Principal Components Analysis and Autoencoder: a dimensionality reduction comparison using a MNIST data set of American Sign Language

Seminar Business Analytics and Quantitative Marketing

Vladyslav Matsiiako (476414) | Anouk Veltman (466547) | Luca Zampierin (454696) | Laura Zwiers (468008)

April 19th, 2021

Abstract

There are many data sets with a very high number of features, which makes it hard to practically use this data because of high time and space complexities. Dimensionality reduction techiques are used for solving this problem. In this paper, we use Principal Component Analysis (PCA), as well as autoencoders, for finding a low-dimensional representation of a MNIST data set consisting of pictures of American Sign Language. We find that both methods can be applied in such a way that the original data structure is retained almost perfectly, with PCA doing this in a computationally more efficient way. Moreover, both the Deep Autoencoder as the Denoising Deep Autoencoder are found to create embeddings that are highly suitable for producing meaningful clusters, and to even outperform the full-dimensional data in this aspect. It is therefore concluded that both dimensionality reduction techniques work well on the data set used, and that it depends on the application which one is preferred.

Contents

1 Introduction

In recent years, there has been a vast increase in the availability and accessibility of data. Previously unsolvable problems can now be tackled by efficiently using the data at hand, often with the help of machine learning techniques. Whether it is finding the most efficient price for property rentals (Ye et al., 2018), predicting the outcome of a soccer game (Leung & Joseph, 2014), or diagnosing a neurological disease (Siuly & Zhang, 2016), the answer often lies within the data. We as humanity simply need to find the right ways to use them.

Even though most research questions can be answered with the data available, many challenges remain. One cause of this is that some data sets are simply too large to be analyzed efficiently. Generally, this can happen for two reasons: (1) a too large number of observations, or (2) the data set contains too many features per observation. The latter is referred to as the curse of dimensionality, which was introduced by Bellman (1961) in his book "*Adaptive Control Processes: A Guided Tour*". There, he mentioned that with too many dimensions it could become too complicated to visualize or process data.

Since simply removing observations or features from data sets is undesirable, as it would mean deleting potentially relevant information, dimensionality reduction techniques were introduced. These techniques aim to find a meaningful lower-dimensional representation of the data by performing transformations (Van Der Maaten, Postma, & Van den Herik, 2009). Reducing dimensionality is particularly useful for two things, namely visualization and obtaining the intrinsic dimensionality of the original data. The idea of the former is that by visualizing a data set, one can obtain insights about its properties. Obviously, we can only visualize up to three dimensions. Knowledge of the intrinsic dimensionality is interesting, as this can aid in understanding data properties and performing proper clustering (Verveer & Duin, 1995). Moreover, representing the data in a smaller feature space reduces time and space complexities and thus increases efficiency (DeMers & Cottrell, 1993). Training a predictive model on a data set of high dimension often makes the model overfit and hence not useful for out-of-sample predictions (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). By reducing the dimensionality, features that are useless in predictive models are removed and the noise in the data decreases.

A data set that is widely used for machine learning was introduced by the Modified National Institute of Standards and Technology (MNIST). This data set, consisting of 60,000 28x28 pixel gray-scale images of handwritten digits, is often referred to as the "Hello World"

implementation of machine learning and has become the most used data set in deep learning (Xiao, Rasul, & Vollgraf, 2017). By applying machine learning techniques, in particular neural networks, this data set can be used for image classification and the training of various image processing systems. In this paper, we use a MNIST data set with pictures of American Sign Language (ASL) gestures. The data allow for a computer to learn to identify the sign depicted in a picture, making it possible to understand ASL more quickly. If implemented efficiently, the use of machine learning techniques on this data could help people with a hearing impairment to communicate more easily.

The data set used has 784 features, hence dimensionality reduction is needed to effectively analyze it. Principal Component Analysis (PCA), as introduced by Pearson (1901), is a very popular method for this, especially due to its simplicity. However, a downside of PCA is the fact that it is a linear technique (Bourlard & Kamp, 1988). A machine learning method that could overcome this downside is multilayer autoencoders. This technique proposes a type of artificial neural network that aims to minimize the Mean Squared Error (MSE) between the input and a reconstruction of it after passing the information through a bottleneck of hidden layers. The idea is to train a network that maintains as much of the original data structure as possible when transforming to a lower-dimensional representation (Betechuoh, Marwala, & Tettey, 2006).

By creating a lower-dimensional representation of the data using the proposed dimensionality reduction techniques, visualization and cluster analysis become possible. A two- or three-dimensional representation of the original data can be made, which gives more insight into the structure of the data. A method that is argued to be able to do this very well, is t-distributed Stochastic Neighbor Embedding (t-SNE) (Van der Maaten & Hinton, 2008). This method is used to get an idea of whether it is possible to distinguish inherently present clusters in the data. By using the K-means (Hartigan & Wong, 1979) and K-medoids (Kaufman & Rousseeuw, 1990) clustering algorithms, this possibility is investigated further. These algorithms aim to assign observations to different clusters where points within a cluster are similar, while points in different clusters are dissimilar to each other. This is particularly useful for the MNIST data set used, as accurate clustering and classification makes it easy for a computer to understand ASL.

Previous research comparing PCA and autoencoders finds varying results. Wang, Yao, and Zhao (2016) observe that for data with repetitive structures, autoencoders are more suitable. Van Der Maaten et al. (2009) reduces the dimensionality of both artificial as real data sets, among which is the handwritten digits MNIST data. This paper finds that PCA is better at retaining the structure of artificial data, while for real data sets, its performance is similar to that of autoencoders.

It thus seems that the relative performance of both methods is highly dependent on the data used.

We compare PCA and autoencoders in their ability to retain the structure of the MNIST data set of ASL, which is a data set that has not been used for this purpose in earlier research. Moreover, the performance of t-SNE in its ability to visualize data is evaluated and we investigate whether the low-dimensional embedding can efficiently be used in clustering algorithms. Finally, we are interested in whether there exists some sort of accuracy-efficiency trade-off between methods. This leads us to the following research question:

> *How do PCA and autoencoders compare in the dimensionality reduction task for a MNIST data set of ASL?*

In the remainder of this paper, we first introduce the data and investigate its main properties. Afterwards, the methods section extensively explains PCA and autoencoders, as well as the different measures for model comparison and clustering algorithms. The results section presents the main findings from these methods. Finally, answers to the research question are summarized in the conclusion, after which these are discussed and suggestions for further research are given.

2 Data

In this paper, we use the Sign Language MNIST data set (Sign Language MNIST, 2017). This data set contains pictures of ASL, where the letters J and Z have been excluded since they require motion. The observations have a one-to-one mapping with the letters of the alphabet, where 0 is the A and 25 represents the Z. The total number of observations is 34,627, of which 27,455 are in the training set and the remaining 7,172 in the test set. For the purpose of this research, we use 1,500 of the test observations as a validation set (see Section 4.3.1). From Figure 1, we see that the training data set is fairly balanced with all the signs taking around 4% of values, which is particularly important for training autoencoders.

Similar to the standard MNIST data set, each row in the data set corresponds to a 28x28 pixel image, and hence there are 784 pixels for each observation. The pictures were edited to be gray-scale, cropped to hands only, resized and made in at least 50 variations to maximize the quantity. Modifications included applying filters while performing 5% random pixelation, adding 15% variations in contrast and brightness, and rotating by up to 3 degrees in both directions. Due to the minimum image sizes, these changes largely affect the picture quality and class separation

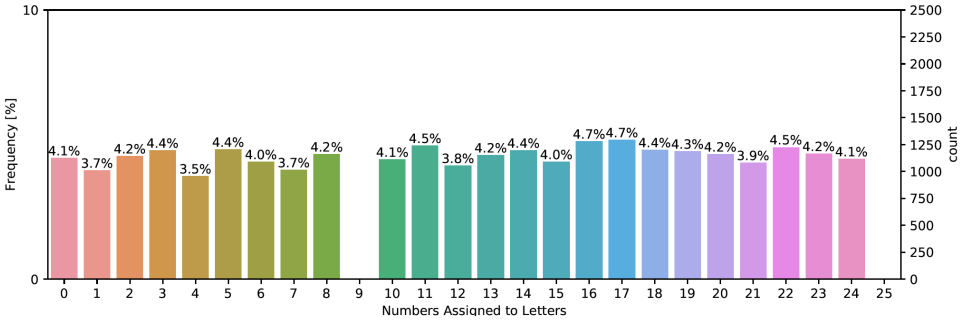
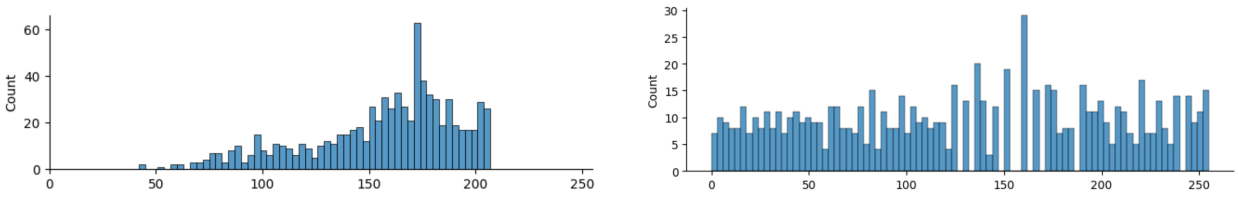


Figure 1. Distribution of Observations in the Training Set

in meaningful and controllable ways. In the final data set, each pixel is assigned a value between 0 and 255 that corresponds to its intensity. Example pictures can be found in Appendix A.

