SHRINKING IMAGES: THE HIDDEN COST OF COMPRESSION

Zeedan Shaikh

2024-09-21

Contents

Introduction

In today's world, handling and processing vast amounts of data efficiently is crucial. One area where this is particularly important is in image compression and feature extraction, where techniques such as **Principal Component Analysis (PCA)** and **Vector Quantization (VQ)** play a pivotal role. This project aims to explore these powerful techniques and their applications in image processing.

The project focuses on leveraging **PCA**, a technique widely used for **dimensionality reduction**, to reduce the complexity of high-dimensional image data while preserving its essential features. Following this, **Vector Quantization** is applied to compress the image by mapping pixel values to a smaller set of representative colors (codewords), resulting in a compressed yet visually coherent output.

Through this project, we embark on a journey that combines theoretical understanding with practical application, creating a vivid transformation of digital images while maintaining data efficiency. The outcomes of the project not only showcase the efficiency of PCA and VQ but also open up discussions on the trade-offs between data compression and quality preservation, making it both educational and visually captivating.

Objective

- We will be using two images :
  1. A colour image of tiles pattern
  2. black and white image of **Prasanta Chandra Mahalanobis**
- We will try in to compress both the images using PCA and VQ

Getting the required packages:

```
library(pacman,quietly=T)
pacman::p_load(jpeg,MASS)
```

