

Benchmarking Deep Clustering Algorithms With ClustPy

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

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

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

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

Abstract—Deep clustering algorithms have gained popularity as they are able to cluster complex large-scale data, like images. Yet these powerful algorithms require many decisions w.r.t. architecture, learning rate and other hyperparameters, making it difficult to compare different methods. A comprehensive empirical evaluation of novel clustering methods, however, plays an important role in both scientific and practical applications, as it reveals their individual strengths and weaknesses. Therefore, we introduce ClustPy, a unified framework for benchmarking deep clustering algorithms, and perform a comparison of several fundamental deep clustering methods and some recently introduced ones. We compare these methods on multiple well known image data sets using different evaluation metrics, perform a sensitivity analysis w.r.t. important hyperparameters and perform ablation studies, e.g., for different autoencoder architectures and image augmentation. To our knowledge this is the first in depth benchmarking of deep clustering algorithms in a unified setting.

Index Terms—Deep Clustering, Data Mining, Unsupervised Learning, Representation Learning, Benchmarking

I. INTRODUCTION

In recent years, a growing number of methods have been published that combine clustering with deep learning. The main advantage is that neural networks allow the processing of very large and complex data sets. These deep clustering (DC) methods can be distinguished both according to the architecture used and according to the type of objective [1]. Many procedures use an ordinary feedforward autoencoder [2] (e.g., DEC [3], IDEC [4], DCN [5], ACe/DeC [6], DipEncoder [7], DKM [8], DeepECT [9], DipDECK [10], DECCS [11]), while others use a convolutional autoencoder [12] (e.g., ENRC [13], DCEC [14], DeepCluster [15], JULE [16], DEPICT [17], DEKM [18]). The type of clustering objective ranges from regular flat clustering techniques, such as DEC [3] or DCN [5], over hierarchical approaches, such as DeepECT [9], to non-redundant clustering, such as ENRC [13].

Due to these structural differences, DC procedures are fundamentally difficult to compare. In addition, deep learning methods generally use a multitude of hyperparameters, such as the number of neurons, the type of optimizer, or the learning rate. A benchmark based on a comparable basis does help to rank the strengths and weaknesses of different methods. We

thus propose a framework that can be used to easily create a fair benchmark of various DC algorithms. For this purpose, we use the open source Python package ClustPy¹, which is based on PyTorch [19] and contains a variety of implementations for DC applications. Based on diverse experiments, we discuss several insights and future directions for DC research.

Our main contributions are:

- First comparison of several fundamental deep clustering algorithms on multiple data sets in a unified framework.
- A study of the impact of important hyperparameters in deep clustering.
- Ablation study on the impact of autoencoder architectures and image augmentation on performance metrics.

II. BACKGROUND AND RELATED WORK

A. Methods

In our analysis, we consider five DC methods with different objectives, that all use a very similar autoencoder (AE) architecture. Furthermore, the optimization of the clustering result runs independent of the specific architecture, which means that it can be exchanged without much effort. Since these algorithms all require an initial clustering result, usually obtained via KMeans [20], we additionally consider the combination of a pretrained AE with KMeans, denoted as AE+KMeans. The AE+KMeans baseline serves as a starting point to measure the improvement of more sophisticated deep clustering methods.

A DC algorithm we consider is DEC [3]. It is often considered the first representative of AE-based DC methods. Here, the objective function compares the data distribution within the embedding with an auxiliary target distribution. The data distribution is quantified by a kernel on the basis of a Student's t-distribution and the deviation to the target distribution is measured using the Kullback-Leibler (KL) divergence. This idea was taken up by IDEC [4]. While DEC uses the reconstruction loss only for pretraining the AE, IDEC also uses it during the optimization of the clustering result. DCN [5] follows a different strategy by not updating the clustering and the embedding simultaneously but consecutively. This allows to optimize an objective function that utilizes hard

*Authors contributed equally.

¹<https://github.com/collinleiber/ClustPy>

TABLE I

REPRODUCIBILITY: ORIGINAL VS REPLICATED SCORES ON MNIST. THE CITATION BEHIND A VALUE INDICATES THE PAPER IT WAS TAKEN FROM. † MEANS THAT A VALUE WAS NOT REPORTED IN ANY PUBLICATION. RESULTS THAT ARE ONLY $\pm 2\%$ POINTS OFF ARE PRINTED IN BOLD.

Algorithm	NMI	Original		Replication		
		ARI	ACC	NMI	ARI	ACC
AE + KMeans	74.7 [4]	67.X [5]	81.8 [3]	70.8	63.4	74.9
DEC [3]	83.7 [4]	75.X [5]	84.3 [3]	82.0	74.4	80.2
IDEC [4]	86.7 [4]	78.6 [7]	88.1 [4]	85.4	78.1	82.5
DCN [5]	81.X [5]	75.X [5]	83.X [5]	81.7	76.9	83.6
ACe/DeC [6]	89.X [6]	†	†	84.4	80.4	86.4
DipEncoder [7]	85.3 [7]	78.7 [7]	†	83.5	76.7	81.7

labels instead of soft ones, which is much closer to the optimization of the KMeans algorithm. The DipEncoder [7] optimizes modalities in the embedding by utilizing the Dip-test of unimodality [21] to obtain multimodal distributions for each pair of clusters. The advantage is, that no specific distribution has to be selected. In contrast to other procedures, ACe/DeC [6] does not use all features of the embedding for clustering but autonomously chooses those features most relevant for clustering. The clustering itself is then performed similar to DCN [5].

B. Architectures

We limit our benchmark to two architectures. First, we look at ordinary feedforward AEs [2]. The particular characteristic of this architecture is that an intentional bottleneck is introduced. The encoder maps the d input features to an embedding of size m , where $m < d$. Subsequently, the decoder maps the m embedded features as precisely as possible to the original d features. Different loss functions can be used for this purpose. A common choice is the mean squared error (MSE). In many cases, the architecture of the decoder is the same as the inverted architecture of the encoder.

