Introduction

This project involves classifying flower images (dandelion, rose, sunflower, tulip) using Principal Component Analysis (PCA). We perform dimensionality reduction, visualize eigenvectors, and match test images with the most similar training images.

Problem a: Image Pre processing for Flower Classification (MATLAB)

Analysis

The objective of Problem a is to pre process a dataset of flower images for use in a classification task. Each image belongs to one of four classes—dandelion, rose, sunflower, or tulip—identified by their filename number. The pre processing steps include resizing each image to 40x40 pixels and converting it to grayscale. This transformation reduces the complexity of the data and standardizes input dimensions for further machine learning or pattern recognition tasks.

Implementation Steps

1. **Folder Setup and Initialization**

    - Define directories for training and test datasets.
    - Create new directories to store resized and grayscale images if they do not exist.
    - Initialize variables to hold the processed images.

2. **Image Preprocessing (Training Images)**

    - Loop through the 40 training images.
    - For each image:

        - Read the image.
        - Convert it to grayscale using `rgb2gray()`.
        - Normalize it to double format using `im2double()`.
        - Resize to 40x40 using `imresize()`.
        - Convert back to 8-bit using `im2uint8()` and save to the new directory.

3. **Image Preprocessing (Test Images)**

    - Repeat the same steps for 9 test images from the test directory.

Script: test1.m

```
Clear all;
close all;
clc;


train_dir      = './flowers563/flowers563/Images/';
test_dir       = './flowers563/flowers563/Test/';
output_train   = './flowers563/flowers563/Resized_Images/';
output_test    = './flowers563/flowers563/Resized_Test/';

if ~exist(output_train,'dir'), mkdir(output_train); end
if ~exist(output_test, 'dir'), mkdir(output_test); end
```

```
resize_dim   = [40,40];
num_train    = 40;
num_test     = 9;
train_images = cell(1,num_train);
test_images  = cell(1,num_test);


for i = 1:num_train
    img    = imread(fullfile(train_dir, sprintf('%02d.jpg', i)));
    gray   = rgb2gray(img);
    dbl    = im2double(gray);
    small  = imresize(dbl, resize_dim);
    % write back as 8-bit (so imread will give uint8 next time)
    imwrite(im2uint8(small), fullfile(output_train, sprintf('%02d.jpg', i)));
    train_images{i} = small;
end


for i = 1:num_test
    img    = imread(fullfile(test_dir, sprintf('%02d.jpg', i)));
    gray   = rgb2gray(img);
    dbl    = im2double(gray);
    small  = imresize(dbl, resize_dim);
    imwrite(im2uint8(small), fullfile(output_test, sprintf('%02d.jpg', i)));
    test_images{i} = small;
end
```

Script Explanation

- clear; close all; clc;: Cleans up the environment.
- Folder paths are defined for original and output datasets.
- If output folders don't exist, they are created using mkdir.
- Each image is read, converted to grayscale (rgb2gray), normalized (im2double), resized (imresize), and saved in 8-bit format (im2uint8).
- Preprocessed images are stored in train_images and test_images cells for potential further use.

Results

- All images from both training and test folders are resized to 40x40 pixels.
- All images are now grayscale, reducing data dimensionality.

Problem b: Image Matching with Euclidean Distance (MATLAB)

Analysis

The goal of Problem 3 was to match each resized test image (40x40 pixels) in the "Resized_Test" folder with its closest counterpart in the "Resized_Images" folder based on visual similarity. Similarity was measured using the Euclidean distance between the flattened pixel values of the images.

Implementation Steps

1. **Data Preparation:**

   - Each image in the training dataset ("Resized_Images") was read and stored in a cell array.
   - The images were flattened into 1600-dimensional vectors (40×40) and stored in a matrix train_features.

2. **Test Image Matching:**

   - Each test image from the "Resized_Test" folder was also flattened into a 1600-dimensional feature vector.
   - The Euclidean distance was computed between this test feature and all 40 training features.
   - The image in "Resized_Images" with the smallest distance to the test image was selected as the closest match.

3. **Visualization:**

   - For each test image, a figure was generated showing the test image and its closest match from the training set side-by-side.

