The performance of various feature extraction methods in dimensionality reduction in the MNIST handwritten dataset

CHUN-HUA, SHIH
112423004
a29424315@gmail.com

CHIA-HENG, KUO
112423066
Henryedu0904@gmail.com

TING-LIN, KUO
112423061
ks05196618@gmail.com

ABSTRACT

This paper provides a comprehensive review of dimensionality reduction techniques used for feature extraction in machine learning and deep learning. It covers linear methods such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA), as well as non-linear approaches like t-distributed Stochastic Neighbor Embedding (t-SNE) and autoencoders. Each technique's principles, implementation details, advantages, and limitations are discussed. Trends over the past decade indicate an increasing focus on non-linear methods, particularly autoencoders and t-SNE.

An experiment comparing the performance of PCA and autoencoders on the MNIST handwritten digit dataset is conducted. Both techniques extract 36-dimensional features, which are reconstructed and input into classifiers. Results show PCA outperforms basic autoencoders in image reconstruction and digit recognition accuracy. However, integrating robustness loss significantly enhances autoencoder performance. This abstract concludes by highlighting the importance of dimensionality reduction in enabling meaningful analysis of high-dimensional data and suggests avenues for further research in optimizing these methods for complex data.

KEYWORDS

dimensionality reduction; feature extraction; principal component analysis (PCA); linear discriminant analysis (LDA); t-SNE; autoencoders; deep learning; handwritten digit recognition; MNIST dataset

1. INTRODUCTION

When collecting raw data or using existing datasets, we often encounter issues such as missing values, noise, data corruption, high dimensionality, and irrelevant attributes. Therefore, the data before data mining becomes particularly important. Data preprocessing techniques include dimensionality reduction, instance selection, discretization, data normalization, and more.

Dimensionality reduction techniques can effectively reduce the complexity of high-dimensional data. This reduces data redundancy, making it easier to analyze. Instance selection removes irrelevant or repetitive data points, improving the overall quality of the data. Converting continuous values into discrete categories, known as discretization, helps to improve the efficiency of algorithms. Data normalization addresses outliers and extreme values, creating a more normalized data distribution. Through data preprocessing, we get a better understanding of the nature of the data and can conduct more meaningful data extraction and analysis to extract meaningful insight and representative knowledge.

Within the field of data preprocessing, two common dimensionality reduction methods are feature selection in machine learning and feature extraction in deep learning. Feature selection involves selecting a subset of the most important and relevant features of the dataset to build machine learning models. It typically includes three methods: filter, wrapper, and embedded approaches. Feature extraction, on the other hand, refers to transforming useful features or attributes from raw data into a more understandable, manageable, and analyzable form. Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Independent Component Analysis (ICA) are classical methods for feature extraction that help machine learning models better understand the data and make effective predictions or classifications.

In some machine learning models, data which has high dimensionality can pose challenges to accuracy in classification, pattern recognition, and visualization. Performing calculations in high-dimensional space may become difficult due to the complexity of the data, potentially leading to issues like the curse of dimensionality and overfitting. Dimensionality reduction techniques are used to reduce the dimensions of data while preserving its essential information, aiming to mitigate computational costs, improve model efficiency, reduce the risk of overfitting, and enhance data understanding.

As an integral part of machine learning in data preprocessing, dimensionality reduction plays a crucial role. There exist various feature extraction methods based on different principles, such as linear-based and deep learning-based approaches. However, whether these methods lead to significant differences in performance on the same dataset or whether a particular feature extraction method is particularly suitable for a specific domain of classification prediction remains a question. Therefore, this study will focus on exploring the performance of various dimensionality reduction methods in feature extraction on handwritten datasets and has designed an experiment.

Our experimental design is based on the experiment in one of the papers. To follow the paper, we also selected MNIST's handwritten pictures as the experimental dataset. And then fed the dataset into an autoencoder and PCA for dimensionality reduction. After that, we observe the results of feature extraction by the two methods and compare whether the new images generated are identifiable. Furthermore, we compare the accuracy, precision, and recall rate of the two through the classifier to check whether the optimization can be achieved. at baseline. We hope to find better dimensionality reduction and classifier methods through this experiment.

2. LITERATURE REVIEW

2.1 Principle of Feature Dimensionality Reduction

When we say a dataset has high-dimensional data, it is said that there are many features of the dataset. Due to advances in technology, high-dimensional data are ubiquitous and frequently encountered by statistical scientists and engineers. [1]

However, this data often contains a large amount of redundant information, including related or repeated elements. At this time, these interferences can be eliminated through dimensionality reduction. Feature dimensionality reduction uses existing feature parameters to form a low-dimensional feature space and overcomes the effects of unnecessary or unrelated information to map the effective information contained in the original features to fewer features. [2]

In the mathematical sense, suppose that there is an n-dimensional vector.

$$X = [x1, x2 \dots x_n]^T \quad (1)$$

X are mapped to a m-dimensional vector Y through a map f , and m << n.

$$Y = [y1, y2 \dots y_m]^T \quad (2)$$

This is the principle of dimensionality reduction, and the low-dimensional vector after dimensionality reduction can represent a certain proportion of the original data before dimensionality reduction.

We can roughly divide feature extraction algorithms for dimensionality reduction into two categories: linear and nonlinear. In the early days, linear methods were mostly studied, and in recent years, nonlinear methods have shown an increasing trend.

2.2 Linear Dimensionality Reduction Techniques

2.2.1 PCA

Principal Component Analysis (PCA) stands out as one of the earliest and most widely utilized techniques, serving as an unsupervised linear method for dimensionality reduction. The primary concept of PCA is to reduce the dimensionality of a dataset through linear transformations while preserving as much of the original feature variance as possible. This involves discovering new variables in the original dataset that are linear functions of the original variables. These linear functions, known as principal components, are uncorrelated with each other while maximizing variance. The applications of PCA span various fields, including machine learning, text mining, image and speech processing, computer vision, visualization, robotic sensor data analysis, biometrics, facial recognition, and more. [1]

The steps of PCA are as follows, assuming that a dataset $x^1, x^2 \dots x^m$ has n-dimension data has to be reduced to k-dimension ($k << n$) using PCA:[2]

1) Standardization: Transform the data into a standardized form to eliminate the influence of scale differences between different features or dimensions on PCA calculations.

$$x_j^i = \frac{x_j^i - \mu_j}{\sigma_j}, \forall j \quad (3)$$

2) Compute the covariance matrix based on the original data.

$$\sum \frac{1}{m} \sum_i^m (x_i)(x_i)^T, \sum \in R^{n*n} \quad (4)$$

3) Compute the eigenvectors and eigenvalues of the covariance matrix.

