Faces in Subspaces

Face Recognition Using Principal Component Analysis

A greyscale image of a person's face can be represented as a matrix $\mathbf{P} \in \mathbb{R}^{h \times w}$, where $\mathbf{P}_{ij}$ denotes the intensity of the pixel at position (i, j). Typically $\mathbf{P}_{ij}$ is an integer in the range $0 - 255$ or is normalized to lie in $[0, 1]$.

The goal of this project is to use the information contained in these matrices to implement a face recognition algorithm inspired by the Eigenfaces approach [1] which leverages PCA to extract the most informative features from the dataset.

Dataset construction

Construct a training set by capturing at least 15 images each of yourself and at least two colleagues. To ensure that PCA extracts meaningful facial features, maintain consistent lighting and eye alignment across all captures. Additionally, prepare an independent test set of $5 - 10$ images per person in order to be able to evaluate your model's performance. Make sure all the images are grayscale and of appropriate size $h \times w$, e.g. 112×92 (this can be implemented as a separate preprocessing step but make sure it's consistent).

Building the subspace

We can use the training set to build the PCA subspace. Flatten each grayscale image in the training set into a vector $\boldsymbol{x}_i^{(k)} \in \mathbb{R}^{h \cdot w}$ by stacking its columns (this vector corresponds to the image i of the person k). We can compute the principal components (PCs) using the data covariance matrix $\mathbf{\Sigma} = \frac{1}{N}\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^\top$, where $\widetilde{\mathbf{X}}$ is the centered data matrix obtained by subtracting the mean face from each image vector. Here N is the total number of training examples. **In which cases can this be problematic? Why (consider the dimensions and the eigenvalues of $\mathbf{\Sigma}$)?**.

It turns out we can avoid working with $\mathbf{\Sigma}$ by considering the inner product matrix $\mathbf{\Sigma}' = \frac{1}{N}\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}}$ (Dual PCA). **How can we use the eigenvectors of $\mathbf{\Sigma}'$ to recover the principal components of $\mathbf{\Sigma}$? Interpret (and visualize) the computed PCs. What consequences does changing a single component of the embedding have in the original data space?**

Provide an explanation and visualization of the (relative) importance of the computed PCs and comment on how we can determine how many to keep in order to obtain good reconstruction quality. Use them to create a low-dimensional visualization of the training data.

Select a random image from the test set and add some i.i.d. Gaussian noise $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$ to each pixel. Then project the noisy image onto the PCA subspace spanned by a sufficient number of principal components and reconstruct it and compare it with the original clean image. Report your observations.

Classification

Once we have built the PCA subspace of an appropriate dimension, we can represent each image as a lower-dimensional embedding given by its coordinates in the PCA basis.

A simple approach to classify new examples is to use a nearest-neighbour classifier (1-NN). For a new, unseen image, project it onto the PCA subspace and compute its distance to all embeddings of the training images. Assign the image to the class of the closest training embedding. To measure distance between embeddings you can use the Euclidean distance. You can also experiment with other distance metrics (e.g. Mahalanobis distance, which has a nice form in PCA space).

You may also experiment with k-nearest neighbours for small values of k in order to smooth out the decision boundary. **What values of k make sense**? You are also encouraged to experiment with other classifiers like SVMs and NNs on the PCA embeddings.

Use the independent test set to evaluate the performance of your classifier for different numbers of principal components (e.g. using classification accuracy). When does the performance begin to plateau? How would you determine the number of principal components needed to achieve good results on unseen data (could another independent dataset alongside the training and test set be beneficial)?

Extra credit: demonstration of your system

Once you have completed the tasks related to PCA and human face recognition, you are encouraged to prepare a short demo, to demonstrate the performance of your system. Using your webcam, you can capture live images and localize faces (e.g. using OpenCV's [2] CascadeClassifier which is based on the Viola Jones detector [3] and is very simple to use. The complete pipeline is explained in [4]. OpenCV is often used alongside C++ and Python, however, Julia bindings exist as well [5]).

After obtaining locations of faces, you can resize them to the appropriate size and run your algorithm on them. Display the results by rendering bounding boxes with classification labels around the detected faces.

NOTE: It is recommended that you prepare a short video taken in similar conditions as the images in the training set since PCA can be very sensitive to changes in lighting.

References

[1] Matthew Turk and Alex Pentland. "Eigenfaces for Recognition". In: *Journal of Cognitive Neuroscience* 3.1 (Jan. 1991), pp. 71–86. ISSN: 0898-929X. DOI: 10.1162/jocn.1991.3.1.71. eprint: https://direct.mit.edu/jocn/article-pdf/3/1/71/1932018/jocn.1991.3.1.71.pdf. URL: https://doi.org/10.1162/jocn.1991.3.1.71.

[2] OpenCV. *Open Source Computer Vision Library*. https://opencv.org/. Accessed March 10, 2026. 2026.

[3] Paul A. Viola and Michael J. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA*. IEEE Computer Society, 2001, I:511–518. DOI: 10.1109/CVPR.2001.990517. URL: https://doi.org/10.1109/CVPR.2001.990517.

[4] OpenCV. *Cascade Classifier*. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. OpenCV 3.4 Documentation, accessed March 10, 2026. 2026.

[5] OpenCV.jl. *OpenCV.jl*. https://juliaimages.org/OpenCV.jl/dev/. Accessed March 10, 2026. 2026.