Some works, such as DEC [3], use a special training method in which individual layers are trained one after the other together with a denoising objective. This is also referred to as stacked autoencoders (SAEs) [22]. This process leads to a significant increase in the required training time. Due to the small architectural differences to regular feedforward AEs, stacked AEs are not considered in our benchmark.

The second architecture we look at in more detail is convolutional AEs [23]. These are particularly suitable for image data, as they are able to detect local relationships using convolutional layers. We use a ResNet18 [24] encoder and decoder for our benchmark.

C. Benchmarking Methodologies

Benchmarking compares different procedures in a uniform setting. While this has already been done for many areas of unsupervised learning, e.g. outlier detection [25], [26] or time-series clustering [27], to the best of our knowledge, there is nothing equivalent for DC. This is, however, very relevant, to make informed choices on architecture and hyperparameters. The only sophisticated evaluation of different DC architectures

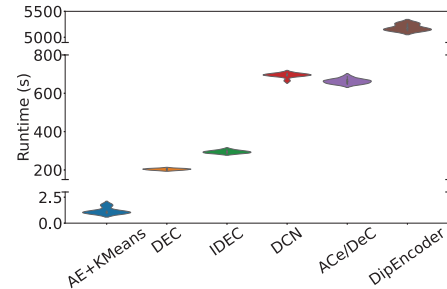


Fig. 1. Runtime of deep clustering algorithms on the MNIST data set.

is presented in [28] and [29], where the performance of a novel approach based on consensus representations and spectral clustering is analyzed. Other comparisons of different DC methods are papers like [1], which gather values from the original publications. A problem with this procedure is that the experimental environments are not identical, which makes it difficult to compare the results. In contrast to this, we compare several DC methods in a unified experimental environment.

III. BENCHMARKING

Since we want to get a broad overview of the performances of the chosen methods, we consider a spectrum of different data sets, evaluation metrics, architectures, and hyperparameters. All experiments can be repeated using the ClustPy package and are available at <https://github.com/collinleiber/BenchmarkingDeepClustering>.

A. Data Sets

We use eight publicly available image data sets that vary in complexity and size. The data sets are Optdigits [30], USPS [31], MNIST [32], Fashion-MNIST [33], Kuzushiji-MNIST [34], CIFAR-10 [35], ImageNet10 [36] and ImageNetDog [37]. Optdigits, USPS, MNIST, Fashion-MNIST (FMNIST) and Kuzushiji-MNIST (KMNIST) contain grayscale images with 10 clusters each. The color-image data sets CIFAR-10 and ImageNet10 illustrate ten different objects like cars, animals and ships. For ImageNetDog we use a subset of 15 different breeds of dogs. Furthermore, the sizes of the images contained in ImageNet10 and ImageNetDog were adjusted to 224x224 pixels. The exact specifications of the data sets can be found in the tables of the corresponding experiments.

B. Performance Metrics

We evaluate the clustering results based on several unsupervised evaluation metrics. These are the normalized mutual information (NMI) [38], unsupervised clustering accuracy (ACC) [39] and adjusted rand index (ARI) [40]. These compare the predicted labels with the ground truth in order to make a statement about the quality of the clustering result. All have a maximum value of 1, which is equal to a perfect match between prediction and ground truth. We report all metrics in % due to space restrictions.

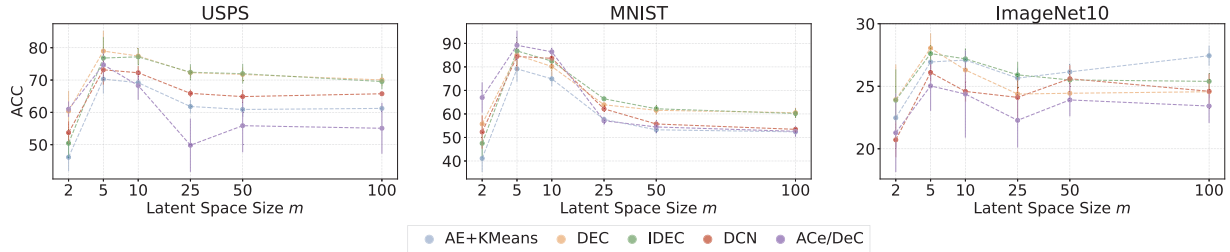


Fig. 2. **Varying Latent Space Size:** Accuracy (ACC) scores (mean and standard deviation) for varying embedding size of the AE with d -500-500-2000- m .

C. Setup

Unless otherwise mentioned, we use the following default configuration. We pretrain the AEs for 100 epochs and execute the clustering procedure for another 150 epochs. Here we use a batch size of 256 and choose the MSE as the reconstruction loss of the AE. The architecture of the AE follows [3] which is equal to the layers d -500-500-2000- m , where d is the number input features and m the number of features in the embedding, which is set to 10. As optimizer we choose ADAM [41] with a constant learning rate of 0.001 for the pretraining and 0.0001 for training the DC algorithm. An exception is ACe/DeC which uses a learning rate of 0.0005 for clustering [6]. All methods use the same pretrained AEs so that the starting conditions are as identical as possible. The initial clustering labels are obtained by executing KMeans in the embedding of the pretrained AE. Each experiment is repeated 10 times. Further, each clustering algorithm gets the correct number of clusters as input parameter and the data sets will be preprocessed by a channel-wise normalization (zero mean and standard deviation of one for each color channel).

IV. EXPERIMENTS

We conduct multiple experiments to evaluate different aspects of the DC methods. Each experiment is self-contained, including the specific setup and a discussion of the results.

A. Reproducibility

First, we check the correctness of the implementations from ClustPy. Therefore, we compare the clustering results reported in the original publications against the results calculated by us. **Setup:** We use MNIST as it is the only data set used in all publications. Note that while all publications used a d -500-500-2000- m AE architecture, we did not make use of learning rate schedulers or the SAE extensions, as explained in Sec. II-B. All DC algorithms are trained for 150 epochs, which might differ from the number of epochs used in the original works. Further, if a metric is not used in a paper, we report the value from the first paper that reports it.

Results: In Tab. I we see that despite our simplified setting we are able to reproduce 9 out of 12 DC results with a margin of error of 2%. Note, that AE+KMeans performs significantly worse than stated in [3], [4] and [5], which might be due to the mentioned differences in AE pretraining. The lower ACC results for DEC and IDEC are likely a consequence of this.

