Coding Sample: Facial Recognition (Eigenfaces with PCA & SVD)

Language: Python
Skills Demonstrated: Linear Algebra (PCA, SVD, eigen decomposition), Image Processing & Computer Vision, Algorithm Design, Feature Engineering, Object-Oriented Programming

Objective

The goal of this program is to implement a facial recognition system using Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). The model classifies faces by identifying "eigenfaces" -- weighted principal components that capture the most significant facial features distinguishing one individual from another.

This project demonstrates both mathematical understanding and applied machine learning implementation on a structured image dataset.

Dataset

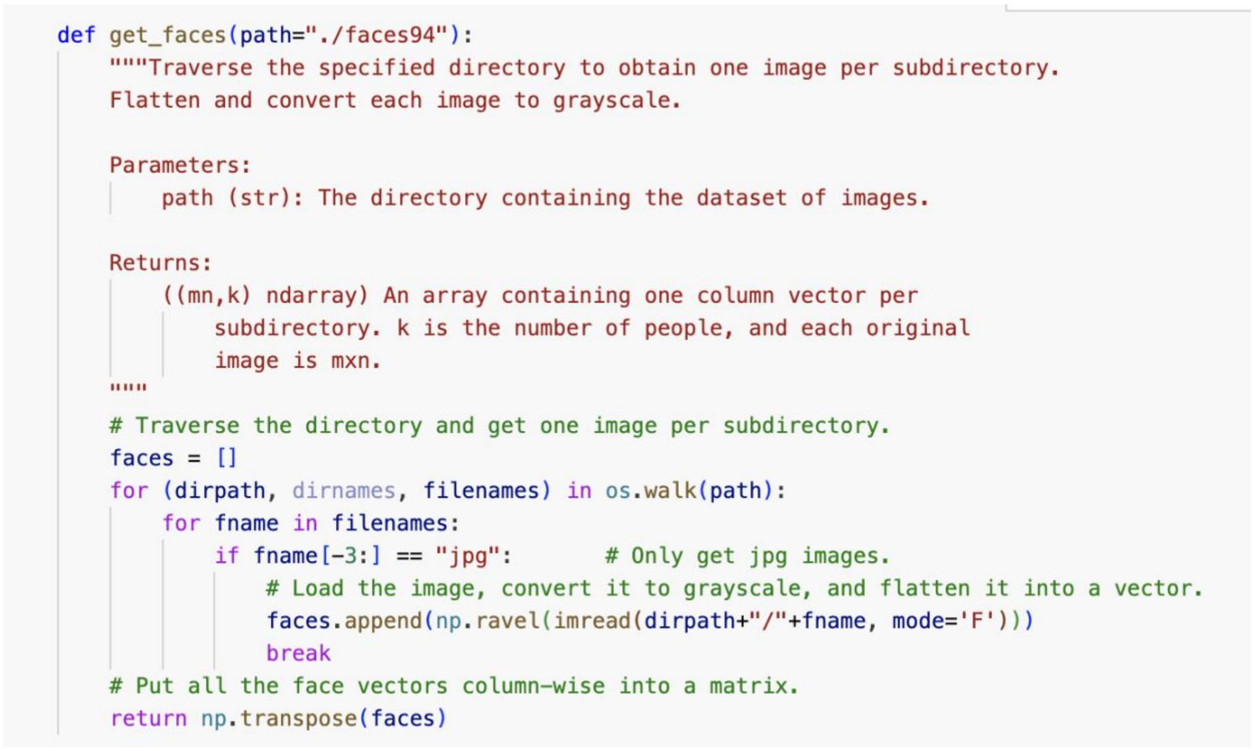
Our dataset contains multiple grayscale photographs of 153 individuals.

- Training: 1 image per individual (153 total)
- Testing: Remaining images used for face-matching validation

Each 200×180-pixel image was converted to grayscale, flattened into a 36,000-dimensional vector, and stored as a column in a single data matrix, where each column corresponds to one facial image.

