Programming Assignment 3

CS 479

https://github.com/alexander-novo/CS479-PA3

Nikhil Khedekar	Mehar Mangat	Alexander Novotny
33%	33%	33%
Section 1	Section 1	Sections 2 and 3

Due: April 28, 2021
Submitted: April 27, 2021

Contents

1 Theory

This assignment explores the usage of Principal Component Analysis for dimensionality reduction in order to perform classifications on a dataset of provided images. The images were of faces from the FERET face database [6], and have a good variety of faces from different genders, ethnicity, age groups, lighting conditions, etc. The classification task to be performed was facial recognition

1.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a method for reducing the dimensionality of high dimensional data while preserving as much information as possible. The way this works is by the projection of the sample data along the principal distribution directions. Intuitively, this can be understood by finding the directions that capture the highest variance of the data and projecting the data along these directions.

For some given sample data $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^N, 1 \leq i \leq M\}$, where M is the number of samples in the data set, the sample covariance matrix $\mathbf{\Sigma}_x$ can be found using eq. (1) and eq. (3), where $\bar{\mathbf{x}}$ is the sample mean and Φ_i are the normalized data points:

$$\bar{\mathbf{x}} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{x}_i \tag{1}$$

$$\mathbf{\Phi_i} = \mathbf{x}_i - \bar{\mathbf{x}} \tag{2}$$

$$\begin{aligned}\mathbf{\Sigma_x} &= \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})^\top \\ &= \frac{1}{M}\sum_{i=1}^{M}\mathbf{\Phi}_i\mathbf{\Phi}_i^\top \end{aligned} \tag{3}$$

Equation (3) can be further simplified to eq. (4) using eq. (5).

$$\mathbf{\Sigma_x} = \frac{1}{M}\mathbf{A}\mathbf{A}^\top, \tag{4}$$

where

$$\mathbf{A} = [\Phi_1\Phi_2...\Phi_M] \tag{5}$$

The eigenvectors and eigenvalues of the covariance matrix $\mathbf{\Sigma}_x$ have the special property that the eigenvectors form an orthogonal basis in $\mathbb{R}^N$ for the sample data with the corresponding eigenvalues representing the variance of the data in these directions. The eigenvectors $\mathbf{u}_i$ and corresponding eigenvalues λ_i are obtained using eq. (6).

$$\mathbf{\Sigma_x u_i} = \lambda_i \mathbf{u}_i \tag{6}$$

However, in image recognition this approach becomes problematic. Performing PCA directly on the image's covariance matrix can be computationally intractable. Even considering small images of 50×50 pixels, the covariance matrix would contain 2500 elements!

Fortunately though, there is a workaround. The rank of the covariance matrix is limited by the number of images in our data-set. So, if the number of images in our data-set is smaller than the number of dimensions of the image, then we can perform the following calculation:

Let $\mathbf{A}$ be a matrix of the images, where each column contains a mean-subtracted image from the data-set. Then the eigenvector decomposition of the covariance matrix $\mathbf{\Sigma}_x$ can be computed as:

$$\mathbf{\Sigma}_x \mathbf{u}_i = \mathbf{A}\mathbf{A}^\top \mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{7}$$

However, we can multiply $\mathbf{A}$ to eq. (7), to get:

$$\mathbf{A}^\top \mathbf{A}\mathbf{A}^\top \mathbf{u}_i = \lambda_i \mathbf{A}^\top \mathbf{u}_i \tag{8}$$

Using the definition in eq. (9) in eq. (8), we can see that $\mathbf{v}_i$ represents the eigenvector for $\mathbf{A}^\top \mathbf{A}$.

$$\mathbf{v}_i = \mathbf{A}^\top \mathbf{u}_i \tag{9}$$

This drastically reduces the necessary size of the covariance matrix. If we had 75 images that were 50×50 before, then the matrix $\mathbf{A}^\top \mathbf{A}$ is 75×75, which is much more reasonable to calculate than a 2500×2500 covariance matrix.

1.2 Mahalanobis Distance

The Mahalanobis Distance e_k between two vectors, $\mathbf{x}$ and $\mathbf{y}$, projected onto the span of the chosen eigenvectors is given by eq. (10).

$$e_k^2 = \sum_{j=1}^{K} \frac{1}{\lambda_j}(x_j - y_j) \tag{10}$$

The advantage Mahalanobis distance has over Euclidean is two-fold in this experiment; Mahalanobis distance accounts for the correlations between variables, and is also scale invariant. The scale in-variance is crucial as it saves us a step in normalization that Euclidean distance would otherwise incorrectly capture.

