UE18EC346 : Advanced Digital Image Processing

Assignment 2

Title: Implementing the k-means clustering algorithm and principal component analysis (PCA) for image compression.

Guide

Dr. Shikha Tripathi

Professor, Dept. of ECE

PES University

Motivation and Introduction

The objective of image compression is to reduce the memory size to be as small as possible while maintaining the similarity with the original image. Our original image contains thousands of colours which are usually ignored by the human visual system and hence can be termed as irrelevant redundancy.

Image compression refer to reducing the dimensions, pixels, or color 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 algorithms take advantage of visual perception and the statistical properties of image data to provide superior results.

The k-means algorithm is a centroid based clustering technique. This technique clusters the dataset into k different clusters. We will utilize k-means clustering algorithm to reduce the number of colours so that it only needs to store certain numbers of RGB values only. Thus, will reduce the image size and making it more efficient in the storage.

Some image compression techniques involving extracting the most useful components of the image (PCA), which can be used for feature summarization or extraction and data analysis. PCA helps to reduce the number of "features" while preserving the variance, whereas clustering reduces the number of "data-points" by summarizing several points by their expectations/means (in the case of k-means). It is a technique for feature extraction — so it combines our input variables in a specific way, at which point we can drop the least important variables while still retaining the most valuable parts of all of the variables. PCA results in the development of new features that are independent of one another.

Thus, owing to the importance of these 2 algorithms in the image compression field, we implement and compare these 2 compression algorithms in an exemplary manner.

Theoretical details and Algorithm

(1)K-means clustering algorithm –

It is an unsupervised learning algorithm which is widely used for unlabelled data in the machine learning field. K-means clustering is the optimization technique to find the ‘k’ clusters or groups in the given set of data points. The data points are clustered together on the basis of some kind of similarity. Initially, it starts with the random

initialization of the ‘k’ clusters and then on the basis of some similarity (like Euclidean distance metric), it aims to minimize the distance from every data point to the cluster centre in each clusters. It is a centroid based clustering technique. There are mainly two iterative steps in the algorithm:

a) Assignment step- Each data point is assigned to a cluster whose centre is nearest to it.

b) Update step- New cluster centres (centroids) are calculated from the data points assigned to the new cluster by choosing the average value of these data points.

These iterative steps continue till the centroids seize to move further from their clusters. We get several clusters separated due to some difference while some data points are grouped together due to similarity. K-means clustering is a type of transform method of compression (lossy compression technique).

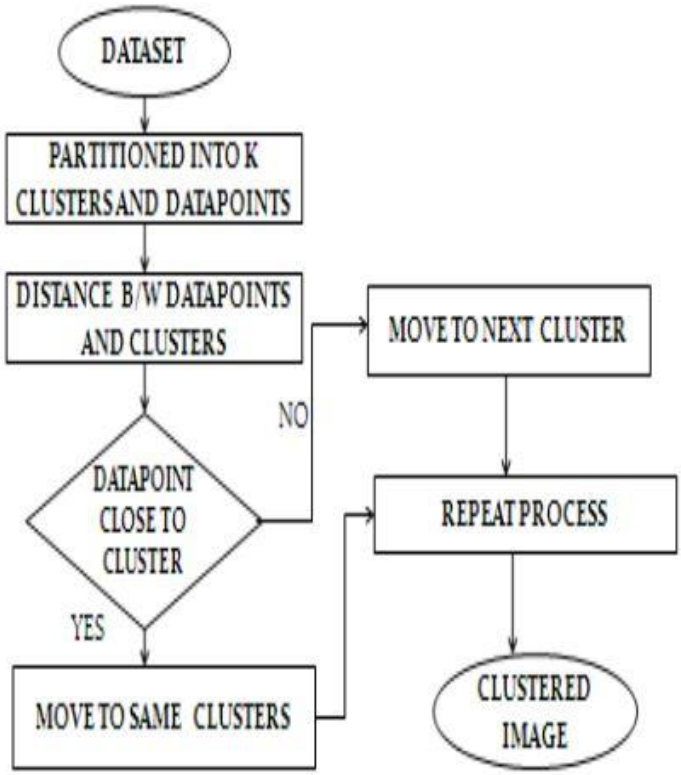
(1.1)K-means clustering to compress images –

In a coloured image, each pixel is of size 3 bytes (RGB), where each colour can have intensity values from 0 to 255. Following combinatorics, the total number of colours which can be represented is 256*256*256 (equal to 16,777,216). Practically, we can visualize only a few colours in an image very less than the above number. So the k-Means Clustering algorithm takes advantage of the visual perception of the human eye and uses few colours to represent the image.