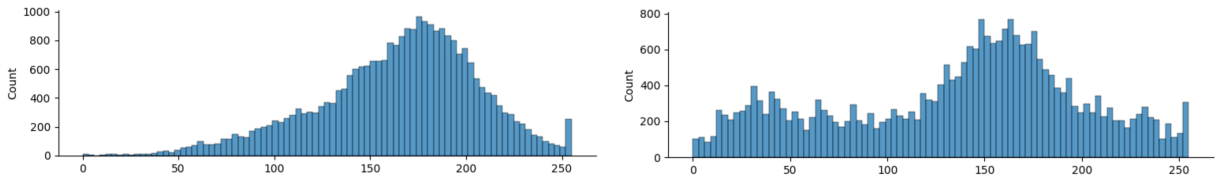
In Figure 2(a), we see that for a random picture, none of the pixels has intensity equal to 0 or 255. In fact, most pixels have an intensity level that is close to the center of the range. In order for the different models to better determine the features inside the data, we increase the contrast of the pictures using histogram equalization. The idea of this method is to map the distribution of the pixel intensity of the input image to that of the uniform distribution (Celik, 2012). To implement this method we employ the *OpenCV* library (Bradski, 2000). In Figure 2(b), it is seen that the pixel intensity distribution becomes much more uniform when contrast is added. The same pattern can be observed when looking at the intensity assigned to the same random pixel over the full data set. Comparing Figures 3(a) and 3(b), we see that the histogram for the contrasted data is more spread out.



(a) Original Data *(b)* Contrasted Data

Figure 2. Distribution of pixels intensity for a random picture (a) and its contrasted version (b)

One reason for believing that contrasting the pictures can be useful, is the fact that computers might have difficulties distinguishing between signs when the distribution of pixels is too narrow. In the reconstructed pictures (Appendix A), it is seen that some features of hands are not easily recognizable. In fact, some autoencoder models get stuck in local minima for these

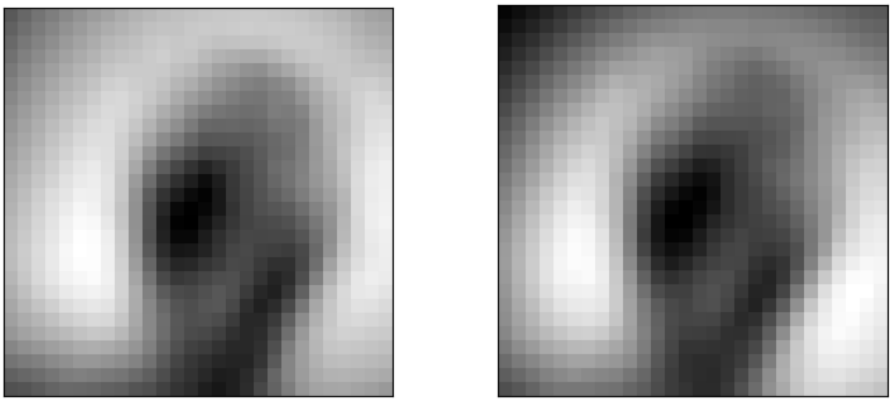


(a) Orignal Data *(b)* Contrasted Data

Figure 3. Distribution of values assigned to a 6x14 pixel (a) and its contrasted version (b)

pictures and end up constantly reconstructing a fist, the average picture (see section 4.3.1). As explained in Section 3.4, the training is based on the MSE between the input and its reconstruction; therefore, if most of the pixels have intensity around the average, it is more likely that the model gets stuck on the average reconstruction. Adding contrast, i.e. having more pixels with extreme values of 0 ad 255, will force these models to learn the relevant features of the picture. This pre-processing step can be further justified by the fact that autoencoders have been proven to perform well for binary MNIST data sets (Wang et al., 2016), which is the most extreme situation where all pixels can only be either white or black. Figure 4(b) presents the average picture after adding contrast and it can be noticed that the features are easier to notice.

The last pre-processing step involves standardizing the data in order to speed up the training procedure. Min-max normalization is employed as this allows for preserving exactly all relationships in the data (Jayalakshmi & Santhakumaran, 2011). We center the data around zero by subtracting the mean intensity of 0.5. Thus, all the observations lay in the range $[-0.5, 0.5]$. The data set of such contrasted and standardized pictures is used in the remainder of the paper.



(a) Original Data *(b)* Contrasted Data

Figure 4. Average intensities across all original pictures (a) and their contrasted versions (b)

3 Methods

3.1 Notation

Dimensionality reduction techniques aim to find a lower-dimensional embedding of the data that captures the most important information. The original data set of contrasted pictures is denoted by $\mathbf{X}$ and the low-dimensional representation is denoted by $\mathbf{Y}$. The number of observations is denoted n, such that $\mathbf{X}$ is a data matrix of n rows and D columns, where D is the number of features. $\mathbf{Y}$ has n rows and d columns, with $D >>> d$. Several formulas in this section also include a neighborhood search, for which the researcher should specify the size of the neighborhood that should be investigated. The number of nearest neighbors employed is denoted by k.

3.2 Intrinsic Dimensionality Estimation

The intrinsic dimensionality d of the data is the appropriate number of features to represent a data set, according to the inherent structure of the data. If one chooses d too small, important aspects of the data might 'collapse' into the same feature, while a too large d can create noisy projections (Levina & Bickel, 2005). Since most dimensionality reduction methods require the researcher to define d as an input parameter, it is important to have a consistent estimator for it. While many different techniques for finding d have been proposed, an optimal method is yet to be found. Fukunaga and Olsen (1971) propose to use an algorithm based on eigenvalues. The idea of the Fukunaga-Olsen algorithm is to divide the data set into small regions and compute the covariance matrix of each region. According to the Karhunen-Loève expansion (Fukunaga, 2013), the dimensionality of the region equals the number of non-zero eigenvalues of this covariance matrix. The intrinsic dimensionality of the full data set is equal to the number of eigenvalues that are significantly different from zero after normalization.

Another method for estimating d is Correlation Dimension (CD) (Grassberger & Procaccia, 2004). In this method, d is calculated as $d = lim_{r \to 0} \frac{ln(C_m(r))}{ln(r)}$, where $C_m(r)$ is the number of distances between observations that are less than r, divided by the total number of distances between points. The downside of CD is that in practice it requires the researcher to specify k as an input parameter. The ideal value of k is unknown, and in some cases the choice of k has a large effect on estimates. Moreover, Eckmann and Ruelle (1992) found that the Grassberger-Procaccia algorithm only yields accurate estimates if the inequality $d < 2log_{10}(n)$ holds true.

Levina and Bickel (2005) argue that Maximum Likelihood Estimation (MLE) performs

better in finding d than other methods. However, this estimation also depends largely on the value of k chosen as model parameter and a definitive estimator can only be found if estimates converge for certain values of k. Because of this non-robustness, we decide to not investigate MLE for intrinsic dimensionality estimation.

3.3 PCA

PCA, as invented by Pearson (1901) and later independently further developed and named by Hotelling (1933), aims to find a low-dimensional representation of the data that describes as much of the original variance of $\mathbf{X}$ as possible. The method calculates a set of principal components, which are linear combinations of the original variables, that are able to explain the variance-covariance structure of $\mathbf{X}$. The different principal components are uncorrelated, thus orthogonal, to each other.

For computing the principal components, the covariance matrix of $\mathbf{X}$ and its eigenvectors and eigenvalues are calculated first. The eigenvectors then represent the directions of the different principal components, while the eigenvalues correspond to their magnitude. That is, the direction of the eigenvector corresponding to the largest eigenvalue is also the direction of the first principal component. The percentage of variance explained by each principal component equals the eigenvalue divided by the sum of all eigenvalues of $\text{cov}(\mathbf{X})$.

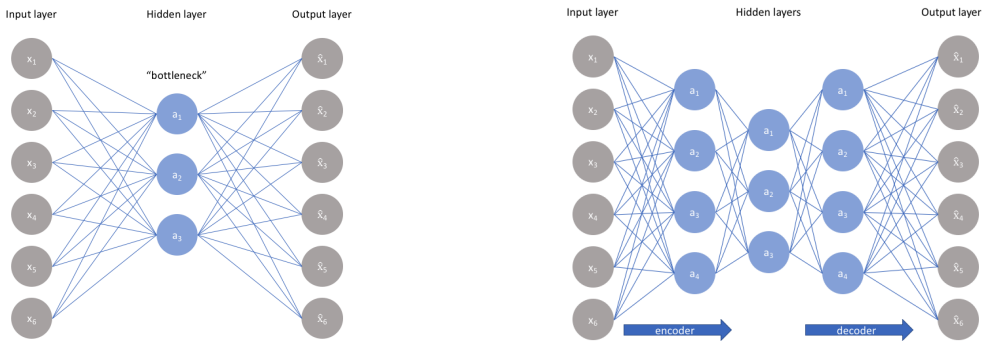
Using the notation suggested by Van Der Maaten et al. (2009), finding $\mathbf{Y}$ in PCA is equivalent to finding a linear mapping $\mathbf{M}$ that solves the following eigenproblem for the eigenvalues of the covariance matrix λ: $\text{cov}(\mathbf{X})\mathbf{M} = \lambda\mathbf{M}$

3.4 Autoencoders

Autoencoders, also known as auto-associative neural network encoders (Wang et al., 2016), are unsupervised learning algorithms. They are used for multiple tasks, such as lossy image compression (Theis, Shi, Cunningham, & Huszár, 2017), image denoising (Vincent et al., 2010) and image generation (Oord et al., 2016). In this paper, however, we focus on using autoencoders for dimensionality reduction. The structural characteristic of this neural network which makes it suitable for this task is the equal number of nodes in the input and output layers. In fact, the objective of the model is to reconstruct a copy of the input data after conveying the information through a lower dimensional subspace.

3.4.1 Single hidden layer autoencoder

The simplest version of this model is depicted in Figure 5(a), where only one hidden layer is inserted between the input and output nodes. Both for single and deep autoencoders, this layer is often referred to as code layer (Hinton & Salakhutdinov, 2006) and its number of nodes corresponds to the lower dimensionality that the researcher wants to obtain (d). On the other hand, the input and output layers both have D nodes corresponding to the dimensionality of the full data, with $D >>> d$. Each node i in the input layer is connected to the hidden layer via edges described by the weight $w_{i,j}$, with j any node in the hidden layer. The hidden layer is connected to the output layer in a similar manner, but we assume symmetric weights for the input-hidden connection and the hidden-output connections. This tied weights assumption is a common practice when working with autoencoders (Li & Nguyen, 2018). Furthermore, we assume no edges within the same layers. The autoencoder will adjust the parameters in order to produce a reconstructed observation that has close resemblance to the original.