1.3 Application

In our notation, we assume that the eigenvalues are indexed in descending order, namely, $\lambda_1 > \lambda_2 > ... > \lambda_N$. Using the eigenvectors as the new basis for our data we can represent our data samples $\mathbf{x} \in \mathbb{R}^N$ by eq. (11)

$$\mathbf{x} - \bar{\mathbf{x}} = \sum_{i=1}^{N} y_i \mathbf{u}_i = y_1 \mathbf{u}_1 + y_2 \mathbf{u}_2 + ... y_N \mathbf{u}_N \tag{11}$$

where $y_1, y_2 ... y_N$ are the coefficients in this new basis and are given by eq. (12).

$$y_i = \frac{(\mathbf{x} - \bar{\mathbf{x}})^\top \mathbf{u}_i}{\mathbf{u}_i \mathbf{u}_i^\top} = (\mathbf{x} - \bar{\mathbf{x}})^\top \hat{\mathbf{u}}_i \tag{12}$$

where $\hat{\mathbf{u}}_i$ represents the unit vector in the direction of $\mathbf{u}_i$. The final dimensionality reduction is carried out on the basis of eq. (11) in that only the components corresponding to the first $K(K << N)$ eigenvectors are kept and the remaining components are discarded. This formulation approximates $\mathbf{x}$ by $\hat{\mathbf{x}}$ for which eq. (11) can be rewritten as eq. (13).

$$\hat{\mathbf{x}} - \bar{\mathbf{x}} = \sum_{i=1}^{N} y_i \mathbf{u}_i = y_1 \mathbf{u}_1 + y_2 \mathbf{u}_2 + ... y_K \mathbf{u}_K \tag{13}$$

eq. (13) can be written in matrix form instead of the summation as eq. (14)

$$\hat{\mathbf{x}} - \bar{\mathbf{x}} = \mathbf{U} \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_K \end{bmatrix} \tag{14}$$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_K]$ is an $N \times K$ matrix with its columns as the K largest eigenvectors of the covariance matrix $\mathbf{\Sigma}_x$. Equation (14) can also be written as

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_K \end{bmatrix} = \mathbf{U}^\top (\hat{\mathbf{x}} - \bar{\mathbf{x}}) \tag{15}$$

The choice of K is made on the basis of the amount of information that we wish to preserve. For a numerical representation of the information preserved, we can specify a threshold t for the ratio of the summations of the first K eigenvalues to that of all N eigenvalues as eq. (16). A threshold of 1 would mean perfect information retention (and consequently, no reduction in dimensionality).

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > t \tag{16}$$

Finding these components y_i have several applications, one of which is the subject of this assignment. These components can be used to simplify the problem of comparing high dimensional data as is discussed in the next section.

1.4 Eigenfaces and Facial Recognition

The first widely successful method of solving the facial recognition problem is the eigenface method [7]. The core of this method relies on the fact that image data can be stacked to form vectors in a dataset after which we can use PCA to reduce the dimensionality of the data and find the components y_i described in the previous section. Similarity in the components implies a similarity in the data provided that the amount of information that is kept (decided using the threshold t) is enough for doing so. As the eigenvectors obtained by this method are essentially images themselves and given that the method was designed for recognizing images of faces, they are termed as eigenfaces. In dealing with images of faces, the mean vector $\bar{\mathbf{x}}$ is also now termed as the mean face. Hence, eq. (11) for image data would semantically represent the difference between a sample image and the mean face being given by a summation of the eigenfaces scaled by their respective components.

Facial recognition intends to solve the problem of finding an image in a given dataset that is visually the most similar to a test image. The comparison of images is a computationally heavy task as it has the complexity of $\mathcal{O}(rc)$ where r, c are the rows and columns of the image respectively. The complexity of this task can be reduced to $\mathcal{O}(K)$ by finding the eigenfaces and scalar components that represent each image and reducing it to K components. This is essentially an application of PCA.

