applied sciences

MDPI

Article

Feature Transformation Framework for Enhancing Compactness and Separability of Data Points in Feature Space for Small Datasets

Mahmoud Maher ElMorshedy *, Radwa Fathalla * and Yasser El-Sonbaty

Computer Science Department, Arab Academy for Science and Technology and Maritime Transport, College of Computing and Information Technology, Alexandria 1029, Egypt; yasser@aast.edu
* Correspondence: elmorshedy@aast.edu (M.M.E.); radwa_fathalla@aast.edu (R.F.)

Abstract: Compactness and separability of data points are two important properties that contribute to the accuracy of machine learning tasks such as classification and clustering. We propose a framework that enhances the goodness criteria of the two properties by transforming the data points to a subspace in the same feature space, where data points of the same class are most similar to each other. Most related research about feature engineering in the input data points space relies on manually specified transformation functions. In contrast, our work utilizes a fully automated pipeline, in which the transformation function is learnt via an autoencoder for extraction of latent representation and multi-layer perceptron (MLP) regressors for the feature mapping. We tested our framework on both standard small datasets and benchmark-simulated small datasets by taking small fractions of their samples for training. Our framework consistently produced the best results in all semi-supervised clustering experiments based on K-means and different seeding techniques, with regards to clustering metrics and execution time. In addition, it enhances the performance of linear support vector machine (LSVM) and artificial neural network (ANN) classifier, when embedded as a preprocessing step before applying the classifiers.

Keywords: compactness; separability; feature space; feature mapping; small datasets; autoencoder; neural networks; support vector machines; classification; semi-supervised clustering

Citation: ElMorshedy, M.M.; Fathalla, R.; El-Sonbaty, Y. Feature Transformation Framework for Enhancing Compactness and Separability of Data Points in Feature Space for Small Datasets. *Appl. Sci.* **2022**, *12*, 1713. https://doi.org/10.3390/app12031713

Academic Editor: Valentino Santucci

Received: 24 December 2021
Accepted: 29 January 2022
Published: 7 February 2022

1. Introduction

One of the fundamental concepts in machine learning is the embedding of data points in an appropriate feature space [1]. The choice of the feature space should be driven by enhancing the topological properties of data clusters, ultimately leading to better performance in pattern recognition tasks such as classification [2–4], clustering [5–7], information retrieval [8–11], and regression [12,13]. Data points and latent features are two spaces where feature engineering approaches operate.

One approach is projecting the data descriptors to an alternative feature space via a hard-coded mathematical function. The evident example of this approach is the use of kernel functions in SVM classifiers, aiming to improve the separability property to allow efficient localization of the hyperplane [14]. Another example is the latent representation yielded on the hidden layers of ANNs [15,16]. The goal is to increase the compactness of the class samples by abstracting the input data from its specifics and details. Clearly there is a difference between SVMs and ANNs in the intermediate goal; one opts for increasing separability while the other increases compactness. In addition, in the former technique, the mapping is hand-engineered kernels, whereas in the latter the mapping function is learned.

On another front, feature transformations retaining the original data space have been limited to feature selection, normalization, and scaling [17]. Conceptually, non-parametric learning approaches such as KNN [18] also belong to this paradigm of feature transformation and can be regarded as a projection of test samples into a subspace of training samples.

When the model is presented with unseen data, it uses a distance function to map the data to the most similar pre-seen example. In essence, it is a classification approach with a hypothetical embedded feature transformation step based on a distance mapping function. Limitations of KNN have been thoroughly discussed in the literature [19–22]. One of its drawbacks is the reliance of the mapping function on a linear distance measure [20] (e.g., Manhattan, Euclidean, Minkowski, and weighted Euclidean distance) between corresponding individual features. However, it neglects the interrelation between features comprising the data samples and their collaborative effect, thus limiting the ability of the distance function in expressing the similarity/dissimilarity of the samples [22]. In addition, the KNN method does not promote properties of compactness and separability of data embeddings in the feature space.

Additionally, oversampling approaches operate both spaces [17,23]. Oversampling creates synthetic examples by relocating feature vectors of given examples along the line segments joining the k nearest neighbor examples of the same class. Although this approach shows improvement in the accuracy of classification tasks in case of imbalanced and small datasets, it does not improve the compactness and separability of data points. It could happen that a minority class sample could have a majority class sample as its neighbor, which will create false positive samples.

State-of-the-art deep learning algorithms have achieved near-human-level accuracy in applications such as computer vision [24–28]. Nevertheless, one challenge remains for these algorithms when faced with inadequate amounts in acquiring adequate amounts of labeled data [29]. Otherwise, these models are subject to overfitting [30] due to the large number of parameters to learn in complex networks which reduce the network's ability to generalize and cause the network to memorize examples. Also, labeling is widely regarded as a manual labor-intensive and resource-consuming process. In other cases, there are great difficulties in the initial acquisition process of the data samples, a problem that is well-known in the field of bioinformatics [31]. All these factors result in the occurrence of small-sized datasets in either supervised or semi-supervised settings, thus limiting the generalization ability of the end classifier [32]. This challenge is exaggerated in the case of the existence of high intra-class variability [33,34], which leads to classes of low density in the feature space. In fact, these challenging factors impact clustering algorithms as well [35]. Concurrently, they have a jarring effect on the clusters, creating disjoint subdivisions and hindering the discovery of the class structure in its entirety in space.

In our work, the main contribution is proposing a model architecture which fully automates the learning of the transformation function that operates on unseen samples so that they resemble approximations of samples encountered in the training set. There are two aspects evident in our framework. First, we preserve the same feature space as the original data points. Second, we relocate input samples in a subspace while improving the goodness criteria of separability and compactness [36–39]. To the best of our knowledge, this is the first time these aspects have been combined to carry out the task of selecting the feature vector embedding space.

To this end, we train two main network models that, combined, comprise the transformation function. We train an autoencoder for feature extraction and reconstruction of the training samples. We create Cartesian product pairs of the training samples resulting in a set of ordered pairs and extract their embeddings from the encoder of the trained autoencoder. Then, a bridging regression network learns to map the embeddings of the first element in each ordered pair (as input variables to the regression network) to the embeddings of the second element in the pair (as target variables). We use the encoder of the autoencoder, the bridging network, and the decoder from the autoencoder to transform the training and test samples of each class to very similar (template-like) samples of the same class resulting in a reduction in intra-class variance and an increase in the inter-class variance. This helps to enhance the accuracy over the transformed test set of an off-the-shelf classifier trained on the training samples augmented with the transformed training samples.

Data augmentation [17] is an effective technique to handle overfitting in the case of small datasets and enhance generalization of the machine learning model, particularly ANNs, by increasing both amount and diversity of data samples. The significance of our work is highlighted in the case of small datasets ($|n|$) with high variability. We fully exploit the samples available in the datasets by a pairing mechanism where we pair each example with all other examples of the same class and we study the samples' inter-relations $|n^2|$, thus incrementing the learning data pool. In addition, we augment the dataset with reconstructions of latent features of the samples after being subjected to a regression step.

Another contribution to this work is the applicability of our proposed model in a semi-supervised clustering setting where a few labeled data samples are utilized to learn clustering of the plentiful unlabeled data in a partially labeled dataset [40,41]. We use the labeled data samples to train our model to learn the transformation function, then we transform the unlabeled data using our trained model. We feed the transformed examples to a clustering technique. We show that not only is the clustering time of the transformed data points significantly reduced compared to using the same clustering techniques on the untransformed data points, but also all clustering metrics used to measure the performance of the clustering techniques notably improve.

Several efforts have been proposed in the field of feature engineering in general and in the settings where there is a focus on improving compactness and separability. Feature engineering is a preprocessing task to improve the prediction performance of a machine learning model on a dataset by applying a transformation function on its feature space. Many feature engineering techniques [42] yielding feature vectors in the same input data space are limited to scaling and normalization or simply applying hand-crafted arithmetic or aggregation functions to the features to construct more suitable features benefitting the machine learning model's performance. One problem of feature engineering is finding a particularly good transformation for a given set of features given a set of transformation functions. In [3], instead of enumerating and trying all transformation functions and their combinations by training and testing models, which is computationally impractical, the authors automate the decision-making process of recommending the transformation functions applied to the features. In our work, our aim is to learn the transformation function, instead of using hand-crafted transformation functions, and using our learnt parameters to relocate the feature space of the data samples.

Intra-class compactness and inter-class separability are commonly studied in the field of machine learning. The two characteristics are crucial measures of how effectively a model produces discriminant features. Approaches vary in tackling the problem of improving compactness and separability of features. In [39], the authors propose a feature optimizer that calculates a center for each class by averaging samples in a mini-batch and optimizes the calculated centers at each iteration by minimizing a feature optimization loss that consists of two distances, the distance of samples in the mini-batch to its correct center, and the other is to its nearest wrong center. Another approach that extends the loss function of a model is proposed by Pilarczyk et al. [36]. They use intra-class variance loss as a regularization term for training deep neural network (DNN) classifiers. This approach is different from the previous approach as the centroids of classes that contribute to calculating the intra-class variance loss are learned through an additional Hadamard layer attached to the convolutional neural network (CNN) architecture where its parameters represent the class centers. Luo et al. [37] proposed a Gaussian-based Softmax function that can be easily implemented and can replace the Softmax function in CNNs. The proposed function improves intra-class compactness and inter-class separability that improves classification on several multi-label classification datasets. Liu et al. [38] proposed to employ a margin constraint in the original Softmax function which explicitly encourages the learning of more discriminative features.

All the aforementioned approaches show the importance of enhancing the discriminant nature of learned features which consequently improves the performance of underlying models. However, a common limitation of these approaches is that they extend or

change the used loss function in DNN architectures limiting the choice of classifiers to DNN classifiers. This conclusion encourages us to tackle the problem using a different approach. Our approach uses a combination of supervised and generative models that create a framework for transforming the learned features into an enhanced feature subspace promoting the compactness and separability of classes. Our proposed model is a pluggable model before any classifier type model not limited by a particular type of classifiers, as in other approaches.

We also investigate the intra-cluster compactness and inter-cluster separability in the field of unsupervised and semi-supervised learning to enhance clustering performance. Most hard clustering techniques such as basic K-means [43] and clustering algorithms are based on K-means [44,45]. These types of clustering algorithm work by only minimizing the within-cluster distance while neglecting the between-cluster distance. Huang et al. [46] proposed a series of extension to the K-means-based clustering algorithms where they utilize the idea of a global centroid and maximizing the distance between clusters' centroids (increasing the between-cluster separation) while still minimizing the distances between data points and the centroids of the clusters (improving the within-cluster compactness). Soft-clustering algorithms that utilize the intra-cluster as well as the inter-cluster distances have also emerged, such as enhanced soft subspace clustering (ESSC) [47] and supervised enhanced soft subspace clustering (SESSC) [48]. SESSC is based on ESSC which considers both the within-cluster compactness and between-cluster separation. In addition, SESSC takes advantage of the label information in a semi-supervised setting. Therefore, SESSC can be used as a standalone classifier or integrated into a fuzzy classifier to further improve its performance.

The remainder of the paper is organized into three sections. Section 2 elaborates on our proposed method in three subsections corresponding to three modules of the proposed framework. Section 3 illustrates our conducted experiments and the evaluation of the proposed framework in two settings, a semi-supervised clustering setting, and a supervised setting. Finally, Section 4 concludes the paper.

2. Method

Given a dataset $D = \left\{D^{Tr},\ D^{Ts}\right\}$ split into two subsets, there are a training set D^{Tr} and a testing set D^{Ts} with $\mathcal{L}_{Tr} = \{\ell_i\}_{i=1}^{N}$ and $\mathcal{L}_{Ts} = \{\ell_i\}_{i=1}^{M}$ where N and M are $\left|D^{Tr}\right|$ and $\left|D^{Ts}\right|$, respectively. Typically, the given dataset is a small dataset or a dataset with very small number of labeled examples compared to the number of unlabeled examples $N \ll M$. The aim for our model is to learn a transformation function $f\left(X^k\right) = Y^k \;\middle|\; Y^k \cong \mu^k,\ k = 0 \ldots K-1$ where K is the number of classes in the dataset. The function maps each sample ($x_i^k \in X^k$) of the same class to a y_i^k, where y_i^k approximates μ^k centroid (mean sample) of the same class.