(a) Single Layer *(b)* Multi Layer

Figure 5. Single layer (a) and multi-layer (b) autoencoders

As depicted in Figure 5, the model can be divided into two parts: the encoder and the decoder. The former transforms the input data into a d-dimensional observation via a function f ($\mathbf{y} = f(\mathbf{x})$) while the latter reconstructs a D-dimensional observation from the information encoded in the code layer via a function g ($\mathbf{x}' = g(\mathbf{y})$). Here $\mathbf{x}$ describes one observation of the input data, $\mathbf{y}$ a low-dimensional observation, and $\mathbf{x}'$ the output value. Hence the autoencoder as a whole can be described by $g(f(\mathbf{x})) = \mathbf{x}'$.

For the single hidden layer model, we can define $\mathbf{W}$ as the weight matrix in the encoder part where each entry $w_{i,j}$ represents the weight between the hidden node j and the input node i.

Furthermore, given the assumptions presented before, the weight matrix in the decoder part $\mathbf{W}^*$ is set equal to $\mathbf{W}^T$. Consequently, by assuming linear activation functions between the layers and by including the biases $\mathbf{b}$ and $\mathbf{c}$ for the hidden and reconstruction layers respectively, the model can be rewritten as in Equations (1) and (2).

$$\text{encoder} : \mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x} \tag{1}$$

$$\text{decoder} : \mathbf{x}' = \mathbf{c} + \mathbf{W}^T\mathbf{y} \tag{2}$$

The model is trained using backpropagation, the algorithm that drastically improved the usability of neural networks and allows a faster and more efficient learning procedure (Rumelhart, Hinton, & Williams, 1986), with the MSE between the input and the reconstruction as loss function. In order to improve efficiency and performance, the parameters of the network are updated only after computing the loss for a batch of B observations. The final loss is thus the average MSE over all the observations in the batch as presented in Equation (3). The pseudocode for this algorithm is presented in Appendix B.

$$\mathcal{L} = \frac{1}{B}\sum_{l=1}^{B}\frac{1}{D}\sum_{k=1}^{D}(x'_{lk} - x_{lk})^2 \tag{3}$$

It can be shown that, for the simple autoencoder with only one hidden layer and linear activation functions in the decoder part, the linear autoencoder is optimal (Bourlard & Kamp, 1988). Since the linear autoencoder with linear decoder corresponds to PCA (Baldi & Hornik, 1989), this implies that PCA would be optimal under these restrictions. In the following sections, we show how autoencoders can be adjusted to possibly perform better than PCA.

3.4.2 Deep Autoencoder

The simple one-layer autoencoder is often not enough to capture all the information included in more complex data sets. Instead, multilayer autoencoders, or Deep Autoencoders, can be employed. To construct this model we include multiple hidden layers such that the dimensionality gradually decreases layer after layer in the encoder until the d-dimensional subspace is achieved. In the decoder, the process is mirrored and the dimensionality increases layer after layer until the D-dimensional space is reconstructed. An example of this structure is depicted in Figure 5(b). The connections between successive layers are described by weight matrices, thus each node i in the layer l is connected to the following layer $l+1$ via edges described by the weight $w_{l,ij}$, with j any node in the *$(l+1)$-th* layer. Furthermore, as typically done with neural network models, we include

a bias term in each layer, except for the input layer. To allow for non-linearity, we assume sigmoid activation functions in all the hidden layers except for the code layer and the output layer where we use a linear activation function. The choice of the linear activation for these two layers is based on the fact that the observations included in our data set are real-valued, due to the gray-scaling. As for the single-layer case, we assume tied weights and no intra-layer connections.

The training procedure for Deep Autoencoders is also based on backpropagation and it uses the same MSE loss function as the single-layer autoencoder (see Equation 3). The backpropagation algorithm requires the initialization of weights and biases using random values; however, it has been shown that the learning procedure can be excessively long and lead to unsatisfactory local minima, i.e. underfitting, in case the parameters are initialized to very inefficient values (Kambhatla & Leen, 1997). A first solution, proposed by Hinton and Salakhutdinov (2006), is to perform a pre-training procedure based on Restricted Boltzmann Machines (RBM) in order to initialize the parameters for the backpropagation algorithm at an already "good enough" solution. Since this first breakthrough, multiple pre-training techniques based on unsupervised learning algorithms have been proposed. In particular, Vincent et al. (2010) have shown that pre-training deep neural networks using stacked denoising autoencoders provides significant improvements in classification performances. We therefore implement this pre-training strategy in this paper. For this, we consider both the so-called masking noise (Vincent et al., 2010), where a percentage ν of the inputs is randomly set to zero, and the Gaussian noise, where the added noise is randomly drawn from a Gaussian distribution with zero mean and standard deviation σ kept as an hyperparameter to be tuned. After pre-training, the full autoencoder is fine-tuned using backpropagation.

When working with machine learning models, it is fundamental to set the correct hyper-parameters in order to achieve the best performance possible. In this research we employ a Random Grid Search, as it has been shown to achieve models that are as good as those found by pure Grid Search in a fraction of the time (Bergstra & Bengio, 2012). We use cross-validation to select the optimal set of hyperparameters among those presented in Table 1. The number of folds used for cross validation is often set to five or ten and we decide to employ the former, simply to avoid a too computationally heavy training procedure. We fix the number of epochs for pre-training and fine-tuning to 20 and 70, respectively, to reduce the complexity of the cross-validation after noticing that the convergence is usually achieved before the 70 epochs of fine-tuning. Similarly, we selected the Adam optimizer (Kingma & Ba, 2014) based on a quick trial-and-error analysis on the training set in order to reduce the choice space and thus speed up the Grid Search. The optimal

model is the one that results in the lowest average validation loss over all the folds. Until today, an exact solution for finding the optimal number of hidden layers has not been found (Stathakis, 2009). We looked into previous research on applying autoencoders to MNIST data sets in order to find possible structures to be included in the Grid Search.

Table 1

Deep autoencoder hyperparameters tested using cross-validation

Hyperparameter	Considered values
Hidden Layers	[800, 400, 200, d], [800, 250, d], [620, 330, 100, d], [620, 330, d], [500, 250, 100, d], [500, 250, d]
Batch size	8, 16, 32, 64, 128
Pre-training noise	'masking': 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 'Gaussian': 0, 0.3, 0.5, 0.75, 1, 1.75, 2
Pre-training learning rate	0.001, 0.002, 0.003, 0.005, 0.01
Fine-tuning learning rate	0.001, 0.002, 0.003, 0.005, 0.01

[1] The parameter of the 'masking' noise is the percentage of pixels that is randomly set to 0.
[2] The parameter of the 'Gaussian" noise is the st. dev. of the normal distribution used to draw the noise.

A common drawback of deep neural networks is their difficult interpretation (Montavon, Samek, & Müller, 2018). Indeed, compared to the explained variance of PCA, the reduced representation resulting from autoencoders is much more complicated to analyze. To gain some understanding, we study the features that are being extracted from the observations by the first hidden layer of the neural network. This analysis can be performed because the connection between the input layer and each node of the first hidden layer is represented by a set of 784 weights that can be then reshaped into a 28x28 matrix. In this matrix, large weights (in absolute value) indicate that the given hidden node is focusing on the corresponding input pixels. To facilitate the analysis, we visualize these matrices in gray-scaled pictures where black and white represent large negative and positive values, respectively. This analysis will ultimately tell us which features of the original picture each hidden node in the first hidden layer is trained to capture.

3.4.3 Denoising Deep Autoencoder

It is well known in the literature that overfitting can be a very challenging issue for deep neural networks (Srivastava et al., 2014). Aware of this challenge, we explore a common technique used to reduce overfitting, namely training the Deep Autoencoder using a noisy version of the data set. We refer to this model as Denoising Deep Autoencoder. In particular, given a certain noise function $C(x)$, the model receives a corrupted version of the original observations as input, defined as $\tilde{x} = C(x)$, and is trained to reconstruct the original clean data set. By using a corrupted input, the

model is forced to learn general properties of the data instead of simply memorizing very specific characteristics of the training data. Consequently, the low-dimensional embedding should be a better representation of the true distribution of the original data (Bengio, Yao, Alain, & Vincent, 2013). Similarly to the pre-training stage, we consider both the Gaussian and the masking noises. Once again, we perform a random Grid Search among the hyperparameters listed in Table 2 with five folds cross validation to choose the optimal model. The pre-selected hyperparameters are the same as for the Deep Autoencoder.

Table 2

Denoising Deep Autoencoder hyperparameters tested using cross-validation

Hyperparameter	Considered values
Hidden Layers	[800, 400, 200, d], [800, 250, d], [620, 330, 100, d], [620, 330, d], [500, 250, 100, d], [500, 250, d]
Batch size	8, 16, 32, 64, 128
Pre-training noise	'masking': 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 'Gaussian': 0, 0.3, 0.5, 0.75, 1, 1.75, 2
Fine-tuning noise	'masking': 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 'Gaussian': 0, 0.3, 0.5, 0.75, 1, 1.75, 2
Pre-training learning rate	0.001, 0.002, 0.003, 0.005, 0.01
Fine-tuning learning rate	0.001, 0.002, 0.003, 0.005, 0.01

[1] The parameter of the 'masking' noise is the percentage of pixels that is randomly set to 0.
[2] The parameter of the 'Gaussian" noise is the st. dev. of the normal distribution used to draw the noise.