B. Runtime Comparison

We compare the runtime of the different DC algorithms over ten runs, excluding the shared pretraining time.

Setup: All experiments were conducted on a Server with two Intel 6326 CPUs (2×16 cores with 2.9 GHz), 512 GB RAM and one NVIDIA A100 (80GB) GPU.

Results: The violin plots in Fig. 1 show that the AE+KMeans execution is very fast at 1-2 seconds. DEC and IDEC as well as DCN and ACe/DeC have a comparable runtime of approximately 300 seconds and 700 seconds respectively, due to their similar optimization schemes. The DipEncoder takes over 5,000 seconds to process MNIST. This is due to the Dip-test that is executed on the CPU and therefore many copy processes take place between the CPU and GPU. As the runtime of the DipEncoder is longer than the combined runtime of all other methods, we excluded it from compute heavy experiments, like detailed hyperparameter investigation.

C. Feed Forward Autoencoder - Layer Sizes

A key performance factor of DC methods is the chosen architecture, with larger neural networks learning more complex structures. We investigate how the performance changes when the default architecture as proposed in [3] is modified.

Setup: We change the AE layers from d -500-500-2000-10 to d -512-256-128-10, reducing the number of parameters by almost 87% from $500d + 1,270,000$ to $512d + 165,120$.

Results: Many of the results for both architectures in Tab. II and Tab. III are very similar. This suggests that even the smaller network is capable of analyzing rather simple data sets like Optdigits or USPS. Only for ACe/DeC the results slightly decreased. For MNIST, the results have even partially improved. The poor performance of AE+KMeans in Tab. II, might be due to an unfortunate pretraining. Longer pretraining may therefore be beneficial for larger networks.

D. Varying Latent Space Size

An important parameter for all autoencoder-based methods is the number of features within the embedding m . A value of $m = 10$ is common for comparing DC methods. We examine how strongly this choice influences the final clustering result.

Setup: For this experiment we use the standard setting as described in Sec. III-C. However, different values for m are utilized. We are investigating $m \in [2, 5, 10, 25, 50, 100]$.

TABLE II

RESULTS LARGE FEEDFORWARD AUTOENCODER: CLUSTERING RESULTS OF VARIOUS DEEP CLUSTERING APPROACHES ON DIFFERENT DATA SETS (N = NUMBER OF SAMPLES, d = NUMBER OF FEATURES, k = NUMBER OF CLUSTERS) USING A FEEDFORWARD AUTOENCODER WITH d -500-500-2000-10 NEURONS. EACH EXPERIMENT IS REPEATED TEN TIMES, AND THE AVERAGE RESULT \pm STANDARD DEVIATION IS STATED IN %. THE BEST AVERAGE RESULT PER DATA SET AND METRIC IS SHOWN IN **BOLD** AND THE RUNNER-UP IS UNDERLINED.

Dataset	Metric	AE+KMeans	DEC	IDEC	DCN	ACe/DeC	DipEncoder
Optdigits [30] ($N = 5620, d = 64, k = 10$)	NMI	80.0 \pm 2.3	87.9 \pm 2.4	<u>87.6 \pm 2.6</u>	83.3 \pm 2.5	83.4 \pm 2.9	87.5 \pm 3.0
	ARI	75.9 \pm 4.8	83.6 \pm 5.1	<u>83.6 \pm 5.4</u>	79.1 \pm 5.3	78.3 \pm 6.3	83.6 \pm 5.7
	ACC	85.6 \pm 4.5	<u>89.3 \pm 4.7</u>	89.4 \pm 4.8	87.2 \pm 4.7	87.5 \pm 5.1	89.2 \pm 5.1
USPS [31] ($N = 9298, d = 256, k = 10$)	NMI	67.2 \pm 1.0	80.0 \pm 1.1	<u>81.1 \pm 1.4</u>	72.8 \pm 1.2	71.7 \pm 2.5	80.8 \pm 1.6
	ARI	56.9 \pm 1.7	<u>72.1 \pm 1.9</u>	73.0 \pm 2.2	63.4 \pm 1.9	61.0 \pm 3.9	73.0 \pm 2.5
	ACC	69.2 \pm 2.0	77.4 \pm 2.5	<u>77.2 \pm 2.7</u>	72.3 \pm 2.1	68.3 \pm 4.4	77.4 \pm 2.6
MNIST [32] ($N = 70000, d = 784, k = 10$)	NMI	70.8 \pm 2.1	82.0 \pm 2.4	85.4 \pm 2.4	81.7 \pm 1.3	84.4 \pm 0.8	83.5 \pm 2.7
	ARI	63.4 \pm 3.6	74.4 \pm 4.9	<u>78.1 \pm 4.5</u>	76.9 \pm 3.0	80.4 \pm 1.0	76.7 \pm 5.3
	ACC	74.9 \pm 3.2	80.2 \pm 4.1	<u>82.5 \pm 3.5</u>	83.6 \pm 3.9	86.4 \pm 1.8	81.7 \pm 4.5
FMNIST [33] ($N = 70000, d = 784, k = 10$)	NMI	56.2 \pm 1.8	59.1 \pm 2.7	62.8 \pm 2.1	58.9 \pm 2.2	60.3 \pm 1.5	58.2 \pm 2.3
	ARI	41.0 \pm 3.1	41.7 \pm 3.7	45.5 \pm 3.6	43.0 \pm 3.3	43.3 \pm 2.6	42.4 \pm 3.4
	ACC	52.5 \pm 4.7	53.6 \pm 4.1	56.0 \pm 5.3	53.9 \pm 5.2	54.7 \pm 3.2	55.3 \pm 3.9
KMNIST [34] ($N = 70000, d = 784, k = 10$)	NMI	47.7 \pm 1.9	54.2 \pm 1.9	54.3 \pm 2.3	51.4 \pm 1.7	53.8 \pm 1.3	53.7 \pm 2.6
	ARI	34.4 \pm 2.6	<u>40.1 \pm 2.8</u>	40.0 \pm 3.0	37.4 \pm 2.8	41.3 \pm 1.4	39.7 \pm 2.8
	ACC	55.5 \pm 2.1	<u>59.0 \pm 2.5</u>	<u>59.5 \pm 2.0</u>	57.6 \pm 2.1	62.0 \pm 2.7	57.1 \pm 3.7
CIFAR10 [35] ($N = 60000, d = 3072, k = 10$)	NMI	10.4 \pm 0.7	10.7 \pm 0.5	10.3 \pm 0.7	11.4 \pm 0.6	9.7 \pm 1.1	10.3 \pm 1.4
	ARI	5.5 \pm 0.4	6.1 \pm 0.4	5.4 \pm 0.2	6.1 \pm 0.4	5.2 \pm 0.7	5.5 \pm 1.6
	ACC	23.2 \pm 0.7	23.5 \pm 1.2	22.9 \pm 0.8	23.6 \pm 0.9	21.8 \pm 1.3	23.0 \pm 1.7
ImageNet10 [36] ($N = 13000, d = 150528, k = 10$)	NMI	14.3 \pm 0.7	14.2 \pm 0.7	14.4 \pm 0.5	15.7 \pm 0.9	12.6 \pm 3.1	14.8 \pm 0.7
	ARI	8.5 \pm 0.7	8.0 \pm 0.6	8.0 \pm 0.5	7.9 \pm 0.6	6.5 \pm 2.0	8.4 \pm 1.4
	ACC	27.1 \pm 0.8	26.3 \pm 1.0	27.2 \pm 0.8	24.6 \pm 0.8	24.4 \pm 3.5	26.7 \pm 1.6
ImageNetDog [37] ($N = 2574, d = 150528, k = 15$)	NMI	6.3 \pm 0.4	5.9 \pm 0.4	<u>6.1 \pm 0.4</u>	5.5 \pm 0.5	5.9 \pm 0.4	6.1 \pm 0.4
	ARI	1.9 \pm 0.2	1.7 \pm 0.2	<u>1.8 \pm 0.2</u>	1.5 \pm 0.2	1.7 \pm 0.1	1.9 \pm 0.2
	ACC	13.6 \pm 0.5	13.3 \pm 0.4	<u>13.5 \pm 0.5</u>	13.0 \pm 0.6	13.4 \pm 0.5	13.7 \pm 0.6