As illustrated in Figure 1, our proposed framework consists of three main modules: feature extraction, feature mapping, and reconstruction. For feature extraction, we train an autoencoder [49] AE, which is a generative model that reconstructs the available training samples in the dataset X^k, through learning their latent representation. In the feature mapping module, we first pair the training samples of each class with all other samples of the same class. Second, we use the trained autoencoder to extract the features of the pairs. Then, we train a mapping network that consists of a set of multilayer perceptron (MLP) regressors $\mathrm{B} = \left\{MLP_1,\ MLP_2,\ \ldots,\ MLP_{N_f}\right\}$, where N_f is the number of latent features in the embedding layer Z of the autoencoder AE. Each MLP regressor in the network learns to output $z_i^s \in Z,\ s = 1,\ 2,\ \ldots,\ N_f$. Finally, we take the mapped latent features $\hat{Z}$ from the feature mapping network B and feed it into the decoder of the trained autoencoder to reconstruct the mapped samples Y^k.

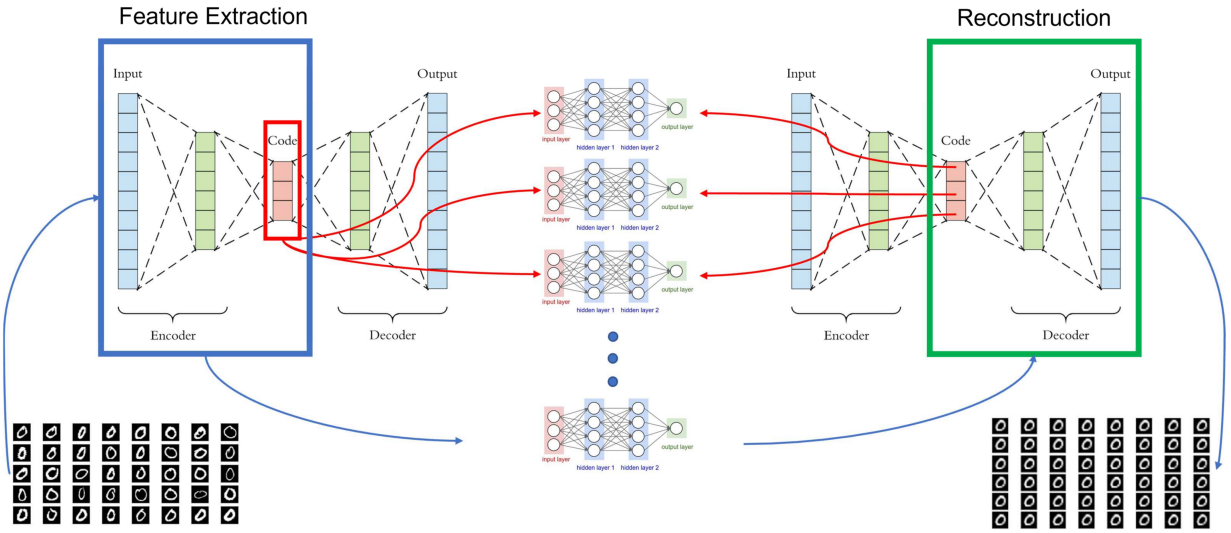


Figure 1. Architecture of the proposed framework.

After training the whole framework, we use it to map the test set examples to a more compact space for intra-class samples and separable for inter-class samples. Evaluating a classifier on the transformed samples gives better accuracy and loss than evaluating the same classifier on the untransformed examples. In the next three subsections, we explain the details of the three modules composing our framework.

2.1. Feature Extraction Using Autoencoders

Autoencoders are neural networks that are unsupervised (they can be thought of as self-supervised) learning models. Autoencoders were first introduced in [50] as neural networks that learn the compressed internal representation of some input while reconstructing the same input. An autoencoder tries to learn a function $g_{W,\,b}(x) = \hat{x}$ by minimizing the reconstruction error of the input so that $\hat{x}$ is as similar as possible to x. The autoencoder achieves this by encoding the input into the feature space using an encoder subnetwork, then decodes it back using a decoder subnetwork [49,51]. The learned representation (latent features) can be extracted and used in different machine learning tasks [52–55]. Following the same notion, we use autoencoders in our framework to extract features Z of samples X^k that are used as input in the next module of the framework for training the bridging network of MLP regressors.

The mathematics of an autoencoder can be formulated as defined in [56] as follows:

$$E : X \rightarrow Z, \tag{1}$$

$$D : Z \rightarrow \hat{X}, \tag{2}$$

Such that (1) and (2) satisfy:

$$\arg min_{\,E,\;D} \mathbb{E}[\Delta(X,\; D \circ E(X))], \tag{3}$$

where $\mathbb{E}$ is the expected value of the reconstruction error function Δ over the distribution of X. E is the encoder function that maps the original input X to a latent representation Z at the bottleneck of the autoencoder. The decoder function D does the opposite by mapping the latent features Z at the bottleneck to the output of the decoder network approximating the original input X. Training the encoder and the decoder networks involves optimizing the networks' parameters $(W,\, b)$ through backpropagating the reconstruction error between the decoder's output and the original input.

In our framework, we train a deep autoencoder network on the training set D^{Tr}, such that the autoencoder learns a latent representation Z for all samples in the dataset

while minimizing the mean-squared error (MSE) (as the error function Δ) of the decoder reconstructions.

2.2. Feature Mapping Network

In this module, we aim to train a network to learn a transformation function that could later be used to map the testing set to a feature space with improved properties of separability and compactness. Our approach is to learn a regression function that transforms the distribution of the embeddings of each class to a more compact distribution. As we shortly show, regressors with high bias and low variance tend to output a value close to the average of the target. Following this observation, we want to force the regression model even more to learn a value close to the average of each feature embedding.

Bias-variance tradeoff is a widely studied problem in the field of machine learning [57,58]. The dilemma was first formally introduced in [59], and it refers to a model property where the estimation error of the model can be decomposed into two terms, known as bias and variance, thus, either bias or variance can affect the performance of a model. The bias error is the difference between the expected (average) prediction of our model and the target correct value which results from oversimplifying the estimated function and missing the regularities in the training data. The variance error is the difference between the average prediction of our model for a given data point which results from the sensitivity of the model to irregularities in the training data.

As illustrated in Figure 2, there is often a tradeoff between the contribution of bias and variance to the estimation error as conflict arises from trying to simultaneously minimize the two error terms. The variance error can be reduced through smoothing the signal which will in effect introduce a high bias as the model will be oversmoothed and will miss the specifics of the underlying data points resulting in an underfitting scenario where both the training and test errors are high. On the other hand, the high bias can be reduced by increasing the complexity (number of fitting parameters) of the model, but then the variance of the model will increase as the model will have more capacity to fit the noise and irregularities of the underlying training data points and not generalizing to the target signal. This scenario is referred to as overfitting, giving a low training error while degrading the predictive performance on the test data.

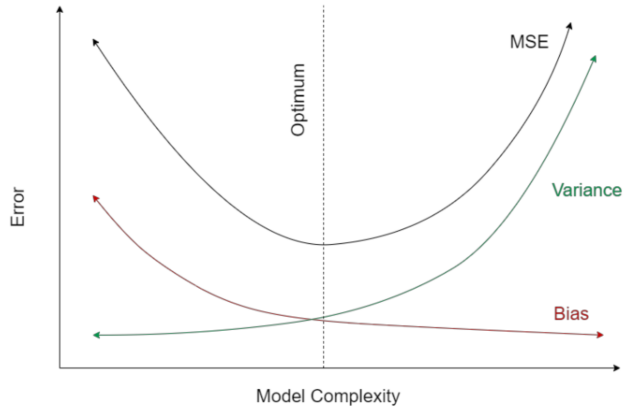


Figure 2. Bias and variance as a function of model complexity.

As we observe, the MLP regressor models for the bridging network learn to predict the average value of the target feature values for the same class. The intuition behind that observation is that the model is not complex enough to fit the underlying data (underfitting to data points), which means the model has too much bias. According to the bias-variance tradeoff, the high bias model will have a low variance to reduce the whole mean-squared error (MSE), which decomposes into the two terms of *Bias* and *Variance*:

$$MSE = Bias^2 + Variance \quad (4)$$

In the book *Doing Bayesian data analysis* [60], the author stated that the mean of a distribution is the value that minimizes the expected squared deviation. As can be mathematically proven (see Appendix A), the mean-squared error has a lower bound value as the variance of Y (the target output) when $\hat{Y}$ (the predicted output) equals the average of Y. Therefore, the model will predict values closer to the average, trying to reach the lower bound of the MSE.

We also found that if we train the regressor with different outputs for the same set of corresponding inputs, the regressor will minimize its error by outputting a value close to the average of the presented target values. A plausible explanation for this behavior is that having different target values for the same input features resembles adding noise to the output. Adding noise to the input or the output (less common) is a well-known regularization method to aid generalization (lowering the variance) and avoid overfitting [61–63]. According to the bias-variance tradeoff theory, reducing the variance error will, in effect, give a high bias model as the model's error decomposes into bias and variance errors. Therefore, adding variant targets for the same input encourages the network's output to be a smooth function of the input.

To feed the regressors with different target outputs for the same latent representation, we pair the features of each example with m examples of the same class. We experimented with different values for m, but we found that pairing each example with all other examples of the same class ($m = |D_k|$) gives better results.

The pairing algorithm can be described as getting the Cartesian Product for each set of samples D_k belonging to the same class with itself. Identical ordered pairs are excluded.

$$T_{CP} = D_k \times D_k = \{(a,\ b) \mid a \in D_k \ and \ b \in \{D_k - \{a\}\}\} \ \forall \ k \in K \tag{5}$$

As described in Algorithm 1, we do the Cartesian product of each class k set of samples by pairing each sample with all other samples of the same class k, such that if we have m examples in a class, we end up having $m \times (m-1)$ samples per class in the Cartesian product set T_{CP}. We form two sets T_1 and T_2 from the ordered pairs of the Cartesian product set T_{CP} such that $T_1 = \{a_i\},\ i = 0,\ 1, \ldots,\ m$ and $T_2 = \{b_j\},\ j = 0,\ 1, \ldots,\ m$ and $i \neq j$, i.e., we form two sets, T_1 for the first elements and T_2 for the second elements of the ordered pairs. The pairing algorithm is applied to the training set D^{Tr} where we group the training samples by labels $\mathcal{L}_{Tr}$, and results in two sets Tr_1 and Tr_2 (ground truth training set for the feature mapping network) with repeated examples in each, where $|Tr_1| = |Tr_2| = m \times (m-1) \times K$.

Algorithm 1. The Pairing Algorithm.

Require: D^{Tr}, $\mathcal{L}_{Tr}$, K
$Group\ D^{Tr}\ by\ \mathcal{L}_{Tr}$
$Tr_1 \leftarrow [\,]$, $Tr_2 \leftarrow [\,]$, $\mathcal{L}_{Tr1} \leftarrow [\,]$, $\mathcal{L}_{Tr2} \leftarrow [\,]$
$st = 0$
for $k \leftarrow 0$ *to* $K-1$ **do**
    $m \leftarrow getClassLength(k)$
    for $i \leftarrow 0$ *to* m **do**
        for $j \leftarrow 0$ *to* m, $j \neq i$ **do**
            $append\ D^{Tr}_{[st+i]}$ *to* Tr_1
            $append\ D^{Tr}_{[st+j]}$ *to* Tr_2
            $append\ \mathcal{L}_{Tr[st+i]}$ *to* $\mathcal{L}_{Tr1}$
            $append\ \mathcal{L}_{Tr[st+j]}$ *to* $\mathcal{L}_{Tr2}$
        end
    $st = st + m$
    end
end

After we have our two training sets (Tr_1, Tr_2) from the pairing algorithm, we extract the features Z_1 and Z_2 (ground truth latent features) for the two sets Tr_1 and Tr_2 using our trained autoencoder AE from the previous module. For the mapping network, we train a set of MLP regressors $\left\{MLP_1,\ MLP_2,\ \ldots,\ MLP_{N_f}\right\}$ where N_f is the number of embedded features $|Z_2|$, such that each MLP regressor corresponds to one latent feature $z^s_{2,\,i}$ of the N_f features in Z_2, and we use $Z_{1,\,i}$ as the same input for all MLP regressors. We can either train the N_f MLP regressors serially or in parallel as they correspond to independent outputs, so we choose to train them in parallel to speed up the training process.

