Kernel Principal Component Analysis for Enhanced Image Compression: A Comparative Study with PCA, Isomap, and LLE

your name

Abstract—**In this paper, I investigate the potential of Kernel Principal Component Analysis (KPCA) for image compression and compare its performance with linear and non-linear dimensionality reduction techniques, namely Principal Component Analysis (PCA), Isomap, and Locally Linear Embedding (LLE). I provide a comprehensive analysis of their strengths, weaknesses, and suitable scenarios for each method. My study includes the implementation of these techniques in the R programming language and their evaluation on two commonly used image datasets: Olivetti Faces and Caltech-101. I also discuss the selection of appropriate kernel functions for KPCA and the impact of hyperparameters on compression performance. My findings reveal the potential of KPCA in certain scenarios for image compression, while also highlighting the challenges faced by Isomap and LLE in this task.**

Index Terms—**image compression, Kernel Function, Principal Component Analysis, Isomap, Locally Linear Embedding, dimensionality reduction**

I. Introduction

Image compression is an essential aspect of digital image processing and communication, aiming to minimize the amount of data required to represent an image while preserving an acceptable level of visual quality [1]. As digital images have become more prevalent in various applications, such as multimedia, web services, and remote sensing, the demand for efficient image compression techniques has grown significantly. This paper will explore various image compression methods, with a particular focus on dimensionality reduction techniques like PCA [2], KPCA [3], Isomap [4], and LLE [5].

There are several common methods of image compression that can be broadly categorized into lossless and lossy techniques. Lossless compression methods ensure that the original image can be perfectly reconstructed from the compressed data without any loss of information. Some popular lossless compression techniques include Run-Length Encoding (RLE) [6], Huffman coding [7], and Lempel-Ziv-Welch (LZW) coding [8]. These methods exploit the redundancy in image data to achieve compression, but the achievable compression ratios are often limited.

Lossy compression techniques, on the other hand, allow for a certain degree of information loss to achieve higher compression ratios. These methods are generally more suitable for natural images, where some loss of image quality can be tolerated. Widely used lossy compression techniques include Discrete Cosine Transform (DCT) [9] used in the JPEG standard, wavelet-based methods like JPEG 2000 [10], and vector quantization techniques such as k-means clustering [11] and Self-Organizing Maps (SOM) [12].

In recent years, dimensionality reduction techniques have emerged as a promising approach to image compression. These methods aim to reduce the dimensionality of the image data while preserving the most significant information. Principal Component Analysis (PCA) is a widely studied linear dimensionality reduction method that has been applied in various fields, including image compression [13]–[18]. PCA transforms the original high-dimensional data into a lower-dimensional representation by projecting it onto a new set of orthogonal axes that maximize the variance. This process captures the most relevant information found in the original data while reducing its dimensionality, leading to effective compression of the image data.

However, PCA has its limitations [19], mainly due to the assumption that the data lies in a linear subspace. This assumption may not always hold true in real-world scenarios, leading to suboptimal compression performance. To address this limitation, non-linear dimensionality reduction techniques like Isomap and Locally Linear Embedding (LLE) have been developed. Isomap is a manifold learning technique that estimates the intrinsic geometry of the data by preserving geodesic distances in the reduced-dimensional space. On the other hand, LLE aims to preserve local neighborhood relationships in the lower-dimensional space by reconstructing each data point using a linear combination of its neighbors.

While Isomap and LLE have shown promise in various applications, including machine learning [20], [21], computer vision [22], [23], and pattern recognition [24], [25], their potential in the context of image compression remains relatively unexplored. This paper will investigate the performance of these non-linear dimensionality reduction techniques in image compression tasks, alongside PCA and KPCA, to provide a comprehensive comparison of their capabilities and limitations.

Kernel Principal Component Analysis (KPCA) is an extension of PCA that leverages kernel functions to perform non-linear dimensionality reduction. It offers a more flexible approach by mapping the input data to a higher-dimensional feature space, where linear techniques such as PCA can be applied. The motivation for using KPCA in image compression stems from its ability to capture complex non-linear relationships in the data, potentially leading to more effective compression than linear methods like PCA. The innovation of KPCA lies in its capability to handle non-linear data structures, making it a promising candidate for image compression tasks where traditional PCA may fall short.

In this paper, I will explore the potential of KPCA for image compression, alongside a comprehensive comparison with PCA, Isomap, and LLE. I will discuss the limitations and challenges faced by each method, their implementation in the R programming language, and their performance on two widely used image datasets: Olivetti Faces [26] and Caltech-101 [27]. Furthermore, I will delve into the selection of appropriate kernel functions for KPCA, the impact of hyper-parameters on the compression performance, and the trade-offs between compression ratio and reconstruction quality. This investigation aims to provide a thorough understanding of the capabilities and potential of KPCA in the context of image compression, paving the way for future research and development in this area.

Ultimately, my goal is to provide a comprehensive analysis of the effectiveness of linear and non-linear dimensionality reduction techniques in image compression tasks. This study will shed light on the strengths and weaknesses of each method, the scenarios in which they are most suitable, and the potential improvements that can be made to enhance their performance. By understanding the capabilities of these techniques, I hope to contribute to the ongoing development of more efficient and robust image compression methods, catering to the ever-growing demands of digital image processing and communication applications.

II. Related Works

A. Image Compression Techniques

Images contain significant spatial correlation and redundancy between neighboring pixels. Compression can be achieved by removing these redundancies. An image compression system consists of a compressor and a decompressor. The compressor contains a preprocessor and an encoder, while the decompressor contains a decoder and a postprocessor.

The compression algorithms can be divided into lossy compression and lossless compression. Lossy compression algorithms remove redundancy and irrelevancy from the image, resulting in loss of information. They provide high compression ratios but with loss of quality. On the other hand, lossless compression algorithms encode the image in a reversible manner, allowing the exact original image to be reconstructed from the compressed data. They provide lower compression ratios but the quality is preserved.

Lossy compression is suitable for applications where minor loss of information is acceptable, such as video streaming and digital photography. Lossless compression is required in medical imaging, satellite imaging, and legal documentation where no information loss is allowed. The selection of a compression algorithm depends on the specific application and the required trade-off between compression ratio and image quality.

1) Lossless Compression:

a) Huffman coding: Huffman coding [7] is a data compression technique that reduces coding redundancy in image data while maintaining image quality. Developed by Huffman, it assigns variable length codewords to symbols in an alphabet based on their probability distribution. The algorithm uses data statistics to represent symbols with shorter codes for more probable symbols. It achieves the shortest average code length for a given probability distribution, assuming all source symbol probabilities are an exact power of ½. The Huffman coding algorithm can be described mathematically as follows:

Given a set of source symbols with probabilities $p_1, p_2, ..., p_n$, where $\sum_{i=1}^{n} p_i = 1$, the Huffman coding algorithm constructs a binary tree with each leaf node representing a source symbol. The probability of a leaf node is equal to the probability of its corresponding source symbol. The binary tree is constructed by iteratively merging the two nodes with the lowest probabilities into a new parent node until a single root node is obtained. The codeword for a source symbol is obtained by traversing the binary tree from the root to the leaf node corresponding to the symbol, assigning a 0 for each left branch and a 1 for each right branch traversed. The codeword for a symbol is the sequence of 0s and 1s assigned during the traversal. The average code length is given by:

$$L = \sum_{i=1}^{n} p_i l_i$$

where l_i is the length of the codeword for symbol i. The Huffman coding algorithm ensures that the average code length is minimized for the given probability distribution.

b) Arithmetic coding: Arithmetic coding [28] is a variable-length coding technique that reduces coding redundancy similar to Huffman coding. However, arithmetic coding is optimal only when all symbol probabilities are an integral power of ½. Glen and Langdon provided a detailed introduction to arithmetic coding.

Arithmetic coding works by dividing the interval between 0 and 1 into smaller intervals that correspond to the probabilities of the message symbols. Each probability is represented by a two-end interval, where the left end is closed, and the right end is open. The next input symbol selects one of these intervals, and the process is repeated. As each symbol is encoded, the selected interval narrows down, and at the end, any number within the final interval can represent the encoded message. The arithmetic coding procedure can be mathematically described as follows:

Given a set of source symbols with probabilities $p_1, p_2, ..., p_n$, where $\sum_{i=1}^{n} p_i = 1$, and an input message consisting of symbols $x_1, x_2, ..., x_m$, the arithmetic coding algorithm computes a single real number in the range [0,1] that represents the encoded message. Initially, the interval [0,1) is divided into sub-intervals, each with a size proportional to the probability of the corresponding symbol. The sub-interval corresponding to the first input symbol is selected, and the process is repeated for the remaining input symbols. After encoding all symbols, the final interval is output as the compressed message. The decoding process is the reverse of the encoding process, where the original message is reconstructed from the compressed message and the probability distribution of the source symbols.

c) Lossless predictive coding: The lossless predictive image compression approach [29] aims to remove interpixel redundancies in images by predicting the current pixel value using nearby pixels and generating new values for coding. The new values represent the error generated from subtracting the predicted value from the original value.

The lossless predictive coding system consists of two parts: the transmitter and the receiver. At the transmitter, the current pixel value is predicted using the closely spaced neighborhood pixel values. The prediction is typically based on linear combinations of the neighboring pixels. The prediction error is then calculated by subtracting the predicted value from the original value. The prediction error is then quantized and coded using variable length coding techniques such as Huffman or arithmetic coding. At the receiver, the prediction error is decoded and added back to the predicted value to reconstruct the original pixel value. The reconstructed pixel values are then used to predict the next pixel value in the image. The predictive coding approach is effective in reducing image redundancy and achieving high compression ratios with minimal loss of image quality.

2) Lossy Compression:

a) Predictive coding: Prediction is a fundamental concept in data compression that enables the encoding of data by representing the error between the actual data and the information predicted from past observations. Predictive coding [30] is widely used in image compression because pixels in images exhibit high correlation among their neighboring samples, resulting in mutual redundancy in the raw data. By decorrelating the data, predictive coding removes mutual redundancy and produces a more efficient and better-compressed coding of the signal.

Predictive coding predicts the next pixel value based on a sequence of reproduced pixel values obtained during the scanning of the image and encodes the difference between the predicted and actual values. The better the prediction, the smaller the transmitted error, resulting in a better coding process. Differential pulse code modulation (DPCM) is a prediction method where the current reproduced pixel is the sum of the predicted pixel value and the quantized error value between the current pixel and the predicted pixel.

