CS 679: Pattern Recognition

University of Nevada, Reno - Spring 2024

Assignment 3

Jaleesa Houle

Due: April 22nd, 2024

Statement: "I declare that all material in this assignment is my own work except where there is clear acknowledgment or reference to the work of others. I understand that both my report and code may be subjected to plagiarism detection software, and fully accept all consequences if found responsible for plagiarism, as explained in the syllabus, and described in UNR's Academic Standards Policy: UAM 6,502."

1. Theory

Experiment A

General background and theory

The theory for Assignment 3 is described by Turk and Pentland (1991). The authors developed a method of face recognition using Principle Component Analysis (PCA). They discovered that, using PCA on a database of images, one could use the resulting basis of eigenvectors to reconstruct, recognize, and detect faces when given a novel image. To implement these ideas, we will use images from the FERET face database (Phillips et al. (2000)). These images show only the face region and have been normalized so that the orientation, position, and size are all similar across images. Additionally, there is a variety of faces with respect to gender, age, and ethnicity.

PCA theory and procedures

PCA is a method of dimensionality reduction. The goal of PCA is to find a basis of eigenvectors (i.e., principle components) which describes the data. PCA is useful in that it de-correlates the data and preserves original variances of the data. This allows one to reduce the number of dimensions of their data for classification while still preserving a large amount of information/variance in the data. To perform PCA on a database of images, the database of images must first be shaped into an array of shape $(H\text{x}W, M)$, where M is the total number of images, H is the height of each image, and W is the width of each image. Given a database of images, the sample mean of the images can be computed as

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

Once the sample mean is computed, it is subtracted from each image ($\Phi_i = \mathbf{x} - \bar{\mathbf{x}}$) so that the data is centered at zero. The resulting matrix of images is then $\mathbf{A} = [\Phi_1\Phi_2...\Phi_M]$. Once the data is centered, we can compute the sample covariance matrix such that

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

Next, we can compute the eigenvectors and eigenvalues of Σ_x using the formula $\mathbf{\Sigma_x u_i} = \lambda_i \mathbf{u}_i$. In practice, this is computationally expensive, since $\mathbf{A}\mathbf{A}^T$ is of shape $(H\text{x}W, H\text{x}W)$. Therefore, instead of using the matrix $\mathbf{A}\mathbf{A}^T$, we can instead use the matrix $\mathbf{A}^T\mathbf{A}$, which is of size $(M\text{x}M)$ and is generally significantly smaller than $\mathbf{A}\mathbf{A}^T$ ($M << H\text{x}W$). This "trick" can be performed due to the relationship between

the two matrices, where if we assume that $(\mathbf{A}\mathbf{A}^T)\mathbf{v}_i = \mathbf{u}_i\mathbf{v}_i$ and $(\mathbf{A}^T\mathbf{A})\mathbf{v}_i = \mu_\mathbf{i}\mathbf{v}_i$, then, multiplying both sides by $\mathbf{A}$ gives

$$\begin{aligned} &\mathbf{A}(\mathbf{A}^\top\mathbf{A})\mathbf{v}_i = \mathbf{A}\mu_i\mathbf{v}_i \text{ , i.e.,} \\ &(\mathbf{A}\mathbf{A}^\top)\mathbf{A}\mathbf{v}_i = \mu_i(\mathbf{A}\mathbf{v}_i). \end{aligned} \tag{2}$$

If we also assume that $(\mathbf{A}\mathbf{A}^T)\mathbf{v}_i = \lambda_i\mathbf{u}_i$, then this manipulation shows that

$$\begin{aligned} \lambda_\mathbf{i} &= \mu_\mathbf{i} \\ \mathbf{u_i} &= \mathbf{A}\mathbf{v}_i. \end{aligned} \tag{3}$$

Since $\lambda_\mathbf{i} = \mu_\mathbf{i}$, it can be shown that the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are the top M eigenvalues of $\mathbf{A}\mathbf{A}^T$. Additionally, the top M eigenvectors of $\mathbf{A}\mathbf{A}^T$ can be found using the relationship $\mathbf{u_i} = \mathbf{A}\mathbf{v}_i$. This relationships allows us to calculate the desired eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^T$ by using the much smaller matrix $\mathbf{A}^T\mathbf{A}$.

Once the eigenvectors of $\mathbf{A}^T\mathbf{A}$ are found and transformed to be the eigenvectors of $\mathbf{A}\mathbf{A}^T$ using Equation 3, they are normalized to unit length (i.e., so that $||\mathbf{u_i} = 1||$). These unit length eigenvectors form a basis, and each $\mathbf{u_i}$ can be transformed back into the image space using the relationship

$$\mathbf{w_{ij}} = \frac{255(\mathbf{u_{ij}} - \mathbf{u}_{\min})}{\mathbf{u}_{\max} - \mathbf{u}_{\min}} \tag{4}$$

where $\mathbf{w_{ij}}$ represents integer values between $[0, 255]$. In projecting these basis vectors back into the image space, we see that each eigenvector looks like a "ghost face", which is why they are typically referred to as eigenfaces. We can reconstruct any image $\mathbf{x}$ in the image gallery by using the equation

$$\mathbf{x} - \bar{\mathbf{x}} \approx \sum_{i=1}^{M} \mathbf{y_i}\mathbf{u_i} \tag{5}$$

where $\mathbf{y_i}$ is an eigen-coefficient which describes the weight of each eigenvector needed to reconstruct $\mathbf{x}$. To reduce dimensionality, we can take the top K largest eigenvalues/eigenfaces ($K << M$) and approximate the image $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{x}} - \bar{\mathbf{x}} \approx \sum_{i=1}^{K} \mathbf{y_i}\mathbf{u_i}. \tag{6}$$

This reduction allows us to preserve some chosen threshold of information within the data, while drastically reducing dimensionality and computational time.

Choosing a threshold for K

Since each eigenvalue found from Σ_x corresponds to variance for a feature, the top K eigenvalues values can be thought of as corresponding to some percentage of

information of the data. To choose a value of K which preserves a desired amount of information/variance from the data, we can use the relationship

$$\frac{\sum_{i=1}^{K} \lambda_{\mathbf{i}}.}{\sum_{i=1}^{M} \lambda_{\mathbf{i}}} > T_r \tag{7}$$

where T_r is a value between 0 and 1 (i.e., to preserve 90% of the data, we would set $T_r = 0.9$ and solve for K. The resulting K eigenvalues and their corresponding eigenvectors would retain 90% of the information from the original data. We could use the approximate reconstruction error ($||\mathbf{x} - \hat{\mathbf{x}}||$) to determine how close the estimation ($\hat{\mathbf{x}}$) is compared to the original image ($\mathbf{x}$).

Face recognition using the Mahalanobis distance
In addition to reconstructing images within the training data, we can also see how well new test images are recognized. During the training phase, we can compute the eigen-coefficients for each training image such that

$$\mathbf{\Omega_i} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_K \end{bmatrix}. \tag{8}$$

Then, we can do the same for an unknown image Φ, by first subtracting the mean face ($\Phi = \mathbf{x} - \hat{\mathbf{x}}$) and then projecting the unknown face into the eigenspace so that

$$y_i = \Phi^T \mathbf{u_i}, \text{ for } i = 1, 2, ..., K. \tag{9}$$

We can then compare the eigen-coefficients of the unknown face (Ω) with the eigen-coefficients of the known faces (Ω_i) and find the closest match such that

$$p = \arg\min_i ||\Omega - \Omega_i||, \text{ for } i = 1, 2, ..., M. \tag{10}$$

For this assignment, we will compute this using the Mahalanobis distance, where

$$e_r = \min_i ||\Omega - \Omega_i|| = \min_i \sum_{j=1}^{K} \frac{1}{\lambda_j} (y_j - y_j^i)^2. \tag{11}$$

Turk and Pentland (1991) refer to e_r as the difference in face space (difs). By computing the difs between an unknown testing image and every training image, we can assign the training image with the smallest distance as the recognized face of the unknown image. In practice this measurement is not perfect, therefore some faces may be recognized as the wrong person. To see how well the e_r measurement does at recognizing a novel face, we will take the top r images corresponding to the r smallest distances between the unknown faces and the faces in the training image database. Then, we will plot a Cumulative Match Characteristic (CMC) curve

(Phillips et al. (2000)) by varying r and counting how many test faces are correctly recognized within the r top matches.

Experiment B

For experiment B, we are asked to remove the first 50 subjects in the given training image database, recompute the eigenvalues/eigenvectors/eigen-coefficients using this reduced sample space, and then find the best match and minimum e_r for each test image. The faces will be reconstructed and compared using the top K eigenfaces and eigen-coefficients which preserve 95% of information in the data. The matches and error distances for $r = 1$ will be compared for intruders (i.e., the first 50 subjects who were removed from the training data) and non-intruders.

Determining a threshold for intruders

After determining the smallest e_r for each test image using the Mahalanobis distance as outlined above, we can normalize the distances (using $e_r/\max(e_r)$) so that the distance error for each test face and its corresponding closest training face are between [0,1]. Then, the True Positive (TP) and False Positive (FP) rates of each match can be found by comparing e_r to a varying threshold (T_r) between [0,1]. The goal is to find a threshold for $e_r < T_r$ where the optimum amount of faces are correctly matched, while intruders are also correctly rejected. The TP and FP rates are found such that

$$\begin{aligned} TP &= \frac{\#\text{ true positives}}{\text{total }\#\text{ of non-intruders}} \\ FP &= \frac{\#\text{ false positives}}{\text{total }\#\text{ of intruders}}. \end{aligned} \tag{12}$$

To produce relatively smooth ROC curves, the TP and FP rates were found at 50 linearly spaced values of T_r between [0,1].

2. Results and Discussion

Experiment A

a.I

Figure 1 show the average face from the given 'fa_H' training image database, which contained 1204 images at a resolution of 48x60 pixels.

Figure 1: The average face based on 1204 training images of resolution 48x60.

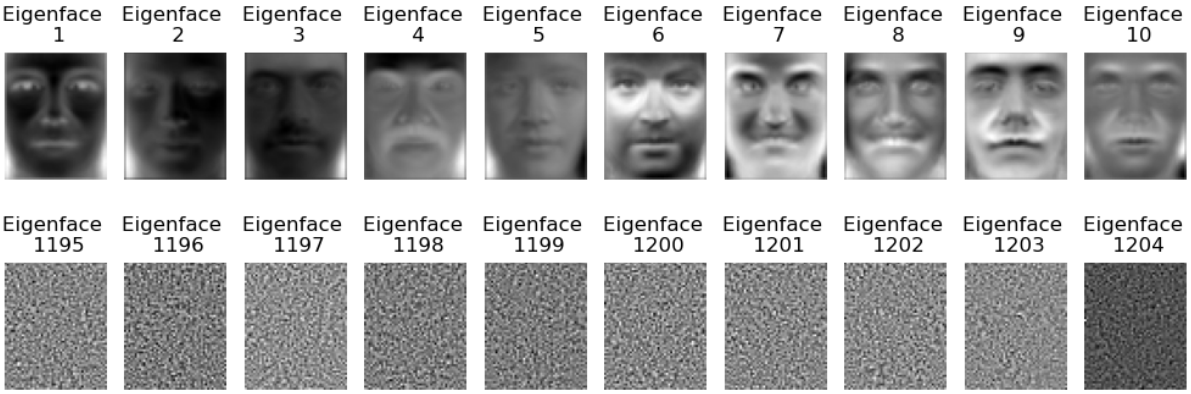


Figure 2: The top 10 and smallest 10 eigenfaces from the high resolution training data.

