Deep Embedded Clustering on Larger Datasets: A Study in Malware Domain

Martin Mocko[1,2,*], Daniela Chudá[2,3]

[1]Faculty of Information Technology, Brno University of Technology, Brno, Czechia
[2]Kempelen Institute of Intelligent Technologies, Bottova 7939/2A, 811 09, Bratislava, Slovakia
[3]Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava, Slovakia

Abstract

In recent years, deep clustering has earned significant attention, while it offers feature learning that is optimized for the clustering task. However, little focus was put on utilizing deep clustering in the malware domain. This puts new requirements on deep clustering models - since malware domain contains larger datasets and involves tasks where utilizing thousands of clusters is appropriate, like malware family identification. So far, deep clustering with larger amounts of clusters remains relatively unexplored. To this end, the main goal of our study is to investigate Deep Embedded Clustering (DEC) and Improved Deep Embedded Clustering (IDEC) in terms of clustering quality in the malware domain - on tabular data, in situations where the data is large and the amount of clusters is large as well. Our study is conducted on public datasets BODMAS and EMBER from the malware domain as well as a private dataset from an industry partner. The results of our work reveal new insights into the clustering settings that need to be met in order for the deep clustering methods to be useful for malware clustering. Most importantly, the results show that improvements with DEC and IDEC can be made in the low-number-of-clusters regime, while the high-number-of-clusters regime showed significant performance degradation after close inspection of the experiment results.

Keywords

deep clustering, neural networks, malware, security, deep embedded clustering.

1. Introduction

In 2025, the amount of newly created malware and Potentially Unwanted Applications (PUA) for the Windows platform reached almost 80 million samples [1]. While annual totals fluctuate, the daily velocity of threats remains overwhelming; current metrics show that the AV-TEST Institute logs upwards of 450,000 new malware and PUA samples every single day [2], forcing automated defensive systems to process hundreds of millions of unique variations annually. Malware experts, however, still need effective solutions to identify the biggest new threats that are yet undetected as the numbers are still massive. One of the ways to identify new malware is through clustering. By assigning new unseen programs into a cluster, we can make statements about their malignancy. This is mainly based on the "purity" of the cluster.

For clusters to be pure, they also need to be relatively specialized - only programs that are really similar should end up in the same clusters. This almost automatically creates higher demand on the amount of clusters that need to be created. The bare minimum for the amount of clusters (in the malware domain) nowadays therefore becomes at least a few thousand clusters, possibly more.

Deep clustering has throughout the years risen in popularity and achieved state of the art results, predominantly in the task of image clustering. However, the malware domain has so far remained largely overlooked when it comes to the utilization of deep clustering techniques. Deep Embedded Clustering (DEC) [3] is one of the most established methods in this area. One of the main reasons one may consider using DEC for clustering is that it is relatively easily modifiable. Since it is an end-to-end clustering framework built purely on neural network concepts, there are multiple parts of the method

ITAT'26: Information Technologies – Applications and Theory, September 25–29, 2026, Biele Karpaty, Slovakia
*Corresponding author.
martin.mocko@kinit.sk (M. Mocko); daniela.chuda@stuba.sk (D. Chudá)
0000-0001-8982-0141 (M. Mocko); 0000-0002-3873-9308 (D. Chudá)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-09

CEUR-WS.org/Vol-4323/paper44.pdf

(blocks) which are easily replaceable. Even more important is the fact that by utilizing DEC, the learned features get additionally optimized for the sole task of unsupervised clustering - in our case, malware clustering. Up until now, there have been some works published that utilize both deep clustering and data from the security domain (more details in Section 2.4). However, a research work properly studying the *suitability* of deep clustering techniques in the *malware domain* is still missing (covered in Section 2).

Previous studies of deep clustering (which were performed on datasets from other domains) were usually evaluated on datasets with a small number of *natural* clusters. To our best knowledge, the largest number of *natural* clusters a deep clustering study dealt with was 1,000 [4]. The studies typically use data that is also labeled, where the evaluation of the performance is made easier by assuming that the number of clusters should be equal to the number of classes. However, the data landscape is much more varied and there are reasons why it can be beneficial to use an even larger amount of clusters - for example to generate clusters that are much more pure. Furthermore, usually when performing the task of clustering, the clusters in the dataset are still unknown (or at least, there is an assumption that new clusters could have been formed). This is the main motivation for our study that involves larger amount of clusters and utilizing other metrics of evaluation of clustering quality.

In our study, we utilize the DEC method according to the 2016 paper by Xie et al. [3] as well as the Improved Deep Embedded Clustering (IDEC) method according to the 2017 paper by Guo et al. [5]. We define one main research question that deals with an investigation of how the usage of a large number of clusters affects the clustering quality of DEC and IDEC. For the purposes of this paper, when we talk about a large number of clusters we mean a ratio $num_of_samples/num_of_clusters$ of 10,000 and less. Based on the results, the usability of these deep clustering methods for malware clustering is assessed. We investigate the deep clustering methods in the regime of both low amount of clusters and high amount of clusters. Evaluation is done by utilizing external clustering measures.

The contribution of this work is the following:

- we investigate the behavior of deep clustering methods in terms of DEC and IDEC in the malware domain with respect to the number of clusters
- we evaluate the respective deep clustering methods on large public benchmark malware datasets (BODMAS, EMBER) and a private industry dataset (SECURITY)
- we include lessons learned from our experiments and provide recommendations to the research community on how to approach deep clustering methods in the malware domain

In Section 2, we introduce the concept of deep clustering and describe the DEC method together with its extensions - IDEC and SDEC, as well as some more recent deep clustering approaches. Section 3 describes the setup of our experiments involving the utilization of deep clustering methods in the malware domain (with focus on larger amounts of clusters). In Section 4, we evaluate our achieved results for the performed experiments and provide lessons learned. Section 5 concludes the work with a summary of our contributions.