(1.2)Approach – K-means clustering will group similar colours together into ‘k’ clusters (say k=64) of different colours (RGB values). Therefore, each cluster centroid is the representative of the colour vector in RGB colour space of its respective cluster. Now, these ‘k’ cluster centroids will replace all the colour vectors in their respective clusters. Thus, we need to only store the label for each pixel which tells the cluster to which this pixel belongs. Additionally, we keep the record of colour vectors of each cluster centre.

Execution time increases as the image dimensions increases or ‘K’ increases. So, initially you can start with a lesser value of ‘k’ in order to quickly get results. There is a trade-off between the execution time and the number of colours represented in reconstructed image. Higher ‘k’ will produce better quality of compressed image but will take longer to execute.

(1.3)The flowchart of the k-means algorithm is shown below –



(1.4)Additional details –

Variance of the image is given by –

$$Variance = \sum_{j=0}^{k} ||x_j - \bar{x}||^2$$

In order to compare the original and compressed image, a set of metrics are introduced to evaluate the compressed image such as -

1)Within Cluster Sum of Square (WCSS), which measures the sum of squared Euclidean distance of all the points within a cluster to its cluster centroid.

2) Between Cluster Sum of Square (BCSS), measures the sum of squared Euclidean distance between all centroids.

3) Explained Variance, measures how many percent that the compressed image can explain the variance of the original image. If each pixel is considered as an individual cluster, then WCSS is equal to 0. Hence, Explained Variance = 100%.

4) Image size, measured in kilobytes to evaluate the reduction/compression performance.

In k-means, the selection of an optimal number of clusters *k* is usually done subjectively through visualization. We use elbow method using the longest

perpendicular distance for objective selection. The perpendicular distance is calculated using the below formula –

$$\mathrm{d} = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

(2) Principal Component Analysis (PCA) –

PCA is a statistical method that uses orthogonal transformations to turn a potentially correlated set of data into a linearly uncorrelated set of data which contain principal components. The number of principal components will be less than or equal to the total number of variables in the original dataset. Furthermore, the principal components are sorted in a way so that the first component contains the largest possible variance in the data, and each succeeding component has the next highest variance.

Principal component analysis is used to extract and visualize the important information contained in a multivariate data table. The PCA synthesizes this information into just a few new variables called principal components. These new variables correspond to a linear combination of the original variables. The number of main components is less than or equal to the number of original variables.

The information contained in a data set corresponds to the variance or the total inertia it contains. The objective of the PCA is to identify the directions (i.e., main axes or main components) along which the variation in the data is maximum. In other words, PCA reduces the dimensions of multivariate data to two or three main components, which can be viewed graphically, losing as little information as possible.

(2.1) Approach - The application of Principal Component Analysis (PCA) for image compression involves calculating the covariance matrix of the input images and its spectral decomposition to extract their Eigen values (EIG) and corresponding Eigen vectors. When the size of the imaged scene and/or the number of input images increases, the calculation of the covariance matrix and its spectral decomposition become practically ineffective and imprecise due to approximate errors.

For a given data set S = {xi}, xi ∈ $\Re$, we consider an m dimensional projection subspace where < nm , the optimal linear projection is defined by the m eigenvectors computed from the covariance matrix of the data set corresponding to the first m largest eigenvalues.

PCA provides a simple and efficient method for image compression. For instance, an image block of size 8x8 can be regarded as a 64-dimensional vector. In encoding process, the inner product of a 64 dimensional vectors and eigenvector is called a

compression code for all image blocks. Moreover, a code book consists of eigenvectors. On the other hand, the summation of vectors calculated by multiplying a compression code and code word in the code book is called a predicted vector which will retrieve the image in decoded process.