The predictive coding encoder consists of two parts: the predictor and the quantizer. The predictor estimates the next value of the image signal using the previously coded elements, while the quantizer quantizes the difference between the predicted value and the original value. The order of the predictor is the number of previous elements used in the prediction. The predictive coding approach is effective in reducing image redundancy and achieving high compression ratios with minimal loss of image quality.

b) Transform coding: Transform coding [31] is a widely used approach in image compression, where pixels in the image domain are transformed into another domain to produce a set of coefficients that provide a more natural and compact representation. This approach allows some coefficients to carry the majority of the energy in the image, while others are likely to be small or zero.

Transform coding is a general scheme for lossy image compression that uses a reversible, linear transform, such as the Fourier transform, to map the image into a set of coefficients, which are then quantized and coded. A good transformation packs as much information as possible into a small number of transform coefficients, and quantization selectively eliminates the coefficients that carry the least information. In the transform coding approach, an $N \times N$ input image is divided into a number of $n \times n$ non-overlapping sub-images (blocks), which are then transformed to generate $(\frac{N}{n})^2$ subimage transform arrays, each of size$n \times n$, and the transform is applied separately to each of these blocks.

c) JPEG standards: The Joint Photographic Expert Group (JPEG) [10] was established by the International Standards Organisation (ISO) in 1987 to develop a still image compression standard. JPEG has several modes, including baseline, lossless, progressive, and hierarchical. Baseline is the most widely used mode, which supports lossy coding using Discrete Cosine Transform (DCT), Huffman coding, and sequential transmission. The extended options include arithmetic coding instead of Huffman coding, progressive sequential or hierarchical modes. An independent lossless mode is also available that uses predictive coding instead of DCT.

In the baseline mode, the image is divided into 8x8 blocks, and DCT is computed over these blocks. The transformed blocks are quantized with a uniform scalar quantizer, zigzag scanned, and entropy coded with Huffman coding. The quantization step size for each of the 64 DCT coefficients is specified in a quantization table, which remains the same for all blocks. The DC coefficients of all blocks are coded separately, using a predictive scheme. The lossless mode is based on a completely different algorithm that uses a predictive scheme based on the nearest three causal neighbors.

The progressive and hierarchical modes of JPEG are both lossy and differ only in the way the DCT coefficients are coded or computed compared to the baseline mode. They allow a reconstruction of a lower quality or lower resolution version of the image by partially decoding the compressed bitstream. JPEG-2000 is an international standard for still image coding that offers improved compression performance and new functionalities compared to the original JPEG standard.

The core coding algorithm of JPEG-2000 is based on embedded block coding with optimized truncation, and it offers a wide range of functionalities in a single bit-stream. JPEG-2000 is based on discrete wavelet transform, scalar quantization, context modeling, arithmetic coding, and post-compression rate allocation. JPEG-2000 provides compression performance as good as or better than conventional JPEG, particularly at low bit rates.

d) Singular value decomposition : Singular Value Decomposition (SVD) [32] is a widely used technique in linear algebra, with important applications in digital image processing, dimensionality reduction, and image compression. SVD's excellent energy compaction in the least square sense makes it a popular choice for image compression. However, the need for recalculation for each sub-image is a significant drawback of SVD. To overcome this limitation and exploit the optimal energy compaction properties of SVD, researchers have focused on developing more efficient SVD coders that effectively code singular values and vectors to reduce computational cost.

SVD is an efficient method for diagonalizing a rectangular matrix A by decomposing it into three matrices U, S, and V, such that $A = USV^T$, where S is a diagonal matrix with singular values along the diagonal, and U and V are orthonormal matrices with left and right singular vectors, respectively. The singular vectors form orthonormal bases, and the relationship $Av_i = s_i u_i$ holds, where s_i is the i-th singular value, u_i and v_i are the i-th left and right singular vectors, respectively. SVD is an approximation technique that reduces any matrix into a smaller invertible and square matrix, making it applicable to any real $m \times n$ matrix.

The computation of SVD involves finding the eigenvalues and eigenvectors of AA^T and $A^T A$, where the eigenvectors of $A^T A$ and AA^T correspond to the columns of V and U, respectively. The singular values are the square roots of the eigenvalues of AA^T or $A^T A$ and are typically arranged in descending order. Another important property of singular values is that they are always real numbers.

e) Vector quantization: Vector quantization (VQ) [33] is a lossy data compression technique that reduces the number of possible pixel values in k-dimensional vectors, instead of scalars. VQ achieves optimal rate-distortion performance, subject to constraints on the memory or block length of the observable signal segment being coded. VQ maps input vectors into a set of code vectors, where similar vectors are mapped to the same code vectors in the codebook. The dictionary of patterns used to approximate the input vectors is called the codebook, and the patterns in the codebook are called code vectors or codewords. The efficiency of VQ comes from its role as a pattern matching technique, where the vector of samples is a pattern that must be approximated by one of the finite set of prototype patterns.

VQ is a clustering method that groups similar vectors (blocks) into one class, mapping features extracted from the sampled input image using pre-processing operations. The simplest way to encode the image with VQ is to independently quantize each input vector, which is called memoryless VQ. Spatial correlation between pixels in an individual vector is efficiently exploited. The input space is not evenly occupied by these vectors, as some input vectors are very common while others hardly ever appear in real images due to the high correlation between neighboring pixel values. Certain structures like edges, flat areas, and slopes are found in almost every image, making VQ superior to predictive and transform coding.

f) Hybrid coding: Habibi proposed a hybrid transform/DPCM coding system [34] that combines the advantages of transform coding and DPCM, while being less vulnerable to their limitations. The proposed one-dimensional hybrid coding method divides an$N \times N$ image into vertical strips of width M, and performs a horizontal transformation using a transformation matrix on each strip to remove the redundancy between pixels situated in the same row of a strip. DPCM is then used to remove the vertical redundancy by coding the transformed coefficients using a parallel linear DPCM predictor. The error signal is quantized using a Lloyd-Max quantizer and transferred to the communication channel. At the receiver, the predicted coefficient signal is added to the quantized error signal to retrieve the original value of the transformed coefficients. The reconstructed value of the image is obtained by performing an inverse transformation.

B. Principal Components Analysis

Principal Component Analysis (PCA) [2] is a technique used to reduce the dimensionality of high-dimensional data by projecting it onto a lower-dimensional subspace while retaining most of the information. The steps involved in PCA are as follows:

1) Standardize the data: Before performing PCA, it is essential to standardize the data, so that variables with larger scales do not dominate the analysis. Standardization involves subtracting the mean from each feature and dividing it by its standard deviation. The standardized data matrix is represented by X.

$$X_{i,j} = \frac{X_{i,j} - \mu_j}{\sigma_j}$$

where $X_{i,j}$ is the element in the ith row and jth column of X, μ_j is the mean of the jth column of X, and σ_j is the standard deviation of the jth column of X.

2) Calculate the covariance matrix: The covariance matrix represents the relationship between the different features in the data. The covariance between two variables x and y is given by:

$$cov(x, y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})$$

where n is the number of observations, x_i and y_i are the ith observations of x and y, and $\overline{x}$ and $\overline{y}$ are the means of x and y, respectively.

The covariance matrix of X is calculated as follows:

$$\Sigma = \frac{1}{n-1} X^T X$$

where Σ is the covariance matrix of X, and the superscript T denotes the transpose of a matrix.

3) Calculate the eigenvectors and eigenvalues of the covariance matrix: The eigenvectors and eigenvalues of the covariance matrix represent the directions and magnitudes of the principal components of the data. The eigenvectors are calculated as follows:

$$\Sigma v = \lambda v$$

where Σ is the covariance matrix, v is the eigenvector, and λ is the corresponding eigenvalue.

The eigenvectors and eigenvalues of Σ can be calculated using standard techniques, such as the power iteration method or Singular Value Decomposition (SVD). The eigenvectors are sorted in descending order of their corresponding eigenvalues.

4) Select the principal components: The principal components are selected based on the amount of variance they explain in the data. The proportion of variance explained by each principal component is given by the ratio of its corresponding eigenvalue to the sum of all the eigenvalues. The first k principal components are selected that explain a sufficiently large proportion of the variance (e.g., 95%).

5) Project the data onto the principal components: The data is projected onto the selected principal components to obtain a lower-dimensional representation. The projection of the standardized data matrix X onto the first k principal components is given by:

$$Z = XV_k$$

where V_k is a matrix consisting of the first k eigenvectors of Σ, and Z is the matrix of projected data.

III. My Methodology

In this study, I aim to investigate the potential of Kernel Principal Component Analysis (KPCA) and traditional Principal Component Analysis (PCA) for image compression and dimensionality reduction, as well as to explore the capabilities of Isomap and Locally Linear Embedding (LLE) in these tasks. My methodology consists of five main steps: data preprocessing, PCA implementation, KPCA implementation, Isomap and LLE implementation, and performance evaluation. In this section, I provide a detailed description of each step and the rationale behind the choices made.

A. Data Preprocessing

Before applying PCA, KPCA, Isomap, and LLE to the image datasets, I preprocess the data to ensure compatibility with the algorithms and to facilitate accurate comparisons. The preprocessing steps are as follows:

1) Grayscale Conversion: Since PCA, KPCA, Isomap, and LLE are primarily designed for feature extraction in single-channel data, I convert the color images in the Caltech-101 Dataset to grayscale. This simplifies the data representation while retaining most of the relevant information for the compression and dimensionality reduction tasks.

2) Data Normalization: To ensure that the pixel values are on a similar scale, I normalize the image data by dividing each pixel value by the maximum possible pixel value (i.e., 255), resulting in pixel values ranging from 0 to 1. This step is crucial for PCA, KPCA, Isomap, and LLE, as it ensures that the algorithms are not biased towards specific pixel value ranges and can effectively capture the underlying manifold structure in the data.

B. PCA Implementation

After preprocessing the image data, I implement PCA for image compression as follows:

1) Data Vectorization: I reshape the images in both datasets into one-dimensional vectors by concatenating the rows of each image. This step is necessary because PCA operates on vectorized data.

