Pattern recognition lab 2

Name	ID
Amin Mohamed Amin El-Sayed	21010310
Rafy Hany Saied Ansi	21010504
Omar Al-Dawy Ibrahim Al-Dawy	21010864

- ## Problem Statement
  - We intend to perform **face recognition**. Face recognition means that for a given image you can tell the subject id. Our database of subjects is very simple. It has 40 subjects. Below we will show the needed steps to achieve the goal of the assignment.

- ## Data Setup
  - Name: **Database of Faces** (formerly ORL Database of Faces).
  - Time Period: Images taken between April 1992 and April 1994.
  - Subjects: **40 distinct people**.
  - Images per subject: **10 images**.
  - Variations:
    - Different times, lighting, facial expressions (eyes open/closed, smiling/not smiling), and facial details (glasses/no glasses).
  - Image Format: PGM (Portable Gray Map).
  - Image Size: **92×112 pixels**.
  - Color: Grayscale **(256 levels)**.
  - Organization:
    - **40 folders (s1, s2, ..., s40)**.
    - **Each folder: 10 images** (1.pgm, 2.pgm, ..., 10.pgm).
  - The data matrix **D** has **shape (400, 10304)**, where each row is a flattened grayscale face image of size 92×112 pixels.
  - The label vector **y** has **shape (400,)**, with each entry being an integer from 1 to 40 representing the subject ID for each image.
  - The dataset is split by taking **even-indexed images for the training set** and **odd-indexed images for the test set**, giving a 50%-50% split for each subject.

- ## PCA Implementation
  - Principal component analysis (PCA) is a **dimensionality reduction** and machine learning method used to simplify a large data set into a smaller set while still maintaining significant patterns and trends.
  - Main purpose is to minimize the **reconstruction error.**

- Steps:
    - **Subtracting the mean** of X from each sample to center the data.
    - Computing the **covariance** of the centered data
    - Calculating **eigenvalues and eigenvectors**, and **sorting** them in descending order.
    - Select the **number of components to keep** based on the specified **threshold** (e.g., 95% of variance).
    - **Project** both **training and testing data** onto the selected principal components.
    - **Reconstruct the images** from the reduced data and visualize how well they approximate the originals.

- **K-Means Clustering**

    - **K-Means Clustering** is an unsupervised machine learning algorithm used to partition data into K distinct clusters based on similarity. It is widely used in:

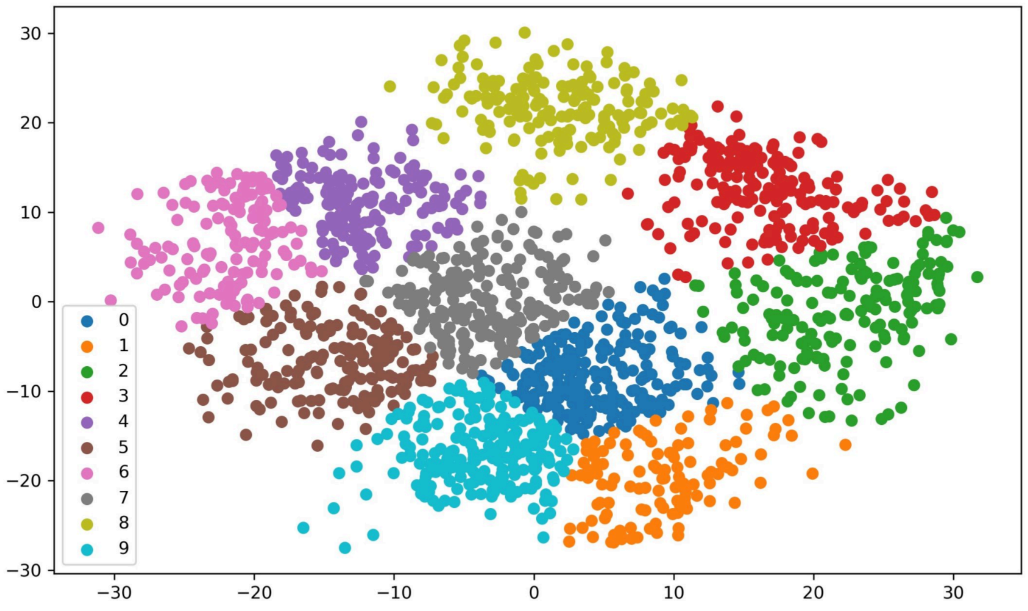
        - ➢ Customer segmentation
        - ➢ Image compression
        - ➢ Anomaly detection
        - ➢ Feature engineering

      The algorithm works by iteratively assigning data points to the nearest cluster centroid and updating centroids until convergence.

