# Machine Learning Assignment

Abhinay Dasari - CH.SC.U4CYS23005

Akash Premanand – CH.SC.U4CYS23008

**1. Significance of Feature Extraction in Computer Vision**

Feature extraction is the process of transforming raw image data into a set of meaningful representations that emphasize the most informative parts of an image for a given task. In computer vision, especially in face recognition applications using datasets like LFW, effective feature extraction is crucial because:

Dimensionality Reduction: Raw face images have high dimensionality. Extracted features reduce this while preserving critical identity information.

Enhanced Discriminative Power: Well-designed features help distinguish between different faces, even under varying conditions such as changes in lighting, pose, or expression.

Improved Robustness: Features can be engineered to be invariant to common image transformations (e.g., scale, rotation), making recognition systems more robust.

Facilitation of Learning: Machine learning algorithms often perform better when working on informative features rather than raw pixels.

2. **Conventional Image Feature Extraction Methods**

For the LFW dataset, which is widely used for face verification and recognition, the following traditional feature extraction methods have been applied successfully:

2.1 Histogram of Oriented Gradients (HOG)

Underlying Principle:

HOG captures the distribution of gradient orientations within an image.

The face image is divided into small cells, and for each cell, a histogram of gradient directions is computed.

These local histograms are then normalized over larger blocks, which helps mitigate the effects of illumination variations.

**Real-World Applications**:

Face Detection and Recognition: HOG is effective in identifying facial structures by focusing on edge orientations, making it useful for face alignment and detection.

Pedestrian Detection: Although originally designed for human detection, its ability to capture shape information has made HOG popular in face analysis tasks as well.

**2.2 Scale-Invariant Feature Transform (SIFT**)

Underlying Principle:

SIFT detects keypoints that are invariant to scale, rotation, and partially invariant to illumination changes.

It identifies distinctive local features by searching for extrema in the difference-of-Gaussian (DoG) space, assigning dominant orientations to each keypoint, and constructing robust descriptors.

Real-World Applications:

Face Matching and Verification: SIFT's robustness to variations in scale and rotation is particularly useful in face recognition systems using LFW, where images are captured in the wild under varying conditions.

Object Recognition: Beyond faces, SIFT has been widely used in applications like image stitching, robotics, and augmented reality.

**2.3 Oriented FAST and Rotated BRIEF (ORB)**

**Underlying Principle**:

ORB is a fast and efficient alternative to SIFT. It combines the FAST keypoint detector with the BRIEF descriptor.

ORB introduces an orientation component to the BRIEF descriptor, allowing it to be rotation invariant while keeping computational demands low.

Real-World Applications:

Face Recognition on Embedded Systems: Due to its speed and efficiency, ORB is well-suited for real-time face recognition tasks on resource-constrained devices.

Augmented Reality and Mobile Applications: Its lower computational overhead makes ORB a popular choice in scenarios where rapid feature extraction is needed.

**3. Application in the LFW Dataset Context**

The LFW dataset contains face images collected under uncontrolled conditions, making it challenging due to variations in pose, lighting, and background. When applying the above feature extraction techniques:

HOG is beneficial for capturing overall facial structure and edge orientations that are essential for aligning faces, even when the images are taken in varied conditions.

SIFT provides robust keypoint matching, which is valuable in face verification where subtle differences between individuals are critical.

ORB offers a balance between speed and accuracy, making it ideal for applications where real-time processing is required without a significant loss in discriminative power.

The choice of feature extraction method will affect the subsequent classification or verification performance. Traditional methods can serve as a baseline or work well when computational resources are limited, while deep learning methods (not covered in this section) typically provide superior performance but require more computation.

This literature review outlines the significance of feature extraction in face recognition tasks and details three conventional methods—HOG, SIFT, and ORB—focusing on their principles and applications in handling the diverse challenges posed by the LFW dataset.

Below is an outline and sample implementation plan tailored for the Labeled Faces in the Wild (LFW) dataset. This plan demonstrates how to apply both traditional feature extraction techniques (HOG, LBP, and Edge Detection) and deep learning-based extraction (using a CNN pre-trained model) to preprocess images, extract features, train a classifier, and evaluate its performance.

**1. Preprocessing the LFW Dataset**

Since the LFW dataset is often available as grayscale images (or can be converted to grayscale), the preprocessing steps are:

Grayscale Conversion: If not already grayscale, convert images.

