A Project Report

on

IMAGE COMPRESSION BASED ON K-MEANS CLUSTERING AND PRINCIPAL COMPONENT ANALYSIS

Submitted in the Partial Fulfillment of the Requirements for the award of

Bachelor of Technology
in
Electronics and Communication Engineering

By

Aman Kumar Sinha (20175006)
Akul Kumar (20175026)
Aman Kumar (20175068)

Under the guidance of

Prof. Haranath Kar
Professor

Department of Electronics and Communication Engineering
Motilal Nehru National Institute of Technology Allahabad
Pryagraj-211004, INDIA

December 2020

UNDERTAKING

We declare that the work presented in this project titled **Image Compression based on K-means Clustering and Principal Component Analysis**, submitted to the DEPARTMENT OF ELECTRONICS and COMMUNICATION ENGINEERING, MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY ALLAHABAD, PRAYAGRAJ for the award of the **Bachelor of Technology** degree in *Electronics and Communication Engineering* is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, we accept that our degree may be unconditionally withdrawn.

Date: 16/12/2020

Aman Kumar Sinha
Akul Kumar
Aman Kumar

Department of Electronics and Communication Engineering

Motilal Nehru National Institute of Technology Allahabad

Prayagraj, INDIA

CERTIFICATE

This is to certify that the work contained in the project titled **Image Compression based on K-means Clustering and Principal Component Analysis**, submitted by **Aman Kumar Sinha**, **Akul Kumar** and **Aman Kumar** in the partial fulfillment of the requirements for the award of Bachelor of Technology in Electronics and Communication Engineering to the Department of Electronics and Communication Engineering, Motilal Nehru National Institute of Technology Allahabad, is a bonafide work of the students carried out under my supervision.

Date: 16th December, 2020
Place: Prayagraj, India

Prof. Haranath Kar
Professor
ECE Department
MNNIT Allahabad

Acknowledgement

I would like to express my special thanks of gratitude to my supervisor Prof. Haranath Kar as well as Head of Department ECE and other faculty members who gave me the golden opportunity to do this wonderful project on the topic **Image Compression based on K-means Clustering and Principal Component Analysis**, which also helped me in doing a lot of Research and I came to know about so many new things. I am really thankful to them.
Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

Date: 16th December, 2020

Place: Prayagraj, India

Aman Kumar Sinha

Akul Kumar

Aman Kumar

ABSTRACT

Image compression is a type of data compression applied to digital images without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from web pages.

In this project, we have implemented the K-means clustering algorithm and applied it to compress an image. In the second part of our project, we have used principal component analysis to find a low-dimensional representation of face images. K-means clustering is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (k), which are represented by their centroids. Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Image compression can be used to reduce the redundancy of the image and to store or transmit data in an efficient form and for website optimization. Sites with uncompressed images can take longer to load, and can cause visitors to bounce. It can also be used for sending and uploading images and for reducing the storage impact on our hard drive as well as databases. Uploading an uncompressed image can take a while, and some email servers have a file size limit.

Table of Contents

List of Figures

Abbreviations

DCT:	Discrete Cosine Transform
GA:	Genetic Algorithm
MSE:	Mean Square Error
PCA:	Principal Component Analysis
PSNR:	Peak Signal to Noise Ratio

Chapter 1
Introduction

1.1 Introduction

Image compression is a type of data compression applied to digital images without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from web pages. Image compression refers to reducing the dimensions, pixels, or colour components of an image so as to reduce the cost of storing or performing operations on them. Some image compression techniques also identify the most significant components of an image and discard the rest, resulting in data compression as well. Image compression can therefore be defined as a data compression approach that reduces the data bits needed to encode an image while preserving image details

1.2 Motivation

The rapid growth of digital technology has paved the way for many image processing applications and motivated the need for better compression algorithms. Image compression is the technique where the size of an image is minimized by keeping the quality of the image to an acceptable level. The main objective of image compression is to decrease the redundancy of the image thereby increasing the capacity of storage and efficient transmission.

Image compression can be used to reduce the redundancy of the image and to store or transmit data in an efficient form and for website optimization. Sites with uncompressed images can take longer to load, and can cause visitors to bounce. It can also be used for sending and uploading images and for reducing the storage impact on our hard drive as well as databases. Uploading an uncompressed image can take a while, and some email servers have a file size limit.

1.3 Basic concepts

1.3.1 K-means algorithm

The K-means algorithm solves the clustering problem of data in multidimensional space. Supposing there is a data set $\{\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)},\ldots,\boldsymbol{x}^{(N)}\}$, it is an n-dimensional data consisting random variable $\boldsymbol{x}$ with N samples. Since the clustering algorithm is unsupervised learning, there is no target attribute y. Assuming that the K value is given, the goal of the K-means clustering algorithm is to divide the data into K clusters. Since the distance between data in the same cluster should be smaller than the distance with the data outside the cluster, μ_k (mean of kth cluster), $k = 1,2,\ldots,K$ is introduced as a vector, which can be considered the center of the Kth cluster, also known as the centroid.

The goal of the K-means algorithm is to find such a group $\{\boldsymbol{\mu}_k\}$, assign each data to the cluster with the centroid closest to itself, and find minimum accumulation distances of each data point with the cluster center μ_k. For each data $\boldsymbol{x}^{(i)}$, introducing a set of corresponding binary variables $r_{ik} \in \{0,1\}, i=1,2,\ldots,N;, k=1,2,\ldots,K$. if $\boldsymbol{x}^{(i)}$ belonging to cluster k, $r_{ik}=1$, and for any other cluster $j \neq k$, $r_{ij}=0$.

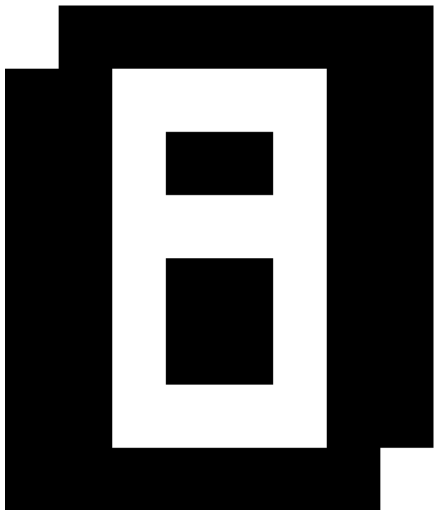
1.3.2 Principal Component Analysis algorithm

Principal component analysis (PCA) is a multivariate technique that analyzes a data table in which observations are described by several inter-correlated quantitative dependent variables. Its goal is to extract the important information from the statistical data to represent it as a set of new orthogonal variables called principal components, and to display the pattern of similarity between the observations and of the variables as points in spot maps. Mathematically, PCA depends upon the eigen-decomposition of positive semi-definite matrices and upon the singular value decomposition (SVD) of rectangular matrices. It is determined by eigenvectors and eigenvalues. Eigenvectors and eigenvalues are numbers and vectors associated to square matrices. Together they provide the eigen-decomposition of a matrix, which analyzes the structure of this matrix such as correlation, covariance, or cross-product matrices. Performing PCA is quite simple in practice. Organize a data set as an $m \times n$ matrix, where m is the number of measurement types and n is the number of trials. Subtract of the mean for each measurement type or row xi. Calculate the SVD or the eigenvectors of the co-variance.

1.4 Existing methods

1.4.1 Transform coding: Discrete Cosine Transform

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The DCT, first proposed by Nasir Ahmed in 1972, is a widely used transformation technique in signal processing and data compression.Discrete Cosine Transforms are the base algorithm for JPEG compression, one of the most widely used image compression algorithms. It works by dividing an image into squares and applying a discrete cosine function to the resulting matrix of values, hence representing the image as a superposition of wavelets rather than pixel values.