2. Related Work

In this Section, we further examine multiple deep clustering methods. For our study and experimentation, we chose DEC and IDEC. This choice is explained in more detail in Section 3.1.

2.1. Deep Embedded Clustering (DEC)

Deep Embedded Clustering, created by Xie et al [3], is arguably the most well-known method out of all deep clustering methods. It is an end-to-end neural network framework solution to the clustering task. The method operates with an independent feature learner, which a clustering optimization process can later influence. Moreover, based on the authors' experiments, an essential aspect of this method is its potential to scale to larger datasets as well (e.g., the inclusion of the Reuters text dataset in their experiments). However, this potential was not investigated further than for 800,000 samples and 10 clusters, which are relatively low numbers in the face of current standards for large datasets.

A high-level view of the DEC method allows us to describe it in the following two steps: 1.) feature learner (pre-)training and 2.) clustering assignments learning. From an architectural viewpoint, DEC uses a Stacked Denoising Autoencoder (SDAE) for feature representation (pre-training). Then, only the encoder part of the Autoencoder is used for training and fine-tuning the DEC method itself.

In formal terms, a simple Autoencoder can be defined as a 2-layer neural network that tries to learn the mapping $\omega : X \rightarrow X$, X being the set of n data samples $\{x_i \in X\}_{i=1}^{n}$. It tries to learn how to reconstruct the input on its output. The encoder part of the SDAE can be denoted as $\phi : X \rightarrow Z$, Z is the set of points that have been transformed into the latent space (the "code"), $\{z_i = \phi(x_i) \in Z\}_{i=1}^{n}$. The decoder part can be defined as $\psi : Z \rightarrow X$.

When the SDAE is fully trained, a traditional k-Means clustering algorithm is trained on the set of latent inputs, Z, to create an initialization of cluster centroids for DEC, μ_j, $j = 1, ..., k$. These cluster centroids can be considered another layer following the feature layer of the encoder. However, it does not compute its output using the typical formula of $y = f(Wx)$. Instead, the weights can be considered the cluster centers, and the activation function can be the Student's t-Distribution used to compute the Q distribution. It defines the similarity between an embedded point z_i and a cluster center μ_j [6]: $q_{ij} = \frac{(1+\|z_i-\mu_j\|^2)^{-1}}{\sum_{j'}(1+\|z_i-\mu_{j'}\|^2)^{-1}}$, where $\|.\|$ denotes the L2-norm and μ_j denotes the center of the j-th cluster. $q_{ij} = [q_{i1}, q_{i2}, ..., q_{ik}]$ denotes a soft assignment of sample i to the j-th cluster. Once the soft assignments are computed, DEC iteratively optimizes them by computing a KL-divergence between the soft assignments Q and an auxiliary target distribution P, which is computed as [6]: $p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}}$. Figure 1 illustrates the DEC method and its components.

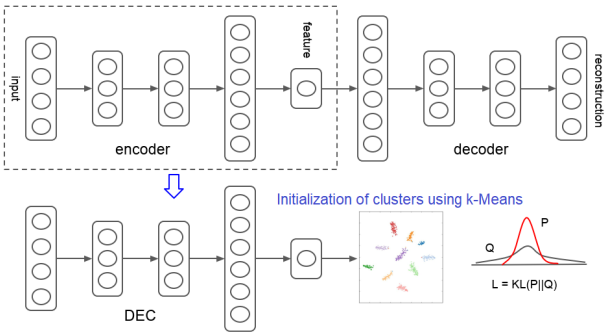


Figure 1: The architecture of Deep Embedded Clustering as described in Xie et al [3]. A typical Stacked Denoising Autoencoder architecture is depicted in the top half of the figure. It is trained to reconstruct its inputs. For training DEC, only the encoder portion of the SDAE is used. We extend the image with a note indicating the initialization of clusters using k-Means, which happens before training DEC.

2.2. Extensions of DEC

Multiple extensions of DEC have been proposed over the last few years. IDEC (Improved Deep Embedded Clustering) [5] identifies the sole optimization of clustering loss for DEC training as a potential source of bad corruption of the learned features, which could result in an under-performance of the overall clustering performance. IDEC tries to improve the learned clustering assignments of DEC by optimizing the reconstruction loss and the clustering loss jointly during training: $L = L_r + \gamma L_c$. The gamma term specifies how much the clustering loss should impact the individual updates. In order to achieve the best results, the authors claim that a gamma of less than 1 should be used. While conducting their experiments, a gamma of 0.1 is the best value for achieving the most accuracy, with a learning rate of 0.1. They report that gamma and learning rate are coupling; therefore, when one is changed, we should adjust the other appropriately. The results seem to point to an improvement in unsupervised clustering accuracy on the tested datasets of about 2% compared to the original DEC method.

Semi-supervised Deep Embedded Clustering (SDEC) [6], proposed in 2019, identifies a way that the

learned features can be improved further. It does so by incorporating additional prior information about the data that researchers, in many cases, possess. Often, one can identify a subset of samples that *should be* kept close together (ideally in a single cluster), also called "must-link constraints". One can also often identify a subset of samples where the samples should definitely *not* belong in a single cluster (ideally, they will be kept far apart from each other), also called "cannot-link constraints". These are often used in techniques implementing semi-supervision.

The information about constraints can be represented by constructing an $n * n$ sparse matrix, n being the number of samples. The model can receive this information during training by incorporating an additional term in the overall loss function. Recall that traditional DEC uses KL divergence on the P and Q as its loss function. The proposed changes in SDEC add an additional term in the following way: $L = KL(P||Q) + \lambda \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} ||z_i - z_k||^2$. The experimental evaluation seems to suggest an improvement in accuracy of under 2%, a little less than IDEC. However, on one sole dataset - 20NG, the accuracy seemed to jump by more than 20%, which, from our perspective, might indicate some underlying problem with the dataset or its processing. The authors also conducted a sensitivity analysis on the number of constraints required to make a noticeable impact on the result. It seems that for datasets where the self-supervision does help, the number of required constraints (pairs) is below n.

