Data Science and Engineering (2020) 5:419–432
https://doi.org/10.1007/s41019-020-00134-0

DeepECT: The Deep Embedded Cluster Tree

Dominik Mautz[1] · **Claudia Plant**[2] · **Christian Böhm**[3]

Received: 2 April 2020 / Revised: 26 June 2020 / Accepted: 26 June 2020 / Published online: 14 July 2020

Abstract
The idea of combining the high representational power of deep learning techniques with clustering methods has gained much attention in recent years. Optimizing a clustering objective and the dataset representation simultaneously has been shown to be advantageous over separately optimizing them. So far, however, all proposed methods have been using a flat clustering strategy, with the actual number of clusters known a priori. In this paper, we propose the Deep Embedded Cluster Tree (DeepECT), the first divisive hierarchical embedded clustering method. The cluster tree does not need to know the actual number of clusters during optimization. Instead, the level of detail to be analyzed can be chosen afterward and for each sub-tree separately. An optional data-augmentation-based extension allows DeepECT to ignore prior-known invariances of the dataset, such as affine transformations in image data. We evaluate and show the advantages of DeepECT in extensive experiments.

Keywords Embedded clustering · Hierarchical clustering · Autoencoder · Deep learning

Abbreviations

ACC	Clustering accuracy
AE	Autoencoder
AE + Complete	AE combined with agglomerative clustering with complete-linkage
AE + Single	AE combined with agglomerative clustering with single-linkage
DEC	Deep Embedded Cluster algorithm [3]
IDEC	Improved Deep Embedded Cluster algorithm [5]
DeepECT	Deep Embedded Cluster Tree
DeepECT + Aug	DeepECT with the optional augmentation extension
DP	Dendrogram purity
Eq.	Equation
LP	Leaf purity
NMI	Normalized mutual information
ReLU	Rectified linear unit
URL	Uniform resource locator

✉ Dominik Mautz
mautz@dbs.ifi.lmu.de

Claudia Plant
claudia.plant@univie.ac.at

Christian Böhm
boehm@dbs.ifi.lmu.de

[1] LMU München, Munich, Germany

[2] Faculty of Computer Science, ds:UniVie, University of Vienna, Vienna, Austria

[3] MCML, LMU München, Munich, Germany

1 Introduction

Clustering algorithms are a fundamental tool for data mining tasks. However, of similar importance is the representation of the data to be clustered and this, in turn, depends on the data domain. In the last decade, deep learning techniques have achieved in areas that were previously very challenging for machine learning and data mining methods. These areas include images, graph structures, text, video, and audio. Many of these success stories have been made in the context of supervised learning. Further, neural network-based, unsupervised representation learning has made it possible to embed these challenging domains into spaces more accessible to classical data mining methods.

In recent years, the idea of simultaneously optimizing a clustering objective and the dataset representation has gained more traction. In this work, we call these methods either embedded clustering or deep clustering. The combined optimization holds the promise of improved results compared to two separate steps: During optimization, better feature representations are learned that enhance the cluster assignments; the cluster assignments, in turn,

provide information to improve the embedding. However, this also makes the task especially challenging, because, with each update of the embedding network, the embedded data space changes, and the clustering method has to adapt to this changing environment.

Typically, the embedding method is an autoencoder, a specific type of neural network. These networks learn to map a data space onto a latent, embedded space and a mapping back to the original space. Usually, these embeddings have a lower dimensionality than the original data domain and, therefore, are easier to visualize.

In this paper, we focus on a novel, specialized clustering layer, the Deep Embedded Cluster Tree (DeepECT) layer. We explicitly do not consider any specific autoencoder type that is used for the embedding. Instead, we only apply a generic feedforward autoencoder architecture and focus on the clustering layer. We expect that DeepECT profits to the same degree from better embeddings through domain-specific autoencoders (e.g., convolutional autoencoders) as other clustering algorithms would do.

DeepECT is inspired by classical hierarchical clustering. It simultaneously improves the embedded features and iteratively grows a cluster tree. In contrast to previous embedded flat clustering methods, the cluster tree represents a hierarchy of clusters that separate populations and subpopulations within the data. DeepECT grows top-down and assigns the data bottom-up. It is optimized with mini-batches, which also makes it suited for large datasets. The final cluster assignments are flexible and can be determined on a by-need basis.

Over other embedded flat clustering techniques, DeepECT has the advantage of not requiring the specification of the number of clusters to be found. This feature is much more crucial for embedded techniques than for traditional clustering settings because other embedded flat clustering techniques actively destroy structural information that is not captured through the clusters and the embedded space is actively optimized to reflect the selected number of clusters. Therefore, the cluster validity and consistency of these methods cannot be evaluated by measures such as the Silhouette coefficient. Even the inclusion of the autoencoder's reconstruction loss cannot fully overcome this behavior. For DeepECT, we can separately choose the level of detail we want to inspect during analysis—i.e., after optimization—for every local structure captured by a sub-tree. This feature is enabled through a new projection-based loss function that we use for DeepECT. It does not penalize orthogonal structures—not yet captured by the cluster tree—allowing that those structures can be found when we let the tree grow in subsequent steps of the optimization process.

We summarize the contributions of this work as follows:

- *Hierarchical clustering layer* We propose a novel clustering layer that builds a cluster tree in an embedded space. Both the embedding and the tree are trained simultaneously, and degenerated solutions are avoided—as opposed to other proposed methods. In contrast to other embedded clustering methods, DeepECT does not need the actual number of clusters during optimization. Instead, the level of detail can be chosen afterward.
- *Novel optimization strategy* We propose a novel projection-based optimization strategy that enhances the cluster boundaries and preserves orthogonal structural information.
- *Optional augmentation* An optional extension, utilizing augmentation methods, allows ignoring known invariances within the data.

2 Deep Embedded Cluster Tree

2.1 Overview

In this section, we discuss the deep embedded cluster tree (DeepECT). An implementation can be found at https://dmm.dbs.ifi.lmu.de/downloads. We focus on the novel DeepECT clustering layer and assume that we are given some generic autoencoder that transforms a data point $\mathbf{x}$ via an encoding function $\text{enc}(\cdot)$ onto an embedded space and a decoding function $\text{dec}(\cdot)$ that reconstructs an embedded data point back onto the original space. This autoencoder has been pre-trained using a differentiable loss function that penalizes the reconstruction error for a data mini-batch $\mathcal{B}$. A popular choice is the mean squared error loss:

$$L^{\text{rec}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \|\mathbf{x} - \text{dec}(\text{enc}(\mathbf{x}))\|_2^2. \tag{1}$$

Yet, the selected loss function is entirely domain-specific, as is the choice of the specific autoencoder architecture.

We combine the reconstruction loss and the clustering loss of the DeepECT layer into a final loss term. Figure 1 shows a sketch of the overall architecture, including the autoencoder and an illustration of the embedded cluster tree. The constructed tree is a binary tree; each node is either a leaf node or an inner node—we call this a split node—with two nodes as children. Each node has a vector representing a point in the embedded space and serving as a cluster center. We use this center as a representative of the assigned data. We combine ideas from both top-down and bottom-up hierarchical strategies. The data point assignment to the nodes follows a bottom-up strategy, whereas the tree grows in a top-down manner. We start with a single root node that we assign all data points to and iteratively split nodes until the tree has the desired number of leaf nodes (or some other user-specified

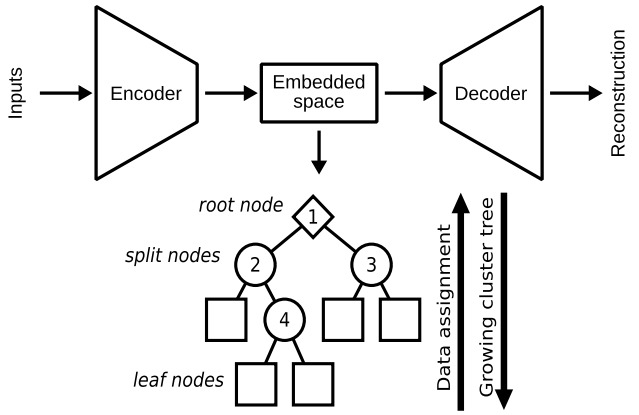


Fig. 1 The figure shows an illustration of the combined architecture of the DeepECT clustering algorithm and an autoencoder. The autoencoder network transforms the input data into an embedded space, where DeepECT is used to cluster it. The numbers in the illustrated cluster tree indicate the split order

criteria is met). Algorithm 1 shows the pseudo-code of the optimization procedure.