(a)

255	0	0	0	0	0	0	0
0	0	255	255	255	255	0	0
0	0	255	0	0	255	0	0
0	0	255	255	255	255	0	0
0	0	255	0	0	255	0	0
0	0	255	0	0	255	0	0
0	0	255	255	255	255	0	0
0	0	0	0	0	0	0	255

(b)

Fig 1.1: (a) Example image in black and white, (b) Its matrix representation

Figure 1.1(a) shows an example image in black and white, within a colour space of 255, its matrix representation is displayed in Figure 1.1(b), with 0 being black, and 255 white pixels. If we apply Direct Cosine Transform equation to the matrix displayed in Figure 1.1(b), we will obtain a representation of our 8x8 grid by superimposing the wavelets shown in Figure 1.2. This process is similar to a two-dimensional Fourier transform, the main difference being the use of cosines exclusively, and the Gamma Function that provides a discrete solution, instead of an infinite continuous one.

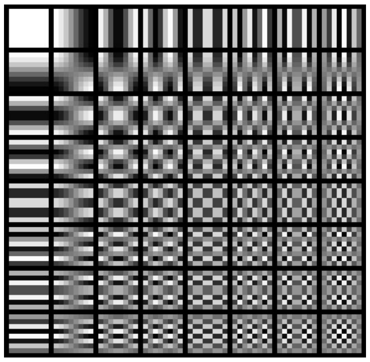
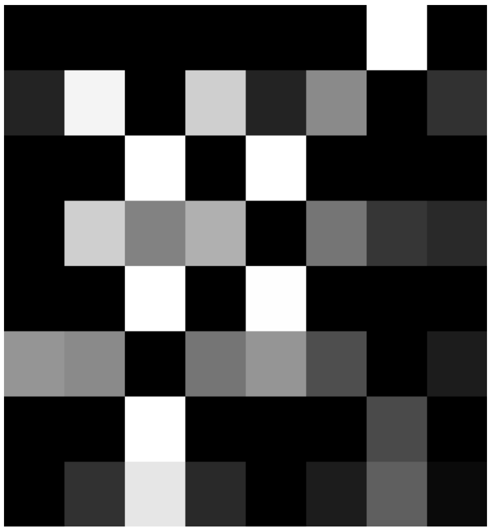


Fig 1.2: Obtained figure after applying DCT equation

The results obtained from the DCT are present in image form in Figure 1.3(a) and in matrix form in Figure1.3(b). In order to recover the original image, each coefficient in matrix in Figure 1.3(b) must be multiplied by the equivalent wavelet in Figure 1.2 and all the results added. This process is known as an Inverse Discrete Cosine Transform. This takes place during the decompression process.


(a)

-3.02	0.00	-2.92	0.00	-1.00	0.00	3.37	0.00
0.14	0.96	-0.18	0.81	0.14	0.54	-0.07	0.19
-0.76	0.00	1.70	0.00	1.84	0.00	-1.29	0.00
-0.39	0.81	0.51	0.69	-0.39	0.46	0.21	0.16
-1.00	0.00	1.84	0.00	1.00	0.00	-0.76	0.00
0.59	0.54	-0.77	0.46	0.59	0.31	-0.32	0.11
-1.84	0.00	2.70	0.00	-0.76	0.00	0.29	0.00
-0.69	0.19	0.90	0.16	-0.69	0.11	0.37	0.04

(b)

Fig 1.3: (a) Image obtained after Inverse DCT, (b) Its matrix representation

The DCT itself does not compress the image, but provides crucial information about the nature of it. The compression process takes place after the DCT. This process, quantisation, will ignore higher frequency patterns (bottom right on Figure 1.3), that are less likely to be perceived as quality loss to the human eye. For a real image, this process would be repeated for all the 8x8, or other size, squares within the entire picture. By reducing the data stored by the amount desired, the process provides compression while limiting the impact of lost data on human perception of the image.

1.4.2 Fractal encoding

Fractals are both naturally occurring phenomenon and mathematical functions that exhibit recurring patterns independently of the scale at which they are observed. This similarity between a mathematical model and nature makes it possible to represent real image data from mathematical equations. Fractal compression is based on the concept of self-similarity, the characteristic that fractal compressible images exhibit. An image with self similarity will have repeating patterns, for example, within a picture of a tree; a leaf may be reproduced by resizing and skewing another leaf, which will achieve similar results to the original image.

Fractal compression algorithms take advantage of Iterated Function Systems. In order to perform this, an image is divided into small squares, in order to form range blocks, this are then compared to larger blocks of the same image, usually with reduced resolution, known as domain blocks. If there is a close enough resemblance between a domain block and a range block, certain operations are applied to the domain block in order to match the range block as closely as possible (Barnsley, 1996). These include: rotation, skew, translation, scale changes, and mirroring.

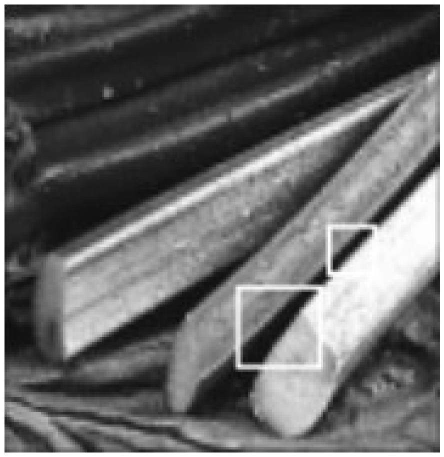


Fig 1.4: Picture of small sticks

Figure 1.4 shows a picture of some sticks, if the small square is the range we're iterating through, and the large square is the domain, we can assume that reducing the domain's size and rotating it slightly, it will produce an image similar to the range. Applying this process to the whole image produces the outcome shown in Figure 1.5.

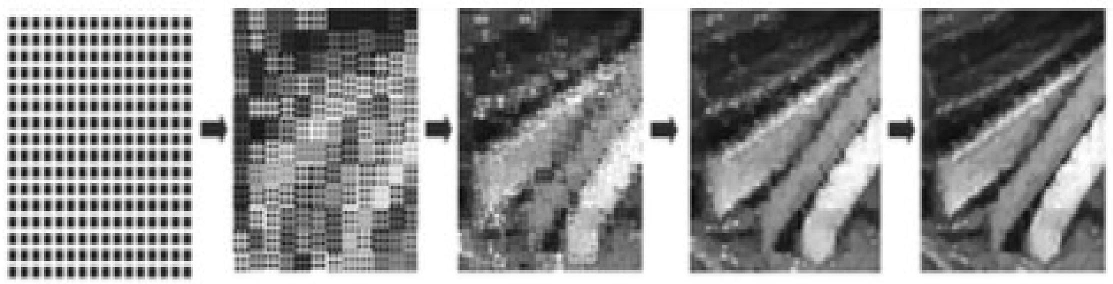


Fig 1.5: Decoding process for fractal image compression

Figure 1.5 shows the decoding process for fractal image compression. It uses the domains in the first image, and applies them in an iterative form in order to generate the subsequent images. The more iterations used, the higher fidelity achieved, at the expense of computational cost and storage size. The aim is to find a compromise between image quality and compression ratio.