2.3. Recent Deep Clustering Works

Since the inception of deep clustering, a plethora of methods have been proposed[1]. However, a lot of them were proposed for usage on image datasets. They often utilize various manipulations of images - operations, which are not transferable to data from other domains. In the case of executable programs, the data is often tabular. On such data, utilizing image manipulations would not make any sense. Nevertheless, we provide examples of some of the more recent deep clustering works below.

DEC-DA [7] is another extension of DEC that proposes incorporating data augmentations (DA) into the learning process. It does so by utilizing random shifts and random rotations of the original images while training the Autoencoder. Superior clustering performance is observed on all of the evaluated image datasets.

Another deep clustering image-based technique is DeepCluster [8]. It utilizes an AlexNet architecture, which has randomly initialized weights. K-Means clustering is trained on embeddings from this network, and the network is then trained in a supervised manner to predict the K-Means pseudo-labels. The approach works well for image data thanks to the incorporation of the convolutional architecture.

SPICE [9] is a framework that divides the clustering process into a) a feature model for measuring the instance-level similarity and b) a clustering head for identifying clustering-level discrepancy. The feature model is trained via a contrastive learning approach, so the model is a hybrid. The cluster-level training is conducted to utilize semantic inconsistencies between clusters. The method heavily utilizes image transformations - they even employ two types - weak and strong transformations.

RUC tries to incorporate robust learning into the deep clustering process to revise misaligned knowledge and alleviate the overconfidence problem in predictions [10]. Its flexible structure allows it to be used in conjunction with already-existing deep clustering models. Co-training and label smoothing is utilized as two prominent robust learning techniques. The model is implemented in conjunction with the SCAN model [11].

2.4. Deep Clustering in the Malware Field

To the best of our knowledge, a proper research publication studying the suitability of deep clustering in the malware domain has not been published. However, some works utilize both deep clustering and some type of security (sometimes malware) data. Moreover, the problem of malware clustering had a lack of publications on *full* public malware benchmark datasets for a long time. However, recently in 2025 a work was published [12] where standard clustering approaches were tested on Ember and Bodmas, thereby setting the baselines (and state-of-the-art results at the time) for malware clustering on

[1] https://github.com/zhoushengisnoob/DeepClustering

the aforementioned datasets. Even more recently, another work [13] exploring modern representation learning methods like self-supervised and tabular representation learning was published, improving the results on both Ember and Bodmas, thus establishing new state-of-the-art.

Zhang et al [14] proposed a method that utilizes Adversarial Autoencoders and deep clustering. They aimed to detect malware in network traffic; therefore, the dataset utilized was USTC-TFC2016. This dataset has around 120k training samples. Although not explicitly stated in the paper, we suppose that the number of clusters utilized for their method was probably in the very low range (i.e., between 2 and 20). The deep clustering method is also not precisely specified. However, based on the method description, our best guess is that the method utilized is IDEC, which we will also utilize in our experiments. The number of epochs on which the deep clustering model was trained was 10. Therefore, the authors could have had a reason to assume that running the method for more epochs does not have the potential to improve results any further. Our experiments will help shed light on the ambiguous areas of this work. However, our main point of criticism of this research work is that this work did not find out whether this approach is suitable for malware clustering (or PE file clustering) because malware has many characteristics that are not detectable via network traffic and is usually represented using a very different feature set.

Another research work connecting malware datasets with deep clustering is by Katebi et al [15]. They utilize Android datasets (Drebin - 5.5k samples, Contagio - 28k samples, and Genome - 1.2k samples) which cover mainly the following features: API calls, Permissions, and Intents. They create only two clusters - malware and benign. The study's main aim is not to make any statements about the suitability of deep clustering in the malware domain. Instead, the aim is to utilize deep clustering in order to be able to create adversarial samples effectively. Regarding the deep clustering method, they are using a modified version of DEC. Instead of employing the standard Student's t-Distribution for cluster-sample similarity, they employ the Pearson correlation coefficient.

3. (I)DEC for a Larger Amount of Clusters

As previously indicated, we identify a strong need in the malware domain to utilize at least hundreds of clusters, if not thousands. This need stems from the vast amount of malware families that exist in the wild. Therefore, for a deep clustering method to be the most useful for malware clustering, it should be able to handle high amounts of clusters and do it successfully without sacrificing clustering quality. We evaluate clustering quality mainly through the metrics of Homogeneity and V-measure. Therefore, our main research question is defined as follows:
Can DEC or IDEC be used in scenarios with large datasets where the number of clusters is also large while maintaining a reasonable clustering quality?

The research question was motivated by a lack of malware deep clustering studies in general, but also the lack of studies that would evaluate the scenario of big datasets, high dimensionality and the need for using a high number of clusters. As far as we are aware, most deep clustering studies are conducted on datasets with sizes up to 100,000 samples [5, 6, 16, 17], but usually not higher. The largest dataset in deep clustering studies we have come across is the ImageNet dataset [4]. Even ImageNet contains only around 1.4 million total samples and only $1,000$ classes.

In contrast, almost 80 million new malware samples created in 2025 alone [1] (over 210,000 samples daily average) make the need for scalable solutions obvious. Besides the massive number of samples, the natural number of clusters in the malware domain can be much larger than the perceived number of classes (usually three - Benign, Malware, and Potentially Unwanted Applications (PUA)). When new samples are created, they may target new, previously unseen vulnerabilities in the system or exploit previously non-existent attack vectors. Similar samples among these then form new malware families. It naturally follows that for any clustering method to be useful in the malware domain, it has to handle at least millions of samples at the minimum and also the vast and diverse space of executable programs.