Figure 2 shows the corresponding top 10 and bottom 10 eigenfaces for the 'fa_H' image database. We can see that the top eigenfaces do indeed look like "ghost faces". Additionally, we can see that the 10 eigenfaces corresponding to the smallest eigenvalues do not resemble faces, indicating that not much (if any) information is preserved by the smallest eigenvalues/eigenvectors obtained using PCA.

a.II

Figure 3 shows the probability of the query (test face) being among the top r faces retrieved from the training gallery when using the top K eigenvalues/eigenvectors which preserve 80% of the information in the data. We can see that approximately 55% of the query faces were correctly identified on the first match, while close to

90% of the query faces are correctly matched if considering the top $r = 50$ images retrieved from the training gallery.

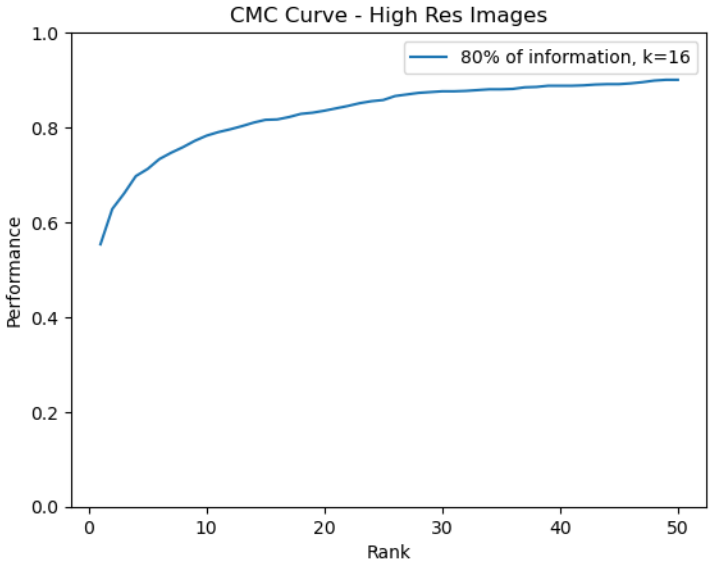


Figure 3: The Cumulative Match Characteristic (CMC) curve corresponding to matches using the top $K = 16$ eigenfaces, which preserve 80% of information in the data.

a.III

Figure 4a exemplifies three query images which were correctly identified on the first match. We can see that the test images are not the same as those in the training gallery, but the variance explained by 80% of information in the data was enough to determine that the two faces were from the same individual.

(a) Three query images that are correctly matched, along with the corresponding best matched training samples.

(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

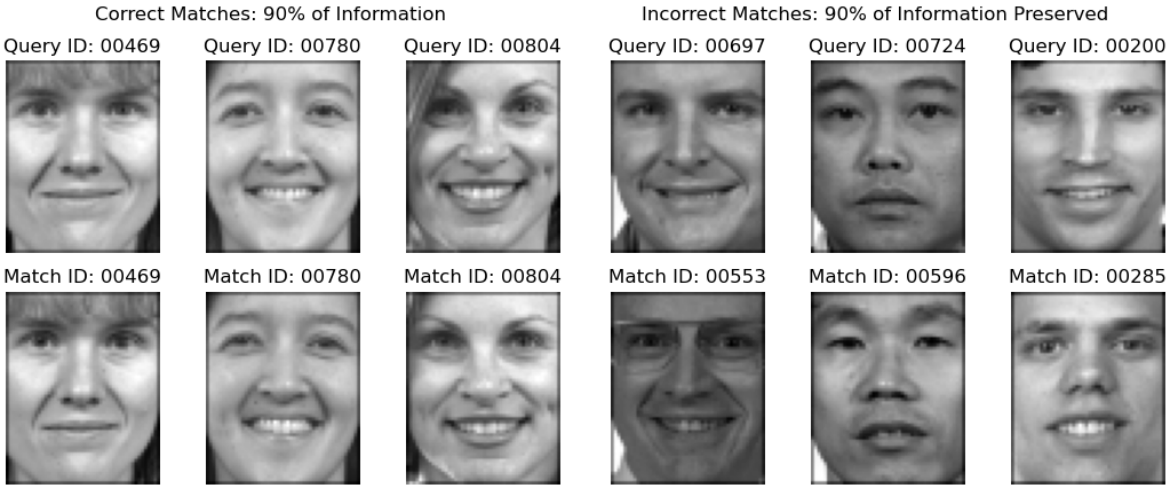
Figure 4: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 80% of information preserved in the data.

a.IV

Figure 4b demonstrates three query images which were incorrectly matched when using the top $K = 16$ eigenvalues/eigenvectors. We can see that the images incorrectly matched have some similar and features, and that classification may be impacted by things like face orientation and lighting.

a.V

Experiments a.I-IV were repeated using the same training and test data and instead considering the top eigenvalues and eigenvectors which correspond to 90% and 95% information preservation. Figures 5a and 5b demonstrate three correctly and incorrectly matched faces (assuming $r = 1$) when using $K = 49$ eigenvectors which correspond to 90% information preservation.

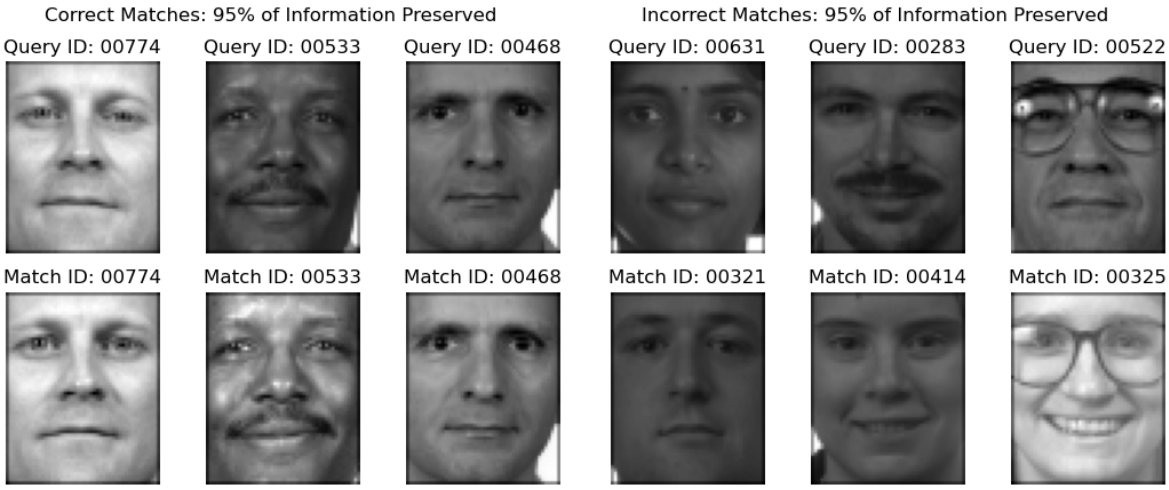


(a) Three query images that are correctly matched, along with the corresponding best matched training samples.

(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

Figure 5: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 90% of information preserved in the data.

Similarly, Figures 6a and 6b demonstrate three correctly and incorrectly matched faces (assuming $r = 1$) when using the top $K = 114$ eigenvectors which correspond to 95% information preservation. By looking at the incorrect matches in Figures 5b and 6b, we can again see similarities in the more distinctive features, such as face orientation, mouth position (i.e., smiling with or without teeth), and the presence of glasses.



(a) Three query images that are correctly matched, along with the corresponding best matched training samples.

(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

Figure 6: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 95% of information preserved in the data.

Figure 7 compares the CMC curves for correct matches in the top r faces when 80%, 90%, and 95% of information is preserved. We can see that when $r = 1$, the matching performance using $K = 16$ eigenvalues/eigenvectors is worse that when using $K = 49$ and $K = 114$ eigenvalues/eigenvectors, respectively. That said, there appears to be only a modest improvement in performance of $r = 1$ matches when using $K = 49$ versus $K = 114$ eigenvalues. This indicates that there are diminishing returns when using additional dimensions to represent and classify these data.

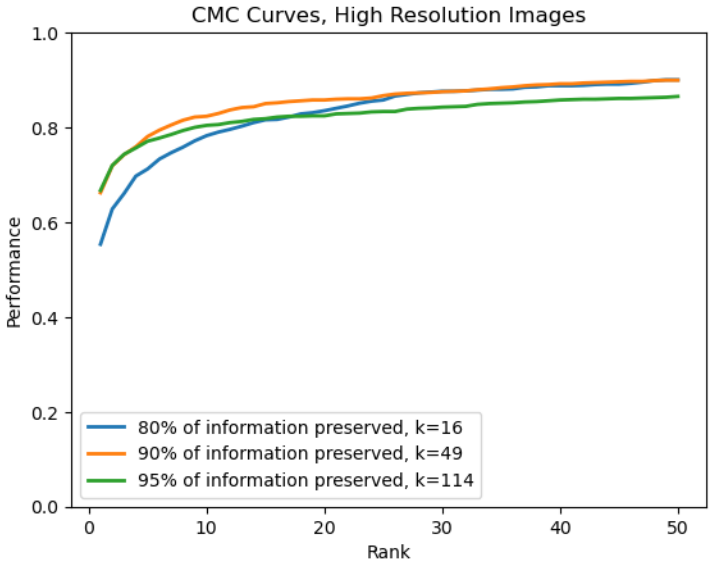


Figure 7: A comparison of the Cumulative Match Characteristic (CMC) curves corresponding to matches using the top $K = 16$, $K = 49$, and $K = 114$ eigenfaces, which preserve 80%, 90%, and 95% of information in the data, respectively.

Interestingly, Figure 7 shows that as r is increased from 1 to 50, the query matching performance using 80% and 90% of information converge to around 0.9 at $r = 50$, while the matching performance using 95% of information appears to reach a steady state around $r = 10$ and does not increase much thereafter, reaching a maximum in performance of about 0.86. These results again indicate that while using more features is helpful for $r = 1$, the value of preserving additional variance in the data may diminish for increasingly large values of K. In order to preserve 95% of information versus 90%, an additional 65 features were used, indicating that the value of information in those eigenvalues was much smaller than the value in the first 49 eigenvalues. Thus, capturing higher levels of variance can sometimes be outweighed by the added complexity and computational requirements needed (i.e., "curse of dimensionality").

Experiment B

For experiment B, we are asked to remove all images from the first 50 subjects from the training gallery. Then, we are asked to compute the new eigenvalues, eigenfaces, and eigen-coefficients when using the top K eigen-coefficients corresponding to 95% of information preserved from the reduced training data. Once this is complete, we can run recognition on the test images and find the images that match with the lowest error e_r. We can then compute the ROC curve of $r = 1$ matches by computing the true and false accept rates of non-intruders and intruders (where intruders are the first 50 subjects that were removed) at various thresholds (where $T_r > e_r$).