2.3. Mapped Features Reconstruction

In the third module of our framework, we use the decoder of the trained autoencoder AE to reconstruct the latent representations $\hat{Z}$ obtained from the regressors, in the feature mapping network module, back to the input space Y^k, where y^k_i approximates μ^k centroid (mean sample) of the same class.

A summary of the transformation pipeline is shown in Algorithm 2. Given a training set D^{Tr} and a test set D^{Ts}, we first extract the features Z_{tr} of D^{Tr} from the trained autoencoder AE and map Z_{tr}, using the trained regressors in the feature mapping network B, to $\hat{Z_{tr}}$, then, we use the decoder of AE to reconstruct $\hat{Z_{tr}}$ to the input space yielding $\hat{D^{Tr}}$. We do the same for D^{Ts} resulting in the transformed test set $\hat{D^{Ts}}$.

Our proposed framework can be embedded as a preprocessing step in semi-supervised and supervised settings. In the semi-supervised setting, we use the limited training (labeled) set D^{Tr} to train our framework and transform the test (unlabeled) set D^{Ts} and cluster the transformed unlabeled set $\hat{D^{Ts}}$ using a clustering algorithm. In the supervised setting, we use an off-the-shelf classifier in the following setting: the plain training set D^{Tr} augmented with the transformed training set $\hat{D^{Tr}}$ using our framework as shown in Algorithm 2. The augmentation of the mapped training samples increases the size of the learning data pool which is beneficial in the case of small size datasets where the lack of training samples increases the chance of overfitting. Our augmentation resembles SMOTE augmentation in that we both operate in the feature space to generate the augmented samples. However, our augmentation is different in that we do not interpolate new data samples among a number of neighboring samples, conversely, we transform the available data samples to a group of similar data samples approximating the mean sample of the target class. Also, SMOTE increases the density of samples of the same class within the same boundaries of data clusters without relocation of clusters which does not provide any improvement in the separability and compactness characteristics.

Algorithm 2. The Transformation Pipeline.

Require:	$D^{Tr},\ D^{Ts}$
1	$Z_{tr} \leftarrow Extract\ features\ of\ D^{Tr}\ from\ the\ trained\ autoencoder\ AE$
2	$\hat{Z_{tr}} \leftarrow Predict\ features\ using\ the\ trained\ MLP\ regressors\ and\ Z_{tr}\ as\ input$
3	$\hat{D^{Tr}} \leftarrow Reconstruct\ the\ predicted\ features\ \hat{Z_{tr}}\ using\ the\ decoder\ of\ AE$
4	$Repeat\ for\ D^{Ts}$

3. Evaluation

To evaluate our proposed model, we tested our model in two different machine learning tasks: semi-supervised clustering and classification. We conducted experiments on five different datasets, and we calculated various clustering metrics for the semi-supervised clustering setting that show the model's ability to achieve its goal of improving compactness and separability among targets. Also, we showed that the model enhances an off-the-shelf classifier's accuracy indicating the classifier is able to achieve better decision boundaries on the target classes.

3.1. Datasets

We evaluated our proposed model on five different datasets. We used the three standard benchmark datasets: MNIST [64], Fashion-MNIST [65], USPS [66], and we further evaluated our model on two already small datasets: the three-target MSTAR dataset [67] which is widely used in automatic target recognition (ATR) of synthetic aperture radars (SAR) images, and the Breast Cancer Wisconsin (WDBC) dataset from the UCI repository [68]. For the benchmark datasets (MNIST, Fashion-MNIST, and USPS), we used a very small subset of the training examples to simulate a limited training data problem, and we appended the remaining training images to the test set.

Both MNIST and Fashion-MNIST datasets contain 60,000 training and 10,000 test samples, while the three-target MSTAR and WDBC datasets contain only 698 and 569 training samples, respectively, which are significantly less than those in the MNIST and Fashion-MNIST datasets. Therefore, we chose at random a ratio of 1% per class of the training images and appended the remaining images to the test set. We ended up with 596 training and 69,404 test images for the MNIST dataset, and 600 training and 69,400 test images for the Fashion-MNIST dataset. For the USPS dataset, which consists of 7291 training and 2007 test images, we selected 10% of the training images per class making a total of 725 training and 8573 test images. The samples of the WDBC dataset were split into 70% (398) training and 30% (171) testing samples. A summary of the number of training and testing samples, number of classes, and input sizes is listed in Table 1.

Table 1. The number of training and testing samples, number of classes, and input sizes for each dataset.

Dataset	MNIST	Fashion-MNIST	USPS	MSTAR	WDBC
No. of Training Samples	596	600	725	698	398
No. of Testing Samples	69,404	69,400	8573	587	171
No. of Classes	10	10	10	3	2
Input Size	28×28	28×28	16×16	128×128	30

3.2. Experimental Setup

We used an autoencoder architecture for feature extraction. The autoencoder architecture vary from dataset to another. A summary of the autoencoder architecture, batch size and number of epochs used for each dataset is presented in Table 2. For some datasets, we increased the number of epochs to test for potential overfitting when the model keeps training. Because of this, the model overfitting is subject to the model's complexity, especially, in the case of small datasets [69]. However, as can be seen from Figure 3, the reconstruction loss of the AE remains smooth as the model keeps training.

Table 2. Autoencoder architecture, batch sizes and number of epochs for each dataset.

Dataset	MNIST	Fashion-MNIST	USPS	MSTAR	WDBC
Flattened Input Size	784	784	256	16,384	30
AE Architecture	512-256-512	512-256-512	512-128-512	256-64-256	32-16-32
Batch Size	50	50	50	50	10
No. of Epochs	1200	1200	600	100	200

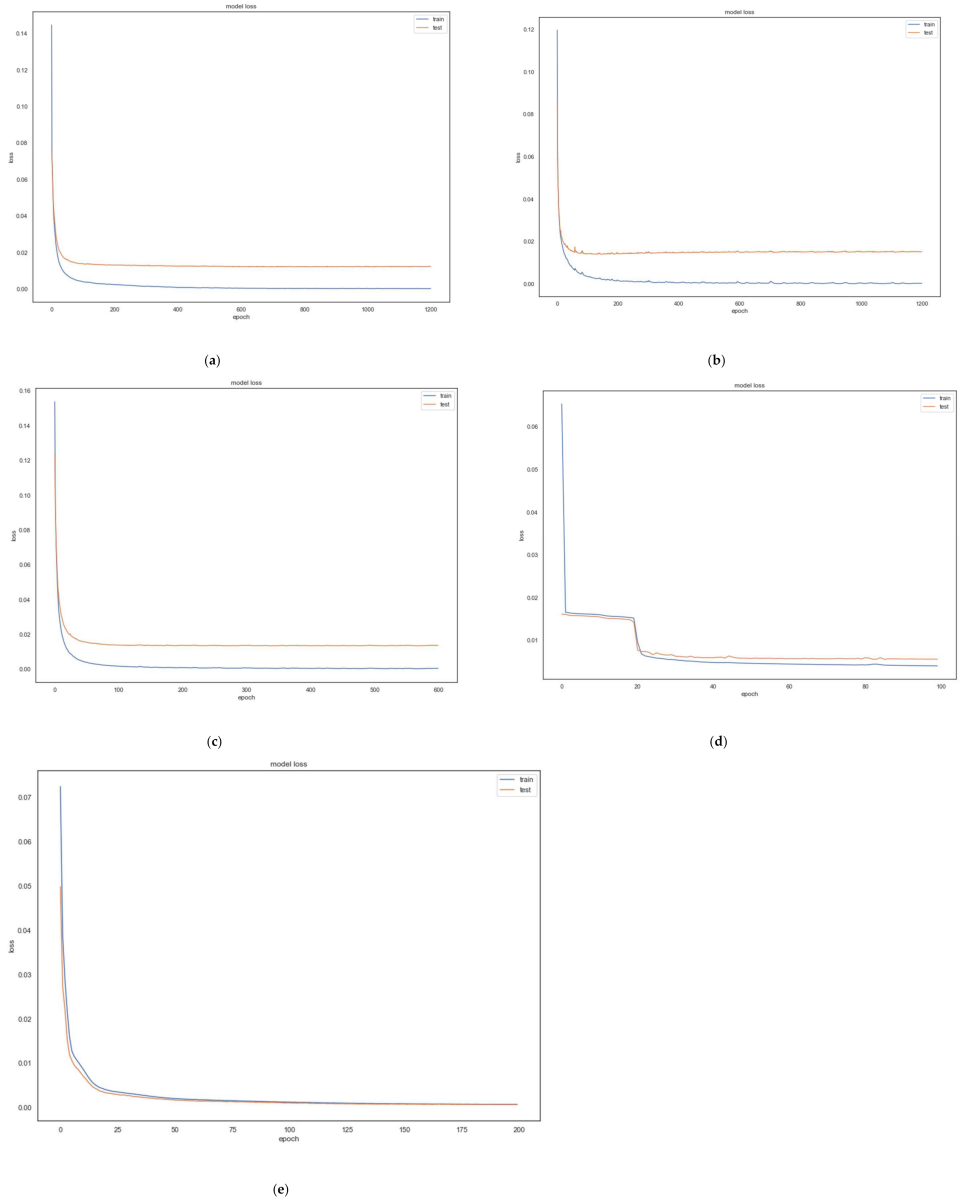


Figure 3. Autoencoder error loss vs. epochs on the five datasets: (**a**) MNIST; (**b**) Fashion-MNIST; (**c**) USPS; (**d**) MSTAR; (**e**) WDBC.

For MLP regressors architecture, we used only one hidden layer of 50 hidden units with ReLU activation function and an output layer of only one output node with linear activation, and we used an adaptive learning rate γ initialized at 0.001. The use of the ReLU activation function stemmed from the advantages of ReLU over sigmoidal functions such as sparsity representation and to avoid the saturation problem of sigmoidal activation functions which cause the vanishing gradient problem [70]. We adopted the same MLP architecture for all MLP regressors in the feature mapping network. We adopted an Adam optimizer for backpropagation of the Mean-Squared Error (MSE) for both the autoencoder and the MLP regressors used in the feature mapping network for the ease of hyperparameter tuning. Figure 3 depicts the MSE loss of the autoencoder for each of the five datasets. For the code and implementation details, please refer to Supplementary Materials.

As per our proposed method, we took the Fashion-MNIST dataset as an example of applying our algorithm. We trained the autoencoder with its corresponding architecture (784-512-256-512-784) on the 600 training images (60 samples per class), then we created

the Cartesian Product set as in Algorithm 1 using the 600 images in two sets Tr_1 and Tr_2 of size $60 \times 59 \times 10 = 35,400$ data samples.

We then extracted features of Tr_1 and Tr_2 using our trained autoencoder resulting in two matrices of features Z_1 and Z_2, both of size $[35,400 \times 256]$. Next, we trained 256 MLP regressors $\{MLP_1, MLP_2, \ldots, MLP_{256}\}$ using Z_1 as input and Z_2 as target variables, where each MLP regressor takes as input the whole feature vector $z_i \in Z_1$ and learns to output one target feature $z_i^s \in Z$, $s = 1, 2, \ldots, 256$, $i = 0, 1, \ldots, 35,400$. To speed up the training process, we distributed the training of the 256 MLP regressors over eight processes that run in parallel such that each process corresponds to 256/8 = 16 MLP regressors.