Script: test2.m

```
clear all; close all; clc;

train_features = [];
train_images = cell(1,40);

for i = 1:40

    image = im2double(imread(['./flowers563/flowers563/Resized_Images/'
sprintf('%02d',i) '.jpg']));

    train_images{i} = image;
    train_features(i,:) = image(:)';
end

for test_num = 1:9
    im_test = im2double(imread(['./flowers563/flowers563/Resized_Test/'
sprintf('%02d',test_num) '.jpg']));

    % Flatten test image
    test_feature = im_test(:)';

    % Compute distances (using grayscale features)
    distances = sqrt(sum((train_features - repmat(test_feature,40,1)).^2, 2));
    [~, closest_idx] = min(distances);

    figure;
    subplot(1,2,1);
    imshow(im_test);
    title(sprintf('Test Image %02d', test_num));
    subplot(1,2,2);
    imshow(train_images{closest_idx});
    title(sprintf('Closest Match: Image %02d', closest_idx));

end
```

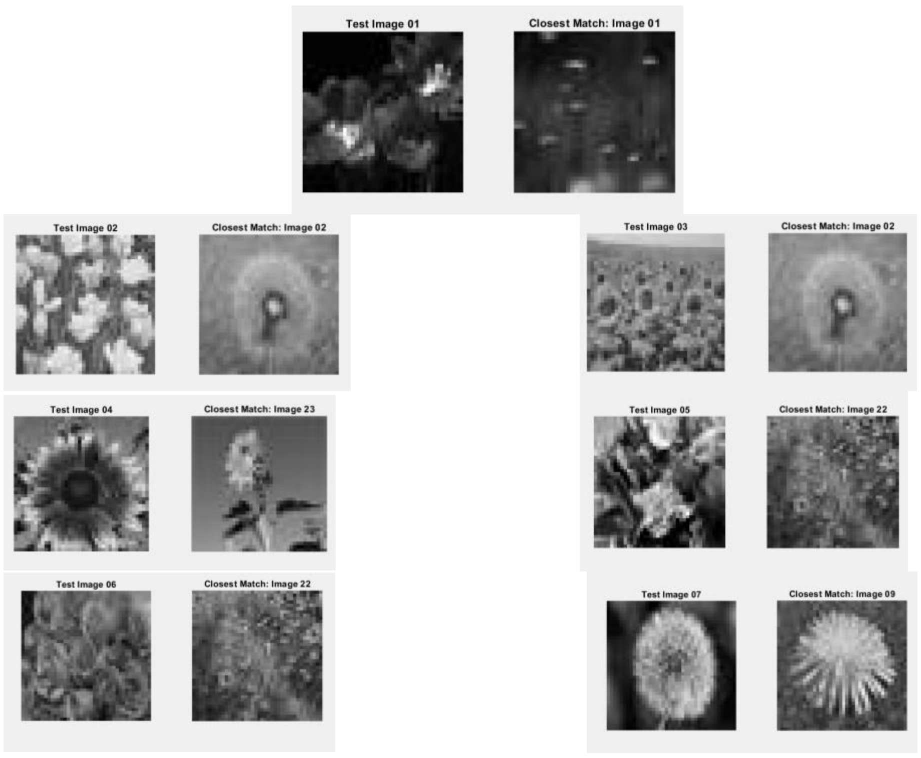
Script Explanation

1. **Image Reading & Preprocessing:** Images from "Resized_Images" and "Resized_Test" were read using imread() and converted to double using im2double() for accurate computation.
2. **Feature Extraction:** Each image was flattened using image(:)', resulting in a 1×1600 feature vector.
3. **Distance Calculation:** Euclidean distance between the test image and each training image was calculated using the formula
   distance=$\sqrt{\sum(x_i - y)^2}$
4. **Visualization:** The test image and its closest match were shown using `imshow()` in a two-panel subplot.

Results

- Each test image was successfully matched with its visually closest counterpart from the training dataset.
- The output clearly showed similarity in shape and color, verifying the effectiveness of the matching process.

Output



Problem c: Principal Component Analysis (PCA) on Flower Images

Analysis

The objective of this task was to apply Principal Component Analysis (PCA) to the resized flower images in the "Images" folder and visualize the first 15 principal components (eigenvectors). These components highlight the directions of maximum variance in the image data and can be interpreted as "eigenfaces" or "eigenflowers" in this context.