(2.2)Additional Information –

In PCA, determining the number of PCs used starts from the target explained variance, then also considered the reduction of image size and number of colours to analyse their similarity with the original image.

There is a trade-off between the variance and the image size. The more variance we want to explain, the bigger is the image size. Hence, we must conclude to choose n principal components because it gives us smaller image size with reasonably high explained variance, and the number of colours is closer to the original image.

Code –

```
# image processing
from PIL import Image
from io import BytesIO
import webcolors

# data analysis
import math
import numpy as np
import pandas as pd

# visualization
import matplotlib.pyplot as plt
from importlib import reload
from mpl_toolkits import mplot3d
import seaborn as sns

# modeling
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler

#import the original image
ori_img = Image.open("images/lena.png")
```

```
plt.imshow(ori_img);

X = np.array(ori_img.getdata())
ori_pixels = X.reshape(*ori_img.size, -1)
print(ori_pixels.shape)

#function for getting size of image

def imageByteSize(img):
    img_file = BytesIO()
    image = Image.fromarray(np.uint8(img))
    image.save(img_file, 'png')
    return img_file.tell()/1024

#check size of original image

ori_img_size = imageByteSize(ori_img)
print(ori_img_size)

#check number of colors in original image

ori_img_n_colors = len(set(ori_img.getdata()))
print(ori_img_n_colors)

# helper functions to get name of colors from rgb values

def closest_colour(requested_colour):
    min_colours = {}
    for key, name in webcolors.CSS3_HEX_TO_NAMES.items():
        r_c, g_c, b_c = webcolors.hex_to_rgb(key)
        rd = (r_c - requested_colour[0]) ** 2
        gd = (g_c - requested_colour[1]) ** 2
        bd = (b_c - requested_colour[2]) ** 2
        min_colours[(rd + gd + bd)] = name
    return min_colours[min(min_colours.keys())]

def get_colour_name(requested_colour):
    try:
        closest_name = actual_name = webcolors.rgb_to_name(requested_colour)
```

```
    except ValueError:
        closest_name = closest_colour(requested_colour)
    return closest_name

#function to show image from array

def plotImage(img_array, size):
    reload(plt)
    plt.imshow(np.array(img_array/255).reshape(*size))
    plt.axis('off')
    return plt

#K Means Clustering

def replaceWithCentroid(kmeans):
    new_pixels = []
    for label in kmeans.labels_:
        pixel_as_centroid = list(kmeans.cluster_centers_[label])
        new_pixels.append(pixel_as_centroid)
    new_pixels = np.array(new_pixels).reshape(*ori_img.size, -1)
    return new_pixels

def calculateBCSS(X, kmeans):
    _, label_counts = np.unique(kmeans.labels_, return_counts = True)
    diff_cluster_sq = np.linalg.norm(kmeans.cluster_centers_ - np.mean(X, axis = 0),
axis = 1)**2
    return sum(label_counts * diff_cluster_sq)

#perform clustering with different number of K

range_k_clusters = (2, 21)

kmeans_result = []