Results: In Fig. 2 we show the ACC results on USPS, MNIST, and ImageNet10. Almost all methods perform best at $m = 5$ or $m = 10$. It seems that there is enough information in this setting to achieve good clustering results, while keeping unnecessary and potentially harmful information low. Overall, the embedding size has a similar impact on all DC algorithms. An exception is AE+KMeans on ImageNet10. Here the additional features lead to an improved result, but except for DEC, which improves slightly from $m = 50$ to $m = 100$, the DC methods did not benefit from the better initialization.

E. Impact of Cluster Initialization

The considered DC algorithms have all in common that they use an initial clustering solution found by KMeans. In this experiment we consider different partitioning-based clustering algorithms for initialization, namely, Expectation Maximization (EM) [42], Spectral Clustering [43] and SubKmeans [44], and compare them against KMeans.

Setup: Here we only analyze DEC, IDEC and DCN, together with the AE baselines. ACe/DeC is not included as it needs a specialized cluster initialization. Further, due to computational constraints we only consider the USPS and MNIST data sets.

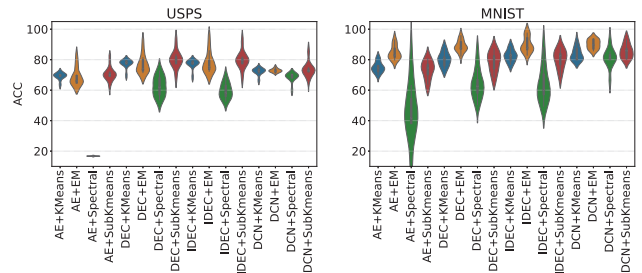


Fig. 3. **Impact of Cluster Initialization:** Accuracy (ACC) scores for varying initial clustering algorithms.

Results: In Fig. 3 we show the impact of the cluster initialization across 10 runs. For USPS the standard KMeans initialization is performing best, together with SubKmeans. Spectral clustering (AE+Spectral) got poor results, which might be due to outliers and/or noise. Despite the poor initial spectral clustering all DC algorithms were able to recover from it, although the performance still suffered. For MNIST, EM (AE+EM) performed best. This head start benefited the DC algorithms and outperformed the KMeans initialization.

TABLE III
RESULTS SMALL FEEDFORWARD AUTOENCODER: CLUSTERING RESULTS OF VARIOUS DEEP CLUSTERING APPROACHES ON DIFFERENT DATA SETS USING A FEEDFORWARD AUTOENCODER WITH $d=512-256-128-10$ NEURONS. THE NOTATION CORRESPONDS TO THAT FROM TAB. II.