2) Mean Centering: I calculate the mean of the vectorized images and subtract it from each image vector. This ensures that the principal components extracted by PCA capture the most relevant variance in the data, as opposed to being influenced by the mean.

3) Covariance Matrix Calculation: I compute the covariance matrix of the mean-centered image vectors. This matrix captures the relationships between different pixel values and is essential for determining the optimal linear projections that capture the most variance in the data.

4) Eigendecomposition: I perform eigendecomposition on the covariance matrix to obtain the eigenvectors and eigenvalues. The eigenvectors correspond to the principal components, and the eigenvalues represent the amount of variance captured by each component.

5) Dimensionality Reduction: I select the top k eigenvectors with the largest corresponding eigenvalues as the principal components. These components are then used to project the original image vectors into a lower-dimensional space, achieving compression.

6) Image Reconstruction: To reconstruct the compressed images, I project the lower-dimensional representations back into the original space using the selected principal components. Finally, I add the mean image vector to obtain the reconstructed images.

C. KPCA Implementation

I implement KPCA for image compression using a similar process, with the main difference being the application of a kernel function to the data:

1) Kernel Function Selection: I explore different kernel functions, including linear, polynomial, Gaussian radial basis function (RBF), and sigmoid kernels, to evaluate their impact on the performance of KPCA for image compression. The choice of kernel function plays a crucial role in determining the effectiveness of KPCA, as it defines the mapping of the original data into the higher-dimensional feature space.

2) Kernel Matrix Calculation: I compute the kernel matrix for the mean-centered image vectors using the selected kernel function. This matrix captures the relationships between the mapped data points in the feature space.

3) Centering the Kernel Matrix: To ensure that the principal components extracted by KPCA capture the most relevant variance in the feature space, I center the kernel matrix.

4) Eigendecomposition: Similar to PCA, I perform eigendecomposition on the centered kernel matrix to obtain the eigenvectors and eigenvalues.

5) Dimensionality Reduction: I select the top k eigenvectors with the largest corresponding eigenvalues and use them to project the original image vectors into a lower-dimensional space in the feature space, achieving compression.

6) Image Reconstruction: To reconstruct the compressed images, I project the lower-dimensional representations back into the original space using the selected eigenvectors and the kernel function. Finally, I add the mean image vector to obtain the reconstructed images.

D. Isomap Implementation

After preprocessing the image data, I implement Isomap for image compression as follows:

1) Data Vectorization: Similar to PCA, I reshape the images in both datasets into one-dimensional vectors by concatenating the rows of each image. This step is necessary because Isomap operates on vectorized data.

2) Neighborhood Graph Construction: I compute the pairwise distances between all image vectors and construct a neighborhood graph by connecting each data point to its k-nearest neighbors. This graph is used to approximate the geodesic distances between data points, which is essential for Isomap's dimensionality reduction process.

3) Geodesic Distance Matrix Calculation: I compute the shortest path distances between all pairs of nodes in the neighborhood graph, approximating the geodesic distances. This matrix captures the pairwise relationships between images in the original high-dimensional space.

4) Multidimensional Scaling: I apply classical multidimensional scaling (MDS) on the geodesic distance matrix to obtain a lower-dimensional representation of the image vectors. This step achieves compression by preserving the pairwise geodesic distances between images in the lower-dimensional space.

5) Linear Regression Model Construction: Due to the non-linear nature of Isomap, the compressed matrix cannot be directly recovered into the original images. Therefore, I construct an additional linear regression model that learns the mapping from the lower-dimensional representation to the original image vectors.

6) Image Reconstruction: To reconstruct the compressed images, I use the linear regression model to predict the original image vectors from the lower-dimensional representations. This step produces the reconstructed images, which can then be compared to the original images to evaluate Isomap's performance in image compression.

E. LLE Implementation

After preprocessing the image data, I implement LLE for image compression as follows:

1) Data Vectorization: Similar to PCA and Isomap, I reshape the images in both datasets into one-dimensional vectors by concatenating the rows of each image. This step is necessary because LLE operates on vectorized data.

2) Neighborhood Selection: I compute the pairwise distances between all image vectors and select the k-nearest neighbors for each data point. This step is crucial for LLE's dimensionality reduction process, as it discovers the local neighborhood structure around each data point.

3) Reconstruction Weights Calculation: For each data point, I calculate the reconstruction weights that best reconstruct it from its k-nearest neighbors in the original high-dimensional space. These weights are obtained by solving a constrained linear least squares problem that minimizes the reconstruction error.

4) Embedding Calculation: Using the reconstruction weights, I construct a lower-dimensional representation of the image vectors that preserves the local neighborhood relationships discovered in the original space. This step achieves compression by embedding the images in a lower-dimensional space while maintaining their local structure.

5) Linear Regression Model Construction: Due to the non-linear nature of LLE, the compressed matrix cannot be directly recovered into the original images. Therefore, I construct an additional linear regression model that learns the mapping from the lower-dimensional representation to the original image vectors.

6) Image Reconstruction: To reconstruct the compressed images, I use the linear regression model to predict the original image vectors from the lower-dimensional representations. This step produces the reconstructed images, which can then be compared to the original images to evaluate LLE's performance in image compression.

IV. My Concrete Construction

A. Concrete PCA-based Image Compression

Here is my concrete construction of dimensionality reduction of an image using PCA, I will use a $n \times p$ matrix $\mathbf{X}_{(n\times p)}$ to represent the image that needs to be compressed. The process can be divided into 7 steps.

1) Standardize the input data: Let $\mathbf{X}_{(n\times p)}$ be the input data matrix, where n is the number of samples (rows) and p is the number of features (columns). To standardize the input data, I compute the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ for each feature and scale the data accordingly:

$$\mathbf{X}_{std_{(n\times p)}} = \frac{\mathbf{X}_{(n\times p)} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

2) Compute the covariance matrix: The covariance matrix $\mathbf{C}_{(p\times p)}$ captures the variance and linear relationships between the features of the input data. It is computed as:

$$\mathbf{C}_{(p\times p)} = \frac{1}{n-1}\mathbf{X}^T_{std_{(n\times p)}}\mathbf{X}_{std_{(n\times p)}}$$

3) Compute the eigenvectors and eigenvalues of the covariance matrix: I compute the eigenvectors and eigenvalues of the covariance matrix $\mathbf{C}_{(p\times p)}$. The eigenvectors represent the principal components (orthogonal axes) of the data, and the eigenvalues indicate the amount of variance captured by each principal component.

Let $\mathbf{E}_{(p\times p)}$ be the matrix of eigenvectors, and $\boldsymbol{\Lambda}_{(p\times p)}$ be the diagonal matrix of eigenvalues:

$$\mathbf{C}_{(p\times p)}\mathbf{E}_{(p\times p)} = \mathbf{E}_{(p\times p)}\boldsymbol{\Lambda}_{(p\times p)}$$

4) Sort the eigenvectors by descending eigenvalues: To retain the most important features, I sort the eigenvectors by their corresponding eigenvalues in descending order.

5) Select the top k eigenvectors: I select the top k eigenvectors, where k is the desired number of principal components. These eigenvectors form the matrix $\mathbf{E}_{k_{(p\times k)}}$.

6) Project the input data onto the selected eigenvectors: I project the standardized input data $\mathbf{X}_{std_{(n\times p)}}$ onto the selected eigenvectors $\mathbf{E}_{k_{(p\times k)}}$ to obtain the reduced data $\mathbf{X}_{red_{(n\times k)}}$:

$$\mathbf{X}_{red_{(n\times k)}} = \mathbf{X}_{std_{(n\times p)}}\mathbf{E}_{k_{(p\times k)}}$$

7) Reconstruct the original data: To reconstruct the original data from the reduced data, I project $\mathbf{X}_{red_{(n\times k)}}$ back onto the selected eigenvectors and rescale the result:

$$\mathbf{X}_{rec_{(n\times p)}} = (\mathbf{X}_{red_{(n\times k)}}\mathbf{E}^T_{k_{(p\times k)}})\cdot\boldsymbol{\sigma}+\boldsymbol{\mu}$$

These are the main steps involved in PCA, along with the corresponding mathematical notation. The reduced data $\mathbf{X}_{red_{(n\times k)}}$ can be used for further analysis or visualization, and the reconstructed data $\mathbf{X}_{rec_{(n\times p)}}$ provides an approximation of the original data in the lower-dimensional space.

B. Concrete KPCA-based Image Compression

Here is my concrete construction of dimensionality reduction of an image using KPCA. I will use a $n \times p$ matrix $\mathbf{X}_{(n\times p)}$ to represent the image that needs to be compressed. The process can be divided into 7 steps.

1) Standardize the input data: Similar to PCA, I first standardize the input data matrix $\mathbf{X}_{(n\times p)}$ by computing the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ for each feature and scaling the data accordingly:

$$\mathbf{X}_{std_{(n\times p)}} = \frac{\mathbf{X}_{(n\times p)}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

2) Compute the kernel matrix: I compute the kernel matrix $\mathbf{K}_{(n\times n)}$ using an appropriate kernel function $k(\cdot,\cdot)$ that maps the input data into a higher-dimensional space:

$$\mathbf{K}_{(n\times n)} = k(\mathbf{X}_{std_{(n\times p)}},\mathbf{X}_{std_{(n\times p)}})$$

3) Center the kernel matrix: I center the kernel matrix $\mathbf{K}_{(n\times n)}$ to ensure that the mean of the transformed data is zero:

$$\mathbf{K}_{cen_{(n\times n)}} = (\mathbf{I}_{(n\times n)}-\frac{1}{n}\mathbf{1}_{(n\times n)})\mathbf{K}_{(n\times n)}(\mathbf{I}_{(n\times n)}-\frac{1}{n}\mathbf{1}_{(n\times n)})$$

where $\mathbf{I}_{(n\times n)}$ is the identity matrix and $\mathbf{1}_{(n\times n)}$ is a matrix of ones.

4) Compute the eigenvectors and eigenvalues of the centered kernel matrix: I compute the eigenvectors and eigenvalues of the centered kernel matrix $\mathbf{K}_{cen_{(n\times n)}}$. The eigenvectors represent the principal components in the higher-dimensional space, and the eigenvalues indicate the amount of variance captured by each principal component.