Afterwards, we used the trained MLP regressors to double the size of the training set D^{Tr} by transforming the 600 original samples, as shown in Algorithm 2, and augmenting the transformed samples $\hat{D^{Tr}}$, so we ended up having a training set $\{D^{Tr}, \hat{D^{Tr}}\}$ of 1200 images. We also transformed the original test set D^{Ts} of 69,400 test images resulting in $\hat{D^{Ts}}$ of 69,400 images as well.

We did the same for all the datasets (each dataset with its AE architecture and hyper-parameters as shown in Table 2) and tested our model in two settings, a semi-supervised clustering setting and a classification setting, explained in the next two subsections.

3.2.1. Semi-Supervised Clustering Setting

In the semi-supervised clustering setting, we measured the performance of clustering original test (unlabeled) samples (D^{Ts}) (without transformation) and clustering transformed test (unlabeled) samples ($\hat{D^{Ts}}$) (using our model trained on the few labeled samples). As a clustering algorithm, we used the K-means algorithm [43] with different centroid initialization methods, K-means++ [71], random, and PCA-based (where we calculated the principal component analysis for each sample in the test set and performed the K-means algorithm on the PCA components). We calculated the clustering evaluation metrics, listed below, for each clustering scenario. The experiment is repeated for each test set of the five datasets and results are presented in Table 3. Besides the clustering evaluation metrics, we calculated the clustering time for each scenario (without and with our model).

Clustering Metrics. We used various clustering metrics that relate to within-cluster and between-cluster measures that related to the compactness and separability properties of the targets. The metrics used and their mathematical formulations are listed below.

- *Inertia* [43]: Within-cluster sum-of-squares criterion.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\|x_i, -, \mu_j\|^2 \right) \tag{6}$$

- *Homogeneity, Completeness, and V-Measure* [72]: Homogeneity measures a score that each cluster contains only members of a single class. Completeness measures a score that all data points of the same class are assigned to the same cluster. V-Measure measures how successfully homogeneity and completeness have been satisfied.

$$V_\beta = \frac{(1+\beta) \times H \times C}{\beta \times H + C} \tag{7}$$

- *Adjusted Rand Index (ARI)* [73]: Measures the similarity of two assignments (ground truth and predicted cluster assignment), ignoring permutations.

$$\mathrm{RI} = \frac{a + b}{C_2^{n_{samples}}} \tag{8}$$

$$ARI = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]} \tag{9}$$

- *Adjusted Mutual Information (AMI)* [74]: Measures the agreement of two assignments (ground truth and predicted cluster assignment), ignoring permutations. Given two cluster assignments U and V, their entropies, mutual information, and adjusted mutual information are defined respectively as:

$$H(U) = -\sum_{i=1}^{|U|} P(i) \log P(i) \tag{10}$$

where $P(i) = |U_i|/N$ is the probability that an object picked from U at random belongs to class U_i. Same for $P'(j)$ and V.

$$H(V) = -\sum_{j=1}^{|V|} P'(j) \log P'(j) \tag{11}$$

$$\mathrm{MI}(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P''(i,j) \log \frac{P''(i,j)}{P(i)P'(j)} \tag{12}$$

where $P''(i,j) = |U_i \cap V_j|/N$ is the joint probability that an object picked at random belongs to both classes U_i and V_j.

$$\mathrm{AMI} = \frac{\mathrm{MI} - E[\mathrm{MI}]}{mean(H(U), H(V)) - E[\mathrm{MI}]} \tag{13}$$

where E is the expected value of the mutual information.

- *Silhouette Coefficient* [75]: The silhouette coefficient for a single data point is defined as follows:

$$S = \frac{b - a}{max(a, b)} \tag{14}$$

where a is the mean distance between the data point and all other points within the same cluster and b is the mean distance between the data point and all other points in the nearest cluster. The mean silhouette coefficient is calculated for all data points, and the higher the score, the better the defined clusters indicating a lower within-cluster distance and a higher between-cluster distance.

As can be seen from Table 3, using our model to map the test set consistently yielded the best results in all clustering performance metrics over all used datasets. An improvement in both the inertia of clusters and the silhouette coefficient indicates that our model achieves its goal of promoting compactness (within-class) and separability (between-class) of data points which subsequently improves all other clustering metrics. Also, using our model to map samples before clustering gives a significant reduction in clustering time of the algorithm, particularly for large datasets (like MNIST and Fashion-MNIST) when applying the K-means++ initialization method.

We also report a comparison to another clustering algorithm (SESSC) [48] on the WDBC dataset. As can be seen from Table 3, their performance on the test set without mapping produces lower values in all calculated clustering metrics relative to using our model on the K-means algorithm with any initialization method. Also, it is noteworthy that applying their algorithm on the mapped test set using our model enhances their performance, which supports our claim that our proposed method is pluggable to any clustering algorithm (hard or soft). We applied their provided code (https://github.com/YuqiCui/SESSC (accessed on 19 January 2022)) on other datasets and followed the guideline for hyperparameter fine tuning mentioned in their paper. However, we were not able to conduct the experiment except for the WDBC dataset. This led to the conclusion that their implementation does not support datasets with relatively high dimensionality.

Table 3. Semi-supervised clustering performance evaluation results. Bolded values indicate the best results achieved in each metric. D^{Ts} is the original unlabeled test set (without transformation) and $\hat{D^{Ts}}$ is the transformed (using our model) unlabeled test set.

Dataset	K-Means Init. Method	Test Set	*Time*	*Inertia*	*H*	*C*	V_β	*ARI*	*AMI*	*S*
MNIST	K-Means++	D^{Ts}	119.802 s	42,141,554	0.421	0.443	0.432	0.322	0.421	0.058
		$\hat{D^{Ts}}$	**32.010 s**	**14,454,470**	**0.658**	**0.680**	**0.669**	**0.586**	**0.658**	**0.396**
	Random	D^{Ts}	62.014 s	42,142,078	0.417	0.439	0.428	0.318	0.417	0.060
		$\hat{D^{Ts}}$	**31.006 s**	**14,454,470**	**0.658**	**0.680**	**0.669**	**0.586**	**0.658**	**0.381**
	PCA-Based	D^{Ts}	14.302 s	42,142,060	0.417	0.439	0.428	0.317	0.417	0.055
		$\hat{D^{Ts}}$	**8.425 s**	**14,454,470**	**0.658**	**0.680**	**0.669**	**0.586**	**0.658**	**0.394**
Fashion-MNIST	K-Means++	D^{Ts}	41.216 s	30,290,316	0.505	0.536	0.520	0.338	0.505	0.151
		$\hat{D^{Ts}}$	**24.270 s**	**7,065,023**	**0.705**	**0.707**	**0.706**	**0.636**	**0.705**	**0.505**
	Random	D^{Ts}	44.563 s	30,243,539	0.493	0.518	0.505	0.348	0.493	0.137
		$\hat{D^{Ts}}$	**38.562 s**	**7,067,789**	**0.693**	**0.712**	**0.702**	**0.607**	**0.693**	**0.510**
	PCA-Based	D^{Ts}	10.170 s	30,820,962	0.488	0.503	0.495	0.344	0.488	0.131
		$\hat{D^{Ts}}$	**8.454 s**	**8,803,305**	**0.675**	**0.690**	**0.682**	**0.587**	**0.675**	**0.479**
USPS	K-Means++	D^{Ts}	1.636 s	1,449,834	0.576	0.583	0.580	0.473	0.575	0.117
		$\hat{D^{Ts}}$	**1.038 s**	**333,792**	**0.847**	**0.848**	**0.848**	**0.863**	**0.847**	**0.615**
	Random	D^{Ts}	1.328 s	1,456,828	0.547	0.569	0.558	0.445	0.546	0.118
		$\hat{D^{Ts}}$	**0.889 s**	**333,788**	**0.847**	**0.847**	**0.847**	**0.863**	**0.847**	**0.571**
	PCA-Based	D^{Ts}	0.478 s	1,449,824	0.577	0.584	0.580	0.474	0.576	0.119
		$\hat{D^{Ts}}$	**0.197 s**	**33,793**	**0.846**	**0.847**	**0.847**	**0.863**	**0.846**	**0.621**
MSTAR	K-Means++	D^{Ts}	3.305 s	9,242,633	0.012	0.016	0.014	0.009	0.009	0.015
		$\hat{D^{Ts}}$	**2.174 s**	**4,280,437**	**0.524**	**0.528**	**0.526**	**0.526**	**0.522**	**0.388**
	Random	D^{Ts}	2.142 s	9,232,361	0.017	0.019	0.018	0.009	0.014	0.024
		$\hat{D^{Ts}}$	**1.589 s**	**4,280,491**	**0.522**	**0.527**	**0.524**	**0.523**	**0.520**	**0.390**
	PCA-Based	D^{Ts}	0.835 s	9,245,153	0.022	0.023	0.023	0.019	0.019	0.019
		$\hat{D^{Ts}}$	**0.590 s**	**4,280,491**	**0.522**	**0.527**	**0.524**	**0.523**	**0.520**	**0.371**
WDBC	K-Means++	D^{Ts}	0.013 s	3454	0.668	0.689	0.678	0.777	0.666	0.391
		$\hat{D^{Ts}}$	**0.010 s**	**958**	**0.904**	**0.904**	**0.904**	**0.953**	**0.904**	**0.873**
	Random	D^{Ts}	0.010 s	3454	0.668	0.689	0.678	0.777	0.666	0.391
		$\hat{D^{Ts}}$	**0.008 s**	**958**	**0.904**	**0.904**	**0.904**	**0.953**	**0.904**	**0.873**
	PCA-Based	D^{Ts}	0.003 s	3454	0.668	0.689	0.678	0.777	0.666	0.391
		$\hat{D^{Ts}}$	**0.002 s**	**958**	**0.904**	**0.904**	**0.904**	**0.953**	**0.904**	**0.873**
	SESSC	D^{Ts}	1.491 s	-	0.871	0.867	0.869	0.930	0.868	0.286
		$\hat{D^{Ts}}$	**1.151 s**	-	**0.954**	**0.950**	**0.952**	**0.977**	**0.952**	**0.711**

3.2.2. Classification Setting

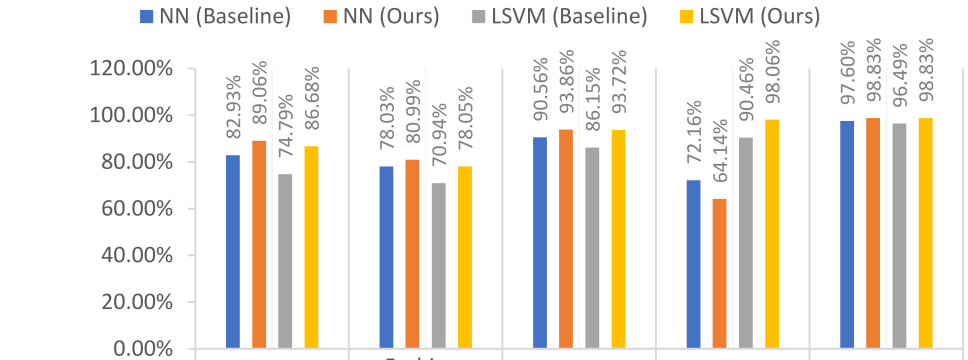
We devised an experiment where we employ our model before training a classifier to map both the training set and the test set. We then trained the classifier on the plain training set (D^{Tr}) augmented with the mapped training set ($\hat{D^{Tr}}$) using our model, and we evaluated the classifier on the mapped test set ($\hat{D^{Ts}}$). We compared our experiment with a baseline experiment where we trained the classifier on the plain training set (D^{Tr}) and evaluated it on the plain test set (D^{Ts}).