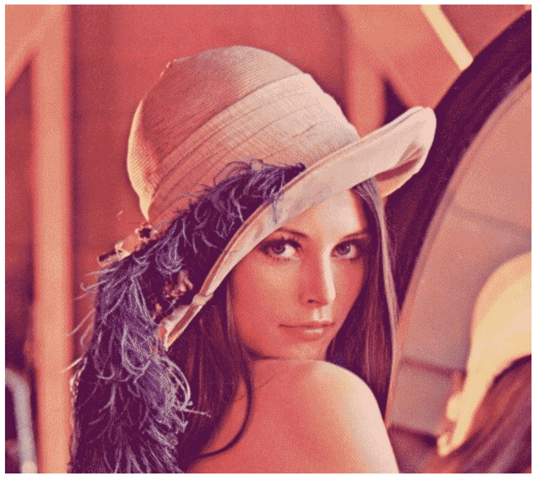
1.4.3 Chroma subsampling

Chroma subsampling takes advantage of the fact that the human eye detects brightness (luminance) more prominently than it detects colour intensity (Chroma). In order to take advantage of this, images have to be encoded in a YCbCr (Luminance, Blue-difference Chroma and Red-difference Chroma) colour space, instead of the traditional RGB (Red, Green Blue). Once the colour space has been changed, the resolution of both red and blue Chroma differences can be dramatically reduced without it being easily perceivable by the human eye.

Figure 1.6 (a), Figure 1.6 (b) and Figure 1.6 (c) show respectively an original full colour image with 256 Chroma points, the same image subsampled to 64 Chroma points, and finally reduced to 8 Chroma points. The last image only has 64 different colours with different luminance applied to them, but the end result is very similar.

(a)

(b)



(c)

Fig 1.6: (a) original full colour image with 256 Chroma points, (b) The same image subsampled to 64 Chroma points, (c) The same image finally reduced to 8 Chroma points

1.4.4 Artificial neural networks

Neural networks use examples to automatically generate rules in order to compute complex calculations. They are generally composed of an input layer, hidden neurons, and an output layer. There may be one or more layer of hidden neurons. Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment.

Neural networks use examples to automatically generate rules in order to compute complex calculations. They are generally composed of an input layer, hidden neurons, and an output layer. There may be one or more layer of hidden neurons (Haykin, 2004). The hidden layer networks are logic functions that, based on input from one or more neurons in preceding layers will output information to one or more neurons in subsequent layers. Each of the connections between the neurons is designed to weight the input into the next neuron, hence if, for example, neuron A on input layer, outputs the value 2.46 to neuron B on hidden layer, there may be a weight, or bias, between both that doubles up the value, and consequently neuron B receives 4.92 as an input. In this case, the weight for the connection (synapses) was 2, doubling up the value, but this weight can be modified by giving a network training data. If a Neural Network is provided with a sufficiently large training data set, eventually the value of the weights in the synapses will be such that the results obtained from a neural network computation will be close to the expected results.

Neural networks have been used to improve image compression extensively. Both as standalone methods as well as combined with other techniques, traditional or more novel (Kattan, 2010).Setiono & Lu (1994) used a standalone neural network in order to compress images, using the image as input data for the network, and the reconstructed (compressed) image as the output. The images used where black and white, and the compression ratio achieved was about 10, producing good quality, albeit noticeably lossy, images. Figure 1.7 shows lena, uncompressed on the left, and compressed with the neural network on the right. The loss of information is particularly noticeable around the hair and the band on the hat, due to this being high frequency areas.

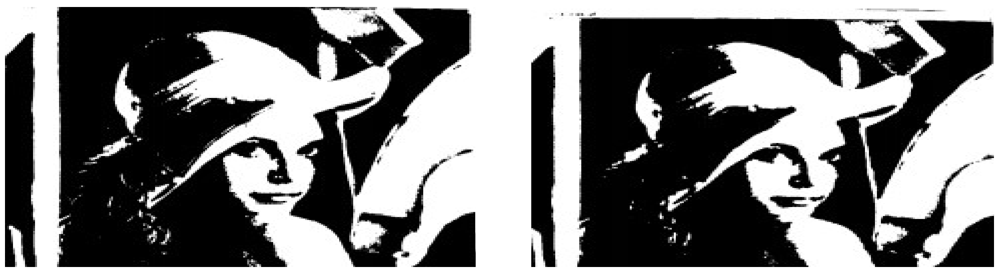

(a) (b)

Fig 1.7: (a) Uncompressed image, (b) Image compressed with neural network

1.4.5 Neural Networks and DCT

Parodi & Passaggio (1994) approached the compression problem by combining DCT and Neural Networks. The approach they take to first divide the image into low and high activity area blocks, which will be compressed at different ratios and by different neural networks. The high and low activity areas are classified as such by using DCT's, high activity areas are those with higher coefficients within the DCT transform, indicating that they contain a wider range of frequencies. Each of these images is posteriorly provided to the corresponding neural network, either high or low capacity, with a number of input neurons proportional to the size of the block to be processed. The images are then compressed by the neural network, which uses the original block as a target for the learning process. The results obtain indicate a significant improvement in image quality over standard neural network compression of images, without compromising compression ratio, they achieve higher PSNR and lower MSE. The authors conclude, after comparing the image to standard JPEG, that despite exhibiting similar PSNR, MSE and compression ratios, the apparent quality of the image is superior, due to the lack of blocking effect (a common artifact in JPEG compressed images), accomplished by storing more bits per pixel on high activity blocks. Parodi & Passaggio (1994) also add that, for images where more blocks are available, more learning will take place by the networks, increasing the final compression ratio.

1.4.6 Neural Networks and fractal encoding

Lee et al (1998) provide a very thorough examination of the image compression process optimisation using neural networks. The approach taken first generates a domain image as well as a range, as is standard for fractal encoding. Once this process has taken place, every domain is mapped to a neuron in the input layer, and every range pixel to one in the output; the luminosity (value of the pixel [0-255]) is the state of the neuron. The neuron connections are formed with the Fractal full search, and each input neuron (domain) may be synapsed to any number of output neurons (range), or none. The network, after a fixed number of iterations, obtains the thresholds and weights required to perform the transforms on the used domain images to obtain the ranges, substituting a step that would usually be taken by the Fractal Encoder. This weights and thresholds are stored and used to generate the decoded image, which obtains, for Lena and Bridge, compression ratios around 7 and PSNR around 27. The authors conclude that the results obtained are comparable to those produced by standard fractal encoding, but unlike the latter, neural networks can be multi-threaded, providing the ability to accelerate the compression and decompression processes dramatically.

1.4.7 Genetic Algorithms

Genetic algorithms can be applied to several image compression problems, but they are mainly used to optimize fractal compression and discrete cosine transforms. Both of these methods can be optimised in different ways

1.4.8 Genetic Algorithms and DCT

One of he approaches taken in order to achieve optimal DCT compression, as is by (Bonyadi & Dehghani, 2008) is to apply a genetic algorithm to improve the quantisation process taken during the final stage of DCT. The process of quantisation is usually performed by applying a zonal mask (Gonzalez & Woods,2007), which dictates which frequencies from the DCT are kept for and which are discarded. This process has a high impact on compression ratio as well as perceived image quality, hence the importance of its optimisation. (Bonyadi & Dehghani, 2008) Use the grey level difference (GLD) in order to quantify the amount of relevant information contained in each DCT frequency coefficient. GLD is the process of normalising the values of each of the coefficients within a DCT square, so that the minimum is 0 and the maximum is 1. The approach taken for the genetic algorithm is to obtain an optimised minimum Peak Signal to Noise Ratio (PSNR) at a target size, hence, the longer the algorithm performs its task, the higher the image quality will be at the given size. PSNR is a standard measurement used as a quality control, (Ichigaya & Sci., 2006). It measures the maximum intensity of signal (valuable data) against the noise (artefacts and other aberrations); a high number indicates a better quality. Their data concludes that PSNR is superior for a standardised set of images to both JPEG and JPEG-2000, achieving higher PSNR for the same amount of bytes per pixel (BPP).

