PRML Project: Face Identification Report

Varshit Manikanta Atharva Deshpande Dishit Sharma Vishesh Sachdeva
Jinang Shah Harish Kumar Ashish Aditya

Indian Institute of Technology, Jodhpur
{b22ai038, b22ee013, b22cs082, b22ai050, b22cs027, b22ee029, b22ee042}@iitj.ac.in

Abstract

Face recognition technology has witnessed significant advancements in recent years, driven by the proliferation of deep learning techniques, improved computational power, and the availability of large-scale labeled datasets. This paper presents a comprehensive review of the different methods through which facial recognition can be achieved, covering feature extraction and classification algorithms. In this project, the dataset chosen is the LFW dataset, upon which we have performed exploratory data analysis to have a complete understanding about the dataset. We have extensively worked upon feature extraction of the LFW dataset using methods such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Convolutional Neural Networks (CNN). The extracted features were fed into multiple classifiers such as Logistic Regression, Support Vector Machines (SVM) and Random Forest algorithm, and the accuracy was noted. The aim of this project is to find the best feature extraction method which leads to the best accuracy on multiple classification techniques, and eventually train a model that would accurately recognize the input face image.

Contents

1 Introduction

Labeled Faces in the Wild (LFW) is an image dataset containing face photographs, collected especially for studying the problem of unconstrained face recognition. It includes over 13,000 images of faces collected from across the web. This project helps us study different feature extraction techniques and the accuracy achieved on various classifiers on the LFW dataset.

1.1 Exploratory Data Analysis (EDA)

The LFW dataset is fetched using the `sklearn.datasets` library. For EDA purposes, we keep the `'min_faces_per_person'` parameter as 40.

We get the following information:

```
Number of samples:  1777
Number of features:  1850
Number of classes:  17
```

Dataset Visualisation:

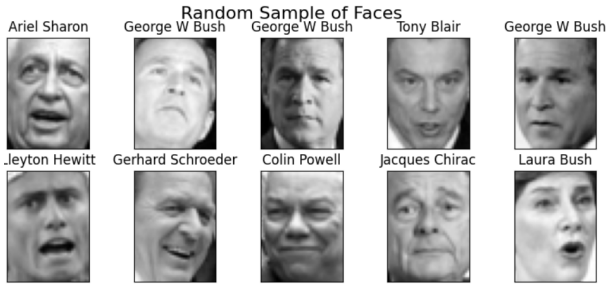

Figure 1: Plotting random faces from the dataset

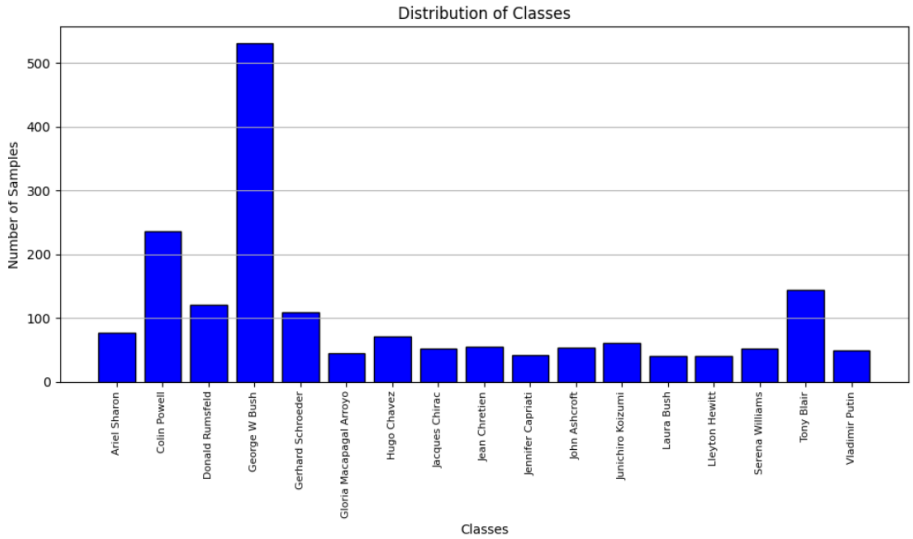

Figure 2: Distribution of classes

We can see here that most of the images belong to the class 'George Bush', followed by 'Colin Powell' with a significant drop.

