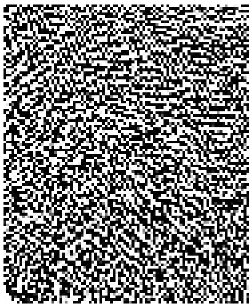
Task 1

Based on the provided code, run the "FaceRecognition.m" file and complete the following points.

1.1 Paste one image of the ATT Face Dataset and the corresponding image after using the 2D Discrete Cosine Transform (DCT)

Firstly, I randomly changed the pre-set image to load and show. The one selected was the one shown in Figure 1

(a) Original Image: *s07/3.pgm*.



(b) DCT applied to original image.

Figure 1: Original face and DCT applied to it.

The image on the right is a representation of the image on the left. This representation si created using the **Discrete cosine transform**, that is, using sums of *cosines*. There are a few different ways of expressing the DCT, but one of the most common ones is:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad \text{for } k = 0, \dots N-1$$

1.2 Using the original configuration parameters (`train = 6` images, `test = 4` images, DCT `coefficients = 10`), plot the resulting DET image and indicate the resulting EER.

We only have to let the whole `FaceRecognition.m` script to execute. At the end, it plots the *DET (Detection Error Trade-Off)* curve, which is a compromise between a False Rejection Rate (*(FRR)*) and a False Acceptance Rate (*FAR*).
The **EER** is the point where $FAR = FRR$, that is, the point where we accept the same percentage of false examples that we reject of positive examples. It is marked with a circle in Figure 2

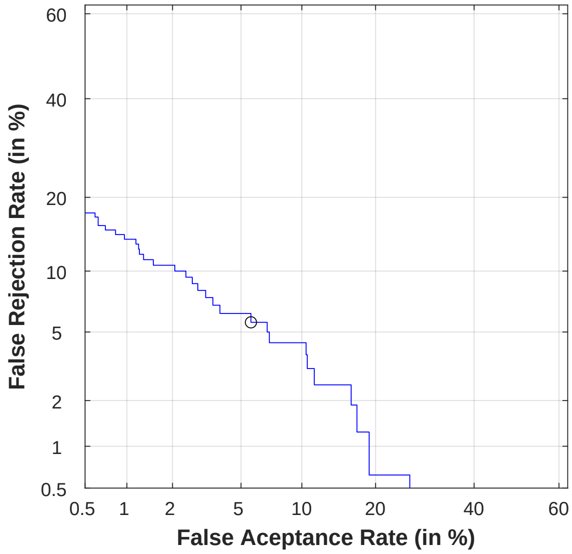


Figure 2: DET curve with marked EER.

We can appretiate in Figure 2 that the EER (circled) is around $\sim 5.5\%$. Also, the code shows on screen that the EER is `EER = 5.6571`.

1.3 Find out the configuration of the DCT coefficients that achieves the best EER results (keeping train = 6 images, test = 4 images). Justify your result, including the resulting DET image and EER value.

In this case where the number of images is small and the techniques that we apply to the image do not take long to execute, we can perform a *grid search* varying the *coefficients* parameter. We have done the following steps:

1. Extracted the important parts of the code file `FaceRecognition.m` and introduced them in a new file (`eer.m`) containing the function `eer(n_train,n_test,param_coeff)` that, given the number of images used for train and test, and given a number of coefficients, computes the EER for those parameters.
2. We have created the script `grid_search_coeff` that executes the code in `eer.m` using the range $\{1, \ldots, 20\}$ for the coefficients and saving the EER result in each case.
3. We plot the results in a chart, remarking the minimum. this is shown in Figure 3.

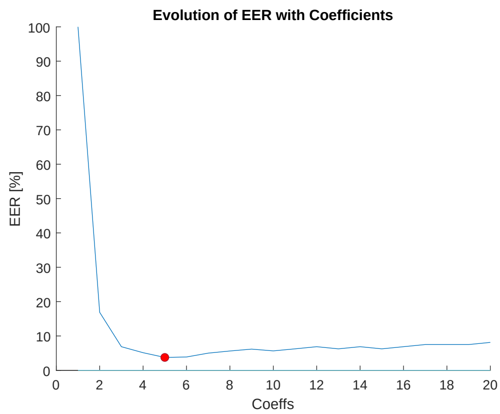


Figure 3: Evolution of the EER with the number of coefficients.