2 Implementation

Similarly to previous assignments, the Eigen [2] C++ template library was used to handle matrix operations. The C++ standard `filesystem` library [1] new to C++17 was also used for reading in the contents of an entire directory for reading in the data set. Many trivially parallelizable sections were parallelized using OpenMP, and all of Eigen's trivially parallelizable algorithms are already parallelized using OpenMP.

The training portion of the program begins by reading all of the training images into a large $M \times N$ Eigen matrix, where M is the number of pixels in each image and N is the number of images. We load the directory contents and the header of a single image to determine the size of the matrix before reading in the images so that reading may be done in parallel. Then, the mean is calculated using a standard parallel sum. Eigen helpfully includes a "broadcasting" module [4] which lets us apply a vector operation to an entire matrix in a similar way to "broadcasting" that vector as a full matrix but without allocating an entire matrix. This is used to calculate $\mathbf{A}$ in eq. (5) by broadcasting the operation of subtracting the mean from the matrix of images. Then, Eigen's `SelfAdjointEigenSolver` [5] library is used to efficiently compute the eigenvalues and eigenvectors of $\mathbf{A}^\top\mathbf{A}$ as discussed in section 1.1, which are already normalized and sorted in ascending order.

According to Eigen's documentation, taking advantage of the symmetric nature of $\mathbf{A}^\top\mathbf{A}$ gives us a speed increase of about 3 times that of a normal eigensolver. Additionally, as discussed in section 1, $\mathbf{A}^\top\mathbf{A}$ may be smaller than $\mathbf{A}\mathbf{A}^\top$ - and in the case of the high resolution images, there are about 1000 images with about 2000 pixels each, so due to the cubic nature of the eigensolver algorithm, the use of the smaller $\mathbf{A}^\top\mathbf{A}$ gives us another speed increase of about 8 times. Multiplying by $\mathbf{A}$ to obtain eigenvectors of $\mathbf{A}\mathbf{A}^\top$ may denormalize the eigenvectors, so Eigen's broadcasting module is used again to normalize them (obtaining $\mathbf{U}$) and project the images onto the calculated eigenvectors by multiplying by $\mathbf{A}$ as in eq. (15) (knowing that the columns of $\mathbf{A}$ are the mean subtracted from each image). The mean image, eigenvalues, projected images, and their labels are then written to a training data file to be read by the second portion of the program.

The testing portion of the program begins by reading all of the testing images into a matrix in a similar fashion to the training portion as well as the training data generated by said portion. Then, K is calculated as in eq. (16) based on the inputted t. The test images are projected in the same way as the training images were, however this time only using the right K columns of $\mathbf{U}$ to project (using Eigen's block operations module [3]). Then the Mahalanobis distance is calculated between each projected testing and training images (taking care to only use the bottom K rows of the projected training image matrix with Eigen's block module). Each projected testing image uses a max heap to keep the running minimum 50 distances as it calculates every distance. Then this heap is sorted and checked for correct matches to classify. These distances and heap calculations are trivially parallelizable for each projected testing image, so OpenMP is used to do it quickly.

3 Results and Discussion

The calculated sample mean images for both data set can be found in fig. 1. Note the appearance of a single pixel-wide border of dark pixels around both means - this seems to come from some sort of artifact in the original data sets consisting of slightly darker pixels on the border.

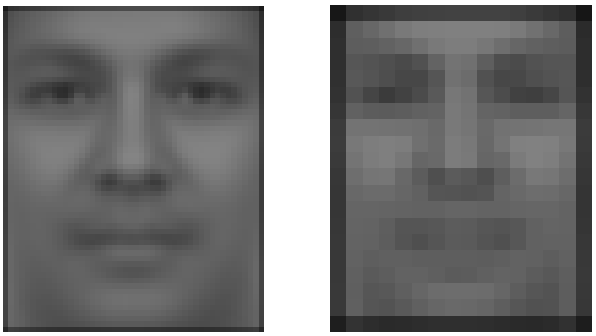

Figure 1: The sample mean face for the high resolution data set (left) and the low resolution data set (right).