3.1. Method Choice

Based on our analysis of research works in the deep clustering space, we selected two representative methods that can be fully utilized for tabular datasets without too many changes or adding additional data. First, we selected DEC as a base representative of a whole class of deep clustering methods. Since IDEC is similar to DEC, reportedly should be able to achieve better performance, and does not require any additional data or transformations, we also chose to utilize it for our deep clustering experiments. Even a recent study of deep clustering methods (on arguably small tabular datasets) showed that IDEC tends to perform better than DEC [18]. As for the implementation of DEC and IDEC, we utilized the Keras implementation by Guo et al. available on GitHub[2]. Adjustments were needed to be made in order to be able to run experiments on datasets that have their total number of samples in the millions.

We also considered using various more recently proposed deep clustering methods (some of which are covered in Section 2). However, most are proposed for image clustering and often utilize image transformations as a vital factor in the learning process. Such transformations cannot be 1-to-1 mapped into the malware domain. Moreover, creating our own set of (possibly) advanced manipulations of executable programs would require a non-trivial amount of work, possibly worthy of a standalone publication of its own. Therefore, SPICE, RUC, DEC-DA, and DeepCluster were considered but disqualified from the list of potential deep clustering candidate methods for our experiments.

3.2. Datasets and Preprocessing

We experimented on three datasets from the malware domain. The first dataset, SECURITY, is a private dataset provided by a leading cybersecurity company. We used 135 features from this dataset, all describing aspects of the Portable Executable (PE) file, either static or dynamic. These binary files were all compiled in C or C++ languages. The three class labels are Benign, Malware, and Potentially Unwanted Applications (PUA). It consists of approximately $9,600,000$ train samples and $34,000$ test samples. The dataset has a strict train-test split because the data for the test set were gathered after creating the train set, which should ensure more generalizability. Summarized information about the datasets can be found in Table 1.

EMBER [19] is a dataset of PE file features publicly available on GitHub . It consists of 800,000 train samples and 200,000 test samples. It contains three labels: Benign, Malware, and "No-label" - a label for samples with a purposely unidentified class. It exists with multiple versions of features. In our experiments, we use the 2^{nd} feature version from 2018 (2,381 features), which is also the latest version at the time of writing this paper.

BODMAS [20] is a dataset from the malware domain, reportedly created with the help of domain experts. It includes timestamps, a binary class label, and carefully curated malware family labels. It consists of 134,435 total samples. Its feature set is generated using the publicly available EMBER feature extraction scripts.

By including publicly available datasets, we can compare whether the clustering performance on private and public datasets is similar. We can then encourage the research community to use these datasets in the future. The sizes of these datasets, being at least an order of magnitude greater than the usual dataset sizes of up to $100,000$, help us investigate deep clustering methods in more realistic scenarios (at least in the malware domain).

The minimum preprocessing necessary to implement was scaling the features into a $[0; 1]$ range since we are dealing with an unsupervised learning problem. However, malware datasets often tend to have more outlier values than other datasets. Therefore, we dealt with them in two ways - employing a RobustScaler from scikit-learn and standardizing the features. We identified columns with the most anomalous values (4) and excluded them from our feature set. The final preprocessing was therefore done in four steps: 1. leave out columns with the most outlier values, 2. transform the data with RobustScaler, 3. standardize the features, and 4. scale the features to a $[0; 1]$ range.

[2] https://github.com/XifengGuo/IDEC

Table 1
Comparison of datasets used in our experiments. SECURITY, EMBER, and BODMAS are datasets from the malware domain - the latter two being also publicly available. MNIST is a dataset from the original DEC study.

Dataset	# samples	# classes	# features	# families
SECURITY	9,600,000	3	141	7.1k
EMBER	1,000,000	3	2,381	N/A
BODMAS	134,435	2	2,381	581
MNIST	70,000	10	784	10

The EMBER dataset suffered from the same problems as the SECURITY dataset. We could not avoid the gradient exploding problem without leaving out certain columns with a high number of outliers. We identified 146 columns that were the biggest culprits and removed them from the feature set, leaving us with 2235 features. Feature scaling and normalizing steps followed the same steps from 2. to 4. as for the SECURITY dataset. Since BODMAS uses the same feature set as EMBER, the same preprocessing was used as for EMBER.

3.3. Experiment Settings

We trained DEC and IDEC models on all datasets mentioned above with most of the hyperparameters fixed. The main hyperparameter we changed was the number of clusters. Our goal was to investigate the behavior of these deep clustering models in the low and high number of clusters regime, as well as the suitability of (I)DEC for malware clustering. Since the sizes of the datasets differ in the total number of samples, it is not reasonable to test them all with the same numbers of clusters. Therefore, different number-of-clusters settings were used for BODMAS because it is the smallest dataset among the utilized malware datasets. We include the "Clustering ratio" column in the results table. The clustering ratio will give the reader an understanding of the theorized average number of samples per cluster for various experiment settings.

Both DEC and IDEC rely on initializing their cluster centers via K-Means. Therefore, we decided to use only one initialization of K-Means instead of the default value of 20. This decision sped up the computation of the whole method since when dealing with scenarios with large numbers of clusters, most of the training time is spent on fitting the k-Means clustering to the data. We ran k-Means on a GPU to speed up the computations. The gamma hyperparameter for IDEC was set to 1 as the learning rate of Adam was left at the 0.001 default.

3.4. Evaluation metrics

Evaluating the clustering performance on malware datasets is a challenging task. Most deep clustering papers (that experiment on datasets from other domains) evaluate datasets where they consider the dataset to be already labeled in terms of clusters - e.g., the number of clusters equals the number of distinct labels. This is also why almost no publications include experiments with the number of clusters, k, higher than the assumed number of actual classes.

In a real-world scenario, especially in the malware domain, the true number of clusters is often unknown. It is usually much higher than the number of class labels (usually three - Malware, Benign, PUA). This is due to the malware families that exist in the malware domain. From previous domain experience, we know that we are looking for a minimum of a few thousand clusters. In reality, the number of clusters could be even in tens or hundreds of thousands, assuming that the goal of clustering files is to group only samples that share a similar behavior specific to a unique family.