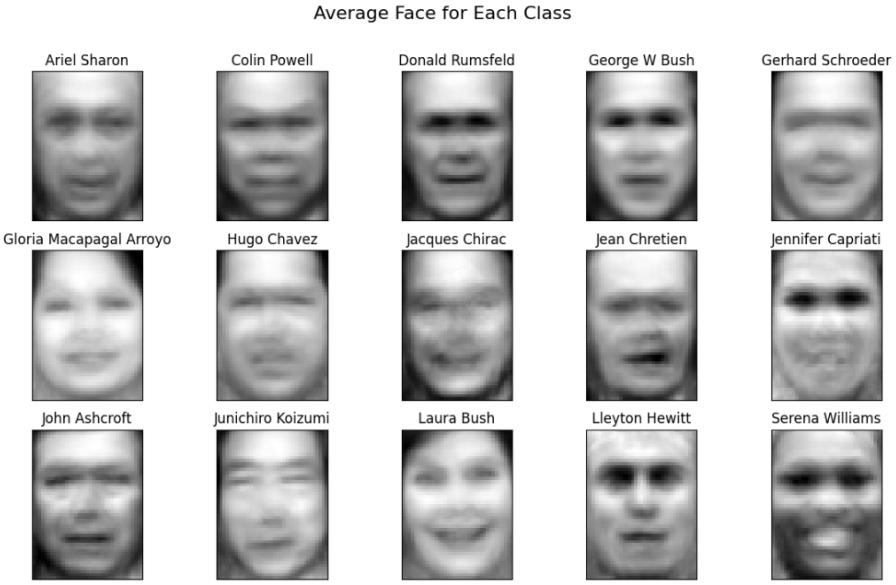

Figure 3: Plotting the average face for each class

The plotted images reflect the typical visual appearance across the entire dataset of images. It represents the central tendency of pixel values in terms of color, brightness, and texture.

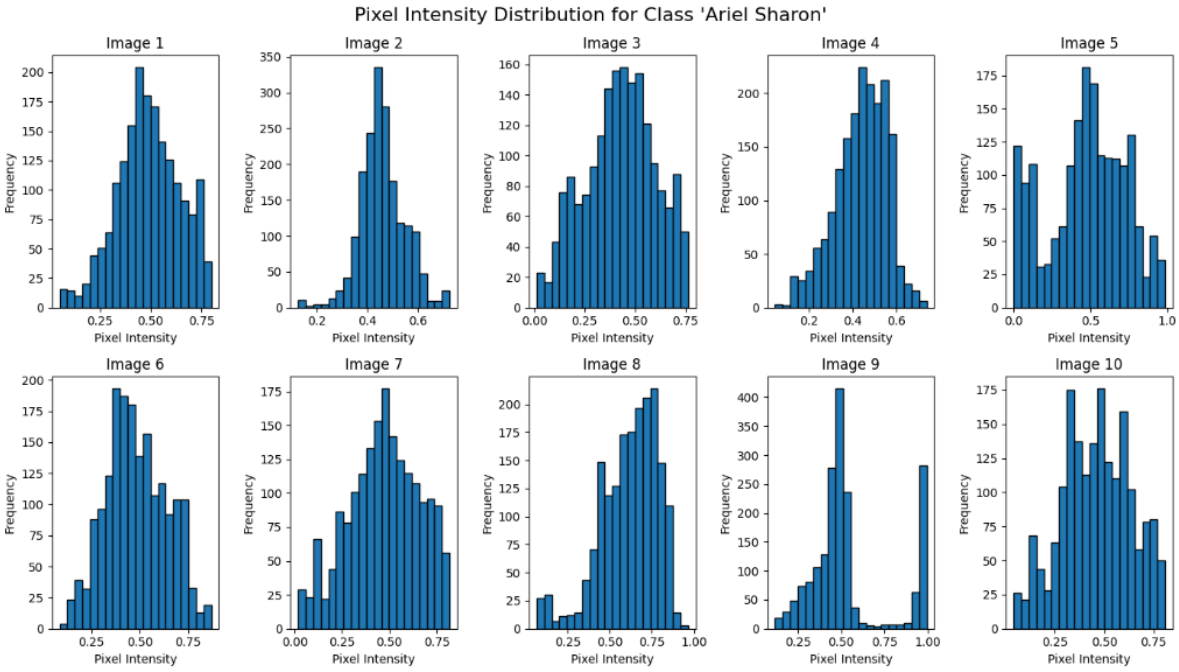

Figure 4: Pixel intensity distribution for a specific class

We can see that for the class 'Ariel Sharon', distribution of pixel intensities in the images is such that most of the pixels lie within the range of 0.3 to 0.6.