We deployed a neural network (NN) and a linear SVM models for the classification task. We used a linear SVM as a classifier because improving accuracy with a linear classifier demonstrates the linear separability enhancement of transformed data points. We measured the accuracy for both classifier models (NN and LSVM) and the loss for the NN model. Experiments were repeated 10 times independently and average accuracy and average loss was calculated over the 10 runs for each experiment and compare the obtained results of the baseline experiment against ours. Both experiments with both

classifiers were run for the five datasets and results are listed in Table 4 and illustrated in Figure 4. From the results, one can observe that the accuracy of both the neural network and the linear SVM classifiers are enhanced in nearly all the datasets. Also, we can see that the LSVM classifier's accuracies with our model are on par with the NN classifier which indicates that the linear separability is enhanced because of our model. We can observe that the SVM classifier outperforms the NN classifier on the MSTAR dataset in both settings, with and without our model. A plausible explanation for why the NN classifier performs poorly on such a dataset is that NN classifiers suffer when the input data dimensionality is high. This drawback of NN classifiers is relatively remedied when the number of training samples is adequately large relative to the number of parameters preventing the plausible overfitting [76]. This is unfortunately not available in our case of small datasets, especially the MSTAR dataset. On the other hand, SVM training requires a much smaller number of data samples as they rely on the support vectors which is a very small subset of the training samples. We also conducted experiments on classifiers used in [48] and added their results on the WDBC dataset in Table 4. The used classifiers were SESSC as a standalone classifier, ESSC_LSE, and SESSC_LSE (the ESSC and the SESSC clustering algorithms integrated with a fuzzy classifier, respectively). We can notice that again using our model to transform the test set improves their performance as other classifiers, which further proves the pluggable feature of our model.

Table 4. Classification results. D^{Tr} is the original training set (without transformation) and $\{D^{Tr}, \hat{D^{Tr}}\}$ is the original training D^{Tr} set augmented with the transformed (using our model) training set $\hat{D^{Tr}}$. Bolded values indicate the best results achieved in each metric.

Dataset	Classifier	Train Set	Test Set	Average Accuracy	Average Loss
MNIST	NN	D^{Tr}	D^{Ts}	0.829278	0.588378
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.890600**	**0.423493**
	LSVM	D^{Tr}	D^{Ts}	0.747853	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.866773**	-
Fashion-MNIST	NN	D^{Tr}	D^{Ts}	0.780324	0.652228
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.809889**	**0.648123**
	LSVM	D^{Tr}	D^{Ts}	0.709376	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.780488**	-
USPS	NN	D^{Tr}	D^{Ts}	0.905552	0.328199
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.938645**	**0.245099**
	LSVM	D^{Tr}	D^{Ts}	0.861530	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.937175**	-
MSTAR	NN	D^{Tr}	D^{Ts}	**0.721635**	**0.528935**
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	0.641397	0.620295
	LSVM	D^{Tr}	D^{Ts}	0.904600	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.980579**	-
WDBC	NN	D^{Tr}	D^{Ts}	0.976023	0.101761
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.988304**	**0.034921**
	LSVM	D^{Tr}	D^{Ts}	0.964912	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.988304**	-
	ESSC_LSE	D^{Tr}	D^{Ts}	0.949708	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.985965**	-
	SESSC	D^{Tr}	D^{Ts}	0.954386	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.992398**	-
	SESSC_LSE	D^{Tr}	D^{Ts}	0.971930	-
		$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.990058**	-



	MNIST	Fashion-MNIST	USPS	MSTAR	WDBC
NN (Baseline)	82.93%	78.03%	90.56%	72.16%	97.60%
NN (Ours)	89.06%	80.99%	93.86%	64.14%	98.83%
LSVM (Baseline)	74.79%	70.94%	86.15%	90.46%	96.49%
LSVM (Ours)	86.68%	78.05%	93.72%	98.06%	98.83%

Figure 4. Classification accuracies on the five datasets.

In addition to the proposed classification setting, we undertook a set of pilot experiments on a limited number of the used datasets. In Set-1 of the pilot experiments, we trained the classifier on only D^{Tr} (without augmenting $\hat{D^{Tr}}$) and tested its performance on the transformed test set $\hat{D^{Ts}}$. Although our pilot experiment improved the classifier performance in most of the datasets, we found that training the classifier with the augmented set consistently gave better results over all datasets since augmentation increased the size of the training set which reduced the classifier overfitting. Also, adding the transformed samples to the training pool helps the classifier to achieve better decision boundaries relative to the transformed test samples $\hat{D^{Ts}}$ as it was trained on the transformed training samples before, which makes it achieve better results. A summary of the pilot experiment results is presented in Table 5 and Figure 5.

Table 5. Set-1 pilot experiment results. D^{Tr} is the original training set (without transformation), D^{Ts} is the original unlabeled test set (without transformation), and $\hat{D^{Ts}}$ is the transformed (using our model) unlabeled test set. Bolded values indicate the best results achieved in each metric.

Dataset	Classifier	Train Set	Test Set	Average Accuracy	Average Loss
MNIST	NN	D^{Tr}	D^{Ts}	0.829278	0.588378
		D^{Tr}	$\hat{D^{Ts}}$	**0.884564**	**0.409574**
	LSVM	D^{Tr}	D^{Ts}	0.747853	-
		D^{Tr}	$\hat{D^{Ts}}$	**0.856637**	-
Fashion-MNIST	NN	D^{Tr}	D^{Ts}	0.780324	0.652228
		D^{Tr}	$\hat{D^{Ts}}$	**0.804369**	**0.593578**
	LSVM	D^{Tr}	D^{Ts}	**0.709376**	-
		D^{Tr}	$\hat{D^{Ts}}$	0.676829	-

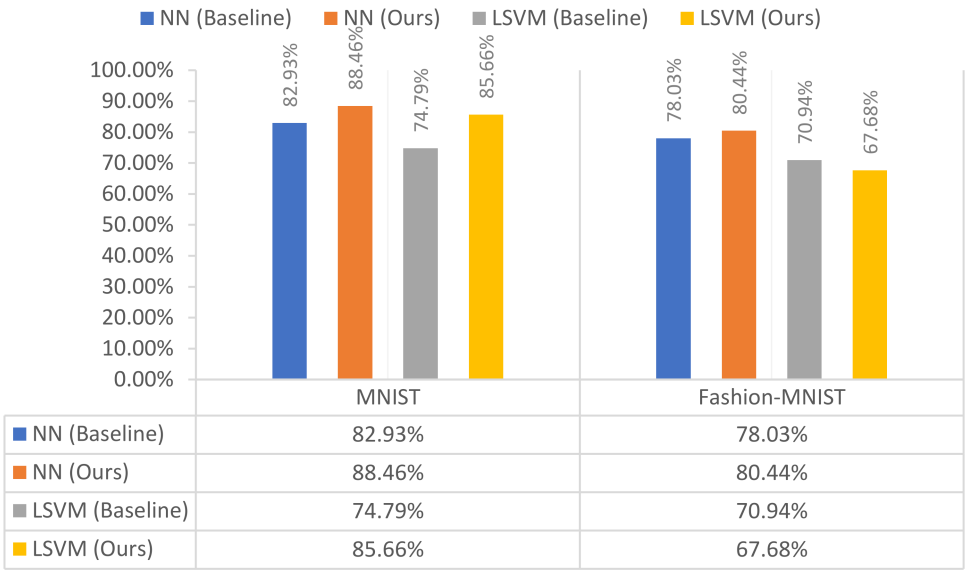


Figure 5. Set-1 pilot experiment accuracies on the MNIST and Fashion-MNIST datasets.

In Set-2 of the pilot experiments, we compared two options. Option 1 was using the reconstructions of the third module as input to the classifier versus the original input data (as demonstrated before). Option 2 was proceeding to the applied classifier with the output of the second module, which were the latent representations obtained from the regressors, versus using the latent representations obtained directly from the autoencoder. Results for semi-supervised clustering and classification are shown in Tables 6 and 7, respectively. Bolded values indicate the best results obtained in each metric except for Inertia where bolded value indicate the best results obtained for each option separately as Inertia is not a normalized metric. The results show that Option 1 is slightly better than Option 2 in most cases in both semi-supervised and supervised settings. In addition, the reconstruction option has the added privilege of not interfering with any subsequent machine learning task. Because the output is of the same dimensionality and nature of the input data, (i.e., no changes are needed for the baseline classification and clustering model setups).

Table 6. Set-2 pilot experiment semi-supervised clustering setting results. D^{Ts} is the original unlabeled test set (without transformation), $\hat{D^{Ts}}$ is the transformed (using our model) unlabeled test set. Z_{ts} and $\hat{Z_{ts}}$ are the features extracted from the autoencoder of the original test set D^{Ts} and the transformed test set $\hat{D^{Ts}}$, respectively. Bolded values indicate the best results achieved in each metric.

Dataset	K-Means Init. Method	Option	Test Set	*Time*	*Inertia*	*H*	*C*	V_β	*ARI*	*AMI*	*S*
MNIST	K-Means++	Option1	D^{Ts}	119.802 s	42,141,554	0.421	0.443	0.432	0.322	0.421	0.058
			$\hat{D^{Ts}}$	32.010 s	**14,454,470**	0.658	**0.680**	0.669	**0.586**	0.658	**0.396**
		Option2	Z_{ts}	19.931 s	14,528,156	0.537	0.544	0.541	0.418	0.537	0.063
			$\hat{Z_{ts}}$	**15.100 s**	**6,730,338**	**0.666**	0.676	**0.671**	0.570	**0.666**	0.329
	Random	Option1	D^{Ts}	62.014 s	42,142,078	0.417	0.439	0.428	0.318	0.417	0.060
			$\hat{D^{Ts}}$	31.006 s	**14,454,470**	0.658	**0.680**	0.669	**0.586**	0.658	**0.381**
		Option2	Z_{ts}	18.312 s	14,510,545	0.526	0.535	0.531	0.397	0.526	0.066
			$\hat{Z_{ts}}$	**11.168 s**	**6,730,342**	**0.666**	0.676	**0.671**	0.570	**0.666**	0.329
	PCA-Based	Option1	D^{Ts}	14.302 s	42,142,060	0.417	0.439	0.428	0.317	0.417	0.055
			$\hat{D^{Ts}}$	8.425 s	**14,454,470**	**0.658**	**0.680**	0.669	**0.586**	0.658	**0.394**
		Option2	Z_{ts}	4.908 s	14,510,560	0.526	0.535	0.531	0.397	0.526	0.062
			$\hat{Z_{ts}}$	**3.618 s**	**6,847,240**	0.655	0.664	0.659	0.567	0.655	0.262

Table 6. *Cont.*

Dataset	K-Means Init. Method	Option	Test Set	*Time*	*Inertia*	*H*	*C*	V_β	*ARI*	*AMI*	*S*
Fashion-MNIST	K-Means++	Option1	D^{Ts}	41.216 s	30,290,316	0.505	0.536	0.520	0.338	0.505	0.151
			$\hat{D^{Ts}}$	24.270 s	**7,065,023**	**0.705**	**0.707**	**0.706**	**0.636**	**0.705**	**0.505**
		Option2	Z_{ts}	14.065 s	12,138,071	0.506	0.523	0.515	0.315	0.506	0.103
			$\hat{Z_{ts}}$	**9.982 s**	**4,670,369**	0.674	0.677	0.676	0.588	0.674	0.325
	Random	Option1	D^{Ts}	44.563 s	30,243,539	0.493	0.518	0.505	0.348	0.493	0.137
			$\hat{D^{Ts}}$	38.562 s	**7,067,789**	**0.693**	**0.712**	**0.702**	**0.607**	**0.693**	**0.510**
		Option2	Z_{ts}	18.293 s	12,179,589	0.510	0.524	0.517	0.331	0.510	0.084
			$\hat{Z_{ts}}$	**10.308 s**	**4,670,369**	0.674	0.677	0.676	0.588	0.674	0.355
	PCA-Based	Option1	D^{Ts}	10.170 s	30,820,962	0.488	0.503	0.495	0.344	0.488	0.131
			$\hat{D^{Ts}}$	8.454 s	**8,803,305**	**0.675**	**0.690**	**0.682**	**0.587**	**0.675**	**0.479**
		Option2	Z_{ts}	3.447 s	12,274,965	0.514	0.530	0.522	0.361	0.514	0.100
			$\hat{Z_{ts}}$	**2.250 s**	**5,529,373**	0.644	0.663	0.653	0.531	0.643	0.329

Table 7. Set-2 pilot experiment classification setting results. Bolded values indicate the comparison result between the two options. D^{Tr} is the original training set (without transformation) and $\{D^{Tr}, \hat{D^{Tr}}\}$ is the original training D^{Tr} set augmented with the transformed (using our model) training set $\hat{D^{Tr}}$. $\{Z_{tr}, \hat{Z_{tr}}\}$ is the features extracted from the autoencoder of the original training set D^{Tr} augmented with the extracted features of the transformed training set $\hat{D^{Tr}}$, respectively.