Dataset	Metric	AE+KMeans	DEC	IDEC	DCN	ACe/DeC	DipEncoder
Optdigits [30] ($N = 5620, d = 64, k = 10$)	NMI	79.4 ± 2.3	86.9 ± 2.3	85.6 ± 2.0	82.4 ± 2.3	79.8 ± 3.3	86.2 ± 1.7
	ARI	73.7 ± 4.2	81.8 ± 4.4	80.4 ± 3.5	76.2 ± 4.3	72.3 ± 5.4	81.5 ± 3.2
	ACC	84.0 ± 3.5	88.2 ± 3.8	87.5 ± 3.1	85.3 ± 3.4	82.3 ± 4.5	88.3 ± 3.2
USPS [31] ($N = 9298, d = 256, k = 10$)	NMI	68.1 ± 1.6	79.9 ± 1.7	79.9 ± 1.5	75.3 ± 1.5	70.5 ± 1.8	80.1 ± 1.8
	ARI	58.6 ± 2.1	71.5 ± 2.8	71.6 ± 2.5	66.1 ± 2.3	61.9 ± 5.2	72.5 ± 2.9
	ACC	69.7 ± 2.8	76.1 ± 3.5	76.1 ± 3.5	73.4 ± 3.0	68.8 ± 5.8	76.9 ± 3.7
MNIST [32] ($N = 70000, d = 784, k = 10$)	NMI	74.3 ± 1.1	83.6 ± 1.8	86.1 ± 0.9	83.1 ± 1.0	81.4 ± 3.7	86.1 ± 1.2
	ARI	68.7 ± 1.8	78.8 ± 2.9	82.3 ± 1.6	80.1 ± 2.1	77.2 ± 6.1	82.5 ± 2.3
	ACC	80.2 ± 1.6	85.4 ± 1.7	87.7 ± 1.1	88.9 ± 2.7	86.0 ± 5.4	87.4 ± 1.2
FMNIST [33] ($N = 70000, d = 784, k = 10$)	NMI	59.4 ± 1.8	61.4 ± 1.7	62.6 ± 1.6	60.6 ± 1.6	59.7 ± 2.1	59.8 ± 1.9
	ARI	42.3 ± 2.4	43.1 ± 2.9	44.5 ± 2.4	42.5 ± 2.0	42.4 ± 3.0	43.1 ± 2.7
	ACC	53.2 ± 3.2	54.3 ± 2.9	54.7 ± 3.2	53.8 ± 2.9	54.3 ± 3.6	55.3 ± 2.3
KMNIST [34] ($N = 70000, d = 784, k = 10$)	NMI	49.7 ± 1.6	54.8 ± 1.7	54.5 ± 1.7	52.2 ± 1.1	52.0 ± 2.3	56.2 ± 2.0
	ARI	36.5 ± 1.9	40.5 ± 1.9	40.6 ± 2.1	38.5 ± 1.9	38.6 ± 2.3	42.8 ± 2.6
	ACC	56.0 ± 2.6	58.0 ± 3.0	58.6 ± 3.1	58.3 ± 1.2	58.5 ± 4.3	59.8 ± 2.7
CIFAR10 [35] ($N = 60000, d = 3072, k = 10$)	NMI	11.7 ± 0.6	11.3 ± 0.4	10.1 ± 0.8	12.2 ± 0.6	9.8 ± 0.7	11.8 ± 1.3
	ARI	6.6 ± 0.4	6.4 ± 0.2	5.6 ± 0.5	6.6 ± 0.4	5.2 ± 0.5	6.3 ± 1.2
	ACC	24.7 ± 0.9	23.9 ± 0.9	22.0 ± 1.0	23.5 ± 1.1	22.0 ± 1.3	24.5 ± 1.8
ImageNet10 [36] ($N = 13000, d = 150528, k = 10$)	NMI	14.7 ± 1.1	15.7 ± 1.2	14.4 ± 1.7	11.7 ± 1.5	6.6 ± 2.3	15.7 ± 1.3
	ARI	8.5 ± 0.9	9.0 ± 1.1	8.0 ± 1.4	5.5 ± 0.7	3.1 ± 1.1	10.2 ± 1.6
	ACC	27.0 ± 1.3	26.9 ± 1.7	26.4 ± 1.8	21.3 ± 1.0	17.9 ± 2.1	28.2 ± 1.6
ImageNetDog [37] ($N = 2574, d = 150528, k = 15$)	NMI	6.1 ± 0.5	5.6 ± 0.3	6.0 ± 0.5	4.7 ± 0.6	4.7 ± 0.8	6.1 ± 0.4
	ARI	1.8 ± 0.2	1.6 ± 0.2	1.7 ± 0.2	1.2 ± 0.3	1.2 ± 0.3	1.9 ± 0.2
	ACC	13.6 ± 0.5	13.1 ± 0.4	13.2 ± 0.5	12.4 ± 0.6	12.6 ± 0.7	13.6 ± 0.5

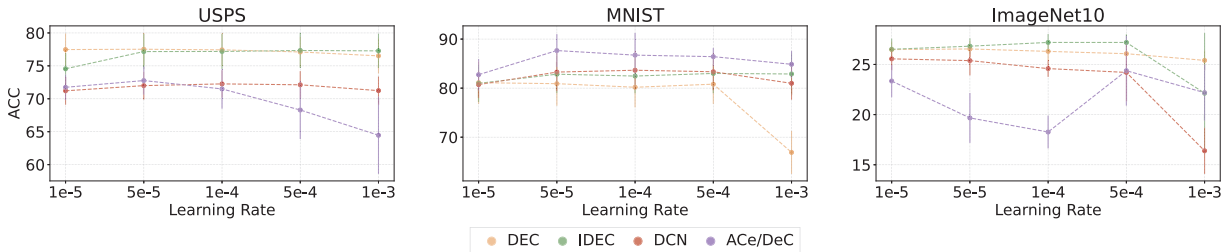


Fig. 4. **Impact of Learning Rate:** Accuracy (ACC) scores (mean and standard deviation) with varying learning rates of the AE with $d=500-500-2000-10$.

F. Impact of Learning Rate

The learning rate is a crucial hyperparameter in deep learning and thus for DC. In real world clustering applications we usually do not have access to ground truth labels to tune the learning rate. Therefore, we investigate how sensitive the considered DC algorithms are to the choice of learning rate.

Setup: For this experiment we only vary the learning rate of the DC methods and keep the pretrained autoencoder the same. We only consider learning rates that are smaller than the learning rate of 0.001 used during pretraining to achieve stable training. Due to computational constraints we use only USPS, MNIST and ImageNet10. Note, that we do not use a scheduler to reduce the learning rate over time.

Results: Fig. 4 shows the impact the learning rate has on

ACC. We see that for all three data sets the methods that are using KMeans-like update mechanisms (ACe/DeC and DCN) are more impacted by the choice of learning rate than the ones that are based on the KL divergence loss (DEC and IDEC). Especially, ACe/DeC performs worse for higher learning rates for USPS. This is probably why in the ACe/DeC paper the learning rate was reduced every 20% of training steps.

G. Generalization Performance

In DC research, and clustering research in general, splitting a data set into a training and testing part is not common practice. We believe that at least for DC, this is not well supported as the learned parameters of a model can easily be used to predict cluster labels on new, unseen data.