Methodology Image 1

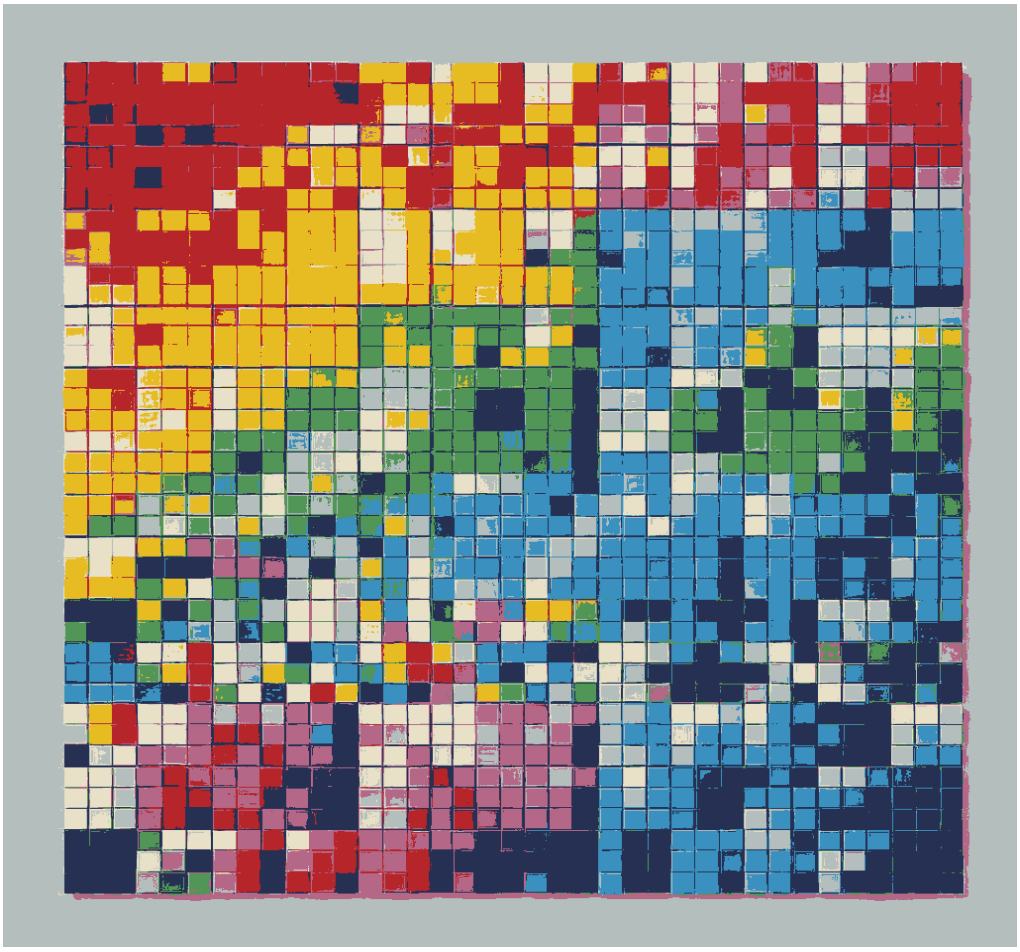
Data preparation

- First we convert the image into a three-dimensional array where Dim 1 and 2 denotes height and width of the image Dim 3 denotes RGB (Red, Green, Blue) Values
- Then the we converted the array in 2 dimensions where Dim 1 denotes Height*Width Dim 2 denotes RGB (Red, Green, Blue) Values

```
image=readJPEG("C:\\Users\\zeeda\\Downloads\\test image.jpeg")
image_dimensions=dim(image)
pixels=matrix(as.numeric(image),nrow=image_dimensions[1]*image_dimensions[2],ncol=3)
```

Applying Vector-Quantization

```
num_colors <- 8
set.seed(42)
kmeans_result <- kmeans(pixels, centers = num_colors)
kmean_clusters=kmeans_result$cluster
codebook <- kmeans_result$centers
quantized_pixels <- codebook[kmeans_result$cluster, ]
quantized_image <- array(quantized_pixels, dim = image_dimensions)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE)
rasterImage(quantized_image, 1, 1, 2,2)
```



Compression of the image is calculated by-

Original Size = Height × Width × 3 (RGB)

Quantized Size= Height × Width × log2(no of clusters)

Compression Ratio= Quantized Size / Original Size

- A compression ratio greater than 1 indicates that the image has been compressed.

Here the compression ratio is

```
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size=image_dimensions[1]*image_dimensions[2]*log(num_colors,2)
Compression_Ratio= Quantized_Size / Original_Size
cat(Compression_Ratio)
```

1

```
num_colors2 <- 4
set.seed(42)
kmeans_result2 <- kmeans(pixels, centers = num_colors2)
codebook2 <- kmeans_result2$centers
quantized_pixels2 <- codebook2[kmeans_result2$cluster, ]
quantized_image2 <- array(quantized_pixels2, dim = image_dimensions)
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size2=image_dimensions[1]*image_dimensions[2]*log(num_colors2,2)
Compression_Ratio2= Quantized_Size2 / Original_Size
Compression_Ratio2
```

Repeating the process for no clusters 4 & 2

```
[1] 0.6666667
```

```
num_colors3 <- 2
set.seed(42)
kmeans_result3 <- kmeans(pixels, centers = num_colors3)
codebook3 <- kmeans_result3$centers
quantized_pixels3 <- codebook3[kmeans_result3$cluster, ]
quantized_image3 <- array(quantized_pixels3, dim = image_dimensions)
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size3=image_dimensions[1]*image_dimensions[2]*log(num_colors3,2)
Compression_Ratio3= Quantized_Size3 / Original_Size
Compression_Ratio3
```