3.5 t-SNE

t-SNE, as introduced by Van der Maaten and Hinton (2008), is a dimensionality reduction technique often used for visualizing data in a two- or three-dimensional space. t-SNE has the ability to capture the original data structure in a lower-dimensional representation and also show naturally present clusters. The idea is based on Stochastic Neighbor Embedding (SNE) (Hinton & Roweis, 2002). SNE works by converting Euclidean distances between data points into probabilities and selecting neighbors based on a Gaussian probability function. The similarity between observation x_i and x_j equals the conditional probability $p_{i|j}$, which could be interpreted as the probability that x_i chooses x_j as its neighbor. $p_{j|i}$ is calculated as in Equation (4), where σ_i is the variance of the Gaussian centered at observation x_i. In Equation (5), we see that $q_{j|i}$ is calculated in a fairly similar way, but for data points in the lower-dimensional embedding and without σ_i included in the formula. Instead, the variance used in this formula is set equal to $\frac{1}{\sqrt{2}}$. Since the probabilities are created to measure pairwise similarities, $p_{i|i}$ and $q_{i|i}$ are both zero.

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i)}{\Sigma_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i)} \tag{4}$$

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\Sigma_{k \neq i} exp(-||y_i - y_k||^2} \tag{5}$$

SNE aims to create a model in which q_i and p_i are as close as possible, which means that the structure of the data is retained in the reduced space. In case of a perfect mapping y_i, the two measures are equal. In order to create a mapping in which q_i and p_i are close, SNE performs optimization on a cost function. This cost function is the sum of so-called Kullback-Leibler divergences and is optimized using a gradient descent algorithm. In SNE, the Kullback-Leibler divergence is not symmetric, which makes optimization computationally difficult. t-SNE overcomes this by employing a symmetric cost function with simpler gradients. Furthermore, by replacing the Gaussian with a student t-distribution, computations are simplified further. t-SNE often outperforms other methods in its ability to visualize data, especially when it comes to showing inherently present clusters. Therefore, it is useful to see if this method is able to show the clusters in the ASL data set, especially because it can give insights into the performance of clustering measures introduced in Section 3.7. Ideally, t-SNE yields visualizations in which pictures of the same sign are close to each other in the embedding, while pictures of different signs are further apart.

The pairwise distances in t-SNE are defined differently than those in SNE. In particular, it holds that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for all i and j. This is ensured by setting $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. For the lower-dimensional representation, a student t-distribution with one degree of freedom is assumed. This results in q_{ij} defined as in Equation (6). These p_{ij} and q_{ij} allow for the gradient of the Kullback-Leibler divergence (Equation 7) to be easier to compute.

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}} \tag{6}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{7}$$

The simple t-SNE procedure, which is fully described in Appendix E, works with a small set of input parameters. First, the perplexity ($Perp$) is a measure that indicates how the algorithm should balance between retaining local and global aspects of the original data. $Perp$ usually has a value between 5 and 50, and in our research it is set to 5 in order to avoid the possibility of ending up with an embedding of uniformly distributed points. The maximum number of iterations is set to 1000 and the momentum term, which reduces the number of iterations used, is set to be 0.5. Finally, the learning rate η is set to 100. These parameters correspond to those suggested for

the digits MNIST data set by Van der Maaten and Hinton (2008). Since t-SNE is known to not perform well on data of too high dimensionality, we perform the method on a representation of the data that is found using autoencoders.

3.6 Model Comparison

The ability of dimensionality reduction techniques to retain the original structure of the data can be measured with two metrics: trustworthiness and continuity (Venna & Kaski, 2006). The trustworthiness measure, as shown in Equation (8), looks at the proportion of data points that are too close to each other in the low-dimensional representation. The continuity measure (Equation (9)), on the other hand, focuses on the points that have been pushed farther away in the low-dimension representation. Both measures result in a value between 0 and 1, where a higher score indicates that the embedding retains the structure of the original data well.

In the equations below, $r(i,j)$ is the rank of the low-dimensional data point j according to the pairwise distances between data points and $\hat{r}(i,j)$ is the equivalent of $r(i,j)$ but for j being a high-dimensional data point. $U_i^{(k)}$ is the set of points that are nearest-neighbors in the low-dimensional space but not in the high-dimensional one, and $V_i^{(k)}$ is the opposite.

$$T(k) = 1 - \frac{2}{nk(2n-3k-1)} \sum_{i=1}^{n} \sum_{j \in U_i^{(k)}} (r(i,j) - k) \tag{8}$$

$$C(k) = 1 - \frac{2}{nk(2n-3k-1)} \sum_{i=1}^{n} \sum_{j \in V_i^{(k)}} (\hat{r}(i,j) - k) \tag{9}$$

3.7 Clustering

The idea of cluster analysis is that observations are divided into disjoint groups, where points within a cluster are similar while points in different clusters are not. Generally, the similarity of points is based on their pairwise distance. For the MNIST data set used in this research, the goal is to find 24 distinct clusters which all represent a single sign. With clustering we wish to compare whether the reduced data set leads to similar clustering results as for the original data set.

3.7.1 *K*-means clustering

For the K-means clustering algorithm (Hartigan & Wong, 1979), K is defined as the desired number of centroids around which clusters are formed (Bradley, Bennett, & Demiriz, 2000). The objective

of this method is to minimize within-cluster variance. The data used as input for the algorithm is denoted by $\mathbf{Z}$, with z_i being an element of the data matrix. In the initialization step, K elements from $\mathbf{Z}$ are chosen as the centers c_j of the starting clusters, resulting in the set $\mathbf{C}$ of initial centroids. Then, the algorithm assigns z_i to cluster C_j if $\text{dist}(z_i, c_j) = min_{c_j \in \mathbf{C}} \text{dist}(z_i, c_j)$, with the distance measure being the squared Euclidean distance. After assigning all z_i to clusters, the center, which is the mean point, of each cluster C_j is calculated and set as the new centroid. This procedure is repeated until the set of centroids does not change anymore. A full description of this procedure can be found in Algorithm 2 in Appendix C. It should be noted that, even in its most simple case, the K-means algorithm is proven to be NP-hard, implying that it has high complexity (Mahajan, Nimbhorkar, & Varadarajan, 2009).

Given this high complexity, K-means often converges to local minima. To partly overcome this issue, the K initial centroids are initialized using the K-means++ method (Arthur & Vassilvitskii, 2006). This method chooses the first centroid c_1 to be a random point from the data, and subsequently the next $(K-1)$ cluster centers are chosen such that they are furthest away from each other. That is, the probability of choosing z_i as a next centroid equals $\frac{\text{dist}(z_i,c_j)^2}{\sum_{z \in \mathbf{Z}} \text{dist}(z_i,c_j)^2}$, with the distance measure being Euclidean and c_j the already chosen centroid that is closest to z_i. The full K-means++ procedure is shown in Algorithm 3 in Appendix C.

3.7.2 *K*-medoids clustering

The K-means algorithm is sensitive to outliers, since means are easily influenced by extreme values. A method that is more robust is K-medoids clustering, also referred to as Partition Around Medoids (PAM) (Kaufman & Rousseeuw, 1990). The main difference compared to K-means is that K-medoids does not use the mean of a cluster as its center, but instead chooses an actual data point to represent the cluster. The K initial medoids are chosen in a similar way as in the K-means initialization. Medoids are then switched around using the cost function (Equation 10). In the cost function, the C_j are the different clusters, z_i are points in the data set used and c_j is the medoid of the j-th cluster. Moreover, the distance function used corresponds to the Euclidean distance. The full algorithm can be found in Appendix D. The drawback of the method is that for large data sets, it is less efficient than K-means because of high time complexity (Han, 2001).

$$\text{cost} = \sum_{j=1}^{K} \sum_{i \in C_j} \text{dist}(z_i, c_j) \tag{10}$$

3.7.3 Clustering performance

The clustering performance is evaluated based on the extent to which data points with the same label are assigned to the same cluster, where the labels correspond to the letter that is depicted in an image. Three measures used for this evaluation are homogeneity, completeness and the V-measure. Homogeneity is satisfied if clusters contain data points that all have the same label and completeness implies that all observations with the same label are assigned to the same cluster. The final measure is the V-Measure, as introduced by Rosenberg and Hirschberg (2007). This measure represents the harmonic mean between homogeneity and completeness, as defined in Equation (11), where h represents the homogeneity of the clustering, c is the completeness and β is a factor that decides the weighting of both measures. In this research, we use $\beta = 1$ because both measures are considered equally important.

$$V = \frac{(1+\beta)hc}{\beta h + c} \tag{11}$$

The three clustering measures all result in a value between 0 and 1, where 1 corresponds to perfect clustering performance. It is of interest to see which low-dimensional embedding can best be used in clustering algorithms, and whether it is possible to get a performance that is similar to that on the non-reduced data.

4 Results

4.1 Intrinsic Dimensionality

We found that the value of the CD estimator depends largely on the number of nearest neighbors chosen as input parameters, and that it often yields estimates that do not satisfy the inequality condition $d < 2log_{10}(n)$. We therefore decide to not investigate this estimator any further. The chosen method for estimating d thus is the eigenvalue approach, based on the Fukunaga-Olsen algorithm (Fukunaga & Olsen, 1971) and is estimated using the *skdim* package in Python (Bac, 2020). With this method, we find an estimate for the intrinsic dimensionality of 13, which is employed throughout the rest of this results section.

4.2 PCA

The first 13 principal components found using PCA, with the use of the *sklearn.decomposition* package of Python (Pedregosa et al., 2011), are able to explain 62.8% of the variance of the original

data. The first component explains 14.3% of this variance and this number decreases to 1.7% for the final one. Based on the 13-dimensional embedding from PCA, we were able to reconstruct an image of decent quality, especially considering that PCA is not a method specifically designed for this. The reconstruction can be found in Appendix F.

4.3 Autoencoders

4.3.1 Deep Autoencoder

To build the Deep Autoencoder model we employ the *PyTorch* library (Paszke et al., 2019). When performing randomized Grid Search with five-fold cross validation on the original (non-contrasted) data, the model gets stuck in a local minimum. With all different combinations of hyperparameters tried, we ultimately end up with picture representations similar to the one depicted in Figure 6, with slightly varying levels of brightness.

Figure 6. Picture representation of autoencoders stuck in a local minimum

Essentially, the model learned the underlying structure of our data - a fist - and returned it for every reconstructed picture. To solve this problem, we decided to increase the contrast of pictures. The technicalities of this step have been thoroughly described in Section 2. Performing the randomized Grid Search on the contrasted data set, we found the following optimal model: hidden layers are [620, 330, 13], batch size is 64, noise type is 'masking' with the noise share of 0.5, the learning rate is set at 0.002 at the pre-training stage and 0.001 at fine-tuning.