Shuwang & Tao (2009) take a different approach to the compression optimisation problem, instead of modifying the mask directly and using the fitness results to change the value of each zonal mask value, the approach taken decides on a threshold for each frequency value below which the zonal mask will disregard the coefficient of the DCT. The fitness function used doesn't rely on PSNR, but instead on the inverse of the Generalised Cross Validation (GCV) function. GCV, under standard conditions, tends to be asymptotic at its optimum point, consequently, after several iterations the result most fitting will be achieved by increasing the amount of coefficients disregarded. Shuwang & Tao (2009) conclude that despite having a clear reduction in size and a small reduction in quality, the final results of the compression have only been evaluated subjectively (by human observation) and do not provide any more data regarding GCV genetic algorithm DCT compression results.

1.4.9 Genetic algorithms and fractal encoding

Genetic algorithms have also been successfully applied to fractal image compression; Mitra & Murthy (1998) take a complex approach to the generation of the genetic algorithm. The main computing intensive processes in fractal compression are searching the domain blocks and transforming the domain block to match the range blocks (Au & Liou, 1997), it is this processes that are optimized by the genetic algorithm from Mtira & Morthy (1998). The fractal compression algorithm used in their experiment contains 1282 domain blocks, and 8 potential transforms to be used, and they optimise the search algorithm for each of the range blocks. The genetic algorithm used differs from standard ones by producing multiple children from each parent, and these mutate and are convined in different ways and to different extents. This makes the process of the search require less iterations, which reduces the computational cost. The fitness function used is a standard Mean Square Error (MSE), comparing differences between the range and domain blocks obtained from the searches, which selects the future parents at the end of the iteration. The process obtains the same results as standard fractal compression (under 1% deviation on compression ratio and PSNR) and performs only 5% of the searches, producing a much more efficient algorithm. Vahdati & Khodadadi (2010) take different approach to the solving of the problem. Vahdati & Khodadadi (2010) combine a genetic algorithm with particle swarm optimization (PSO), PSO’s are similar to genetic algorithms in that they share a random starting population, called particles, instead of chromosomes. The main difference, is that unlike the chromosomes, the particles have a velocity, which means that they change values within iterations independently of mutations and crossovers. The particles also have memory, keeping the velocity over iterations, unlike chromosomes on genetic algorithms (GA), the mutations of which have no correlation to previous generations. Vahdati & Khodadadi (2010) first use the same number of iterations for PSO as they do for GA, first they optimise the search algorithm by performing the PSO, looking for random local optimum domain matches for the ranges, by having the particles on the swarm travel around the domain, trying to find close resemblances to a range image with each iteration. This are stored as “time” since departure of the particle, so the location of the matches is stored more efficiently than using coordinates. After the ideal optimum maches have been found, the results of the PSO are fed to the GA, as the starting population of chromosomes. The GA is performed in the standard way, using MSE as a fitness function, and stopping after a given number of iterations: the same used for the PSO. The results obtained provide a significant improvement in efficiency compared to a full search, standard, fractal compression algorithm. The improvement for the standard Lena, Peppers, and Baboon, providing an increase in speed by a factor of 71.25, 78.62 and 73 respectively, at comparative PSRN and around 10% lower bits per pixel.

Chapter 2
Image Compression using K-means algorithm

In this chapter, we will implement the K-means algorithm and use it for image compression. We will first start on an example 2D dataset that will help us gain an intuition of how the K-means algorithm works. After that, we will use the K-means algorithm for image compression by reducing the number of colors that occur in an image to only those that are most common in that image.

2.1 Implementing K-means

The K-means algorithm is a method to automatically cluster similar data examples together. Concretely, we are given $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$ and want to group the data into a few cohesive clusters.

The intuition behind K-means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.
The K-means algorithm is as follows:

```
% Initialize centroids
centroids = kMeansInitCentroids(X, K);
for iter = 1:iterations
% Cluster assignment step: Assign each data point to the closest centroid idx(i)
%corresponds to c^(i), the index of the centroid assigned to example i
idx = findClosestCentroids(X, centroids);
% Move centroid step: Compute means based on centroid assignments
centroids = computeMeans(X, idx, K);
end
```

The inner-loop of the algorithm repeatedly carries out two steps:

(i) Assigning each training example x(i) to its closest centroid.

(ii) Recomputing the mean of each centroid using the points assigned to it.

The K-means algorithm will always converge to some final set of means for the centroids. Note that the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, in practice the K-means algorithm is usually run a few times with different random initializations. One way to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion). We will implement the two phases of the K-means algorithm separately in the next sections.

2.1.1 Finding closest centroids

In the cluster assignment phase of the K-means algorithm, the algorithm assigns every training example x(i) to its closest centroid, given the current positions of centroids. Specifically, for every example i we set

$$c^{(i)} := j \quad \text{that minimizes} \quad ||x^{(i)} - \mu_j||^2,$$

where $c^{(i)}$ is the index of the centroid that is closest to $x^{(i)}$, and μ_j is the position (value) of the j'th centroid. Note that $c^{(i)}$ corresponds to idx(i) in the implemented code.

2.1.2 Computing centroid means

Given assignments of every point to a centroid, the second phase of the algorithm recomputes, for each centroid, the mean of the points that were assigned to it. Specifically, for every centroid k we set

$$\mu_k = 1/|C_k|\sum x^{(i)}$$

where C_k is the set of examples that are assigned to centroid k. Concretely, if two examples say x(3) and x(5) are assigned to centroid k = 2, then we should update

$$\mu_2 = 1/2(x(3) + x(5))$$

We can implement this using a loop over the centroids. We can also use a loop over the examples, but if we can use a vectorized implementation that does not use such a loop, our code may run faster.