$$u^T \sum = \lambda \mu$$

$$U = \begin{bmatrix} | & \cdots & | \\ u_1 & \ddots & u_n \\ | & \cdots & | \end{bmatrix}, u_i \in R^n \quad (5)$$

4) Select the top *m* eigenvectors of the covariance matrix. These eigenvectors will serve as the basis for the data in the new feature space, projecting the original data into a *k*-dimensional subspace.

$$x_i^{new} = \begin{bmatrix} u_1^T x_i \\ \cdots \\ \cdots \\ u_k^T x_i \end{bmatrix} \in R \quad (6)$$

However, PCA also has some limitations. First, due to its reliance on second-order statistics, even though the principal components are statistically uncorrelated, they may still exhibit high statistical dependence, as PCA focuses on describing linear structures in the data and ignores potential nonlinear relationships. If the data components have nonlinear dependencies, PCA may not provide a complete and accurate representation of the data. Secondly, because PCA uses the least squares method, which is insensitive to outliers, it may lead to performance degradation in the presence of outliers.[1] Lastly, as PCA is a non-parametric method, it may yield suboptimal results when the data has specific structures or patterns.[3]

Therefore, PCA has developed many extensions to address these challenges and improve efficiency. For example, Robust PCA (RPCA) and Expectation Robust PCA (ERPCA) are suitable for different types and sizes of image data and handle outliers. Local PCA (LPCA) analyzes local neighborhoods within a dataset, suitable for image and speech data, while Sparse PCA (SPCA) deals with sparse data such as gene expression data. The Generalized PCA (GPCA), was developed specifically to handle high-dimensions data with the number of subspaces unknown.[1]

2.2.2 LDA

Linear Discriminant Analysis (LDA) is a supervised machine learning technique utilized for dimensionality reduction and classification tasks. It preserves the class information within the dataset while reducing its dimensions. The primary objective of LDA is to optimize the ratio of inter-class variance to intra-class variance in order to enhance the distinguishability of different classes. [4] LDA finds applications in a diverse range of fields, such as bioinformatics for gene expression analysis, marketing for customer segmentation, finance for credit risk assessment, astronomy for stellar spectra classification, and remote sensing for land cover mapping and crop monitoring.

2.2.2.1 Implementation Steps [4]

LDA is a supervised algorithm (unlike PCA); hence, we need to have the target classes specified. The implementation of LDA can be divided into the following steps:

1) Data Preparation

The initial step involves data preparation, which entails dividing the data into a training set and a test set and encoding each data point as a vector. The training set is utilized for training the Linear Discriminant Analysis (LDA) model, while the test set is employed to assess the model's performance. Typically, the data is divided into training and test sets in a 70% to 30% ratio. Each data point is encoded as a vector, with each component representing a specific feature. For instance, in a dataset containing facial images, each data point can be represented as a vector where individual elements correspond to facial features like eye position, nose size, etc.

2) Computing the Mean Vector and Covariance Matrix

The next step is to compute the mean vector and covariance matrix of the data set. The mean vector of a class is the center of the data points of the class. The covariance matrix of the data set represents the relationship between the features of the data set.

3) Projecting the data onto the LDA subspace

The LDA subspace is the direction of the largest variance of the features of the data set. The goal of this step is to project the data onto the LDA subspace.

4) Classification

The final step is to classify the data points using the LDA subspace.

The advantages of LDA include handling multiple dependent variables, reducing error rates, and facilitating easier interpretation of between-group differences. [3] While LDA is widely used for data reduction, it faces limitations. When the dimensionality of the data matrix is much higher than the number of samples, LDA struggles to identify a lower-dimensional space. As a result, the within-class matrix becomes singular. This challenge referred to as the Small Sample Size (SSS) problem. [1]

To address this issue, researchers have suggested several strategies. One approach involves eliminating the null space of the within-class scatter matrix. Alternatively, an intermediate dimensionality reduction technique like PCA can be employed to first transform the data into a full-rank space before applying LDA. Regularization methods that stabilize the solution of singular linear systems have also been explored. As a result, numerous extensions to LDA have emerged to tackle the SSS issue, including regularized LDA (RLDA), direct LDA (DLDA), null space LDA, generalized LDA (GEDA), kernel discriminant LDA (KDLDA), and hybrid techniques combining PCA and LDA. [1]

2.2.2.2 Comparison between PCA and LDA

- PCA is an unsupervised dimensionality reduction algorithm, while LDA is a supervised dimensionality reduction algorithm. [5]
- PCA reduces dimensionality by maximizing the variance of the data, while LDA reduces dimensionality by maximizing the difference between classes. [6]
- PCA is often used for exploratory data analysis and preprocessing of data for machine learning algorithms, while LDA is often used for classification and feature selection. [7]

2.3 Non-linear Dimensionality Reduction Techniques

2.3.1 t-SNE

The t-distributed stochastic neighbor embedding (t-SNE) is a non-linear technique for reducing the dimensionality of data, introduced by Laurens van der Maaten and Geoffrey Hinton in 2008. [6] This approach aims to identify non-linear structures within data by leveraging local neighborhoods, geodesic distances, and other graph-theoretic principles. [7]

The features of t-SNE are as follows:

- Randomness:

The t-SNE algorithm has randomness. Compared with the linear method of PCA, the experimental results of t-SNE are different each time. [7]

- Interpretability:

When interpreting the results of t-SNE, we should pay special attention to the following:

First, it is meaningless to compare the sizes of different stacks because t-SNE adjusts the block size according to the congestion of the neighbors. Secondly, it is meaningless to compare the distance between stacks because t-SNE is not a cluster that is close to each other. Much like each other, distance in this embedding space has no intuitive properties of distance. Finally, t-SNE cannot be used to find outliers because the process of t-SNE will bring the outliers closer to a certain group, and the outlier characteristics will not be obvious. [7]

Although t-SNE is an improved method of SNE. t-SNE still has some challenges to conquer. There are three points:

- Cost of Computation:

The computational cost of t-SNE is higher than that of linear dimensionality reduction methods such as PCA. This may pose a constraint for datasets that are extremely large.[8]

- Parameter tuning:

The selection of hyperparameters (such as perplexity) can have an impact on how well t-SNE performs. It frequently takes careful tweaking to get desirable results.

- Problems with Visualization:

Interpreting the sizes and distances between clusters in the low-dimensional embedding can be challenging, as you pointed out. t-SNE is not as well suited for accurate distance measurements as it is for the qualitative investigation of data patterns.

