DATE:22/10/2024

GOA UNIVERSITY

GOA BUSINESS SCHOOL

PROGRAMME: MSc. Data Science (Part I)

PAPER: CSD-502 Machine Learning

TOPIC: PRINCIPAL COMPONENT ANALYSIS (PCA)

DONE BY SANJEEL SANTOSH VELIP

Introduction to Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a powerful dimensionality reduction technique used in machine learning and data analysis. It aims to reduce the number of variables (or dimensions) in a dataset while retaining as much variability (or information) as possible. PCA achieves this by transforming the data into a new coordinate system where the greatest variance by any projection of the data lies on the first axis (called the first principal component), the second greatest variance on the second axis, and so on.

PCA is widely used for data visualization, noise reduction, and, as demonstrated in this assignment, image compression. By representing high-dimensional data (such as an image with three color channels) in a lower-dimensional space, PCA can compress the data efficiently while preserving the critical visual information.

Application: Image Compression with PCA

Problem Overview

Images typically consist of three color channels (Red, Green, and Blue), each contributing to the overall image. These channels can contain redundant information, and reducing the dimensionality of this data can help compress the image without losing too much visual quality.

In this application, PCA is applied to compress an image by reducing the number of principal components used to reconstruct each color channel. Fewer components result in higher compression but potentially lower image quality.

Step-by-Step Approach:

1. **Loading and Preprocessing the Image:** The image is first loaded and converted into a matrix representing the RGB values. Each channel (Red, Green, and Blue) is separated for individual processing.
2. **Applying PCA on Each Channel:** For each color channel, PCA is applied to reduce its dimensionality. The process involves:

- Centering the data by subtracting the mean of the channel.
- Applying PCA to extract the top k principal components.
- Reconstructing the image using the compressed data by performing an inverse PCA transform.

3. **Reconstruction of the Image:** The three compressed channels are then recombined to form a compressed image. The number of principal components (k) controls the amount of compression. Fewer components (k) lead to a more compressed image with lower quality, while higher k values preserve more detail.
4. **Visualizing the Results:** The results of the image compression are visualized by displaying the original image and the reconstructed images using different values of k (e.g., k=100, k=50, and k=20). This allows us to compare the quality of compression visually.