The 10 largest and smallest eigenfaces can be found in figs. 2 and 3. Note that the largest eigenfaces all look like faces in some way, while the smallest eigenfaces might as well be random noise. This is a good justification for dimensionality reduction using PCA - these smallest eigenfaces are contributing very little meaningful data. Also note that each eigenface calculated from the low resolution data corresponds to one calculated from the high resolution data almost exactly. This makes sense, since the low resolution images are just sampled version of the high resolution images, so it would make sense that the principal components are just sampled version of the principle components of the high resolution images. It also implies that principle components persist through downsampling. Note that eigenfaces 7 and 8 correspond but are inverted between data sets - this is an example of how inverted eigenvectors are still eigenvectors. Finally, note how many of the eigenfaces look seemingly male - this is probably due to the over-representation of male-presenting subjects in the data set.

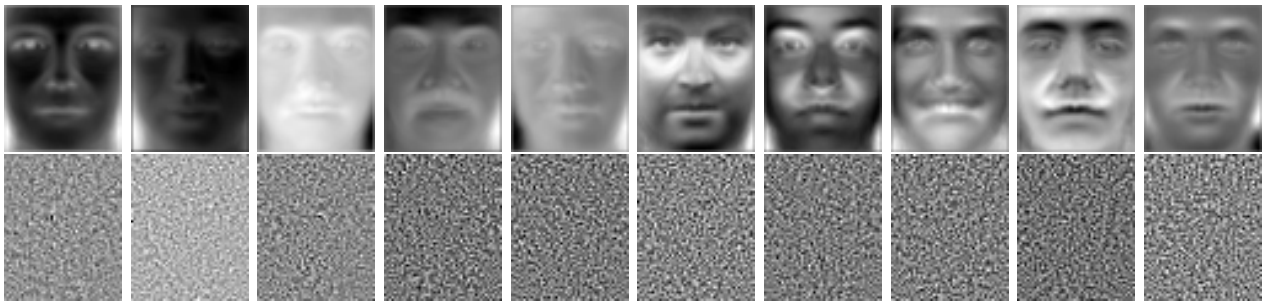

Figure 2: The 10 largest eigenfaces in decreasing order for the high resolution data set, followed by the 10 smallest eigenfaces in increasing order.

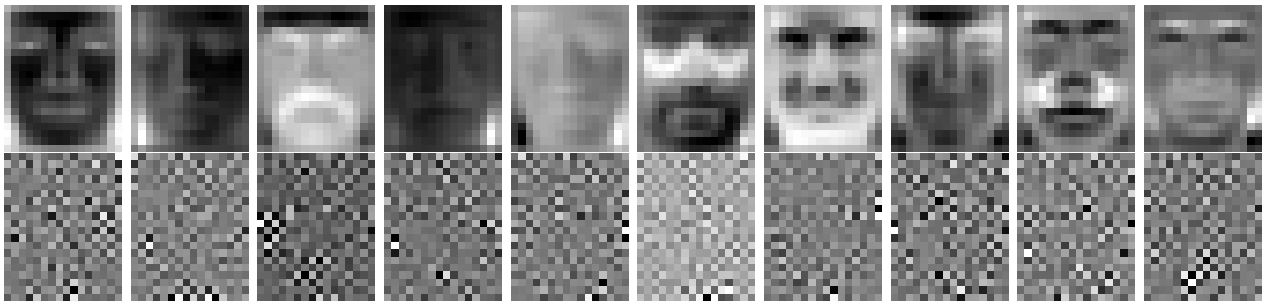

Figure 3: The same figure as fig. 2, but for the low resolution data set.

A comparison of classification accuracies between data sets can be found in fig. 4. There are a couple of interesting points to be found in the high resolution data set: while the 95% information classifier starts as the most performant classifier (for $N = 1$), it is quickly surpassed by the other classifiers, becoming the least performing classifier at $N = 50$. Also, while the 80% classifier starts as the least performant classifier, it is at times the most performant and eventually always at least matches the performance of the other classifiers. The low resolution classifiers, however, do not demonstrate this quality - the lower information classifiers remain lower performing. This face, along with the fact that the most performant low resolution classifiers at either end of the spectrum outperform the high resolution classifiers leads us to believe that the dimensionality of the data is still too large after performing PCA on the high resolution images, while the low resolution images benefit from low dimensionality data from the beginning. Despite this, the plots suggest that lowering the amount of information preserved by PCA for the high resolution images would only decrease the performance for $N = 1$ (even if it could recover performance for larger N), supporting our theory from above that downsampling, at times, may outperform PCA. A final interesting point is while there seems to be little improvement to be made on these classifiers for large N, there may be some room for improvement for $N = 1$. Note that the gaps between accuracies at $N = 1$ shrink as we increase the amount of data, and in the high resolution data set, the gap between 90% and 95% information is almost nonexistent, implying that preserving more information will not improve the accuracy. Meanwhile, the gap between 90% and 95% information in the low resolution data set is still quite large, implying there is some room for improvement by preserving more information - even though it already performs better than the high resolution data set at only 95%.