2.3.1.1 SNE Algorithm

Stochastic Neighbor Embedding (SNE) is a powerful algorithm used for dimensionality reduction and visualization of high-dimensional data in a lower-dimensional space. Developed by Geoffrey Hinton and Sam Roweis in 2002. At its core, SNE operates by modeling the similarity between pairs of data points in both the original high-dimensional space and the lower-dimensional embedding space. The algorithm aims to preserve local relationships, ensuring that nearby points in the original space remain close together in the lower-dimensional representation.

The key idea behind SNE is to define a probability distribution over pairs of data points in a high-dimensional space, representing the similarity between them. This probability distribution is then transformed into a similar distribution in lower-dimensional space. The transformation is achieved by minimizing the Kullback-Leibler (KL) divergence between the two distributions, effectively ensuring that similar points are mapped to nearby locations in the lower-dimensional embedding.

One notable feature of SNE is its ability to handle non-linear relationships and complex data structures. By focusing on local relationships and adaptively adjusting the embedding to preserve them, SNE can reveal intricate patterns and clusters within the data.

However, SNE's performance can be sensitive to its hyperparameters, particularly the perplexity parameter, which controls the number of neighbors considered for each data point during the embedding process. Selecting an appropriate perplexity value is crucial for obtaining meaningful embeddings. After a few years, they provide a new method called t-SNE, which we introduced above. [8]

2.3.1.2 Implementation

t-SNE assumes that the data is a low-dimensional manifold uniformly sampled in a high-dimensional Euclidean space. The purpose is to solve the crowding problem at SNE. t-SNE adopts two improved methods, namely simplifying the gradient formula and switching to the t distribution instead of the original Gaussian distribution in the low-dimensional space. The advantage of expressing it in terms of the t distribution is that the t distribution is biased toward the long tail of the Gaussian distribution, so that sample points with medium and close distances in high dimensions can have a larger distance after mapping, and the t distribution is less affected by outliers. When the number of samples is small, the distribution can still be simulated without being affected by noise. [9]

t-SNE first calculates pairwise similarities between all data points in a high-dimensional space with a Gaussian kernel, capturing local

relationships that are crucial for understanding the structure of the data set. Afterwards, try to optimize both similarity measures.
In application, t-SNE attempts to map high-dimensional data points to a 2- or 3-dimensional space for qualitative visual observation while retaining pairwise similarity. Finally, gradient descent is used to minimize the divergence. Optimize low-dimensional embeddings to a stable state. [8], [10]

Taken together, t-SNE emphasizes modeling dissimilar datapoints through large pairwise distances and modeling similar datapoints through small pairwise distances. Moreover, due to these characteristics of the t-SNE cost function and the approximate scale invariance of the student t-distribution, optimizing the t-SNE cost function is much easier compared to optimizing the cost functions of SNE. [8]

2.3.2 Autoencoder

Autoencoder (AE) is a symmetrical neural network structure in which the middle layer is a fully connected layer that maps all input neurons to each output neuron. Autoencoder is indeed broad and encompasses various learning models based on fully connected feedforward artificial neural networks and other types of artificial neural networks, even those unrelated to this structure. [11]

Autoencoders and their variants are commonly used as feature extraction methods. Through training neural network models, they learn implicit feature representations in the data, which can be applied to scenarios such as dimensionality reduction, feature extraction, and generative modeling. Based on different design philosophies and model structures, autoencoders can be categorized into several types, as shown in Figure 1.

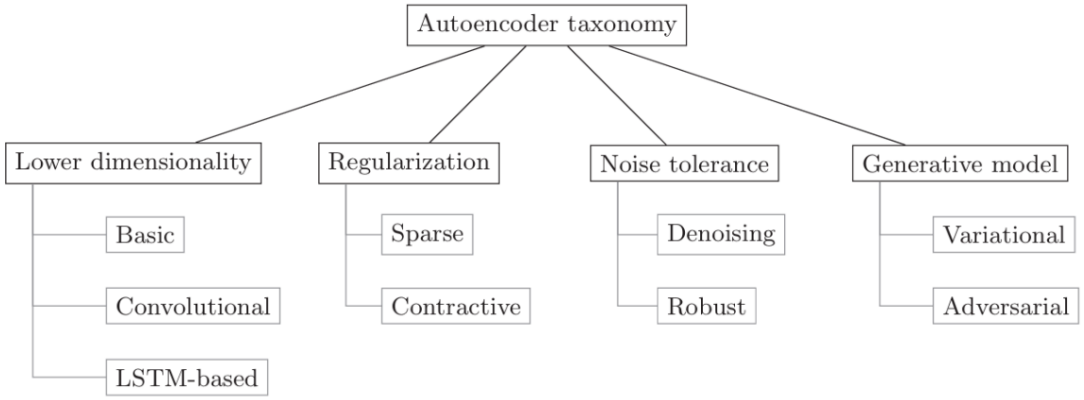


Figure 1. Autoencoder Taxonomy

- Basic Autoencoder

Basic Autoencoder (AE) [11] with symmetric structure is an unsupervised method of neural network structure. Obtain some representative features by compressing the raw data, learn from them, and try to reconstruct the original input from a small number of features. Its architecture consists of two main parts, as illustrated in Figure 2.

The first part is the input data. Input data can be structured or unstructured. The second part is the encoder, responsible for encoding the original input data to compress it into a lower-dimensional representation. The last part is the decoder, which attempts to reconstruct the low-dimensional features back to the original input as closely as possible. The AE can have as many layers as needed, just placed symmetrically in the encoder and decoder. In addition, except for the input and output layers, the others will be called hidden layers.
During the training period, the autoencoder aims to teach the encoder to better compress features, while the decoder learns to reconstruct features, ultimately minimizing the discrepancy between the original input and the reconstructed input. For an autoencoder, using the compressed representation for training and recognition purposes constitutes feature extraction.