for k in range(*range_k_clusters):
    # CLUSTERING
    kmeans = KMeans(n_clusters = k,
                    n_jobs = -1,
                    random_state = 123).fit(X)
```

```
    # REPLACE PIXELS WITH ITS CENTROID
    new_pixels = replaceWithCentroid(kmeans)

    # EVALUATE
    WCSS = kmeans.inertia_
    BCSS = calculateBCSS(X, kmeans)
    exp_var = 100*BCSS/(WCSS + BCSS)

    metric = {
        "No. of Colors": k,
        "Centroids": list(map(get_colour_name, np.uint8(kmeans.cluster_centers_))),
        "Pixels": new_pixels,
        "WCSS": WCSS,
        "BCSS": BCSS,
        "Explained Variance": exp_var,
        "Image Size (KB)": imageByteSize(new_pixels)
    }

    kmeans_result.append(metric)
kmeans_result = pd.DataFrame(kmeans_result).set_index("No. of Colors")

# plotting the different colour reduced images

fig, axes = plt.subplots(4, 5, figsize=(15,15))

# PLOT ORIGINAL IMAGE
axes[0][0].imshow(X.reshape(*ori_img.size, 3))
axes[0][0].set_title("Original Image: {} Colors".format(ori_img_n_colors), fontsize = 20)
axes[0][0].set_xlabel("Image Size: {:.3f} KB".format(ori_img_size), fontsize = 15)
axes[0][0].set_xticks([])
axes[0][0].set_yticks([])

# PLOT COLOR-REDUCED IMAGE
for ax, k, pixels in zip(axes.flat[1:], kmeans_result.index, kmeans_result["Pixels"]):
    compressed_image = np.array(pixels/255).reshape(*ori_img.size, 3)
    ax.imshow(compressed_image)
    ax.set_title("{} Colors".format(k), fontsize=20)
```

```
    ax.set_xlabel("Explained Variance: {:.3f}%\nImage Size: {:.3f}
KB".format(kmeans_result.loc[k, "Explained Variance"],kmeans_result.loc[k, "Image
Size (KB)"]),fontsize=15)
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
fig.suptitle("IMAGE WITH INCREASING NUMBER OF COLORS", size = 30, y = 1.03,
fontweight = "bold")
plt.show()

# To calculate optimal number of clusters

def locateOptimalElbow(x, y):
    # START AND FINAL POINTS
    p1 = (x[0], y[0])
    p2 = (x[-1], y[-1])

    # EQUATION OF LINE: y = mx + c
    m = (p2[1] - p1[1]) / (p2[0] - p1[0])
    c = (p2[1] - (m * p2[0]))

    # DISTANCE FROM EACH POINTS TO LINE mx - y + c = 0
    a, b = m, -1
    dist = np.array([abs(a*x0+b*y0+c)/math.sqrt(a**2+b**2) for x0, y0 in zip(x,y)])
    return np.argmax(dist) + x[0]

def calculateDerivative(data):
    derivative = []
    for i in range(len(data)):
        if i == 0:
            # FORWARD DIFFERENCE
            d = data[i+1] - data[i]
        elif i == len(data) - 1:
            # BACKWARD DIFFERENCE
            d = data[i] - data[i-1]
        else:
            # CENTER DIFFERENCE
            d = (data[i+1] - data[i-1])/2
        derivative.append(d)
```

```
    return np.array(derivative)

def locateDrasticChange(x, y):
    # CALCULATE GRADIENT BY FIRST DERIVATIVE
    first_derivative = calculateDerivative(np.array(y))

    # CALCULATE CHANGE OF GRADIENT BY SECOND DERIVATIVE
    second_derivative = calculateDerivative(first_derivative)

    return np.argmax(np.abs(second_derivative)) + x[0]

optimal_k = []
for col in kmeans_result.columns[2:]:
    optimal_k_dict = {}