Dataset	Classifier	Option	Train Set	Test Set	Average Accuracy	Average Loss
MNIST	NN	Option1	D^{Tr}	D^{Ts}	0.829278	0.588378
		Option2	Z_{tr}	Z_{ts}	0.817143	0.728606
		Option1	$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.890600**	**0.423493**
		Option2	$\{Z_{tr}, \hat{Z_{tr}}\}$	$\hat{Z_{ts}}$	0.881613	0.415675
	LSVM	Option1	D^{Tr}	D^{Ts}	0.747853	-
		Option2	Z_{tr}	Z_{ts}	0.817158	-
		Option1	$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.866773**	-
		Option2	$\{Z_{tr}, \hat{Z_{tr}}\}$	$\hat{Z_{ts}}$	0.736704	-
Fashion-MNIST	NN	Option1	D^{Tr}	D^{Ts}	0.780324	0.652228
		Option2	Z_{tr}	Z_{ts}	0.782837	0.770872
		Option1	$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	**0.809889**	**0.648123**
		Option2	$\{Z_{tr}, \hat{Z_{tr}}\}$	$\hat{Z_{ts}}$	0.808163	0.728820
	LSVM	Option1	D^{Tr}	D^{Ts}	0.709376	-
		Option2	Z_{tr}	Z_{ts}	0.763546	-
		Option1	$\{D^{Tr}, \hat{D^{Tr}}\}$	$\hat{D^{Ts}}$	0.780488	-
		Option2	$\{Z_{tr}, \hat{Z_{tr}}\}$	$\hat{Z_{ts}}$	**0.800102**	-

In addition, to further prove the effectiveness of our proposed method, we have provided a comparison with different conventional methods that work on the improvement of classification accuracy of the two naturally small datasets used in this research (WDBC and MSTAR). First, we compared the WDBC dataset to other existing methods that improve the classification accuracy over the dataset using different techniques, mostly feature selection. The WDBC dataset was first normalized and then split into 70–30% training and testing sets. The comparison results are shown in Table 8. As can be seen from the table, our proposed method achieves the second highest accuracy which exhibits the model's effectiveness on this dataset.

Table 8. The WDBC dataset comparison results of the proposed method with other methods. The results are sorted in descending order. Bolded values are the results obtained from this study.

Method	Classification Accuracy
Rahman et al. [77]	99.40%
Proposed Method + SESSC	**99.23%**
Proposed Method + LSVM	**98.83%**
Yoon et al. [78]	98.80%
Zhang et al. [79]	98.36%
Murugan et al. [80]	98.19%
Nekkaa et al. [81]	97.88%
Aalaei et al. [82]	97.30%
Mafarja et al. [83]	97.10%
Azhagusundari et al. [84]	73.78%

We also compared our MSTAR three-target classification accuracy result (98.06%) with other research done on classification accuracy enhancement on the same dataset. We compared with some conventional methods that do not employ data augmentation such as [85] that uses a joint sparse representation (JRS) model, [86] which makes use of phase and amplitude as well as image data, a sparse representation classifier, and Riemannian manifolds, Euclidean distance restricted autoencoder [87], and a hierarchical recognition system that is based on a constrained restricted Boltzmann machine (RBM) [88]. In addition, we included a method that uses data augmentation (deep learning method based on visual cortical system [89]). The comparison results presented in Table 9 show that the LSVM using our model achieves a comparable accuracy to other research results; also, it is worth noting the effect of data augmentation on classification accuracy such as in [89] where using data augmentation enhances the classification accuracy by 4.6%.

Table 9. The MSTAR three-target dataset comparison results of the proposed method with other methods. The results are sorted in descending order. Bolded value is the result of this study.

Method	Classification Accuracy
Proposed Method + LSVM	**98.06%**
Ni et al. [89] (With Data Augmentation)	96.70%
Dong et al. [86]	96.10%
Cui et al. [88]	95.31%
Deng et al. [87]	94.14%
Zhang et al. [85]	93.20%
Ni et al. [89] (Without Data Augmentation)	92.10%

Besides the quantitative results that demonstrate our model's ability to improve the compactness and separability properties, we show qualitative results through visualizing the mapped features in the feature space using t-SNE [90]. Figure 6 depicts the t-SNE of the plain test set vs. the mapped test set for the five datasets. One can observe that the classes are more separable and compact, especially overlapped classes as, for example, shown by the digits 4 and 9 in the MNIST dataset (highlighted in red circles).

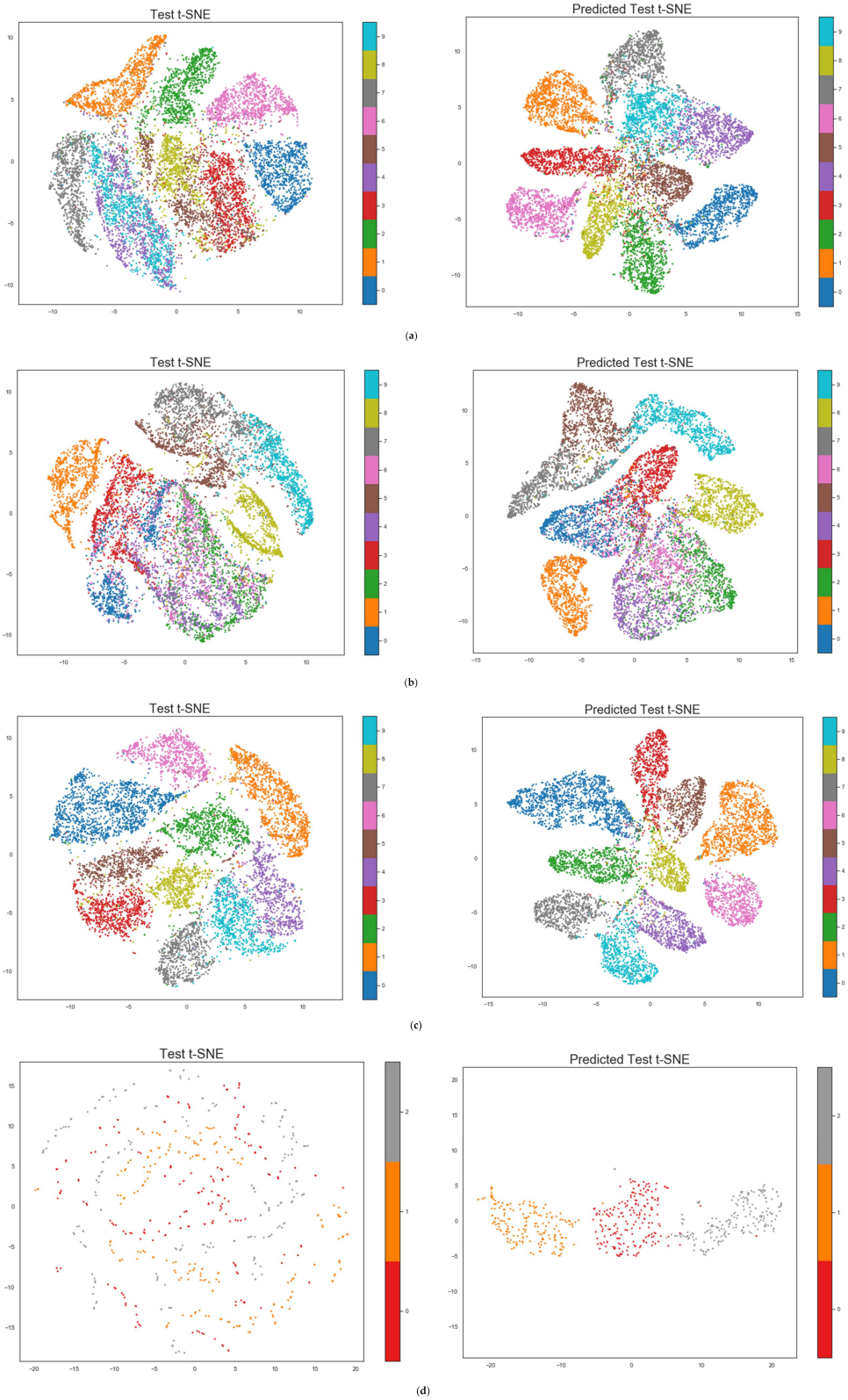


Figure 6. *Cont.*

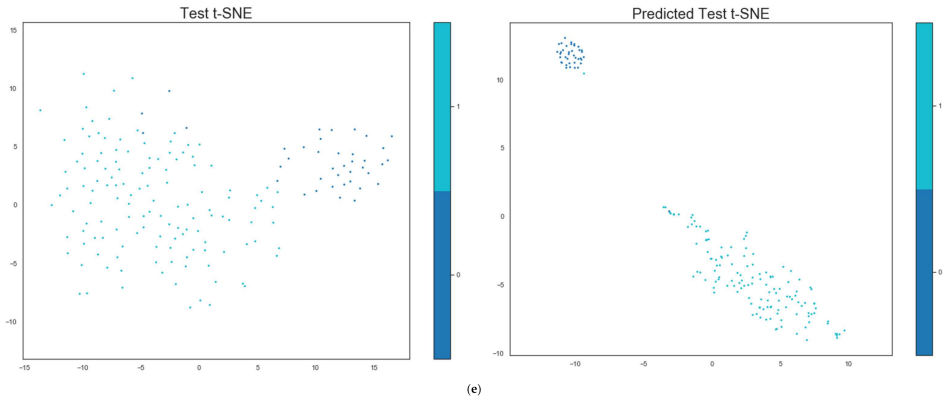


Figure 6. TSNE visualizations of the test set without mapping (left) vs. mapped with our model (right): (**a**) MNIST; (**b**) Fashion-MNIST; (**c**) USPS; (**d**) MSTAR; (**e**) WDBC.

4. Conclusions

This paper proposes a framework that has the potential to improve the compactness and separability properties of data points, which consistently boosts the performance of any off-the-shelf classifier, or a semi-supervised clustering model. We have utilized the generative capacity of autoencoders to reconstruct data points, thus preserving the original input feature space. This characteristic of our proposed framework makes it easily pluggable before a classifier model (not limited by a specific classifier type) or semi-supervised clustering model. We have also utilized an array of MLP regressors for mapping the latent representation of the input samples to approximations of the centroids of the classes. We conducted experiments on five different datasets (MNIST, Fashion-MNIST, USPS, MSTAR, and WDBC) to demonstrate the results of our proposed method. We concluded that plugging our model as a preprocessing step gives better results in all the datasets in both classification and semi-supervised clustering tasks. We also compared the proposed method to other existing methods on enhancing the classification accuracy over the WDBC and MSTAR datasets. We illustrated the importance of augmenting small datasets with the produced approximation of the centroids of the classes and how it further improves the classifier results.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/app12031713/s1.