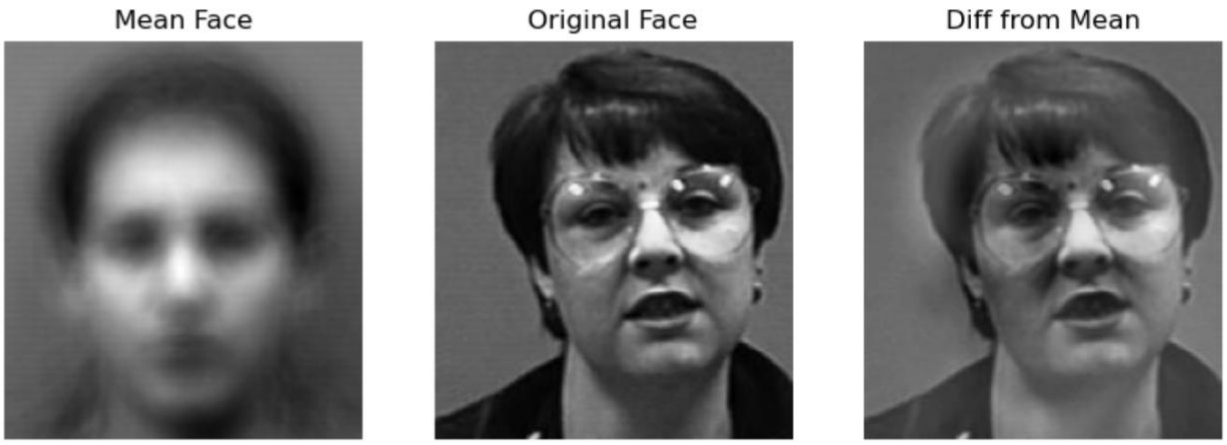
Implementation Overview

Photos were imported, standardized, and stacked into a data matrix representation.



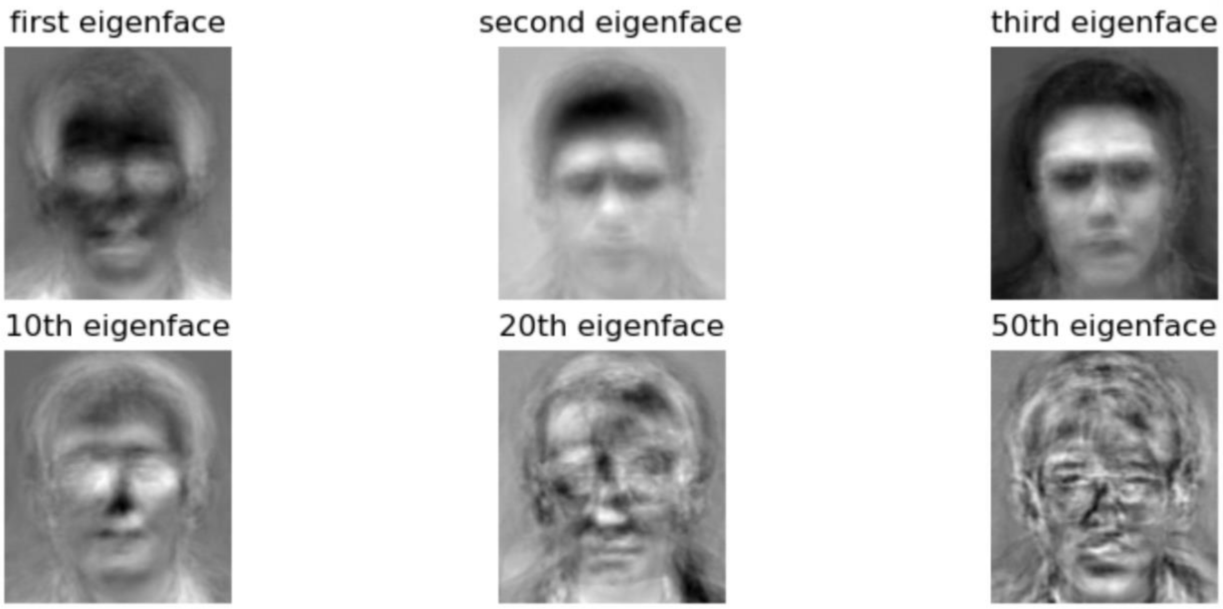
```python
def get_faces(path="./faces94"):
    """Traverse the specified directory to obtain one image per subdirectory.
    Flatten and convert each image to grayscale.

    Parameters:
        path (str): The directory containing the dataset of images.

    Returns:
        ((mn,k) ndarray) An array containing one column vector per
            subdirectory. k is the number of people, and each original
            image is mxn.
    """
    # Traverse the directory and get one image per subdirectory.
    faces = []
    for (dirpath, dirnames, filenames) in os.walk(path):
        for fname in filenames:
            if fname[-3:] == "jpg":       # Only get jpg images.
                # Load the image, convert it to grayscale, and flatten it into a vector.
                faces.append(np.ravel(imread(dirpath+"/"+fname, mode='F')))
                break
    # Put all the face vectors column-wise into a matrix.
    return np.transpose(faces)
```