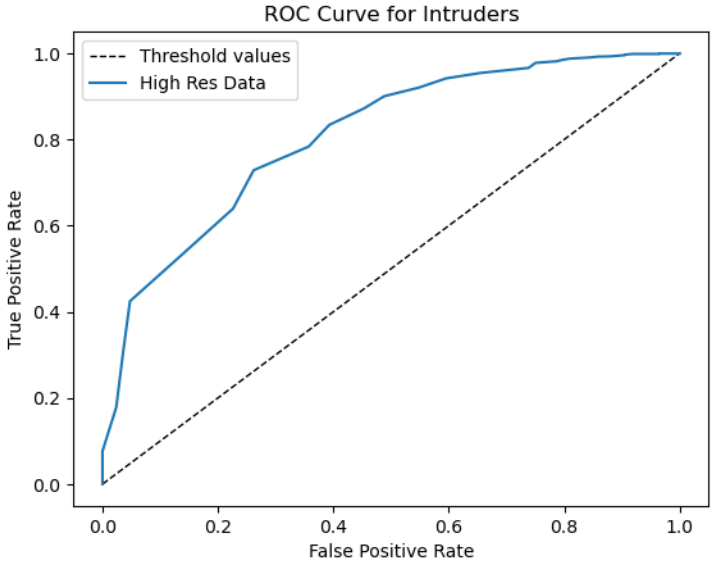


Figure 8: ROC curve comparing true positives and false positives of query matches (assuming $r = 1$) when using the top K eigenvectors corresponding to 95% of information preserved from the reduced training data set. The black dashed shows the threshold values used to accept matches, normalized between [0,1].

Figure 8 demonstrates TP and FP rates when accepting matches using a varying threshold $T_r > e_r$. At $T_r = 0$, all matches are rejected, while at $T_r = 1$ all matches are accepted. The goal would be to find the threshold which maximizes true posi-

tives and minimizes false positives, so that as many intruders are rejected as possible while still correctly accepting as many non-intruders as possible.

Experiment C

In experiment C we are asked to perform the same computational analysis as in (A) and (B) with a low resolution (16x20) version of the training and testing datasets.

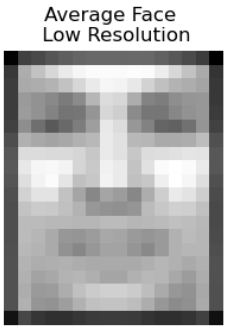


Figure 9: The average face based on 1204 training images of resolution 16x20.

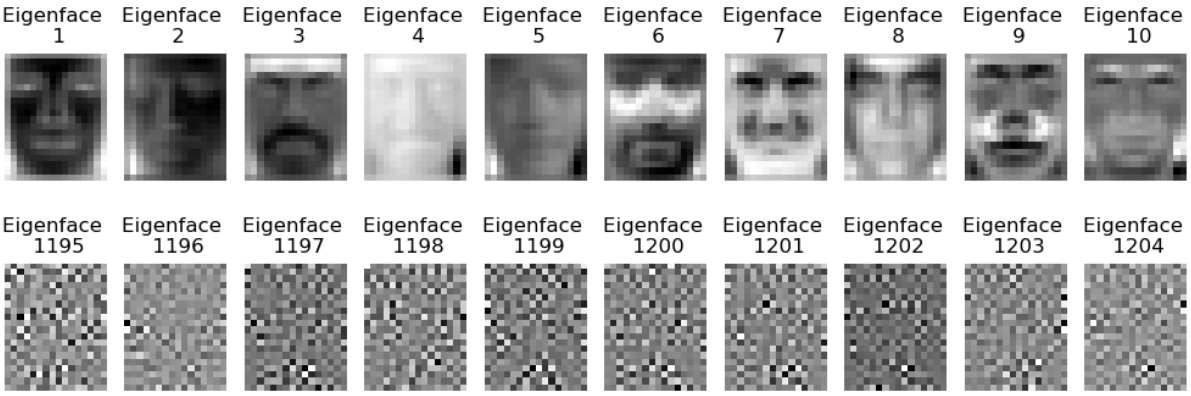


Figure 10: The top and bottom 10 eigenfaces from the low resolution training data.

We can see, when comparing Figures 1 and 9 that the average faces still look similar, indicating that the low resolution average face is a down sampled version of the high resolution average face. Similarly, Figures 2 and 10 demonstrate that the eigenfaces in the low resolution data also appear to be similar to those in the high resolution data, albeit downsampled. As an aside, it can be seen that in some cases, an eigenface may be light or dark (for example, in Figure 9, eigenface 1 is dark while eigenface 4 is very light). This occurs when calculating eigenvectors of the covariance matrix, where some unit eigenvectors may be the same in magnitude but multiplied by -1. Depending on the computational solver used to compute the eigenvalues and eigenvectors, different eigenvectors may be positive or negative. Whether an eigenvector is multiplied by -1 or not does not change the accuracy or interpretation of results.

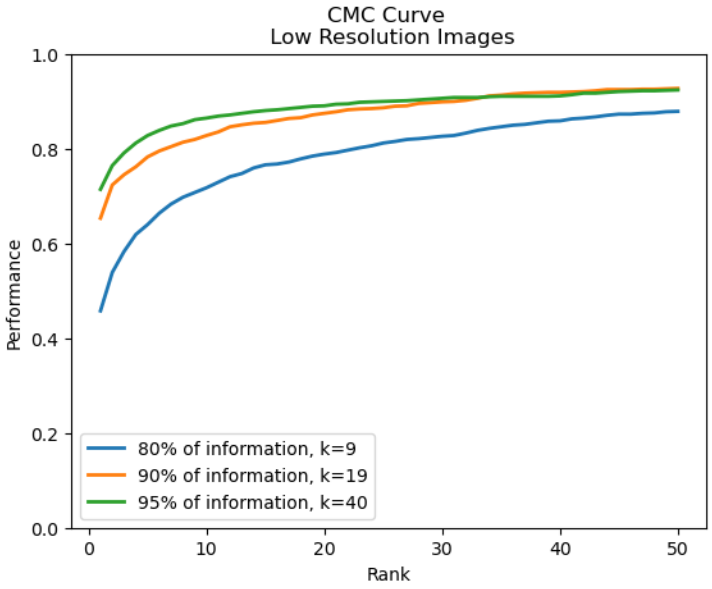


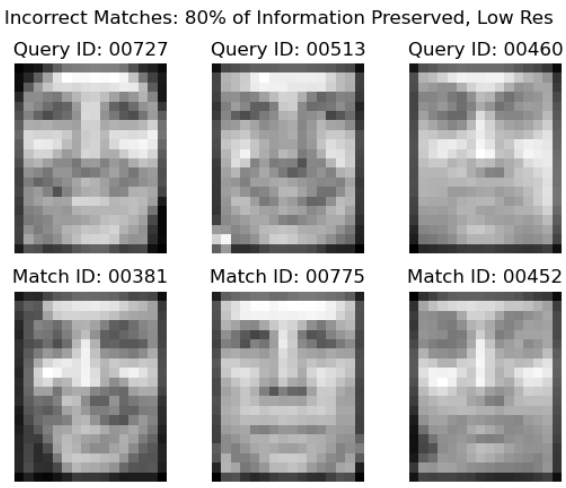
Figure 11: A comparison of the Cumulative Match Characteristic (CMC) curves corresponding to matches using the top $K = 9$, $K = 19$, and $K = 40$ eigenfaces, which preserve 80%, 90%, and 95% of information in the low resolution data, respectively.

When analyzing the CMC curves depicted in Figure 11, it's evident that employing more features yields improved performance across all r values. For instance, at $r = 1$, matching accuracy is below 50% when using eigen-coefficients corresponding to 80% of preserved information. In contrast, approximately 65% and 71% of queries are correctly matched at $r = 1$ when preserving 90% ($K = 19$) or 95% ($K = 40$) of information, respectively. Additionally, we can see that fewer eigenvalues/coefficients are necessary to retain 80%, 90%, and 95% of information in the low resolution images.

Comparing these findings to Figure 7, we observe that at $r = 1$, performance with 80% information preservation is worse for low resolution images (around 45%) compared to the high resolution data (approximately 55%). However, for 90% and 95% preservation, the CMC curves demonstrate superior performance with low resolution data compared to high resolution data.

These results suggest that an abundance of features (i.e., pixels in this case) isn't always advantageous for pattern recognition and classification, and too few features can also impede performance. In the case of low resolution images, only $K = 8$ features are necessary for recognition at 80% information preservation, while for high resolution images, $K = 18$ features are required, reflecting the lower overall information in low resolution images.

(a) Three query images that are correctly matched, along with the corresponding best matched training samples.



(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

Figure 12: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 80% of information preserved in the low resolution data.

The number of features dramatically changes when preserving 90% and 95% of information. For instance, for 90% preservation in high resolution data, $K = 49$, whereas in low resolution data, $K = 19$ suffice. Similarly, for 95% preservation, $K = 114$ features are required in high resolution data, while only $K = 40$ are needed in low resolution images.

Despite employing more features in high resolution images, low resolution images achieve higher accuracy at $r = 1$ for preserving 90% and 95% of information, with slightly superior overall performance as r increases from 1 to 50. This outcome underscores the "curse of dimensionality" concept, emphasizing the importance of considering both lower and upper feature thresholds, along with the desired information preservation level, when utilizing PCA for dimensionality reduction.

(a) Three query images that are correctly matched, along with the corresponding best matched training samples.

(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

Figure 13: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 90% of information preserved in the low resolution data.

(a) Three query images that are correctly matched, along with the corresponding best matched training samples.

(b) Three query images that are incorrectly matched, along with the corresponding best matched training samples.

Figure 14: A comparison of correctly and incorrectly matched faces assuming $r = 1$ and using the top K eigenvectors which correspond to 95% of information preserved in the low resolution data.

Figures 12, 13, and 14 give examples of three correctly and incorrectly matched queries (assuming $r = 1$) for the low resolution gallery when using the top K eigenvalues corresponding to 80%, 90%, and 95% of information preserved in the data. We can see that strong distinguishing features can still be made out, while other more minutiae features are not easily seen.

Experiment D

For experiment D, we are asked to repeat experiment B with the low resolution images. As was the case in (b), images from the first 50 subjects were removed from the training database. New eigenvalues, eigenvectors, and eigen-coefficients were computing using this reduced training face gallery. Then, recognition was performed using the test images with the top K eigen-coefficients corresponding to 95% of information preserved from the training data. ROC curves were used to compare the true and false accept rates of non-intruders and intruders (i.e., the first 50 subjects that were removed).

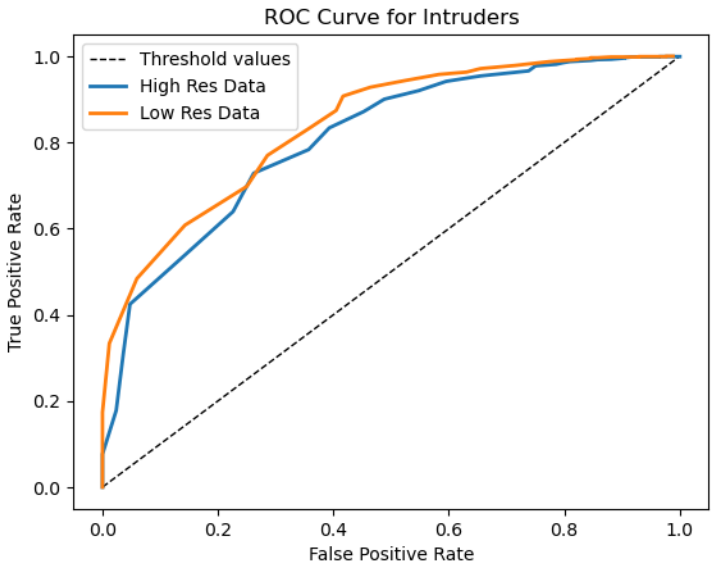


Figure 15: ROC curve comparing true positives and false positives of query matches (assuming $r = 1$) in the low resolution and high resolution data sets when using the top K eigenvectors corresponding to 95% of information preserved.