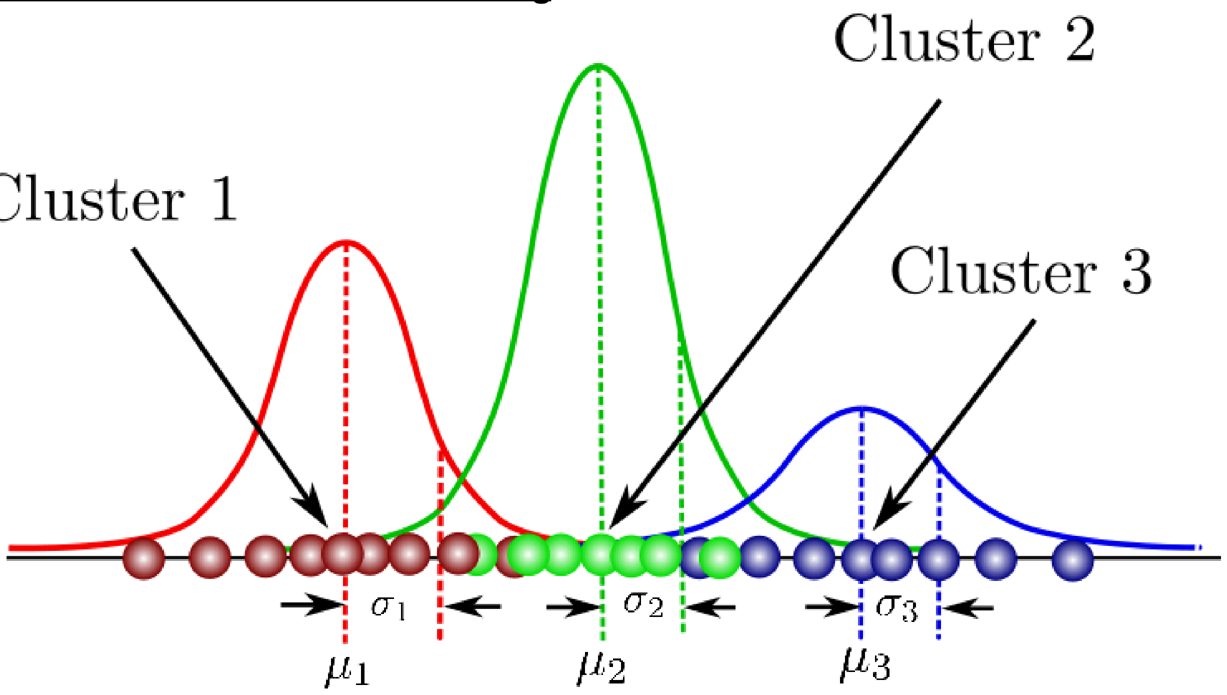
    - **Steps**

        - ➢ **Initialization** :
            - Choose the number of clusters (K). Randomly initialize K centroids (cluster centers).
        - ➢ **Assignment Step** :
            - Assign each data point to the nearest centroid (typically using Euclidean distance). Each point belongs to the cluster with the closest centroid.
        - ➢ **Update Step**:
            - Recompute the centroids as the mean of all points in each cluster. The centroid moves to the center of its assigned points.

- **Convergence Check** :
    - Repeat Steps 2 & 3 until: Centroids stop moving significantly (tolerance threshold reached). Maximum iterations completed.
- **Output Final cluster assignments :**
    - for all data points. Optimized centroids representing each cluster.



- **<u>Gaussian Mixture Model Clustering</u>**



  - **<u>GMM in nutshell</u>**

    A Gaussian Mixture Model is a probabilistic model and an example of Expectation-Maximization techniques that assumes data points are generated from a mixture of several Gaussian distributions (also called normal distributions) with unknown parameters. Each Gaussian represents a cluster in the data, but unlike hard clustering (e.g., k-means), GMM softly assigns each data point to all clusters with certain probabilities.