As we can see, the **optimal** number of coefficients is 5, obtaining an $EER = 3.750$. We can appretiate how using a small number of coefficients, the EER is very high (the features are not captured). When using more than 5, the EER slowly starts to increase.

1.4 Once selected the best configuration of the DCT coefficients (in previous point), analyze the influence of the number of training images in the system performance. Include the EER result achieved for each case (from $train = 1$ to $train = 7$). Justify the results.

In this case, we can make good use of our previously defined function `eer` and adapt the code in `grid_search_coeff` to make it vary the parameter `n_train` (and, thus, `n_test` since $n\_test = total - n\_train$).
We create a new script called `grid_search_ntrain` that uses $n\_train = \{1, \ldots, 9\}$ and we run the script, obtaining the graph in Figure

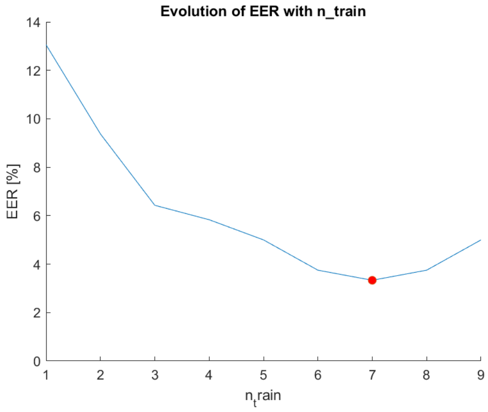


Figure 4: Evolution of the EER with the number of coefficients.

The minimum value of the EER is obtained when we use $n\_train = 7$ images, obtaining $EER = 3.33$, which is a lower value than the one we obtained previously when we optimized the coefficients.

With this and the previous question, what we have done is typically called **grid search**, that is, searching for the best hyperparameters of our model. Usually, the number of images in train and test subsets is not an hyperparameter, since the quantity of available images is much higher, and a partition of $70 - 75\%$ of the size is chosen for train and the rest for the test set. In this case, we reached that the train set has the 70% of the total size of our dataset.
The results are positive, since we have searched for the combination that **optimizes** the EER in this particular problem. As a quick recall, the optimal parameters when we seek to **minimize the EER** are:

1. $n\_train = 7, n\_test = 3$
2. $coefficients = 5$

Task 2

The goal of this task is to change the feature extraction module. Instead of using DCT coefficients as in Task 1, you must consider Principal Component Analysis (PCA) to extract more robust features. You can use the pca.m function available in Matlab. For the training phase, you should follow:

```
[coeff_PCA,MatrixTrainPCAFeats,latent] = pca(MatrixTrainFeats);
meanTrainMatrix=mean(MatrixTrainFeats);
```

It is important to remark that the PCA function must be applied once for all training users and samples (not one PCA per user as this would provide specific `coeff_PCA` parameters per user). For the test phase, you should follow:

> For each test, subtract the meanTrainMatrix, and multiply by the `coeff_PCA` transformation matrix in order to obtain the test features in the PCA domain.

For more information, check Matlab Help: https://es.mathworks.com/help/stats/pca.html

Preparation

Note.- The code of this preparation section can be found in the file `PCA_EER.m`.

To begin with this task, we must adapt the code used before in order to change the way the feature extraction is done. We are told that we must apply PCA as it is usually done: to a $M \in \mathcal{M}_{r\times c}$ matrix with r examples with c features per example. The idea is to reduce that matrix to another matrix $M_{PC} \in \mathcal{M}_{r\times c'}$, where $r' < r$, but keeping the most relevant information about each image.
Thus, we change the way we store the images, we now *flatten* each of them into a single row to create the just mentioned M matrix. We declare in this case a Matrix of size $Train \cdot 40 \times (image\_length \cdot image\_width)$.

```
size = length*width;
%Initialize the Feature and Label Matrix for both train and test
MatrixTrainFeats=zeros(Train*40,size);
MatrixTestFeats=zeros(Test*40,size);
```