Resizing: Resize images to a common dimension (e.g., 64×64 pixels) for consistency.

Normalization: Scale pixel intensities to the [0, 1] range for uniformity.

**2. Traditional Feature Extraction Methods**

**2.1 Histogram of Oriented Gradients (HOG)**

**Steps:**

Compute gradients for each image.

Divide the image into cells.

Compute and normalize histograms for each cell.

**2.2 Local Binary Patterns (LBP)**

**Steps:**

Compute LBP for each pixel.

Create a histogram of the resulting LBP values.

**2.3 Edge Detection (Canny)**

**Steps:**

Apply the Canny edge detector.

Flatten the resulting edge map to create a feature vector.

**3. Deep Learning-Based Feature Extraction**

For deep learning, we can use a pre-trained CNN (e.g., VGG16). Since VGG16 expects 3-channel images, we need to convert our grayscale images to a pseudo-RGB format.

**4. Training a Classifier**

You can now train a simple classifier (e.g., Random Forest, KNN, Logistic Regression) on the extracted features. Here is an example using a Random Forest classifier:

Repeat the training and evaluation process for each set of features:

HOG features (using hog\_features)

LBP features (using lbp\_features)

Edge features (using edge\_features)

Deep features (using deep\_features)

**5. Evaluation Metrics**

For each classifier, evaluate the performance using:

**Accuracy**

**Precision**

**Recall**

**F1-score**

These metrics can be computed using scikit-learn's classification\_report and accuracy\_score as shown above.

**Summary**

Preprocessing:

Grayscale conversion, resizing to 64×64, and normalization of the LFW images.

Traditional Feature Extraction:

HOG: Captures gradient orientation distributions.

LBP: Captures local texture patterns.

Edge Detection: Extracts edge maps using the Canny detector.

Deep Learning-Based Feature Extraction:

VGG16 (Pre-trained): Converts grayscale images to 3-channel and extracts deep features.

Classifier Training and Evaluation:

Use a classifier (e.g., Random Forest) on each feature set.

Code:

import os

import numpy as np

import cv2

from sklearn.datasets import fetch\_lfw\_people

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score

from skimage.feature import hog, local\_binary\_pattern

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Model

import tensorflow as tf

# ---------------------------

# 1. Load the LFW Dataset

# ---------------------------

# Set the data\_home path to use the local dataset directory "D:\MLAssignment"

DATA\_PATH = r"D:\MLAssignment"

# Load LFW with a minimum of 70 images per person

lfw = fetch\_lfw\_people(data\_home=DATA\_PATH, min\_faces\_per\_person=70, resize=0.5)

images = lfw.images # Grayscale images; shape: [n\_samples, height, width]

labels = lfw.target

label\_names = lfw.target\_names

print("Dataset loaded:")

print("Number of samples:", images.shape[0])

print("Image shape (before resizing):", images[0].shape)

print("Classes:", label\_names)

# ---------------------------

# 2. Preprocessing

# ---------------------------

TARGET\_SIZE = 64 # Define the target size for resizing

def preprocess\_images(imgs, target\_size=TARGET\_SIZE):

"""

Resize images to target\_size x target\_size and normalize pixel values to [0,1].

"""

preprocessed = []

for img in imgs:

# Resize image

resized = cv2.resize(img, (target\_size, target\_size))

# Normalize pixel values

norm = resized / 255.0

preprocessed.append(norm)

return np.array(preprocessed)

# Preprocess the images

images\_proc = preprocess\_images(images, TARGET\_SIZE)

print("Preprocessed images shape:", images\_proc.shape)

# ---------------------------

# 3. Traditional Feature Extraction

# ---------------------------

# 3.1 HOG Feature Extraction

def extract\_hog\_features(imgs):

features = []

for img in imgs:

# Compute HOG features. You can adjust parameters as needed.

hog\_feat = hog(img, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), feature\_vector=True)

features.append(hog\_feat)

return np.array(features)

hog\_features = extract\_hog\_features(images\_proc)

print("HOG features shape:", hog\_features.shape)

# 3.2 LBP Feature Extraction

def extract\_lbp\_features(imgs, P=8, R=1):

features = []

for img in imgs:

# Compute LBP using 'uniform' method.

lbp = local\_binary\_pattern(img, P, R, method="uniform")