  - **<u>Initialization</u>**

    The algorithm does not guarantee a global optimal solution so the final result depends on the initial conditions as it will reach a global or local optimal answer.
    The initial conditions are the means of each gaussian and its covariance and there are many ways to choose them
    1. Choose random samples to be your means.
    2. Apply K-means and make the resultant centroids the initial guess for means
    3. Choose the covariance to be an identity matrix
    4. Choose the covariance to be the data covariance

    In our implementation we used random random samples to be the means or random samples to compare them and for the covariance we used the data covariance for all gaussians.
  - **<u>Expectation Step</u>**

    We need to calculate the responsibility for each sample with each gaussian we have using the following rule

$$\Pr(z^{(i)} = k \mid \mathbf{x}^{(i)}) = \frac{\Pr(z = k)\, p(\mathbf{x} \mid z = k)}{\sum_{\ell} \Pr(z = \ell)\, p(\mathbf{x} \mid z = \ell)}$$

Where p(x|z=k) is calculated by

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

but for a more numerically stable version we take the Ln of the equation so it will be as follows

$$\log r_{ik} = \log(\pi_k) + \log \mathcal{N}(x_i \mid \mu_k, \Sigma_k) - \log\left(\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)\right)$$

Where ln(p(x|z=k)) is calculated by

$$\log \mathcal{N}(x_i \mid \mu_k, \Sigma_k) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)$$

Then at the end we remove the ln by taking exponent power for each value.

- **Maximization Step**

The target here is to update the parameters to get better results by making the gaussians fit the data better than the previous step.
We update three parameters gaussian weight (π), mean (**μ**) and covariance (**Σ**)
Using these formulas

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{i=1}^{N} r_k^{(i)} \mathbf{x}^{(i)}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k}\sum_{i=1}^{N} r_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^{\top}$$

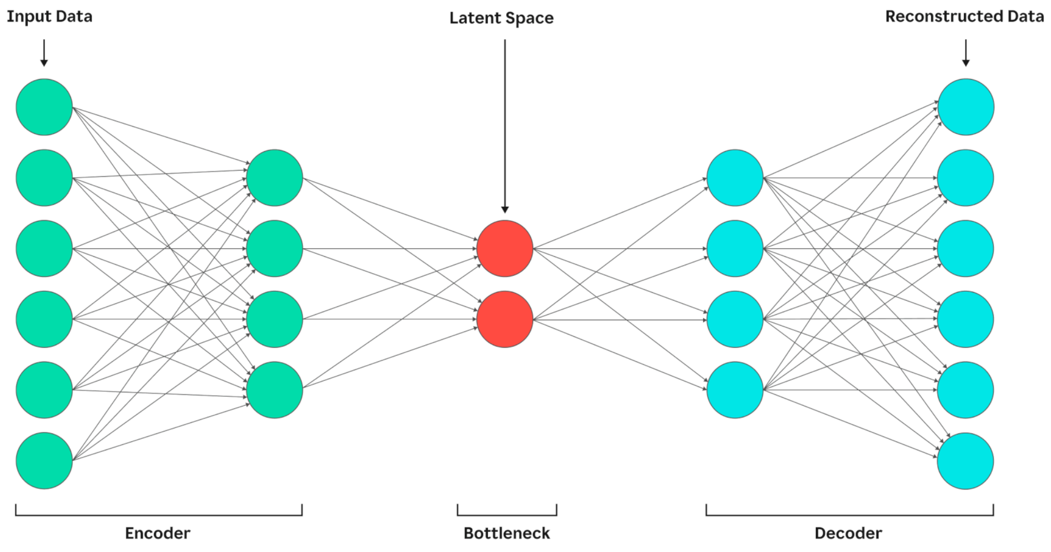
$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{i=1}^{N} r_k^{(i)}$$

- **Convergence criteria**

  We use the log likelihood criteria to detect the convergence of the model as if the absolute difference of the current log likelihood and the previous one is smaller than an epsilon value then the model has converged to an optimal solution whether it is global or local.

$$\log p(\mathbf{X} \mid \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(i)} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

- **Autoencoder**
  - An **Autoencoder** is a type of **neural network used for unsupervised learning**, typically for **dimensionality reduction, data compression, or feature learning**. It consists of two parts:
    - **Encoder**: Compresses the input into a lower-dimensional representation (encoding).
    - **Decoder**: Reconstructs the original input from this lower-dimensional encoding.
  - The goal of the autoencoder is to minimize the difference between the input and its reconstructed output.