Then, we have to flatten (convert each image matrix $M \in \mathcal{M}_{r\times c}$ to a vector in $\mathbb{R}^{r\cdot c}$, and introduce in the corresponding *Train/Test Matrix*. We have slightly changed how the code of the original `FaceRecognition.m` does this part, making it a little bit easier. The result is the following:

```
for j=1:10
im=imread(images(j).name);
im=double(im);
% Flatten image and add it to big matrix
im_flat = reshape(im.',1,[]);

%%%  Training Dataset
if j <= Train
    MatrixTrainFeats((i-1)*Train + j, : ) = im_flat;
    MatrixTrainLabels((i-1)*Train + j, 1) = i;
%%% Test dataset
else
    MatrixTestFeats((i-1)*Test + (j - Train), : ) = im_flat;
    MatrixTestLabels((i-1)*Test + (j - Train), 1) = i;
end
```

Lastly, as we are indicated in the task, we have to extract the principal components of the matrix that contains the features of **all** the images, that is: `MatrixTrainFeats`. The MatLab method `pca` returns the mean μ and the PCA coefficients, so we can use them to project the Test set into the feature space.

```
% PCA on Training matrix
[PCA_coeffs,MatrixTrainPCA,latent,none,explained,mu] = pca(MatrixTrainFeats);
% Project Test Set
MatrixTestPCA = (MatrixTestFeats - mu)*PCA_coeffs;
```

We remark the information contained in each of the returned variables from PCA:

- `PCA_coeffs`$\in \mathcal{M}_{p\times p}$ are the coefficients of the principal components.
- `MatrixTrainPCA` contains the representation of the original data in the principal component feature space.
- `latent` contains the Hotelling's T-squared statistic for each observation in X.
- `explained` contains the percentage of the total variance explained by each principal component.
- `mu` contains the estimated mean of each variable in the original data.

The distance computation used in `FaceRecognition.m` is kept, so we do not modify it. We are now ready to perform the required sub-tasks.

2.1 Using the parameters $train = 6$ and $test = 4$, paste the DET curve and indicate the EER when using all the PCA components.

Using the previously commented code, we run the `PCA_EER` script with the mentioned sizes for train and test subsets and obtain the DET curve plotted in Figure 5. In this subtask, we use **all components** obtained by PCA.