Algorithm 1: DeepECT Optimization

Input: dataset $\mathcal{D}$; max number of leave nodes: L; split interval: T; maximum iterations: MaxIter; pruning threshold: P
Output: embedded cluster tree
Pre-train autoencoder
Create root node of tree
for iter $\in [0, \ldots, \text{MaxIter}]$ **do**
  foreach $n \in$ Nodes **do**
    if $w_n <$ P **then**
      Remove node and replace parent node with the sibling node (see Section II-G)
    end
  end
  if iter % T $==$ 0 **then**
    Grow tree by splitting the leaf node with sum of highest squared distances (see Section II-F)
  end
  Select minibatch from dataset
  Assign data points to leaf nodes and split nodes (see Section II-B)
  Determine loss according to Eq. (6)
  Optimize parameters with an gradient based optimizer and Eq. (4)
end

2.2 Object Assignment

The data point assignment is executed in a bottom-up fashion. We assign each data point $\mathbf{x}$ of a mini-batch $\mathcal{B}$ to its closest leaf node:

$$\arg\min_{n \in \mathcal{L}} \|\mu_n - \text{enc}(\mathbf{x})\|_2,$$

where $\mathcal{L}$ is the set of all leaf nodes. Each split node obtains the data points assigned to its two child nodes.

2.3 Node Center Loss

Since the embedded space also changes between update steps, we have to adapt the nodes' centers to the changing environment. The centers of the leaf nodes are trainable parameters, and we optimize them accordingly. We do this by penalizing the squared difference between the leaf nodes' centers μ_n and the mean value of the data points of the mini-batch assigned to this leaf node:

$$L^{\text{NC}} = \frac{1}{|\mathcal{L}|} \sum_{n \in \mathcal{L}} \left\| \mu_n - \frac{1}{|\mathcal{B}_n|} \sum_{\mathbf{x} \in \mathcal{B}_n} \text{sg}[\text{enc}(\mathbf{x})] \right\|_2, \tag{2}$$

where we denote the set of all data points assigned to a node n as $\mathcal{B}_n$ and regard the embedded data points and the encoder as constant, which we indicate by the stop gradient operator sg[·]. Keeping the embedding constant ensures that we only push the center of a node toward the data mean, but not the data points toward the node center. Updating the embeddings is the purpose of the next loss function, and we explain in the next section the reason behind it. We divide by the number of leaf nodes to get the mean loss over the leaf nodes.

The centers of the split nodes are not trainable parameters but are determined based on the leaf nodes in the node's sub-tree. Calculating the centers of the inner nodes has two advantages. First, we have fewer parameters the optimization algorithm has to keep track of. Second, representing the inner node as trainable parameters can lead to situations where a split node center and the two centers of the respective children become inconsistent, i.e., the center of a split node may not lay on or even near the connection line between the centers of its child nodes. Defining the centers of split nodes based on their child nodes centers circumvents this issue.

We determine the split node centers as weighted averages over the child-nodes:

$$\mu_n = \frac{1}{w_\text{l} + w_\text{r}} (w_\text{l} \mu_\text{l} + w_\text{r} \mu_\text{r}), \tag{3}$$

where l and r are the indices of the left and right child of this node and w_l and w_r represent their weights. The weights represent the number of data points assigned to this node over several update steps and guarantee that a calculated center is an suitable representative for both children, even in unbalanced situations.

Finally, we have to update the weights w_n of each node used in Eq. (3). We represent the weights as an exponential

moving average over the number of assigned data points within each mini-batch. The weight of node n at iteration t is updated in the following way:

$$w_n^{(t)} = 0.5 w_n^{(t-1)} + 0.5|\mathcal{B}_n|. \tag{4}$$

The 50:50 split showed a suitable trade-off in our initial experiments, and we, therefore, kept it for all of our experiments.

2.4 Node Data Compression Loss

Our optimization goal for the DeepECT-layer is to strengthen the boundary—i.e., enlarging the margin—between the data point partitions assigned to each pair of sibling-nodes within the cluster tree. We achieve this through a compression loss, in which we penalize the distance between data points and their assigned node centers. This penalization ensures that data points are embedded closer to the assigned node centers in subsequent iterations. A naive idea would be to penalize the Euclidean distance between the center of a node and its respective data points. However, this has the adverse effect that structural information orthogonal to the line connecting the two centroids is destroyed. These orthogonal structures might be relevant to the ancestors of these nodes. Figure 2 shows the situation with this naive approach. The plot illustrates an example of an embedded space with three ground truth clusters. The cluster tree in this example has only one root node and two leaf nodes. The hollow black squares represent the two centers of the leaf nodes. We can see that over several optimization steps, and we indeed get the desired effect that the margin between the two populations represented by the two leaf nodes is increased. However, at the same time, the structural information orthogonal to the two node centers is also destroyed. This destruction occurs although the reconstruction loss should counteract this behavior.

Therefore, we propose to penalize the distance between the data point and the center of a node when projected onto the line connecting the center of the node and its sibling center. We determine the projection onto the connection line with the following formula:

$$\rho_n = \mathrm{sg}\left[\frac{\mu_n - \mu_m}{\|\mu_n - \mu_m\|_2}\right],$$

where n is the node's index, for which we need the projection direction and m is the index of its sibling node. In all cases, where we use this projection, we regard it as a constant. We define the compression loss as follows:

$$L^{\mathrm{DC}} = \frac{1}{|\mathcal{N}| \cdot |\mathcal{B}|} \sum_{n \in \mathcal{N}} \sum_{\mathbf{x} \in \mathcal{B}_n} |\rho_n^{\mathsf{T}} \big(\mathrm{sg}[\mu_n] - \mathrm{enc}(\mathbf{x})\big)|, \tag{5}$$

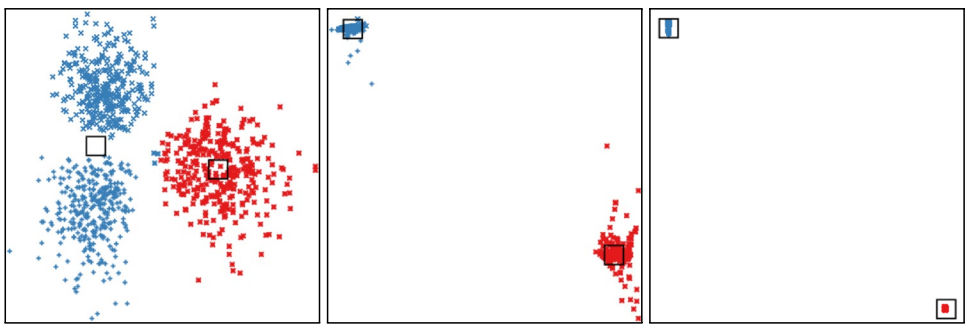
where $\mathcal{N}$ is the set of all node indices excluding the root node and we regard both ρ_n and μ_n as constants. Further, we use the absolute value instead of the Euclidean norm because the term inside is—due to the projection—a scalar value. By dividing through the number of nodes and samples in the batch, we get the mean loss for each data point and node.

The effect of optimizing the projected compression loss is shown in Fig. 3. Again, we can see that the margin between the two node centers increases, but—in contrast to the unprojected example—the structural information orthogonal to the connection line is kept intact. The objects on the left side still show the structure of two distinctive groups that was lost in the naive unprojected version.