The ROC curves for face recognition using low resolution or high resolution images are compared in Figure 15. We can see that for the majority of thresholds, the low resolution data performs better (less area above the curve). This indicates, once again, that having less features can be beneficial when working with a finite dataset.

Part E

For part (e), we are asked to discuss the effect of using low-resolution images. From the results above, we can see that using low-resolution images improved performance in cases where 90% and 95% of information was preserved for recognition. Additionally, significantly less features were needed for preserving those percentages of information when using low resolution data. These results indicate that having larger numbers of features is not always beneficial. It is possible that recognition using the higher resolution data would improve if more training data was added, though it is difficult to know how much additional data would be needed to offset the additional amount of features used.

In the case where only 80% of information was preserved, recognition performance

was much better for the high resolution data. This indicates that although using less features can be beneficial for reducing computational time and model complexity, reducing dimensionality too much can still hinder performance.

The ROC curves comparing FP and TP recognition rates for 95% of information preserved also indicate better performance when using the low resolution data in comparison with the high resolution data (based on the area above the curve). Even so, the differences between the two curves are not drastically different, and one might expect that the high resolution data would perform similarly or better if using the top K eigenvalues corresponding to 80% of information preserved, based on the performances shown in Figures 7 and 11. Overall, these results demonstrate the importance of investigating optimal feature selection and dimensionality reduction when classifying data.

References

Duda, R. O., Hart, P. E., et al. (2006). *Pattern classification*. John Wiley & Sons.

Phillips, P. J., Moon, H., Rizvi, S. A., and Rauss, P. J. (2000). The feret evaluation methodology for face-recognition algorithms. *IEEE Transactions on pattern analysis and machine intelligence*, 22(10):1090–1104.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86.

Appendix

Code

Instructions to run code and generate figures:

All code used to generate the data and figures in this assignment is included in separate files which consist of the following:

'EigenFace.py': This script contains all relevant code for this assignment.

'CS679_ASSIGNMENT3_JaleesaHoule.ipynb': The actual data generation and analysis was performed in a Jupyter Notebook environment. A pdf of this notebook is also included in the pages below.

To run this code, download the python script and the Jupyter Notebook file into the same directory. Open the Jupyter Notebook and select 'run all'. This should re-run all of the analyses conducted for this assignment. The code requires installation of packages Open-CV, SciPy, Numpy, Pandas, and Matplotlib. This code was run using Python 3.8.15.

In addition to the code above, I have included the saved training data as .npz files. Instructions are included within the Jupyter Notebook if you wish to run only the testing phase and use precalculated training data.

Lastly, the test and training image files used for analyses are included in the 'Faces_FA_FB' directory so that the code can be easily executed when downloaded.

CS679_Assignment3_JaleesaHoule

April 21, 2024

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy
import cv2
import os
import argparse
import pandas as pd

import EigenFace as EF
```

1 Experiment (A): Use fa_H for training (i.e., to compute the eigenfaces and build the gallery set) and fb_H for testing (query). So, there will be 1204 images for training and 1196 images for testing.

1.1 Training phase:

[2]:
```
training_dir_highres ="Faces_FA_FB/fa_H/"
#print(os.listdir(training_dir_), len(os.listdir(dir_)))

experiment_a = EF.EigenFaces(mode='train')

experiment_a.read_images(training_dir_highres)
experiment_a.find_eigenfaces()
experiment_a.get_eigen_coefficients()

experiment_a.save_data(filename='training_data_highres')
```

```
data saved as file training_data_highres.npz
```

1.2 (a.I) Show (as an image) the following:

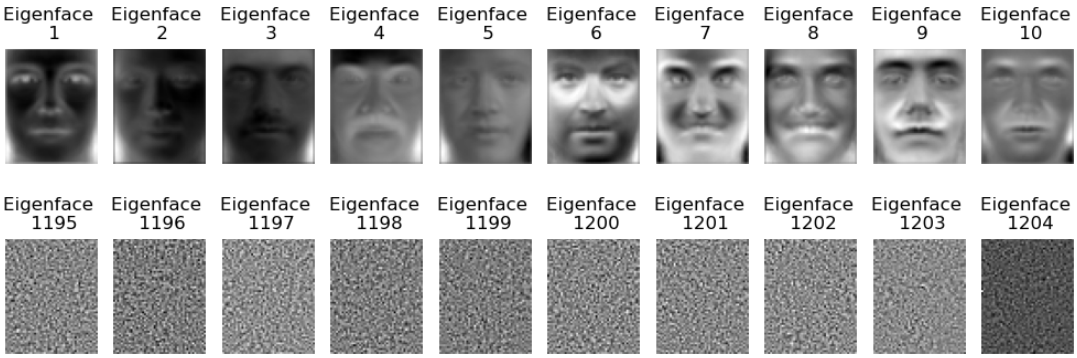
1.2.1 The average face

[3]:
```
fig = plt.subplots(figsize=(5,3))
plt.imshow(experiment_a.mean_face,cmap = plt.cm.gray)
plt.axis('off')
plt.title('Average Face \n High Resolution')
```

[3]:
```
Text(0.5, 1.0, 'Average Face \n High Resolution')
```

1.2.2 The eigenfaces corresponding to the 10 largest eigenvalues and the eigenfaces corresponding to the 10 smallest eigenvalues.

[4]:
```
fig, ax = plt.subplots(nrows=2, ncols=10, layout='tight', figsize=(10,4))
for i in range(10):
    ax[0,i].imshow(experiment_a.eigenfaces[:,i].reshape(60,48),cmap = plt.cm.
 ↪gray)
    ax[0,i].set_title('Eigenface \n %s' % (i+1))
    ax[0,i].axis('off')
for j,i in enumerate(np.arange(1194,1204)):
    ax[1,j].imshow(experiment_a.eigenfaces[:,i].reshape(60,48),cmap = plt.cm.
 ↪gray)
    ax[1,j].set_title('Eigenface \n %s' % (i+1))
    ax[1,j].axis('off')
```



1.3 Testing phase:

1.4 (a.II) In this experiment, consider the top eigenvectors (eigenfaces) preserving 80% of the information in the data. Project the query images onto this set of eigenvectors after subtracting the average face (from the training set). Then, compute the Mahalanobis distance between the eigen-coefficient vectors for each pair of training and query images as the matching distance.

[5]:
```
test_dir_highres ="Faces_FA_FB/fb_H/"
```

[6]:
```
# change mode to testing and then read in the test images
experiment_a.mode='test'
experiment_a.read_images(test_dir_highres)
experiment_a.get_eigen_coefficients()
```

1.4.1 *Note: if you want to skip running the training phase and instead use the precomputed training data, you can simply initialize with the saved data by running:*

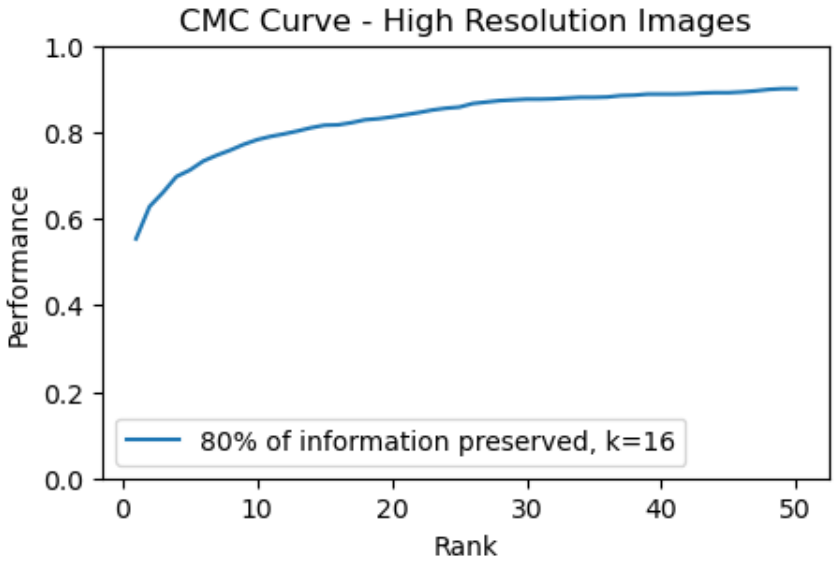
» trainingdata = np.load('training_data_highres.npz')

» experiment_a = EF.EigenFaces(mode='test', preloaded_training_data=trainingdata)

[7]:
```
highres_80p_df = experiment_a.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.8)

highres_thresh_80percent = []

for i in np.arange(1,51):
    highres_thresh_80percent.append(np.sum(highres_80p_df['isMatch_r'+str(i)])/␣
↪len(highres_80p_df))


fig= plt.subplots(figsize=(5,3))
```

```
plt.plot(np.arange(1,51),highres_thresh_80percent, label='80% of information␣
 ↪preserved, k=' + '%s' % experiment_a.k)
plt.ylim(0,1)
plt.xlabel('Rank')
plt.ylabel('Performance')
plt.title('CMC Curve - High Resolution Images')
plt.legend()
```

[7]: <matplotlib.legend.Legend at 0x7fd245baa8b0>



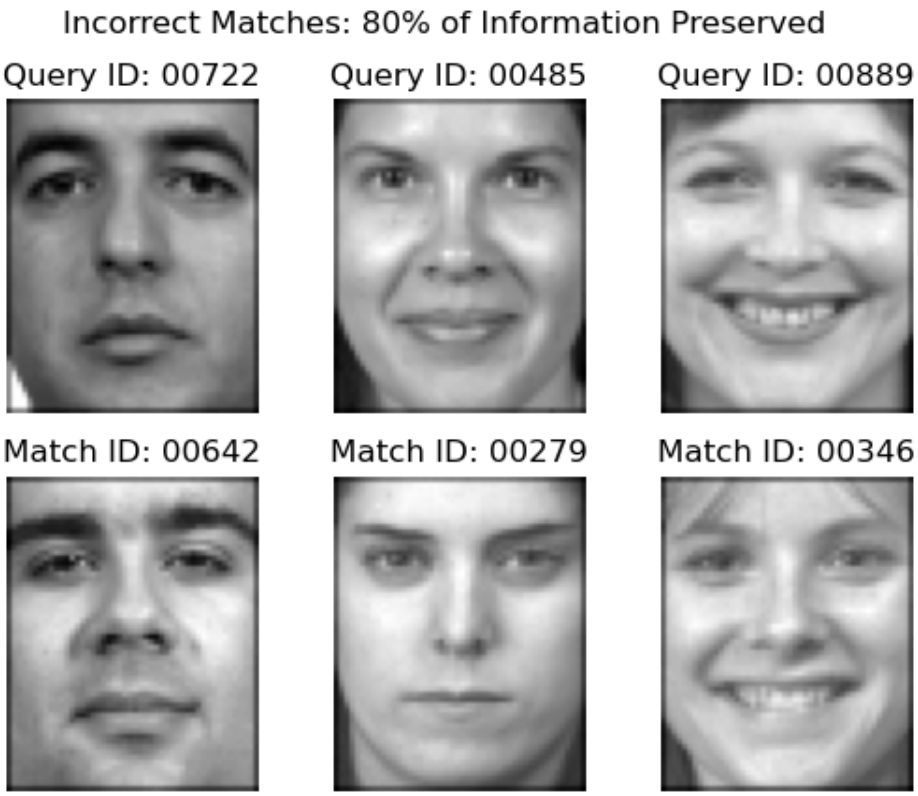
1.5 (a.III) Assuming r=1, show 3 query images that are correctly matched, along with the corresponding best matched training samples.

[8]:
```
correct_r1s = highres_80p_df[highres_80p_df.isMatch_r1==True]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 80% of Information Preserved\n \n \n')