Author Contributions: Conceptualization, M.M.E. and R.F.; methodology, M.M.E. and R.F.; software, M.M.E.; validation, M.M.E., R.F. and Y.E.-S.; formal analysis, R.F. and Y.E.-S.; investigation, M.M.E.; resources, M.M.E. and R.F.; data curation, M.M.E.; writing—original draft preparation, M.M.E.; writing—review and editing, R.F. and Y.E.-S.; visualization, M.M.E.; supervision, R.F and Y.E.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this study are available online. MNIST: http://yann.lecun.com/exdb/mnist/ (accessed on 10 December 2021); Fashion-MNIST: https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion (accessed on 10 December 2021); USPS: https://www.kaggle.com/bistaumanga/usps-dataset (accessed on 10 December 2021); MSTAR: https://www.sdms.afrl.af.mil/index.php?collection=mstar&page=targets (accessed on 10 December 2021); WDBC: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) (accessed on 10 December 2021).

Acknowledgments: The authors would like to thank all anonymous reviewers and editors for their helpful suggestions for the improvement of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Proof that the Mean-Square Error across a dataset D has a lower bound value of $\mathrm{Var}(Y)$:

Let Y be the true output and $\hat{Y}$ the predicted output of the network, then the expected value $\mathbb{E}_D$ of the mean-squared error over the entire dataset D is $\mathbb{E}_D\left[\left(Y-\hat{Y}\right)^2\right]$.

With μ denoting the mean of Y, we can express $\mathbb{E}_D\left[\left(Y-\hat{Y}\right)^2\right]$ as:

$$\begin{aligned}\mathbb{E}_D\left[\left(Y-\hat{Y}\right)^2\right] &= \mathbb{E}_D\left[\left(Y-\mu+\mu-\hat{Y}\right)^2\right] \\ &= \mathbb{E}_D\left[\left((Y-\mu)+\left(\mu-\hat{Y}\right)\right)^2\right] \\ &= \mathbb{E}_D\left[\left((Y-\mu)^2+\left(\mu-\hat{Y}\right)^2+2(Y-\mu)\left(\mu-\hat{Y}\right)\right)\right]\end{aligned} \tag{A1}$$

Since expectation of the sum is the sum of expectations, and constants can be pulled out of expectations, the expression can be written as follows:

$$\begin{aligned}&\mathbb{E}_D\left[\left((Y-\mu)^2+\left(\mu-\hat{Y}\right)^2+2(Y-\mu)\left(\mu-\hat{Y}\right)\right)\right] \\ &\quad= \mathbb{E}_D\left[(Y-\mu)^2\right]+\mathbb{E}_D\left[\left(\mu-\hat{Y}\right)^2\right]+2\left(\mu-\hat{Y}\right)\mathbb{E}_D[(Y-\mu)]\end{aligned} \tag{A2}$$

Given the definition of variance and that the expectation of a constant equals the constant, the expression is written as follows:

$$\begin{aligned}&\mathbb{E}_D\left[(Y-\mu)^2\right]+\mathbb{E}_D\left[\left(\mu-\hat{Y}\right)^2\right]+2\left(\mu-\hat{Y}\right)\mathbb{E}_D[(Y-\mu)] \\ &\quad= Var(Y)+\left(\mu-\hat{Y}\right)^2+2\left(\mu-\hat{Y}\right)\mathbb{E}_D[(Y-\mu)]\end{aligned} \tag{A3}$$

The term $\mathbb{E}_D[(Y-\mu)]$ can be evaluated as follows:

$$\mathbb{E}_D[(Y-\mu)] = \mathbb{E}_D[Y]-\mathbb{E}_D[\mu] = \mu-\mu = 0 \tag{A4}$$

which gives,

$$\begin{aligned}Var(Y)+\left(\mu-\hat{Y}\right)^2+2\left(\mu-\hat{Y}\right)\mathbb{E}_D[(Y-\mu)] &= Var(Y)+\left(\mu-\hat{Y}\right)^2+2\left(\mu-\hat{Y}\right)\times 0 \\ &= Var(Y)+\mu-\hat{Y}\end{aligned} \tag{A5}$$

Thus,

$$\mathbb{E}_D\left[\left(Y-\hat{Y}\right)^2\right] \geq Var(Y) \tag{A6}$$

with equality holding when $\hat{Y}=\mu$.

References

1. Storcheus, D.; Rostamizadeh, A.; Kumar, S. A Survey of Modern Questions and Challenges in Feature Extraction. In Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015, PMLR, Montreal, QC, Canada, 8 December 2015; pp. 1–18.
2. Wang, M.; Lin, L.; Wang, F. Improving Short Text Classification through Better Feature Space Selection. In Proceedings of the 2013 Ninth International Conference on Computational Intelligence and Security, Washington, DC, USA, 14–15 December 2013; pp. 120–124.
3. Nargesian, F.; Samulowitz, H.; Khurana, U.; Khalil, E.B.; Turaga, D. Learning Feature Engineering for Classification. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 2529–2535.
4. Zhang, P.; Shi, B.; Smith, C.D.; Liu, J. Nonlinear Feature Space Transformation to Improve the Prediction of MCI to AD Conversion. In Proceedings of the Medical Image Computing and Computer Assisted Intervention—MICCAI 2017, Quebec

City, QC, Canada, 11–13 September 2017; Descoteaux, M., Maier-Hein, L., Franz, A., Jannin, P., Collins, D.L., Duchesne, S., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 12–20.
5. Guo, X.; Zhu, E.; Liu, X.; Yin, J. Deep Embedded Clustering with Data Augmentation. In Proceedings of the 10th Asian Conference on Machine Learning, PMLR, Beijing, China, 4 November 2018; pp. 550–565.
6. Guo, X.; Liu, X.; Zhu, E.; Yin, J. Deep Clustering with Convolutional Autoencoders. In Proceedings of the International Conference on Neural Information Processing, Guangzhou, China, 14–18 November 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 373–382.
7. Em, Y.; Gag, F.; Lou, Y.; Wang, S.; Huang, T.; Duan, L.-Y. Incorporating Intra-Class Variance to Fine-Grained Visual Recognition. In Proceedings of the 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, China, 10–14 July 2017; pp. 1452–1457.
8. Passalis, N.; Iosifidis, A.; Gabbouj, M.; Tefas, A. Variance-Preserving Deep Metric Learning for Content-Based Image Retrieval. *Pattern Recognit. Lett.* **2020**, *131*, 8–14. [CrossRef]
9. Wu, G.; Han, J.; Guo, Y.; Liu, L.; Ding, G.; Ni, Q.; Shao, L. Unsupervised Deep Video Hashing via Balanced Code for Large-Scale Video Retrieval. *IEEE Trans. Image Process.* **2018**, *28*, 1993–2007. [CrossRef] [PubMed]
10. Gysel, C.V.; De Rijke, M.; Kanoulas, E. Neural Vector Spaces for Unsupervised Information Retrieval. *ACM Trans. Inf. Syst. TOIS* **2018**, *36*, 1–25. [CrossRef]
11. Yu, J.; Lu, Y.; Qin, Z.; Zhang, W.; Liu, Y.; Tan, J.; Guo, L. Modeling Text with Graph Convolutional Network for Cross-Modal Information Retrieval. In Proceedings of the Pacific Rim Conference on Multimedia, Hefei, China, 21–22 September 2018; Springer: Cham, Switzerland, 2018; pp. 223–234.
12. Jean, N.; Xie, S.M.; Ermon, S. Semi-Supervised Deep Kernel Learning: Regression with Unlabeled Data by Minimizing Predictive Variance. *arXiv* **2018**, arXiv:1805.10407.
13. Zhu, C.; Idemudia, C.U.; Feng, W. Improved Logistic Regression Model for Diabetes Prediction by Integrating PCA and K-Means Techniques. *Inform. Med. Unlocked* **2019**, *17*, 100179. [CrossRef]
14. Mathew, J.; Pang, C.K.; Luo, M.; Leong, W.H. Classification of Imbalanced Data by Oversampling in Kernel Space of Support Vector Machines. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *29*, 4065–4076. [CrossRef]
15. Becker, M.; Lippel, J.; Stuhlsatz, A.; Zielke, T. Robust Dimensionality Reduction for Data Visualization with Deep Neural Networks. *Graph. Models* **2020**, *108*, 101060. [CrossRef]
16. Zhou, L.; Wang, Z.; Luo, Y.; Xiong, Z. Separability and Compactness Network for Image Recognition and Superresolution. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 3275–3286. [CrossRef]
17. Shorten, C.; Khoshgoftaar, T.M. A Survey on Image Data Augmentation for Deep Learning. *J. Big Data* **2019**, *6*, 60. [CrossRef]
18. Sun, J.; Du, W.; Shi, N. A Survey of KNN Algorithm. *Inf. Eng. Appl. Comput.* **2018**, *1*, 770. [CrossRef]
19. Guo, G.; Wang, H.; Bell, D.; Bi, Y.; Greer, K. KNN Model-Based Approach in Classification. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Catania, Italy, 3–7 November 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 986–996.
20. Abu Alfeilat, H.A.; Hassanat, A.B.; Lasassmeh, O.; Tarawneh, A.S.; Alhasanat, M.B.; Eyal Salman, H.S.; Prasath, V.S. Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. *Big Data* **2019**, *7*, 221–248. [CrossRef] [PubMed]
21. Ertuğrul, Ö.F. A Novel Distance Metric Based on Differential Evolution. *Arab. J. Sci. Eng.* **2019**, *44*, 9641–9651. [CrossRef]
22. Jiao, L.; Geng, X.; Pan, Q. BP *k* NN: *k*-Nearest Neighbor Classifier with Pairwise Distance Metrics and Belief Function Theory. *IEEE Access* **2019**, *7*, 48935–48947. [CrossRef]
23. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-Sampling Technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
24. Greenwald, N.F.; Miller, G.; Moen, E.; Kong, A.; Kagel, A.; Fullaway, C.C.; McIntosh, B.J.; Leow, K.; Schwartz, M.S.; Dougherty, T. Whole-Cell Segmentation of Tissue Images with Human-Level Performance Using Large-Scale Data Annotation and Deep Learning. *bioRxiv* **2021**. [CrossRef] [PubMed]
25. Zhang, X.; Luo, H.; Fan, X.; Xiang, W.; Sun, Y.; Xiao, Q.; Jiang, W.; Zhang, C.; Sun, J. Alignedreid: Surpassing Human-Level Performance in Person Re-Identification. *arXiv* **2017**, arXiv:1711.08184.
26. Zhuang, J.; Hou, S.; Wang, Z.; Zha, Z.-J. Towards Human-Level License Plate Recognition. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018 ; pp. 306–321.
27. Munasinghe, S.; Fookes, C.; Sridharan, S. Human-Level Face Verification with Intra-Personal Factor Analysis and Deep Face Representation. *IET Biometr.* **2018**, *7*, 467–473. [CrossRef]
28. Matek, C.; Schwarz, S.; Spiekermann, K.; Marr, C. Human-Level Recognition of Blast Cells in Acute Myeloid Leukaemia with Convolutional Neural Networks. *Nat. Mach. Intell.* **2019**, *1*, 538–544. [CrossRef]
29. Zhao, W. Research on the Deep Learning of the Small Sample Data Based on Transfer Learning. In Proceedings of the AIP Conference Proceedings, Chongqing City, China, 27–28 May 2017; AIP Publishing LLC: New York, NY, USA, 2017; Volume 1864, p. 020018.
30. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
31. Cao, Y.; Geddes, T.A.; Yang, J.Y.H.; Yang, P. Ensemble Deep Learning in Bioinformatics. *Nat. Mach. Intell.* **2020**, *2*, 500–508. [CrossRef]