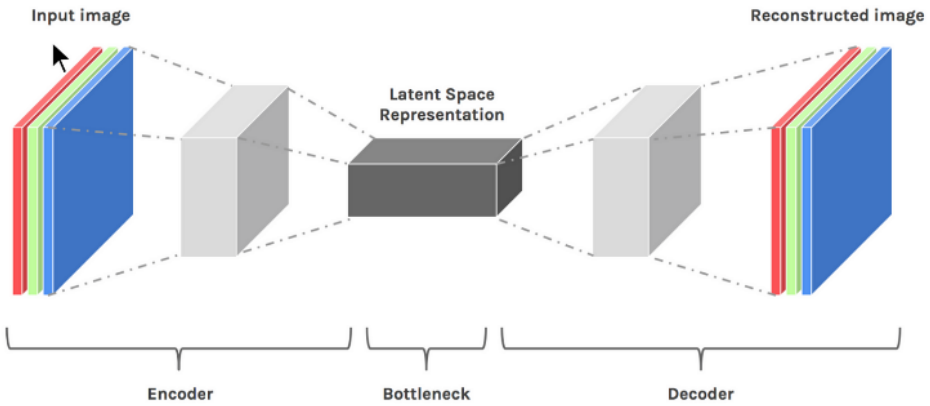


Figure 2. Autoencoder structure (source: medium)

- Sparse Autoencoder

Compared to traditional autoencoder, Sparse Autoencoder (SAE) [12] adds sparsity constraints to the loss function. Its architecture is shown in Figure 3. This allows the encoder to learn a more concise feature representation (latent space), thereby enhancing the model's generalization ability. Sparsity loss can be achieved using L1 regularization, which sets many weights in the feature representation to 0, effectively deactivating certain nodes (white). Additionally, it penalizes a large amount of useless information in the feature representation, effectively penalizing certain nodes (gray). This forces SAE to learn features without reducing the number of nodes, enabling it to effectively learn more useful feature representations.

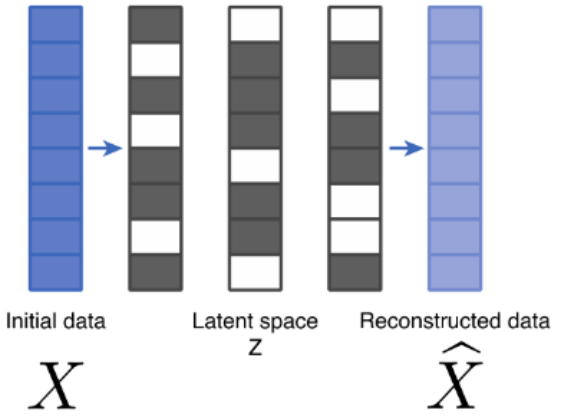


Figure 3. Sparse Autoencoder (source: [13])

- Denoising Autoencoder

A Denoising Autoencoder (DAE) [14] is specifically designed to learn robust feature representations by removing noise from input data. Its architecture is shown in Figure 4. In a standard autoencoder, the model learns to reconstruct clean input data from noisy input data, which can lead to overfitting and may not generalize well to unseen data.

In contrast, a Denoising Autoencoder is trained to reconstruct clean input data from corrupted or noisy input data. During training, noise is intentionally added to the input data, and the model is trained to minimize the reconstruction error between the denoised output and the original clean input. By doing so, the Denoising Autoencoder learns to extract meaningful features that are resilient to noise and can effectively denoise input data.

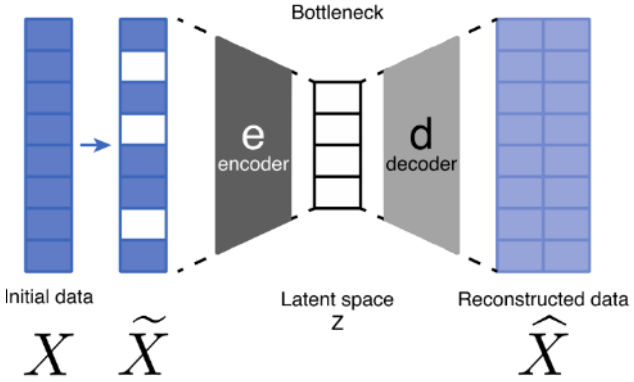


Figure 4. Denoising Autoencoder (source: [13])

- Variational Autoencoder

The Variational Autoencoder (VAE) , as illustrated in Figure 5, differs from traditional autoencoders in that the encoder of VAE does not directly produce feature representations (latent space), but instead generates a set of parameters for a probability distribution. These distributions can be used to generate latent encodings. Specifically, the encoder in VAE produces a set of mean vectors and variance vectors, and uses them to generate latent encodings. To ensure that the generated encodings adhere to a standard normal distribution, VAE restricts the shapes of these distributions through Kullback-Leibler (KL) divergence, thereby modeling the latent distribution of the data. Its main advantage is that VAE can generate new data samples, and the generated samples exhibit diversity. That is, VAE can control the latent features of the data, thereby achieving data transformation.

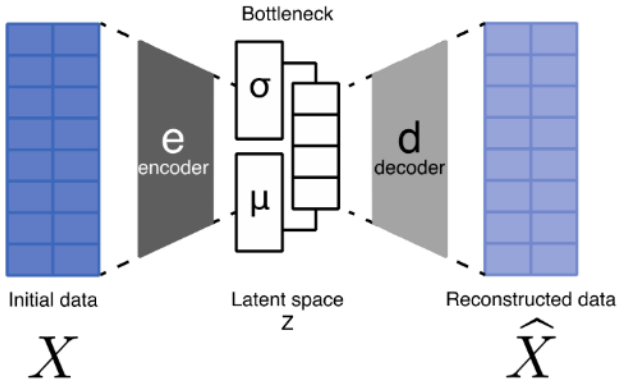


Figure 5. Variational Autoencoder (source: [13])

- Robust Autoencoder

The main feature of a Robust Autoencoder (RAE) [11] is its enhanced robustness to noise and outliers in input data. In traditional autoencoders, the model's objective is to encode input data into a compressed representation and then decode it back to a reconstructed form similar to the original data. However, when there is noise or outliers in the data, traditional autoencoders are susceptible to distortion or inaccuracies in the reconstructed data.

To address this issue, Robust Autoencoder introduces robustness loss functions, which exhibit stronger tolerance to outliers and noise. Typically, these loss functions constrain the difference between the reconstruction and the original data within a reasonable range, allowing the model to better adapt to the variability in the input data.

3. RESULT

3.1 The Trends on Discussing Dimensionality Reduction

We reviewed a total of 20 pieces of feature extraction literature in the past 10 years and counted the trends of different dimensionality reduction methods based on whether each paper discussed the topic.

As shown in Figure 6, linear dimensionality reduction techniques predominantly employ two methods for dimensionality reduction: PCA and LDA. The former is the more popular one because a total of 17 papers discussing dimensionality reduction mention PCA. Non-linear Dimensionality Reduction Techniques typically employ three methods for dimensionality reduction: t-SNE, Autoencoder, and NMF, with t-SNE and Autoencoder being the more frequently discussed methods.