Let $\mathbf{A}_{(n\times n)}$ be the matrix of eigenvectors, and $\boldsymbol{\Gamma}_{(n\times n)}$ be the diagonal matrix of eigenvalues:

$$\mathbf{K}_{cen_{(n\times n)}}\mathbf{A}_{(n\times n)} = \mathbf{A}_{(n\times n)}\boldsymbol{\Gamma}_{(n\times n)}$$

5) Sort the eigenvectors by descending eigenvalues: Similar to PCA, I sort the eigenvectors by their corresponding eigenvalues in descending order.

6) Select the top k eigenvectors: I select the top k eigenvectors, where k is the desired number of principal components. These eigenvectors form the matrix $\mathbf{A}_{k_{(n\times k)}}$.

7) Project the input data onto the selected eigenvectors: I project the standardized input data $\mathbf{X}_{std_{(n\times p)}}$ onto the selected eigenvectors $\mathbf{A}_{k_{(n\times k)}}$ to obtain the reduced data $\mathbf{X}_{red_{(n\times k)}}$:

$$\mathbf{X}_{red_{(n\times k)}} = \mathbf{K}_{cen_{(n\times n)}}\mathbf{A}_{k_{(n\times k)}}$$

8) Reconstruct the original data: Reconstructing the original data from the reduced data in KPCA is not as straightforward as in PCA. To do this, I need to compute the pre-image of $\mathbf{X}_{red_{(n\times k)}}$ in the input space. The pre-image can be found using various techniques, such as gradient descent or regression models.

Once the pre-image is obtained, I project it back onto the input space and rescale the result:

$$\mathbf{X}_{rec_{(n\times p)}} = (\text{pre-image}(\mathbf{X}_{red_{(n\times k)}}))\cdot\boldsymbol{\sigma}+\boldsymbol{\mu}$$

These are the main steps involved in KPCA, along with the corresponding mathematical notation. The reduced data $\mathbf{X}_{red_{(n\times k)}}$ can be used for further analysis or visualization, and the reconstructed data $\mathbf{X}_{rec_{(n\times p)}}$ provides an approximation of the original data in the lower-dimensional space. It is obvious that reconstructing the original data in KPCA is more complex than in PCA, and the quality of the reconstruction depends on the kernel function and the pre-image estimation method used.

C. Concrete Isomap-based Image Compression

Here is my concrete construction of dimensionality reduction of an image using Isomap. I will use a $n \times p$ matrix $\mathbf{X}_{(n\times p)}$ to represent the image that needs to be compressed. The process can be divided into 7 steps.

1) Standardize the input data: Let $\mathbf{X}_{(n\times p)}$ be the input data matrix, where n is the number of samples (rows) and p is the number of features (columns). To standardize the input data, I compute the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ for each feature and scale the data accordingly:

$$\mathbf{X}_{std_{(n\times p)}} = \frac{\mathbf{X}_{(n\times p)}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

2) Compute the pairwise distance matrix: I compute the pairwise distance matrix $\mathbf{D}_{(n\times n)}$ between all image vectors in the standardized data matrix $\mathbf{X}_{std_{(n\times p)}}$.

3) Construct the k-nearest neighbor graph: Based on the pairwise distance matrix $\mathbf{D}_{(n\times n)}$, I construct a k-nearest neighbor graph by connecting each data point to its k-nearest neighbors. This graph is used to approximate the geodesic distances between data points.

4) Compute the geodesic distance matrix: I compute the shortest path distances between all pairs of nodes in the k-nearest neighbor graph, approximating the geodesic distances. This results in the geodesic distance matrix $\mathbf{G}_{(n\times n)}$.

5) Apply classical multidimensional scaling: I apply classical multidimensional scaling (MDS) on the geodesic distance matrix $\mathbf{G}_{(n\times n)}$ to obtain a lower-dimensional representation of the image vectors in the matrix $\mathbf{X}_{red_{(n\times k)}}$:

$$\mathbf{X}_{red_{(n\times k)}} = \text{MDS}(\mathbf{G}_{(n\times n)})$$

6) Construct a linear regression model: Due to the nonlinear nature of Isomap, I cannot directly recover the original images from the compressed matrix. Therefore, I construct a linear regression model that learns the mapping from the lower-dimensional representation $\mathbf{X}_{red_{(n\times k)}}$ to the standardized image vectors $\mathbf{X}_{std_{(n\times p)}}$.

7) Reconstruct the original data: To reconstruct the original data from the reduced data, I use the linear regression model to predict the standardized image vectors from the lower-dimensional representations:

$$\mathbf{X}_{std_{rec_{(n\times p)}}} = \text{LinearRegression}(\mathbf{X}_{red_{(n\times k)}})$$

Finally, I rescale the reconstructed standardized data $\mathbf{X}_{std_{rec_{(n\times p)}}}$ to obtain the reconstructed images:

$$\mathbf{X}_{rec_{(n\times p)}} = \mathbf{X}_{std_{rec_{(n\times p)}}} \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$$

These are the main steps involved in Isomap, along with the corresponding mathematical notation. The reduced data $\mathbf{X}_{red_{(n\times k)}}$ can be used for further analysis or visualization, and the reconstructed data $\mathbf{X}_{rec_{(n\times p)}}$ provides an approximation of the original data in the lower-dimensional space.

D. Concrete LLE-based Image Compression

Here is my concrete construction of dimensionality reduction of an image using LLE. I will use a $n\times p$ matrix $\mathbf{X}_{(n\times p)}$ to represent the image that needs to be compressed. The process can be divided into 7 steps.

1) Standardize the input data: Let $\mathbf{X}_{(n\times p)}$ be the input data matrix, where n is the number of samples (rows) and p is the number of features (columns). To standardize the input data, I compute the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ for each feature and scale the data accordingly:

$$\mathbf{X}_{std_{(n\times p)}} = \frac{\mathbf{X}_{(n\times p)} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

2) Compute the pairwise distance matrix: I compute the pairwise distance matrix $\mathbf{D}_{(n\times n)}$ between all image vectors in the standardized data matrix $\mathbf{X}_{std_{(n\times p)}}$.

3) Construct the k-nearest neighbor graph: Based on the pairwise distance matrix $\mathbf{D}_{(n\times n)}$, I construct a k-nearest neighbor graph by connecting each data point to its k-nearest neighbors. This graph is used to capture the local neighborhood structure of the data.

4) Compute the reconstruction weights: For each data point, I compute the reconstruction weights that best reconstruct it from its k-nearest neighbors in the original high-dimensional space. These weights are obtained by solving a constrained linear least squares problem that minimizes the reconstruction error. The result is a reconstruction weight matrix $\mathbf{W}_{(n\times n)}$.

5) Compute the lower-dimensional embedding: Using the reconstruction weight matrix $\mathbf{W}_{(n\times n)}$, I construct a lower-dimensional representation of the image vectors in the matrix $\mathbf{X}_{red_{(n\times k)}}$ that preserves the local neighborhood relationships discovered in the original space:

$$\mathbf{X}_{red_{(n\times k)}} = \text{LLEEmbedding}(\mathbf{W}_{(n\times n)})$$

6) Construct a linear regression model: Due to the nonlinear nature of LLE, I cannot directly recover the original images from the compressed matrix. Therefore, I construct a linear regression model that learns the mapping from the lower-dimensional representation $\mathbf{X}_{red_{(n\times k)}}$ to the standardized image vectors $\mathbf{X}_{std_{(n\times p)}}$.

7) Reconstruct the original data: To reconstruct the original data from the reduced data, I use the linear regression model to predict the standardized image vectors from the lower-dimensional representations:

$$\mathbf{X}_{std_{rec_{(n\times p)}}} = \text{LinearRegression}(\mathbf{X}_{red_{(n\times k)}})$$

Finally, I rescale the reconstructed standardized data $\mathbf{X}_{std_{rec_{(n\times p)}}}$ to obtain the reconstructed images:

$$\mathbf{X}_{rec_{(n\times p)}} = \mathbf{X}_{std_{rec_{(n\times p)}}} \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$$

These are the main steps involved in LLE, along with the corresponding mathematical notation. The reduced data $\mathbf{X}_{red_{(n\times k)}}$ can be used for further analysis or visualization, and the reconstructed data $\mathbf{X}_{rec_{(n\times p)}}$ provides an approximation of the original data in the lower-dimensional space.

E. Performance Evaluation

To compare the performance of PCA, KPCA, Isomap, and LLE for image compression, I evaluate the quality of the reconstructed images using three widely used metrics: Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM). MSE measures the average squared difference between the original and reconstructed images, while PSNR evaluates the ratio between the maximum possible pixel value and the MSE. SSIM is a metric that considers the structural information, luminance, and contrast in the images, providing a more comprehensive assessment of image quality. Lower MSE values, higher PSNR values, and higher SSIM values indicate better image reconstruction quality.

In addition to these quantitative metrics, I also provide qualitative comparisons through visual inspection of the original and reconstructed images. This allows for the assessment of the preservation of image details and the introduction of artifacts during the compression process.

I perform the evaluation for different values of k (i.e., the number of principal components retained) to analyze the trade-off between compression ratio and reconstruction quality. Moreover, I investigate the impact of various kernel functions on the performance of KPCA to identify the most suitable function for image compression.

In summary, my methodology provides a rigorous and comprehensive evaluation of the potential of PCA, KPCA, Isomap, and LLE for image compression. By comparing their performance, I aim to shed light on the advantages and limitations of each technique in the context of image compression. In particular, the advantages of KPCA include its ability to capture nonlinear relationships in the data, which may lead to better reconstruction quality for certain types of images. Moreover, the flexibility in selecting the kernel function allows for the adaptation of KPCA to various image characteristics, potentially improving its overall performance.

V. My Implementation in R

In this section, I provide a detailed overview of the R programming language implementation of PCA, KPCA, Isomap, and LLE for image compression. I chose R due to its versatility and rich ecosystem of data analysis libraries.

A. Key R Libraries and Functions

I utilized the following R libraries and functions for my implementation:

1) **dplyr**: A library for data manipulation that provides a consistent set of verbs to work with data frames.
2) **kernlab**: A package containing kernel-based machine learning methods, which I used to perform KPCA.
3) **Rdimtools**: A package for dimensionality reduction techniques, used for Isomap and LLE.
4) **glmnet**: A package for fitting generalized linear models via penalized maximum likelihood, used to build linear regression models for image recovery with Isomap and LLE.
5) **openml**: A package for accessing the OpenML database, which I used to fetch the Olivetti Faces Dataset.
6) **purrr**: A package for functional programming, which I used to simplify the code and make it more readable.
7) **ggplot2**: A package for creating elegant data visualizations, used for plotting the results.
8) **tidyr**: A package for cleaning and reshaping data, used for preprocessing steps.
9) **imager**: A package for image processing, used for loading and manipulating images.