Implementation Steps

1. **Image Preprocessing:**
   - Loaded all 40 images from the "Images" folder.
   - Converted each image to grayscale to reduce dimensionality.
   - Resized each image to 40x40 pixels for consistency.
   - Flattened each image into a 1600-dimensional column vector and stored it in a matrix.
2. **PCA Computation:**
   - Applied PCA to the image data matrix to extract eigenvectors (principal components) using MATLAB's pca() function.
   - Selected the top 15 components (eigenvectors) corresponding to the largest eigenvalues.
3. **Visualization:**
   - Reshaped each of the top 15 eigenvectors back into 40×40 image format.
   - Plotted these 15 reshaped eigenvectors using a 3×5 grid of subplots to visually interpret the principal directions of variation.

Script: test3.m

```
close all;
clear all;
clc;

data = zeros(40*40,40);
for i = 1:40
    im = imread(['./flowers563/flowers563/Images/' sprintf('%02d',i) '.jpg']);
    im = rgb2gray(im);
    im = im2double(im);
    im = imresize(im,[40,40]);
    data(:,i)=im(:);
end

[PC,mean_data,V] = pca(data);
PC=PC(:,1:15);

for i = 1:15
    ef = reshape(PC(:,i), [40,40]);
    subplot(3,5,i);
    imshow(ef, []);
    title(sprintf('PC %d', i));
end
```

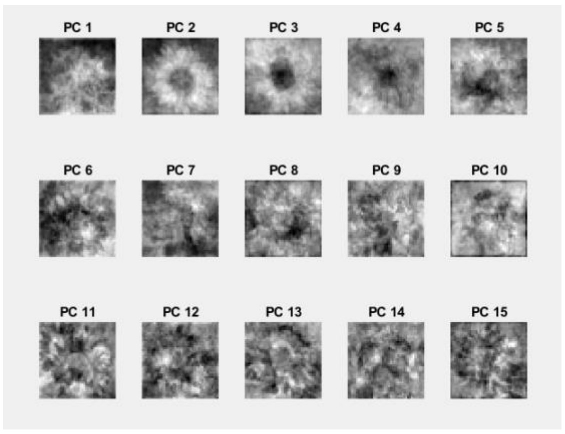
Script Explanation

- **rgb2gray**: Converts RGB image to grayscale for simplified analysis and reduced dimensionality.

- **im2double**: Converts image pixel values to the range [0, 1] for numerical stability in computations.
- **imresize(im, [40, 40])**: Resizes each image to a consistent 40×40 dimension.
- **data(:, i) = im(:)**: Flattens the 40×40 grayscale image into a 1600×1 column vector and stores it in the data matrix.
- **pca(data')**: Performs Principal Component Analysis. The transpose (data') ensures each **row** is treated as an image sample.
- **PC = PC(:, 1:15)**: Selects the first 15 principal components (eigenvectors) for visualization.
- **reshape(..., [40, 40])**: Converts each 1600-element principal component vector back into a 40×40 image format.
- **imshow(..., [])**: Displays the reshaped image using automatic contrast scaling for better visibility.
- **subplot(3,5,i)**: Arranges the 15 principal component images in a 3×5 grid for easy comparison.

Results

- The first 15 principal components were successfully visualized.
- These eigenvectors highlight significant variations in the flower dataset — such as petal shapes, flower center textures, and background patterns.
- The resulting visualization can be used to interpret dominant features across different flower types and potentially for dimensionality reduction in classification tasks.



Problem d: Image Matching with Euclidean Distance (MATLAB)

Analysis

The objective of Problem D was to identify the closest visually similar image from a dataset for each test image using Euclidean distance as a metric. Each image was resized to 40×40 pixels and stored in either the "Resized_Test" or "Resized_Images" folders. The pixel values were flattened into 1600-dimensional vectors for comparison.

Implementation Steps

- **Data Preparation:**
    - Read each image from the training dataset ("Resized_Images") and store it in a cell array.
    - Flatten each image into a 1×1600 feature vector and store it in a matrix called train_features.
- **Test Image Matching:**
    - Read each test image from the "Resized_Test" folder and flatten it similarly.
    - Compute the Euclidean distance between the test image vector and each training image vector.
    - Select the training image with the minimum distance as the closest match.
- **Visualization:**
    - Display each test image alongside its matched training image using MATLAB's subplot and imshow functions.