2.5 Complete Loss

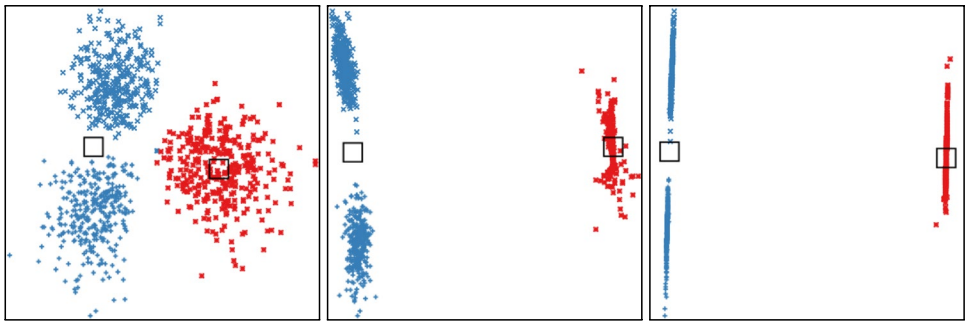
The complete loss function combines the three above-defined losses: (a) the autoencoder reconstruction loss L^{rec} to preserve local structures, (b) the node center loss L^{NC} to adapt the node centers to a changed embedded space, and (c) the data compression loss that improves the cluster separation L^{DC}. We combine all these losses by summing them up:

$$L = L^{\mathrm{DC}} + L^{\mathrm{NC}} + L^{\mathrm{rec}}, \tag{6}$$



(a) Step 1 (b) Step 100 (c) Step 1000

Fig. 2 The plots illustrate the deficiencies of the compression loss without projection. The data point colors represent leaf node assignments. We can see that orthogonal information is destroyed. Therefore, splitting one of the leaf nodes would not result in sufficiently large margins between the two potential leaf nodes



(a) Step 1 (b) Step 100 (c) Step 1000

Fig. 3 The plots show the effect of the compression loss with projection as we use it in DeepECT. We can see that structures orthogonal to the two leaf node centers are preserved. (Colors represent leaf node assignments)

where we refrain from introducing weights between the different losses for simplicity.

2.6 Growing the Tree

We start the optimization with only a single root node—that is also a split node—and grow the tree by splitting nodes after a certain number of update steps until we reach a previously defined number of leaf nodes.

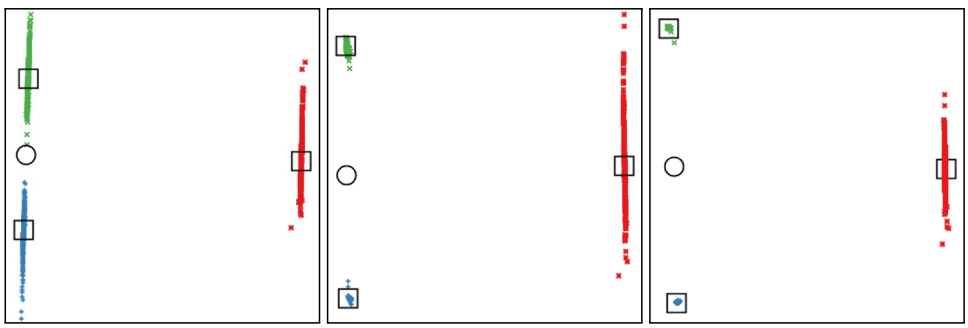
Growing a tree by one leaf node is straight forward. We transform the dataset (or a representative sub-sample of it) onto the embedded space. Then, we determine the leaf node with the highest sum of squared distances between its center and the assigned data points. We selected this rule because it provides a good balance between the number of data points and data variance for this cluster.

Next, we split the selected node and attach two new leaf nodes to it as children. We determine the initial centers for these new leaf nodes by applying two-means (k-means with $k = 2$) to the assigned data points.

Our experiments show that usually, 500–1000 mini-batch update steps between splits are sufficient to ensure that the embedded space has adapted to the new leaf nodes. Figure 4 shows what happens when we split the example data shown in Fig. 3 after 1000 steps [diagram (c)] and optimize it further. The black circle represents the computed center of the split node created by splitting the leaf node on the left side in the plots shown in Fig. 3. We can see that the compression loss now also strengthens the margin between the two clusters previously represented by just one leaf node.

2.7 Tree Pruning

The alternating assignment and update steps—which are utilized by almost all centroid-based clustering methods—are susceptible to degenerated situations, in which a cluster center does not get assigned any data points. This situation is more severe in the context of deep clustering because from one iteration to the next, the embeddings may change considerably. Most proposed flat clustering methods ignore these degenerate situations completely [1]. We avoid these degenerate results in DeepECT by pruning the tree during optimization. When we find ourselves in a situation, where we do not assign a node any data points over many optimization steps—i.e., the node dies out —all data points the parent gets assigned originate from the sibling node. Therefore, we can replace this parent node with the sibling and remove the dead node from the tree. We can detect this behavior if the exponential moving average w converges toward zero. The actual value should be set depending on the batch size and the number of leaf nodes. For our experiments, we set the threshold value for such a pruning to $w < 0.1$.



(a) Step 1001 (b) Step 1100 (c) Step 2000

Fig. 4 The plots show the effect of the projected compression loss after splitting the left leaf node after 1000 steps (as shown in Fig. 3). We can see that the data points on the left are compressed into two distinctive clusters, whereas the single leaf node on the right still preserves its orthogonal structure

Pruning the tree reduces its complexity and helps to preserve structural information by preventing the compression loss from acting on a 'living' node against its 'dead' sibling.

2.8 Optional: Extension with Input Augmentation

Image data augmentation has been shown to improve the accuracy of supervised learning tasks considerably [2]. Thereby, the algorithm learns to ignore known invariances within the data. Examples are rotations or translations of objects in images, a slight shift of the window in time series, or synonym substitution in text data.

In this section, we show a simple extension of DeepECT that exploits such an augmentation for the unsupervised clustering task. The key idea is that a (randomly) augmented object aug($\mathbf{x}$)—which we consider as constant—must be assigned to the same nodes as the original object $\mathbf{x}$. We can then penalize the distance between the node centers and both the original and the augmented objects in the same fashion as in Eq. 5:

$$L^{\mathrm{DCA}} = \frac{1}{|\mathcal{N}| \cdot |\mathcal{B}|} \sum_{n \in \mathcal{N}} \sum_{\mathbf{x} \in \mathcal{B}_n} \left[|\rho_n^{\mathsf{T}}(\mathrm{sg}[\mu_n] - \mathrm{enc}(\mathbf{x}))| + |\rho_n^{\mathsf{T}}(\mathrm{sg}[\mu_n] - \mathrm{enc}(\mathrm{sg}[\mathrm{aug}(\mathbf{x})]))| \right]. \tag{7}$$

We define the center loss by the average of the original and the augmented data and replace Eq. 2:

$$L^{\mathrm{NCA}} = \frac{1}{|\mathcal{L}|} \sum_{n \in \mathcal{L}} \left\| \mu_n - \mathrm{sg}\left[\frac{1}{2|\mathcal{B}_n|} \sum_{\mathbf{x} \in \mathcal{B}_n} (\mathrm{enc}(\mathbf{x}) + \mathrm{enc}(\mathrm{aug}(\mathbf{x}))) \right] \right\|_2, \tag{8}$$

The complete loss of DeepECT with augmentation is then defined as:

$$L = L^{\mathrm{DCA}} + L^{\mathrm{NCA}} + L^{\mathrm{recA}}, \tag{9}$$

where L^{recA} is the reconstruction loss for the original batch and the augmented data objects.

3 Experiments

We evaluate our proposed method DeepECT on four commonly used deep learning datasets: MNIST, USPS, Fashion-MNIST, and Reuters. Table 1 shows the statistics of all datasets used in the experiments. MNIST and USPS are both image datasets containing handwritten digits. The Fashion-MNIST dataset contains images of fashion products, such as images of clothing, shoes, and bags. The Reuters dataset contains news articles in four top categories, and we use the same representation as described in [3].