Our main metrics for evaluating clustering performance in the malware domain were chosen as Homogeneity and V-Measure [21], computed on both train and test datasets. For the record, to evaluate test datasets, we do not re-train the clustering methods; instead, we take the already trained methods and predict which clusters they belong to from the train set. Homogeneity will give us an idea of the "purity" of the learned cluster assignments. We consider Homogeneity measured on the test set the most

important metric. Domain experts prefer clusters with high Homogeneity, as the clusters should not mix dissimilar samples together. Given that H represents the Shannon Entropy, we define Homogeneity h as:

$$h = \begin{cases} 1, & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)}, & \text{otherwise} \end{cases} \tag{1}$$

where $H(C)$ represents class entropy of the true class labels and $H(C|K)$ represents Conditional Entropy of the true class labels, C, given the cluster assignments, K.

However, Homogeneity can suffer from a similar problem that Purity and Entropy measures do, which is that they are likely to improve monotonically with the number of clusters in the result, up to a degenerate maximum where there are as many clusters as data points [21]. Therefore, with this drawback in mind, we also employ V-Measure as our second-best option for evaluating the quality of the cluster assignments. We use the computation formula for Homogeneity and V-Measure defined by Rosenberg et al. [21].

Deep clustering papers often employ Unsupervised Clustering Accuracy (UCA) as one of the main evaluation metrics [3, 5]. The three labels we generally have at our disposal for the malware datasets will not tell the whole picture. Therefore, we should rely on something other than UCA. It does not work well when there is a high imbalance between predicted clusters and "known clusters" (class labels). Worse yet, the computation of UCA requires large amounts of memory when dealing with large numbers of clusters (e.g., in the thousands or more).

Since both DEC and IDEC need to be initialized via another clustering method, we divide the evaluation into two parts. The first part covers the best-achieved results for the two methods. This includes the results achieved right after initialization. In the second part, we dive deeper into the performance of the models during their training and report training curves.

4. Evaluation

4.1. Best-Achieved Results

The following paragraphs summarize the information about the best-achieved results during our experiments. Since the methods themselves include initialization through (K-Means) clustering, the best reported results could in fact have been measured during model initialization (and often this was the case). The concise version of the results for our experiments can be found in Table 2. We trained the deep clustering models on more numbers of clusters than shown in the table; we show only selected numbers for brevity purposes. For all datasets included in the experiments, Homogeneity and V-Measure are reported on both the train and test sets of the respective datasets, except for DEC on BODMAS.

For SECURITY and EMBER, the maximum number of clusters we settled on experimenting with was $50,000$. For EMBER, this was done to avoid scenarios where DEC would necessarily have to create clusters with a very small average number of samples per cluster. In the case of SECURITY, the highest value for the number of clusters was decided based on computational requirements. To stress that the ratio of the number of samples to the number of clusters is important to keep in mind, we indicate this value in the "C.Ratio" column. It indicates the average theorized number of samples per cluster.

The results are best interpreted in terms of Homogeneity measured on test data. Table 2 clearly shows that both the DEC and IDEC methods achieved relatively good results in terms of Homogeneity (indicated by "H-test") on the SECURITY dataset. When we increased the number of clusters, we saw a substantial rise in Homogeneity every time up until $50,000$ clusters, peaking at 79.05% for DEC and 79.48% for IDEC. For $50,000$ clusters, we can observe a relatively significant difference in Homogeneity on the train set between DEC and IDEC - by more than 8%. The increase in Homogeneity when the number of clusters is also increased is expected to some degree - as we already alluded to in Section 3.4. Therefore, the V-Measure metric (which includes Completeness) is reported to counteract these effects.

As can be seen, the best results in V-Measure on the test data for the SECURITY dataset can also be observed when we utilize the largest number of clusters.

We also report metrics other than Homogeneity on the test set for this experiment. The most interesting observations are whether the other metrics are also the highest in cases where Homogeneity was the highest. For the SECURITY dataset, we observe Homogeneity measured on the train set as the highest for $50,000$ clusters for both DEC and IDEC. For V-Measure, the best achieved value on the test set agrees on $50,000$ clusters. However, the best result on the train set is for 10 clusters. The reason why is that Completeness reaches much greater values in the lower-number-of-clusters settings since we use only three classes (Benign, Malware, and PUA) as external (true) labels.

We observe a similar situation for EMBER. We can see an increase in Homogeneity every time we increase the number of clusters. However, the increase is not as rapid as for the SECURITY dataset. We can observe a much better Homogeneity score for higher numbers of clusters for IDEC (87.62% vs 56.75%). This could indicate the superiority of IDEC for some datasets. However, for the train data, Homogeneity is still not great, even for the best result on EMBER (43.76%). Therefore, we encourage other researchers to use EMBER or similar publicly available datasets for deep clustering experimentation. Regarding V-Measure, for IDEC, the best results were achieved for $50,000$ clusters; however, for DEC, it was for $1,000$ clusters.

For BODMAS, the DEC experiments did not track V-Measure, so these values are missing. However, we can see that the results regarding Homogeneity on both the train and test sets are relatively similar. DEC and IDEC achieve results that are very close to each other, showing no significant differences. An increase in Homogeneity is observed for higher numbers of clusters - the same as for SECURITY and BODMAS.

Overall, the behavior of DEC and IDEC seems similar, except for how the methods behaved on EMBER. The deep clustering methods achieved relatively high Homogeneity values when they utilized a larger amount of clusters, suggesting that in order to have pure clusters, a larger number of clusters is the way forward. We can see a dip in clustering quality for EMBER, suggesting that the dataset contains samples that are hard to cluster with the provided feature set. We can see that two public datasets, EMBER and BODMAS, which have the same feature set but different samples can perform wildly differently in the clustering task.