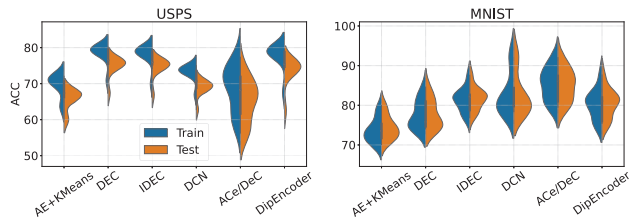


Fig. 5. **Generalization Performance:** Accuracy (ACC) scores for cluster predictions on train and test splits.

Setup: In this experiment we consider USPS and MNIST, as they are both data sets of handwritten digits. USPS is smaller and has a lower resolution than MNIST, which makes it a useful check for overfitting. We first pretrain 10 AEs with d -500-500-2000-10 on the PyTorch train splits of USPS and MNIST, respectively, and perform the DC algorithms. After training, we predict the cluster labels for the train data and for the unseen test data and measure the accuracy.

Results: In Fig. 5 we see that the same AE architecture overfits on USPS, while generalizing well on MNIST where train and test distributions roughly match. On USPS ACe/DeC suffers less from overfitting, but the variance is higher overall.

H. Impact of Augmentation

Augmentations are a common way to infuse domain knowledge into DC methods. Augmentation usually benefits all methods by learning cluster assignments that are invariant to certain transformations. For example, when clustering handwritten digits, we know that the cluster assignment should not change when we slightly rotate the digit.

Setup: We focus on the grayscale data sets as their set of augmentations is much smaller than for the color images. The augmentations we consider are random rotations, translations and shearing.² We first pretrain the d -500-500-2000-10 on the augmented data and then use the original and augmented data during the DC procedures.

Results: Tab. IV shows the impact of augmentation on clustering performance. We can see that just using the augmented data during pretraining improves the baseline AE+KMeans considerably. Except for a single instance all DC algorithms further benefit from the additional domain knowledge. For example, the maximum ACC achieved has increased from 86.4 to 94.5 on MNIST.

I. Impact of Architecture

We replace the feedforward AE backbone with a ResNet18 AE to measure the impact of an architecture that has an inductive bias suited for images.

Setup: We exchange the feedforward AE of the DC methods by a ResNet18 convolutional AE. All other settings remain the same, including the embedding size $m = 10$ and number of pretraining epochs.

²We use PyTorch's `torchvision.transforms.RandomAffine(degrees=(-16, +16), translate=(0.1, 0.1), shear=(-8, 8), fill=0)` function.

Results: Tab. V shows that the results regarding KMnist are improved for IDEC, DCN and the DipEncoder. The DipEncoder also greatly improved its performance on Optdigits. The results on the more complex data sets ImageNet10 and ImageNetDog, on the other hand, have unexpectedly decreased drastically. An explanation for the sub-optimal results might be that the hyperparameter settings of the feedforward AE (learning rate, training time, no weight decay, etc.) are not suited for the ResNet18 AE. A detailed tuning of the backbone is out of scope of this work, as our focus is the relative performance of DC algorithms given a pretrained model.

V. DISCUSSION

Clustering Performance: To summarize our results we found that across all settings and data sets the originally formulated DEC and its extension IDEC perform quite well. The newly introduced algorithms ACe/Dec and DipEncoder can occasionally outperform them, but the results are quite comparable.

Choice of Autoencoder: The feedforward AE performs strongly for the grayscale image data sets, but fails to learn good features for the more complex color image data sets. Other self-supervised learning approaches like masked image modeling [45] or contrastive learning [46] might be better suited to learn an initial representation for clustering images.

Hyperparameter Selection: Hyperparameter selection in unsupervised learning is a big challenge, especially for DC methods that have many possible settings. Our experiments on the embedding size and learning rate provide further evidence for this dependence.

Overfitting: The generalization performance experiments indicate that overfitting can be a problem, especially on small data sets like USPS. This result suggests that future benchmarks of novel DC algorithms should use a train and test split similar to supervised learning.

Importance of Initial Clustering: All experiments show that the initial clustering is important for all considered DC methods. The quality of the initial clustering is dependent on the chosen initial clustering algorithm and the learned embedding. While this was expected, it provides further motivation for researching initialization and pretraining strategies for DC.

VI. CONCLUSION AND FUTURE WORK

We performed the first in-depth benchmark of DC algorithms in a common framework. We found several interesting insights, such as the impact of choice of architecture, the robust performance of established methods like DEC/IDEC and the problem of overfitting on small data sets. Motivated by these initial results, we want to extend the benchmark to more algorithms, data sets, pretraining methods and architectures in future work. Further, we are interested in exploring other pretraining strategies besides autoencoding, like contrastive learning and masked image modeling.

REFERENCES

- [1] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv preprint arXiv:1801.07648*, 2018.

TABLE IV
RESULTS AUGMENTATION: CLUSTERING RESULTS OF VARIOUS DEEP CLUSTERING APPROACHES ON DIFFERENT DATA SETS USING A FEEDFORWARD AUTOENCODER WITH $d=500-500-2000-10$ NEURONS AND ADDITIONAL IMAGE AUGMENTATIONS. THE NOTATION CORRESPONDS TO THAT FROM TAB. II.