Table 1 Statistics of datasets used in the experiments

Name	Type	# Points	# Dimensions	# Classes
MNIST	Image	70,000	784	10
USPS	Image	9298	256	10
Fashion-MNIST	Image	70,000	784	10
Reuters	Text	685,071	2000	4

3.1 Experimental Setup

We focus our experiments on the evaluation of our new clustering layer. Therefore, we refrain from using more elaborated autoencoder architectures. Instead, we use the same generic fully connected autoencoder layout for all experiments, as used in [3]. As mentioned before, we expect that all methods would gain equally from more sophisticated and domain-specific architectures. However, a standard autoencoder architecture is sufficient to show the viability of DeepECT compared to the baseline competitors. Hence, we use the same generic autoencoder architecture, as proposed in [4] and which also used in [3, 5] for the purpose of clustering the embedded space. The feedforward encoder in this architecture has the dimensions d-500–500–2000–10, and the decoder network has a mirrored layout. We use ReLU activations and the mean squared error reconstruction loss from Eq. (1).

We pre-train ten autoencoders for each dataset and use these same pre-trained networks for all experiments and comparison methods. Using these pre-trained autoencoders ensures that each method has the same starting conditions for the embedded space and that variations in the clustering quality do not merely stem from qualitatively different autoencoders. The pre-training setup is similar to the one described in [3]. We pre-train the autoencoders as denoising autoencoders with a 20% corruption rate. First, we perform a layer-wise pre-training with dropout after each layer (with a rate of 20%) and 20,000 steps per layer. Then, we fine-tune the whole network for 50,000 steps without dropout. We use input corruption only for the pre-training and not for the actual optimization of DeepECT and its baseline methods. For all experiments, we use Adam [6] (learning rate $= 0.0001$, $\beta_1 = 0.9, \beta_2 = 0.999$) as the optimization algorithm and a mini-batch size of 256 samples. For the combined optimization, we train for additional 50,000 iterations to ensure convergence.

For DeepECT, our initial experiments with synthetic data showed that splitting the tree every 500 optimization steps yields promising results and more extended step sizes did not further increase the performance. For this

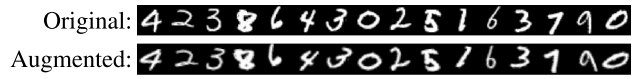


Fig. 5 The plots show a sample of original MNIST digits and a randomly augmented version

reason, we keep this schedule without adjusting it for the experiments on real-world datasets. The same applies to the pruning threshold mentioned in Sect. 2.7. For MNIST, Fashion-MNIST, and USPS, we grow the trees until they contain twenty leaf nodes. For the Reuters dataset, we set the maximal number of leaf nodes to twelve because it has fewer ground truth clusters. This way, we have two times and three times the actual number of clusters. We consider these values sufficient to capture essential structures of the selected datasets for the purpose of this paper. We use the same number of leaf nodes for the hierarchical baseline methods.

For the image datasets, we additionally experimented with the augmentation extension DeepECT + Aug. We start with the same pre-trained autoencoders as in the other experiments. Further, we stick to the same optimization schedule as described above for the experiments with the non-augmented versions of DeepECT. In each iteration, we use the original mini-batch and its augmented counterpart to optimize the loss function in Eq. 9, instead of the non-augmented loss in Eq. 6. We create the augmented version of each image of a mini-batch, by applying on-the-fly a random affine transformation. The affine transformation randomly rotates and shears the image in the range of $[-15;15]$ degrees. Also, it moves the digit randomly up to two pixels in any direction. Figure 5 shows an example of this augmentation for MNIST.

3.2 Evaluation Methods

We evaluate the cluster hierarchy of DeepECT with the dendrogram purity (DP) and leaf purity (LP) measure. We describe both below. Further, we evaluate the cluster tree against flat baseline methods. For this, we use the well-known normalized mutual information (NMI) and clustering accuracy (ACC) [3]. We include these for completeness and to show that DeepECT is also competitive in scenarios, where one expects a flat cluster structure and knows the actual number of clusters in dataset. To determine a k cluster partition from a cluster tree, we use the assignments to the k nodes that were leaf nodes after the first $k-1$ splits.

3.2.1 Dendrogram Purity

The dendrogram purity measure [7, 8] can be used to evaluate the cluster tree against a flat ground truth partition. It is the expected purity of the sub-tree given by the least common ancestor node for two randomly sampled data points of the same class. It is 1.0 if and only if all data points belonging to one class in the ground truth are assigned to some pure sub-tree, and it approaches 0 for randomly generated trees.

The explicit formula is defined in [8] as:

$$\mathrm{DP} = \frac{1}{|\mathcal{P}|} \sum_{k=1}^{K} \sum_{\substack{x,y \in C_k \\ \wedge x \neq y}} \mathrm{pur}(\mathrm{dan}(\mathrm{lca}(x,y)), C_k),$$

where $C_1, \ldots, C_K$ are the data point sets corresponding to the ground truth classes, $\mathrm{lca}(x,y)$ is the least common ancestor node of x and y in the cluster tree, $\mathrm{dan}(z)$ is the set of data points assigned to the node z in the cluster tree, $\mathrm{pur}(S,T) = |S \cap T|/|S|$ is the purity measure, and $\mathcal{P} = \{(x,y) \mid \exists C \in \{C_1, \ldots, C_K\} : x,y \in C \wedge x \neq y\}$ is the set of all data point pairs that belong to the same class. The dendrogram purity can be computed efficiently and accurately in a bottom-up recursion on the cluster tree.

3.2.2 Leaf Purity

Besides using dendrogram purity, we introduce another measure that we call leaf purity (LP). It is the weighted-average purity of the leaf nodes w.r.t. to the majority class of the objects assigned to a leaf node, given by the formula:

$$\mathrm{LP} = \frac{1}{|\mathcal{D}|} \sum_{L \in \mathcal{L}_\mathcal{D}} |L| \max_{C \in \{C_1, \ldots, C_K\}} \mathrm{pur}(L, C),$$

where $\mathcal{L}_\mathcal{D}$ is the set of sets containing the data points assigned to the leaf nodes.

3.2.3 Tree Height Dependence of Purity Measures

Comparing dendrogram and leaf purity of two cluster trees is only directly possible if both trees have the same number of leaf nodes. However, sub-trees can always be collapsed into leaf nodes to fulfill this requirement. Therefore, we collapse the bottom-up linkage-trees of the baseline methods—in the order of linkage—by compressing sub-trees into leaf nodes until we have the same number of merge steps

left as split-nodes in the top-down trees of DeepECT and Bisecting-*K*-means. This process ensures that both methods are comparable w.r.t. the hierarchical evaluation measures.

3.3 Hierarchical Clustering Baselines

As a baseline for evaluating the hierarchical properties, we cluster the embedded data with the classical hierarchical clustering algorithms bisecting-*k*-means (AE + Bisecting), single-linkage (AE + Single), and complete-linkage (AE + Complete). Since none of these classical algorithms can optimize the embedded space, we also explore the simple idea of combining the flat embedded clustering algorithm IDEC [5] with single-linkage and complete-linkage. IDEC is a method that combines the clustering layer of DEC [3] with the reconstruction loss of the autoencoder. First, we run IDEC with the number of clusters set to a value higher than the expected number of clusters—in our case, we set it equal to the maximal number of leaf nodes we use for DeepECT. Then, we consider these IDEC cluster centers as representatives of the assigned data points and try to recover a hierarchical clustering structure by performing single-linkage and complete-linkage on the cluster centers (IDEC + Single and IDEC + Complete). A similar technique is proposed in [9] for classical, non-embedded settings with *k*-means instead of IDEC.

3.4 Flat Clustering Baselines

As a baseline for evaluating the performance of DeepECT in a flat clustering setting, we use *k*-means on the embedded data of the pre-trained autoencoder (AE+*k*-means) and IDEC [5]. If we ignore the advantages of more domain-specific and sophisticated autoencoder architectures, IDEC is currently one of the best embedded-clustering methods. In contrast to DeepECT, we have to set the actual number of clusters in the ground truth during optimization for IDEC and *k*-means. Further, we set the hyperparameter of IDEC for the reconstruction loss to 0.1 as described in [5].