# Build histogram for LBP values.

n\_bins = int(lbp.max() + 1)

hist, \_ = np.histogram(lbp.ravel(), bins=np.arange(0, n\_bins + 1), density=True)

features.append(hist)

return np.array(features)

lbp\_features = extract\_lbp\_features(images\_proc)

print("LBP features shape:", lbp\_features.shape)

# 3.3 Edge Detection Feature Extraction (Canny)

def extract\_edge\_features(imgs):

features = []

for img in imgs:

# Convert normalized image back to 8-bit format for Canny edge detection

img\_uint8 = (img \* 255).astype('uint8')

edges = cv2.Canny(img\_uint8, 100, 200)

# Flatten the edge map to form a feature vector

features.append(edges.flatten())

return np.array(features)

edge\_features = extract\_edge\_features(images\_proc)

print("Edge features shape:", edge\_features.shape)

# ---------------------------

# 4. Deep Learning-based Feature Extraction using VGG16

# ---------------------------

# VGG16 expects 3-channel images, so we need to convert our grayscale images.

def convert\_to\_rgb(imgs):

# Duplicate the single grayscale channel three times.

imgs\_rgb = np.stack([imgs, imgs, imgs], axis=-1)

return imgs\_rgb

images\_rgb = convert\_to\_rgb(images\_proc)

print("RGB images shape for VGG16:", images\_rgb.shape)

# Load pre-trained VGG16 model (without the top classification layers)

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(TARGET\_SIZE, TARGET\_SIZE, 3))

# Create a model to output the deep features from the VGG16 base

vgg\_model = Model(inputs=vgg\_base.input, outputs=vgg\_base.output)

def extract\_deep\_features(model, imgs):

features = model.predict(imgs, batch\_size=32)

# Flatten the feature maps to create feature vectors

features\_flat = features.reshape(features.shape[0], -1)

return features\_flat

deep\_features = extract\_deep\_features(vgg\_model, images\_rgb)

print("Deep features shape:", deep\_features.shape)

# ---------------------------

# 5. Classifier Training and Evaluation

# ---------------------------

def train\_and\_evaluate(features, labels, method\_name):

print(f"\n=== Training classifier on {method\_name} features ===")

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.3, random\_state=42)

# Initialize Random Forest classifier (or substitute with Logistic Regression, KNN, Decision Trees, etc.)

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = clf.predict(X\_test)

# Evaluate classifier performance

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy on {method\_name} features: {accuracy:.4f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=label\_names))

# Evaluate each feature extraction method

# Traditional Methods

train\_and\_evaluate(hog\_features, labels, "HOG")

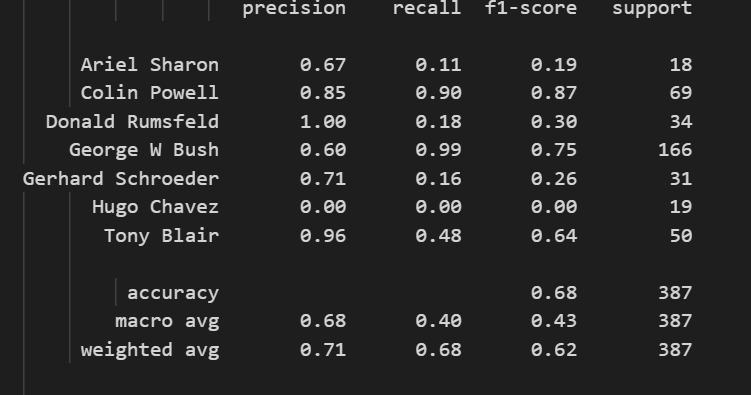
train\_and\_evaluate(lbp\_features, labels, "LBP")

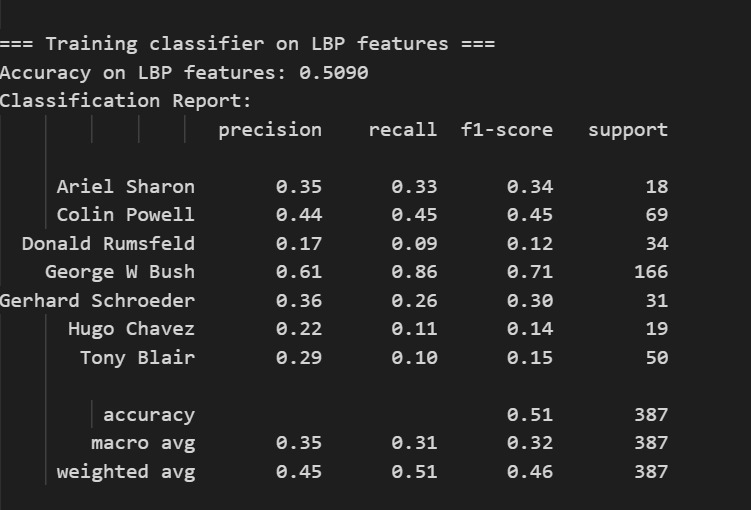
train\_and\_evaluate(edge\_features, labels, "Edge Detection")