    optimal_k_dict["Metric"] = col
    if col == "Image Size (KB)":
        optimal_k_dict["Method"] = "Derivative"
        optimal_k_dict["Optimal k"] = locateDrasticChange(kmeans_result.index,
kmeans_result[col].values)
    else:
        optimal_k_dict["Method"] = "Elbow"
        optimal_k_dict["Optimal k"] = locateOptimalElbow(kmeans_result.index,
kmeans_result[col].values)
    optimal_k.append(optimal_k_dict)
optimal_k = pd.DataFrame(optimal_k)

k_opt = optimal_k["Optimal k"].max()
print(k_opt)

#compare original and color reduced image

ori = {
    "Type": "Original",
    "Pixels": X,
    "No. of Colors": ori_img_n_colors,
    "Image Size (KB)": ori_img_size,
    "Explained Variance": 100
}
color_reduced = {
```

```
    "Type": "Color-Reduced",
    "Pixels": kmeans_result.loc[k_opt, "Pixels"],
    "No. of Colors": k_opt,
    "Image Size (KB)": kmeans_result.loc[k_opt, "Image Size (KB)"],
    "Explained Variance": kmeans_result.loc[k_opt, "Explained Variance"]
}
ori_vs_kmeans = pd.DataFrame([ori, color_reduced]).set_index("Type")

# PCA

fig, axes = plt.subplots(1, 4, figsize=(15,5))
cmap_list = ["Reds", "Greens", "Blues"]

axes[0].imshow(ori_pixels)
axes[0].axis("off")
axes[0].set_title("Combined", size = 20)

for idx, ax, px in zip(range(3), axes[1:], ori_pixels.T):
    ax.imshow(px.T, cmap = cmap_list[idx])
    ax.axis("off")
    ax.set_title(cmap_list[idx][:-1], size = 20)

plt.tight_layout()
fig.suptitle("IMAGES OF EACH COLOR CHANNEL", size = 30, y = 1.03, fontweight =
"bold")
plt.show()

# Principal component of RGB channel

res = []
cum_var = []
X_t = np.transpose(X)
for channel in range(3):
    # SEPARATE EACH RGB CHANNEL
    pixel = X_t[channel].reshape(*ori_pixels.shape[:2])

    # PCA
    pca = PCA(random_state = 123)
    pixel_pca = pca.fit_transform(pixel)
```

```
    pca_dict = {
        "Projection": pixel_pca,
        "Components": pca.components_,
        "Mean": pca.mean_
    }
    res.append(pca_dict)

    # EVALUATION
    cum_var.append(np.cumsum(pca.explained_variance_ratio_))

# visualise PC of each channel

scaler = MinMaxScaler()
scaled_pixels = [scaler.fit_transform(res[i]["Components"])*255 for i in range(3)]

fig, axes = plt.subplots(1, 4, figsize=(15,5))
cmap_list = ["Reds", "Greens", "Blues"]

axes[0].imshow(np.array(scaled_pixels).T/255)
axes[0].axis("off")
axes[0].set_title("Combined", size = 20)

for idx, ax, px in zip(range(3), axes[1:], scaled_pixels):
    ax.imshow((px/255).T, cmap = cmap_list[idx])
    ax.axis("off")
    ax.set_title(cmap_list[idx][:-1], size = 20)
plt.tight_layout()

fig.suptitle("PRINCIPAL COMPONENTS OF EACH COLOR CHANNEL", size = 30, y = 1.03,
fontweight = "bold")
plt.show()

cum_var_df = pd.DataFrame(np.array(cum_var).T * 100,
                          index = range(1, pca.n_components_+1),
                          columns = ["Explained Variance by Red",
                                     "Explained Variance by Green",
                                     "Explained Variance by Blue"])
cum_var_df["Explained Variance"] = cum_var_df.mean(axis = 1)
```