2.2 K-means algorithm on example dataset

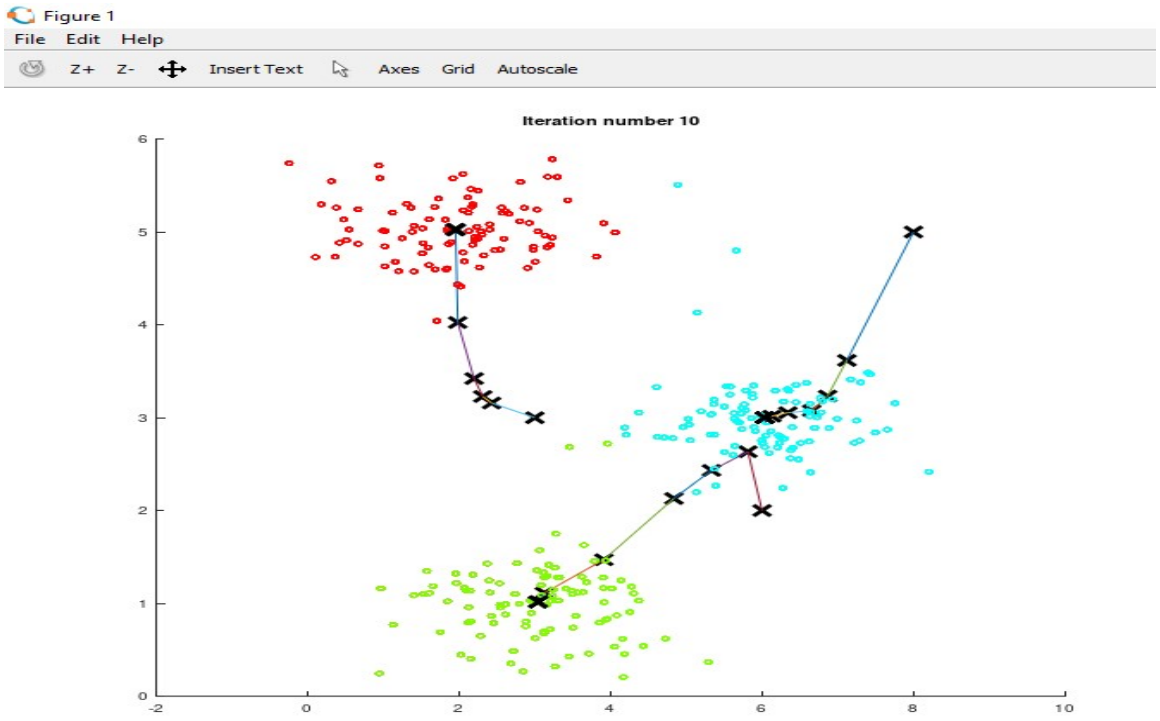


Fig 2.1: K-means algorithm on example dataset

After having completed the two functions (findClosestCentroids and computeCentroids), the next step will run the K-means algorithm on a toy 2D dataset to help us understand how K-means works. Our functions are called from inside the runKmeans.m script. Notice that the code calls the two functions we implemented in a loop. When we run the next step, the K-means code will produce a visualization that steps you through the progress of the algorithm at each iteration.
We press enter multiple times to see how each step of the K-means algorithm
changes the centroids and cluster assignments. At the end, our figure look as the one displayed in Figure 2.1.

2.3 Random initialisation

The initial assignments of centroids for the example dataset were designed so that we will see the same figure as in Figure 2.1. In practice, a good strategy for initializing the centroids is to select random examples from the training set.

In this part of the exercise, we should complete the function kMeansInitCentroids.m with the following code:

```
% Initialize the centroids to be random examples
% Randomly reorder the indices of examples
randidx = randperm(size(X, 1));
% Take the first K examples as centroids
centroids = X(randidx(1:K), :);
```

The code above first randomly permutes the indices of the examples (using randperm). Then, it selects the first K examples based on the random permutation of the indices. This allows the examples to be selected at random without the risk of selecting the same example twice

2.4 Image compression with K-means

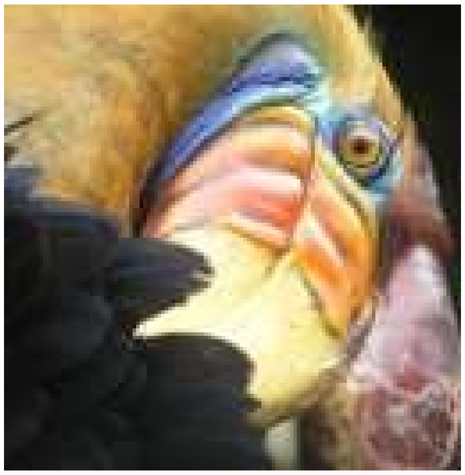


Fig 2.2: The original 128 X 128 image

In this part, we will apply K-means to image compression. In a straightforward 24-bit colour representation of an image, each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255) that specify the red, green and blue intensity values. This encoding is often referred to as the RGB encoding. Our image contains thousands of colours, and in this part, we will reduce the number of colours to 16 colours. By making this reduction, it is possible to represent (compress) the photo in an efficient way. Specifically, we only need to store the RGB values of the 16 selected colours, and for each pixel in the image we now need to only store the index of the colour at that location (where only 4 bits are necessary to represent 16 possibilities). We will use the K-means algorithm to select the 16 colours that will be used to represent the compressed image. Concretely, we will treat every pixel in the original image as a data example and use the K-means algorithm to find the 16 colours that best group (cluster) the pixels in the 3-dimensional RGB space. Once we have computed the cluster centroids on the image, we will then use the 16 colours to replace the pixels in the original image.

2.4.1 K-means on pixels

In Octave/MATLAB, images can be read in as follows:

```
% Load 128x128 color image (bird small.png)
A = imread('bird small.png');
%We will need to have installed the image package to used imread. If we do not have
% the image package installed, we should instead change the following line to
load('bird small.mat');
```

This creates a three-dimensional matrix A whose first two indices identify a pixel position and whose last index represents red, green, or blue. For example, A(50, 33, 3) gives the blue intensity of the pixel at row 50 and column 33. The code first loads the image, and then reshapes it to create an m x 3 matrix of pixel colors (where m = 16384 = 128 x 128), and calls your K-means function on it.

After finding the top K = 16 colours to represent the image, we can now assign each pixel position to its closest centroid using the findClosestCentroids function. This allows us to represent the original image using the centroid assignments of each pixel. Notice that we have significantly reduced the number of bits that are required to describe the image. The original image required 24 bits for each one of the 128 x 128 pixel locations, resulting in total size of 128 x 128 x 24 = 393,216 bits. The new representation requires some overhead storage in form of a dictionary of 16 colors, each of which require 24 bits, but the image itself then only requires 4 bits per pixel location. The final number of bits used is therefore 16 x 24 + 128 x 128 x 4 = 65,920 bits, which corresponds to compressing the original image by about a factor of 6.

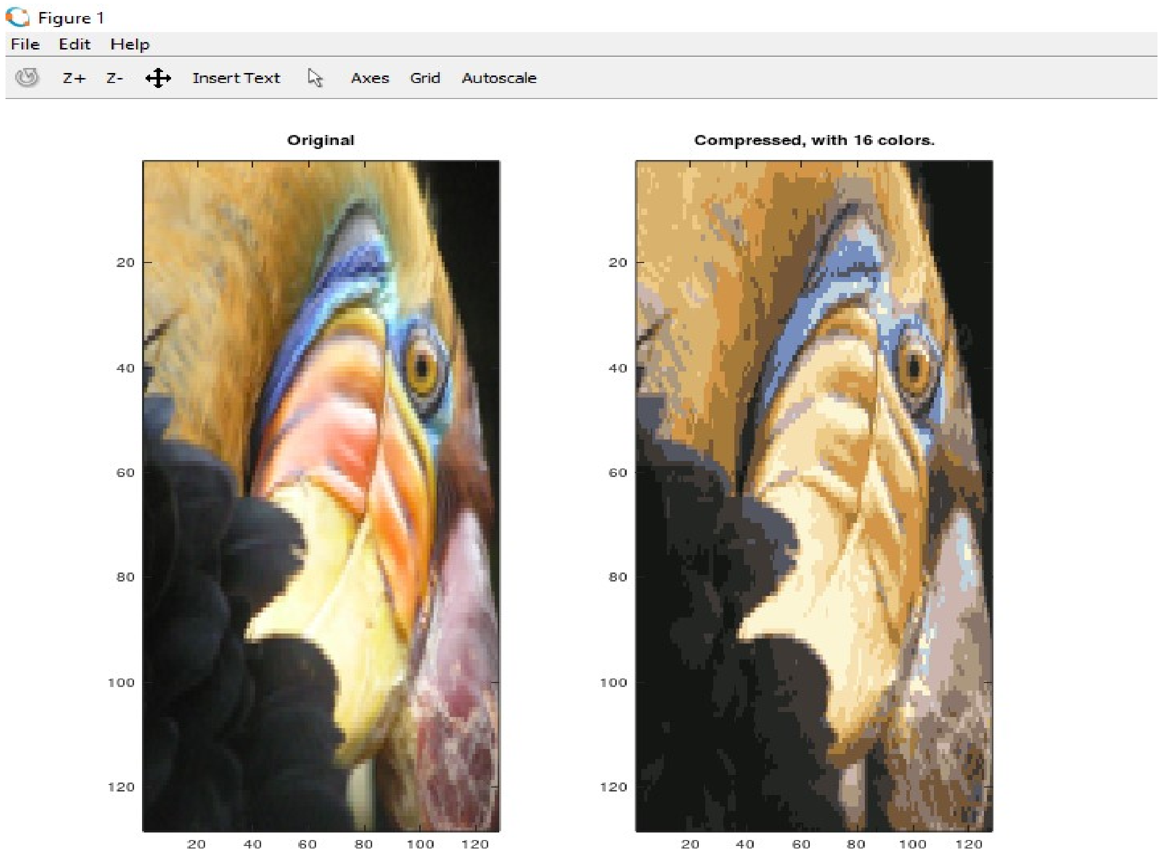


Fig 2.3: Original and reconstructed image (when using K-means to compress the image).