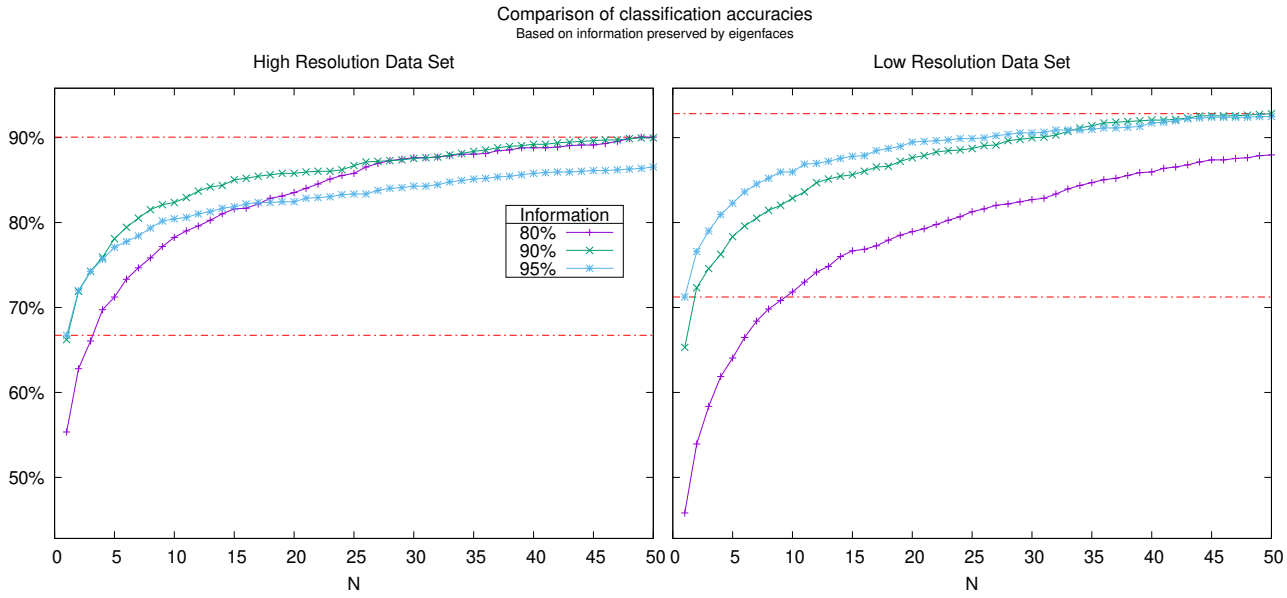

Figure 4: A comparison of classification accuracy against number of images considered (N), the percentage of information preserved by eigenfaces, and the data set used. The most performant classifier at each end of the spectrum of N is marked in red for each data set.

A selection of faces matched with the faces they were classified as are available in figs. 5 and 6. Something interesting to note is that these selections of images were chosen arbitrarily, and yet two of the three incorrectly classified images seem to be of either female or female-presenting subjects, and the images they were matched against depict similarly female or female-presenting subjects - all despite female-presenting subjects making up a relatively smaller number of subjects in the data set than male-presenting subjects. This is probably the reason - as explained above, PCA picked the principle components of male faces due to their over-representation, and now struggles to classify female faces which probably project weaker onto the span of male principle components. However, since all images of female-presenting subjects project weaker onto this span, they are relatively closer together than all of the images depicting male-presenting subjects, so PCA is still able to tell that they depict female-presenting subjects. This partially persists through to the low resolution images, where this time only one of the incorrectly classified images is female (incorrect image number 2). However, it is still matched to the downsampled version of the same image its high resolution version was matched to. This suggests that the difference between male-presenting and female-presenting face largely persist through downsampling, but some features become unrecognizable.