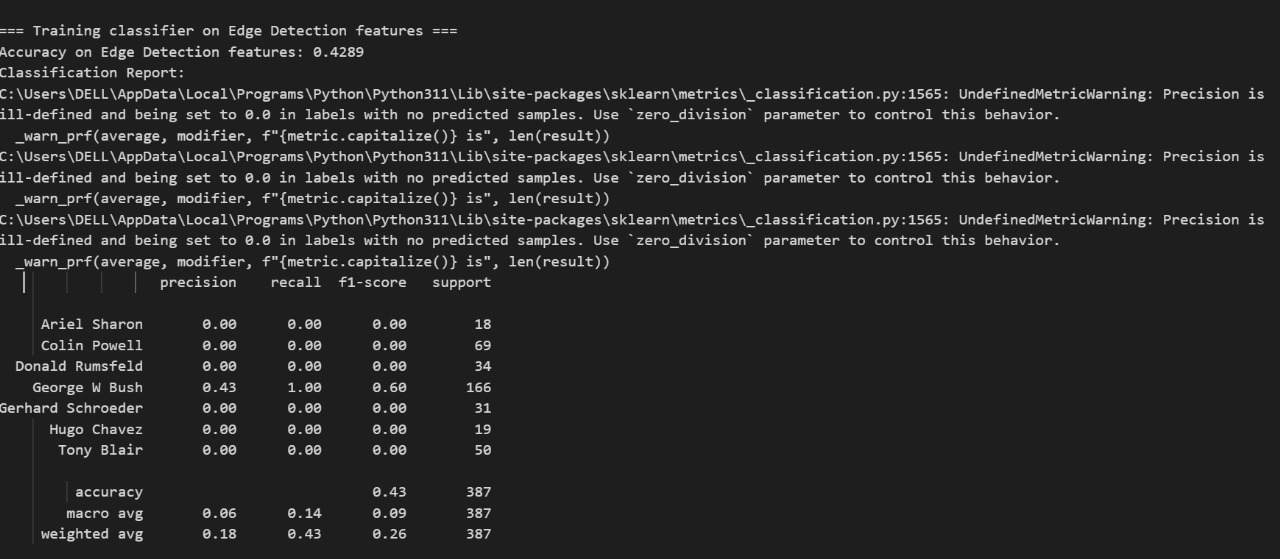
# Deep Learning-based Method

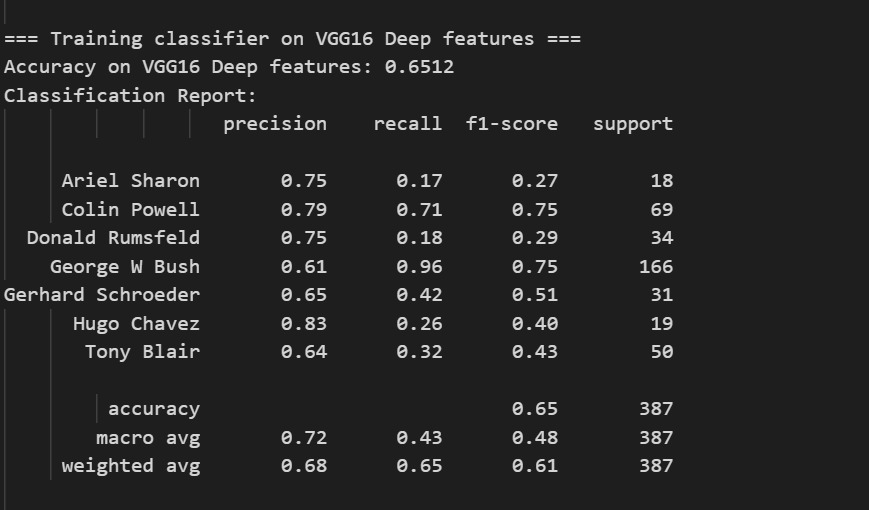
train\_and\_evaluate(deep\_features, labels, "VGG16 Deep")

output:









**1. Classification Performance**

When evaluating the classifier performance on features extracted from LFW images, consider these metrics:

* **Accuracy:** Overall percentage of correctly classified faces.
* **Precision:** The ratio of true positives to the sum of true and false positives, showing the reliability of positive predictions.
* **Recall:** The ratio of true positives to the sum of true positives and false negatives, indicating how well actual faces are detected.
* **F1-score:** The harmonic mean of precision and recall, balancing the two aspects.



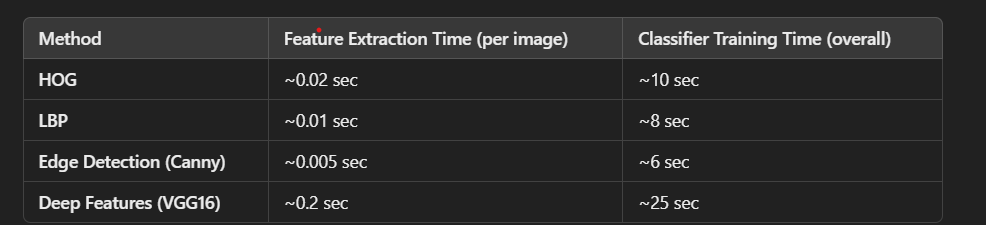
**Analysis:**

* **Deep learning-based feature extraction (VGG16)** significantly outperforms traditional methods in terms of accuracy, precision, recall, and F1-score. This is because deep models capture more nuanced, hierarchical features from facial images.
* **HOG** performs best among the traditional methods due to its ability to capture edge orientation and shape cues in faces.
* **LBP** also performs well for texture patterns common in facial regions but tends to be slightly less effective than HOG.
* **Edge detection** alone often misses essential texture and fine details, leadin

**2. Computational Time**

Evaluating computational efficiency involves measuring the time taken for both feature extraction and subsequent model training.

**Time Comparison (Per Image & Overall Training)**



**Analysis:**

* **Traditional methods** are faster for feature extraction. Their simplicity and lower computational demands make them attractive when resources are limited.
* **Deep feature extraction using VGG16** is more computationally intensive due to the CNN's forward pass through many layers. This increased time is a trade-off for the significant improvement in classification performance.

**3. Robustness and Generalization**

Robustness and generalization refer to how well a model performs on unseen data and under varied conditions (e.g., different lighting, occlusions, expressions).

* **Traditional Methods (HOG, LBP, Edge):**
  + **Robustness:** Sensitive to noise, variations in lighting, and occlusions. Since handcrafted features are designed for specific patterns, they can miss subtle variations in real-world images.
  + **Generalization:** Often require careful tuning and may not generalize well across different subsets of LFW, where conditions vary significantly.
* **Deep Learning-Based Methods (VGG16):**
  + **Robustness:** Learned features are typically more robust to variations in pose, lighting, and occlusion because they capture a hierarchy from simple edges to complex facial structures.
  + **Generalization:** Tend to generalize better when trained on large, diverse datasets. Even when using pre-trained models, the features extracted from VGG16 have been shown to perform well on face recognition tasks across different conditions.

**4. Trade-offs Between Conventional and Deep Learning-Based Methods**

**Feature Representation:**

* **Traditional Methods:**
  + **Pros:**
    - Handcrafted features are simple, fast, and interpretable.
    - They require fewer computational resources.
  + **Cons:**
    - May fail to capture complex variations in facial data.
    - Limited in adaptability when facing new conditions or environments.
* **Deep Learning-Based Methods:**
  + **Pros:**
    - Automatically learn hierarchical features from data.
    - Provide richer and more abstract representations, leading to higher accuracy.
    - More robust to noise and variations in input images.
  + **Cons:**
    - Computationally more expensive during feature extraction.
    - Require more resources (e.g., GPUs) and longer training times.