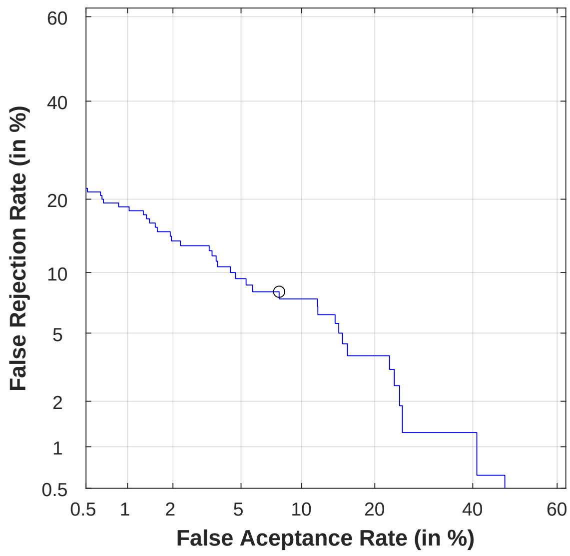


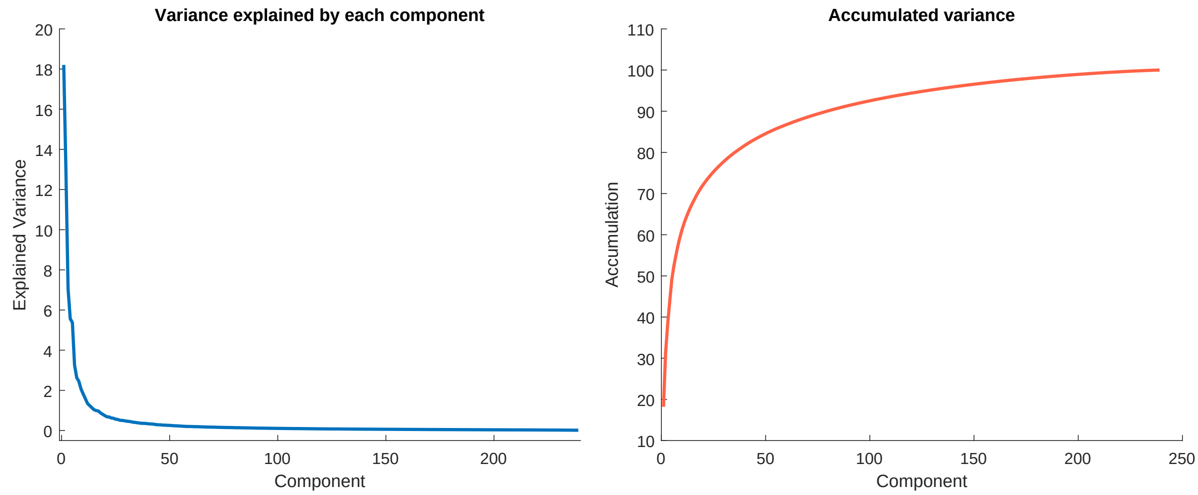
Figure 5: DET curve for PCA with $n\_train = 6$ and $n\_test = 4$, using **all components**.

We obtain an $EER = 8.125$, which is approximately 2.5% higher than the one we obtained in the simplest case using the DCT. Using all the components returned from PCA is not the best option.

2.2 A key aspect for the PCA is the number of components considered. Analyze and represent how the EER value changes in terms of the number of PCA components. Give your explanation about the results achieved.

As we already know, we obtain the principal components by diagonalizing the covariance matrix of the data. This way, we obtain principal directions $v_1, \ldots, v_n$ and eigenvalues $\lambda_1, \ldots, \lambda_n$, which indicate the **percentage** of the variance explained by each component. The components are sorted by the variance explained (its eigenvalue λ_i. The most common technique when using PCA is **selecting** a number of components that explain a decent amount of the variance.

In order to determine how many components we want to select, let us first do an analysis of the variance explained. We find (by obtaining the number of elements of the variable `explained`) that PCA returns 239 components. Let us show in Figure 6 the percentage of variance explained by each of them and the accumulated variance explained by the components $1, \ldots, i$ for each $i = 1, \ldots, 239$.



(a) Variance explained by each component.

(b) Accumulated variance explained.

Figure 6: Explained variance representations.

We can observe in the graphics that using approximately 50 components we are explaining $\approx 80\%$ of the total variance, which is usually enough to be able to classify the images well. Also, from the component 50 onwards, the components explain less than 1% of the total variance, which means that they are possibly not really relevant.

We now compute the EER for each number of components. We create a fragment of code that *iterates* over all the possible number of components (1 to 239) and computes the EER for each of them, and gets the number of components which results in a smaller EER. The result is shown in Figure 7.

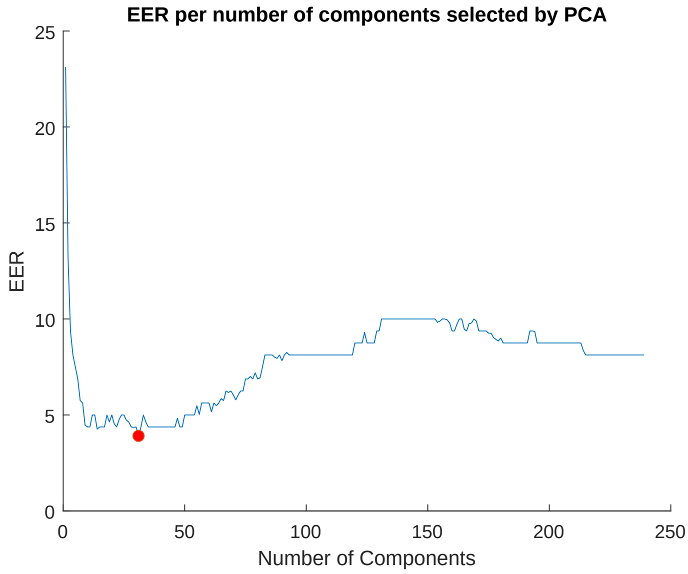


Figure 7: EER obtained using each possible number of components returned by PCA.