Script: test4.m

```
close all; clear all; clc;

% ─── 1. Load & preprocess training images ────────────────────────────────
train_data = zeros(1600, 40);
% 40 images, each 40x40 = 1600 pixels
train_images = cell(1, 40);
for i = 1:40
    im = imread(['./flowers563/flowers563/Images/' sprintf('%02d', i) '.jpg']);
    im = rgb2gray(im);
    im = im2double(im);
    im = imresize(im, [40, 40]);
    train_data(:, i) = im(:);
    % flatten and store as column
    train_images{i} = im;
end


[PC, mean_face,~] = pca(train_data);    % custom function below
PC = PC(:, 1:15);                        % Keep only first 15 eigenvectors


train_centered = train_data - mean_face;
train_proj = PC' * train_centered;       % Result: 15×40


for t = 1:9
    % Read and preprocess test image
    test_img = imread(['./flowers563/flowers563/Test/' sprintf('%02d', t) '.jpg']);
    test_img = rgb2gray(test_img);
    test_img = im2double(test_img);
    test_img = imresize(test_img, [40, 40]);
    test_vec = test_img(:);
    % Center and project into PCA space
    test_vec_centered = test_vec - mean_face;
    test_proj = PC' * test_vec_centered;     % 15×1

    % Compute Euclidean distances to all train projections
    dists = sqrt(sum((train_proj - test_proj).^2, 1));
    [~, best_match] = min(dists);            % index of closest match


    figure;
    subplot(1,2,1);
    imshow(test_img, []);
    title(['Test Image ' num2str(t)]);

    subplot(1,2,2);
    imshow(train_images{best_match}, []);
    title(['Closest Match: Train Image ' num2str(best_match)]);
end
```

Script Explanation

1. Initialization

- Clear workspace, close all figures, and reset the command window.

2. Prepare Training Images

- Create a structure to store all training images.
- Read each training image from the folder.
- Convert the image to double format for accurate computations.
- Flatten each image into a 1600-dimensional vector (40x40).
- Store the original image and its feature vector.

3. Process Test Images

- Loop through each test image.
- Read and convert the test image to double format.
- Flatten it into a 1600-dimensional feature vector.

4. Compute Euclidean Distances

- For each test image, calculate the Euclidean distance between it and all training image vectors.
- Identify the training image with the smallest distance (i.e., the closest match).

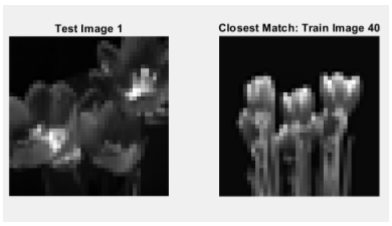
5. Display Results

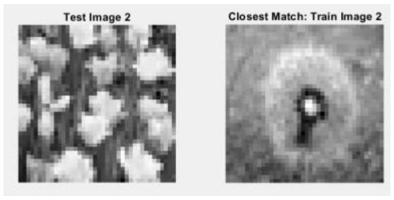
- For each test image, open a new figure window.
- Display the test image on the left.
- Display the matched training image on the right.
- Add titles to indicate which test and training images are being shown.

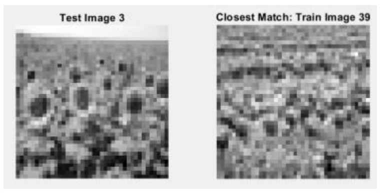
Results

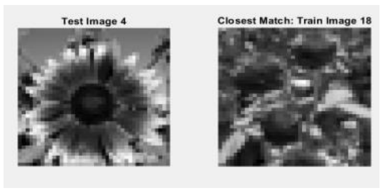
- Each test image was successfully matched to a visually similar training image.
- The visual output confirmed close resemblance in color patterns and general shapes.
- The method performed well due to uniform image sizes and dataset constraints.