3.5 General Results

The general results—averaged over the ten pre-trained autoencoders—for the hierarchical evaluation using dendrogram purity and leaf purity measures for DeepECT and the hierarchical baseline algorithms are shown in Table 2. DeepECT consistently produces cluster trees of high quality and is the top-performing algorithm by a wide margin. We can also see that the augmentation extension further improves the results considerably for MNIST and USPS. The results of DeepECT with and without the augmentation extension for the Fashion-MNIST dataset are similar because the dataset authors chose to pre-process all images such that each fashion item has a normalized representation. The results of the classical methods can be explained by their inability to enhance the embedding. The leaf purity values for DeepECT indicate that the method is able to create homogeneous sub-populations. If we compare the leaf purity values of DeepECT and the hierarchical IDEC + Center-linkage variants to the other baselines' leaf purity values, we can see that the combined optimization of the clustering and autoencoder—of both methods—indeed improves the homogeneity of local structures. However, the IDEC + Center-linkage is also unable to extract a coherent hierarchical structure.

Table 3 shows the experimental results for the flat clustering comparison methods based on the same pre-trained autoencoders. Since we use the same pre-trained autoencoders, we can directly see the influence of the respective clustering objective. Both IDEC and DeepECT benefit from the combined optimization compared to *k*-means, which cannot optimize the embedding. Table 4 shows the results of more centroid-based clustering methods taken from the respective publication. More details about these methods can be

Table 2 Our experiments show that DeepECT is the top-performing algorithm in terms of dendrogram purity (DP) and leaf purity (LP)

Method	MNIST		USPS		Fashion-MNIST		Reuters	
	DP	LP	DP	LP	DP	LP	DP	LP
DeepECT	*0.82 ± 0.03*	*0.93 ± 0.02*	*0.72 ± 0.03*	*0.85 ± 0.04*	**0.47 ± 0.03**	**0.67 ± 0.03**	**0.71 ± 0.05**	**0.84 ± 0.05**
DeepECT + Aug	**0.94 ± 0.02**	**0.98 ± 0.01**	**0.89 ± 0.03**	**0.94 ± 0.03**	*0.44 ± 0.04*	0.60 ± 0.05	n.a.	n.a.
IDEC + Single	0.39 ± 0.09	0.60 ± 0.08	0.61 ± 0.09	0.72 ± 0.06	0.34 ± 0.04	0.54 ± 0.03	0.52 ± 0.04	0.67 ± 0.04
IDEC + Complete	0.40 ± 0.05	0.60 ± 0.08	0.61 ± 0.09	0.72 ± 0.06	0.35 ± 0.03	0.54 ± 0.03	0.52 ± 0.04	0.67 ± 0.04
AE + Bisecting	0.53 ± 0.02	0.78 ± 0.02	0.39 ± 0.02	0.69 ± 0.02	0.38 ± 0.02	*0.64 ± 0.03*	*0.63 ± 0.03*	*0.76 ± 0.03*
AE + Single	0.11 ± 0.00	0.11 ± 0.00	0.12 ± 0.00	0.17 ± 0.00	0.10 ± 0.00	0.10± 0.00	0.36 ± 0.00*	0.44 ± 0.00*
AE + Complete	0.25 ± 0.04	0.45 ± 0.05	0.20 ± 0.04	0.40 ± 0.07	0.26 ±0.04	0.44 ±0.03	0.41 ± 0.02*	0.54 ± 0.04*

For results marked with a *, we had to use a random subset of the dataset with 100,000 objects and the same class distribution, because of memory limitations. The values are averages and the standard deviation for the ten pre-trained autoencoders. Best value in bold; runner up is italicized

Table 3 This table shows that DeepECT is even competitive when compared to flat clustering methods that are given the true number of clusters during optimization and have therefore an unfair and unrealistic advantage over DeepECT

Method	MNIST		USPS		Fashion-MNIST		Reuters	
	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC
DeepECT	0.83 ± 0.02	*0.85 ± 0.04*	0.71 ± 0.02	0.71 ± 0.04	**0.60 ± 0.05**	*0.52 ± 0.06*	*0.47 ± 0.05*	**0.72 ± 0.04**
DeepECT + Aug	**0.93 ± 0.02**	**0.95 ± 0.04**	**0.86 ± 0.02**	**0.82 ± 0.06**	*0.59 ± 0.04*	0.50 ±0.05	n.a.	n.a.
IDEC	*0.86 ± 0.01*	*0.85 ± 0.03*	*0.76 ± 0.02*	*0.74 ± 0.03*	0.58 ±0.02	**0.53 ± 0.03**	**0.50 ± 0.03**	*0.67 ± 0.03*
AE + *k*-means	0.70 ± 0.02	0.77 ± 0.02	0.49 ± 0.03	0.56 ± 0.03	0.52 ± 0.02	0.48 ± 0.02	0.39 ± 0.07	0.65 ± 0.05

All methods started from the same pre-trained autoencoders. The values are averages and the standard deviation for the ten pre-trained autoencoders. Best value in bold; runner up is italicized

Table 4 This table shows DeepECT in the context of other deep clustering methods using *k*-means like flat clustering objectives.

Method	Architecture	Reported	MNIST	USPS	Reuters
DeepECT	Fully connected AE	Avg (best)	0.85 (0.90)	0.71 (0.77)	0.72 (0.77)
DeepECT + Aug	Fully connected AE	Avg (best)	0.95 (0.98)	0.82 (0.88)	–
DEC [3]	Fully connected encoder	Best	0.84	–	0.72
DBC [10]	Convolutional encoder + DEC objective	Best	0.96	0.74	–
IDEC [5]	Fully connected AE + DEC objective	Best	0.88	0.76	0.76
DCN [11]	Fully connected AE	Best	0.83	–	0.80
DEPICT [12]	Convolution AE	Best	0.96	0.96	–
SDEC [13]	Fully connected AE + Semi-Supervised DEC	Best	0.86	0.76	–
best DKM [14]	Fully connected AE	Avg	0.85	0.75	0.58

The shown clustering accuracy values are taken from the respective publication and therefore use different autoencoders (also different architectures). Further, we indicate if the paper reports the highest achieved (best) or average (avg) value. All of these methods have the advantage over DeepECT that they were provided with a dataset's actual number of clusters during optimization

found in Sect. 4. We can see that DeepECT also performs well compared to these methods. However, we can also see that the autoencoder architecture influences the clustering result considerably. For instance, DBC differs from DEC only by the use of a convolutional autoencoder but achieves superior results. Yet, the selected autoencoder architecture is independent to the selected clustering layer.

Of course, this comparison of flat clustering objectives and DeepECT is unfair toward the latter, because the competing methods are given the true number of clusters during optimization, whereas for DeepECT, we only use this information during evaluation. Nevertheless, we can see that the ordinary version of DeepECT can keep up with these methods in terms of raw NMI and ACC measures and that the augmentation extension DeepECT + Aug shows substantial improvements over the results of DeepECT, because it can ignore known invariances within the data. These results show that DeepECT is also competitive in scenarios, where one expects a flat cluster structure, but does not know the number of clusters and inspects the cluster tree recursively.

3.6 Detailed Evaluation

In this section, we take a closer look at the resulting DeepECT-trees for the above datasets. Since the USPS dataset's findings are comparable to the one of MNIST—as both represent handwritten digits—we omit these results for brevity.