    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
 ↪= plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[correct_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
```

```
    ax[1,j].axis('off')
```

1.6 (a.IV) Assuming r=1, show 3 query images that are incorrectly matched, along with the corresponding mismatched training samples.

[9]:
```
incorrect_r1s = highres_80p_df[highres_80p_df.isMatch_r1==False]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(incorrect_r1s), 3)):
    plt.suptitle('Incorrect Matches: 80% of Information Preserved\n \n \n')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[incorrect_r1s.query_idx.
 ↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[incorrect_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



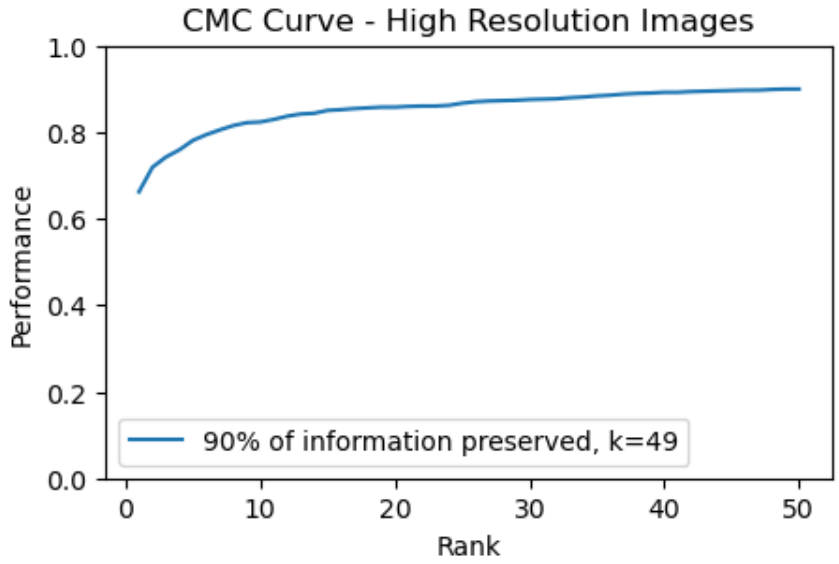
1.7 (a.V) Repeat (a.II – a.IV) by keeping the top eigenvectors corresponding to 90% and 95% of the information in the data. Plot the CMC curves on the same graph for comparison purposes. If there are significant differences in terms of identification accuracy in (a.II) and (a.V), try to explain why. If there are no significant differences, explain why too.

1.7.1 90% of information in data

[10]:
```
highres_90p_df = experiment_a.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.9)

highres_thresh_90percent = []

for i in np.arange(1,51):
    highres_thresh_90percent.append(np.sum(highres_90p_df['isMatch_r'+str(i)])/␣
↪len(highres_90p_df))
    #plt.plot(i, np.sum(q['isMatch_r'+str(i)])/ len(q),'.')


fig= plt.subplots(figsize=(5,3))
```

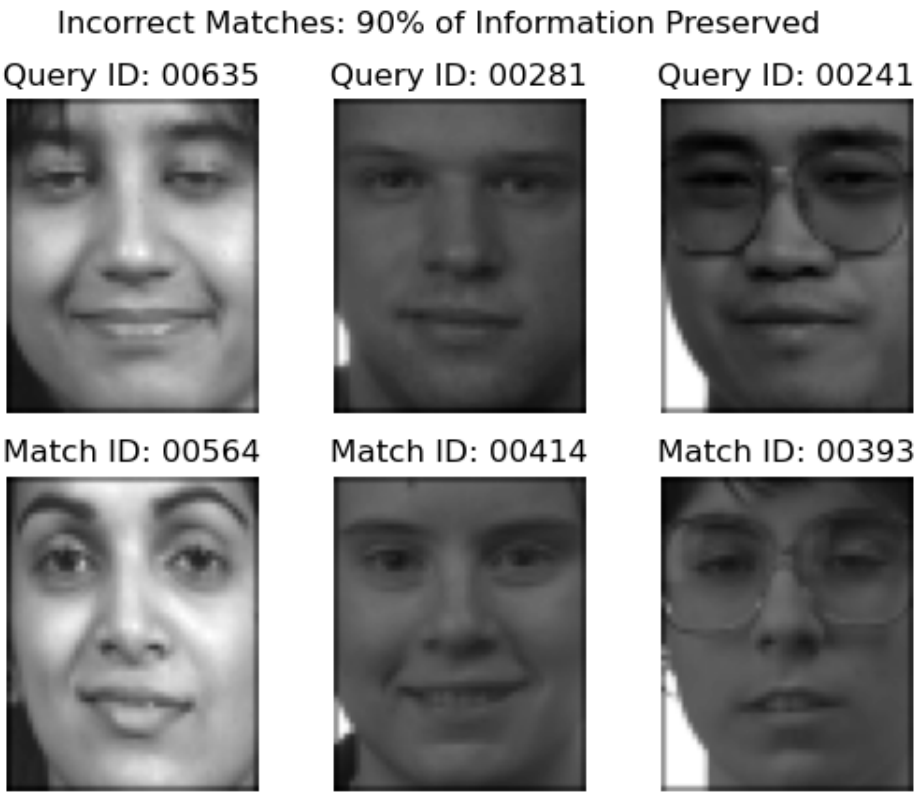
```
plt.plot(np.arange(1,51),highres_thresh_90percent, label=('90% of information␣
 ↪preserved, k=' + '%s' % experiment_a.k))
plt.ylim(0,1)
plt.xlabel('Rank')
plt.ylabel('Performance')
plt.title('CMC Curve - High Resolution Images')
plt.legend()
```

[10]: <matplotlib.legend.Legend at 0x7fd2440be100>



[11]:
```
correct_r1s = highres_90p_df[highres_90p_df.isMatch_r1==True]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 90% of Information Preserved \n \n \n')

    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
 ↪= plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[correct_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```

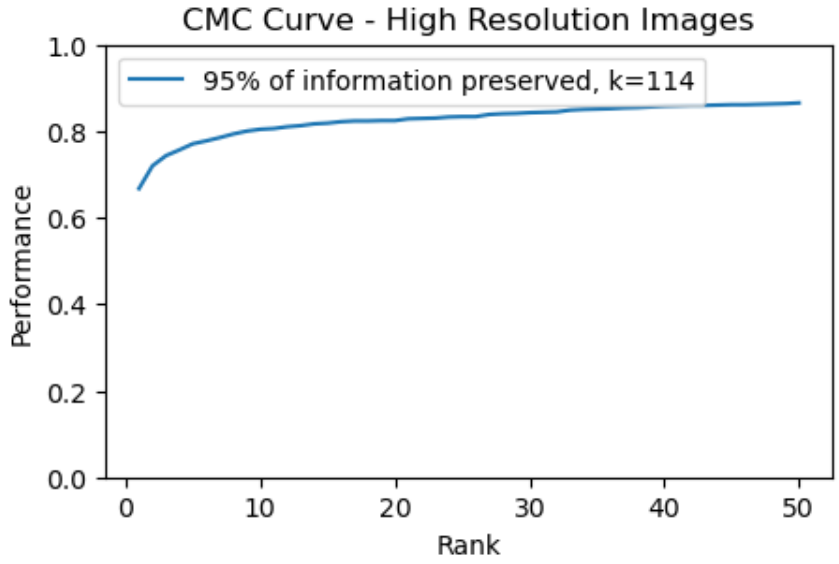
[12]:
```
incorrect_r1s = highres_90p_df[highres_90p_df.isMatch_r1==False]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(incorrect_r1s), 3)):
    plt.suptitle('Incorrect Matches: 90% of Information Preserved \n \n \n')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[incorrect_r1s.query_idx.
 ↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[incorrect_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



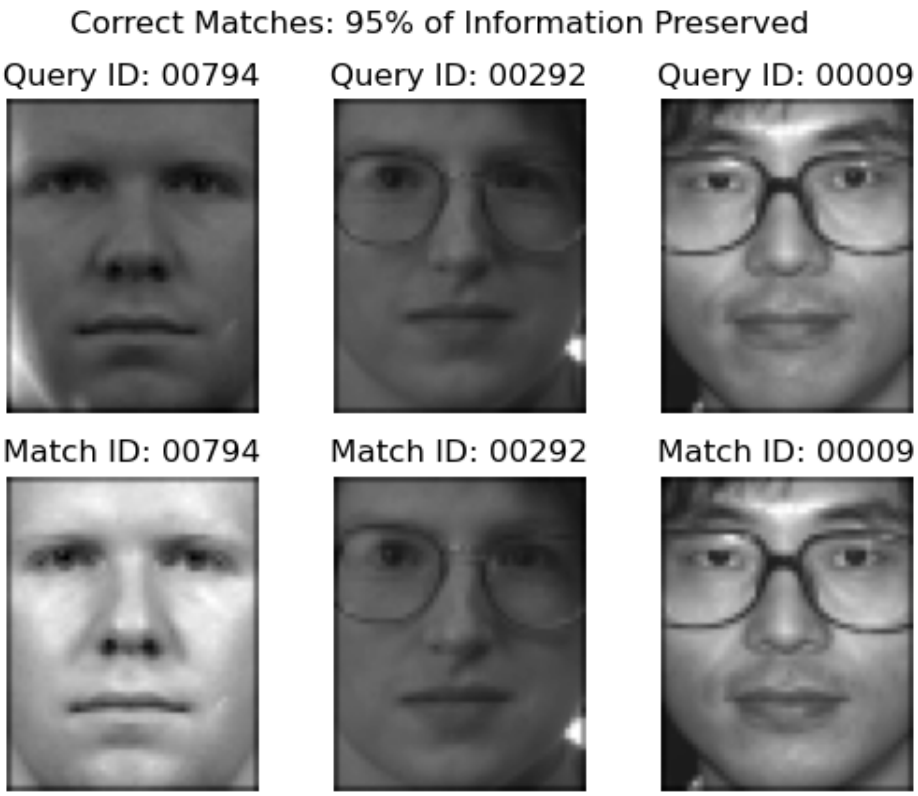
1.7.2 95% of information in data

[13]:
```
highres_95p_df = experiment_a.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.95)

highres_thresh_95percent = []

for i in np.arange(1,51):
    highres_thresh_95percent.append(np.sum(highres_95p_df['isMatch_r'+str(i)])/␣
↪len(highres_95p_df))


fig= plt.subplots(figsize=(5,3))
plt.plot(np.arange(1,51),highres_thresh_95percent, label=('95% of information␣
↪preserved, k=' + '%s' % experiment_a.k))
plt.ylim(0,1)
plt.xlabel('Rank')
plt.ylabel('Performance')
plt.title('CMC Curve - High Resolution Images')
plt.legend()
```