Dataset	Metric	AE+KMeans	DEC	IDEC	DCN	ACe/DeC	DipEncoder
Optdigits [30] ($N = 5620, d = 64, k = 10$)	NMI	85.1 ± 2.5	90.8 ± 2.3	90.6 ± 2.0	87.4 ± 2.5	86.5 ± 2.1	90.9 ± 2.4
	ARI	82.0 ± 4.9	87.0 ± 4.5	87.4 ± 4.4	84.1 ± 4.9	82.4 ± 4.1	88.3 ± 4.8
	ACC	89.6 ± 4.3	91.9 ± 4.2	92.2 ± 3.9	90.5 ± 4.1	89.3 ± 3.7	92.9 ± 4.2
USPS [31] ($N = 9298, d = 256, k = 10$)	NMI	78.5 ± 0.7	89.2 ± 1.6	90.5 ± 1.1	83.6 ± 1.0	86.8 ± 1.5	86.4 ± 2.1
	ARI	71.6 ± 2.3	86.7 ± 3.0	88.0 ± 2.8	77.0 ± 3.2	80.7 ± 4.9	79.9 ± 4.5
	ACC	78.6 ± 3.5	88.5 ± 3.8	89.6 ± 2.9	81.1 ± 4.0	83.5 ± 5.5	82.7 ± 4.7
MNIST [32] ($N = 70000, d = 784, k = 10$)	NMI	77.8 ± 1.0	90.7 ± 0.7	92.3 ± 1.9	90.6 ± 1.4	90.2 ± 1.3	89.4 ± 0.4
	ARI	72.2 ± 2.0	86.3 ± 0.8	89.1 ± 4.3	88.6 ± 4.2	89.8 ± 2.7	85.0 ± 1.3
	ACC	81.6 ± 2.1	88.6 ± 1.9	91.4 ± 4.9	92.4 ± 5.1	94.5 ± 3.2	87.6 ± 2.1
FMNIST [33] ($N = 70000, d = 784, k = 10$)	NMI	60.4 ± 2.0	57.6 ± 1.9	62.8 ± 1.2	60.9 ± 1.4	59.8 ± 0.9	55.3 ± 1.2
	ARI	43.1 ± 2.1	41.1 ± 2.6	45.6 ± 2.2	44.1 ± 1.8	42.8 ± 1.8	39.5 ± 1.7
	ACC	55.3 ± 3.0	52.6 ± 3.5	56.2 ± 2.9	55.2 ± 2.6	53.8 ± 2.8	51.6 ± 0.8
KMNIST [34] ($N = 70000, d = 784, k = 10$)	NMI	51.0 ± 1.4	63.5 ± 1.8	63.0 ± 2.0	57.8 ± 1.5	52.4 ± 4.0	62.3 ± 2.1
	ARI	34.0 ± 3.2	45.2 ± 3.5	44.3 ± 4.4	39.3 ± 4.0	35.1 ± 4.0	47.7 ± 2.2
	ACC	56.1 ± 3.1	62.3 ± 2.8	62.1 ± 3.2	59.3 ± 3.2	53.3 ± 4.4	64.3 ± 3.0

TABLE V
RESULTS RESNET18 CONVOLUTIONAL AUTOENCODER: CLUSTERING RESULTS OF VARIOUS DEEP CLUSTERING APPROACHES ON DIFFERENT DATA SETS USING A RESNET18 CONVOLUTIONAL AUTOENCODER WITH AN EMBEDDING SIZE OF $m = 10$. THE NOTATION CORRESPONDS TO THAT FROM TAB. II.

Dataset	Metric	AE+KMeans	DEC	IDEC	DCN	ACe/DeC	DipEncoder
Optdigits [30] ($N = 5620, d = 8 \times 8, k = 10$)	NMI	74.4 ± 2.8	81.9 ± 2.8	87.8 ± 1.8	83.5 ± 2.3	77.1 ± 3.6	92.1 ± 3.4
	ARI	69.2 ± 4.6	75.6 ± 4.5	82.5 ± 3.6	75.5 ± 5.1	69.8 ± 5.1	90.3 ± 5.5
	ACC	81.0 ± 4.1	85.4 ± 3.5	88.7 ± 3.1	84.0 ± 4.1	80.0 ± 3.5	94.7 ± 3.5
USPS [31] ($N = 9298, d = 16 \times 16, k = 10$)	NMI	60.1 ± 2.4	69.8 ± 3.8	80.2 ± 1.7	55.0 ± 10.3	62.9 ± 4.6	76.2 ± 4.1
	ARI	50.1 ± 3.0	61.4 ± 3.8	71.3 ± 2.9	34.5 ± 12.1	54.1 ± 4.6	67.9 ± 4.9
	ACC	64.2 ± 3.0	69.9 ± 2.7	75.3 ± 3.9	46.0 ± 10.3	65.6 ± 4.5	73.6 ± 3.6
MNIST [32] ($N = 70000, d = 28 \times 28, k = 10$)	NMI	64.3 ± 2.2	77.4 ± 4.4	84.9 ± 2.6	84.3 ± 2.6	58.5 ± 11.8	83.2 ± 3.1
	ARI	55.5 ± 3.3	68.2 ± 5.6	74.8 ± 3.9	72.5 ± 4.5	47.5 ± 13.6	76.2 ± 5.2
	ACC	68.3 ± 3.3	77.0 ± 4.2	78.5 ± 3.6	76.9 ± 4.1	62.4 ± 12.9	83.2 ± 5.9
FMNIST [33] ($N = 70000, d = 28 \times 28, k = 10$)	NMI	51.9 ± 0.9	51.1 ± 3.3	58.6 ± 1.8	60.1 ± 2.3	50.9 ± 4.3	56.5 ± 2.8
	ARI	35.5 ± 1.4	35.3 ± 4.7	41.0 ± 2.6	40.0 ± 3.4	35.3 ± 4.7	40.3 ± 4.0
	ACC	49.3 ± 1.8	51.4 ± 5.2	53.5 ± 2.8	52.0 ± 4.6	49.3 ± 4.0	54.0 ± 4.7
KMNIST [34] ($N = 70000, d = 28 \times 28, k = 10$)	NMI	49.0 ± 1.2	55.7 ± 2.7	61.9 ± 1.2	61.7 ± 1.7	33.2 ± 4.4	63.8 ± 1.4
	ARI	37.9 ± 1.2	44.1 ± 2.9	46.7 ± 1.1	44.8 ± 2.5	20.1 ± 4.4	51.1 ± 1.5
	ACC	61.2 ± 2.0	62.9 ± 3.2	67.5 ± 2.0	66.5 ± 2.6	37.9 ± 5.4	68.4 ± 2.1
CIFAR10 [35] ($N = 60000, d = 32 \times 32 \times 3, k = 10$)	NMI	11.9 ± 0.3	8.3 ± 1.5	9.9 ± 0.8	6.7 ± 1.0	6.6 ± 1.4	13.0 ± 0.7
	ARI	6.6 ± 0.2	4.6 ± 0.9	5.9 ± 0.5	3.1 ± 0.7	3.3 ± 0.8	7.6 ± 0.4
	ACC	24.6 ± 0.8	21.5 ± 1.5	22.6 ± 1.3	17.6 ± 1.3	19.0 ± 1.8	26.2 ± 1.1
ImageNet10 [36] ($N = 13000, d = 224 \times 224 \times 3, k = 10$)	NMI	2.5 ± 0.2	8.2 ± 3.5	3.0 ± 0.2	7.1 ± 0.2	13.2 ± 1.8	3.3 ± 0.8
	ARI	1.2 ± 0.1	3.3 ± 1.1	1.6 ± 0.1	3.6 ± 0.2	7.0 ± 1.1	1.6 ± 0.5
	ACC	14.4 ± 0.3	18.4 ± 1.8	15.7 ± 0.3	19.0 ± 0.6	24.4 ± 1.5	15.7 ± 0.7
ImageNetDog [37] ($N = 2574, d = 224 \times 224 \times 3, k = 15$)	NMI	2.8 ± 0.3	6.1 ± 2.2	2.6 ± 0.2	4.2 ± 0.2	6.1 ± 0.3	3.1 ± 0.3
	ARI	0.5 ± 0.1	1.4 ± 0.6	0.4 ± 0.1	1.0 ± 0.1	1.8 ± 0.2	0.6 ± 0.1
	ACC	11.0 ± 0.3	12.5 ± 1.2	10.5 ± 0.3	12.0 ± 0.3	13.2 ± 0.4	11.3 ± 0.3

