5756. IEEE.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hsu, C.-C. and Lin, C.-W. (2017). Cnn-based joint clustering and representation learning with feature drift compensation for large-scale image data. *arXiv preprint arXiv:1705.07091*.

Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. (2017). Learning discrete representations via information maximizing self augmented training. *arXiv preprint arXiv:1702.08720*.

Huang, P., Huang, Y., Wang, W., and Wang, L. (2014). Deep embedding network for clustering. In *International Conference on Pattern Recognition (ICPR)*, pages 1532–1537. IEEE.

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

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

LeCun, Y. (1998). The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Li, F., Qiao, H., Zhang, B., and Xi, X. (2017). Discriminatively boosted image clustering with fully convolutional auto-encoders. *arXiv preprint arXiv:1703.07980*.

Liu, H., Shao, M., Li, S., and Fu, Y. (2016). Infinite ensemble for image clustering. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1745–1754.

Lukic, Y., Vogt, C., Dürr, O., and Stadelmann, T. (2016). Speaker identification and clustering using convolutional neural networks. In *International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

Nene, S. A., Nayar, S. K., Murase, H., et al. (1996). Columbia object image library (COIL-20). Technical report.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, pages 849–856.

Premachandran, V. and Yuille, A. L. (2016). Unsupervised learning using generative adversarial training and clustering.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Saito, S. and Tan, R. T. (2017). Neural clustering: Concatenating layers for better projections.

Song, C., Liu, F., Huang, Y., Wang, L., and Tan, T. (2013). Auto-encoder based data clustering. In *Iberoamerican Congress on Pattern Recognition (CIARP)*, pages 117–124. Springer.

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

Trigeorgis, G., Bousmalis, K., Zafeiriou, S., and Schuller, B. (2014). A deep semi-nmf model for learning hidden representations. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1692–1700.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408.

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*, 11:2837–2854.

Wang, Z., Chang, S., Zhou, J., Wang, M., and Huang, T. S. (2016). Learning a task-specific deep architecture for clustering. In *Proceedings of the SIAM International Conference on Data Mining (ICDM)*, pages 369–377.

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

Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2016a). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *arXiv preprint arXiv:1610.04794*.

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

Zheng, Y., Tan, H., Tang, B., Zhou, H., et al. (2016). Variational deep embedding: A generative approach to clustering. *arXiv preprint arXiv:1611.05148*.