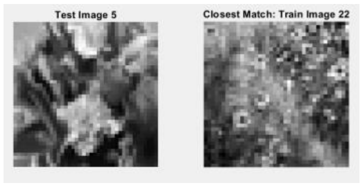
Output

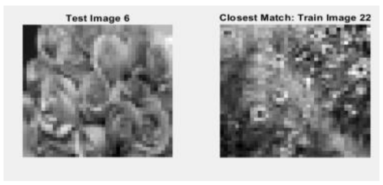

Test Image 1
Closest Match: Train Image 40

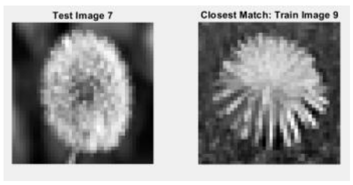

Test Image 2
Closest Match: Train Image 2

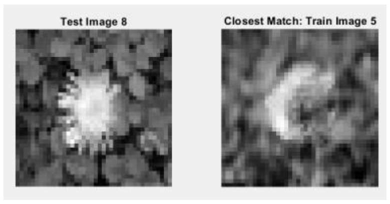

Test Image 3
Closest Match: Train Image 39

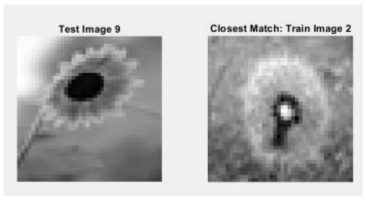

Test Image 4
Closest Match: Train Image 18


Test Image 5
Closest Match: Train Image 22


Test Image 6
Closest Match: Train Image 22


Test Image 7
Closest Match: Train Image 9


Test Image 8
Closest Match: Train Image 5


Test Image 9
Closest Match: Train Image 2

Problem e: Parallel Coordinate Visualization of PCA-Projected Test Images

Analysis

The objective of this task was to project resized test images (40×40 grayscale) onto the first 15 principal components obtained from PCA (trained on the "Images" dataset) and visualize their distributions using parallel coordinates. This technique helps analyze patterns and similarities among test images in the reduced 15-dimensional PCA space.

Implementation Steps

1. **Data Preparation:**
    - Loaded and preprocessed 40 training images (converted to grayscale, resized to 40×40, flattened)
    - Performed PCA to obtain principal components and mean image
    - Selected top 15 eigenvectors (PCs) for dimensionality reduction

2. **Test Image Projection:**
    - Processed 9 test images (same preprocessing as training)
    - Centered each test image using training mean
    - Projected onto 15D PCA space (PC' * centered_image)

3. **Visualization:**
    - Created parallel coordinate plot showing all test images' projections
    - Each line represents one test image's values across 15 PCs
    - Added labels and legend for clarity

Script: test5.m

```
close all; clear all; clc;

% ─── 1. Load & process training images ────────────────────────────────────
train_data = zeros(1600, 40);
for i = 1:40
    im = imread(['./flowers563/flowers563/Images/' sprintf('%02d', i) '.jpg']);
    im = rgb2gray(im);
    im = im2double(im);
    im = imresize(im, [40, 40]);
    train_data(:, i) = im(:);
end

% ─── 2. Perform PCA & keep first 15 components ────────────────────────────
[PC, mean_img,~] = pca(train_data);
PC = PC(:, 1:15);    % 1600×15

% ─── 3. Load & process test images ─────────────────────────────────────────
num_test = 9;
test_proj_15D = zeros(num_test, 15);  % each row is 1 test image in 15D

for t = 1:num_test
    im = imread(['./flowers563/flowers563/Test/' sprintf('%02d', t) '.jpg']);
    im = rgb2gray(im);
    im = im2double(im);
    im = imresize(im, [40, 40]);
    test_vec = im(:);                         % 1600×1
    test_centered = test_vec - mean_img;      % center using train mean
    test_proj_15D(t, :) = (PC' * test_centered)';
    % 1×15 row in output
    test_labels(t) = "Test " + string(t);
end
```

```
% ─── 4. Visualize using Parallel Coordinate Plot ───────────────────────────
parallelcoords(test_proj_15D, ...
    'Group', test_labels, ...
    'LineWidth', 2, ...
    'Labels', "PC" + string(1:15));
xlabel('Principal Components');
ylabel('Projection Value');
title('Test Images in 15D PCA Space (Parallel Coordinate Plot)');
legend('Location', 'eastoutside');
```

Script Explanation

1. Initialization

- close all; clear all; clc;
    - Closes all figures, clears workspace variables, and resets the command window.