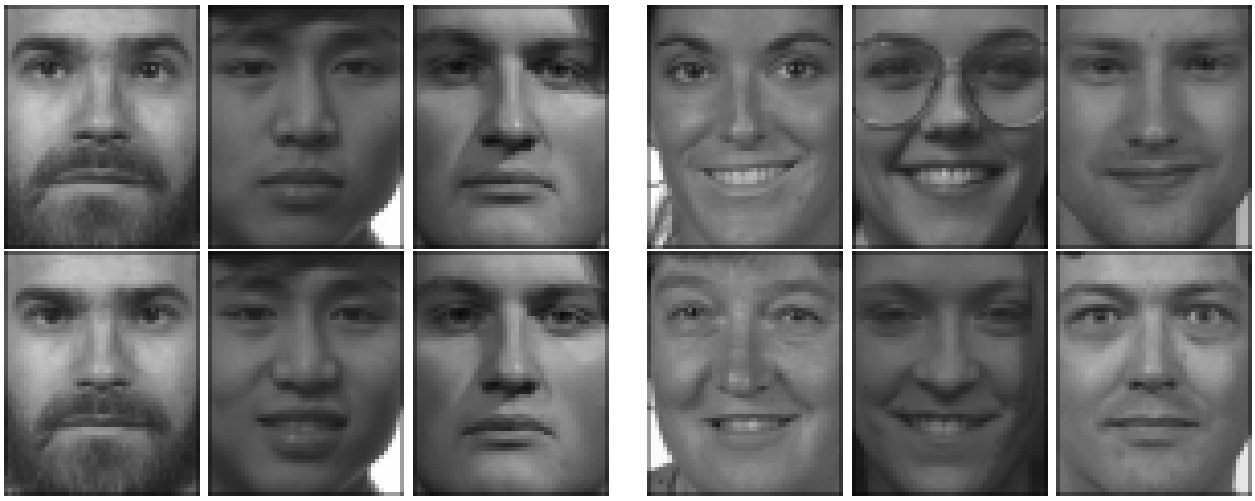

Figure 5: High resolution testing images (top) and the training images they were matched against (bottom). Images which were correctly classified (i.e. both pictures are of the same subject) on the left, and incorrect classifications on the right.

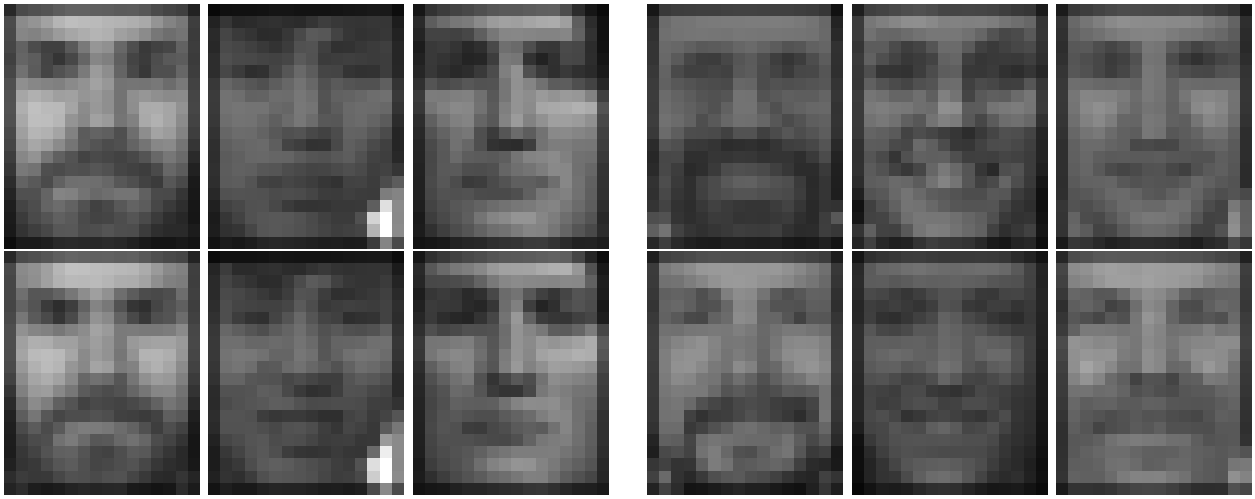

Figure 6: The same figure as fig. 5, but with images from the low resolution data set.