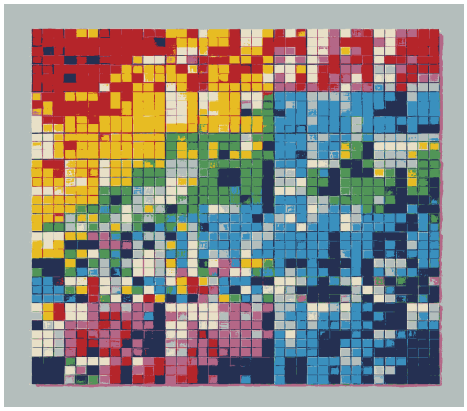
```
[1] 0.3333333
```

- Ploting all the images side by side

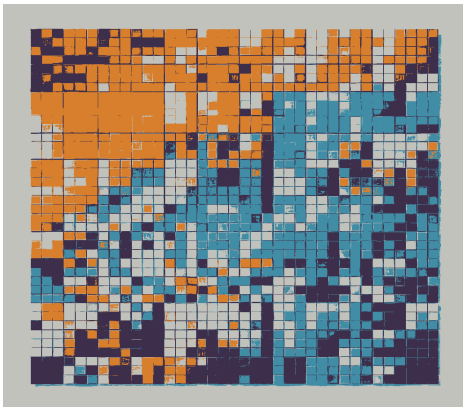
```
par(mfrow=c(2,2))
plot(1:2, type='n', xlab='', ylab='', axes=FALSE,main = "Original Image")
rasterImage(image, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 8 ")
rasterImage(quantized_image, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 4 ")
rasterImage(quantized_image2, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 2 ")
rasterImage(quantized_image3, 1, 1, 2, 2)
```