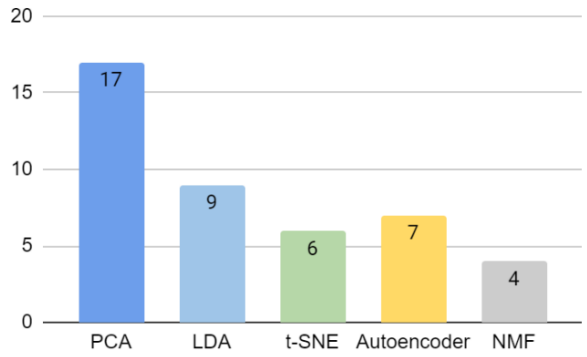
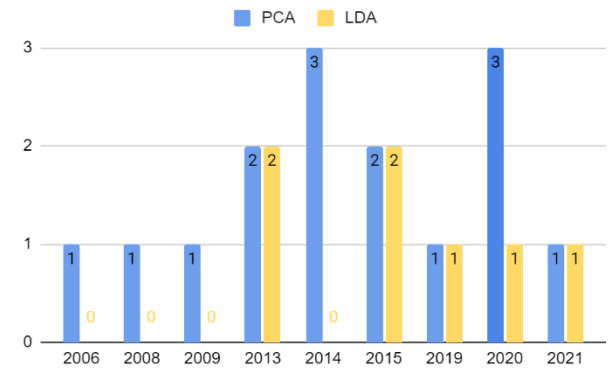


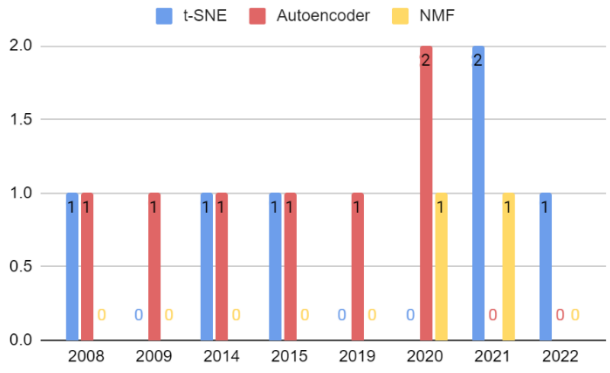
Figure 6. Statistics on the number of papers discussing dimensionality Reduction

Then, based on statistics from different years, we tried to find out the changes in trends in recent years. As shown in Figure 7, we can see that PCA will definitely be discussed in papers published every year, but LDA will not necessarily be discussed. Among nonlinear methods, the NMF method is less popular, while autoencoder and t-SNE have an increasing trend.

However, since the number of observed papers is far less than the number of papers published by publishers each year, This finding may not be accurate because there may be other unknown situations that have not been discovered.



(a) Linear Dimensionality Reduction Techniques



(b) Non-linear Dimensionality Reduction Techniques

Figure 7. Dimensionality Reduction paper statistics in different years

3.2 Dataset

3.2.1 What is MNIST?

The MNIST dataset, short for the Modified National Institute of Standards and Technology database[handwrite], stands as a pivotal resource in the realm of image processing and machine learning research. Primarily, MNIST is renowned for its extensive collection of handwritten digits, serving as a cornerstone for training and evaluating various algorithms in these domains.[15]
Initially derived by amalgamating samples from MNIST's original datasets, MNIST underwent significant preprocessing. This entailed standardizing the black and white images to a 28x28 pixel frame and applying anti-aliasing techniques, thereby introducing grayscale variations. [16]

Comprising a total of 70,000 images, the MNIST dataset is divided into 60,000 training images and 10,000 testing images. [17] Notably, each subset draws from MNIST's training and testing datasets, ensuring a comprehensive representation of handwritten digits for both training and evaluation purposes.

The simplicity and accessibility of the MNIST dataset have made it a go-to resource for researchers and practitioners in the field of machine learning and computer vision. Its well-defined structure and relatively small size allow for efficient training and testing of models, while its diversity in handwriting styles provides a challenging yet manageable task for image recognition algorithms.

Over the years, the MNIST dataset has been extensively used in numerous research projects and applications, ranging from handwriting recognition to neural network development and benchmarking. Its widespread adoption has facilitated the advancement of techniques and methodologies in these fields, serving as a common ground for c

3.2.2 Why is MNIST?

- Simplicity: MNIST consists of 28x28 grayscale images of handwritten digits (0 through 9). Its simplicity makes it easy to understand and work with.
- Availability: MNIST is widely available and can be easily accessed through libraries like TensorFlow, PyTorch, and scikit-learn. It's a standard benchmark dataset often used in tutorials, research papers, and competitions.
- Size: The 60,000 training images and 10,000 test images in MNIST are enough for many learning algorithms to show off their strengths and weaknesses. Its compact size allows for quick model training, which makes it perfect for experimentation.

3.2.3 How to implement?

In the implementation of t-SNE, TensorFlow serves as a powerful framework providing a suite of tools and functionalities. With TensorFlow, developers can efficiently build and deploy t-SNE models for various tasks such as dimensionality reduction, visualization, and clustering. Firstly, import the necessary libraries and datasets, divide it into training sets and test sets and then implement the method of TensorFlow's rich collection of APIs and utilities. The code can be using tfds.load('MNIST', split=['train', 'test'])" functions, respectively.[18]

3.3 Evaluation metrics

3.3.1 Loss Function

In the model classification task, our main goal is to minimize the difference between the predicted value and the actual value through the prediction of the model. However, in practical applications, due

to the influence of various factors, it is difficult for us to train a completely accurate model, so there is usually a certain degree of error between the predicted value and the actual value. This error is often called a loss. [19]

Mathematically, we can express this loss with the following formula:

$$\text{Loss} = y - \hat{y} \tag{7}$$

Which y represents the actual category and $\hat{y}$ represents the predicted category.

We can get a summary: the performance of the model depends largely on the design of the loss function, because it directly affects the learning and optimization process of the model. In this measurement, smaller loss means better model performance. [20]

According to the Tensorflow document [18], the loss function includes the cross-entropy loss function (Cross-Entropy Loss), the Huber loss function, etc. These loss functions have different advantages and applicability in different tasks and scenarios. As the loss function in this experiment, we use Mean Square Error (MSE). MSE is defined as the average of the squared differences between predicted and true values. [21]The calculation process for MSE is to square the error between the model prediction value and the true value, and then find the average value. The mathematical formula is as follows: [21]

$$MSE=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y_i})^2 \tag{8}$$

By importing MSE, we can more accurately measure the accuracy of model predictions. At the same time, MSE is a commonly used loss function in machine learning. It is characterized by being more sensitive to outliers. It plays a role in balancing the model and the prediction performance on the entire data set during the training process. [20]

3.3.2 Confusion matrix [22]

Confusion matrix is a widely used method to solve classification problems. Table 1 shows an example of a confusion matrix for binary classification.