The ROC curve depicted in fig. 7 allows us to compare the high/low resolution classifiers when applied to intruder detection. It seems that, like above, the low resolution classifier outperforms the high resolution classifier (as it has reduced area above the curve).

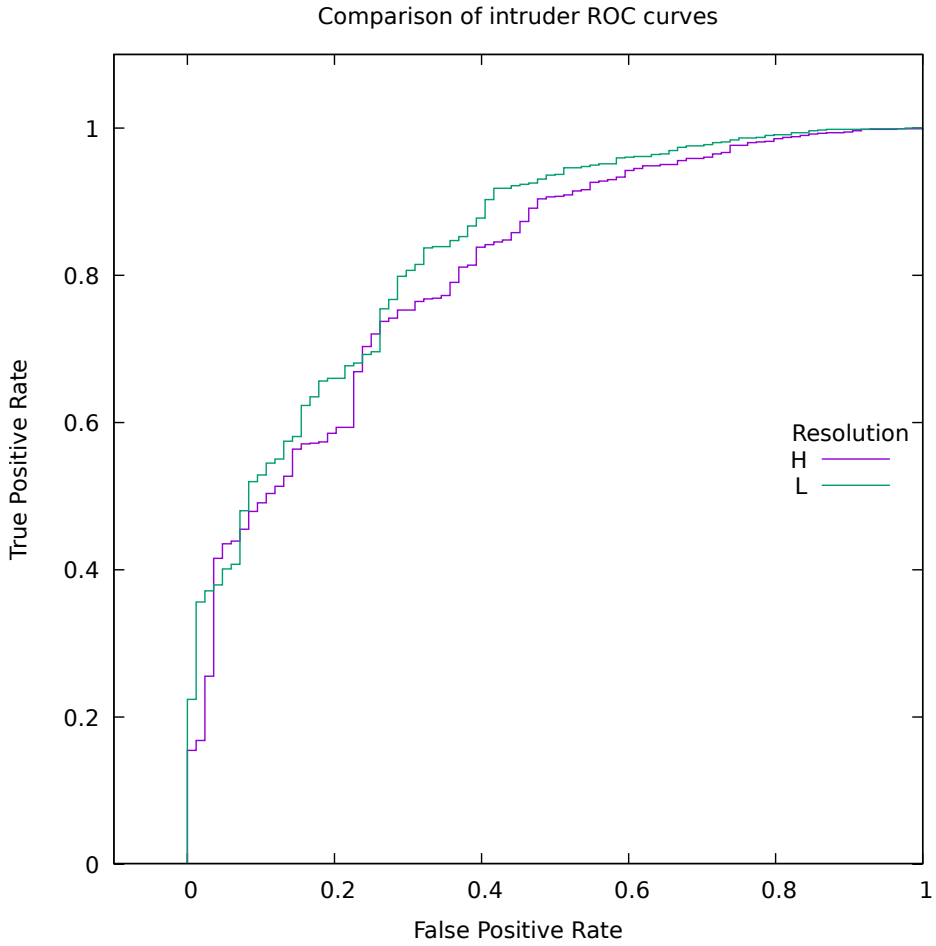

Figure 7: A comparison of intruder-detection ROC curves based on resolution.

References

[1] *C++17 `filesystem` library*. URL: `https://en.cppreference.com/w/cpp/filesystem`.

[2] *Eigen*. URL: `https://eigen.tuxfamily.org/index.php?title=Main_Page`.

[3] *Eigen Block Operations Module*. URL: `https://eigen.tuxfamily.org/dox/group__TutorialBlockOperations.html`.

[4] *Eigen Broadcasting Module*. URL: `https://eigen.tuxfamily.org/dox/group__TutorialReductionsVisitorsBroadcasting.html#TutorialReductionsVisitorsBroadcastingBroadcasting`.

[5] *Eigen `SelfAdjointEigenSolver` library*. URL: `https://eigen.tuxfamily.org/dox/classEigen_1_1SelfAdjointEigenSolver.html`.

[6] P.J. Phillips et al. "The FERET evaluation methodology for face-recognition algorithms." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.10 (2000), pp. 1090–1104. DOI: `10.1109/34.879790`.

[7] M.A. Turk and A.P. Pentland. "Face recognition using eigenfaces." In: *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1991, pp. 586–591. DOI: `10.1109/CVPR.1991.139758`.