Original Image

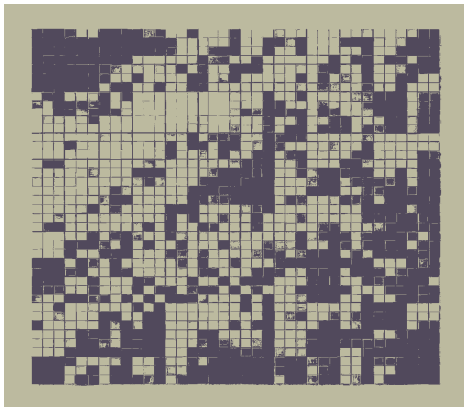
No. of Clusters 8



No. of Clusters 4



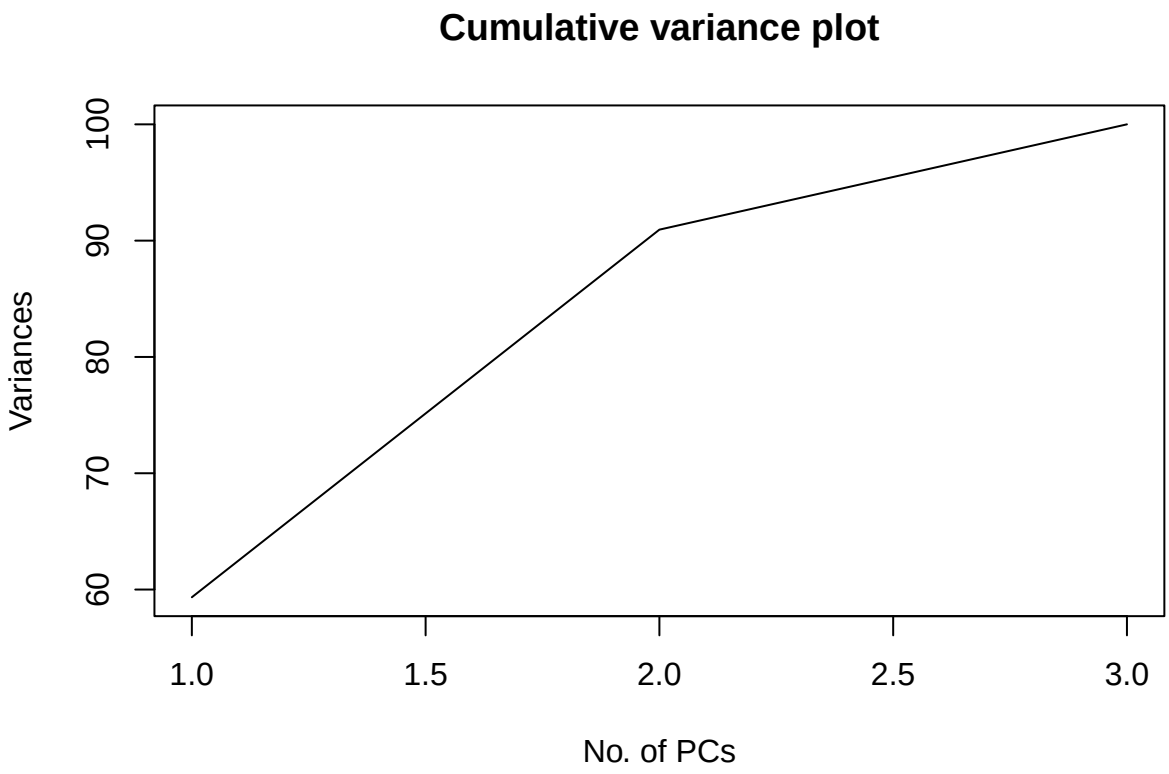
No. of Clusters 2



Applying Principal Component Analysis

- We will reduce the no of columns of the data by using principle components
- We will choose no. of components using Cumulative variance plot

```
pixels_scaled=scale(pixels)
pca_result=prcomp(pixels_scaled,center = T, scale.=T)
a=(cumsum(pca_result$sdev^2)/sum(pca_result$sdev^2))*100
plot(a,xlab="No. of PCs",ylab="Variances",main="Cumulative variance plot",type = "l")
```



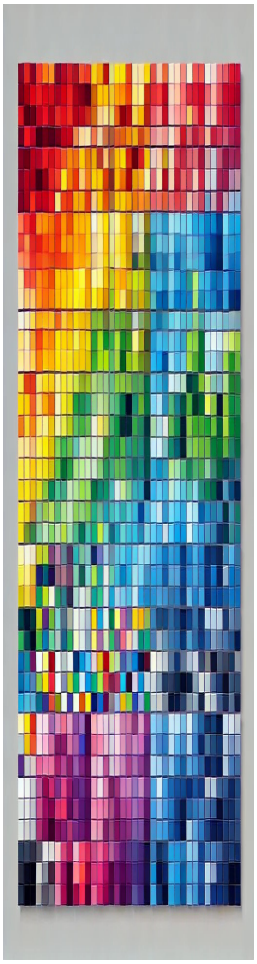
- 2 Pc's are capturing 90 percent variance *Transforming the data*

```
pixels= pca_result$x[,1:2]
```

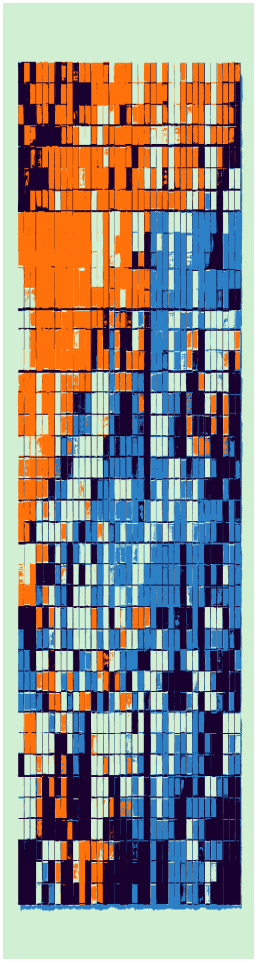
- First we need to fix the the no. of elements of the codebook i.e. to fix no. of clusters for k-means so we will repeat the process for different no. of clusters
- Then apply k-means and find the optimal cluster means
- Replace all the values in the cluster by it's cluster mean
- Reconstruct the image again

```
num_colors <- 4
set.seed(42)
kmeans_result <- kmeans(pixels, centers = num_colors)
kmean_clusters=kmeans_result$cluster
codebook <- kmeans_result$centers
quantized_pixels <- codebook[kmeans_result$cluster, ]
recons_pix= quantized_pixels %*% t(pca_result$rotation[,1:2])
recons_pix=scale(recons_pix,center = -attr(pixels_scaled, "scaled:center"),scale = 1/attr(pixels_scaled
recons_pix=(recons_pix-min(recons_pix))/(max(recons_pix)-min(recons_pix))
quantized_image <- array(recons_pix, dim = image_dimensions)
par(mfrow=c(1,3))
plot(1:2, type='n', xlab='', ylab='', axes=FALSE,main = "Original Image")
rasterImage(image, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE,main="VQ with PCA")
rasterImage(quantized_image, 1, 1, 2, 2 )
plot(1:2, type='n', xlab='', ylab='', axes=FALSE,main="VQ without PCA")
rasterImage(quantized_image2, 1, 1, 2,2)
```