Finally, we can view the effects of the compression by reconstructing the image based only on the centroid assignments. Specifically, we can replace each pixel location with the mean of the centroid assigned to it. Figure 2.3 shows the reconstruction we obtained. Even though the resulting image retains most of the characteristics of the original, we also see some compression artifacts.

Chapter 3
Image Compression using Principal Component Analysis algorithm

In this chapter, we will use principal component analysis (PCA) to perform dimensionality reduction. We will first experiment with an example 2D dataset to get intuition on how PCA works, and then use it on a bigger dataset of 5000 face image dataset.

3.1 Example dataset

To understand how PCA works, we will first start with a 2D dataset which has one direction of large variation and one of smaller variation. The script will plot the training data (Figure 3.1). In this part, we will visualize what happens when we use PCA to reduce the data from 2D to 1D. In practice, we might want to reduce data from 256 to 50 dimensions, but using lower dimensional data in this example allows us to visualize the algorithms better.

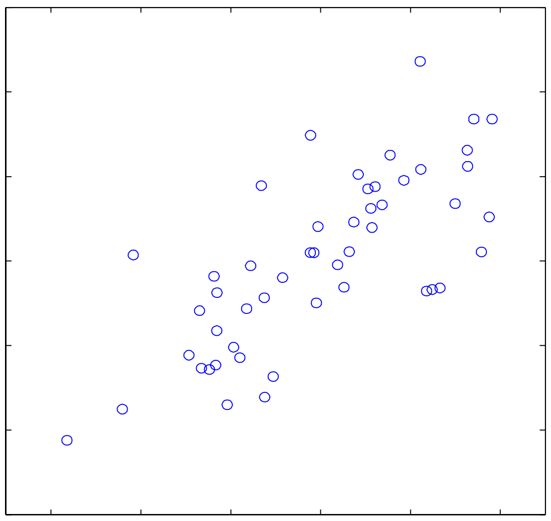


Fig 3.1: Example Dataset 1

3.2 Implementing PCA

In this part, we will implement PCA. PCA consists of two computational steps. First, we compute the covariance matrix of the data. Then, we use Octave/MATLAB's SVD function to compute the eigenvectors U1,U2,…,Un. These will correspond to the principal components of variation in the data.

Before using PCA, it is important to first normalize the data by subtracting the mean value of each feature from the dataset, and scaling each dimension so that they are in the same range. After normalizing the data, we can run PCA to compute the principal components. First, we should compute the covariance matrix of the data, which is given by:

$$\sum = 1/m(X'X)$$

where X is the data matrix with examples in rows, and m is the number of examples. Note that $\sum$ is a n x n matrix and not the summation operator. After computing the covariance matrix, we can run SVD on it to compute the principal components. In Octave/MATLAB, we can run SVD with the following command:

```
[U, S, V] = svd(Sigma)
```

where U will contain the principal components and S will contain a diagonal matrix.

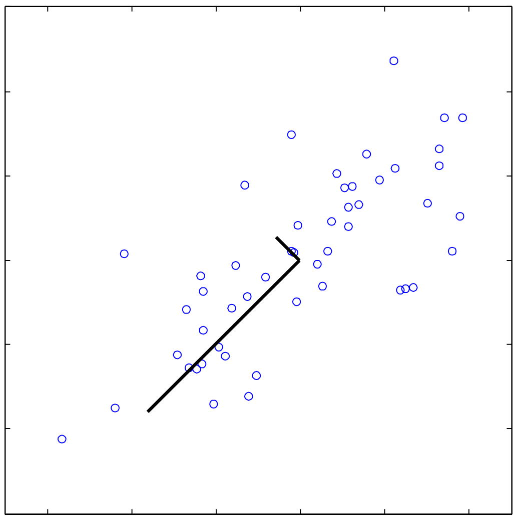


Fig 3.2: Computed eigenvectors of the dataset

Once we have completed this, the script will run PCA on the example dataset and plot the corresponding principal components found (Figure 3.2). The script will also output the top principal component (eigen vector) found, and we see an output of about [-0.707, -0.707]. (It is possible that Octave/MATLAB may instead output the negative of this, since U1 and -U1 are equally valid choices for the first principal component.)

3.3 Dimensionality Reduction with PCA

After computing the principal components, we can use them to reduce the feature dimension of our dataset by projecting each example onto a lower dimensional space, x(i) → z(i) (e.g., projecting the data from 2D to 1D). In this, we will use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space. In practice, if we were using a learning algorithm such as linear regression or perhaps neural networks, we could now use the projected data instead of the original data. By using the projected data, we can train our model faster as there are less dimensions in the input.

3.3.1 Projecting the data onto the principal components

Specifically, we are given a dataset X, the principal components U, and the desired number of dimensions to reduce to K. We should project each example in X onto the top K components in U. Note that the top K components in U are given by the first K columns of U, that is

U reduce = U(:, 1:K)

Once we have completed the code. The code will project the first example onto the first dimension and we see a value of about 1.481 (or possibly -1.481, if we get -U1 instead of U1).

3.3.2 Reconstructing an approximation of the data

After projecting the data onto the lower dimensional space, we can approximately recover the data by projecting them back onto the original high dimensional space. Our task is to project each example in Z back onto the original space and return the recovered approximation in X_recovered.

Once we have completed the code. We will be able to recover an approximation of the first example and we see a value of about [-1.047 -1.047].

3.3.3 Visualising the projection

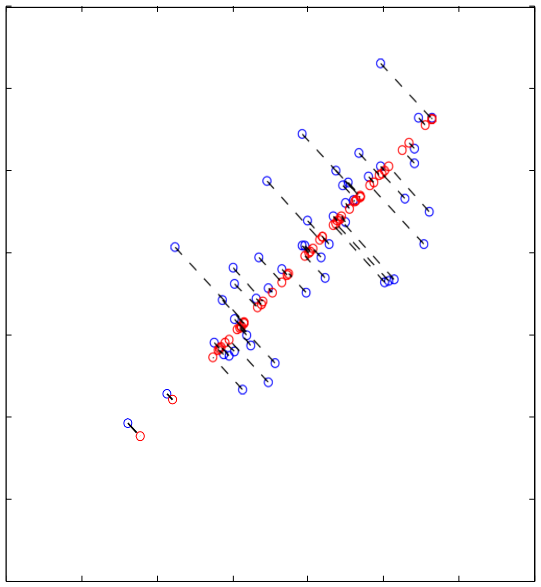


Fig 3.3: The normalized and projected data after PCA.

After completing this, we will now perform both the projection and approximate reconstruction to show how the projection affects the data. In Figure 3.3, the original data points are indicated with the blue circles, while the projected data points are indicated with the red circles. The projection effectively only retains the information in the direction given by U1.

3.4 Face Image Dataset

In this part, we will run PCA on face images to see how it can be used in practice for dimension reduction. The dataset ex7faces.mat contains a dataset X of face images, each 32 x 32 in grayscale. Each row of X corresponds to one face image (a row vector of length 1024). The next step will be to load and visualize the first 100 of these face images (Figure 3.4).