Table 1. Confusion Matrix for Binary Classification

		Actual Class	
		Positive	Negative
Predicted Class	True	TP	FP
	False	FN	TN

In the confusion matrix, we usually take the model's prediction results and the actual observed sample conditions as two dimensions. Among them, the prediction label represents the model's classification prediction of the data, while the actual samples reflect the true situation of the data. Combining these two dimensions in a positive and negative way forms a two-dimensional 2x2 matrix. [22]

The output of the matrix can be divided into the following four parts, The output "TN" in the confusion matrix represents True Negative, which refers to the number of samples that are correctly classified as negative examples. Similarly, "TP" stands for True Positive, indicating the number of samples that are correctly classified as positive examples. On the other hand, "FP" stands for False Positive, indicating the actual number of negative examples that are incorrectly classified as positive; "FN" stands for False Negative, indicating the actual number of negative examples that are incorrectly classified as negative. The number of positive examples. One of the most commonly used metrics while performing classification is accuracy. Accuracy of a model is calculated using the given formula below. [22], [23]

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{9}$$

In this case, the number of correctly predicted samples refers to the number of samples for which the model's predicted class matches the actual class, while the total number of samples represents the entire dataset's sample size. Although the process of calculating accuracy is very simple, intuitive, and easy for researchers to understand, accuracy is localized, and other indicators may be more useful for evaluating performance. [23]

3.3.3 Precision and recall

Two crucial metrics that are commonly used in classification tasks and provide insightful information about how well machine learning models are performing are precision and recall. Precision measures a model's ability to predict positive values accurately, indicating the percentage of correctly identified positive predictions among all positive predictions. An alternative name for it is Positive Predictive Value (PPV). Conversely, recall measures how well the model captures positive outcomes; it expresses the percentage of correctly identified positive predictions among all real positives. Another name for this metric is sensitivity. The following formulas can be used to calculate precision and recall. [22]

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}\text{ , Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{10}$$

We discovered that each of the three assessment metrics offers insightful data that is useful for assessing model performance. Together, these metrics provide researchers with a better understanding of the advantages and disadvantages of their classification models, allowing for more improvements and well-informed decision-making. [22]

4. DISCUSSION

Based on the literature reviewed in this study, it can be observed that the earliest dimensionality reduction techniques, such as linear PCA and LDA, are still being actively researched and widely applied across various fields. These fundamental linear methods have stood the test of time and continue to be enhanced through the development of numerous extensions tailored to specific applications. However, traditional linear dimensionality reduction methods have limitations when dealing with complex nonlinear structures in data, leading to the emergence of statistical-based nonlinear dimensionality reduction techniques like KPCA and t-SNE. These nonlinear methods have proven more effective in capturing intricate relationships and patterns present in high-dimensional datasets. With the advancements in machine learning and deep learning, recent literature has introduced new techniques and models that combine dimensionality reduction with deep learning. One of these is the Autoencoder, as discussed in this paper, which utilizes deep learning networks to automatically learn feature representations, further enhancing the effectiveness of feature extraction. This represents a significant direction of development in this field currently.

After our research, we found that dimensionality reduction is a critical technique with widespread applications across diverse domains such as image processing, natural language processing, bioinformatics, and many more. Its significance is particularly emphasized in the era of big data, where managing and extracting meaningful insights from vast amounts of high-dimensional data has become paramount. Different domains have unique challenges

and requirements when it comes to dimensionality reduction. As a result, this technology continues to be extensively researched and developed to meet these varied requirements, with a focus on improving performance, scalability, and domain-specific adaptations.

The evolution of dimensionality reduction techniques is driven by several factors, including the increasing complexity and volume of data and the demand for more effective and efficient data analysis methods. Researchers are continually exploring new approaches, such as integrating dimensionality reduction with deep learning architectures, leveraging advances in optimization techniques, and developing hybrid methods that combine the strengths of different algorithms. This ongoing research and innovation effort highlights the significance of dimensionality reduction in modern data analytics, where it plays a crucial role in enabling meaningful insights and discoveries from high-dimensional and complex datasets across various domains. As data grows in size and complexity, developing strong and flexible dimensionality reduction methods will remain a critical focus area. This will encourage cooperation across different fields and drive progress in data-driven solutions.

5. EXPERIMENT

5.1 Experimental Framework

Our experiment is based on one of the papers [10], showing the results of the dimensionality reduction of PCA and Autoencoder on the MNIST handwritten dataset. However, this article does not use a classifier to verify the results after feature extraction, nor does it compare the performance of the two after dimensionality reduction with the original baseline. Therefore, we decided to conduct an experiment to actually compare the performance of the two classifiers. In order to follow the paper, we also selected MNIST handwriting pictures as the experimental dataset. And then we fed the dataset into our chosen feature extraction methods, which are autoencoder and PCA for dimensionality reduction. After being parsed by the encoder, a set of reduced and reconstructed images will be generated, as well as the result of the decoder trying to restore the original image.

As mentioned before, the purpose of the experiment is to observe whether PCA and Autoencoder can find a small number of representative features and to examine whether they can reconstruct the low-dimensional features back to the original image as closely as possible.

5.2 Experimental Procedure

These experiments were conducted in a Python environment using the Tensorflow and Keras libraries..

Step 1:

Using 60 thousand training data and 10 thousand test data from MNIST, each data point is a 28x28-pixel image. In the data pre-processing part, we first converted the image into a one-dimensional vector of 784 elements as input data and then input it into the encoder layers of PCA and Autoencoder, respectively. After feature extraction to 36 dimensions and reconstruction, the final output is returned to a one-dimensional vector of 784 elements, which is then converted back to a 28*28-pixel image.

Step 2:

After extracting features through PCA and Autoencoder respectively, resulting in new images of 36 dimensions. They are then separately input into classifiers for digit recognition prediction. The classifiers used here are KNN and SVM.

5.3 Experimental Results

5.3.1 Images Reconstructed after Dimensionality Reduction

As shown in Figure 8, reconstructed images are obtained by extracting 36-dimensional features through PCA and autoencoders, respectively. Figure 8(a) and Figure 8(b) show the reconstructed images using a basic autoencoder, where the former is from our experimental results and the latter is extracted from the paper. Upon observation, it's evident that the digits reconstructed by the basic autoencoder differ noticeably from the original images, while those from the paper appear slightly clearer. However, it is certain that the basic autoencoder cannot identify key differences based solely on 36 features.

