Project I: PCA, Autoencoder and FLD for Analyzing Huamn Faces

1 Objectives

Human face is a very important pattern and has been extensively studied in the past 30 years in vision, graphics, and human computer interaction, for tasks like face recognition, human identification, expression, animation etc. An important step for these tasks is to extract effective representations from the face images. The principal component analysis is found to be a good representation. This project will compare three types of representations in the context of dimension reduction: i) Two generative methods — PCA (Linear) and Autoencoder (non-linear); and ii) a discriminative method Fisher Linear Discriminants (FLD).

Dataset We provide a dataset of 1,000 faces (`./images`) and each has 128 x 128 pixels. The face images are pre-processed so that the background and hair are removed. Each face has a number of landmarks, 68 points per image, which are identified by OpenFace (`./landmarks`). These landmarks should be aligned so that the appearance (i.e. grey level, intensity on pixels) can be compared meaningfully across face. Such alignment (geometry for image or time warping for speech) is quite common in other patterns/data. To study the discriminative method, we divide the dataset into two categories: male (`./male_images`) and female (`./female_images`), with the corresponding landmarks (`./male_landmarks` and `./female_landmarks`).

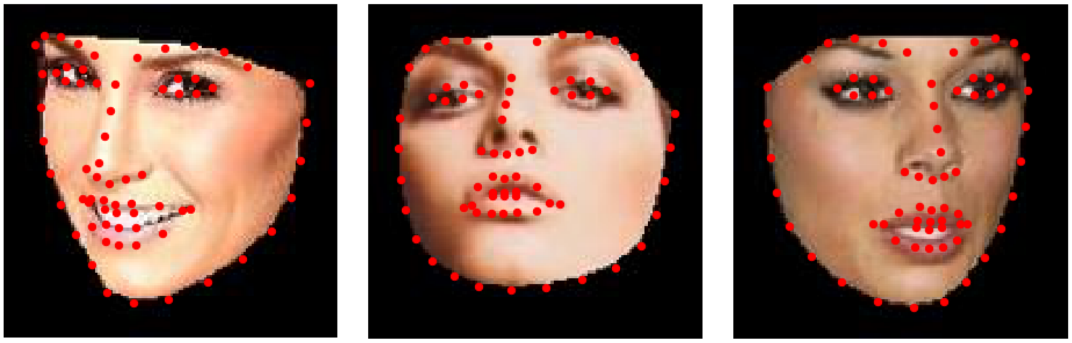

Figure 1: Example faces with 68 landmarks from CelebA. The data set contains 1000 images from CelebA, and they are cropped to 128 × 128 pixels by the OpenFace. These faces have different colors, illuminations, identities, viewing angles, shapes, and expressions.

2 Part 1: ASM and AAM model for face recognition

The experiment includes the following steps.

2.1 PCA: a linear method

Divide the 1000 faces into two parts: the first 800 faces form a training set, and the remaining 200 faces form the test set. In the following steps, you always use the training set to calculate the eigen-vectors and eigen-values, and calculate the reconstruction errors for the test images using the eigen-vectors computed from the training set.

1. Compute the mean and first $K = 50$ eigen-faces for the training images with no landmark alignment, and use them to reconstruct the remaining 200 test faces. i) **Display** the first 10 eigen-faces; ii) **Plot** 10 reconstructed faces and the corresponding original faces; and iii) **Plot** the total reconstruction error (squared intensity difference between the reconstructed images and their original ones) per pixel (i.e. normalize the error by the pixel number, and average over the testing images) over the number of eigen-faces $K = 1, 5, 10, 15, ..., 50$.

   Note that, PCA is usually applied on the intensity or gray images. For the color images, you can first transform the color image from the RGB color model to the HSV (hue, saturation, value) model [1]. Then apply PCA on the V channel. In python, you can use `skimage.color.rgb2hsv` and `skimage.color.hsv2rgb` to transform the images between RGB and HSV.