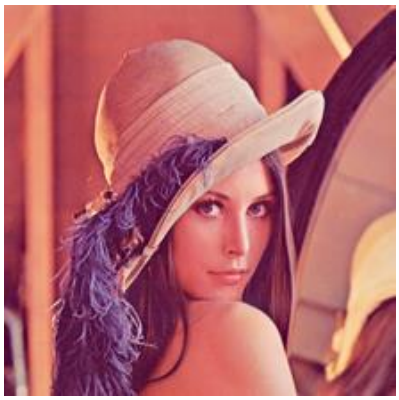
```
#iterating with differnt number of PCs

pca_results = []
for n in range(1, pca.n_components_+1):
   # SELECT N-COMPONENTS FROM PC
   temp_res = []
   for channel in range(3):
      pca_channel = res[channel]
      pca_pixel = pca_channel["Projection"][:, :n]
      pca_comp = pca_channel["Components"][:n, :]
      pca_mean = pca_channel["Mean"]
      compressed_pixel = np.dot(pca_pixel, pca_comp) + pca_mean
      temp_res.append(compressed_pixel.T)
   compressed_image = np.transpose(temp_res)

   pca_dict = {
      "n": n,
      "Pixels": compressed_image,
      "Explained Variance": cum_var_df["Explained Variance"][n],
      "Image Size (KB)": imageByteSize(compressed_image),
      "No. of Colors": len(np.unique(np.uint8(compressed_image).reshape(-1, 3), axis =
0))
   }

   pca_results.append(pca_dict)

pca_results = pd.DataFrame(pca_results).set_index("n")

# plotting some results

n_pca_images = (3, 4)
end_pc = pca_results[pca_results["Explained Variance"] >= 95].index[0]

fig, axes = plt.subplots(*n_pca_images, figsize=(20,15))
plot_n = np.linspace(1, end_pc, n_pca_images[0]*n_pca_images[1]-1, endpoint =
True, dtype = int)

# PLOTTING ORIGINAL IMAGE
```

```
axes[0][0].imshow(X.reshape(*ori_img.size, 3))
axes[0][0].set_title("Original Image", fontsize = 20)
axes[0][0].set_xlabel("Image Size: {:.3f} KB".format(ori_img_size), fontsize = 15)
axes[0][0].set_xticks([])
axes[0][0].set_yticks([])

for ax, n in zip(axes.flat[1:], plot_n):
   # PLOTTING COMPRESSED IMAGE
   ax.imshow(pca_results.loc[n, "Pixels"]/255)
   ax.set_title("{} Principal Component(s)".format(n), fontsize = 20)
   ax.set_xlabel("Explained Variance: {:.3f}%\nImage Size: {:.3f}
KB".format(pca_results.loc[n, "Explained Variance"],
                                                    pca_results.loc[n, "Image Size (KB)"]),
              fontsize = 15)
   ax.set_xticks([])
   ax.set_yticks([])
plt.tight_layout()
fig.suptitle("IMAGE WITH INCREASING NUMBER OF PRINCIPAL COMPONENTS", size =
30, y = 1.03, fontweight = "bold")
plt.show()

# selecting optimal number of PCs

line_colors = "ygr"
fig, axes = plt.subplots(1, 3, figsize=(15,5))
for ax, metric in zip(axes, pca_results.columns[1:]):
   sns.lineplot(x = pca_results.index, y = metric, data = pca_results, ax = ax)
   ax.set_xlabel("No. of Principal Components")

   if metric == "Explained Variance":
      lookup_n_var = []
      for idx, exp_var in enumerate([90, 95, 99]):
         lookup_n = pca_results[pca_results[metric] >= exp_var].index[0]
         lookup_n_var.append(lookup_n)
         ax.axhline(y = exp_var, color = line_colors[idx], linestyle = '--',
                  label = "{}% Explained Variance (n = {})".format(exp_var, lookup_n))
         ax.plot(lookup_n, exp_var, color = line_colors[idx], marker = 'x', markersize = 8)
         ax.set_ylabel("Cumulative Explained Variance (%)")
      ax.legend()
```

```
        continue
    elif metric == "Image Size (KB)":
        y_val = ori_img_size
        line_label = "n = {} (Size: {:.2f} KB)"
    elif metric == "No. of Colors":
        y_val = ori_img_n_colors
        line_label = "n = {} (Colors: {})"

    ax.axhline(y = y_val, color = 'k', linestyle = '--', label = "Original Image")
    for idx, n_components in enumerate(lookup_n_var):
        lookup_value = pca_results.loc[n_components, metric]
        ax.axvline(x = n_components, color = line_colors[idx], linestyle = '--',
                   label = line_label.format(n_components, lookup_value))
        ax.plot(n_components, lookup_value, color = line_colors[idx], marker = 'x',
markersize = 8)
    ax.legend()
plt.tight_layout()
fig.suptitle("METRICS BY NUMBER OF PRINCIPAL COMPONENTS", size = 30, y = 1.07,
fontweight = "bold")
plt.show()

target_exp_var = 95
n_opt = pca_results[pca_results["Explained Variance"] >= target_exp_var].index[0]
print(n_opt)

#compare to original image

pc_reduced = {
    "Type": "PC-Reduced",
    "Pixels": pca_results.loc[n_opt, "Pixels"],
    "No. of Colors": pca_results.loc[n_opt, "No. of Colors"],
    "Image Size (KB)": pca_results.loc[n_opt, "Image Size (KB)"],
    "Explained Variance": pca_results.loc[n_opt, "Explained Variance"]
}
ori_vs_pca = pd.DataFrame([ori, pc_reduced]).set_index("Type")

#comparision of Kmeans and PCA
```