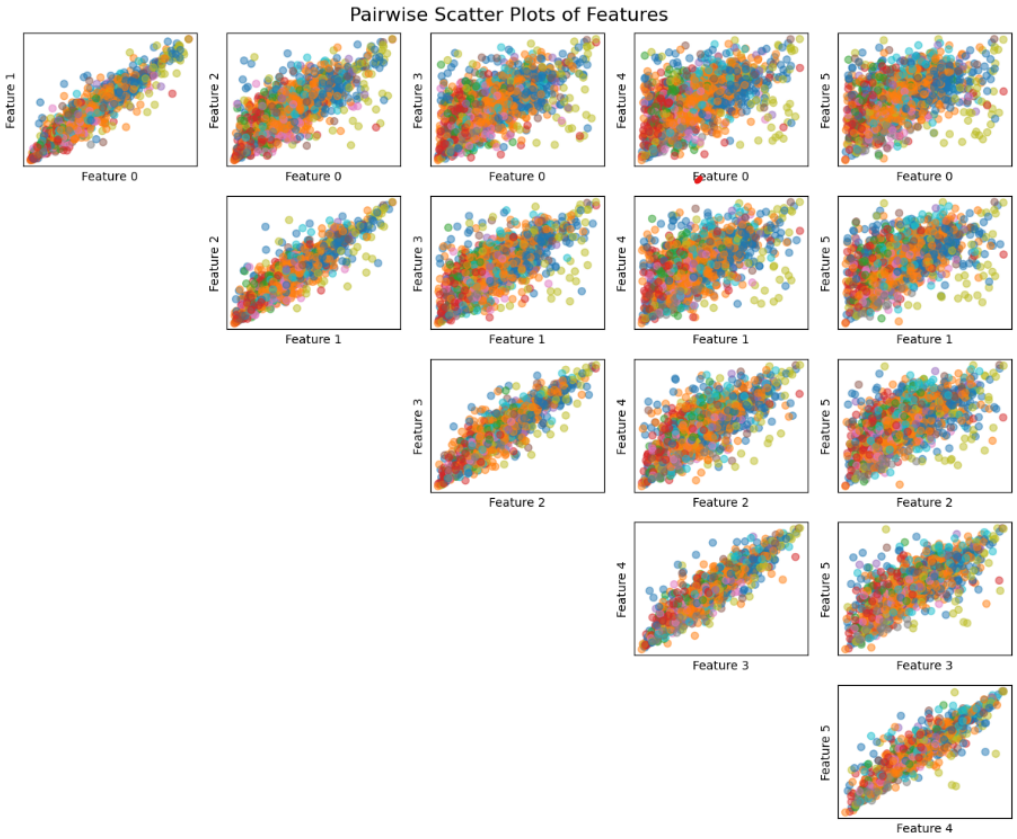

Figure 5: Pairwise scatter plots examining relationships between pairs of features