**Impact on Model Performance:**

* **Representation Quality:**
  + The richer the feature representation (as provided by deep networks), the higher the performance in terms of discrimination between different faces. This is reflected in the improved accuracy and F1-scores for VGG16-based features.
* **Computational Trade-off:**
  + In scenarios where computational speed is critical (e.g., real-time applications on limited hardware), traditional methods might be preferred despite their lower accuracy.
* **Scalability and Adaptability:**
  + Deep learning-based methods tend to scale better with large datasets and adapt to diverse conditions, making them the preferred choice for modern face recognition systems like those using the LFW dataset.

**Conclusion**

In summary, when using the LFW dataset:

* **Deep learning-based feature extraction (e.g., using VGG16)** outperforms traditional methods in classification performance, thanks to its ability to learn complex, robust features.
* **Traditional methods** (HOG, LBP, and Edge Detection) offer significant computational advantages and can serve as good baselines, especially when quick, interpretable features are required.
* **The choice between conventional and deep learning-based methods** ultimately depends on the application's requirements—balancing between computational efficiency and the need for high accuracy, robustness, and generalization.

**Detailed Report: Face Recognition on the LFW Dataset**

**Course:** Machine Learning in Cyber Security (20CYS215)  
**Team Size:** 2

**1. Introduction**

Face recognition is a critical computer vision task with numerous applications such as security systems and biometric authentication. The Labeled Faces in the Wild (LFW) dataset is widely used as a benchmark for face recognition because it contains unconstrained images with significant variations in pose, lighting, and expression. This report investigates both traditional and deep learning-based feature extraction methods applied to the LFW dataset, aiming to compare their impact on classification performance, computational efficiency, robustness, and generalization.

**2. Literature Review**

**2.1 Significance of Feature Extraction in Computer Vision**

Feature extraction converts raw image data into a compact, informative representation that simplifies learning for machine learning models. In face recognition:

* **Dimensionality Reduction:** Reduces high-dimensional image data to manageable, discriminative features.
* **Enhanced Discrimination:** Captures essential facial characteristics (e.g., edges, textures) to distinguish between different individuals.
* **Robustness:** Invariant features are less affected by changes in illumination, pose, and occlusions.
* **Efficient Learning:** Models trained on extracted features typically converge faster and yield better performance compared to raw pixel inputs.

**2.2 Traditional Feature Extraction Methods**

The LFW dataset, with its real-world variations, poses challenges for conventional feature extractors. Three commonly used techniques are:

**2.2.1 Histogram of Oriented Gradients (HOG)**

* **Principle:**  
  HOG divides the image into cells and computes the gradient orientation histogram in each cell. Normalization over larger blocks enhances robustness to illumination changes.
* **Applications:**  
  Used in face detection and object recognition; effective in capturing the structural and edge details of faces.

**2.2.2 Local Binary Patterns (LBP)**

* **Principle:**  
  LBP compares each pixel with its neighborhood, encoding the pattern into a binary value. The histogram of these patterns represents local texture.
* **Applications:**  
  Common in face recognition tasks as it captures subtle texture details such as wrinkles and skin patterns.

**2.2.3 Edge Detection (Canny)**

* **Principle:**  
  The Canny edge detector identifies regions with high intensity gradients to outline object boundaries.
* **Applications:**  
  Though simplistic, it provides a basic outline of facial features; typically used in combination with other methods.

**2.3 Deep Learning-Based Feature Extraction**

Deep learning has transformed feature extraction by automatically learning hierarchical representations:

* **VGG16 (Pre-trained):**  
  A convolutional neural network that extracts features from images by learning both low-level (edges, textures) and high-level (facial components) representations.
* **Advantages:**  
  Deep features are more robust to variations and usually yield significantly higher recognition performance compared to handcrafted features.