- [2] D. H. Ballard, "Modular learning in neural networks," in *Proceedings of the sixth National Conference on artificial intelligence-volume 1*, 1987, pp. 279–284.
- [3] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*. PMLR, 2016, pp. 478–487.
- [4] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation," in *Ijcai*, vol. 17, 2017, pp. 1753–1759.
- [5] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*. PMLR, 2017, pp. 3861–3870.
- [6] L. Miklautz, L. G. Bauer, D. Mautz, S. Tschitschek, C. Böhm, and C. Plant, "Details (don't) matter: Isolating cluster information in deep embedded spaces," in *IJCAI*, 2021, pp. 2826–2832.
- [7] C. Leiber, L. G. Bauer, M. Neumayr, C. Plant, and C. Böhm, "The dipencoder: Enforcing multimodality in autoencoders," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 846–856.
- [8] M. M. Fard, T. Thonet, and E. Gaussier, "Deep k-means: Jointly clustering with k-means and learning representations," *Pattern Recognition Letters*, vol. 138, pp. 185–192, 2020.
- [9] D. Mautz, C. Plant, and C. Böhm, "Deep embedded cluster tree," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 1258–1263.
- [10] C. Leiber, L. G. Bauer, B. Schelling, C. Böhm, and C. Plant, "Dip-based deep embedded clustering with k-estimation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 903–913.
- [11] L. Miklautz, M. Teuffenbach, P. Weber, R. Perjuci, W. Durani, C. Böhm, and C. Plant, "Deep clustering with consensus representations," in *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2022, pp. 1119–1124.
- [12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [13] L. Miklautz, D. Mautz, M. C. Altinigneli, C. Böhm, and C. Plant, "Deep embedded non-redundant clustering," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5174–5181.
- [14] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*. Springer, 2017, pp. 373–382.
- [15] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 132–149.
- [16] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5147–5156.
- [17] K. Ghasedi Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5736–5745.
- [18] W. Guo, K. Lin, and W. Ye, "Deep embedded k-means clustering," in *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2021, pp. 686–694.
- [19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimselshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [20] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.
- [21] J. A. Hartigan and P. M. Hartigan, "The dip test of unimodality," *The annals of Statistics*, pp. 70–84, 1985.
- [22] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. 12, 2010.
- [23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [25] V. Škvára, J. Francá, M. Zorek, T. Pevný, and V. Šmídl, "Comparison of anomaly detectors: context matters," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 6, pp. 2494–2507, 2021.
- [26] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 142–32 159, 2022.
- [27] A. Javed, B. S. Lee, and D. M. Rizzo, "A benchmark study on time series clustering," *Machine Learning with Applications*, vol. 1, p. 100001, 2020.
- [28] S. Affeldt, L. Labiod, and M. Nadif, "Spectral clustering via ensemble deep autoencoder learning (sc-edae)," *Pattern Recognition*, vol. 108, p. 107522, 2020.
- [29] —, "Caeclust: A consensus of autoencoders representations for clustering," *Image Processing On Line*, vol. 12, pp. 590–603, 2022.
- [30] E. Alpaydin and C. Kaynak, "Optical Recognition of Handwritten Digits," UCI Machine Learning Repository, 1998, DOI: <https://doi.org/10.24432/C50P49>.
- [31] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 5, pp. 550–554, 1994.
- [32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [33] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
- [34] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *arXiv preprint arXiv:1812.01718*, 2018.
- [35] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
- [36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.
- [37] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, vol. 2, no. 1. Citeseer, 2011.
- [38] T. O. Kvalseth, "Entropy and correlation: Some comments," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 3, pp. 517–519, 1987.
- [39] Y. Yang, D. Xu, F. Nie, S. Yan, and Y. Zhuang, "Image clustering using local discriminant models and global integration," *IEEE Transactions on Image Processing*, vol. 19, no. 10, pp. 2761–2773, 2010.
- [40] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, pp. 193–218, 1985.
- [41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [42] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B*, vol. 39, no. 1, pp. 1–22, 1977.
- [43] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.
- [44] D. Mautz, W. Ye, C. Plant, and C. Böhm, "Towards an optimal subspace for k-means," in *KDD*. ACM, 2017, pp. 365–373.
- [45] C. Feichtenhofer, H. Fan, Y. Li, and K. He, "Masked autoencoders as spatiotemporal learners," 2022.
- [46] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 1597–1607.