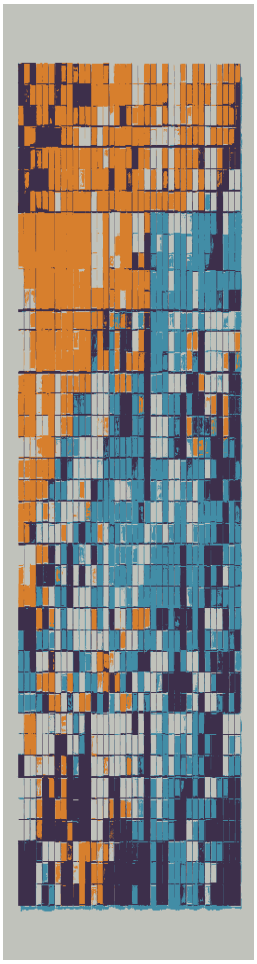
Original Image



VQ with PCA



VQ without PCA



Methodology Image 2

Data preparation

- First we convert the image into a three-dimensional array where Dim 1 and 2 denotes height and width of the image Dim 3 denotes RGB (Red, Green, Blue) Values
- Then the we converted the array in 2 dimensions where Dim 1 denotes Height*Width Dim 2 denotes RGB (Red, Green, Blue) Values

```
image=readJPEG("C:\\Users\\zeeda\\Downloads\\test image 2.jpeg")
image_dimensions=dim(image)
pixels=matrix(as.numeric(image),nrow=image_dimensions[1]*image_dimensions[2],ncol=3)
```

- **For a greyscale image the value of RGB components are equal.** No need to apply PCA .

Applying Vector-Quantization

```
num_colors <- 8
set.seed(42)
kmeans_result <- kmeans(pixels, centers = num_colors)
kmean_clusters=kmeans_result$cluster
codebook <- kmeans_result$centers
quantized_pixels <- codebook[kmeans_result$cluster, ]
quantized_image <- array(quantized_pixels, dim = image_dimensions)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE)
rasterImage(quantized_image, 1, 1, 2,2)
```

Here the compression ratio is

```
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size=image_dimensions[1]*image_dimensions[2]*log(num_colors,2)
Compression_Ratio= Quantized_Size / Original_Size
cat(Compression_Ratio)
```

```
1
```

```
num_colors2 <- 4
set.seed(42)
kmeans_result2 <- kmeans(pixels, centers = num_colors2)
codebook2 <- kmeans_result2$centers
quantized_pixels2 <- codebook2[kmeans_result2$cluster, ]
quantized_image2 <- array(quantized_pixels2, dim = image_dimensions)
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size2=image_dimensions[1]*image_dimensions[2]*log(num_colors2,2)
Compression_Ratio2= Quantized_Size2 / Original_Size
Compression_Ratio2
```

Repeating the process for no clusters 4 & 2

```
[1] 0.6666667
```

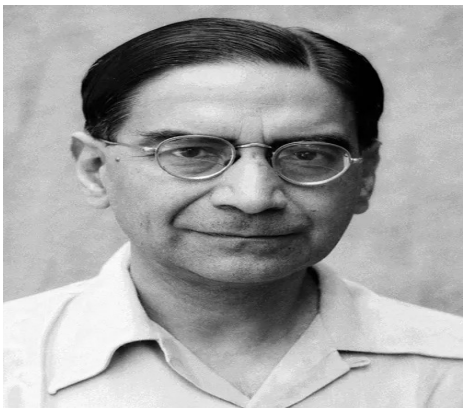
```
num_colors3 <- 2
set.seed(42)
kmeans_result3 <- kmeans(pixels, centers = num_colors3)
codebook3 <- kmeans_result3$centers
quantized_pixels3 <- codebook3[kmeans_result3$cluster, ]
quantized_image3 <- array(quantized_pixels3, dim = image_dimensions)
Original_Size=image_dimensions[1]*image_dimensions[2]*image_dimensions[3]
Quantized_Size3=image_dimensions[1]*image_dimensions[2]*log(num_colors3,2)
Compression_Ratio3= Quantized_Size3 / Original_Size
Compression_Ratio3
```

```
[1] 0.3333333
```