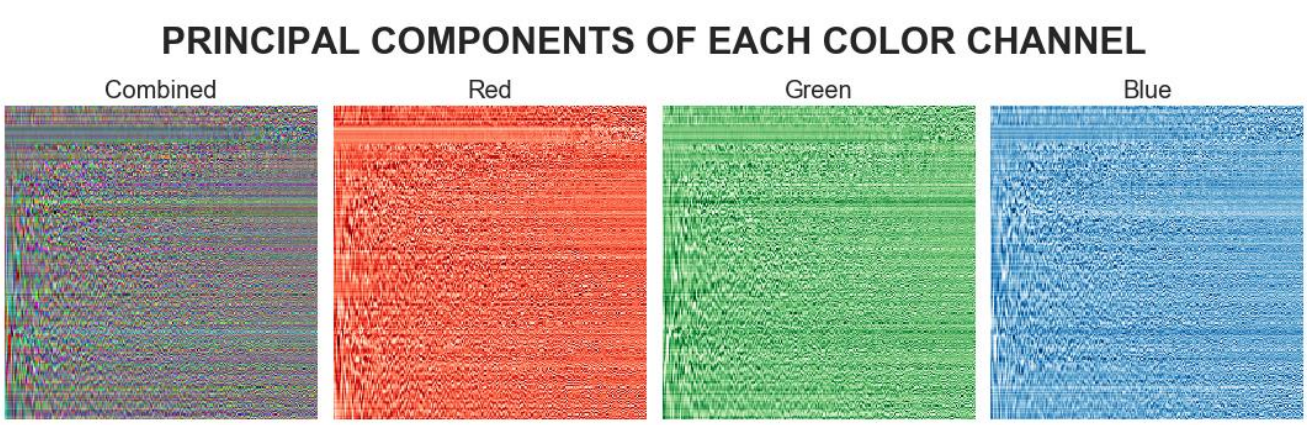
```
reduction_kmeans = (1-final_compare.loc["Color-Reduced", "Image Size (KB)"] /
ori_img_size) * 100
reduction_pca = (1-final_compare.loc["PC-Reduced", "Image Size (KB)"] /
ori_img_size) * 100
print("Image Size Reduction using K-Means: {:.3f}%".format(reduction_kmeans))
print("Image Size Reduction using PCA: {:.3f}%".format(reduction_pca))
```