B. Step-by-Step Implementation

I implemented the PCA, KPCA, Isomap, and LLE image compression methods using R and the libraries mentioned above. Here is a step-by-step guide to my implementation:

1) **Load dataset**: I used the `fetch_101` function to load a single grayscale image from the Caltech-101 Dataset with its original dimensions. This function returns a list containing the image data as a vector (`X`) and the original dimensions of the image (`ori_shape`).
2) **Perform dimensionality reduction**: The `DimensionalityReduction` function takes the image data, original dimensions, the desired compression method (PCA, KPCA, Isomap, or LLE), and the number of principal components as input. It first scales the input data (`X_std`) and then calls the appropriate function to perform dimensionality reduction based on the chosen method. The reconstructed image data (`X_std_reconstructed`) is then returned.
3) **Perform PCA**: In the `perform_pca` function, I used the `prcomp` function from the base R `stats` package to perform PCA. I then used the `predict` function to reconstruct the image data.
4) **Perform KPCA**: In the `perform_kpca` function, I used the `kpca` function from the `kernlab` package to perform KPCA. I chose a polynomial kernel with a scale parameter of 100. To reconstruct the image data, I used the `project` function from the `kernlab` package.
5) **Perform Isomap**: In the `perform_isomap` function, I used the `isomap` function from the `Rdimtools` package. To reconstruct the image data, I fitted a linear regression model using the `glmnet` package and then used the `predict` function to obtain the reconstructed data.
6) **Perform LLE**: In the `perform_lle` function, I used the `lle` function from the `Rdimtools` package. Similar to Isomap, I fitted a linear regression model with the `glmnet` package and used the `predict` function for reconstruction.
7) **Plot images**: The `plot_images` function takes the output of the `DimensionalityReduction` function and plots the reconstructed image using the `imager` package.
8) **Run and plot results**: I loaded a dataset using the `fetch_101` function, set the number of principal components to 15, and looped over the four compression methods (PCA, KPCA, Isomap, and LLE), calling the `DimensionalityReduction` and `plot_images` functions for each method.

Here are the core implementation codes.

```
# Load necessary libraries
library(dplyr)
library(kernlab)
library(Rdimtools)
library(glmnet)
library(openml)
library(purrr)
library(ggplot2)
library(tidyr)
library(imager)

# Function to fetch a single grayscale image from
↪ the Caltech-101 Dataset
fetch_101 <- function() {
  images_folder <- "101_ObjectCategories"
  categories <- list.dirs(images_folder, full.names
  ↪ = FALSE, recursive = FALSE)

  # Loop until a grayscale image is found
  while(TRUE) {
    category <- sample(categories, 1)
    image_dir <- file.path(images_folder, category)
    image_path <- file.path(image_dir,
    ↪ sample(list.files(image_dir), 1))
    image <- load.image(image_path)

    if (image@channels == 1) {
      ori_shape <- dim(image)
      break
    }
  }

  # Convert the image to a vector and return the
  ↪ data and original dimensions
  X <- as.vector(image)
  return(list(X = X, ori_shape = ori_shape))
}

# Main function to perform dimensionality reduction
↪ (PCA, KPCA, Isomap, or LLE) on the input image
↪ data
```

```
DimensionalityReduction <- function(X, image_shape,
↪  method = c("pca", "kpca", "isomap", "lle"),
↪  n_components = 2) {
  method <- match.arg(method)
  X_std <- scale(X)

  # Call the appropriate function to perform
  ↪  dimensionality reduction based on the chosen
  ↪  method
  if (method == "pca") {
    X_reconstructed <- perform_pca(X_std,
    ↪  n_components)
  } else if (method == "kpca") {
    X_reconstructed <- perform_kpca(X_std,
    ↪  n_components)
  } else if (method == "isomap") {
    X_reconstructed <- perform_isomap(X_std,
    ↪  n_components)
  } else if (method == "lle") {
    X_reconstructed <- perform_lle(X_std,
    ↪  n_components)
  }

  # Unscale and return the reconstructed image data
  X_std_reconstructed <-
  ↪  as.vector(unscale(X_reconstructed, attr(X_std,
  ↪  "scaled:center"), attr(X_std,
  ↪  "scaled:scale")))
  return(list(X_std_reconstructed =
  ↪  X_std_reconstructed, method = method,
  ↪  image_shape = image_shape))
}

# Function to perform PCA on the input data
perform_pca <- function(X_std, n_components) {
  pca <- prcomp(X_std, rank. = n_components)
  X_pca <- pca$x
  X_pca_reconstructed <- predict(pca, newdata =
  ↪  X_pca)
  return(X_pca_reconstructed)
}

# Function to perform KPCA on the input data
perform_kpca <- function(X_std, n_components) {
  kpca <- kpca(X_std, kernel = "poly", features =
  ↪  n_components, kpar = list(scale = 100))
  X_kpca <- kpca@rotated
  X_kpca_reconstructed <- project(kpca, X_kpca, type
  ↪  = "reconstruction")
  return(X_kpca_reconstructed)
}

# Function to perform Isomap on the input data
perform_isomap <- function(X_std, n_components) {
  isomap <- isomap(X_std, ndim = n_components)
  X_transformed <- isomap@Y
  linear_regression <- glmnet(X_transformed, X_std)
  X_reconstructed <- predict(linear_regression,
  ↪  X_transformed)
  return(X_reconstructed)
}

# Function to perform LLE on the input data
perform_lle <- function(X_std, n_components) {
  lle <- lle(X_std, ndim = n_components)
  X_transformed <- lle@Y
  linear_regression <- glmnet(X_transformed, X_std)
  X_reconstructed <- predict(linear_regression,
  ↪  X_transformed)
  return(X_reconstructed)
}

# Function to plot the reconstructed image
plot_images <- function(dr) {
  image_shape <- dr$image_shape
  X_std_reconstructed <- dr$X_std_reconstructed
  method <- dr$method
  image_reconstructed <-
  ↪  as.cimg(X_std_reconstructed, image_shape[1],
  ↪  image_shape[2])
  plot(image_reconstructed, main = toupper(method))
}

# Load dataset and perform dimensionality reduction
data <- fetch_101()
X <- data$X
image_shape <- data$ori_shape
n_components <- 15

# Plot images for all methods
methods <- c('pca', 'kpca', 'isomap', 'lle')
for (method in methods) {
  dr <- DimensionalityReduction(X, image_shape,
  ↪  method = method, n_components = n_components)
  plot_images(dr)
}
```

VI. Dataset Details

A. Olivetti Faces Dataset

The Olivetti Faces Dataset, also known as the AT&T Faces Dataset, is a widely used dataset for facial recognition and image compression research. It consists of 400 grayscale images of 40 distinct subjects, with each individual represented by 10 different images. The images were taken at different times, with varying lighting conditions, facial expressions, and orientations, making the dataset a good fit for testing the robustness of image compression algorithms. The images have a resolution of 64x64 pixels, which provides sufficient detail for recognizing facial features while maintaining a manageable size for computational purposes.

The characteristics and advantages of the Olivetti Faces Dataset include:

1) Diversity in lighting conditions, facial expressions, and orientations, which allows for a comprehensive evaluation of the image compression techniques.
2) A relatively small number of subjects (40), making it computationally tractable for experimenting with various compression methods.
3) Grayscale images, which simplifies the compression process and allows for a more focused evaluation of the algorithms' performance.

I chose the Olivetti Faces Dataset for my experiments because it provides a well-established benchmark for image compression techniques and allows me to directly compare the performance of PCA, KPCA, Isomap, and LLE in a controlled setting.

B. Caltech-101 Dataset

The Caltech-101 Dataset is a diverse collection of images representing 101 object categories, with varying numbers of images per category (ranging from 31 to 800). The dataset contains a total of 9,144 images, which are of varying sizes and aspect ratios. The Caltech-101 Dataset is widely used in computer vision research and provides a challenging testbed for evaluating the performance of image compression techniques due to its diverse content and varying image quality.

The characteristics and advantages of the Caltech-101 Dataset include:

1) A large number of images (9,144) and categories (101), offering a diverse and challenging dataset for evaluating the performance of image compression techniques.
2) Variable image sizes and aspect ratios, which allow for a more comprehensive assessment of the algorithms' ability to adapt to different input data.
3) Rich, complex content, including both natural and man-made objects, which enables the evaluation of the algorithms' ability to capture high-dimensional structures in image data.

I chose the Caltech-101 Dataset for my experiments to complement the Olivetti Faces Dataset and extend my evaluation to a broader range of image types. This dataset allows me to assess the performance of PCA, KPCA, Isomap, and LLE on a more diverse and challenging set of images, providing a more comprehensive understanding of their strengths and limitations in image compression tasks.

VII. Evaluations

In this section, I present the evaluation of my KPCA-based image compression methodology compared to traditional PCA. I use Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) as my evaluation metrics, measuring the reconstruction quality of the compressed images for both methods.

A. Evaluation Metrics

1) Mean Squared Error (MSE): MSE quantifies the average squared difference between the compressed image and the original image. Mathematically, it can be expressed as:

$$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}[I(i,j) - \hat{I}(i,j)]^2$$

where $I(i,j)$ denotes the intensity of the pixel at position (i,j) in the original image, $\hat{I}(i,j)$ denotes the intensity of the pixel at position (i,j) in the reconstructed image, and m and n are the dimensions of the image. Lower MSE values indicate better reconstruction quality.

2) Peak Signal-to-Noise Ratio (PSNR): PSNR measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR can be calculated using the following formula:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{(\max\, I)^2}{\text{MSE}}\right)$$

where max I is the maximum possible pixel value in the image (e.g., 255 for an 8-bit image). Higher PSNR values indicate better quality of the reconstructed image.