4.2. Contribution of Deep Clustering Methods

The reported results in Table 2, however, only show the *best* achieved results for the methods on the datasets. So far, we have not looked more closely at how the evaluation metrics change during the training of DEC and IDEC. It turns out that assessment of the contribution of the deep clustering methods to the end clustering result is crucial for proper evaluation. Recall that in order to be able to use the DEC and IDEC methods, a K-Means initialization step is required. Therefore, when we show performance plots during (I)DEC iterations, the starting (leftmost) point is simply the performance measured for K-Means trained on the Autoencoder embeddings (although it is, in fact, (I)DEC right after its initialization and before the start of training). Only the following points show the contribution of DEC and IDEC - i.e., whether they actually improved the end clustering result. The performance plots show an interesting aspect of the tested deep clustering methods that could otherwise go unnoticed.

Figures 2 and 3 show the performance plots for IDEC on BODMAS to illustrate the behavior of the utilized deep clustering methods. Very similar performance plots can be shown for EMBER and SECURITY; however, we show only IDEC performance on BODMAS for conciseness. Figure 2 shows the performance in the low-number-of-clusters regime, while Figure 3 shows the performance in the high-number-of-clusters regime.

Together, these figures show a different view of the deep clustering methods. In the low-number-of-clusters regime, the performance at the start of training (I)DEC is not always maximal, and in these situations, utilizing DEC and IDEC may help improve the results. The performance improved for BODMAS when we set k to equal 10, 20, and 25 clusters. Interestingly, for $k = 5$, the clustering performance did not improve. These results can be contrasted with the results in the original (I)DEC

Table 2
Results showing the best achieved results on utilized datasets for DEC and IDEC. Due to space constraints, we needed to use abbreviations for measured variables. "# Clust." stands for the number of clusters, "C.Ratio" for the clustering ratio (number of training samples divided by the number of clusters), "H-test" for Homogeneity measured on test data, "H-train" for Homogeneity measured on train data, "VM-train" for V-Measure on train data, and "VM-test." for V-Measure on test data.

Dataset	DC method	# Clust.	C.Ratio	H-test	H-train	VM-test	VM-train
SECURITY	DEC	10	960,000	15.17%	29.23%	6.37%	**13.89%**
		100	96,000	32.66%	39.30%	8.03%	10.86%
		1,000	9,600	56.54%	53.55%	10.33%	10.78%
		10,000	960	73.68%	63.56%	11.79%	10.63%
		50,000	192	**79.05%**	**65.25%**	**12.04%**	10.44%
	IDEC	10	960,000	15.92%	23.56%	6.50%	**11.01%**
		100	96,000	34.62%	36.44%	8.31%	9.80%
		1,000	9,600	55.27%	49.48%	9.95%	9.81%
		10,000	960	75.63%	67.80%	11.63%	10.25%
		50,000	192	**79.48%**	**73.46%**	**12.27%**	10.13%
EMBER	DEC	10	80,000	17.08%	7.92%	8.72%	5.26%
		1,000	800	46.70%	22.50%	**11.41%**	**7.60%**
		10,000	80	43.88%	21.41%	11.12%	7.44%
		50,000	16	**56.75%**	**26.44%**	11.30%	7.38%
	IDEC	10	80,000	19.56%	9.76%	10.23%	6.47%
		1,000	800	56.14%	26.88%	11.99%	8.02%
		10,000	80	75.85%	35.44%	13.18%	8.54%
		50,000	16	**87.62%**	**43.76%**	**13.50%**	**9.12%**
BODMAS	DEC	10	13500	44.59%	45.04%	-	-
		100	1350	72.37%	72.46%	-	-
		581	232	86.18%	85.57%	-	-
		1000	135	**90.04%**	**88.72%**	-	-
	IDEC	10	13500	37.20%	38.60%	17.46%	18.15%
		100	1350	70.12%	70.42%	**18.96%**	**19.04%**
		581	232	86.76%	85.85%	18.51%	18.26%
		1000	135	**89.40%**	**88.12%**	17.99%	17.63%

papers. The datasets used for evaluation contained a low number of real, natural clusters. Therefore, the number of clusters k (the parameter of k-Means) was also set to equal the real number of natural clusters. Experimentation with datasets containing much larger numbers of naturally ocurring clusters were not performed. This may be one of the contributing factors to explain why we observe different results in our examined research scenario.

However, in the high-number-of-clusters regime, both DEC and IDEC degrade the results in terms of Homogeneity significantly. We can see a decrease from values as high as 90% Homogeneity to around 50%. If we were to run the methods for even more iteration points, the clustering performance would likely degrade even further. By saving checkpoints of the method at respective iteration points, one could avoid this problem and choose the best-performing model. However, if the clustering performance is sure to degrade, it makes little sense to keep training these models.

To show the performance of the utilized deep clustering methods on a dataset where the number of real clusters is much lower, we also utilized the MNIST dataset. We trained IDEC for the low-number-of-clusters regime on MNIST. The results can be found in Figure 4. The deep clustering methods show the best improvements on MNIST in the low-number-of-clusters regime, even better than on BODMAS. It can be seen that both the Homogeneity and V-Measure improve with IDEC iterations, although the best results are arguably reached relatively soon in the first few iterations.