We computed the mean face and subtracted it from each image to emphasize the unique features of each individual. This preprocessing step effectively amplifies the differences between faces while reducing common background variation.



Using Singular Value Decomposition, we decomposed the matrix into three components: U, Sigma, and V Transpose. The columns of U represent the principal directions in facial data space. These orthonormal vectors correspond to “eigenfaces,” which each encode a distinct kind of facial variation such as lighting, expression, or facial geometry.

```python
def __init__(self, path='./faces94'):
    """Initialize the F, mu, Fbar, and U attributes.
    This is the main part of the computation.
    """
    self.F = get_faces(path)
    self.mean = np.mean(self.F, axis=1)
    self.shifted = self.F - self.mean[:, np.newaxis]
    self.U, _, _ = np.linalg.svd(self.shifted, full_matrices=False)
```



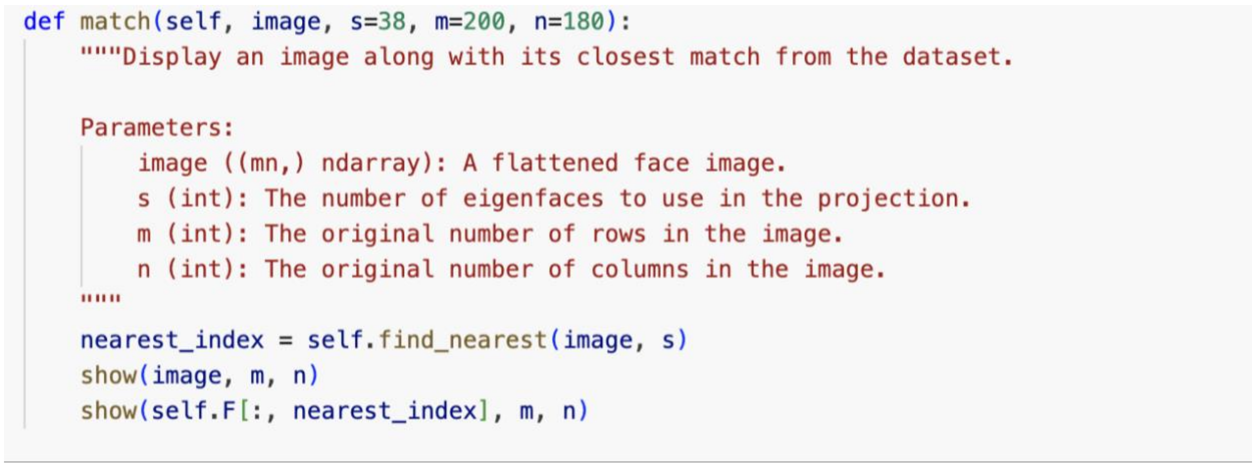
Each image was then expressed as a weighted linear combination of these eigenfaces, where each coefficient represents how much that eigenface contributes to the reconstructed image. This decomposition allows each face to be described using only a small set of numerical weights instead of all pixel values.

```python
def project(self, A, s):
    """Project a face vector onto the subspace spanned by the first s
    eigenfaces, and represent that projection in terms of those eigenfaces.

    Parameters:
        A((mn,) or (mn,l) ndarray): The array to be projected.
        s(int): the number of eigenfaces.
    Returns:
        ((s,) ndarray): An array of the projected image of s eigenfaces.
    """
    return self.U[:, :s].T @ A
```