3.6.1 MNIST Results

A closer look at the resulting DeepECT-trees for the MNIST dataset shows some exciting properties of different subpopulations within the handwritten digits. Two illustrative examples are shown in Fig. 6 and can be found in the ordinary and augmented extension of DeepECT. The node purity of the depicted sub-trees for the digit 7' is 98% and containing almost all instances of this class. It contains two leaf nodes. One leaf node shows sevens with a small crossbar as it is commonly written in Europe, the other leaf node shows this digit as it is more commonly written in the USA. The second sub-tree contains almost all instances of the digit '2' with a purity of 97%. This sub-tree also contains two leaf nodes, each with specific characteristics. The first leaf node contains instances that are more curly and have a distinctive

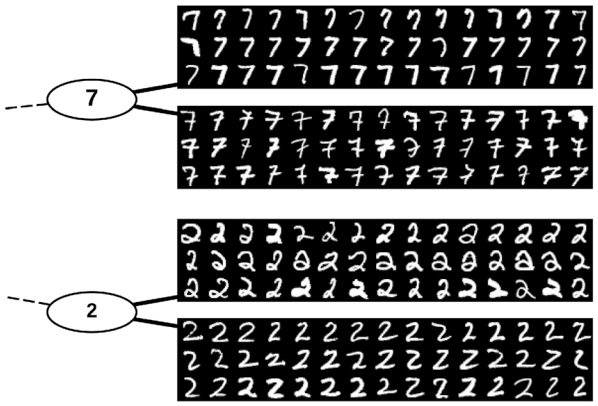


Fig. 6 The plots show two extracted sub-trees from interesting sub-populations of the MNIST dataset found by DeepECT. These are the digits seven (with and without a middle-crossbar) and two (a curly and a 'streamlined' version, looking more like the character 'Z'). The shown digits are randomly sampled

loop at the bottom part. The second leaf node contains a more 'streamlined' version of this digit, looking like the character 'Z.' The shown sub-trees build a natural hierarchy for the respective digit, and one can easily imagine that these findings can be of interest to a researcher. Other shape depending groupings of digits can also be found in lower parts of the tree, for instance, the written versions of the digits '4' and '9' share many characteristics. Consequently, they often can be found grouped together as a sub-tree containing only these two digit types.

3.6.2 Reuters Results

The Reuters dataset contains four unbalanced top categories (first-level labels) with the following class distribution: Cooperate/Industrial with 44%, Government/Social with 24%, Markets with 24%, and Economics with 8%. This dataset is explained in more detail in [15]. The categories for each news article were chosen by hand and are, therefore, to some extent subjective. Further, each top category has several additional overlapping sub-categories (second-level labels)—and sub-sub-categories (third-level labels)—with over 96% of the articles belonging to two or more sub-categories. Table 5 shows a DeepECT result for this dataset. We can see that the first two splits separate most of the Government/Social—sub-tree starting at the node 3—and Markets—sub-tree starting at the node 5—categories from the other two categories. The Government/Social sub-tree then differentiates further into topics of the sub-categories such as sports, war and crime, domestic and international politics. The Markets category also further differentiates into different aspects of the respective sub-categories. For instance, the leaf nodes in the last two rows are concerned with different sub-sub-categories of the sub-category Commodity Markets. The leaf nodes in the middle are mostly related to Corporate/Industrial and Economics. They are not as well separated as the other two sub-trees. Yet, even there, we can find interesting leaf nodes. For instance, the seventh leaf node (row) from the top shares news articles labeled with the sub-categories Performance (of Corporate/Industrial) and the Economic Performance (of Economics) and it seems reasonable to expect related words for those two sub-sub-categories.

3.6.3 Fashion-MNIST Results

The Fashion-MNIST contains ten different classes of clothes, shoes and bags, namely T-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. A resulting cluster tree of our method is shown in Fig. 7. The leaf nodes are represented as randomly sampled objects

Table 5 This table shows a cluster tree for the Reuters dataset

Cluster tree	Corporate/industrial	Markets	Government/social	Economics	Sub-categories (overlapping ground truth)
	–	–	99.5%	–	Sports
	–	–	99.2%	–	War/Civil-War, Crime/Law-Enf., Domestic Politics, International Relations, Disasters/Accidents
	1.1%	–	97.6%	1.1%	Domestic Politics, International Relations, Elections, Defence, War/Civil-War, Personalities/People
	–	–	99.2%	–	International Relations, Domestic Politics, War/Civil-War, European Community
	72.1%	9.6%	–	17.8%	Performance, Government Finance, Funding/Capital
	78.7%	7.5%	8.8%	5.0%	Performance, Ownership Changes, Funding/Capital, Markets/Marketing, Strategy/Plans
	54.7%	19.7%	1.6%	2.4%	Performance, Equity Markets, Money Markets, Monetary/Economic, Economic Performance
	25.0%	64.6%	10.4%	–	Commodity Markets, Performance, Weather, Capacity/Facilities, Equity Markets
	–	83.4%	–	15.8%	Money Markets, Leading Indicator
	1.7%	97.2%	–	–	Equity Markets, Money Markets, Bond Markets
	–	99.4%	–	–	Commodity Markets
	–	95.6%	–	4.6%	Commodity Markets, Money Markets, Bond Markets

The first column shows the cluster tree itself, and each row represents one leaf node. The next four columns show the percentages of documents for the four top categories (first-level labels) assigned to each leaf node. For clarity, we omit the values for occurrences below 1%. The last column shows the most frequent sub-categories (second-level labels) for each leaf node with more than 5% occurrences (max. five)

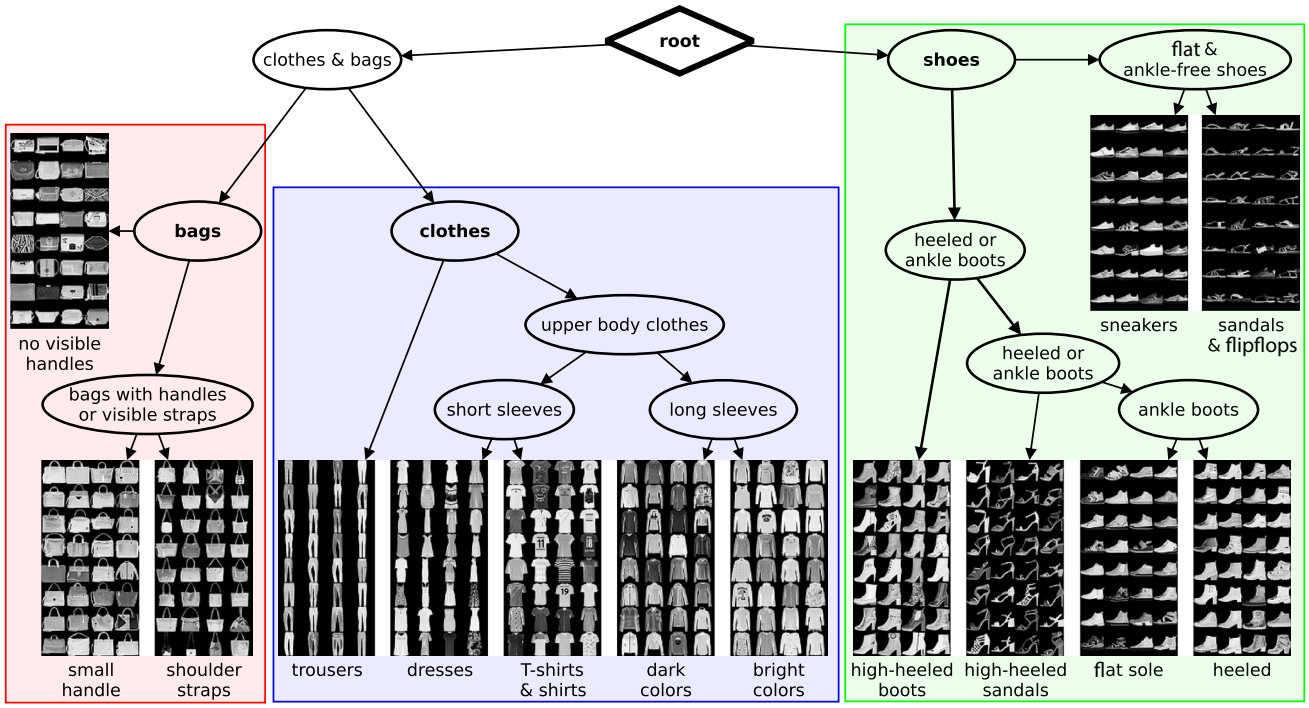


Fig. 7 The diagram shows a cluster tree for the Fashion-MNIST dataset. Each leaf node shows randomly sampled objects assigned to it. The labels are interpretations by the authors. The colored areas highlight the three dominant sub-trees representing three types of objects found in the dataset: bags, clothes, and shoes