The main lessons learned from our performed experiments for DEC and IDEC on the respective datasets can be summarized with the following new observations:

1. DEC and IDEC can be run with a large number of clusters and can be utilized for malware

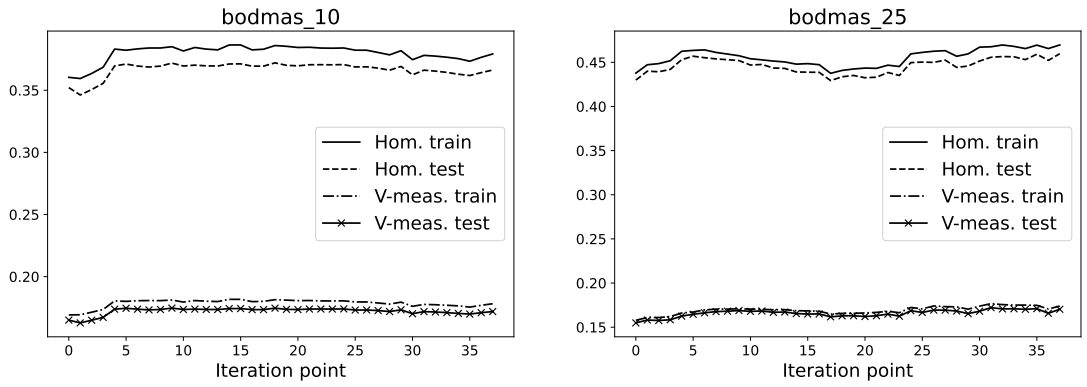


Figure 2: IDEC trained on BODMAS in the low-number-of-clusters regime

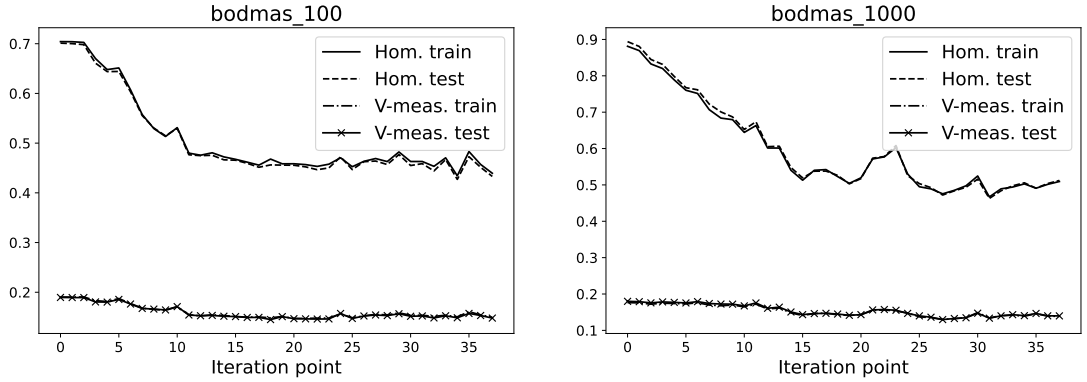


Figure 3: IDEC trained on BODMAS in the high-number-of-clusters regime

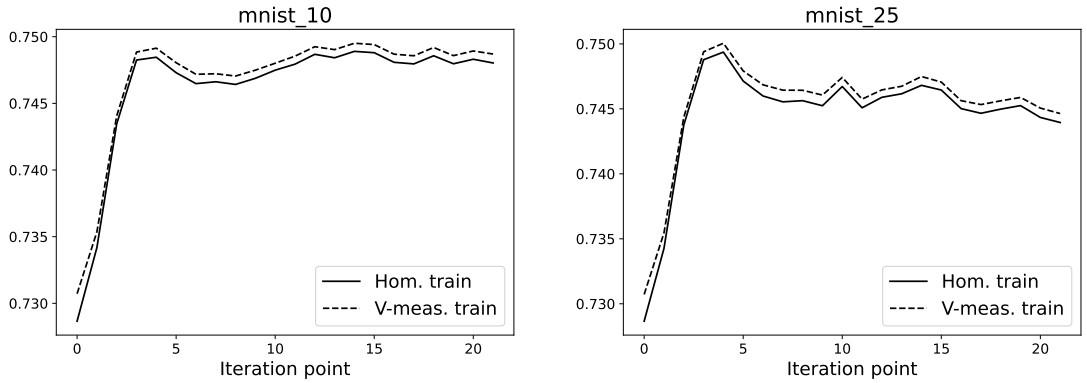


Figure 4: IDEC trained on MNIST in the low-number-of-clusters regime

clustering.

2. DEC and IDEC can be run with a much larger sample size than what previous research was conducted on.

3. In certain situations - particularly when we want to utilize a high number of clusters - the utilization of DEC and IDEC does not seem to bring additional benefit when compared with K-Means trained on Autoencoder embeddings (which one needs to train to initialize these methods).

4. Publicly available malware datasets, such as EMBER and BODMAS, can exhibit similar clustering behavior to real-world private malware datasets.

5. IDEC seems to perform better overall than DEC - however, the effect size is not too big (barring the results on EMBER).

Therefore, we recommend utilizing deep clustering models for scenarios when we know we will work with relatively few clusters. Also, the results achieved on MNIST indicate that a dataset containing only a lower number of real, natural clusters benefits DEC and IDEC even further. The size of the total samples in the dataset could also be a factor that influences the overall quality of the clustering results achieved.

5. Conclusion

This study investigated DEC and IDEC, two well-established deep clustering methods, in the malware domain, a setting that is both highly demanding in scale and largely unexplored from a deep clustering perspective. While deep clustering has gained considerable popularity in computer vision, its applicability to tabular malware data required rethinking both architecture and evaluation approach. Notably, we focused on the settings of a larger number of samples and scenarios where a larger number of clusters is required since the malware domain contains a large number of natural clusters. We selected two public datasets and one private dataset from the malware domain for our experiments. We achieved seemingly strong Homogeneity values of 79.48%, 87.62%, and 90.04% as the best results on SECURITY, EMBER, and BODMAS datasets.

The seemingly good results were inspected further via analysis using performance plots. The actual contribution of DEC and IDEC methods was assessed. In the high-number-of-clusters regime, the contribution of DEC and IDEC methods was almost none. The methods managed to degrade the clustering performance in terms of Homogeneity from 90% to as low as 50% in some instances. In the low-number-of-clusters regime, DEC and IDEC perform better and, in many cases, can improve the clustering performance. Our further analysis on MNIST indicates that the best performance can be achieved when the number of real natural clusters is relatively low.