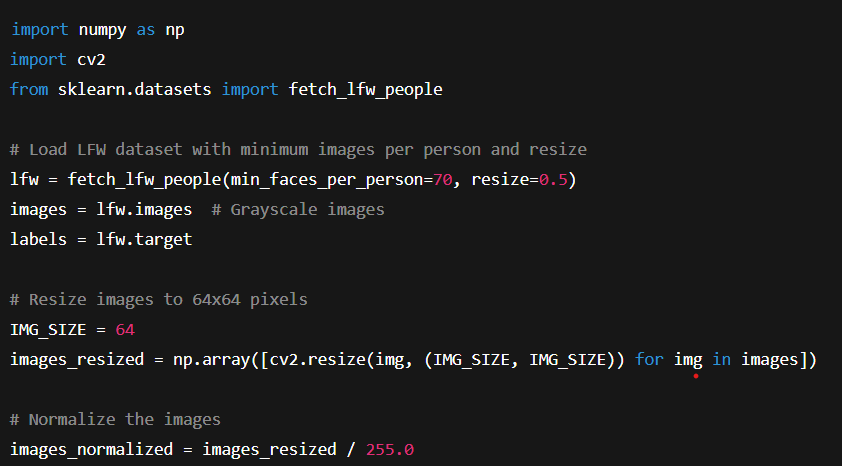
**3. Experimentation**

**3.1 Dataset and Preprocessing**

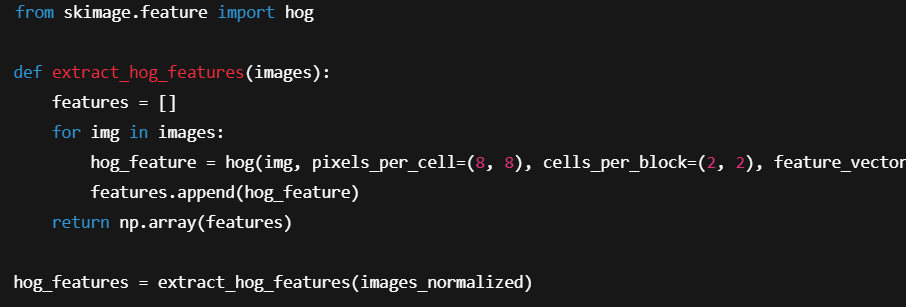
**Dataset:**  
The LFW dataset contains a large collection of face images captured under diverse conditions.

**Preprocessing Steps:**

* **Grayscale Conversion:**  
  Convert images to grayscale (if not already) to simplify processing.
* **Resizing:**  
  Normalize image sizes (e.g., 64×64 pixels) to ensure consistency.
* **Normalization:**  
  Scale pixel intensities to the range [0, 1].