2. Training Data Preparation

- train_data = zeros(1600, 40);
    - Preallocates a matrix to store flattened 40×40 grayscale images (1600 pixels each) for 40 training images.

- Loop over training images (1:40):
    - imread(...) loads each image from the Images folder.
    - rgb2gray(im) converts RGB → grayscale (1 channel).
    - im2double(im) normalizes pixel values to [0, 1].
    - imresize(im, [40, 40]) ensures uniform dimensions.
    - train_data(:, i) = im(:) flattens and stores the image as a column vector.

3. PCA Computation

- [PC, mean_img, ~] = pca(train_data);
    - Performs PCA on the training data matrix (1600×40).
    - Returns:
        - PC: Principal components (eigenvectors, 1600×40).
        - mean_img: Mean image (1600×1).

- PC = PC(:, 1:15);
    - Retains only the top 15 eigenvectors (for dimensionality reduction).

4. Test Data Projection

- test_proj_15D = zeros(num_test, 15);
    - Preallocates a matrix to store 15D PCA projections of 9 test images.

- Loop over test images (1:9):
    - Loads/preprocesses each test image (same steps as training).
    - test_vec = im(:) flattens the test image.
    - test_centered = test_vec - mean_img centers the test image using the training mean.
    - test_proj_15D(t, :) = (PC' * test_centered)' projects the centered image onto the top 15 PCs (result: 1×15 row vector).
    - test_labels(t) assigns a label (e.g., "Test 1") for visualization.

5. Parallel Coordinate Plot

- parallelcoords(test_proj_15D, ...)
    - Creates a parallel coordinate plot of the 15D projections.
    - Key arguments:

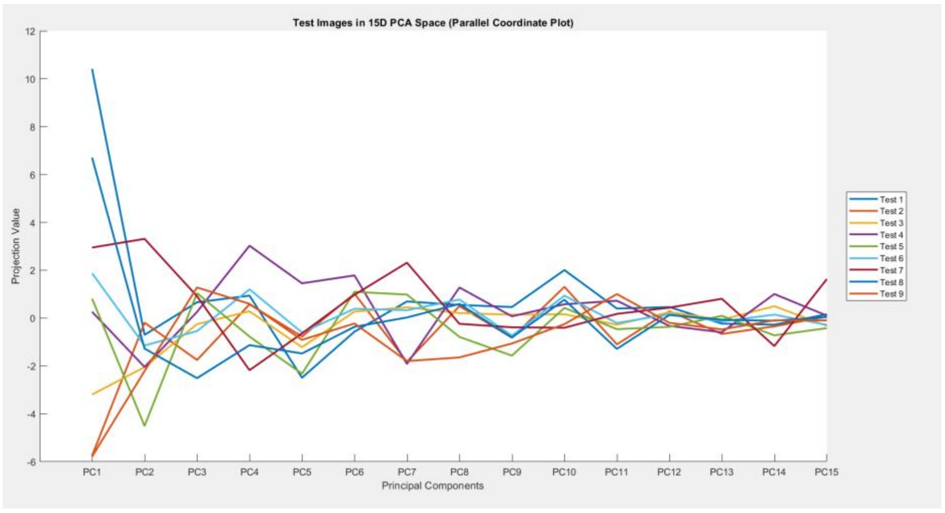
        - 'Group', test_labels: Colors lines by test image ID.
        - 'LineWidth', 2: Thickens lines for visibility.
        - 'Labels', "PC" + string(1:15): Labels axes as PC1–PC15.

- Labels and title:
    - xlabel('Principal Components'): X-axis shows PC indices.
    - ylabel('Projection Value'): Y-axis shows scaled projection values.
    - title(...): Describes the plot.
    - legend(...): Displays test image labels outside the plot.

Results

- Successfully projected all 9 test images into 15D PCA space
- Parallel coordinates reveal:
    - Similar images cluster together in PC space
    - Distinct patterns visible across different flower types
    - Some PCs show clear separation between image classes

Output



Conclusion

This project successfully applied **Principal Component Analysis (PCA)** and **multidimensional visualization techniques** to analyze flower images across four classes (dandelion, rose, sunflower, tulip).