Comparing Figure 8(c) and Figure 8(d), there's little difference between our experimental results and those from the paper. What's more surprising is that the reconstructed images almost perfectly match the original images, with only minor distortions visible. This suggests that incorporating robustness loss functions can effectively improve the performance of the autoencoder. Finally, in Figure 8(e), the images generated by PCA are significantly blurrier compared to those generated by the autoencoder.

5.3.2 Use classifiers to compare the Performance of PCA & Autoencoder after Dimensionality Reduction

In step 2, we used SVM and KNN as our classifiers.

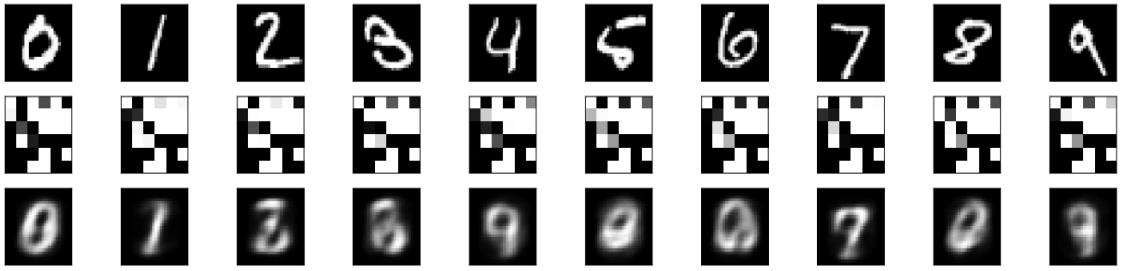
- Accuracy

According to the experimental results, the baseline performance is 0.97, 0.98 accuracy, PCA is 0.98, 0.98, autoencoder is 0.83, 0.82, and robust AE is 0.93, 0.96.

Overall, the accuracy of PCA is comparable to the baseline, but the autoencoder still has some room for improvement. Compared with basic autoencoder, after adding the adjustment of the loss function, it can be observed that Robust autoencoder has obviously greatly improved its performance.

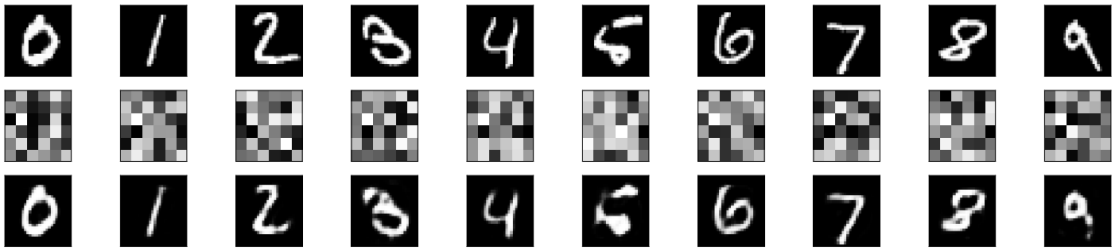
Table 2. Accuracy

	SVM	KNN
Baseline	0.98	0.97
PCA	0.98	0.98
Autoencoder (AE)	0.83	0.82
Robust autoencoder (RAE)	0.93	0.96

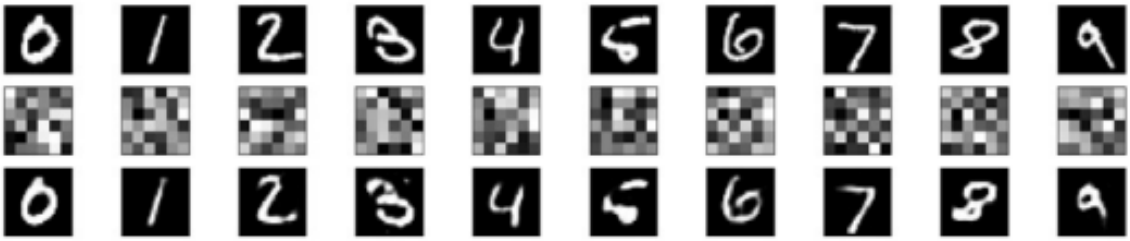


(a) Basic Autoencoder (results come from our experiment)

(b) Basic Autoencoder (results come from referenced papers)



(c) Robust autoencoder (results come from our experiment)



(d) Robust autoencoder (results come from referenced papers)

(e) PCA

Figure 8. Results produced by AE and PCA. Each group shows test samples, activations of the encoding layer and reconstructions.

- Confusion Matrix

Figure 9 shows the confusion matrix of the SVM prediction of baseline. The diagonal line in the matrix is the number of correct predictions. Taking baseline as an example, 1 has 1,126 correct predictions, the number 3 has correct predictions 995 times, and the smaller number is 5, only 871 times. Taking this matrix as an example, we can find that 5 is the most confusing number, while 1 is not easily confused.

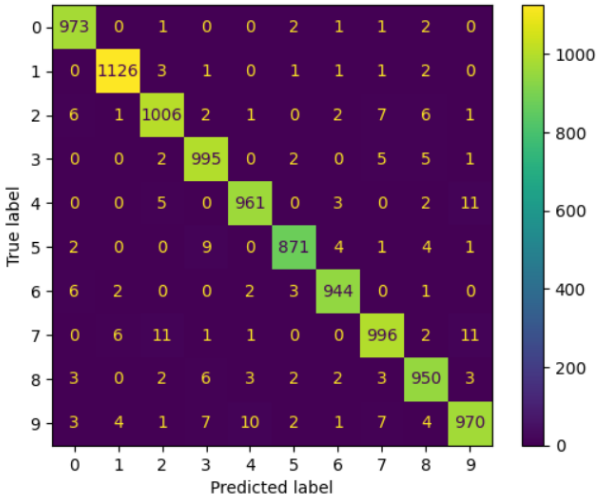


Figure 9. the confusion matrices of the SVM prediction of baseline

- Classification report

The classification report is composed of multiple indicators, which can more comprehensively judge the prediction results of a classifier. Classification reports include precision, recall, and F1 scores for each class. Figure 10 shows the classification report for the SVM predictions of baseline.