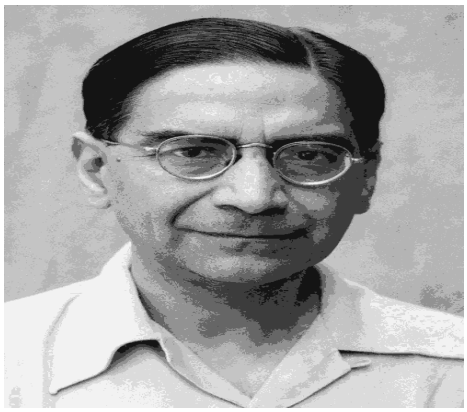
- Ploting all the images side by side

```
par(mfrow=c(2,2))
plot(1:2, type='n', xlab='', ylab='', axes=FALSE,main = "Original Image")
rasterImage(image, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 8 ")
rasterImage(quantized_image, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 4 ")
rasterImage(quantized_image2, 1, 1, 2, 2)
plot(1:2, type='n', xlab='', ylab='', axes=FALSE, main= " No. of Clusters 2 ")
rasterImage(quantized_image3, 1, 1, 2, 2)
```

Original Image



No. of Clusters 8



No. of Clusters 4

No. of Clusters 2

summary findings

Image No.	No of Clusters	Compression ratio
1	8	1
1	4	0.67
1	2	0.33
2	8	1
2	4	0.67
2	2	0.33

Conclusion

In this project, **"Shrinking Images: The Hidden Cost of Compression"**, we explored the balance between image compression and quality preservation using **Principal Component Analysis (PCA)** and **Vector Quantization (VQ)**. Through the application of PCA, we successfully reduced the dimensionality of the image's RGB color space, capturing the most significant features while discarding less important information. Vector Quantization further compressed the image by grouping similar pixel colors into clusters, reducing storage requirements.

Our analysis demonstrated that while compression techniques like PCA and VQ significantly reduce file sizes, they also introduce some level of information loss. This trade-off was evident in the form of minor visual distortions, such as color shifts and loss of fine details, as we reduced the number of components and clusters. However, the key insight is that by intelligently choosing the number of components and clusters, we can achieve a high degree of compression while retaining most of the image's visual integrity.

Discussion

The use of PCA in image compression is a powerful method for reducing redundancy in the data. By transforming correlated RGB color channels into uncorrelated principal components, we effectively capture the essence of the image with fewer variables. This reduction in dimensionality allows for more efficient storage and faster processing, especially in large datasets where full-resolution images would be too computationally expensive.

However, PCA introduces a few challenges: - **Loss of Color Fidelity**: When reducing the number of principal components, some subtle color details are lost, especially in areas where colors are closely related. - **Trade-off Between Compression and Quality**: There is an inherent trade-off between how much an image is compressed and how much quality is lost. This is particularly noticeable when too few principal components are retained.

Similarly, **Vector Quantization** allows for compression by reducing the number of unique colors (codewords) in the image. While it is effective, it can also introduce artifacts such as color banding or loss of smooth gradients, depending on the complexity of the image.

In future work, more advanced techniques like **Autoencoders** or **Wavelet-based Compression** could be explored to further enhance image compression while minimizing visual quality loss. Additionally, applying these techniques to various types of images (e.g., natural landscapes vs. synthetic images) could reveal more about the strengths and weaknesses of PCA and VQ in different contexts.

Overall, the project highlights the importance of balancing compression techniques with visual fidelity, particularly in applications where both data efficiency and quality are critical.