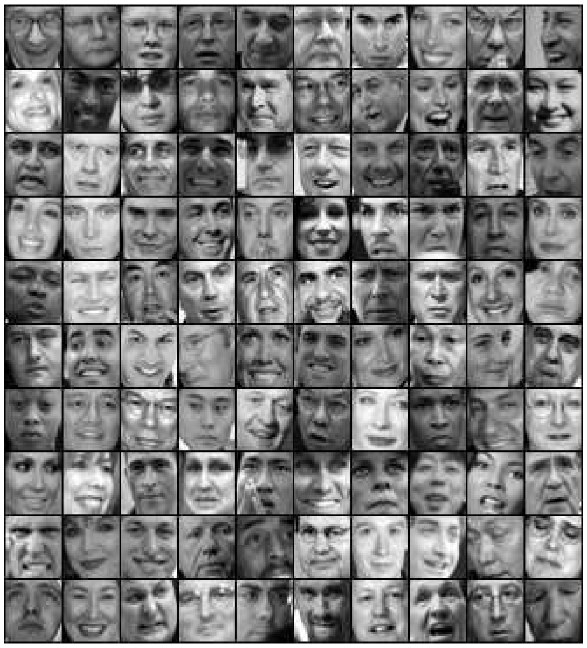


Fig 3.4: Faces dataset

3.4.1 PCA on faces

To run PCA on the face dataset, we first normalize the dataset by subtracting the mean of each feature from the data matrix X. After running PCA, we will obtain the principal components of the dataset. Notice that each principal component in U (each row) is a vector of length n (where for the face dataset, n = 1024). It turns out that we can visualize these principal components by reshaping each of them into a 32 x 32 matrix that corresponds to the pixels in the original dataset. The script ex7 p displays the first 36 principal components that describe the largest variations (Figure 3.5). If needed, we can also change the code to display more principal components to see how they capture more and more details.

3.4.2 Dimensionality Reduction

Now that we have computed the principal components for the face dataset, we can use it to reduce the dimension of the face dataset. This allows us to use our learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions. This can help speed up our learning algorithm.

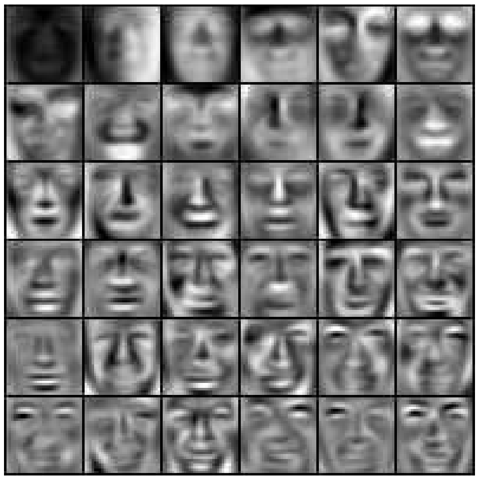


Fig 3.5: Principal components on the face dataset

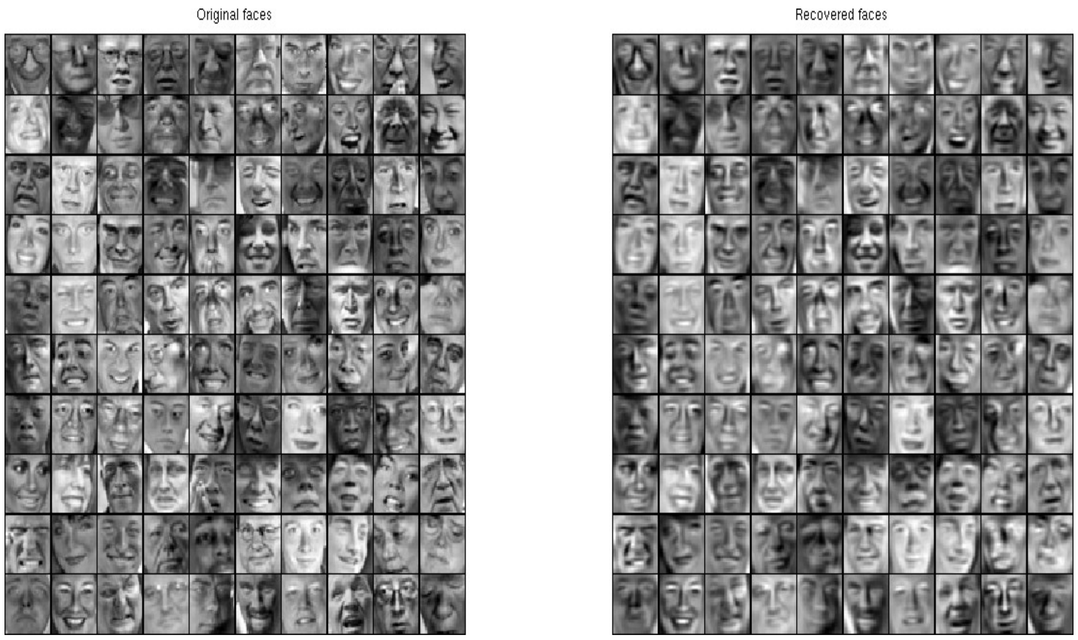


Fig 3.6: Original images of faces and ones reconstructed from only the top 100 principal components.

The next part will project the face dataset onto only the first 100 principal components. Concretely, each face image is now described by a vector z(i) € R100. To understand what is lost in the dimension reduction, we can recover the data using only the projected dataset. In this, an approximate recovery of data is performed and the original

and projected face images are displayed side by side (Figure 3.6). From the reconstruction, you can observe that the general structure and appearance of the face are kept while the fine details are lost. This is a remarkable reduction (more than 10 x) in
the dataset size that can help speed up our learning algorithm significantly. For example, if we were training a neural network to perform person recognition (given a face image, predict the identity of the person), we can use the dimension reduced input of only a 100 dimensions instead of the original pixels.

Chapter 4
K-means algorithm vs PCA algorithm

In this chapter we will try to provide advantages and disadvantages of using K-means algorithm and PCA algorithm and also try to provide the scenarios in which we should use either of the algorithms

4.1 Advantages of K-means algorithm

- It is relatively simple and easy to implement than other algorithms.
- It can scale to large datasets well and easily adapts to new examples.
- It always guarantees convergence
- It can easily adapt to new examples
- It generalizes to clusters of different shapes and sizes such as elliptical clusters

4.2 Disadvantages of K-means algorithm

- It is dependent on initial values of centroid. Here we take any random value
- It is difficult to cluster outliers while using K-means algorithm
- We have to choose K (number of clusters) manually.
- Algorithm time complexity increases with increase in number of dimensions

4.3 Advantages of PCA algorithm

- PCA removes correlated features
- PCA reduces overfitting
- PCA improves visualization and improves algorithm performance

4.4 Disadvantages of PCA algorithm

- In PCA algorithm, independent variable becomes less interpretable
- We must do data normalization before using PCA algorithm
- There is some amount of information loss while reconstruction of images

4.5 K-means vs PCA

K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster. The approach k-means follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster. Since clustering algorithms including k-means use distance-based measurements to determine the similarity between data points, it's recommended to standardize the data to have a mean of zero and a standard deviation of one since almost always the features in any dataset would have different units of measurements such as age vs income. Given k-means iterative nature and the random initialization of centroids at the start of the algorithm, different initializations may lead to different clusters since k-means algorithm may stuck in a local optimum and may not converge to global optimum. Therefore, it's recommended to run the algorithm using different initializations of centroids and pick the results of the run that that yielded the lower sum of squared distance.

K-means algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc. K-means algorithm is good in capturing structure of the data if clusters have a spherical-like shape. It always tries to construct a nice spherical shape around the centroid. That means, the minute clusters have a complicated geometric shapes, k-means does a poor job in clustering the data. K-means algorithm doesn't let data points that are far-away from each other share the same cluster even though they obviously belong to the same cluster.