As we can observe, using less than 50 components (which is the number of components that can explain approximately 80% of the variance), results in the lowest values of the EER. The EER increases when we increment the number of components, and this **makes sense**, since we have already mentioned that after the $50-$th component, the rest of them explain less than 1% of the variance, which means that they are **not significant**.

All in all, our results in terms of the EER are coincident with our preliminar study of the explained variance by the components.

2.3 Indicate the optimal number of PCA components and paste the resulting DET curve together with the EER achieved. Compare the results using PCA with the DCT features considered in Task 1.

After the loop that iterates through all possible components, we added some code that outputs the minimum of the computed EERs. This code outputs the following:

```
Min EER is obtained with 31 components
Minimum EER is  3.910256e+00
Explained Variance 7.826086e+01
```

As we can see, we obtain that selecting **31 components** results in $EER = 3.91$, which is the minimum that we can obtain using PCA. Also, we are explaining approximately a 78% of the total variance, which is quite a high quantity. We **state** that the optimal number of PCA components to use in this problem is 31. The resulting DET curve is plotted in Figure 8.

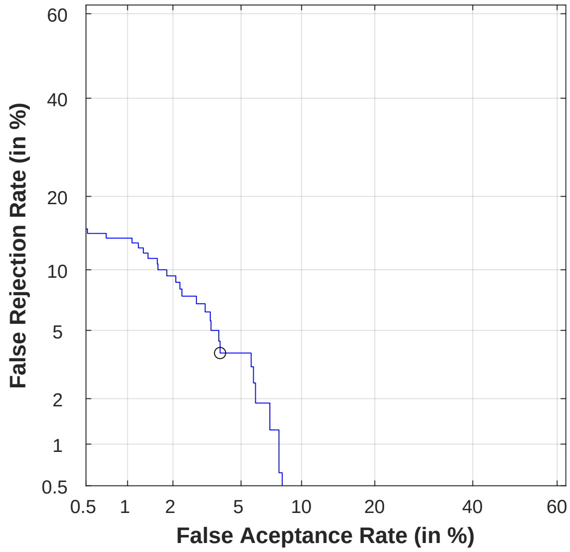


Figure 8: DET curve for $n\_components = 31$ in PCA feature selection.

2.3.1 Comparison with the previous method

Now we want to compare the results obtained using PCA with the results obtained using the feature extractor that used the DCT. For PCA we used a fixed $n\_train = 6$ and $n\_test = 4$. Remember that, in the DCT method, the best results were achieved using $n\_train = 7, n\_test = 3$. However, we have tested this configuration (and also $n\_train = n\_test = 5$) in the PCA feature extraction, but the results have only gotten worse (we obtained higher values for the EER).
With this in consideration, we have resumed all the information until this point in Table 1. For each of the feature extraction methods, we include the original hyperparameters and the optimal hyperparameters selection.

Method/Hyperparameters	Starting	Best
DCT	5.5	3.33
PCA	8.125	3.91

Table 1: Comparison of the results obtained with both used methods.

PCA obtains an EER 0.6% higher than the one obtained by using the DCT, which is worse than the first case but not really in a determinant quantity. However, we know that PCA focus on explaining the variance of the data, so the extracted features might be more relevant when generalizing this code for a larger dataset, so we would **choose PCA** as the most relevant way of extracting features.

Extra Task

The goal of this task is to improve the matching module. Instead of using a simple distance comparison, you must consider Support Vector Machines (SVM). In this case, you should train one specific SVM model per user using the training data (train = 6 images). Features extracted using the PCA module developed in Task 2 must be considered in this Task. You can use the fitcsvm function available in Matlab. For the training phase, you should follow:

```
SVMModel = fitcsvm(...)
```

For the test phase, you should follow:

```
[label,score]= predict(SVMModel,MatrixTestFeats);
```

to obtain the scores for each user model. For more information, check Matlab Help: https://es.mathworks.com/help/stats/fitcsvm.html?lang=en

3.1 Using the parameters train = 6 and test = 4, paste the DET curves and indicate the EERs in the following cases: 1) regarding the KernelFunction parameter of the SVM (using all PCA components), and 2) regarding the number of PCA components considered for the feature extraction module (using the KernelFunction polynomial and starting with 3 PCA components).

3.1.1 Using the KernelFunction

3.1.2 Changing the number of selected PCA components