The model has a MSE of 0.00281 on the training data, and it does not seem to be overfitting based on Figure 7. Figure 8 shows the reproduced pictures of both the train and test set after being reduced from 784 to 13 dimensions. Unfortunately, the reconstructed pictures for the test set look significantly worse. This leads us to think that the model is still overfitting, even though this is not seen on the validation graph.

In fact, when checking the MSEs of reconstructions on the 1500 observations from our test set (taken out specifically for the purpose of extra validation measures) after 20, 50, and 70

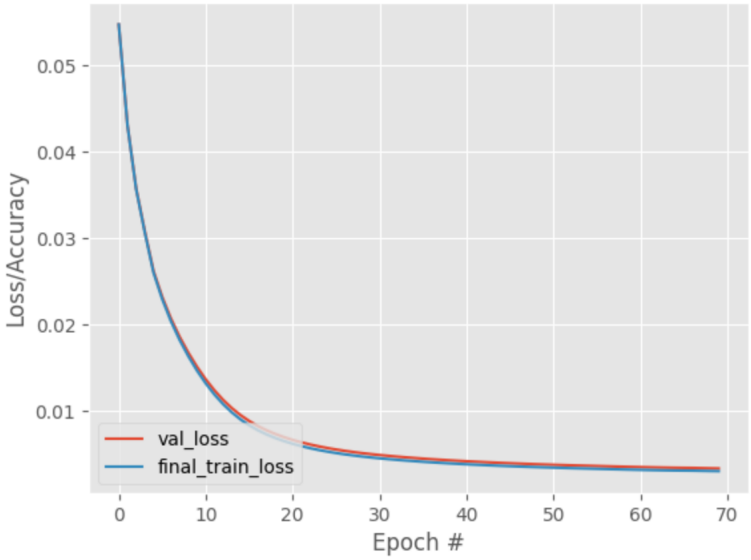


Figure 7. MSE of the train and validation sets for the optimal Deep Autoencoder

epochs of fine-tuning, we get the following values: 0.02749, 0.03168, and 0.03340, respectively. This increasing trend is a clear sign of overfitting, which was not visible in the cross-validation. This happens due to the nature of our data set: to increase the number of observations, new pictures were generated by adjusting existing ones (as described in Section 2). Due to shuffling, there are very similar pictures in both the training and validation data sets, which makes the model to fit almost perfectly there. Nevertheless, since such "extremely similar" pictures are not present in the test set, the performance is reduced.

Because of this, we decide to split the test set into two parts and to use the first 1500 observations as an extra validation set. This helps to determine the point at which the training of the model should be stopped. By doing that, we can observe a typical trend of increasing validation errors after a certain point. The results can be seen in Figure 9, from which we can conclude that training the model until 9 epochs is optimal. By doing this, we find the reconstructed pictures as in Figure 10. The loss on the test set now equals 0.02406. Overall, it can be seen that even though the performance on the train set is lower, the test set reconstructions are significantly better.

Additionally, to grasp what the trained model has learned to capture, we visualize a random sample of the features detected by the first hidden layer. Here, the darkest and brightest pixels represent respectively large negative and positive weights. The gray pixels indicate that the weight is almost zero. Consequently, each 28x28 square indicates which input pixels the hidden node is focusing on and what type of features are detected. We can see that even though some

occasional meaningful features occur, most of them resemble the structure of a hand in specific pictures - another sign of slight overfitting. We aim to solve this problem in the next section.

(a) Training Data

(b) Test Data

Figure 8. Reconstruction of 10 random pictures for the optimal Deep Autoencoder model after 70 epochs of fine-tuning

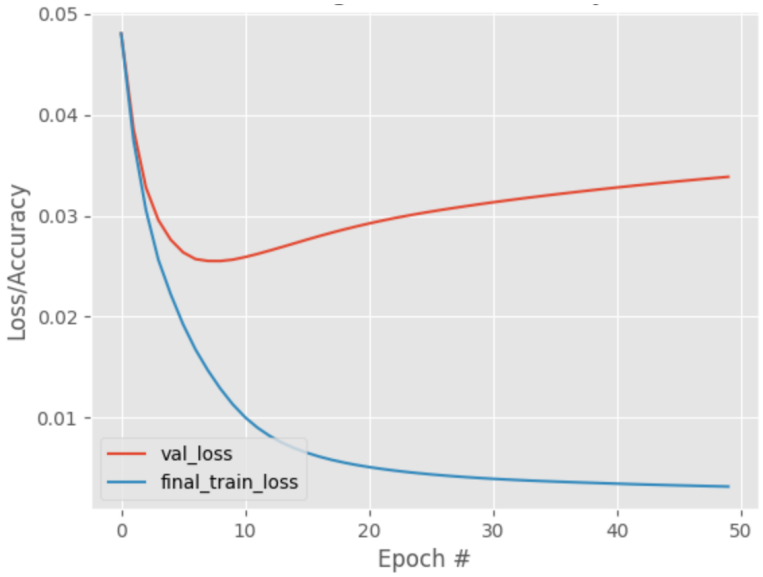


Figure 9. MSEs of the train and test (1500 randomly selected observations) sets for the optimal Deep Autoencoder

(a) Training Data

(b) Test Data

Figure 10. Reconstruction of 10 random pictures for the optimal Deep Autoencoder model after 9 epochs of fine-tuning

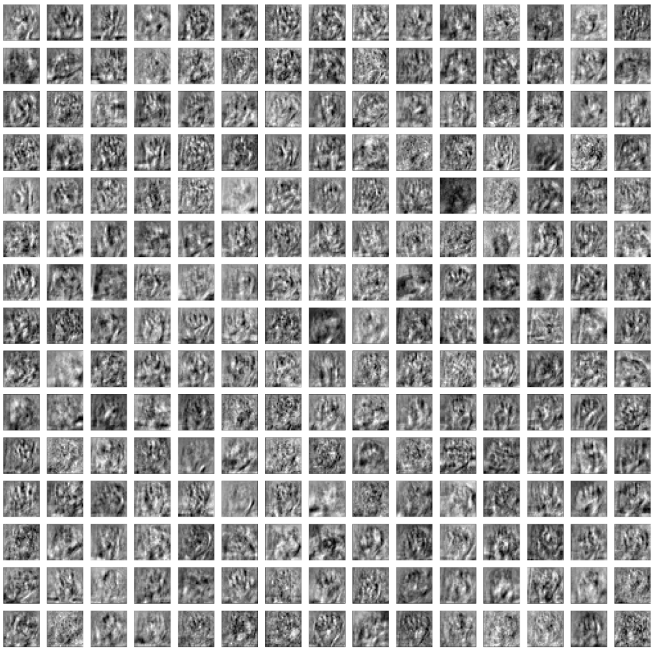


Figure 11. 225 random features detected by the first layer of Deep Autoencoder

4.3.2 Denoising Deep Autoencoder

The Denoising Deep Autoencoder is constructed using the *PyTorch* library (Paszke et al., 2019). The Random Grid Search results in the following optimal model: hidden layers are [620, 330, 13], batch size is 64, pre-training noise type is Gaussian with $\sigma = 0.1$, fine-tuning noise type is Gaussian with $\sigma = 0.3$, the learning rate is set to 0.01 and 0.001 for pre-trainign and fine-tuning respectively.

The training-validation loss graph of this particular model can be found in Figure 12. After 70 epochs the average MSE per picture on the training and validation sets are respectively 0.00693 and 0.00738. The pattern displayed by the losses seems to show that the Denoising Deep Autoencoder is slightly slower in training, in fact the losses cross the 0.01 threshold only after 40 epochs against the 20 epochs of the normal Deep Autoencoder, but it seems to converge to a very similar value for both the training and the validation loss.

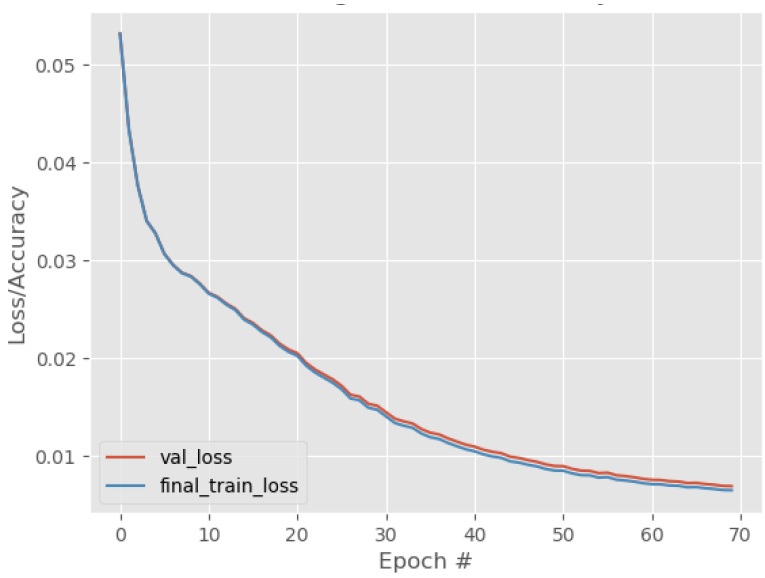


Figure 12. MSE of the train and validation sets for the optimal Denoising Deep Autoencoder

A sample of the reconstructed pictures for both the training and the test sets are presented in Figure 13. It can be observed that, as expected given the convergence to a similar average loss, the training observations are reconstructed almost perfectly as it was for the Deep Autoencoder. On the other hand, the reconstructions for the test set seem significantly better. This is confirmed by the average loss on the 1500 validation observations taken from the test set being 0.0231 after 70 epochs, which is lower than what it was for the Deep Autoencoder.

