Lab 11 Eigenfaces Face Reccognition

Turgunboev Dadakhon

Importing libraries

```
import os
import cv2
import numpy as np
```

Uploading and processing images

```
Loads images from a folder and converts them to vectors.
All images are converted to grayscale and reduced to the same size.

def load_images_from_folder(folder, img_size=(64, 64),
max_images=None):
    X = []
    y = []
    people = os.listdir(folder)
    for label, person in enumerate(people):
        person_folder = os.path.join(folder, person)
        if not os.path.isdir(person_folder): continue
        for file in os.listdir(person_folder):
            if file.endswith('.jpg'):
                img_path = os.path.join(person_folder, file)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
                img = cv2.resize(img, img_size)
                X.append(img.flatten())
                y.append(label)
                if max_images and len(X) >= max_images:
                    return np.array(X), np.array(y)
    return np.array(X), np.array(y)

X, y = load_images_from_folder('lfw-deepfunneled', img_size=(64, 64),
max_images=1000)
print(f"Uploaded {X.shape[0]} img  with dim {X.shape[1]}")

Uploaded 1000 img  with dim 4096
```

Implementing the mean and covariance

```
def compute_mean(X):
    N, D = X.shape
    mean = np.zeros(D)
    for i in range(N):
        mean += X[i]
    return mean / N

def compute_covariance(X, mean):
    N, D = X.shape
    cov = np.zeros((D, D))
    for i in range(N):
        diff = X[i] - mean
        cov += np.outer(diff, diff)
    return cov / N
```

Split into training and test samples

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Finding eigenvectors

```
mean_vector = compute_mean(X_train)
cov_matrix = compute_covariance(X_train, mean_vector)

eigvals, eigvecs = np.linalg.eigh(cov_matrix)
sorted_indices = np.argsort(eigvals)[::-1]
eigvals = eigvals[sorted_indices]
eigvecs = eigvecs[:, sorted_indices]
```

Projection and reconstruction

```
def project(X, mean, eigvecs, r):
    return (X - mean) @ eigvecs[:, :r]

def reconstruct(X_proj, mean, eigvecs, r):
    return X_proj @ eigvecs[:, :r].T + mean
```

Nearest neighbor recognition

```
def predict(X_train_proj, y_train, X_test_proj):
    y_pred = []
    for test_vec in X_test_proj:
        dists = [np.linalg.norm(test_vec - train_vec) for train_vec in
X_train_proj]
        y_pred.append(y_train[np.argmin(dists)])
    return np.array(y_pred)
```

Accuracy and visualization

```
import matplotlib.pyplot as plt
from IPython.display import display, Markdown
```

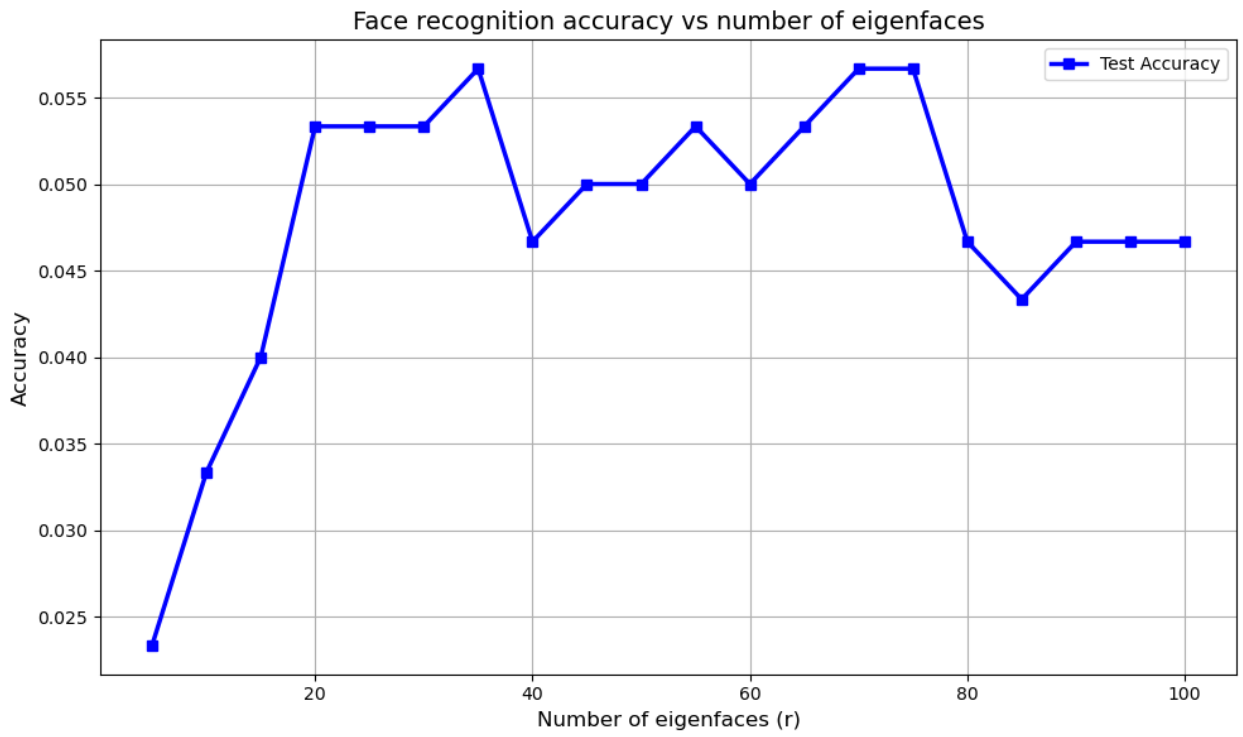
```python
plt.figure(figsize=(10, 6))

# Main accuracy plot
plt.plot(
    rs, accuracies,
    label='Test Accuracy',
    color='blue',
    linewidth=2.5,
    marker='s',
    markersize=6
)

# Axis labels and title
plt.xlabel('Number of eigenfaces (r)', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Face recognition accuracy vs number of eigenfaces',
fontsize=14)
plt.grid(True)
plt.legend()
plt.tight_layout()

# Save figure
plt.savefig('accuracy_vs_r_improved.png')
plt.show()

caption = """
**Figure:** Accuracy of face recognition as a function of the number
of eigenfaces (r).
We observe an increase in accuracy with more components up to around r
= 70, after which performance saturates or even degrades.
This suggests the presence of a low-dimensional manifold in face image
data, consistent with Lecture 11.
"""
display(Markdown(caption))
```


Face recognition accuracy vs number of eigenfaces
Test Accuracy
Accuracy
0.025
0.030
0.035
0.040
0.045
0.050
0.055
Number of eigenfaces (r)
20
40
60
80
100

```
<IPython.core.display.Markdown object>
```