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.97	0.98	1032
3	0.97	0.99	0.98	1010
4	0.98	0.98	0.98	982
5	0.99	0.98	0.98	892
6	0.99	0.99	0.99	958
7	0.98	0.97	0.97	1028
8	0.97	0.98	0.97	974
9	0.97	0.96	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Figure 10. the classification report of the SVM prediction of baseline

6. SUMMARY

Based on the content of this experiment, we can draw the following conclusions:

1. PCA outperforms Autoencoder on the MNIST handwritten digit dataset: The experimental results show that PCA's accuracy in reconstructing images after dimensionality reduction is superior to that of Autoencoder. This could be attributed to PCA's ability to more effectively capture the variability of the dataset and retain more useful information, resulting in better performance in digit recognition tasks.
2. Improvement in performance with Robust Autoencoder: The experiment mentions that the performance was significantly improved with the adjusted loss function of the Robust Autoencoder. This indicates that parameter adjustments and optimization within Autoencoder can enhance its representation capabilities for complex datasets, thus improving its performance in digit recognition tasks.
3. Possibilities for further research: Although this experiment provides a preliminary comparison between PCA and Autoencoder on MNIST, there are still many questions to explore further. For example, the generalization of these results can be validated by using larger datasets or other pattern recognition tasks, and further optimization of the Autoencoder model design can be explored to enhance its performance.

In summary, this experiment provides us with insights into the performance comparison between PCA and Autoencoder in handwritten digit recognition tasks and offers valuable references and insights for future related research.

7. REFERENCE

[1] S. Nanga *et al.*, "Review of Dimension Reduction Methods," *JDAIP*, vol. 09, no. 03, pp. 189–231, 2021, doi: 10.4236/jdaip.2021.93013.

[2] G. T. Reddy *et al.*, "Analysis of Dimensionality Reduction Techniques on Big Data," *IEEE Access*, vol. 8, pp. 54776–54788, 2020, doi: 10.1109/ACCESS.2020.2980942.

[3] "A Review of Dimensionality Reduction Techniques," vol. 4, no. 3, 2013.

[4] S. Balakrishnama and A. Ganapathiraju, "Linear Discriminant Analysis—A Brief Tutorial," vol. 11, Jan. 1998.

[5] "What are the key differences between PCA and LDA in Dimensionality Reduction Techniques?" Accessed: Mar. 23, 2024. [Online]. Available: https://www.linkedin.com/advice/0/what-key-differences-between-pca-lda-dimensionality-7memc

[6] A. Kumar, "PCA vs LDA Differences, Plots, Examples," Analytics Yogi. Accessed: Mar. 23, 2024. [Online]. Available: https://vitalflux.com/pca-vs-lda-differences-plots-examples/

[7] S. K. Vungarala, "PCA vs LDA — No more confusion!," Medium. Accessed: Mar. 23, 2024. [Online]. Available: https://medium.com/@seshu8hachi/pca-vs-lda-no-more-confusion-fc21fb8d06e9

[8] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[9] "資料降維與視覺化：t-SNE 理論與應用." Accessed: Mar. 23, 2024. [Online]. Available: https://www.mropengate.com/2019/06/t-sne.html

[10] L. van der Maaten, "Accelerating t-SNE using Tree-Based Algorithms," *Journal of Machine Learning Research*, vol. 15, no. 93, pp. 3221–3245, 2014.

[11] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines," *Information Fusion*, vol. 44, pp. 78–96, Nov. 2018, doi: 10.1016/j.inffus.2017.12.007.

[12] "sparseAutoencoder_2011new.pdf." Accessed: Mar. 23, 2024. [Online]. Available:

https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf
[13] D. Pratella, S. Ait-El-Mkadem Saadi, S. Bannwarth, V. Paquis-Fluckinger, and S. Bottini, "A Survey of Autoencoder Algorithms to Pave the Diagnosis of Rare Diseases," *International Journal of Molecular Sciences*, vol. 22, no. 19, Art. no. 19, Jan. 2021, doi: 10.3390/ijms221910891.
[14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, in ICML '08. New York, NY, USA: Association for Computing Machinery, Jul. 2008, pp. 1096–1103. doi: 10.1145/1390156.1390294.
[15] "Support vector machines speed pattern recognition," Vision Systems Design. Accessed: Mar. 22, 2024. [Online]. Available: https://www.vision-systems.com/home/article/16737424/support-vector-machines-speed-pattern-recognition
[16] C. Yadav and L. Bottou, "Cold Case: The Lost MNIST Digits." arXiv, Nov. 04, 2019. doi: 10.48550/arXiv.1905.10498.
[17] E. Kussul and T. Baidyk, "Improved method of handwritten digit recognition tested on MNIST database," *Image and Vision Computing*, vol. 22, no. 12, pp. 971–981, Oct. 2004, doi: 10.1016/j.imavis.2004.03.008.
[18] "mnist | TensorFlow Datasets." Accessed: Mar. 23, 2024. [Online]. Available: https://www.tensorflow.org/datasets/catalog/mnist
[19] "Amazon.com: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems: 9781492032649: Géron, Aurélien: 圖書." Accessed: Mar. 22, 2024. [Online]. Available: https://www.amazon.com/_/dp/1492032646?smid=ATVPDKIKX0DER&_encoding=UTF8&tag=oreilly20-20
[20] T. Huang, "機器/深度學習: 基礎介紹-損失函數(loss function)." [Online]. Available: https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-%E5%9F%BA%E7%A4%8E%E4%BB%8B%E7%B4%B9-%E6%90%8D%E5%A4%B1%E5%87%BD%E6%95%B8-loss-function-2dcac5ebb6cb
[21] H. Marmolin, "Subjective MSE Measures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 3, pp. 486–489, May 1986, doi: 10.1109/TSMC.1986.4308985.
[22] A. Kulkarni, D. Chong, and F. A. Batarseh, "5 - Foundations of data imbalance and solutions for a data democracy," in *Data Democracy*, F. A. Batarseh and R. Yang, Eds., Academic Press, 2020, pp. 83–106. doi: 10.1016/B978-0-12-818366-3.00005-8.
[23] M. Hofer, G. Strauß, K. Koulechov, and A. Dietz, "Definition of accuracy and precision—evaluating CAS-systems," *International Congress Series*, vol. 1281, pp. 548–552, May 2005, doi: 10.1016/j.ics.2005.03.290.

8. GROUP DIVISION OF WORK

Name	Work description	Contribution
CHUN-HUA, SHIH 112423004	1. Searching Autoencoder topic 2. Collate results 3. Drafting Sections 1, 2.1, 2.3.2, 3.1, 5, 6 4. Devising empirical experiment 5. Implementation (AE) 6. Results & analysis (AE)	33%
CHIA-HENG, KUO 112423066	1. Searching LDA, t-SNE topic 2. Collate results 3. Drafting Sections 1, 2.2, 2.3.1, 3.2, 3.3, 4 4. Devising empirical experiment 5. Proof-reading	33%
TING-LIN, KUO 112423061	1. Searching PCA, LDA topic 2. Collate results 3. Drafting Sections 1, 2.2, 4, 8 4. Devising empirical experiment 5. Implementation (PCA) 6. Results & analysis (PCA)	33%