Still, the reconstruction of the test pictures is far from perfect. As it was for the Deep Autoencoder, the training and validation loss do not seem to indicate any clear form of overfitting as we increase the number of fine-tuning epochs. The two losses seem to separate and follow two different paths as we reach epoch 20, but there is no typical pattern that would indicate overfitting. Therefore, even the Denoising Deep Autoencoder seems to not be able to completely overcome the complexities of the data described in Section 4.3. Nevertheless, the separation between training and validation losses is much more evident for this model. To find the optimal number of fine-tuning

(a) Traning Data

(b) Test Data

Figure 13. Reconstruction of 10 random pictures for the optimal Denoising Deep Autoencoder model after 70 epochs of fine-tuning

epochs we again use 1500 observations from the test set as a validation set. The graph of the losses is presented in Figure 14 and we can see that after 20 epochs the average MSE for the test data seems to stabilize at around 0.023 while the training loss continues to decrease. Particularly, the minimum test loss of 0.0221 is reached after 27 epochs.

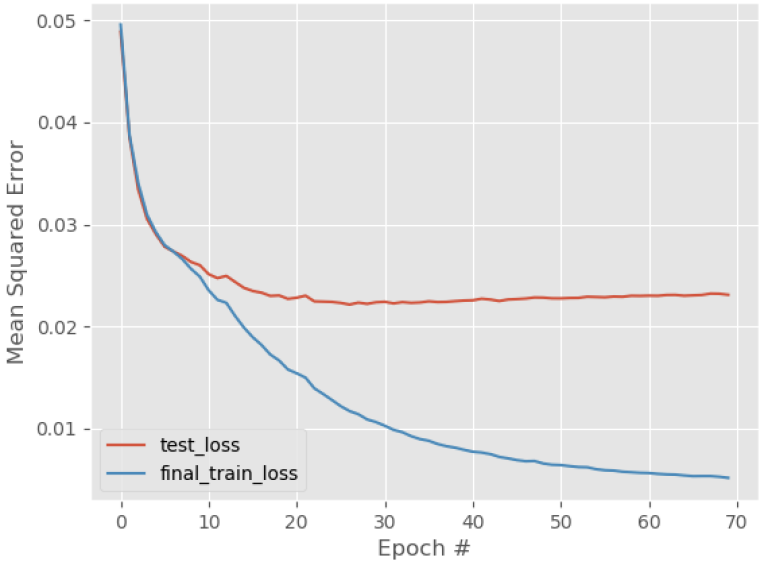


Figure 14. MSEs of the train and test (1500 randomly selected observations) sets for the optimal Denoising Deep Autoencoder

(a) Training Data

(b) Test Data

Figure 15. Reconstruction of 10 random pictures from the test set for the optimal Denoising Deep Autoencoder model after 27 epochs of fine-tuning

It should be noted that the overfitting issue is much less obvious for the Denoising Deep Autoencoder compared to the Deep Autoencoder. This shows that adding noise to the fine-tuning procedure does indeed help in reducing the overfitting. After imposing early stopping at 27 epochs, we find the reconstructions shown in Figure 15(a) and (b). Both the final average loss on the test set of 0.00227 and the minimal improvements in the reconstructions show that employing early-stopping does not improve the performance of the Denoising Deep Autoencoder as much as for the Deep Autoencoder. This implies that using noised pictures helps to generalize the model and reduce overfitting.

It is of interest to understand what the model is trying to capture when reducing the information to a lower dimension. As for the Deep Autoencoder, in Figure 16 we present a sample of the features that are detected by the first hidden layer of the model. The interpretation is the same as for the Deep Autoencoder model. Although most of the nodes randomly selected here do not seem to detect very interesting features, we do perceive some meaningful detectors showing up, such as local blob detectors and locally oriented edge detectors (Vincent, Larochelle, Bengio, & Manzagol, 2008). Comparing these representations with those presented in Section 4.3.1 it can be appreciated that the features detected by the Denoising Deep Autoencoder are more general, which explains the difference in performance on the test set reconstructions.

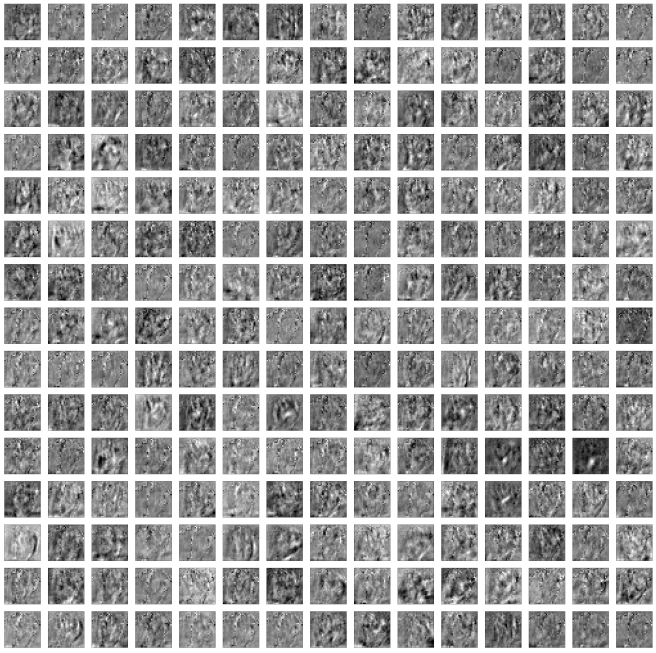
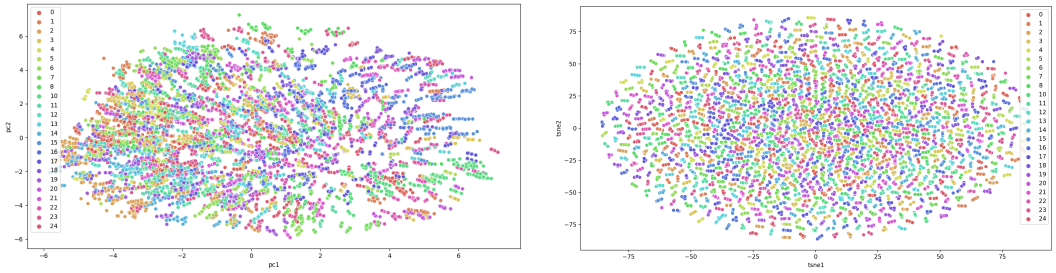


Figure 16. 225 random features detected by the first layer of Denoising Deep Autoencoder

4.4 t-SNE

We apply t-SNE on the 13-dimensional embedding obtained by the Denoising Deep Autoencoder using the *sklearn.manifold* package in Python (Pedregosa et al., 2011). The scatter plot corresponding to the two-dimensional representation obtained by t-SNE, constructed with the *seaborn* package (Waskom, 2021), is depicted in Figure 17(b). The plot looks like a homogeneous ball in which no natural clustering is visible, and can be argued to be no better compared to the two-dimensional visualization obtained with PCA (Figure 17(a)). While Van der Maaten and Hinton (2008) finds two-dimensional visualizations in which inherently present clusters are visible using t-SNE on the digits MNIST data set, this does not happen for our data.

One possible explanation for the disappointing performance of t-SNE is that the data used in this research is not suitable for proper clustering. This hypothesis is further investigated in Section 4.6. t-SNE results could also differ largely from those found in other papers because the MNIST data set of ASL is not binary, but gray-scaled and hence has values between 0 and 255. This property of the data could lead to the algorithm getting stuck in local optima.



(a) PCA *(b)* t-SNE

Figure 17. Two-dimensional Scatter Plots of PCA and t-SNE Embeddings

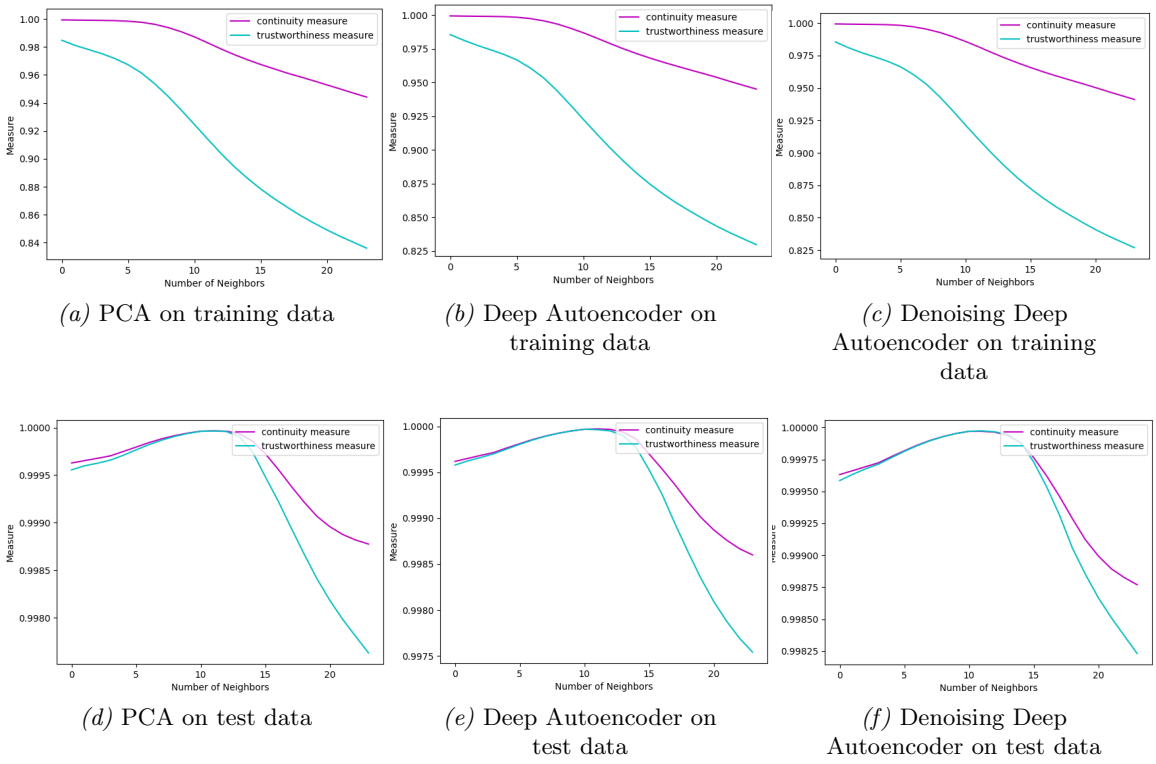
4.5 Model comparison

Since calculating the continuity and trustworthiness is very memory-heavy, we were unable to calculate these on the full train data set. Instead, we run the measures on a sub-sample containing half of the observations in the training set, which is 13,727. This sub-sample is selected based on a fixed random seed and created using the *numpy.random* package (Harris et al., 2020). The two measures are calculated using the Python package *coranking.metrics* (Jackson, 2018), which bases its calculations on Lee and Verleysen (2009).