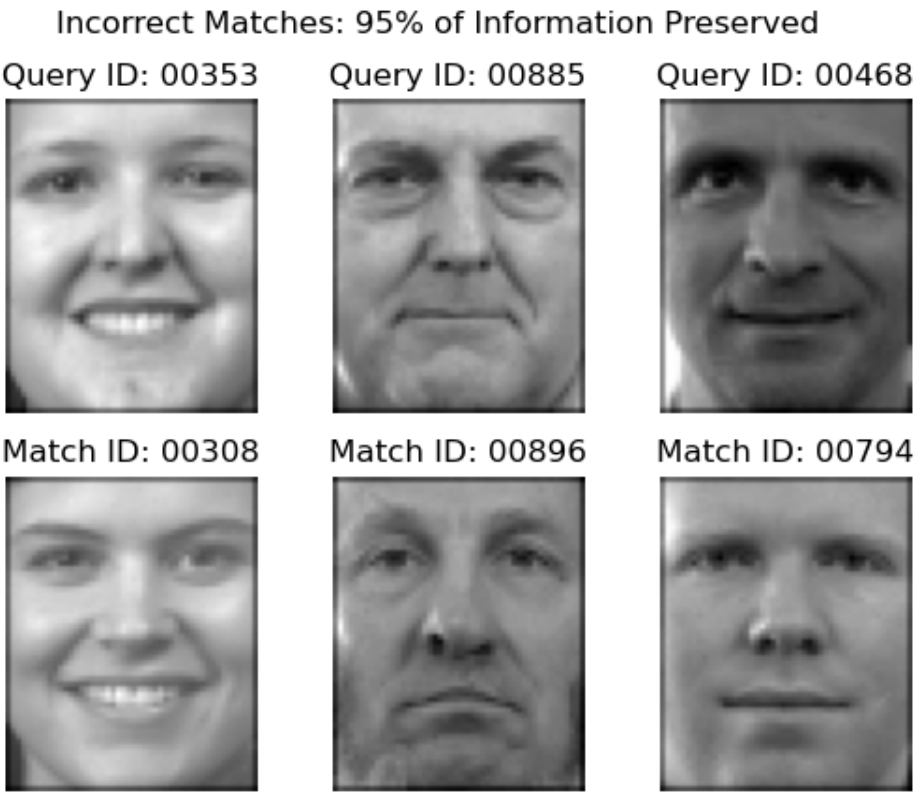
[13]: <matplotlib.legend.Legend at 0x7fd248712340>



[14]:
```
correct_r1s = highres_95p_df[highres_95p_df.isMatch_r1==True]

fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 95% of Information Preserved \n \n \n')

    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
 ↪= plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[correct_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



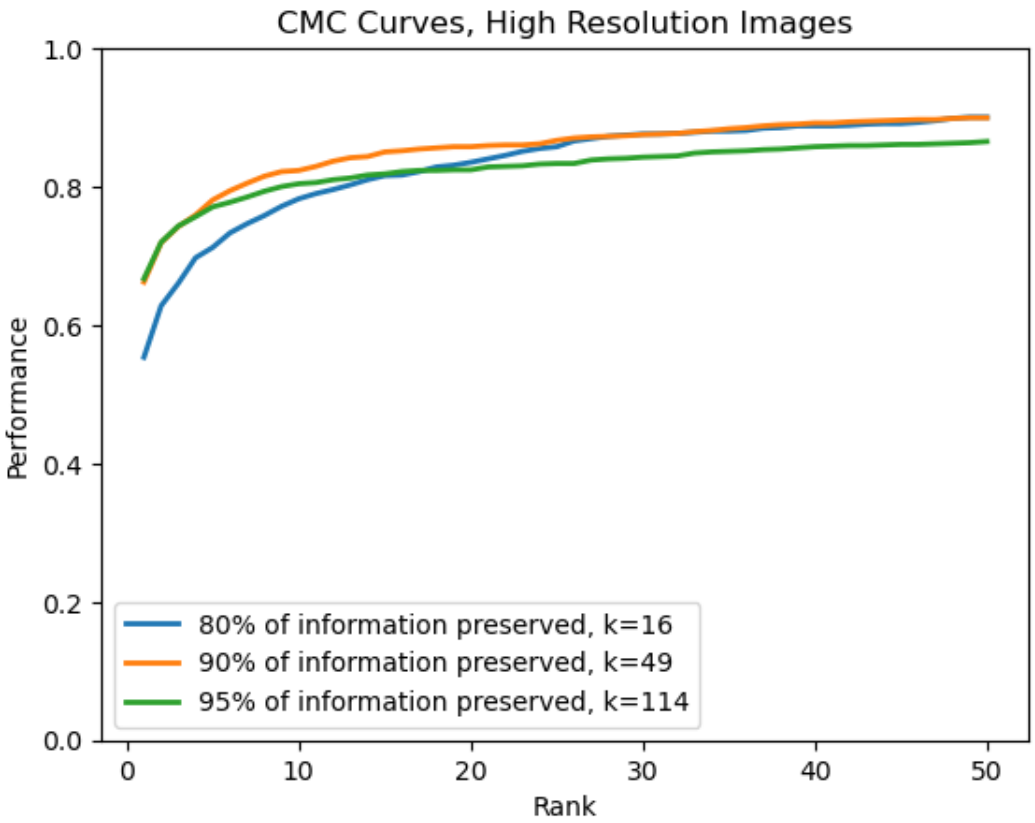
[15]:
```
incorrect_r1s = highres_95p_df[highres_95p_df.isMatch_r1==False]

fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(incorrect_r1s), 3)):
    plt.suptitle('Incorrect Matches: 95% of Information Preserved \n \n \n')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_a.test_images[incorrect_r1s.query_idx.
 ↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_a.training_images[incorrect_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



1.7.3 plotting all CMC Curves together

[16]:
```
plt.title('CMC Curves, High Resolution Images')
plt.plot(np.arange(1,51),highres_thresh_80percent,lw=2, label='80% of␣
 ↪information preserved, k=16')
plt.plot(np.arange(1,51),highres_thresh_90percent,lw=2, label='90% of␣
 ↪information preserved, k=49')
plt.plot(np.arange(1,51),highres_thresh_95percent,lw=2, label='95% of␣
 ↪information preserved, k=114')
plt.ylim(0,1)
plt.xlabel('Rank')
plt.ylabel('Performance')
plt.legend()
```

[16]: <matplotlib.legend.Legend at 0x7fd243e1ae50>



2 Experiment (b) In this experiment, you will test the performance of the eigenface approach on faces not in the gallery set (i.e., intruders).

2.1 For this, remove all the images corresponding to the first 50 subjects in fa_H (please note that a given subject might have more than one image in fa_H); let's call the reduced set fa2_H.

[17]:
```
training_dir_reduced ="Faces_FA_FB/fa2_H/"
test_dir_highres ="Faces_FA_FB/fb_H/"


experiment_b = EF.EigenFaces(mode='train')
experiment_b.read_images(training_dir_reduced)
experiment_b.find_eigenfaces()
experiment_b.get_eigen_coefficients()
experiment_b.save_data(filename='training_data_highres_intruders')
```

```
data saved as file training_data_highres_intruders.npz
```

2.2 Perform recognition using fa2_H for training (gallery) and fb_H for testing (query). Since the training set has changed, you would need to compute a new eigenspace for this experiment (i.e., compute the new covariance matrix and its eigenvalues/eigenvectors). Use the eigenvectors corresponding to 95% of the information in the data (i.e., do not experiment with different percentages as in (a)).

[18]:
```
trained_data = np.load('training_data_highres_intruders.npz')

experiment_b = EF.EigenFaces(mode='test', preloaded_training_data=trained_data)
experiment_b.read_images(test_dir_highres)
experiment_b.get_intruder_info(training_dir_highres, n_remove=50)
experiment_b.get_eigen_coefficients(thresh=0.95)
```

[19]:
```
experiment_b.recognize_faces(n_faces='all', rank=1, check_matches=[1], thresh=0.
↪95)
```

[19]:

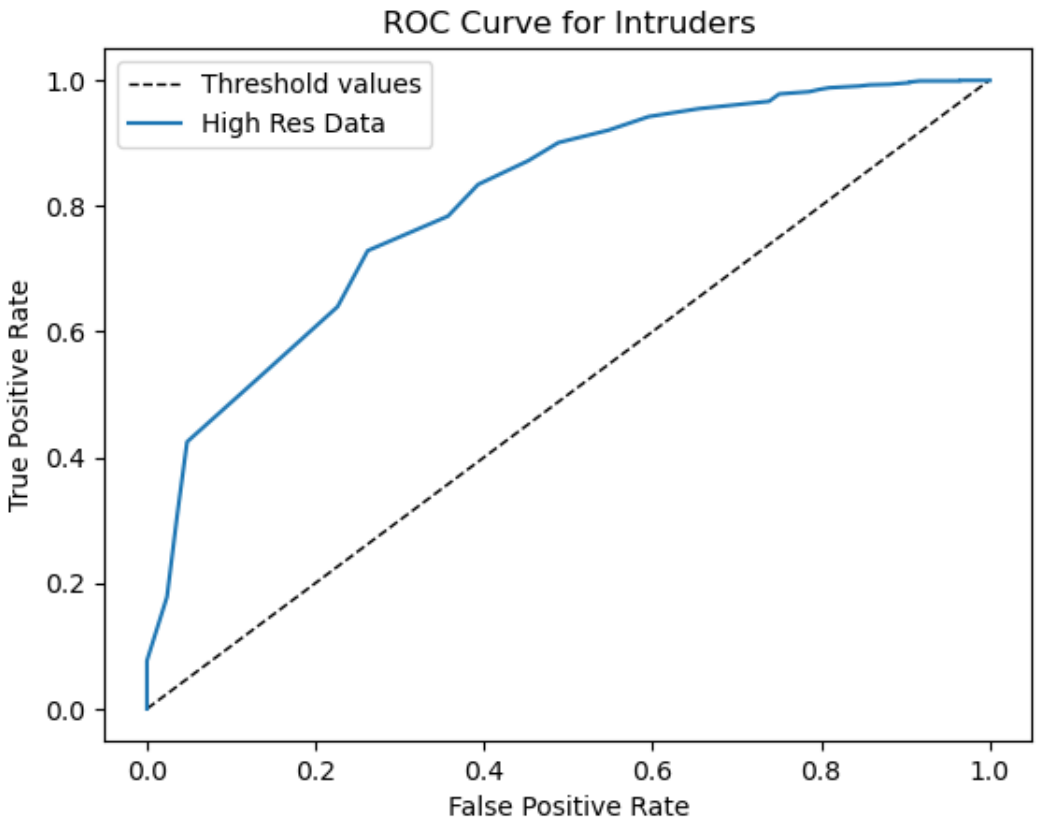
	query_id	query_idx	best_match	match_idx	dist_error	isMatch_r1
0	00514	0	[00514]	[333]	[43.41029273792352]	True
1	00146	1	[00146]	[83]	[55.8232479666125]	True
2	00571	2	[00571]	[87]	[28.61280568772876]	True
3	00441	3	[00300]	[698]	[83.61307999242855]	False
4	00875	4	[00875]	[89]	[184.8542201129846]	True
…	…	…	…	…	…	…
1191	00925	1191	[00925]	[1043]	[54.49363438864108]	True
1192	00878	1192	[00878]	[1048]	[100.54027191685259]	True
1193	00427	1193	[00427]	[1047]	[23.59255296984264]	True
1194	00998	1194	[00998]	[1046]	[76.63102920033884]	True
1195	00002	1195	[00298]	[115]	[69.15118362866737]	False