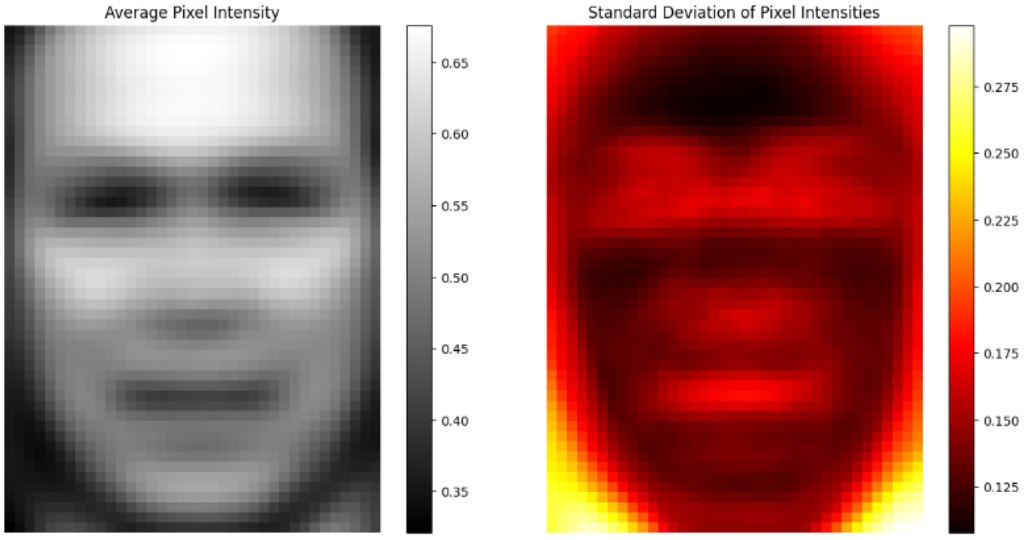

Figure 6: Plotting average pixel intensity and standard deviation side by side

2 Approaches Tried

We tried the following approaches for feature reduction (dimensionality reduction):

1. **Principal Component Analysis (PCA):** 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. The idea of PCA is simple: reduce the number of variables of a data set, while preserving as much information as possible.
2. **Linear Discriminant Analysis (LDA):** LDA is a dimensionality reduction algorithm that projects data points into one dimension and clusters them by maximising the between-cluster distances and minimising the within-cluster distances.

We tried the following approaches for feature extraction:

1. **Convolutional Neural Network (CNN):** Convolutional Neural Networks (CNNs) are deep learning models designed for processing structured grid data like images. They employ convolutional layers to detect patterns hierarchically, pooling layers to reduce dimensionality, and fully connected layers for classification. CNNs excel in tasks like image recognition, object detection, and image segmentation.
2. **Local Binary Pattern (LBP):** Local Binary Pattern (LBP) is a texture descriptor for analyzing local patterns in images. It encodes the relationship between a pixel and its neighbors by comparing intensity values. LBP extracts texture features robust to illumination changes and is widely used in facial recognition, texture classification, and object detection tasks.
3. **Histogram of Gradients (HoG):** Histogram of Gradients (HoG) is a feature descriptor technique for object detection in images. It calculates gradients' magnitude and orientation in small regions, then constructs a histogram of gradient orientations to represent local shape and appearance. HOG is commonly used in pedestrian detection, human pose estimation, and object recognition tasks.

For classification, we used the following classifiers:

1. **Random Forest**
2. **Logistic Regression**
3. **Support Vector Machines (SVM)**
4. **Artificial Neural Network (ANN)**

3 Experiments

3.1 Linear Discriminant Analysis (LDA)

LDA was imported from the `sklearn.discriminant_analysis`. We created a class called lda which had the following features:

1. **__init__(self, n_components)**: Initializes the LDAclass object with a specified number of components (`n_components`).
2. **fit_transform(self, X_train, Y_train)**: Computes the Linear Discriminant Analysis (LDA) transformation on the training data X_train with corresponding labels Y_train. Calculates the within-class scatter matrix (Sw) and between-class scatter matrix (Sb) based on the input data and labels. Computes the eigenvectors and eigenvalues of the matrix product `np.linalg.inv(Sw) @ Sb`. Sorts the eigenvectors and eigenvalues in descending order of magnitude. Selects the principal components corresponding to the highest eigenvalues, as determined by `n_components`. Projects the training data onto the selected principal components. Prints the eigenvalues of the first 10 components (for diagnostic purposes). Returns the transformed data `X_projected`.
3. **transform(self, X):** Transforms the input data X using the previously computed principal components. Projects the input data onto the selected principal components. Returns the transformed data.
4. **fischerfaces(self):** Reshapes the principal components into Fischerfaces format. Fischerfaces are a visualization technique for the principal components, especially in face recognition tasks. Reshapes the principal components to the shape (number of components, 16, 12), assuming a specific image size. Returns the reshaped principal components for visualization.