assigned to it. The labels of each node are our interpretation based on the objects assigned to the respective node. We can see that DeepECT found an entirely natural-looking hierarchy within this dataset. First, the images are split into three categories: *clothes*, *shoes*, and *bags*. We highlighted these sub-trees with colored areas. Within each sub-tree, we can find natural hierarchies. The category of *bags* distinguishes between bags without a visible strap/handle, bags with small handles, and bags with a shoulder strap. The ground truth does not distinguish between these types of bags and assigns them all to the same class. The *clothes* category is first divided into trousers and clothes for the upper body. These are then again partitioned into short and long sleeves. Here, the length of the sleeve must be seen relative to the total length of the respective garment because each item is normalized to appear of the same size within the image, i.e., dresses and shirts appear to be of the same size. The *shoe* category also shows some interesting characteristics. First, smaller and bigger shoes are distinguished. The smaller shoes are then further divided into sandals and sneakers. The bigger shoes have either a flat sole, a small heel, or are high-heeled. Building the hierarchy based on these features runs against the ground truth classes of sneakers, sandals, and ankle boots. Nevertheless, it is—from an appearance perspective—a valid and informative hierarchy for shoes.

3.7 Applicability for Prediction Tasks on MNIST

We also evaluate DeepECT in a prediction task. Thereby, we keep the autoencoders and the clustering optimization procedure as described above. In contrast to the experimental evaluation above, we only use the first 50.000 samples (training set) of the dataset MNIST during the cluster tree optimization. After optimization, we evaluate the clustering performance of the cluster tree on the previously unseen, remaining 20.000 data points (test set).

In this experiment, we get for the test set a dendrogram purity of 0.73 ± 0.08 and a leaf purity of 0.85 ± 0.06, which is a slight drop compared to the values in Table 2. Nevertheless, the result is robust enough to allow for limited label-predictions of previously unseen data points directly by the cluster tree. However, in most cases, we would train a classifier based on the found cluster structures. The same applies, for the embedding itself, where we can utilize, for instance, the supervised autoencoder [16] loss to enhance the found embedding.

3.8 Experiments Summary

In summary, we think that the shown experiments on four real-world datasets show clearly the utility and effectiveness of the DeepECT cluster tree. Finding these kind of structures and

selecting the level of detail to be analyzed make DeepECT a valuable method for data scientists.

4 Related Work

Our proposed method DeepECT touches two aspects of the vast literature on clustering: hierarchical methods and embedded methods that utilize autoencoders.

Hierarchical clustering algorithms are a well-established area within data mining and an overview of many well-known methods can be found in [17]. Agglomerative clustering algorithms are a family of hierarchical bottom-up strategies with single-linkage and complete-linkage as the most prominent members. In each step, the two closest clusters are merged based on some defined cluster distance measure. In single-linkage, the cluster distance is defined as the distance between the two closest points of two clusters. In complete-linkage, the cluster distance is defined as the maximum distance between the points of two clusters [18, p. 895ff.]. BCM [7] is a recent Bayesian approach to bottom-up agglomerative clustering, that uses hypothesis testing to decide if clusters should be merged. The Bisecting-K-means algorithm [19] is a top-down, divisive method that applies k-means with $k = 2$ recursively on the previous clustering result. PERCH [8] is a non-greedy, incremental algorithm that aims to build a cluster tree that scales to both massive numbers of data points and clusters. GHC [20] assigns data points softly to each sub-tree and optimizes a differentiable cost function. All these methods are well tested in classical settings; however, in combination with a nonlinear embedding, they can only be used after the embedding is learned. This is the advantage of DeepECT, which interacts with the embedding in such a way that both mutually improve each other.

Deeply embedded clustering methods have recently gained much attention. So far, the primary focus has merely been on flat-clustering objectives that need the expected number of clusters during optimization. Within this field, many algorithms utilize k-means-like centroids as cluster representatives. DEC [3] can be seen as the first algorithm that successfully combines autoencoders with a clustering objective. Its clustering objective utilizes a soft-assignment to the centroids using a Student-t kernel. These cluster assignments are then hardened using the Kullback–Leibler divergence and an auxiliary target distribution. Thereby, it only uses the encoder part of a pre-trained autoencoder. DBC [10] replaces the feedforward autoencoder with a convolutional autoencoder. IDEC [5] extends DEC by combining its clustering objective with the autoencoder's reconstruction loss. SDEC [13] combines the DEC objective with a semi-supervised setting, where the user can provide pairwise constraints of objects that should or should not belong to the same cluster. Both DKM [14] and DEPICT [21] soft-assign data points to clusters centroids based on Gaussian kernels. Thereby, DEPICT hardens these assignments similar to DEC using the Kullback–Leibler divergence between these assignments and an auxiliary target distribution. The authors of DKM also explores the idea of using autoencoders without pre-training. DCN [11] combines autoencoder with the training scheme of k-means by alternating between updating the autoencoder parameters, then the data point assignments, and finally the cluster centers. The clustering objective of ENRC [22] aims to find multiple, non-redundant, k-means-like partitions in an autoencoder's embedded space.

However, not all methods utilize centroids. The following algorithms also aim to produce a flat clustering partition, yet they follow other ideas than centroids to provide a flat partition of a dataset. DSC-Nets [23] introduces a so-called self-expressive layer, a fully connected layer without bias and nonlinear activation, which is put between the encoder and decoder functions. This layer aims to encode each data point as a linear combination of the other samples in the same subspace. In deep subspace clustering [24, 25], one aims to find clusters together with a corresponding subspace of the embedded space, which exhibits the cluster structure. JULE [26] is a clustering algorithm for images that interprets the optimization of the convolutional autoencoder as a recurrent process and utilizes an affinity measure also used in agglomerative clustering to generate clusters in a bottom-up process. However, the final result is nevertheless a flat clustering. VaDE [27] is a generative, variational Bayesian approach that utilizes a Gaussian mixture model as the latent variable of a variational autoencoder. The ClusterGAN [28] harnesses a categorical distribution in the generative part of the generative adversarial network architecture. Two recent surveys [29, 30] provide a broader overview of other embedded clustering methods. All of the above-described methods utilize a pre-trained autoencoder for the initial embedding. Further, they need the expected number of clusters for optimization. This also means that only those structures captured by the clusters are improved. Moreover, other structural information is actively destroyed through their loss functions. As a consequence, the final embedded space will resemble the chosen number of clusters, regardless of the actual structure within the dataset and validation methods such as the Silhouette coefficient will fail. Figure 2 shows that even the inclusion of a reconstruction loss will not entirely prevent this. DeepECT circumvents this behavior through its novel projected compression loss. Our experiments show that with a sufficiently large tree, all relevant

structures will be successfully captured. The level of detail can then be chosen during analysis.

5 Conclusion

In this work, we proposed the Deep Embedded Cluster Tree (DeepECT). It simultaneously improves the nonlinear features embedding and iteratively grows a binary tree of clusters capturing (hierarchical) structures. Our experiments show that DeepECT outperforms the hierarchical baseline methods by a wide margin. We believe that is DeepECT is an attractive tool when dealing with complex data domains that profit from nonlinear transformations—such as images—and one either does not know the exact number of clusters or expects a hierarchical data structure. The optional data augmentation extension can improve performance by avoiding known invariances of the dataset. Future efforts may be directed toward experiments, where DeepECT is combined with domain-specific embedding methods.

Author Contributions All authors contributed to the development of methods presented in this paper as well as paper writing.