To directly answer our stated research question: DEC and IDEC cannot reliably be used in the high-cluster regime while maintaining reasonable clustering quality. While both methods can technically run at scale, they fail to improve upon K-Means initialization and actively degrade Homogeneity in such settings. Given the operational scale of the malware domain — where hundreds of thousands of new samples appear daily — these findings suggest that practitioners should carefully weigh whether DEC or IDEC justify the computational overhead over simpler K-Means baselines. DEC and IDEC as deep clustering methods for malware clustering or clustering of other similar kinds of data are the most viable when we have: a) millions of samples, b) relatively high dimensionality (more than 100 features), and c) do not require a large number of clusters.

Since the number of deep clustering methods is relatively large, but many of the methods are based in the image domain, further work could involve utilizing deep clustering methods that require the usage of transformations of inputs. In the malware domain, attackers utilize transformations of executable programs to fool the detection methods that try to detect malicious programs. Such transformations could serve as an excellent basis to replace the image transformations that are right now utilized in the deep clustering methods.

Acknowledgments

Funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I05-03-V02-00064 (EMA) and by the Slovak Research and Development Agency under the contract No. APVV-23-0292 (DyMAX).

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and Gemini in order to: Grammar, spelling check, and for minor paragraph rephrasing.

References

[1] AV-Test GmbH, New malware and pua, 2025. URL: https://portal.av-atlas.org/?s=a2c32502a2ebbdf504ff1c57b93db98af72dac4c&c=eyJtYWx3YXJlRGV2ZWxvcG1lbnRUeXBlIjoiT25seU5ldyIsInN0YXRUeXBlIjpbIldJTl9NQUxXQVJFIiwiV0lOX1BVQSJdLCJzdGFja2VkIjp0cnVlLCJjaGFydFJlc29sdXRpb24iOjEwfQ%3D%3D, [Online; accessed 07-July-2026].

[2] J. Park, A. Ji, M. Park, M. S. Rahman, S. E. Oh, Malcl: Leveraging gan-based generative replay to combat catastrophic forgetting in malware classification, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 658–666.

[3] J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, in: International conference on machine learning, PMLR, 2016, pp. 478–487.

[4] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, A. Joulin, Unsupervised learning of visual features by contrasting cluster assignments, Advances in Neural Information Processing Systems 33 (2020) 9912–9924.

[5] X. Guo, L. Gao, X. Liu, J. Yin, Improved deep embedded clustering with local structure preservation., in: Ijcai, 2017, pp. 1753–1759.

[6] Y. Ren, K. Hu, X. Dai, L. Pan, S. C. Hoi, Z. Xu, Semi-supervised deep embedded clustering, Neurocomputing 325 (2019) 121–130.

[7] X. Guo, E. Zhu, X. Liu, J. Yin, Deep embedded clustering with data augmentation, in: Asian conference on machine learning, PMLR, 2018, pp. 550–565.

[8] M. Caron, P. Bojanowski, A. Joulin, M. Douze, Deep clustering for unsupervised learning of visual features, in: Proceedings of the European conference on computer vision (ECCV), 2018, pp. 132–149.

[9] C. Niu, H. Shan, G. Wang, Spice: Semantic pseudo-labeling for image clustering, IEEE Transactions on Image Processing 31 (2022) 7264–7278.

[10] S. Park, S. Han, S. Kim, D. Kim, S. Park, S. Hong, M. Cha, Improving unsupervised image clustering with robust learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 12278–12287.

[11] W. Van Gansbeke, S. Vandenhende, S. Georgoulis, M. Proesmans, L. Van Gool, Scan: Learning to classify images without labels, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X, Springer, 2020, pp. 268–285.

[12] M. Mocko, J. Ševcech, D. Chudá, Clustering malware at scale: A first full-benchmark study, in: M. Dalla Preda, S. Schrittwieser, V. Naessens, B. De Sutter (Eds.), Availability, Reliability and Security, Springer Nature Switzerland, Cham, 2025, pp. 231–251.

[13] M. Mocko, D. Chudá, Self-supervised representations for binary program clustering: From empirical study to retrieval-augmented learning, 2026. URL: https://arxiv.org/abs/2608.02348, [Online; accessed 10-June-2026].

[14] L. Zhang, J. Yin, J. Ning, Y. Wang, B. Adebisi, J. Yang, A novel unsupervised malware detection method based on adversarial auto-encoder and deep clustering, in: 2022 9th International Conference on Dependable Systems and Their Applications (DSA), IEEE, 2022, pp. 224–229.

[15] M. Katebi, A. Rezakhani, S. Joudaki, Adcas: adversarial deep clustering of android streams, Computers and Electrical Engineering 95 (2021) 107443.

[16] K. Do, T. Tran, S. Venkatesh, Clustering by maximizing mutual information across views, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 9928–9938.

[17] Z. Dang, C. Deng, X. Yang, K. Wei, H. Huang, Nearest neighbor matching for deep clustering, in:

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 13693–13702.

[18] S. Abrar, A. Sekmen, M. D. Samad, Effectiveness of deep image embedding clustering methods on tabular data, 2022. URL: https://arxiv.org/abs/2212.14111, [Online; accessed 10-May-2026].

[19] H. S. Anderson, P. Roth, EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models, ArXiv e-prints (2018). URL: http://arxiv.org/abs/1804.04637, [Online; accessed 21-April-2026].

[20] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, Bodmas: An open dataset for learning based temporal analysis of pe malware, in: 2021 IEEE Security and Privacy Workshops (SPW), IEEE, 2021, pp. 78–84.

[21] A. Rosenberg, J. Hirschberg, V-measure: A conditional entropy-based external cluster evaluation measure, in: Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), 2007, pp. 410–420.