  - Input_dim: Number of input features
  - Hidden_dim: Number of neurons in the hidden layer
  - Encoding_dim: Number of neurons in the bottleneck layer

- **Encoder**:
    - a **fully connected layer** reduces the input dimensions to **hidden_dim**.
    - a **ReLU** activation is applied for non-linearity.
    - a second layer further reduces the data from **hidden_dim to encoding_dim**

- **Decoder**:
    - It takes the compressed data (encoding) of size encoding_dim and expands it back to **hidden_dim**.
    - A **ReLU** activation is applied.
    - it maps the data back from **hidden_dim to the original input_dim**.

- **Forward Pass**:
    - Takes an input tensor x, passes it through the **encoder** to get the encoding, and then passes the encoding through the **decoder** to reconstruct the input.
    - The output is the reconstructed data, which is compared to the original data during training.

- **Evaluation**
  - **Using PCA**
    - **K-Means**
      - **Training**

K / Alpha	0.8	0.85	0.9	0.95
20	0.420	0.415	0.4	0.41
40	0.625	0.625	0.6	0.605
60	0.785	0.76	0.75	0.735

Best model with k = 60 and alpha = 0.8

      - **Testing**
        Test Accuracy: 0.765
        Test F1-score: 0.75260
    - **GMM**
      - **Training**
        - **Using Random Initialization**
          Accuracy:

K / Alpha	0.8	0.85	0.9	0.95
20	0.4	0.365	0.29	0.225
40	0.58	0.525	0.47	0.375
60	0.725	0.695	0.61	0.525

Best model with K = 60 and Alpha = 0.8

- **<u>Using K-means Initialization</u>**

  Accuracy:

K / Alpha	0.8	0.85	0.9	0.95
20	0.435	0.41	0.395	0.41
40	0.725	0.73	0.725	0.73
60	0.88	0.855	0.885	0.855

Best model with K = 60 and Alpha = 0.9

- **<u>Testing</u>**
  - **<u>Using Random Initialization</u>**

    Accuracy = 0.71

    F1-Score = 0.6794
  - **<u>Using K-Means Initialization</u>**

    Accuracy = 0.8

    F1-Score = 0.7971

- **<u>Using Autoencoder</u>**
  - **<u>K-Means</u>**
    - **<u>Training</u>**

K	20	40	60
Accuracy	0.445	0.62	0.745

Best model with k = 60

- **<u>Testing</u>**

  Accuracy: 0.7

  F1-Score: 0.6801

- **<u>GMM</u>**
  - **<u>Training</u>**
    - **<u>Using Random Initialization</u>**

K	20	40	60
Accuracy	0.375	0.525	0.735

Best model with K = 60

- **<u>Using K-Means</u>**

K	20	40	60
Accuracy	0.46	0.695	0.8

Best model with K = 60

- **<u>Testing</u>**
  - **<u>Using Random Initialization</u>**
    Accuracy = 0.705
    F1-Score = 0.6593
  - **<u>Using K-Means</u>**
    Accuracy = 0.69
    F1-score = 0.6622

➢ **<u>Comparison between PCA and Autoencoder</u>**
We have seen that PCA gave better accuracy than Autoencoder so this might be as our data is better to be projected on a linear space rather than a non-linear space.

➢ **<u>Comparison between K-Means and GMM</u>**
As we have shown that GMM with random initialization is worse than k-means and k-means is worse than GMM with k-means as initialization.

➢ **<u>Relation between alpha and GMM and K-means</u>**
From previous results. The increase in alpha the decrease in accuracy due to :
- ❖ Due to the curse of dimensionality : as the dimensions increase, the distance tends to infinity, making it very huge, so the variance in the same cluster is huge.
- ❖ PCA with alpha=0.95 may be too aggressive, reducing dimensions to a point where: Cluster separation becomes ambiguous in the compressed space. Critical variance for distinguishing faces is lost.
- ❖ Alpha=0.8 strikes a balance: Reduces dimensions enough to avoid the curse of dimensionality. Preserves sufficient structure for K-Means to work effectively.

➢ **<u>Relation between k and GMM and K-means</u>**
From previous results. The increase in K the increase in accuracy due to :
- ❖ With more clusters: Each cluster becomes "purer" (dominated by one true class). The majority-voting accuracy metric benefits from finer-grained groupings.