Funding No funding was received.

Availability of Data and Materials The implementations of DeepECT and the baseline methods, the pre-trained autoencoders and scripts to reproduce the results shown in the Experiments section are available under https://dmm.dbs.ifi.lmu.de/downloads. The datasets are available at the following locations:

Dataset	Available at
Fashion-MNIST	https://github.com/zalandoresearch/fashion-mnist
MNIST	http://yann.lecun.com/exdb/mnist/
Reuters	http://jmlr.csail.mit.edu/papers/volume5/lewis04a/
USPS	https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#usps

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Caron M, Bojanowski P, Joulin A, Douze M (2018) Deep clustering for unsupervised learning of visual features. In: Ferrari V, Hebert M, Sminchisescu C, Weiss Y (eds) Computer Vision—ECCV—15th European conference, Munich, Germany, 8–14 September 2018, Proceedings, Part XIV, ser. Lecture notes in computer science, vol 11218. Springer, pp 139–156. https://doi.org/10.1007/978-3-030-01264-9_9
2. Perez L, Wang J (2017) The effectiveness of data augmentation in image classification using deep learning. CoRR, vol abs/1712.04621 . arXiv:1712.04621
3. Xie J, Girshick RB, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: Balcan M, Weinberger KQ (eds) Proceedings of the 33nd international conference on machine learning, ICML , New York City, NY, USA, 19–24 June 2016, ser. JMLR workshop and conference proceedings, vol 48. JMLR.org, 2016, pp 478–487. http://proceedings.mlr.press/v48/xieb16.html
4. van der Maaten L (2009) Learning a parametric embedding by preserving local structure. In: Dyk DAV, Welling M (eds) Proceedings of the twelfth international conference on artificial intelligence and statistics, AISTATS , Clearwater Beach, Florida, USA, 16–18 April 2009, ser. JMLR Proceedings, vol 5. JMLR.org, 2009, pp 384–391. http://proceedings.mlr.press/v5/maaten09a.html
5. Guo X, Gao L, Liu X, Yin J (2017) Improved deep embedded clustering with local structure preservation. In: Sierra C (ed) Proceedings of the 26th international joint conference on artificial intelligence, IJCAI , Melbourne, Australia, 19–25 August 2017, ijcai.org, 2017, pp 1753–1759. https://doi.org/10.24963/ijcai.2017/243
6. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, 7–9 May, Conference Track Proceedings, 2015. arXiv:1412.6980
7. Heller KA, Ghahramani Z (2005) Bayesian hierarchical clustering. In: Raedt LD, Wrobel S (eds) Machine learning, Proceedings of the 22nd international conference (ICML), Bonn, Germany, 7–11 August 2005, ser. ACM international conference proceeding series, vol 119. ACM, pp 297–304. https://doi.org/10.1145/1102351.1102389
8. Kobren A, Monath N, Krishnamurthy A, McCallum A (2017) A hierarchical algorithm for extreme clustering. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, Halifax, NS, Canada, 13–17 August 2017. ACM, pp 255–264. https://doi.org/10.1145/3097983.3098079
9. Peterson AD, Ghosh AP, Maitra R (2018) Merging k-means with hierarchical clustering for identifying general-shaped groups. Stat 7(1):e172
10. Li F, Qiao H, Zhang B (2018) Discriminatively boosted image clustering with fully convolutional auto-encoders. Pattern Recognit 83:161–173. https://doi.org/10.1016/j.patcog.2018.05.019
11. Yang B, Fu X, Sidiropoulos ND, Hong M (2017) Towards k-means-friendly spaces: simultaneous deep learning and clustering. In: Precup D, Teh YW (eds) Proceedings of the 34th international conference on machine learning, ICML, Sydney,

NSW, Australia, 6–11 August 2017, ser. Proceedings of machine learning research, vol 70. PMLR, pp 3861–3870. http://proceedings.mlr.press/v70/yang17b.html
12. Ghasedi Dizaji K, Herandi A, Deng C, Cai W, Huang H (2017) Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In: Proceedings of the IEEE international conference on computer vision, pp 5736–5745
13. Ren Y, Hu K, Dai X, Pan L, Hoi SC, Xu Z (2019) Semi-supervised deep embedded clustering. Neurocomputing 325:121–130
14. Fard MM, Thonet T, Gaussier É (2018) Deep k-means: jointly clustering with k-means and learning representations. CoRR, vol abs/1806.10069. arXiv:1806.10069
15. Lewis DD, Yang Y, Rose TG, Li F (2004) RCV1: a new benchmark collection for text categorization research. J Mach Learn Res 5:361–397
16. Le L, Patterson A, White M (2018) Supervised autoencoders: improving generalization performance with unsupervised regularizers. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in neural information processing systems, vol 31. Curran Associates Inc, Red Hook, pp 107–117
17. Murtagh F, Contreras P (2012) Algorithms for hierarchical clustering: an overview. Wiley Interdiscip Rev Data Min Knowl Discov 2(1):86–97. https://doi.org/10.1002/widm.53
18. Murphy KP (2012) Machine learning—a probabilistic perspective. Adaptive computation and machine learning series. MIT Press, Cambridge
19. Steinbach M , Karypis G, Kumar V et al (2000) A comparison of document clustering techniques. In: 6th ACM SIGKDD (Boston), World text mining conference, vol 400. ACM, pp 525–526
20. Monath N, ZaheerM, Silva D , McCallum A, Ahmed A (2019) Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. In: Teredesai A, Kumar V, Li Y, Rosales R, Terzi E, Karypis G (eds) Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining, KDD, Anchorage, AK, USA, 4–8 August 2019. ACM, pp 714–722. https://doi.org/10.1145/3292500.3330997
21. Dizaji KG, Herandi A, Deng C, Cai W, Huang H (2017) Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In: IEEE international conference on computer vision, ICCV , Venice, Italy, 22–29 October 2017. IEEE Computer Society, pp 5747–5756. https://doi.org/10.1109/ICCV.2017.612
22. Miklautz L, Mautz D, Altinigneli C, Böhm C, Plant C (2020) Deep embedded non-redundant clustering. In: To be published in proceedings of the conference on artificial intelligence. AAAI
23. Ji P, Zhang T, Li H, Salzmann M, Reid ID (2017) Deep subspace clustering networks. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R (eds) Advances in neural information processing systems 30: annual conference on neural information processing systems, 4–9 December 2017, Long Beach, CA, USA, pp 24–33. http://papers.nips.cc/paper/6608-deep-subspace-clustering-networks
24. Ji P, Zhang T, Li H, Salzmann M, Reid I (2017) Deep subspace clustering networks. In: Advances in neural information processing systems, pp 24–33
25. Zhang T, Ji P, Harandi M, Hartley R, Reid I (2018) Scalable deep k-subspace clustering. In: Asian conference on computer vision. Springer, pp 466–481
26. Yang J, Parikh D, Batra D(2016) Joint unsupervised learning of deep representations and image clusters. In: 2016 IEEE conference on computer vision and pattern recognition, CVPR , Las Vegas, NV, USA, 27–30 June 2016. IEEE Computer Society, pp 5147–5156. https://doi.org/10.1109/CVPR.2016.556
27. Jiang Z, Zheng Y, Tan H, Tang B, Zhou H (2017) Variational deep embedding: an unsupervised and generative approach to clustering. In: Proceedings of the 26th international joint conference on artificial intelligence, pp 1965–1972
28. Mukherjee S, Asnani H, Lin E, Kannan S (2019) Clustergan: latent space clustering in generative adversarial networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 4610–4617
29. Aljalbout E, Golkov V, Siddiqui Y, Cremers D (2018) Clustering with deep learning: taxonomy and new methods. CoRR, vol abs/1801.07648. arXiv:1801.07648
30. Min E, Guo X, Liu Q, Zhang G, Cui J, Long J (2018) A survey of clustering with deep learning: from the perspective of network architecture. IEEE Access 6:39 501–39 514. https://doi.org/10.1109/ACCESS.2018.2855437