32. Mishra, S.; Yamasaki, T.; Imaizumi, H. Improving Image Classifiers for Small Datasets by Learning Rate Adaptations. In Proceedings of the 2019 16th International Conference on Machine Vision Applications (MVA), Tokyo, Japan, 27–31 May 2019; pp. 1–6.
33. Li, X.; Yu, L.; Chang, D.; Ma, Z.; Cao, J. Dual Cross-Entropy Loss for Small-Sample Fine-Grained Vehicle Classification. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4204–4212. [CrossRef]
34. Lohit, S.; Wang, Q.; Turaga, P. Temporal Transformer Networks: Joint Learning of Invariant and Discriminative Time Warping. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 12426–12435.
35. Bradley, P.S.; Fayyad, U.M. Refining Initial Points for K-Means Clustering. In Proceedings of the ICML, Madison, WI, USA, 24–27 July 1998; Citeseer; Morgan Kaufmann: San Francisco, CA, USA, 1998; Volume 98, pp. 91–99.
36. Pilarczyk, R.; Skarbek, W. On Intra-Class Variance for Deep Learning of Classifiers. *arXiv* **2019**, arXiv:1901.11186. [CrossRef]
37. Luo, Y.; Wong, Y.; Kankanhalli, M.; Zhao, Q. $\mathcal{G}$-Softmax: Improving Intraclass Compactness and Interclass Separability of Features. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 685–699. [CrossRef]
38. Liu, W.; Wen, Y.; Yu, Z.; Yang, M. Large-Margin Softmax Loss for Convolutional Neural Networks. In Proceedings of the ICML, New York, NY, USA, 20–22 June 2016; Volume 2, p. 7.
39. Li, C.; Liu, Z.; Ren, J.; Wang, W.; Xu, J. A Feature Optimization Approach Based on Inter-Class and Intra-Class Distance for Ship Type Classification. *Sensors* **2020**, *20*, 5429. [CrossRef] [PubMed]
40. Zeng, H.-J.; Wang, X.-H.; Chen, Z.; Lu, H.; Ma, W.-Y. CBC: Clustering Based Text Classification Requiring Minimal Labeled Data. In Proceedings of the Third IEEE International Conference on Data Mining, Melbourne, FL, USA, 22 November 2003; pp. 443–450.
41. Shukla, A.; Cheema, G.S.; Anand, S. Semi-Supervised Clustering with Neural Networks. In Proceedings of the 2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM), New Delhi, India, 24–26 September 2020; pp. 152–161.
42. Zheng, A.; Casari, A. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2018; ISBN 978-1-4919-5319-8.
43. Lloyd, S. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [CrossRef]
44. Chan, E.; Ching, W.-K.; Ng, M.; Huang, J. An Optimization Algorithm for Clustering Using Weighted Dissimilarity Measures. *Pattern Recognit.* **2004**, *37*, 943–952. [CrossRef]
45. Huang, J.Z.; Ng, M.K.; Rong, H.; Li, Z. Automated Variable Weighting in K-Means Type Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.* **2005**, *27*, 657–668. [CrossRef] [PubMed]
46. Huang, X.; Ye, Y.; Zhang, H. Extensions of Kmeans-Type Algorithms: A New Clustering Framework by Integrating Intracluster Compactness and Intercluster Separation. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 1433–1446. [CrossRef] [PubMed]
47. Deng, Z.; Choi, K.-S.; Chung, F.-L.; Wang, S. Enhanced Soft Subspace Clustering Integrating Within-Cluster and between-Cluster Information. *Pattern Recognit.* **2010**, *43*, 767–781. [CrossRef]
48. Cui, Y.; Wang, H.; Wu, D. Supervised Enhanced Soft Subspace Clustering (SESSC) for TSK Fuzzy Classifiers. *arXiv* **2020**, arXiv:2002.12404.
49. Bank, D.; Koenigstein, N.; Giryes, R. Autoencoders. *arXiv* **2021**, arXiv:2003.05991.
50. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. *Learning Internal Representations by Error Propagation*; California University, San Diego La Jolla Institute for Cognitive Science: San Diego, CA, USA, 1985.
51. Meng, Q.; Catchpoole, D.; Skillicom, D.; Kennedy, P.J. Relational Autoencoder for Feature Extraction | IEEE Conference Publication | IEEE Xplore. Available online: https://ieeexplore.ieee.org/abstract/document/7965877/ (accessed on 2 December 2021).
52. Ryu, S.; Choi, H.; Lee, H.; Kim, H. Convolutional Autoencoder Based Feature Extraction and Clustering for Customer Load Analysis. *IEEE Trans. Power Syst.* **2020**, *35*, 1048–1060. [CrossRef]
53. Liu, Y.; Xie, D.; Gao, Q.; Han, J.; Wang, S.; Gao, X. Graph and Autoencoder Based Feature Extraction for Zero-Shot Learning. In Proceedings of the Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; International Joint Conferences on Artificial Intelligence Organization: Macao, China, 2019; pp. 3038–3044.
54. Luo, X.; Li, X.; Wang, Z.; Liang, J. Discriminant Autoencoder for Feature Extraction in Fault Diagnosis. *Chemom. Intell. Lab. Syst.* **2019**, *192*, 103814. [CrossRef]
55. Polic, M.; Krajacic, I.; Lepora, N.; Orsag, M. Convolutional Autoencoder for Feature Extraction in Tactile Sensing. *IEEE Robot. Autom. Lett.* **2019**, *4*, 3671–3678. [CrossRef]
56. Baldi, P. Autoencoders, Unsupervised Learning, and Deep Architectures. In Proceedings of the ICML Workshop on Unsupervised and Transfer Learning, JMLR Workshop and Conference Proceedings, Washington, DC, USA, 27 June 2012; pp. 37–49.
57. Doroudi, S. The Bias-Variance Tradeoff: How Data Science Can Inform Educational Debates—Shayan Doroudi. 2020. Available online: https://journals.sagepub.com/doi/full/10.1177/2332858420977208 (accessed on 2 December 2021).
58. Mehta, P.; Bukov, M.; Wang, C.-H.; Day, A.G.R.; Richardson, C.; Fisher, C.K.; Schwab, D.J. A High-Bias, Low-Variance Introduction to Machine Learning for Physicists. *Phys. Rep.* **2019**, *810*, 1–124. [CrossRef] [PubMed]
59. Geman, S.; Bienenstock, E.; Doursat, R. Neural Networks and the Bias/Variance Dilemma. *Neural Comput.* **1992**, *4*, 1–58. [CrossRef]
60. Kruschke, J. *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan*; Academic Press: Boston, MA, USA, 2014; ISBN 978-0-12-405916-0.
61. An, G. The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. *Neural Comput.* **1996**, *8*, 643–674. [CrossRef]

62. Bishop, C.M. Training with Noise Is Equivalent to Tikhonov Regularization. *Neural Comput.* **1995**, *7*, 108–116. [CrossRef]
63. Neelakantan, A.; Vilnis, L.; Le, Q.V.; Sutskever, I.; Kaiser, L.; Kurach, K.; Martens, J. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv* **2015**, arXiv:1511.06807.
64. Lecun, Y. The Mnist Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 10 December 2021).
65. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
66. Hull, J.J. A Database for Handwritten Text Recognition Research. *IEEE Trans. Pattern Anal. Mach. Intell.* **1994**, *16*, 550–554. [CrossRef]
67. Mossing, J.C.; Ross, T.D. Evaluation of SAR ATR Algorithm Performance Sensitivity to MSTAR Extended Operating Condition-sProceedings of the Aerospace/Defense Sensing and ControlsOrlando, FL, USA, 13–17 April 1998, pp. 554–565.
68. Graff, C. *UCI Machine Learning Repository*; University of California, School of Information and Computer Science: Irvene, CA, USA, 2019; Volume 4, Available online: http://archive.ics.uci.edu/ml (accessed on 20 December 2021).
69. Bischl, B.; Mersmann, O.; Trautmann, H.; Weihs, C. Resampling Methods for Meta-Model Validation with Recommendations for Evolutionary Computation. *Evol. Comput.* **2012**, *20*, 249–275. [CrossRef]
70. Glorot, X.; Bordes, A.; Bengio, Y. Deep Sparse Rectifier Neural Networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
71. Arthur, D.; Vassilvitskii, S. K-Means++: The Advantages of Careful Seeding. Available online: http://ilpubs.stanford.edu:8090/778/ (accessed on 2 December 2021).
72. Rosenberg, A.; Hirschberg, J. 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), Prague, Czech Republic, 28–30 June 2007; Association for Computational Linguistics: Prague, Czech Republic, 2007; pp. 410–420.
73. Steinley, D. Properties of the Hubert-Arable Adjusted Rand Index. *Psychol. Methods* **2004**, *9*, 386–396. [CrossRef]
74. Vinh, N.X.; Epps, J.; Bailey, J. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.* **2010**, *11*, 2837–2854.
75. Rousseeuw, P.J. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *J. Comput. Appl. Math.* **1987**, *20*, 53–65. [CrossRef]
76. Zanaty, E.A. Support Vector Machines (SVMs) versus Multilayer Perception (MLP) in Data Classification. *Egypt. Inform. J.* **2012**, *13*, 177–183. [CrossRef]
77. Rahman, M.A.; Muniyandi, R.C. An Enhancement in Cancer Classification Accuracy Using a Two-Step Feature Selection Method Based on Artificial Neural Networks with 15 Neurons. *Symmetry* **2020**, *12*, 271. [CrossRef]
78. Yoon, H.; Park, C.-S.; Kim, J.S.; Baek, J.-G. Algorithm Learning Based Neural Network Integrating Feature Selection and Classification. *Expert Syst. Appl.* **2013**, *40*, 231–241. [CrossRef]
79. Zhang, Y.; Gong, D.; Hu, Y.; Zhang, W. Feature Selection Algorithm Based on Bare Bones Particle Swarm Optimization. *Neurocomputing* **2015**, *148*, 150–157. [CrossRef]
80. Murugan, A.; Sridevi, T. An Enhanced Feature Selection Method Comprising Rough Set and Clustering Techniques. In Proceedings of the 2014 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 18–20 December 2014. [CrossRef]
81. Nekkaa, M.; Boughaci, D. A Memetic Algorithm with Support Vector Machine for Feature Selection and Classification. *Memetic Comput.* **2015**, *7*, 59–73. [CrossRef]
82. Aalaei, S.; Shahraki, H.; Rowhanimanesh, A.; Eslami, S. Feature Selection Using Genetic Algorithm for Breast Cancer Diagnosis: Experiment on Three Different Datasets. *Iran. J. Basic Med. Sci.* **2016**, *19*, 476–482. [PubMed]
83. Mafarja, M.; Mirjalili, S. Whale Optimization Approaches for Wrapper Feature Selection. *Appl. Soft Comput.* **2018**, *62*, 441–453. [CrossRef]
84. Azhagusundari, B. An Integrated Method for Feature Selection Using Fuzzy Information Measure. In Proceedings of the 2017 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, India, 23–24 February 2017; pp. 1–6.
85. Zhang, H.; Nasrabadi, N.M.; Zhang, Y.; Huang, T.S. Multi-View Automatic Target Recognition Using Joint Sparse Representation. *IEEE Trans. Aerosp. Electron. Syst.* **2012**, *48*, 2481–2497. [CrossRef]
86. Dong, G.; Kuang, G. Target Recognition in SAR Images via Classification on Riemannian Manifolds. *IEEE Geosci. Remote Sens. Lett.* **2015**, *12*, 199–203. [CrossRef]
87. Deng, S.; Du, L.; Li, C.; Ding, J.; Liu, H. SAR Automatic Target Recognition Based on Euclidean Distance Restricted Autoencoder. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 3323–3333. [CrossRef]
88. Cui, Z.; Cao, Z.; Yang, J.; Ren, H. Hierarchical Recognition System for Target Recognition from Sparse Representations. *Math. Probl. Eng.* **2015**, *2015*, e527095. [CrossRef]
89. Ni, J.C.; Xu, Y.L. SAR Automatic Target Recognition Based on a Visual Cortical System. In Proceedings of the 2013 6th International Congress on Image and Signal Processing (CISP), Hangzhou, China, 16–18 December 2013; Volume 2, pp. 778–782.
90. van der Maaten, L.; Hinton, G. Viualizing Data Using T-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.