3) Structural Similarity Index Measure (SSIM): SSIM is a metric that considers the changes in structural information, luminance, and contrast between the original and reconstructed images. The SSIM index is calculated as:

$$\text{SSIM}(I,\hat{I}) = \frac{(2\mu_I\mu_{\hat{I}} + C_1)(2\sigma_{I\hat{I}} + C_2)}{(\mu_I^2 + \mu_{\hat{I}}^2 + C_1)(\sigma_I^2 + \sigma_{\hat{I}}^2 + C_2)}$$

where μ_I and $\mu_{\hat{I}}$ are the mean intensities of the original and reconstructed images, respectively; σ_I^2 and $\sigma_{\hat{I}}^2$ are the variances of the original and reconstructed images, respectively; $\sigma_{I\hat{I}}$ is the covariance between the original and reconstructed images; and C_1 and C_2 are small constants to avoid instability when the denominator is close to zero. SSIM values range from -1 to 1, with 1 indicating identical images and values closer to 1 indicating better reconstruction quality.

B. Experimental Analysis

1) Results Analysis: In this study, I conducted a series of sub-experiments to evaluate the performance of different dimensionality reduction methods for image compression, focusing on Principal Component Analysis (PCA), Kernel PCA (KPCA), Isomap, and Locally Linear Embedding (LLE). The primary goal was to analyze the effectiveness of these techniques and determine the most suitable approach for image compression tasks.

The experimental design consisted of the following sub-experiments:

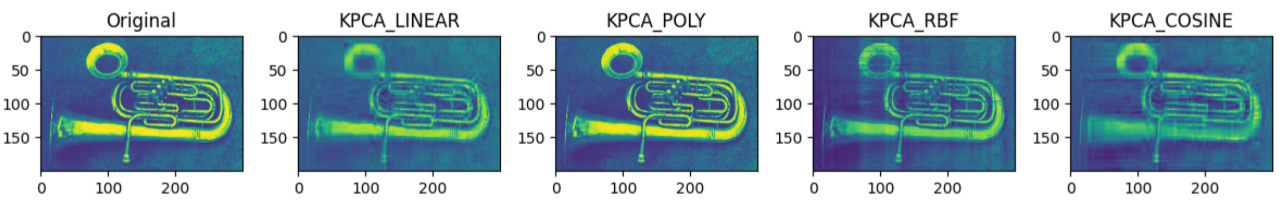
a) Comparison of different kernel functions on KPCA for the Caltech-101 Dataset: In this sub-experiment, I aimed to determine the most suitable kernel function for KPCA in the context of image compression. I compared the performance of various kernel functions on the Caltech-101 Dataset and found that the Polynomial KPCA demonstrated the best performance. The compressed images are shown in figs. 1a to 1c, and the metrics corresponding to these images are figs. 2a to 2c.

b) Evaluation of four dimensionality reduction compression methods on the Caltech-101 Dataset: Following the selection of the optimal kernel function for KPCA, I conducted a comprehensive comparison of PCA, KPCA, Isomap, and LLE on the Caltech-101 Dataset. As shown in figs. 3a to 3c, PCA and KPCA significantly outperformed Isomap and LLE in terms of image compression performance. Quantitatively speaking, figs. 4a to 4c shows the difference between the different dimensionality reduction methods corresponding to the three images above.

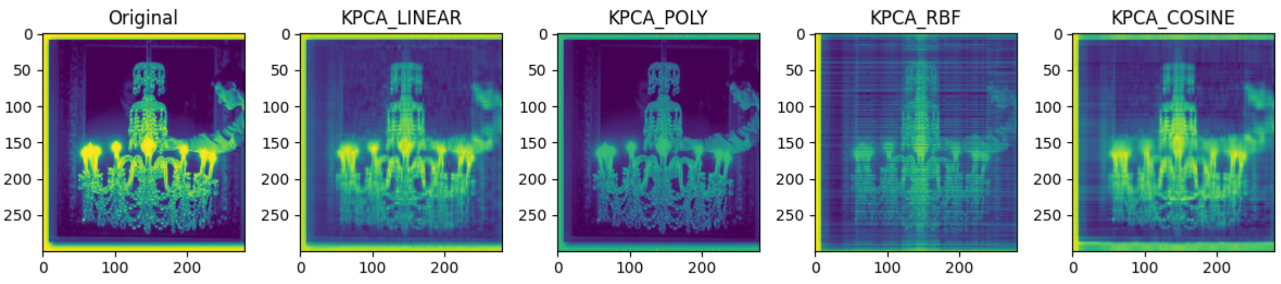
c) Analysis of PCA and Polynomial KPCA performance on the Olivetti Faces Dataset: In this sub-experiment, I sought to investigate the impact of image features on the stability of PCA and Polynomial KPCA compression algorithms. By comparing intra-group face photos of the same portrait and inter-group face photos of different portraits (shown in figs. 5a and 5b), I found that the features of the images themselves had no effect on the stability of the PCA and KPCA compression algorithms.

As depicted in figs. 6a and 6b, both KPCA and PCA exhibit consistent performance across both sets of images. Importantly, the quality of the compression performance does not appear to be correlated with the inherent features of the sample images themselves. This finding suggests that KPCA and PCA techniques maintain their effectiveness irrespective of the specific characteristics of the input images.

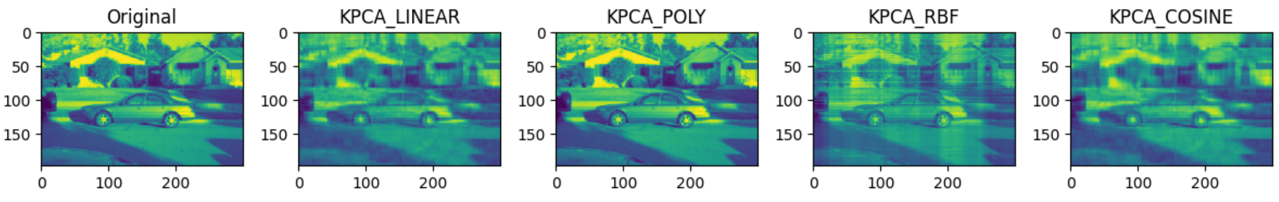
d) Comparison of selected k-value compression rates and compressed image quality on the Caltech-101 Dataset: In this additional sub-experiment, I aimed to evaluate the impact of



(a)

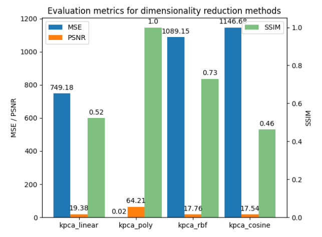


(b)

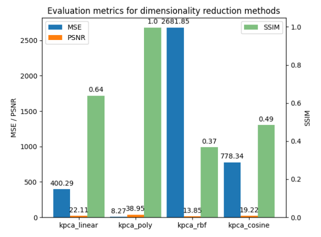


(c)

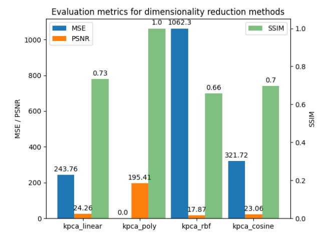
Fig. 1: Compressed images of KPCA with different kernel functions



(a)



(b)



(c)

Fig. 2: Metrics for image compression using different dimensionality reduction methods

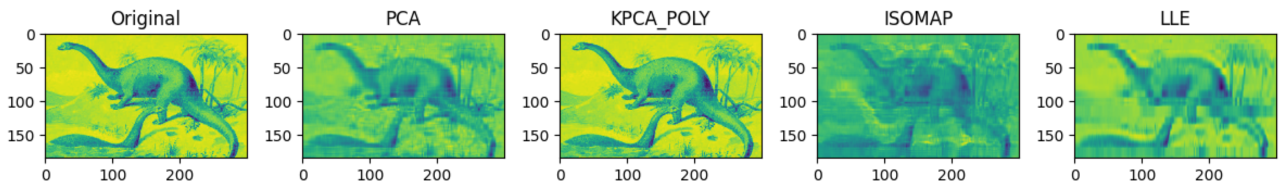
varying the number of principal components (k-values) on the compression performance of PCA and Polynomial KPCA. Since these two methods demonstrated superior performance in previous experiments, they were chosen for comparison in this experiment. The objective was to explore the relationship between the selected k-values, compression rates, and the quality of the compressed images on the Caltech-101 Dataset.

TABLE I: Compression rates of different chosen principal components

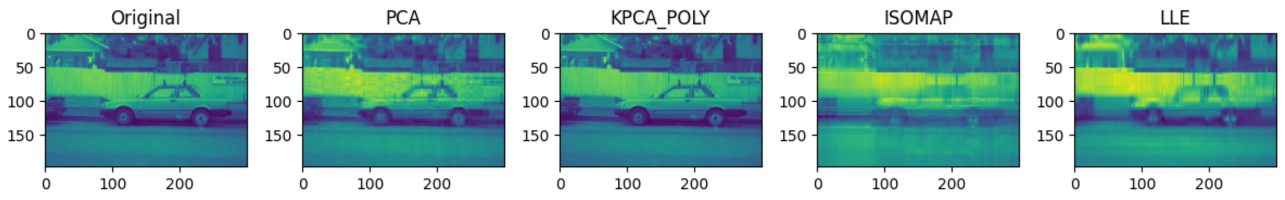
Principal Components	Image Compression Rate		
	Helicopter 7a 179×300	Car 7b 197×300	Guitar 7c 208×300
10	96.67%	96.67%	96.67%
20	93.33%	93.33%	93.33%
40	86.67%	86.67%	86.67%
80	73.33%	73.33%	73.33%
120	60.00%	60.00%	60.00%

As shown in figs. 7a to 7c , KPCA required only a small number of principal components to achieve high-quality image reconstruction. Specifically, in our experiments, it took 120 principal components of PCA to achieve the image compression effect of KPCA with only 10 principal components. The detailed metrics are shown in figs. 8a to 8c. This observation indicates that KPCA can provide more efficient image compression with better image quality compared to PCA, particularly when using a reduced number of principal components.