We used the following classifiers on LDA projected features:

1. **ANN**
2. **Random Forest**
3. **SVM**

3.2 Principal Component Analysis (PCA)

A class called PCA is created having the following functions:

1. **__init__:** Initializes PCA object and stores the `n_components`. Also initializes empty variables `eigenvalues`, `eigenvectors`, and `means`.
2. **fit:** Calculates the covariance matrix, then obtains eigenvectors and eigenvalues, providing insights into feature contributions to total variance.
3. **fit_transform:** Returns the given array projected onto principal components.
4. **explained_variance:** Returns the first `n_components` number of eigenvalues in decreasing order.
5. **explained_variance_ratio:** Returns the ratio of variance captured by the first `n_components` number of eigenvalues in decreasing order.
6. **components:** Returns principal components.
7. **get_eigenvalues:** Returns all eigenvalues.
8. **get_eigenvectors:** Returns all eigenvectors.

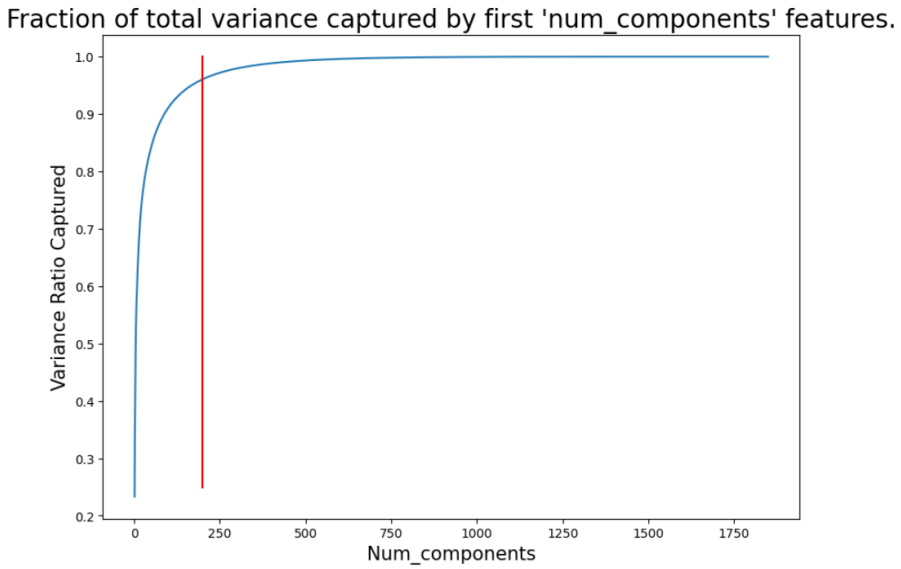

Figure 7: Fraction of total variance captured by first '`num_components`' features

Only about 230 features are responsible for more than 95% of the total variance.

Plotting the first 10 eigenfaces:

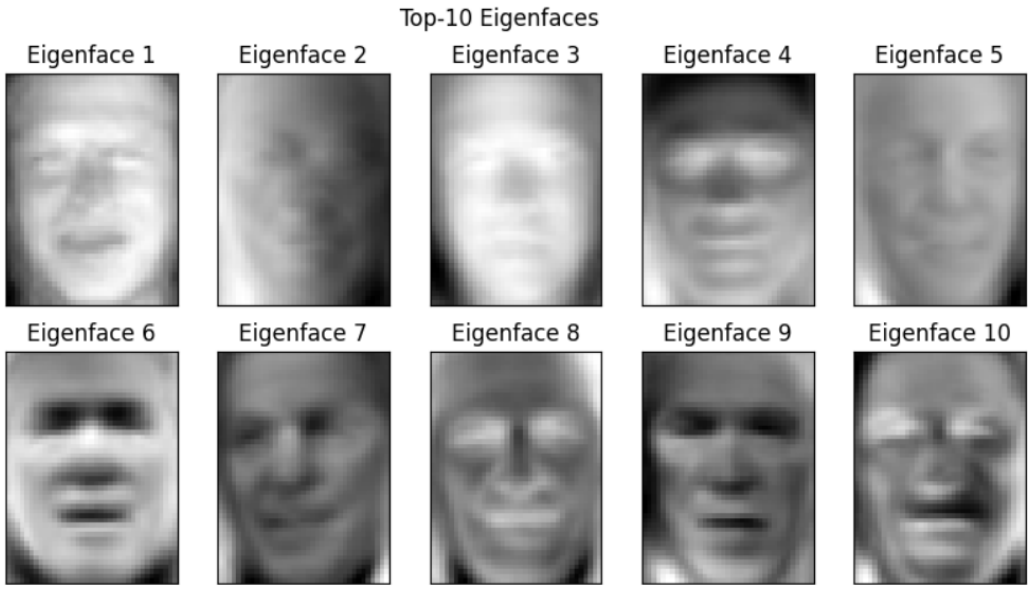

Figure 8: First 10 eigenfaces

We used 200 such eigenfaces to reconstruct the input face image:

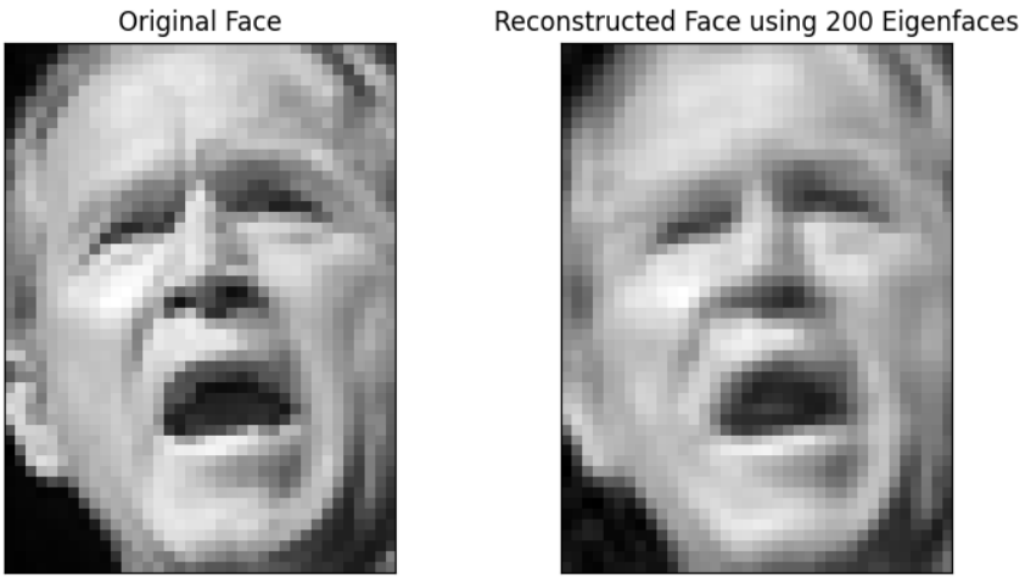

Figure 9: Original and reconstructed faces

On the PCA projected dataset, we ran the following classifiers:

1. **Artificial Neural Network (ANN)**
2. **Random Forest**
3. **Support Vector Machines (SVMs)**

3.3 Convolutional Neural Network (CNN)

Feature extraction of LFW dataset images using CNN consists of the following steps:

1. **Fetching dataset:** Dataset was fetched from the `sklearn.datasets` library.
2. **Model Building:** Here, we used a pretrained model called resnet50 from the `torchvision.models` library. ResNet50 was chosen due to its unique feature of skip connections to dodge traditional problems faced by deep neural networks. The skip connections allow for better feature reusability, improved training convergence and a state-of-the-art performance on various image classification benchmarks. This is why we chose to use ResNet50 as our model. The last layer is removed to get a feature vector instead.
3. **Feature Extraction Function:** The image pixel values are between 0 to 1, which makes the image incompatible for PIL array to Image conversion. Thus the image pixel values are restructured before being preprocessed to meet the input requirements of the ResNet model. Then the features are extracted and returned as output.
4. **Batch Division:** A batch size of 28 is decided and the images are split accordingly.
5. **Feature extraction:** The features are extracted and saved as a .pt file so that they can be used later.

```
(2): ReLU(inplace=True)
(3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
(4): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
```

Figure 10: Glimpse of ResNet Architecture

We used the following classifiers on the CNN extracted features:

1. **Logistic Regression**
2. **Support Vector Machines (SVMs**

3.4 Local Binary Pattern (LBP)

We have done the following in order to extract features using LBP:

1. **Neighborhood Intensity Comparison:** The code defines a function `get_pixel` to compare the intensity value of a pixel with its neighboring pixels. If the intensity of the neighboring pixel is greater than or equal to the intensity of the center pixel, it assigns a value of 1, otherwise, it assigns 0. This function is used to create a binary pattern for each pixel based on its local neighborhood.
2. **Local Binary Pattern (LBP) Calculation:** The function `lbp_calculated_pixel` calculates the Local Binary Pattern (LBP) value for a given pixel location in the image. It computes the LBP value by comparing the intensity of the center pixel with its neighboring pixels in a circular pattern. The binary values obtained from the comparison are multiplied with a sequence of powers of 2, whose summation results in a single integer value representing the LBP pattern for that pixel.
3. **Visualization:** The lbp functions are used to calculate the extracted features of a sample image. The LBP image is then visualized.

