# pca_on_images

August 27, 2025

## 1 PCA on Images — From Scratch

### 1.1 1. Introduction & Theory

- **Principal Component Analysis (PCA)** is a dimensionality reduction technique.
- In image processing, PCA is also called the **Karhunen–Loève Transform (KLT)**.
- Key properties:
  - Decorrelates features (covariance of transformed data is diagonal).
  - Enables **compression** by discarding components with small eigenvalues.
  - Works as a **change of basis**, but is **data-dependent**.
- If the original data is Gaussian, decorrelation implies independence of features.

## 2. Mathematical Background

```
Given \(N\) images, each reshaped into a \(D\)-dimensional vector:

- Data matrix:
\[
X \in \mathbb{R}^{N \times D}, \quad X_c = X - \mu
\]
where \(\mu\) is the mean image.

- Covariance matrix:
\[
\Sigma = \frac{1}{N-1} X_c^\top X_c
\]

- Eigen-decomposition:
\[
\Sigma w_i = \lambda_i w_i
\]

- Projection of an image \(x\):
\[
z = W^\top (x - \mu)
\]

- Approximation using top \(M\) eigenvectors:
```

```
\[
\tilde{x} = \mu + \sum_{i=1}^M z_i w_i
\]

- **Explained variance ratio**:
\[
V(M) = \frac{\sum_{i=1}^M \lambda_i}{\sum_{i=1}^D \lambda_i}
\]
```

## 1.2 3. Load and Visualize Dataset

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces, load_digits
from sklearn.utils import Bunch

def load_faces():
    try:
        data = fetch_olivetti_faces(shuffle=True, random_state=0)
        return Bunch(images=data.images, flat=data.data, target=data.target,␣
 ↪name="olivetti")
    except Exception as e:
        print("Olivetti not available, using Digits dataset:", e)
        d = load_digits()
        images = np.array([np.kron(img, np.ones((2,2))) for img in d.images])
        return Bunch(images=images, flat=images.reshape(len(images), -1),
                     target=d.target, name="digits")

faces = load_faces()
X = faces.flat
h, w = faces.images.shape[1:3]
print("Dataset:", faces.name, "Shape:", X.shape)

plt.figure()
plt.imshow(faces.images[0])
plt.title("Sample Image")
plt.axis("off")
plt.show()
```

## 1.3 4. PCA Implementation (SVD-based)

[ ]:
```python
def pca_images(X, k=None):
    # Center data
    X_mean = X.mean(axis=0)
    Xc = X - X_mean
    n_samples = X.shape[0]
    # Compute PCA via SVD (numerically stable)
    U, S, VT = np.linalg.svd(Xc, full_matrices=False)
    components = VT            # rows are principal directions
    explained_variance = (S**2) / (n_samples - 1)
    explained_ratio = explained_variance / explained_variance.sum()

    if k is not None:
        comps = components[:k]
        proj = Xc @ comps.T
        rec = proj @ comps + X_mean
        return comps, explained_ratio, proj, rec, X_mean
    return components, explained_ratio, X_mean
```

### 1.4 5. Mean Image and First 10 Eigenfaces

[ ]:
```
components, explained_ratio, X_mean = pca_images(X)
mean_img = X_mean.reshape(h, w)

plt.figure()
plt.imshow(mean_img)
plt.title("Mean Image")
plt.axis("off")
plt.show()

# Show first 10 eigenfaces (each in its own figure)
num_show = 10
for i in range(num_show):
    plt.figure()
    plt.imshow(components[i].reshape(h, w))
    plt.title(f"Eigenface {i+1}")
    plt.axis("off")
    plt.show()
```

### 1.5 6. Image Reconstruction with Different Numbers of Components

[ ]:
```
import numpy as np

np.random.seed(0)
sample_idx = np.random.choice(X.shape[0], size=4, replace=False)
ks = [5, 15, 50, 100]

for k in ks:
    comps, ratio, proj, rec, X_mean = pca_images(X, k=k)
    for idx in sample_idx:
        plt.figure()
        plt.imshow(rec[idx].reshape(h, w))
        plt.title(f"Reconstruction (k={k}) - sample {idx}")
        plt.axis("off")
        plt.show()
```

## 1.6 7. Explained Variance Curve

[ ]:
```
_, explained_ratio, _ = pca_images(X)
plt.figure()
plt.plot(np.cumsum(explained_ratio))
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Variance Retention")
plt.grid(True)
plt.show()
```

## 1.7 8. Applications & Notes

- **Compression**: Store only top (k) coefficients per image (plus the basis and mean).
- **Recognition**: Use PCA projections as features (e.g., eigenfaces for face recognition).
- **Denoising**: Truncating small components often removes noise-like details.
- **High-dimensional trick**: When (N  D), avoid forming a $(D \times D)$ covariance matrix; use SVD on centered data.