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analysing data much easier and faster for machine learning algorithms without extraneous variables to process. So to sum up, the idea of PCA is simple, reduce the number of variables of a data set, while preserving as much information as possible.

Chapter 5
Conclusions

5.1 Conclusion

Based on the results and discussions presented in this report, K-means and PCA algorithm can be used to compress images. Compressing an image using the K-means and PCA algorithm is a lossy compression. When K is larger, the compression ratio becomes larger and when K is lower, compression ratio become lower. When the image pixels are large, we should set a larger number of iterations for better compression. Random initialization of centroids is necessary to get optimal results. Also, we should try to choose that value of k which meets our compression requirements.

K-means clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of k-means is to group data points into distinct non-overlapping subgroups. It does outstanding job when the clusters have a kind of spherical shapes. However, it suffers as the geometric shapes of clusters deviates from spherical shapes. Moreover, it also doesn't learn the number of clusters from the data and requires it to be pre-defined. To be a good practitioner, it's good to know the assumptions behind algorithms or methods so that you would have a pretty good idea about the strength and weakness of each method. This helps us decide when to use each method and under what circumstances.

K-means algorithm is a type of unsupervised learning that can be used to probe data structures. Clustering is the process of dividing data into multiple clusters, each of which consists of one or more similar data. The clustering algorithm requires the greatest similarity between the data of the same cluster, while the data of different clusters has the smallest similarity between the clusters. Unlike the classification learning, the clustering algorithm is an unsupervised learning method. The clustering algorithm does not need to label the categories of the samples, but divides the data set into several clusters according to the similarity of the samples. Therefore, the clusters of data are not predefined, but are defined according to the similarity of the characteristics of the samples. Therefore, the input cluster data does not need to be pre-marked.

Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation which converts a set of correlated variables to a set of uncorrelated variables. PCA is a most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

One benefit of PCA is that we can examine the variances associated with the principal components. One can conclude that most interesting dynamics occur only in the first k dimensions. Both the strength and weakness of PCA is that it is a non-parametric analysis. There are no parameters to tweak and no coefficients to adjust based on user experience the answer are unique and independent of the user. This same strength can also be viewed as a weakness. If one knows a priori some features of the dynamics of a system, then it makes sense to incorporate these assumptions into a parametric algorithm or an algorithm with selected parameters. Thus, the appropriate parametric algorithm is to first convert the data to the appropriately centered polar coordinates and then compute PCA. The quantity of principal components used in compression influences the recovery of the original image from the compacted image. This tool allows significant savings of storage space, which can be critical in clinical applications and in processing large volumes of data. As a secondary property, these components also have the potential of reflecting the complexity of the image, enabling their correlation with the texture of the image.

5.2 Limitation if any

The main drawback of using image compression techniques like K-means and PCA algorithm, besides its computational requirements, is their complexity. The fact that they require training sets, and sometimes, supervision, may decrease widespread adoption, due to the difficulty generalizing the application of these algorithms that this drawback carries, but the results obtained are positive, and so is the outlook for increased compression ratios at reasonable computational costs. It is dependent on initial values of centroid hence we take any random value. It is difficult to cluster outliers while using K-means algorithm. We have to choose K (number of clusters) manually. Algorithm time complexity increases with increase in number of dimensions.

Another difficulty in using the K-means algorithm is the choice of K. In theory, there is no standard selection formula. The K value is manually selected by according to different problems. Taking different K values, the value of the cost function J is different. As the value of K increases, the J value decreases accordingly. When the number K of clusters is equal to the number of data points, J is reduced to zero. In practice, the K value can be selected according to the curves of K and J. The K v/s J curve is similar to the elbow of a person. When the K is equal to 1, the value of J is large. When K drops to the inflection point, the J rapidly drops and reaches the elbow position. Thereafter, as the K value increases, the J value decreases very slowly. This inflection point is the optimal choice of the number of clusters K.

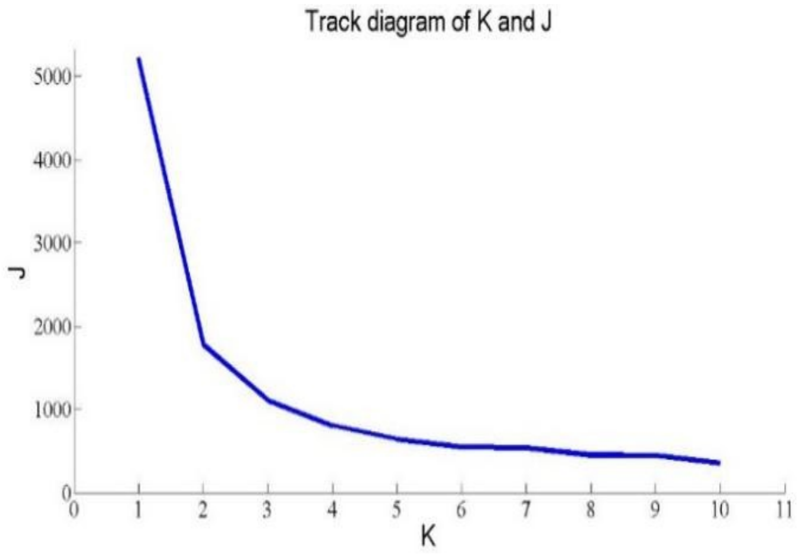


Fig 5.1: Track diagram of K and J

The new principal components are not interpretable, which may be a deal-breaker in some settings. In addition, you must still manually set or tune a threshold for cumulative explained variance. In PCA algorithm, independent variable become less interpretable. We must do data normalization before using PCA algorithm. There is some amount of information loss while reconstruction of images**.** We must choose the smallest value of k for which 99% of variance is retained so as to get a significant number of principal components. The image then reconstructed will have less information loss.

5.3 Future Scope of work

There are still a lot of research opportunities regarding the combination of the relatively novel machine learning techniques, and traditional learning algorithms. Compression algorithms that had computationally privative costs, such as K-means and PCA algorithms, are becoming more feasible by combining them with other techniques. As technology advances, neural network experimentation may become more accessible. At the moment, genetic algorithms provide more immediate results.

The rapid growth of digital technology has paved the way for many image processing applications and motivated the need for better compression algorithms. Image compression is the technique where the size of an image is minimized by keeping the quality of the image to an acceptable level. The main objective of image compression is to decrease the redundancy of the image thereby increasing the capacity of storage and efficient transmission. So we can take advantage of various image compression algorithms to compress image without much information loss and reducing its space and time complexity.

References

(i) for books

[1] Gonzalez, R. C., & Woods, R. E. (2007). *Digital Image Processing* (3rd edition ed.) London: Prentice Hall.

[2] Haykin, S. (2004). *A comprehensive foundation Neural Networks* (2nd edition ed.). Hamilton: Prentice Hall.

(ii) for journals

[3] Chang, D. X., Zhang, X. D., & Zheng, C. W. (2009). A genetic algorithm with gene rearrangement for k-means clustering. Pattern Recognition, 42(7), 1210-1222.

[4] Kattan, A. (2010). Universal Intelligent Data Compression Systems: A Review Computer Science and Electronic Engineering Conference (pp. 1-10). Colchester: IEEE.

[5] Malwinder Kaur, Navdeep Kaur, “A Literature Survey on Lossless Compression”, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 3, March 2015.

[6] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu A. Y.(2002). An efficient k-means clustering algorithm: analysis and implementation. IEEE. Transactions on Pattern Analysis & Machine Intelligence, 24(7), 881-892.

(iii) for conference proceedings

[7] Ding, C., & He, X. (2004). K-means clustering via principal component analysis. International Conference on Machine Learning