Code Implementation:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from PIL import Image
import os
# Load and represent the image as a matrix (RGB)
image_path = 'IMG20220621115001-01.jpeg'
image = Image.open(image_path)
image_matrix = np.array(image)
# Extract the base name of the original image
base_name = os.path.splitext(os.path.basename(image_path))[0]
# Function to apply PCA on each color channel
def apply_pca_on_channel(channel, k):
    mean_channel = np.mean(channel, axis=0)
    centered_channel = channel - mean_channel
    pca = PCA(n_components=k)
    compressed_channel = pca.fit_transform(centered_channel)
    reconstructed_channel = pca.inverse_transform(compressed_channel)
    reconstructed_channel += mean_channel
    reconstructed_channel = np.clip(reconstructed_channel, 0, 255)
    return reconstructed_channel
```

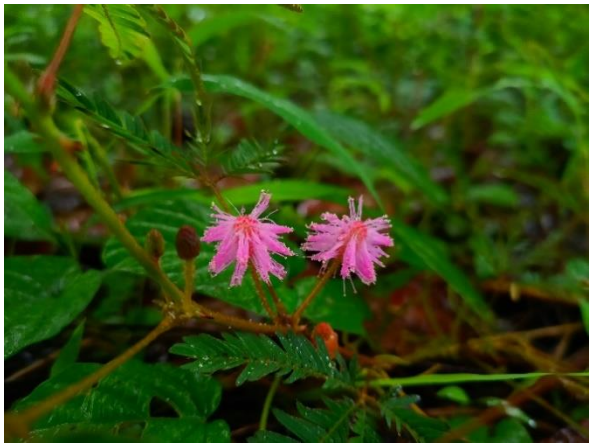
```
# Function to create and save a single frame with multiple images
def save_images_in_one_frame(k_values, filename):
    fig, ax = plt.subplots(1, len(k_values) + 1, figsize=(15, 5))
    ax[0].imshow(image_matrix)
    ax[0].set_title('Original Image')
    ax[0].axis('off')
    for i, k in enumerate(k_values):
        R_channel = image_matrix[:, :, 0]
        G_channel = image_matrix[:, :, 1]
        B_channel = image_matrix[:, :, 2]
        R_compressed = apply_pca_on_channel(R_channel, k)
        G_compressed = apply_pca_on_channel(G_channel, k)
        B_compressed = apply_pca_on_channel(B_channel, k)
        reconstructed_image = np.stack((R_compressed, G_compressed, B_compressed), axis=2).astype(np.uint8)
        compressed_image_filename = f'{base_name}_k={k}.jpg'
        Image.fromarray(reconstructed_image).save(compressed_image_filename)
        ax[i + 1].imshow(reconstructed_image)
        ax[i + 1].set_title(f'{k} Components')
        ax[i + 1].axis('off')
    plt.tight_layout()
    plt.savefig(filename, dpi=300)
    plt.show()
# Values of k to try
k_values = [100, 50, 20]
save_images_in_one_frame(k_values, 'compressed_images_frame.jpg')
```

Results and Analysis:

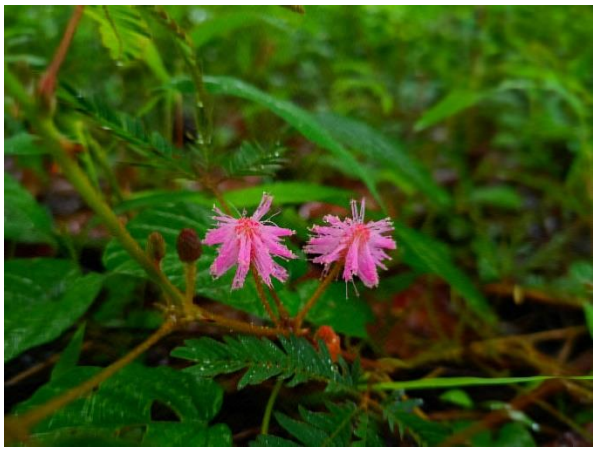
The following values of k were used for the compression process:

- k = 100: The image is compressed while maintaining a high level of detail. Most of the essential visual information is preserved, with only minor artifacts.
- k = 50: The image shows noticeable degradation in quality, but it is still visually acceptable for many purposes. The compression is more efficient, but some details are lost.
- k = 20: This represents a highly compressed version of the image, with substantial quality loss. The image appears blurry, and fine details are lost. However, the overall structure and colors are still recognizable.

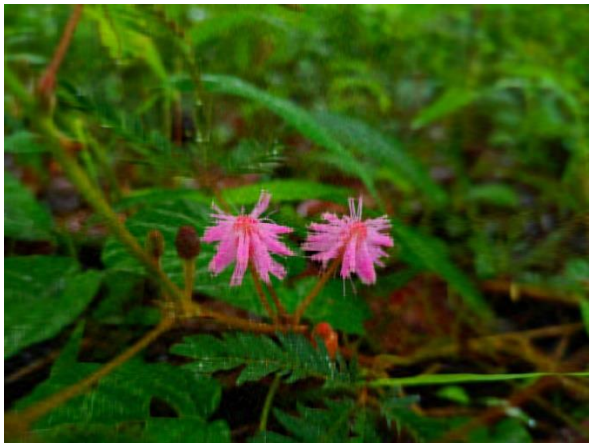
k-value	Visual Quality	File Size
Original	High	3.74 mb
k = 100	Slightly reduced quality	0.99 mb
k = 50	Noticeable quality	945 kb
k = 20	Blurry, major loss of detail	832 kb



Original



k = 100



k = 50

k = 20

Key Insights:

- There is a trade-off between image quality and file size: fewer principal components lead to smaller file sizes but reduced visual quality.
- PCA is an effective method for compressing images when storage is a constraint, especially for applications where slight quality loss is acceptable.

References

1. Jolliffe, I. T. (2002). Principal Component Analysis. Springer Series in Statistics.
2. Smith, L. I. (2002). A tutorial on Principal Components Analysis. Retrieved from https://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
3. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
4. Shlens, J. (2014). A tutorial on Principal Component Analysis. arXiv preprint arXiv:1404.1100.
5. Sharma, A. (2019). How to use PCA for image compression. Retrieved from https://towardsdatascience.com/how-to-use-pca-for-image-compression-65d5a089d8f8