2. Compute the mean and first K=50 eigen-warping of the landmarks for the training faces. Here warping means displacement of points on the face images. i) **Display** the first 10 eigen-warpings (you need to add the mean to make it meaningful), and use them to reconstruct the landmarks for the test faces. ii) **Plot** the reconstruction error (in terms of distance) over the number of eigen-warpings $K = 1, 5, 10, 15, ..., 50$ (again, the error is averaged over all the testing images).

3. Combine the two steps above. Our objective is to reconstruct images based on top 10 eigen-vectors for the warping and then top K (say 50) eigen-vectors for the appearance. For the training images, we first align the images by warping their landmarks into the mean position, and then compute the eigen-faces (appearance) from these aligned images. For each testing face: i) project its landmarks to the top 10 eigen-warpings, you get the reconstructed landmarks. (here you lose a bit of geometric precision of reconstruction). ii) Warp the face image to the mean position and then project to the top k (say K=50) eigen-faces, you get the reconstructed image at mean position (here you further lose a bit of appearance accuracy). iii) Warp the reconstructed faces in step ii) to the positions reconstructed in step i). Note that this new image is constructed from 60 numbers. Then compare the reconstructed faces against the original testing images (here you have loss in both geometry and appearance). **Plot** 20 reconstructed faces and their corresponding original faces. **Plot** the reconstruction errors per pixel against the number of eigen-faces $K = 1, 5, 10, 15, ..., 50$.

   `mywarper.py` provides a function `warp(image, original_landmark, warped_landmark)` for you to use.

4. Synthesize random faces by random sampling of the landmarks (based on the top 10 eigen-values and eigen-vectors in the wrapping analysis) and a random sampling of the appearance (based on the top 50 eigen-values and eigen-vectors in the intensity analysis). **Display** 50 synthesized face images. As we discussed in class, each axis (i.e. the i-th eigen-vector) has its own unit, that is $\sqrt{\lambda_i}$, the square-root of the i-th eigen-value. Specifically, the eigen-faces and eigen-warpings for the landmarks can be seen as the basis functions for the generated faces, while we need to sample the coefficients (latent vector) at each eigen-axis.

[1] `https://en.wikipedia.org/wiki/HSL_and_HSV`

2.2 Autoencoder: a non-linear method

PCA reconstructs an image (or other type of data) linearly with orthonormal basis (eigen-vectors) learned from the covariance matrix of the training data. Auto-encoder is a non-linear extension of PCA, which extracts the feature and reconstructs the data by a deep neural network. An autoencoder[2] learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data. This forces the auto-encoder to engage in dimensionality reduction. Figure 2 shows an illustration of an auto-encoder with convolutional structures.

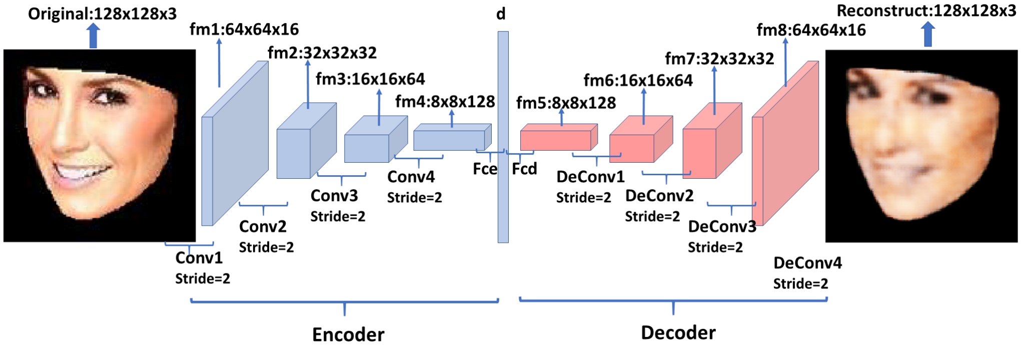

Figure 2: Illustration of Auto-encoder with convolutional architectures. An auto-encoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.