Results and Observations –



Original Image

Visualisation of Colour Reduced images with different number of colours



Principal Components of each colour channel of the original image

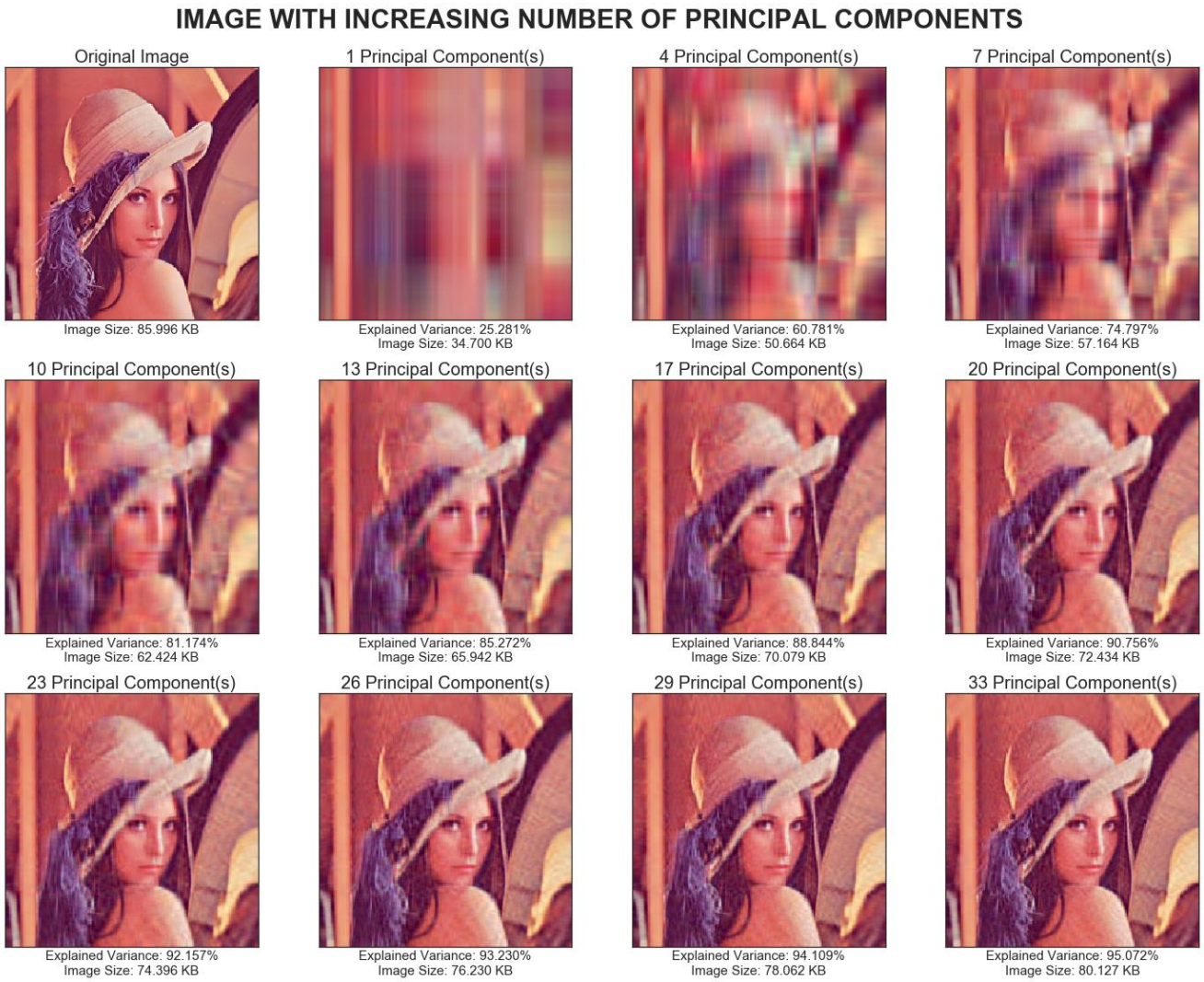


Image reconstructed using different number of principal components

Comparison of Compressed images

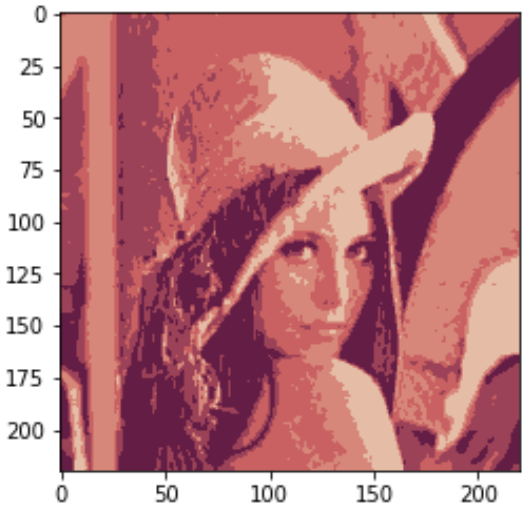


Image Compressed by using PCA followed by KMeans

Sl. No	Image	Explained Variance	Image Size(KB)	Relative reduction in Size
1	Original Image	100%	85.996	-
2	K-Means Compressed Image	95.916%	18.048	79.012%
3	PCA Compressed Image	95.072%	80.127	6.825%
4	Image Compressed by PCA followed by KMeans	89.539%	9.255	89.237%

Both KMeans And PCA are lossy compression techniques in the sense that some of the data is lost during compression.

KMeans compresses by reducing the number of colours required to represent the image. It is reduces the psycho visual redundancy in the image. By using KMeans

algorithm we only retain the most occurring colours in the image and discard the less occurring colours.

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is 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. Using PCA on images reduces the spatial redundancy in the image. We will utilize PCA to reduce the image size by selecting a certain number of principal components to be used so that it only stores the important pixels to preserve the variance of the original image making it more efficient in the storage.

Using both PCA and KMeans simultaneously reduces the image size but the variance in the output image is greatly decreased as can be see n from the results obtained above.

References –

1) https://ieeexplore.ieee.org/abstract/document/5702000
2) https://ias-iss.org/ojs/IAS/article/view/1611
3) https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.782&rep=rep1&type=pdf
4) https://ieeexplore.ieee.org/abstract/document/1017616
5) https://www.researchgate.net/publication/266890475_A_New_Technique_for_Image_Compression_Using_PCA