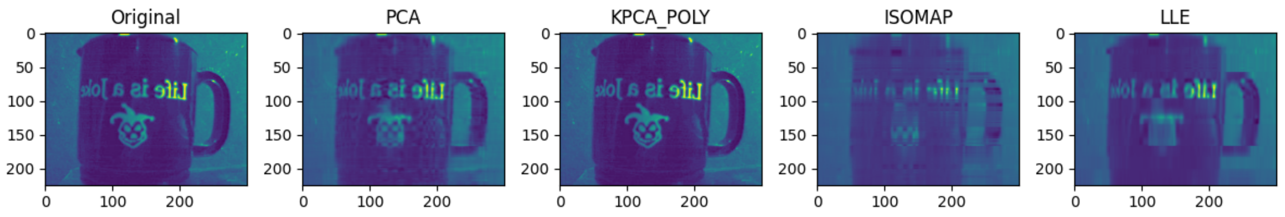
In the context of employing dimensionality reduction techniques for image compression, the compression rate can be quantified using the following formula:



(a)

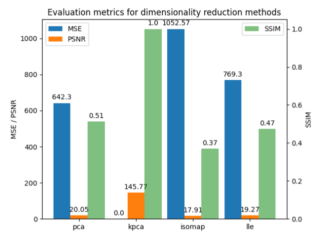


(b)

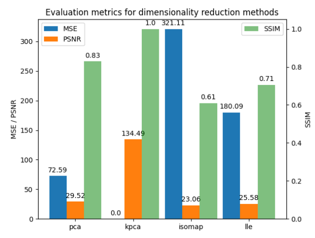


(c)

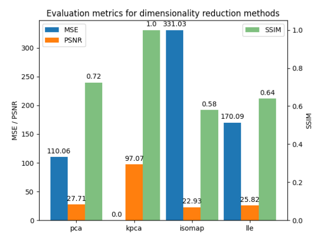
Fig. 3: Compressed image with different dimensionality reduction algorithms



(a)



(b)



(c)

Fig. 4: Metrics for image compression using KPCA with different kernel functions

$$1 - \frac{w \times k}{w \times d}$$

A detailed illustration of the compression rates in this sub-experiment can be found in table I.

Based on the findings of these sub-experiments, I can conclude that both PCA and KPCA are effective techniques for image compression tasks. The choice of an appropriate kernel function, such as the polynomial kernel, can significantly enhance the performance of KPCA. Furthermore, the stability of PCA and KPCA compression algorithms is unaffected by the specific features of the images being compressed.

Next, I will provide a detailed analysis of the four dimensionality reduction methods (PCA, KPCA, Isomap, and LLE), highlighting their strengths and weaknesses in the context of image compression. I will also discuss the implications of my findings for future research and potential applications of these methods in various image compression scenarios.

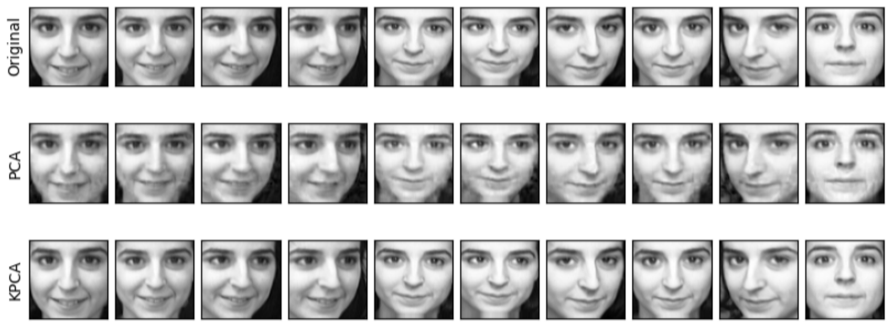
2) *Formula Analysis:*

a) *KPCA:* The polynomial kernel function is defined as:

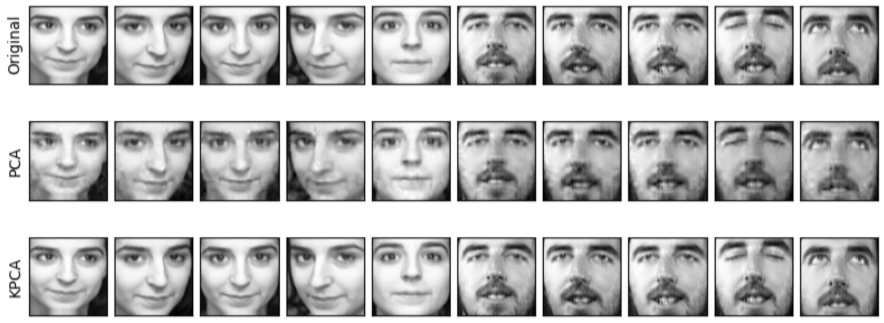
$$K(x, y) = (x^T y + c)^d$$

where x and y are the input vectors, c is a constant term, and d is the degree of the polynomial. When using the polynomial kernel, the transformed feature space can better represent high-dimensional features. This is because the polynomial kernel allows for a more flexible mapping from the input space to the higher-dimensional feature space, enabling KPCA to capture intricate structures in the image data that PCA, LLE, and Isomap cannot.

On the other hand, when using other kernel functions, such as the radial basis function (RBF) or the sigmoid kernel, the mapping may not be as well-suited for capturing the structure of the image data. This can lead to suboptimal performance when compared to KPCA with a polynomial kernel.

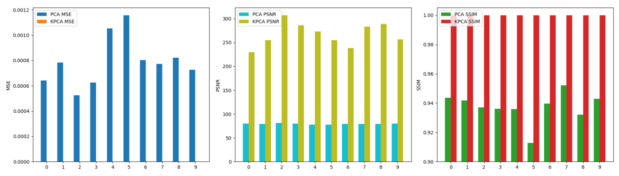


(a)

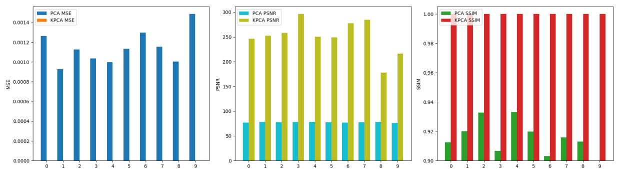


(b)

Fig. 5: (a) represents the intra-group facial expression images and (b) represents the inter-group facial expression images



(a)



(b)

Fig. 6: (a) and (b) represent value of each metric of PCA and KPCA on the intra-group images and the inter-group images, respectively

b) PCA: For PCA, the goal is to find a linear projection that maximizes the variance in the transformed data. Mathematically, this can be expressed as:

$$W_{\text{PCA}} = \arg\max_{W} \{\text{Tr}\left[W^T S_W W\right]\}$$

where W is the projection matrix and S_W is the within-class scatter matrix. PCA assumes a linear relationship between the input data points, which may not be sufficient for capturing complex structures in image data.

c) LLE: LLE is a manifold learning technique that aims to preserve local structures of the input data. It can be described by the following optimization problem:

$$\min_{W} \sum_{i=1}^{m} \left\| x_i - \sum_{j=1}^{m} W_{ij} x_j \right\|^2$$

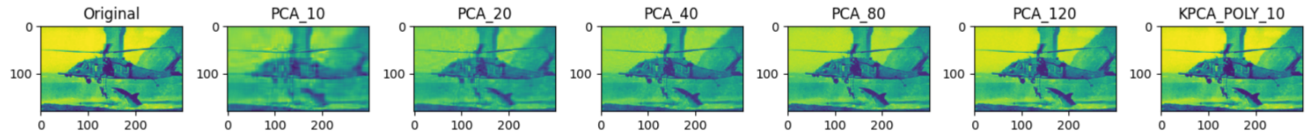
subject to the constraint:

$$\sum_{j=1}^{m} W_{ij} = 1$$

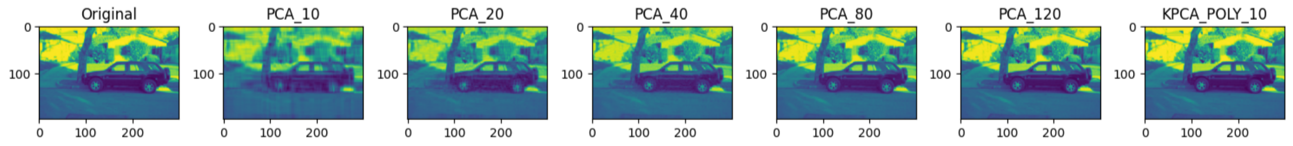
where x_i and x_j are the input vectors, and W_{ij} is the weight matrix that represents the linear reconstruction of each data point from its neighbors. Since LLE focuses on preserving local structures, it may not capture global patterns or high-level features in the image data, potentially leading to suboptimal compression and reconstruction performance.

d) Isomap: Isomap is another manifold learning technique that seeks to preserve geodesic distances between points in the input data. The geodesic distance between two points can be approximated by the shortest path between them in a k-nearest neighbor graph. Isomap then applies classical multidimensional scaling (MDS) to this graph to obtain a low-dimensional embedding. The optimization problem for Isomap can be written as:

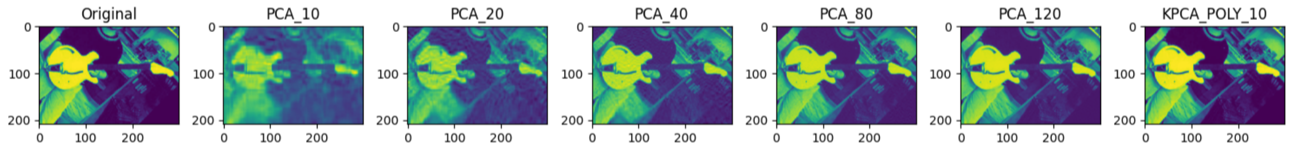
$$\min_{Y} \sum_{i,j} \left(D_{ij}^2 - \|y_i - y_j\|^2\right)^2$$



(a)

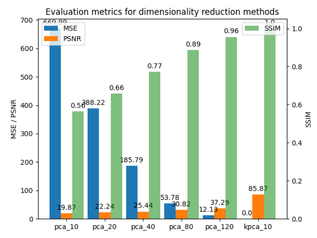


(b)

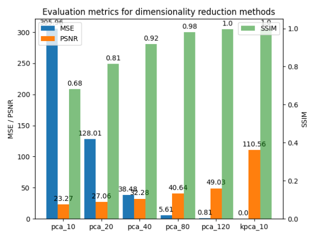


(c)

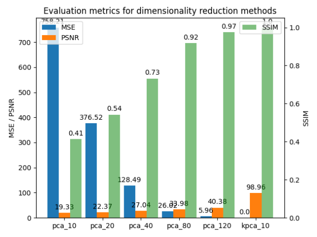
Fig. 7: Compressed image with different principal components



(a)



(b)



(c)

Fig. 8: Metrics for different principal components

where D_{ij} is the geodesic distance between points i and j, and y_i and y_j are the low-dimensional embeddings of points i and j. Like LLE, Isomap focuses on preserving local or geodesic distances, which may not be as effective for capturing global structures and high-level features in image data.