Both measures are calculated for k=1 until 25, in order to overcome possible biases that relate to the number of neighbors chosen. The measures are shown in Figure 18. We see that for both the test and training set, the measures for PCA are quite similar to those found with the different autoencoders. On the subsample of the training data, the trustworthiness measure is around 0.98 for k=1 and 0.83 for k=25 for all models. The continuity measure decreases from being 1.00 with the minimum number of neighbors to 0.94 for k=25. On the test data, the measures are slightly higher, and basically always 1.00 after rounding.

Comparing the measures across the different embeddings, we find that PCA yields the highest trustworthiness values on the training data. Between the different autoencoders, the Deep Autoencoder gives slightly better values for most k. For the continuity measure, the Deep Autoencoder performs best on the training data, followed by the PCA embedding. On the test data, the Denoising Deep Autoencoder performs best in both trustworthiness and continuity. Between PCA and the Deep Autoencoder, it depends on the value of k which of the two yields a higher value. It can thus be argued that the Denoising Deep Autoencoder works better for the test set, while the Deep Autoencoder is better at keeping the structure of the training data. This finding can be linked to the fact that the Denoising Deep Autoencoder suffers from overfitting less than the Deep

Autoencoder. Overall, however, the differences between models are very small and the measures are close to 1 in all cases. We can conclude that all three embeddings retain the structure of the original data well, and it cannot be concluded from these measures alone which model is best.



(a) PCA on training data

(b) Deep Autoencoder on training data

(c) Denoising Deep Autoencoder on training data

(d) PCA on test data

(e) Deep Autoencoder on test data

(f) Denoising Deep Autoencoder on test data

Figure 18. Trustworthiness and Continuity on Training (Top) and Test (Bottom) Data

4.6 Clustering

The high-dimensional data, as well as the embeddings obtained from both PCA and the two autoencoders, are used for clustering. Cluster analysis is performed in Python using the *sklearn.cluster* and *sklearn_extra.cluster* packages, and the measures introduced in Section 3.7.3 are computed with the *sklearn.metrics* package (Pedregosa et al., 2011). Table 3 shows the measures for the training data and Table 4 shows those for the test data. The best performing technique for each data set and clustering method is shown in bold.

We see that even for the non-reduced data, the measures have relatively low values. This implies that, for some reason, it is hard for the clustering algorithms to actually distinguish between different signs. This is not surprising, as also t-SNE (Figure 17) is unable to show inherently present clusters. When looking at the original pictures in Appendix A, it seems that this bad clustering performance might be due to some signs having too much resemblance. For instance, there are

Table 3

Clustering Measures Training Set

	K-means				K-medoids			
	Full	PCA	Deep AE	Denoising AE	Full	PCA	Deep AE	Denoising AE
Homogeneity	**0.359**	0.341	0.355	0.358	0.319	0.311	0.332	**0.340**
Completeness	**0.363**	0.343	0.359	0.361	0.329	0.319	0.336	**0.346**
V-measure	**0.361**	0.342	0.357	0.359	0.324	0.315	0.334	**0.343**

[1] Deep AE and Denoising AE refer to the Deep Autoencoder and Denoising Deep Autoencoder.

Table 4

Clustering Measures Test Set

	K-means				K-medoids			
	Full	PCA	Deep AE	Denoising AE	Full	PCA	Deep AE	Denoising AE
Homogeneity	0.484	0.460	0.469	**0.504**	0.426	0.469	**0.471**	0.470
Completeness	0.493	0.463	0.472	**0.504**	0.440	0.471	0.471	**0.478**
V-measure	0.488	0.461	0.470	**0.504**	0.433	0.470	0.471	**0.474**

[1] Deep AE and Denoising AE refer to the Deep Autoencoder and Denoising Deep Autoencoder.

several 'fist' signs (A, E, M, N and S) that are very similar. However, even when applying K-means to a data set with these specific signs excluded and $K = 19$, the clustering performance does not improve. Hence, it seems like clustering in general is challenging on this particular data set. Furthermore, for the most part K-means outperforms K-medoids, which could possibly imply that there are little outliers in the data (Angiulli & Pizzuti, 2005).

When comparing clustering results on the full data to those on the representations from PCA and the two autoencoder models, the difference in performance is relatively small. We see that the embedding created with PCA performs slightly worse than the full data in all cases except for the K-medoids on the test data. The data representation from the Denoising Deep Autoencoder outperforms the full data, PCA embedding, and the Deep Autoencoder for both clustering algorithms on the test set and for K-medoids on the training set. Instead, for K-means on the training set the Denoising Deep Autoencoder is second only to the full representation. Hence, the Denoising Deep Autoencoder is the most successful at capturing information from the data set which is useful for clustering. The Deep Autoencoder always outperforms PCA, and for K-medoids this method is even able to yield better results than the full data set. This implies that both autoencoders have the ability to create an embedding that is suitable for clustering. It should also be noted that, overall, the test set is better suitable for creating meaningful clusters than the training set. Most likely this is caused by the two sets having slightly different properties.

5 Conclusion

This paper studies the performance of different dimensionality reduction techniques on a MNIST data set of hand signs. Since the original data set has 784 features, having a well-performing embedding of lower dimension can contribute to more practical usage of it. The intrinsic dimensionality of the data set is estimated to be equal to 13 using the eigenvalue approach, and a representation of this dimensionality is created using both PCA as well as two different autoencoders. PCA is a computationally efficient method to find this representation, and yields a set of principal components that together explain over 60% of total variance.

Autoencoders, on the other hand, is a method with higher computational complexity. A Deep Autoencoder cannot reproduce any picture other than the average one before contrast is added to all pictures. With added contrast, it reproduces the training data very well, but is less successful for the test set due to overfitting. To overcome this issue, we applied early stopping to the Deep Autoencoder, and also tried using a Denoising Deep Autoencoder. The latter produces training and validation losses that are similar to those from the Deep Autoencoder, but it reproduces the test set better. Still, this model showed some signs of overfitting, which is why early stopping is applied again. This resulted in minimal improvements, implying that the added noise already resolved most of the overfitting.

When investigating the ability of PCA and autoencoders to retain the structure of the original data set, it can be seen that all three models capture the data structure well. On the training data, the Denoising Deep Autoencoder has the worst performance, while this method performs best for the test data. The Deep Autoencoder has performance that is similar to that of PCA. Overall, however, the differences between the measures are very small and hence there is no reason to believe that one of the models completely outperforms the others on this aspect.

The original data contains labels for each observation that correspond to the letter shown in the image. This allows for measuring the performance of various clustering algorithms on both the original as well as the reduced-space data. When reducing the data to a two-dimensional space with t-SNE, another dimensionality reduction technique, it seems that the different signs do not naturally cluster together in the original data set. This finding is confirmed by the performance of the K-means and K-medoids clustering algorithms, where even for the full data, these algorithms do not create meaningful clusters. Comparing this performance to that of the low-dimensional embeddings, we see that PCA produces worse clusters than the full data set, but the Denoising

Deep Autoencoder always outperforms it. The Deep Autoencoder is more successful than PCA, but does not always outperform the full data. On the test set, both autoencoders perform well, implying that these are able to capture information relevant to clustering analysis well.

The aim of this paper is to answer the research question:

How do PCA and autoencoders compare in the dimensionality reduction task for a MNIST data set of ASL?

We can answer this question by first noting that both techniques are able to create an embedding of the data that retains the original structure very well. Moreover, both autoencoders, especially the Denoising Deep Autoencoder, are able to capture the information needed to perform well in clustering algorithms. Looking at a possible accuracy-efficiency trade-off, one could argue that it is better to use PCA in some cases, because it is computationally easier. This holds specifically in cases where the embedding is not used for clustering. Overall, it can be concluded that both methods can be successfully applied to reduce the dimensionality of a MNIST data set of ASL, and that comparisons between the methods mostly depend on the goal of the reduction.

6 Discussion

6.1 Limitations

A first limitation of this research relates to the intrinsic dimensionality. As discussed in Section 3.2, there is no definitive way of estimating it and we were only able to find a single robust estimator for it. Ideally, we would have had several different estimators pointing towards using d=13 in our research, as this would increase the confidence in this estimator.

Moreover, the fact that clustering algorithms do not perform well, even on the original data, can be argued to be a limitation. Despite the low performance, we can still compare across data representations, but it is not ideal. However, the two-dimensional visualization obtained with t-SNE suggests that the low clustering performance does not relate to ill-performing algorithms. Instead, it seems that the data **at hand** (pun intended) is simply not suitable for proper clustering.

The main limitations of this paper relate to the data properties. Earlier research on the topic of dimensionality reduction often uses MNIST data containing binary values. Since the images from our data set are gray-scaled, we work with a much larger range of data values. By adding contrast, we are able to overcome some of the issues created by this property of the data, but it

remains difficult to actually find meaningful results. Moreover, since the original data set is created in such a way that there are many similar pictures included, it might be difficult to generalize results to other sets of images showing hand signs.

6.2 Suggestions for Further Research

Many limitations of this data can be related to the properties of the data used in the research. While earlier research shows that the methods applied here work very well on the 'Hello World' MNIST data set, our results are not fully satisfactory. It would be of interest to see if the data set used can be transformed to being binary, such that it resembles the standard MNIST data better, and to investigate whether this transformation yields more satisfactory results.