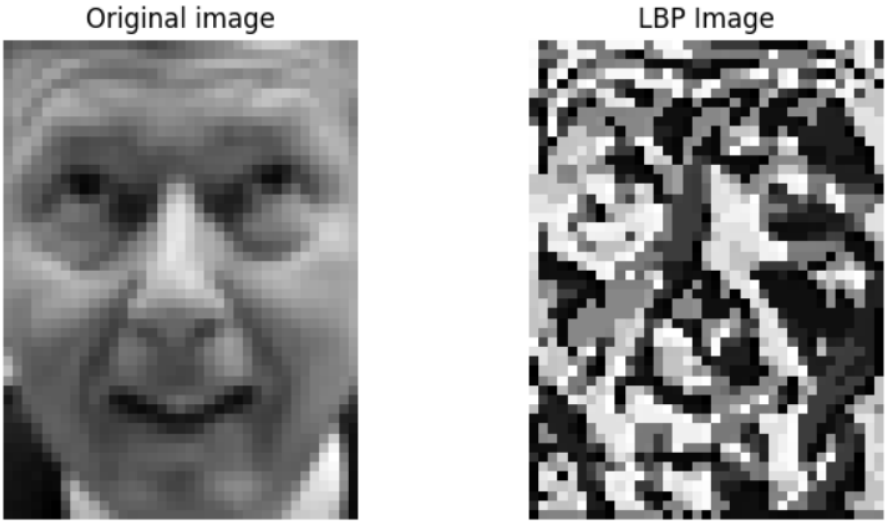

Figure 11: Original and LBP Image Comparison

We used the following classifiers for LBP feature extraction:

1. **Logistic Regression**
2. **Support Vector Machines (SVMs)**

3.5 Histogram of Gradients (HoG)

We have done the following to extract features of LFW images using HoG:

1. **Image Preprocessing:** The code resizes the input image to a fixed size of (1282, 642) pixels using the resize function. This resizing step is crucial for standardizing the input size and reducing computational complexity. It ensures that all images fed into the algorithm have the same dimensions, which is necessary for consistent feature extraction.
2. **Feature Extraction:** The `compute_hog` function calculates the Histogram of Oriented Gradients (HoG) features from the resized image. HoG features capture the distribution of gradient orientations in the image, providing essential information for object detection and recognition tasks. The resulting features (fd) are extracted with specified parameters and can be utilized for image classification purposes.
3. **Visualization:** The HoG extracted features image is visualized and compared with the original image.

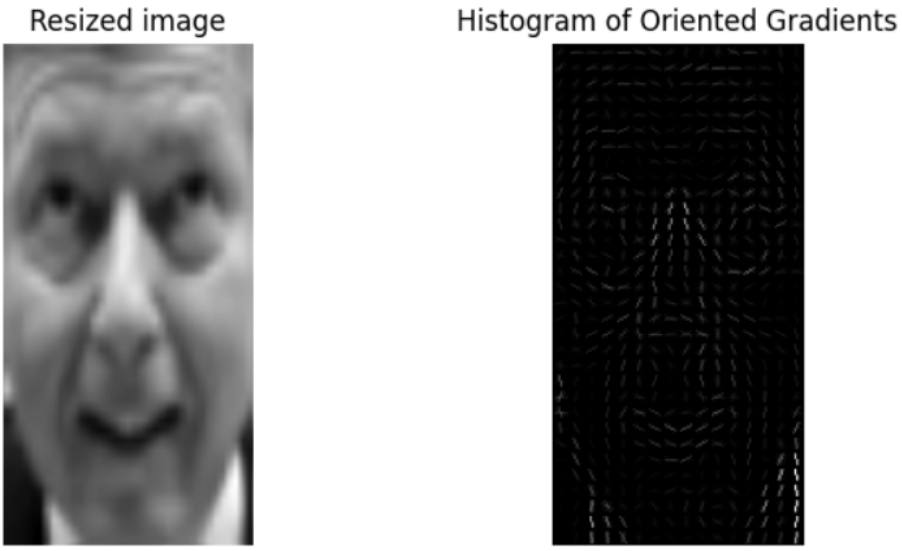

Figure 12: Original and HoG image comparison

We have used the following classifiers on the HoG extracted features:

1. **PCA - Logistic Regression**
2. **PCA - ANN**
3. **PCA - SVM**

4 Results

Table 1: Classifier Accuracies

PCA	ANN	75.94%
	Logistic Regression	78.37%
	Decision Tree	37.64%
	Random Forest	37.36%
	SVM(Linear)	77.25%
	SVM(Poly)	79.21%
	SVM(RBF)	79.21%
LDA	Decision Tree	58.18%
	Random Forest	78.94%
	ANN	74.87%
	SVM(Linear)	28.34%
	SVM(Poly)	74.59%
	SVM(RBF)	76.47%
CNN	Logistic Regression	**89.52**%
	SVM(Linear)	85.55%
	SVM(Poly)	**86.12**%
	SVM(RBF)	**86.12**%
HoG	PCA-ANN	**86.12**%
	PCA-Logistic Regression	**86.52**%
	PCA-SVM(Linear)	85.39%
	PCA-SVM(Poly)	66.29%
	PCA-SVM(RBF)	**88.39**%
	LDA-Decision Tree	57.81%
	LDA-Random Forest	74.6%
LBP	PCA-Logistic Regression	68.91%
	PCA-SVM(Linear)	71.16%
	PCA-SVM(Poly)	44.19%
	PCA-SVM(RBF)	29.96%
	PCA-Decision Tree	33.59%
	PCA-Random Forest	35.93%

5 Summary

This project tries to understand the different techniques of facial recognition and finds out the best possible combination of classifiers and feature extraction methods and/or feature reduction methods to train the most accurate model.

We experimented with LDA, PCA, CNN, LBP, HoG and used classifiers such as ANN, Random Forest, SVM, Logistic Regression. The results tell us that CNN gives an all-round accurate performance when it is paired with any classifier. The highest accuracy achieved is 89.52% with CNN + Logistic Regression combination. Other methods such as HoG also give promising results. This helps us understand the effectiveness of different techniques in facial recognition applications. We also observed that the pairing of feature extraction and reduction techniques before classification leads to better results (eg. HoG + PCA + SVM(RBF)).

These results provide valuable insights into the performance of various techniques and their suitability for facial recognition tasks, offering guidance for future research and application development in this field.

A Contribution of each member

1. **Varshit Manikanta (B22AI038)**:
   (a) Github Repo Management
   (b) HoG and LBP Feature Extraction
   (c) Demo Code Page
2. **Atharva Kanad Deshpande (B22EE013)**:
   (a) CNN Feature Extraction and Classification
   (b) Project Report
3. **Dishit Sharma (B22CS082)**:
   (a) Exploratory Data Analysis (EDA)
   (b) Web Page
4. **Jinang Shah (B22CS027)**:
   (a) PCA Implementation
   (b) SVM (Linear, Poly, RBF) Implementation on original data and extracted features of LBP, HoG.
5. **Vishesh Sachdeva (B22AI050):**
   (a) Decision Tree-Random Forest Implementation from scratch
   (b) Project Video
6. **Harish (B22EE029):**
   (a) ANN implementation on original dataset and on extracted features from LBP, HoG.
7. **Kumar Ashish Aditya (B22EE042):**
   (a) Video PPT

B Bibliography:

1. https://www.sciencedirect.com/topics/engineering/local-binary-pattern
2. https://www.sciencedirect.com/topics/engineering/local-binary-pattern#:~:text=1.3%20Local%20binary%20pattern,possible%20edges%20in%20the%20image.
3. https://datagen.tech/guides/image-datasets/lfw-dataset/#:~:text=Labeled%20Faces%20in%20the%20Wild%20(LFW)%20is%20an%20image%20dataset,problem%20of%20unconstrained%20face%20recognition.
4. https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset
5. https://www.youtube.com/watch?v=h-z9-bMtd7w
6. https://www.youtube.com/watch?v=5nZGnYPyKLU