Matching and Classification

To identify a face, a new image is projected into the eigenface space to produce a vector of weights. The Euclidean (L2) distance between this vector and all training vectors is calculated, and the smallest distance corresponds to the most likely match.

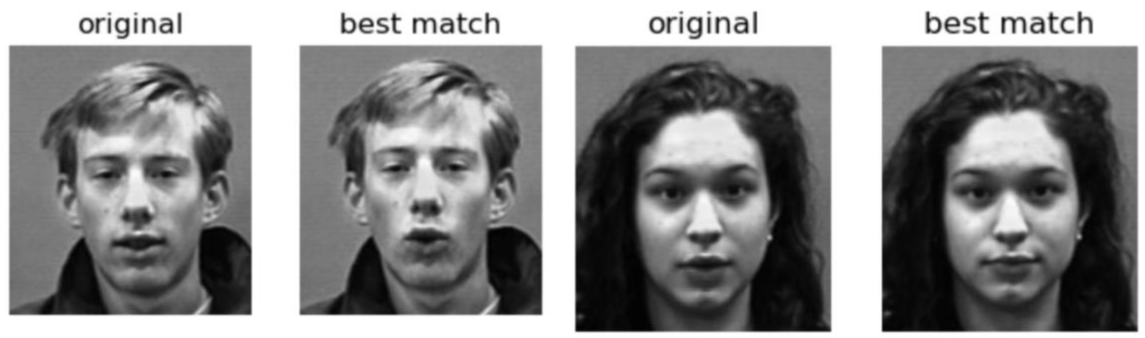
```python
def find_nearest(self, g, s=38):
    """Find the index j such that the jth column of F is the face that is
    closest to the face image 'g'.

    Parameters:
        g ((mn,) ndarray): A flattened face image.
        s (int): the number of eigenfaces to use in the projection.

    Returns:
        (int): the index of the column of F that is the best match to
                the input face image 'g'.
    """
    F_hat = self.project(self.shifted, s)
    g_hat = self.project(g - self.mean, s)
    j = np.argmin([la.norm(F_hat[:, i] - g_hat) for i in range(len(F_hat[0]))])
    return j
```



```
def match(self, image, s=38, m=200, n=180):
    """Display an image along with its closest match from the dataset.

    Parameters:
        image ((mn,) ndarray): A flattened face image.
        s (int): The number of eigenfaces to use in the projection.
        m (int): The original number of rows in the image.
        n (int): The original number of columns in the image.
    """
    nearest_index = self.find_nearest(image, s)
    show(image, m, n)
    show(self.F[:, nearest_index], m, n)
```

Results

The model successfully matched unseen photos to their correct identities in most test cases, demonstrating strong recognition accuracy under consistent lighting and pose conditions. Empirical testing showed dependable identification performance within the dataset's constraints.



Discussion and Future Improvements

While this implementation achieves strong baseline performance for similar images, real-world robustness can be improved through:

- Data augmentation, introducing lighting, pose, and expression variation
- Alignment preprocessing, such as eye and facial landmark alignment to normalize position

This project demonstrates the ability to connect theoretical mathematics and applied computer vision. It highlights strengths in algorithm design, efficient implementation, and systematic model evaluation—all core capabilities relevant to both machine learning research and multimodal AI development.

Note: This overview was written with the help of various LLMs. The images of people were screenshots from a friend's code as my images were inverted. All the code is completely my own.