[1196 rows x 6 columns]

[20]:
```
experiment_b.get_ROC_error_info()
```

[21]:
```
plt.title('ROC Curve for Intruders')
plt.plot(experiment_b.ROC_thresholds, experiment_b.ROC_thresholds,'--', lw=1,␣
↪color='k', label='Threshold values')
plt.plot(experiment_b.false_positive_rate, experiment_b.true_positive_rate,␣
↪label='High Res Data')

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
```

[21]:
```
<matplotlib.legend.Legend at 0x7fd248751640>
```



3 Experiment C: Repeat experiment (a) using fa_L for training (gallery) and fb_L for testing.

[22]:
```
training_dir_lowres ="Faces_FA_FB/fa_L/"

experiment_c = EF.EigenFaces(mode='train')

experiment_c.read_images(training_dir_lowres)
experiment_c.find_eigenfaces()
experiment_c.get_eigen_coefficients()

experiment_c.save_data(filename='training_data_lowres')
```

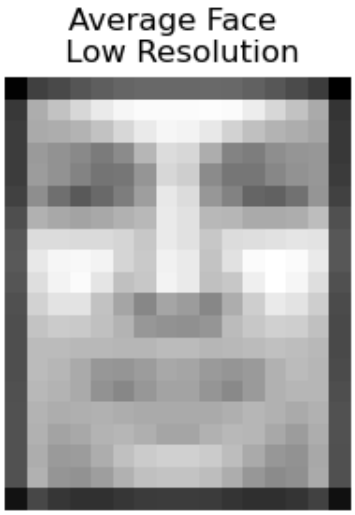
```
data saved as file training_data_lowres.npz
```

3.1 (c.I) Show (as an image) the following:

3.1.1 The average face

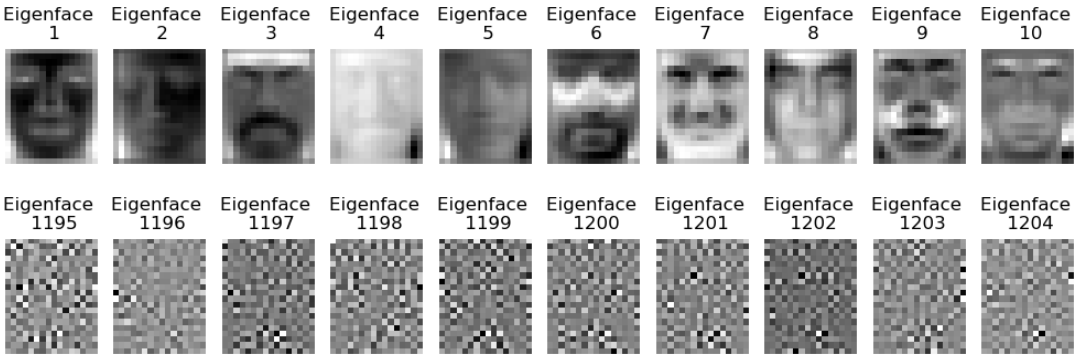
[23]:
```
fig = plt.subplots(figsize=(5,3))
plt.imshow(experiment_c.mean_face,cmap = plt.cm.gray)
plt.axis('off')
plt.title('Average Face \n Low Resolution')
```

[23]:
```
Text(0.5, 1.0, 'Average Face \n Low Resolution')
```



3.1.2 The eigenfaces corresponding to the 10 largest eigenvalues and the eigenfaces corresponding to the 10 smallest eigenvalues.

[24]:
```
fig, ax = plt.subplots(nrows=2, ncols=10, layout='tight', figsize=(10,4))
for i in range(10):
    ax[0,i].imshow(experiment_c.eigenfaces[:,i].reshape(experiment_c.
 ↪image_height,experiment_c.image_width),cmap = plt.cm.gray)
    ax[0,i].set_title('Eigenface \n %s' % (i+1))
    ax[0,i].axis('off')
for j,i in enumerate(np.arange(1194,1204)):
    ax[1,j].imshow(experiment_c.eigenfaces[:,i].reshape(experiment_c.
 ↪image_height,experiment_c.image_width),cmap = plt.cm.gray)
    ax[1,j].set_title('Eigenface \n %s' % (i+1))
    ax[1,j].axis('off')
```



3.2 Testing phase:

3.3 (c.II) In this experiment, consider the top eigenvectors (eigenfaces) preserving 80% of the information in the data. Project the query images onto this set of eigenvectors after subtracting the average face (from the training set). Then, compute the Mahalanobis distance between the eigen-coefficient vectors for each pair of training and query images as the matching distance.

[25]:
```
test_dir_lowres ="Faces_FA_FB/fb_L/"

experiment_c.mode='test'

experiment_c.read_images(test_dir_lowres)

experiment_c.get_eigen_coefficients()
```

3.3.1 *Note: if you want to skip running the training phase and instead use the precomputed training data, you can simply initialize with the saved data by running:*

» trainingdata = np.load('training_data_lowres.npz')

» experiment_c = EF.EigenFaces(mode='test', preloaded_training_data=trainingdata)

[26]:
```
lowres_80p_df = experiment_c.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.8)

lowres_thresh_80percent = []

for i in np.arange(1,51):
    lowres_thresh_80percent.append(np.sum(lowres_80p_df['isMatch_r'+str(i)])/␣
↪len(lowres_80p_df))

print('k eigenfaces used:', experiment_c.k)
```

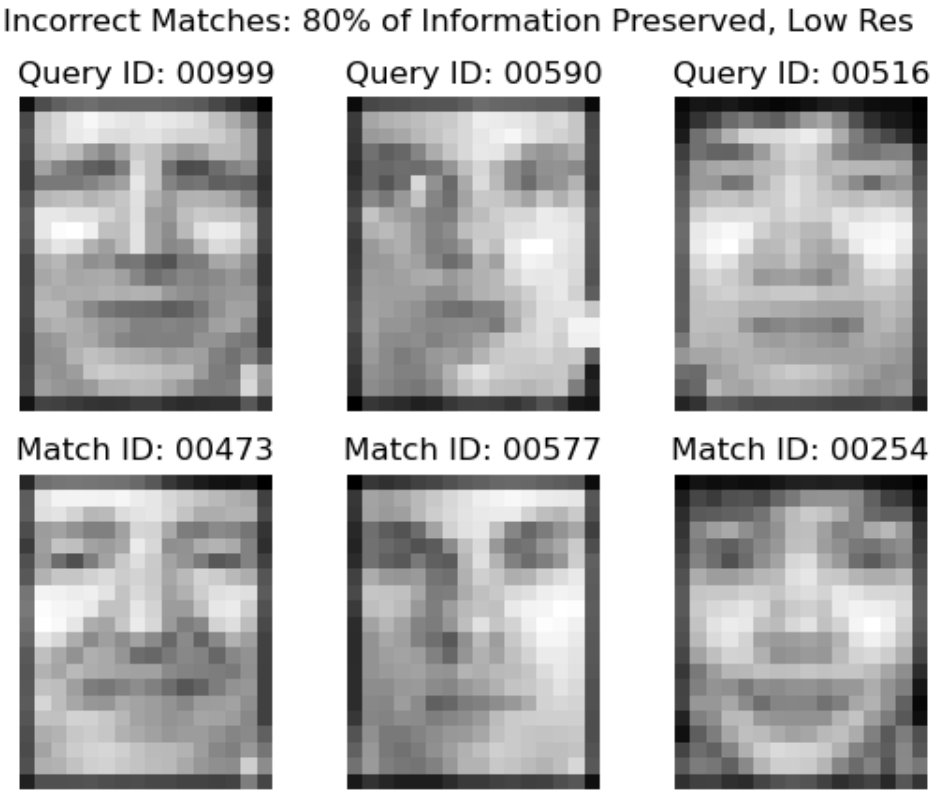
```
k eigenfaces used: 9
```

3.4 (c.III) Assuming r=1, show 3 query images that are correctly matched, along with the corresponding best matched training samples.

[27]:
```
correct_r1s = lowres_80p_df[lowres_80p_df.isMatch_r1==True]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 80% of Information Preserved, Low Res␣
 ↪\n\n\n\n')

    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_c.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
 ↪= plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[correct_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```

3.5 (c.IV) Assuming r=1, show 3 query images that are incorrectly matched, along with the corresponding mismatched training samples.

[28]:
```
incorrect_r1s = lowres_80p_df[lowres_80p_df.isMatch_r1==False]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Incorrect Matches: 80% of Information Preserved, Low Res')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_c.test_images[incorrect_r1s.query_idx.
↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[incorrect_r1s.match_idx.
↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



3.6 (c.V) Repeat (c.II – c.IV) by keeping the top eigenvectors corresponding to 90% and 95% of the information in the data. Plot the CMC curves on the same graph for comparison purposes. If there are significant differences in terms of identification accuracy in (a.II) and (a.V), try to explain why. If there are no significant differences, explain why too.

3.6.1 90% of information in data

[29]:
```
lowres_90p_df = experiment_c.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.9)

lowres_thresh_90percent = []

for i in np.arange(1,51):
    lowres_thresh_90percent.append(np.sum(lowres_90p_df['isMatch_r'+str(i)])/␣
↪len(lowres_90p_df))

print('k eigenfaces used:', experiment_c.k)
```

```
k eigenfaces used: 19
```

[30]:
```
correct_r1s = lowres_90p_df[lowres_90p_df.isMatch_r1==True]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 90% of Information Preserved, Low Res␣
↪\n\n\n\n')

    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_c.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
↪= plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[correct_r1s.match_idx.
↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```

[31]:
```
incorrect_r1s = lowres_90p_df[lowres_90p_df.isMatch_r1==False]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(incorrect_r1s), 3)):
    plt.suptitle('Incorrect Matches: 90% of Information Preserved, Low Res')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_c.test_images[incorrect_r1s.query_idx.
 ↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[incorrect_r1s.match_idx.
 ↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```

3.6.2 95% of information in data

[32]:
```
lowres_95p_df = experiment_c.recognize_faces(n_faces='all', rank=50,␣
↪check_matches=np.arange(1,51), thresh=0.95)

lowres_thresh_95percent = []

for i in np.arange(1,51):
    lowres_thresh_95percent.append(np.sum(lowres_95p_df['isMatch_r'+str(i)])/␣
↪len(lowres_95p_df))

print('k eigenfaces used:', experiment_c.k)
```