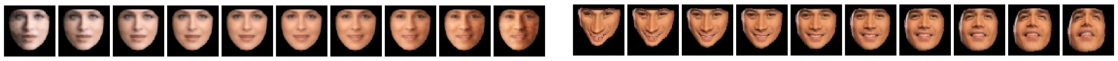

Figure 3: Left: appearance interpolation by auto-encoder. Right: generated faces with geometric variances by auto-encoder.

Tables 1 and 2 give suggested network structures for the auto-encoders of appearance and landmarks. You are welcomed to try any structure you want. Please make sure the dimensions of latent vectors are 50 for appearance and 10 for landmarks, for fair comparison with PCA. Suggested training parameters: epoch 300, batch size 100, AdamOptimizer, learning rate 7e-4. You can use either PyTorch or Tensorflow.

Table 1: Suggested structure for appearance auto-encoder.

	Encoder	Decoder
0	Conv-(Channel 16, Kernel 5, Stride 2), LeakyReLU	Deconv-(Channel 128, Kernel 8, Stride 1), LeakyReLU
1	Conv-(Channel 32, Kernel 3, Stride 2), LeakyReLU	Deconv-(Channel 64, Kernel 3, Stride 2), LeakyReLU
2	Conv-(Channel 64, Kernel 3, Stride 2), LeakyReLU	Deconv-(Channel 32, Kernel 3, Stride 2), LeakyReLU
3	Conv-(Channel 128, Kernel 3, Stride 2), LeakyReLU	Deconv-(Channel 16, Kernel 3, Stride 2), LeakyReLU
4	Fc-(Channel 50), LeakyReLU	Deconv-(Channel 3, Kernel 5, Stride 2), Sigmoid

[2] https://en.wikipedia.org/wiki/Autoencoder

Table 2: Suggested structure for landmark auto-encoder.

	Encoder	Decoder
0	Fc-(Channel 100), LeakyReLU	Fc-(Channel 100), LeakyReLU
1	Fc-(Channel 10), LeakyReLU	Fc-(Channel 68*2), Sigmoid

1. Please re-perform the **experiments (3)** in Section 2.1 by replacing PCA with auto-encoder. Specifically, the landmarks can be reconstructed and generated by the auto-encoder with fully-connected architecture, while the two-dimensional face images can be reconstructed and generated by the auto-encoder with convolutional architecture. It is worth to note that the extracted features or each dimension of the latent vector of the auto-encoder is not orthonormal as PCA.

2. Interpolation: choose the first 4 dimensions of the latent variables of appearance that have the maximal variance, and **show** the interpolation results on each dimension while keeping the other dimensions fixed. Choose the first 2 dimensions of the latent variables of landmarks that have the maximal variance, and get the interpolation results on each dimension while keeping the other dimensions fixed. **Show** the results by warping a chosen face by the generated landmarks. Figure 3 shows an interpolation result for appearance and landmark respectively.

3 Part 2: Fisher faces for gender discrimination.

We have divided the 1000 faces into male (412) and female faces (588). Then you can randomly choose 800 faces as the training set and the remaining 200 faces as testing set.

(1) Find the Fisher face that distinguishes male from female using the training sets, and then test it on the 200 testing faces and **report** the error rate. This Fisher face mixes both geometry and appearance difference between male and female.

(2) Compute the Fisher face for the key point (geometric shape) and Fisher face for the appearance (after aligning them to the mean position) respectively. Project all the faces to the 2D-feature space learned by the fisher-faces, and **visualize** how separable these points are.

[The within-class scatter matrix is very high dimensional, you may compute the Fisher faces over the reduced dimensions in steps (2) and (3) in section 2.1: i.e. each face is now reduced to 10 dimensional geometric vector + 50 dimensional appearance vector. After the Fisher linear Discriminant analysis, we represent each face by as few as 2 dimensions for discriminative purpose and yet, it can tell apart male from female!]