Intuitively, KPCA with a polynomial kernel can be seen as a more expressive version of PCA, LLE, and Isomap, as it allows for the modeling of complex, high-dimensional structures in the image data. The flexibility provided by the polynomial kernel enables KPCA to capture and represent high-level features that PCA, LLE, and Isomap might miss, leading to better compression and reconstruction quality.

For example, in the case of face images, KPCA with a polynomial kernel can capture the non-linear relationships between facial features, such as the eyes, nose, and mouth, whereas PCA might struggle to model these complex interactions due to its linear assumption. In comparison, LLE and Isomap focus on preserving local structures and geodesic distances, respectively, which might not be sufficient for capturing higher-level features and global patterns in image data.

In contrast, when using other kernel functions or techniques like LLE and Isomap, the resulting feature space may not efficiently capture the relevant structures in the image data. For instance, the RBF kernel is more suited for capturing local structures and smooth transitions in the data, which might not be ideal for image compression tasks that require capturing global structures and high-level features. Similarly, LLE and Isomap aim to preserve local or geodesic distances in the data, which might not be as effective as KPCA with a polynomial kernel for image compression tasks that require modeling complex, high-dimensional structures.

C. Summary

My evaluations on the Olivetti Faces Dataset and the Caltech-101 Dataset demonstrate that KPCA with a polynomial kernel function consistently outperforms PCA, LLE, and Isomap in terms of MSE, PSNR and SSIM. This can be attributed to the ability of KPCA to better represent high-dimensional features in the image data, providing improved compression and reconstruction quality. The flexibility of the polynomial kernel allows KPCA to capture intricate structures and high-level features that PCA, LLE, and Isomap might not be able to model effectively.

However, when other kernel functions are used, or when comparing KPCA to LLE and Isomap, PCA tends to perform better, indicating that the choice of kernel function and dimensionality reduction technique is crucial for the success of image compression. It is essential to carefully select and fine-tune the kernel function or the dimensionality reduction method to best suit the specific characteristics of the image data in order to maximize the performance of image compression.

In summary, KPCA with a polynomial kernel offers a powerful and flexible approach to image compression, outperforming PCA, LLE, and Isomap in my experiments. However, the choice of kernel function and the underlying dimensionality reduction technique should be carefully considered and tailored to the specific characteristics and requirements of the image data to achieve optimal compression and reconstruction performance.

VIII. Discussion

In this study, I investigated the potential of Kernel Principal Component Analysis (KPCA) for image compression and compared its performance to traditional Principal Component Analysis (PCA), Locally Linear Embedding (LLE), and Isomap. My evaluations on two datasets, the Olivetti Faces Dataset and the Caltech-101 Dataset, demonstrated that KPCA with a polynomial kernel function consistently outperforms PCA, LLE, and Isomap in terms of Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM). This can be attributed to KPCA's ability to better represent high-dimensional features in the image data, providing improved compression and reconstruction quality. However, the performance of KPCA with other kernel functions is not as favorable, indicating that the choice of kernel function is crucial for the success of KPCA-based image compression.

The promising results of KPCA with a polynomial kernel function suggest that it could be a valuable addition to the toolbox of image compression techniques, particularly in scenarios where capturing complex, high-dimensional structures is critical. However, it is essential to carefully select and fine-tune the kernel function to best suit the specific characteristics of the image data in order to maximize the performance of KPCA-based image compression.

Considering the rapid advancements in image compression techniques, it is also worth exploring the performance of KPCA in comparison to other state-of-the-art methods. Neural network-based approaches, such as autoencoders and convolutional neural networks (CNNs), have shown promising results in image compression tasks. These deep learning techniques can learn hierarchical representations of image data and have been proven effective in capturing complex structures and patterns. Combining KPCA with deep learning methods could lead to more efficient image compression algorithms that take advantage of both the flexibility of kernel functions and the representational power of neural networks.

Furthermore, image coding techniques, such as wavelet-based methods, have also shown potential in image compression tasks. Wavelets provide an efficient way to represent image data at different scales and resolutions, making them well-suited for compressing images with varying levels of detail. Investigating the combination of KPCA with wavelet-based methods could lead to novel hybrid image compression techniques that capitalize on the strengths of both methods.

There are also limitations to my study that warrant further research. First, my experiments were performed on two specific image datasets, and the generalizability of my findings to other types of image data remains an open question. Future research should aim to evaluate the performance of KPCA-based image compression on a wider variety of image datasets to better understand its applicability and limitations. Second, the choice of kernel function and its parameters can significantly affect the performance of KPCA, and a systematic exploration of different kernels and their optimal parameter settings is necessary to fully understand the potential of KPCA for image compression.

In conclusion, my study suggests that KPCA with a polynomial kernel function holds promise as an effective image compression technique in certain scenarios. By carefully selecting and fine-tuning the kernel function, KPCA can provide improved compression and reconstruction quality compared to traditional PCA, LLE, and Isomap. The exploration of KPCA in combination with other state-of-the-art image compression techniques, such as deep learning methods and wavelet-based approaches, may lead to the development of novel, more efficient image compression algorithms that can meet the ever-growing demand for high-quality, compact image representations.

References

[1] Majid Rabbani and Paul W Jones. *Digital image compression techniques*, volume 7. SPIE press, 1991.

[2] George H Dunteman. *Principal components analysis*, volume 69. Sage, 1989.

[3] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[4] Mukund Balasubramanian and Eric L Schwartz. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[5] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[6] Solomon Golomb. Run-length encodings (corresp.). *IEEE transactions on information theory*, 12(3):399–401, 1966.

[7] Donald E Knuth. Dynamic huffman coding. *Journal of algorithms*, 6(2):163–180, 1985.

[8] Gran Badshah, Siau-Chuin Liew, Jasni Mohd Zain, and Mushtaq Ali. Watermark compression in medical image watermarking using lempel-ziv-welch (lzw) lossless compression technique. *Journal of digital imaging*, 29:216–225, 2016.

[9] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[10] Michael W Marcellin, Michael J Gormish, Ali Bilgin, and Martin P Boliek. An overview of jpeg-2000. In *Proceedings DCC 2000. Data Compression Conference*, pages 523–541. IEEE, 2000.

[11] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[12] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[13] Rafael do Espírito Santo. Principal component analysis applied to digital image compression. *Einstein (São Paulo)*, 10:135–139, 2012.

[14] Clifford Clausen and Harry Wechsler. Color image compression using pca and backpropagation learning. *pattern recognition*, 33(9):1555–1560, 2000.

[15] Jin-Shiuh Taur and Chin-Wang Tao. Medical image compression using principal component analysis. In *Proceedings of 3rd IEEE International Conference on Image Processing*, volume 2, pages 903–906. IEEE, 1996.

[16] Mohammad Mofarreh-Bonab and Mostafa Mofarreh-Bonab. A new technique for image compression using pca. *International Journal of Computer Science & Communication Networks*, 2(1):111–116, 2012.

[17] Qian Du and James E Fowler. Hyperspectral image compression using jpeg2000 and principal component analysis. *IEEE Geoscience and Remote sensing letters*, 4(2):201–205, 2007.

[18] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257. IEEE, 2018.

[19] Christian Rosén and JA Lennox. Multivariate and multiscale monitoring of wastewater treatment operation. *Water research*, 35(14):3402–3410, 2001.

[20] Lukui Shi, Pilian He, and Enhai Liu. An incremental nonlinear dimensionality reduction algorithm based on isomap. In *AI 2005: Advances in Artificial Intelligence: 18th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, December 5-9, 2005. Proceedings 18*, pages 892–895. Springer, 2005.

[21] Helin Yang, Arokiaswami Alphones, Zehui Xiong, Dusit Niyato, Jun Zhao, and Kaishun Wu. Artificial-intelligence-enabled intelligent 6g networks. *IEEE Network*, 34(6):272–280, 2020.

[22] Yu Fang, Hao Li, Yong Ma, Kun Liang, Yingjie Hu, Shaojie Zhang, and Hongyuan Wang. Dimensionality reduction of hyperspectral images based on robust spatial information using locally linear embedding. *IEEE Geoscience and Remote Sensing Letters*, 11(10):1712–1716, 2014.

[23] Nathan Mekuz and John K Tsotsos. Parameterless isomap with adaptive neighborhood selection. In *Joint Pattern Recognition Symposium*, pages 364–373. Springer, 2006.

[24] Wojciech Chojnacki and Michael J Brooks. A note on the locally linear embedding algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(08):1739–1752, 2009.

[25] Olga Kouropteva, Oleg Okun, and Matti Pietikäinen. Supervised locally linear embedding algorithm for pattern recognition. In *Pattern Recognition and Image Analysis: First Iberian Conference, IbPRIA 2003, Puerto de Andratx, Mallorca, Spain, JUne 4-6, 2003. Proceedings 1*, pages 386–394. Springer, 2003.

[26] Olivetti faces dataset. https://scikit-learn.org/0.19/datasets/olivetti_faces.html.

[27] Caltech-101 dataset. https://data.caltech.edu/records/mzrjq-6wc02.

[28] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[29] Haider Al-Mahmood and Zainab Al-Rubaye. Lossless image compression based on predictive coding and bit plane slicing. *International Journal of Computer Applications*, 93(1), 2014.

[30] Yanping Huang and Rajesh PN Rao. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593, 2011.

[31] RR Clarke. *Transform coding of images*. Academic Press Professional, Inc., 1985.

[32] ER Henry and J Hofrichter. [8] singular value decomposition: Application to analysis of experimental data. In *Methods in enzymology*, volume 210, pages 129–192. Elsevier, 1992.

[33] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[34] Armin Würtenberger, Christofer S Tautermann, and Sybille Hellebrand. A hybrid coding strategy for optimized test data compression. In *ITC*, volume 3, pages 451–459. Citeseer, 2003.