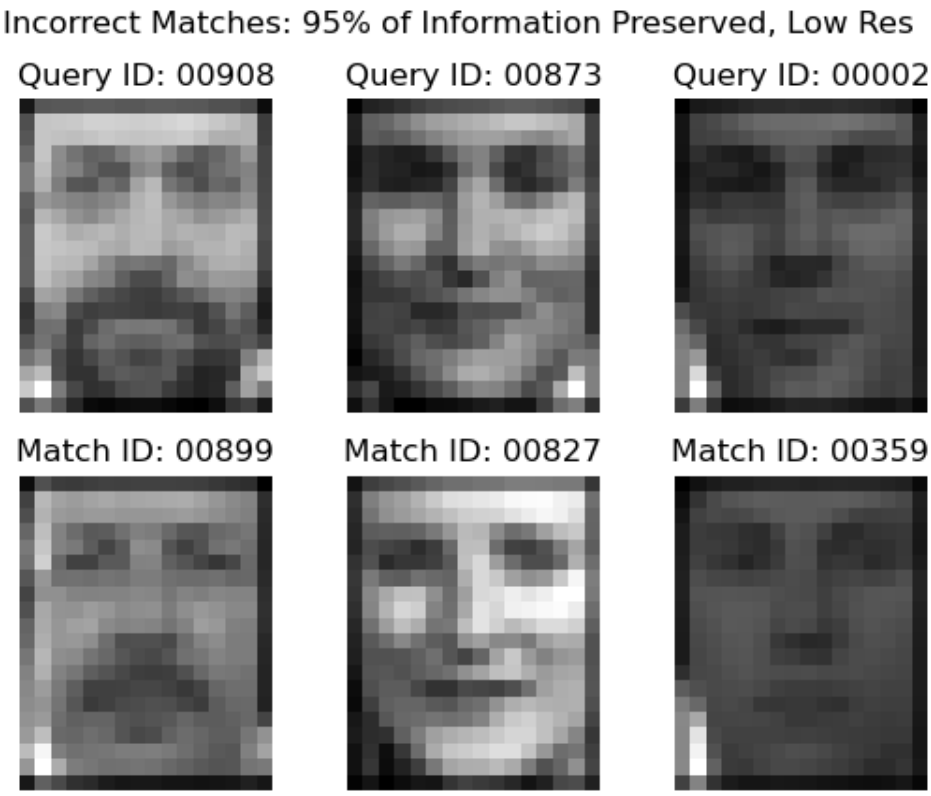
```
k eigenfaces used: 40
```

[33]:
```
correct_r1s = lowres_95p_df[lowres_95p_df.isMatch_r1==True]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(correct_r1s), 3)):
    plt.suptitle('Correct Matches: 95% of Information Preserved, Low Res␣
↪\n\n\n\n')
```

```
    ax[0,j].set_title('Query ID: %s' % correct_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % correct_r1s.best_match.iloc[i][0])

    ax[0,j].imshow(experiment_c.test_images[correct_r1s.query_idx.iloc[i]],cmap␣
↪= plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[correct_r1s.match_idx.
↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```

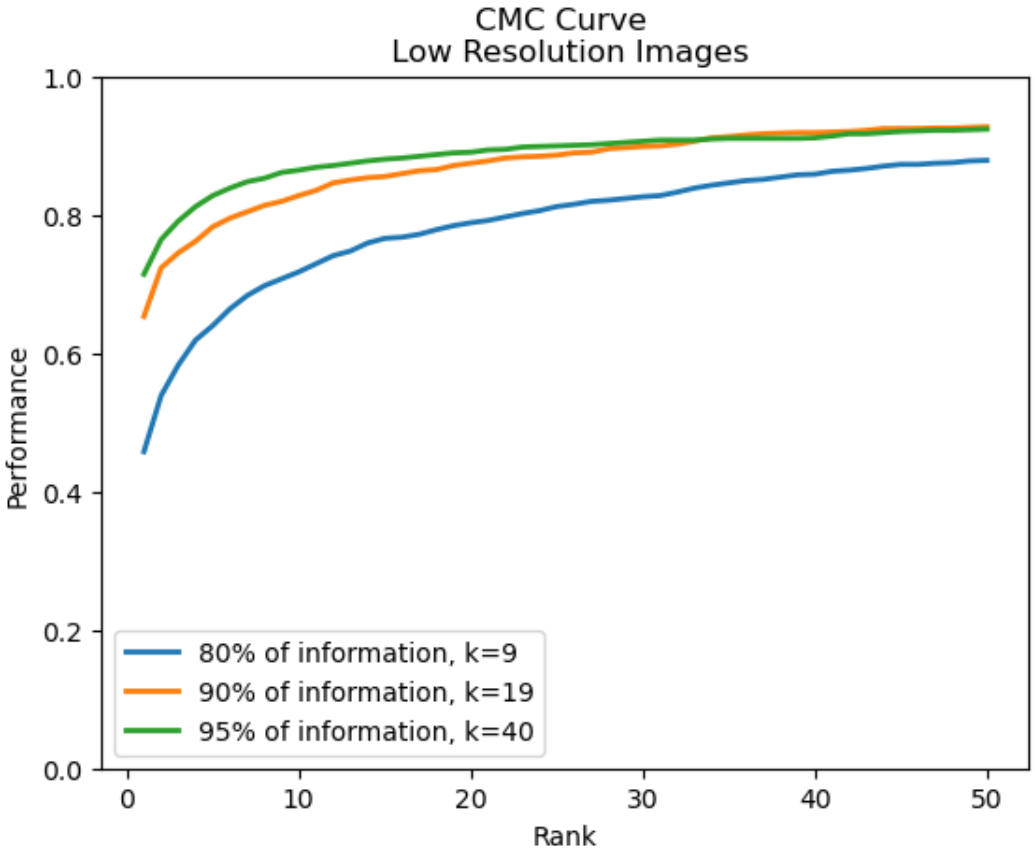
[34]:
```
incorrect_r1s = lowres_95p_df[lowres_95p_df.isMatch_r1==False]
fig,ax = plt.subplots(nrows=2, ncols=3)
for j,i in enumerate(np.random.choice(len(incorrect_r1s), 3)):
    plt.suptitle('Incorrect Matches: 95% of Information Preserved, Low Res')

    ax[0,j].set_title('Query ID: %s' % incorrect_r1s.query_id.iloc[i])
    ax[1,j].set_title('Match ID: %s' % incorrect_r1s.best_match.iloc[i][0])
```

```
    ax[0,j].imshow(experiment_c.test_images[incorrect_r1s.query_idx.
↪iloc[i]],cmap = plt.cm.gray)
    ax[1,j].imshow(experiment_c.training_images[incorrect_r1s.match_idx.
↪iloc[i][0]],cmap = plt.cm.gray)

    ax[0,j].axis('off')
    ax[1,j].axis('off')
```



[35]:
```
plt.title('CMC Curve \n Low Resolution Images')
plt.plot(np.arange(1,51),lowres_thresh_80percent,lw=2, label='80% of␣
 ↪information, k=9')
plt.plot(np.arange(1,51),lowres_thresh_90percent,lw=2, label='90% of␣
 ↪information, k=19')
plt.plot(np.arange(1,51),lowres_thresh_95percent,lw=2, label='95% of␣
 ↪information, k=40')
plt.ylim(0,1)
plt.xlabel('Rank')
plt.ylabel('Performance')
plt.legend()
```

[35]: <matplotlib.legend.Legend at 0x7fd23161eeb0>



4 Experiment (d): Remove all the images of the first 50 subjects from fa_L; let's call the reduced set as fa2_L. Repeat experiment (b) using fa2_L for training (gallery) and fb_L for testing.

[36]:
```
training_dir_reduced_lowres ="Faces_FA_FB/fa2_L/"

experiment_d = EF.EigenFaces(mode='train')

experiment_d.read_images(training_dir_reduced_lowres)
experiment_d.find_eigenfaces()
experiment_d.get_eigen_coefficients()

experiment_d.save_data(filename='training_data_lowres_intruders')
```

```
data saved as file training_data_lowres_intruders.npz
```

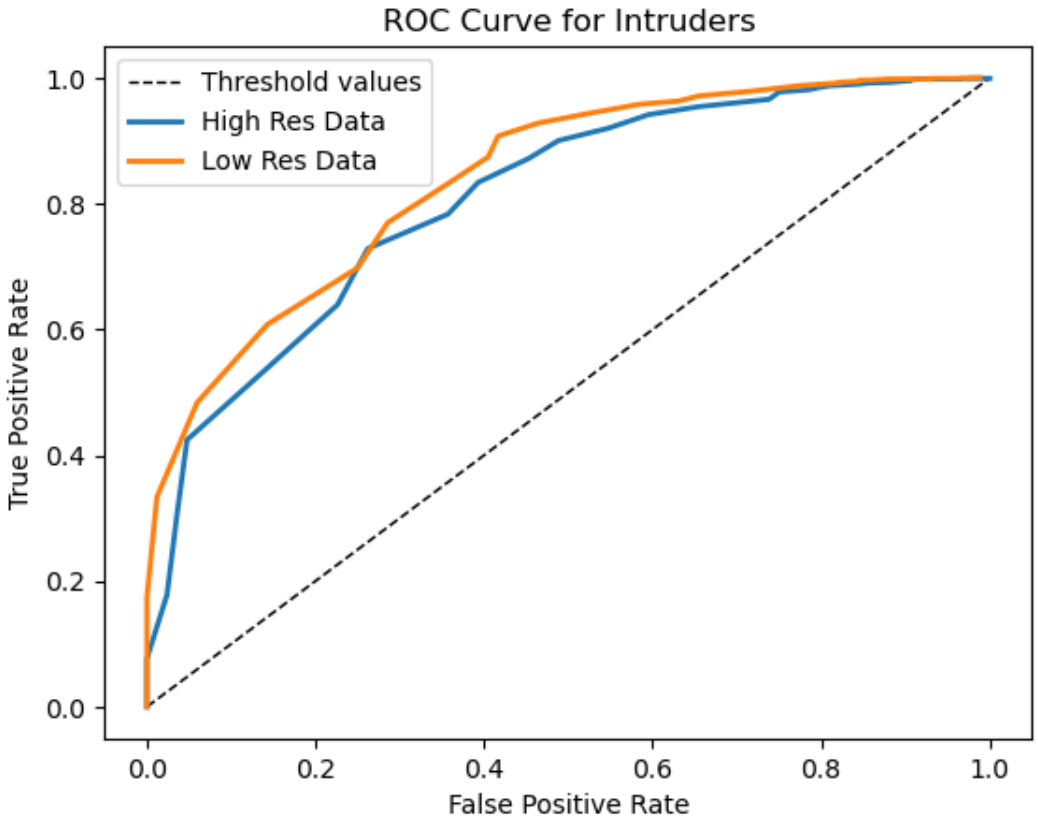
4.1 Perform recognition using fa2_L for training (gallery) and fb_L for testing (query). Since the training set has changed, you would need to compute a new eigenspace for this experiment (i.e., compute the new covariance matrix and its eigenvalues/eigenvectors). Use the eigenvectors corresponding to 95% of the information in the data (i.e., do not experiment with different percentages as in (a)).

[37]:
```
experiment_d.mode='test'
experiment_d.read_images(test_dir_lowres)
experiment_d.get_eigen_coefficients(thresh=0.95)
```

[38]:
```
experiment_d.recognize_faces(n_faces='all', rank=1, check_matches=[1], thresh=0.
 ↪95)
experiment_d.get_intruder_info(training_dir_lowres, n_remove=50)
experiment_d.get_ROC_error_info()
```

[39]:
```
plt.title('ROC Curve for Intruders')
plt.plot(experiment_b.ROC_thresholds, experiment_b.ROC_thresholds,'--', lw=1,␣
 ↪color='k', label='Threshold values')
plt.plot(experiment_b.false_positive_rate, experiment_b.
 ↪true_positive_rate,lw=2, label='High Res Data')
plt.plot(experiment_d.false_positive_rate, experiment_d.
 ↪true_positive_rate,lw=2, label='Low Res Data')

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
```

[39]:
```
<matplotlib.legend.Legend at 0x7fd2487fa580>
```


ROC Curve for Intruders
Threshold values
High Res Data
Low Res Data
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0