Besides the data being gray-scaled, we also suspect that the way the data set is constructed influences results. In fact, the data set is constructed from only 1704 pictures of hands. By applying different filters and rotations, the large data set is created, but this data set might not be representative for data of actual different hands performing ASL. Therefore, we suggest to apply the same methods to different images of hand signs to see if the results are similar.

With respect to autoencoders, the limited time available did not allow for exploring all available options. In the cross-validated Random Grid Search, we set the maximum number of investigated combinations of hyperparameters to 30 in order to reduce computational time. In fact, each combination took approximately one hour to run. It would thus be suggested to perform a more thorough parameter search. Besides, there exist many models that could overcome overfitting other than Denoising Autoencoders, such as Variational Autoencoders (Kingma & Welling, 2013). We therefore suggest studying whether these models can further improve the performance.

Finally, we were unable to compare between all different models and methods in this research. For further research, it would be interesting to apply a variety of other dimensionality reduction techniques to the data set to find whether these are able to capture it properties better than PCA and autoencoders. Besides, one could investigate whether more advanced clustering algorithms find more meaningful clusters in this data. Finally, training a classifier to the data and its low-dimensional embeddings could give more insight into whether it is possible for a computer to distinguish between hand signs.

References

Angiulli, F., & Pizzuti, C. (2005). Outlier mining in large high-dimensional data sets. *IEEE Transactions on Knowledge and Data Engineering*, *17*(2), 203–215.

Arthur, D., & Vassilvitskii, S. (2006). *k-means++: The advantages of careful seeding* (Tech. Rep.). Stanford.

Bac, J. (2020). *Scikit-dimension: Intrinsic dimension estimation in Python.* Retrieved from: https://scikit-dimension.readthedocs.io/.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, *2*(1), 53–58.

Bellman, R. E. (1961). *Adaptive control processes: a guided tour*. Princeton University Press.

Bengio, Y., Yao, L., Alain, G., & Vincent, P. (2013). Generalized denoising auto-encoders as generative models. *arXiv preprint arXiv:1305.6663*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(2).

Betechuoh, B. L., Marwala, T., & Tettey, T. (2006). Autoencoder networks for hiv classification. *Current Science*, 1467–1473.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*(4), 291–294.

Bradley, P. S., Bennett, K. P., & Demiriz, A. (2000). Constrained k-means clustering. *Microsoft Research, Redmond*, *20*(0), 0.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Celik, T. (2012). Two-dimensional histogram equalization and contrast enhancement. *Pattern Recognition*, *45*(10), 3810–3824.

DeMers, D., & Cottrell, G. W. (1993). Non-linear dimensionality reduction. In *Advances in neural information processing systems* (pp. 580–587).

Eckmann, J. P., & Ruelle, D. (1992). Fundamental limitations for estimating dimensions and lyapunov exponents in dynamical systems. *Physica D: Nonlinear Phenomena*, *56*(2-3), 185–187.

Fukunaga, K. (2013). *Introduction to statistical pattern recognition.* Elsevier.

Fukunaga, K., & Olsen, D. R. (1971). An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, *100*(2), 176–183.

Grassberger, P., & Procaccia, I. (2004). Measuring the strangeness of strange attractors. In *The theory of chaotic attractors* (pp. 170–189). Springer.

Han, J. (2001). Spatial clustering methods in data mining: A survey. *Geographic Data Mining and Knowledge Discovery*, 188–217.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Hartigan, J. A., & Wong, M. A. (1979). A k-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *28*(1), 100–108.

Hinton, G. E., & Roweis, S. T. (2002). Stochastic neighbor embedding. In *Nips* (Vol. 15, pp. 833–840).

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

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417.

Jackson, S. (2018). *Coranking library in Python.* Retrieved from: https://coranking.readthedocs.io/.

Jayalakshmi, T., & Santhakumaran, A. (2011). Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, *3*(1), 1793–8201.

Kambhatla, N., & Leen, T. K. (1997). Dimension reduction by local principal component analysis. *Neural Computation*, *9*(7), 1493–1516.

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: an introduction to cluster analysis* (Vol. 344). John Wiley & Sons.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Lee, J. A., & Verleysen, M. (2009). Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, *72*(7-9), 1431–1443.

Leung, C. K., & Joseph, K. W. (2014). Sports data mining: predicting results for the college football games. *Procedia Computer Science*, *35*, 710–719.

Levina, E., & Bickel, P. J. (2005). Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems* (pp. 777–784).

Li, P., & Nguyen, P.-M. (2018). On random deep weight-tied autoencoders: Exact asymptotic analysis, phase transitions, and implications to training. In *International conference on learning representations.*

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2009). The planar k-means problem is np-hard. In *International workshop on algorithms and computation* (pp. 274–285).

Montavon, G., Samek, W., & Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, *73*, 1–15.

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., & Kavukcuoglu, K. (2016). Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rosenberg, A., & Hirschberg, J. (2007). 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)* (pp. 410–420).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sign language MNIST, Version 1. (2017). Retrieved from: https://www.kaggle.com/datamunge/sign-language-mnist.

Siuly, S., & Zhang, Y. (2016). Medical big data: neurological diseases diagnosis through medical data analysis. *Data Science and Engineering*, *1*(2), 54–64.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing*, *30*(8), 2133–2147.

Theis, L., Shi, W., Cunningham, A., & Huszár, F. (2017). Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*.

Van der Maaten, L., & Hinton, G. E. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimensionality reduction: a comparative. *Journal of Machine Learning Research*, *10*(66-71), 13.

Venna, J., & Kaski, S. (2006). Local multidimensional scaling. *Neural Networks*, *19*(6-7), 889–899.

Verveer, P. J., & Duin, R. P. W. (1995). An evaluation of intrinsic dimensionality estimators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(1), 81–86.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, *11*(12).

Wang, Y., Yao, H., & Zhao, S. (2016). Auto-encoder based dimensionality reduction. *Neurocomputing*, *184*, 232–242.

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. Retrieved from `https://doi.org/10.21105/joss.03021`

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Ye, P., Qian, J., Chen, J., Wu, C.-h., Zhou, Y., De Mars, S., ... Zhang, L. (2018). Customized regression model for airbnb dynamic pricing. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining* (pp. 932–940).

Appendices

A Example pictures in MNIST ASL data set

Figure A.1: Example of pictures from MNIST data set of American Sign Language

B Backpropagation Algorithm

Algorithm 1: Backpropagation

Result: Updated parameters

Initialize all parameters (weights and biases) using random values;

while *iteration* $<$ *maximum number of iterations* **do**

  for *each batch in the training set* **do**

    Feed the input to the network and calculate the loss function;

    for *each layer from the first hidden layer* **do**

      for *each node* **do**

        Calculate the weighted sum from the previous layer;

        Add the bias;

        Apply the activation function;

      end

    end

    Calculate the specified loss function;

    Propagate the errors backward;

    for *each layer from the output layer* **do**

      Compute gradients of hidden layer parameters;

    end

    Apply optimization algorithm to update all parameters in the model

  end

end

C *K*-means algorithms

Algorithm 2: K-means

Select K points as the initial centroids: $\mathbf{C}= \{c_1, c_2, ..., c_K\}$;

repeat

  Assign each data point to the closest c_J;

  Recompute c_j to be the mean of the j-th cluster;

until ***C*** *does not change*;

Algorithm 3: K-means++ initialization (Arthur & Vassilvitskii, 2006)

Select a random point z_i from the data as the first centroid c_1;

repeat

  Choose the next center c_j, picking $z_i \in \mathbf{Z}$ with probability $\frac{\text{dist}(z_i,c_j)^2}{\sum_{z \in \mathbf{Z}} \text{dist}(z_i,c_j)^2}$;

until K *centroids are chosen*;

Continue as with the standard K-means algorithm;

D *K*-medoids algorithm

Algorithm 4: K-medoids

Select K points from the data as initial medoids;

Associate each data point to the closest medoid;

while *the cost decreases* **do**

  for *each medoid* m *and each non-medoid* o **do**

    Swap m and o, assign each data point to the closest medoid and recompute the cost;

    if *total cost is more than in the previous step* **then**

      Undo the swap;

    end

  end

end

E t-SNE

Algorithm 5: Simple t-SNE algorithm (Van der Maaten & Hinton, 2008)

cost function parameters: perplexity $Perp$,

optimization parameters: number of iterations, learning rate η, momentum $\alpha(t)$.

Result: two-dimensional data representation $\mathbf{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$

begin

compute pairwise affinities $p_{j|i}$ with perplexity $Perp$;

set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$;

set a sample initial solution $\mathbf{Y}^{(0)}$;

for *iteration* $<$ *maximum number of iterations* **do**

compute low-dimensional affinities q_{ij};

compute gradient $\frac{\delta C}{\delta Y}$;

set $\mathbf{Y}^{(t)} = \mathbf{Y}^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(\mathbf{Y}^{(t-1)} - \mathbf{Y}^{(t-2)})$

end

end

F PCA Image Reconstruction

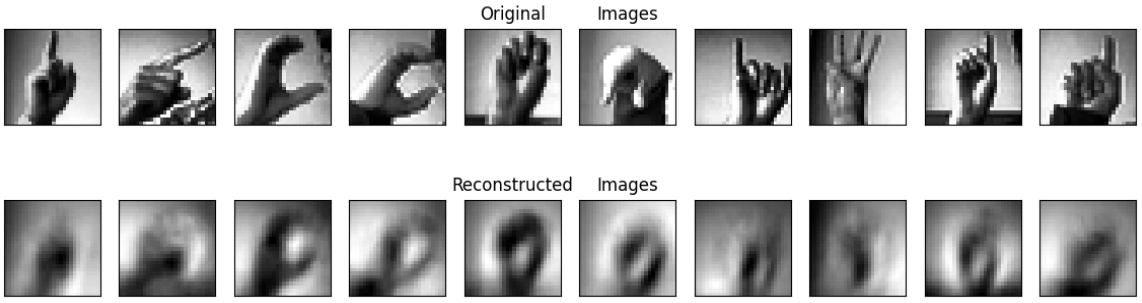


Figure F.1. PCA image reconstruction