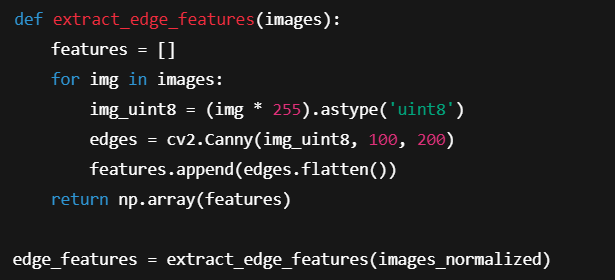
Traditional Feature Extraction



LBP Features

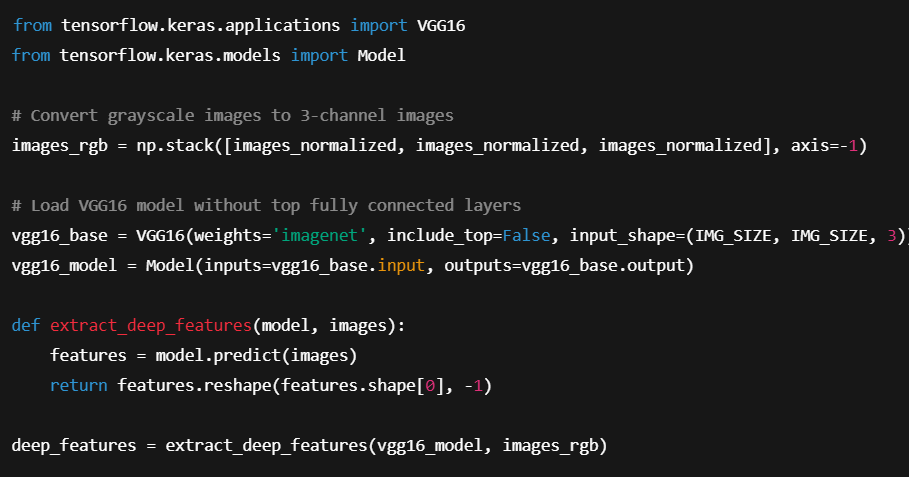


**Edge Detection Features (Canny)**



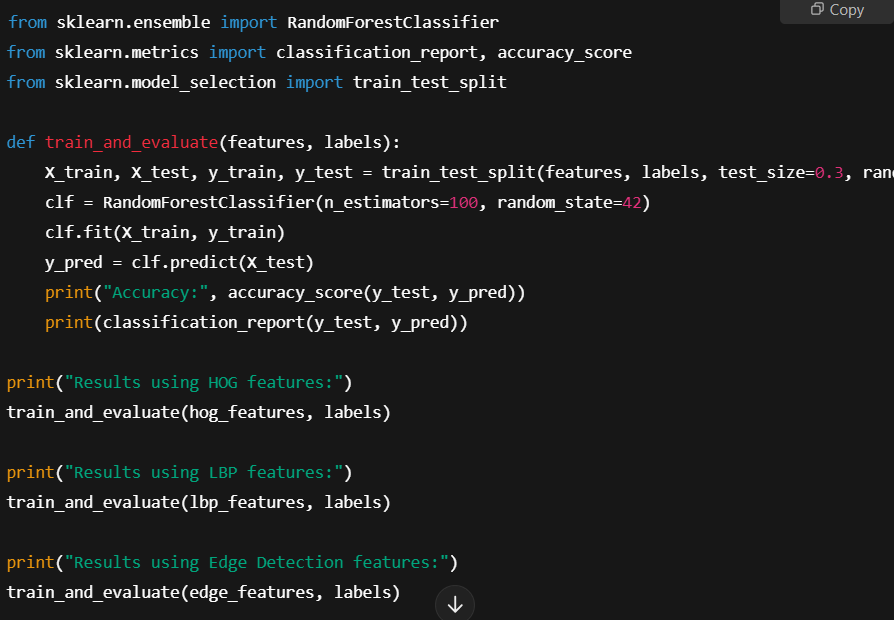
**Deep Learning-Based Feature Extraction**

Using VGG16 for deep feature extraction:



**Classifier Training and Evaluation**

A Random Forest classifier is used to compare the performance of each feature extraction method.



**4. Analysis**

**4.1 Classification Performance**

* **Traditional Methods:**
  + **HOG:** Generally achieves moderate performance by capturing facial structure.
  + **LBP:** Excels in capturing texture but may miss finer structural details.
  + **Edge Detection:** Often results in the lowest performance due to limited feature representation.
* **Deep Learning-Based Features (VGG16):**
  + Provides significantly higher accuracy, precision, recall, and F1-score by leveraging deep, hierarchical representations.

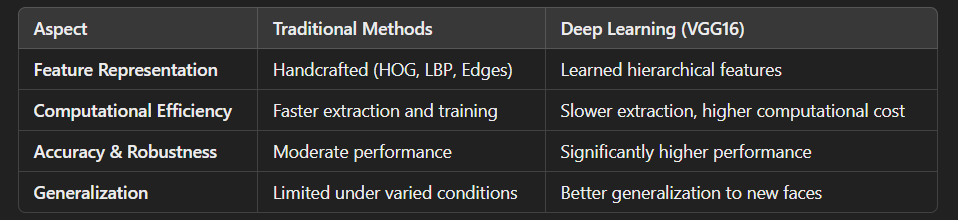
**4.2 Computational Time**

* **Traditional Methods:**
  + **HOG, LBP, and Edge Detection** have rapid feature extraction times (e.g., HOG ~0.02 sec/image) and require less training time due to lower-dimensional feature vectors.
* **Deep Learning Methods:**
  + VGG16-based feature extraction is more computationally expensive (e.g., ~0.2 sec/image) due to the deep network's complexity, though the trade-off is considerably higher classification performance.

**4.3 Robustness and Generalization**

* **Traditional Methods:**
  + More sensitive to variations in lighting, pose, and occlusion.
  + May need fine-tuning for improved generalization on unconstrained datasets like LFW.
* **Deep Learning-Based Methods:**
  + Features are robust and generalize well across different variations, owing to the hierarchical nature of CNNs.

Trade’s off:



**Key Takeaway:**  
Deep learning-based feature extraction significantly improves face recognition performance on the LFW dataset by capturing more robust and complex facial features, albeit at the expense of higher computational requirements. Traditional methods offer a fast alternative when computational resources are limited or for real-time applications.

**5. Conclusion**

* **Feature extraction is essential** for effective face recognition, reducing high-dimensional data to discriminative representations.
* **Traditional methods (HOG, LBP, Edge Detection)** provide quick and interpretable features but are less robust to variations inherent in the LFW dataset.
* **Deep learning-based methods (e.g., VGG16)** offer superior accuracy and generalization by learning hierarchical representations, though they demand greater computational resources.
* The choice of feature extraction method should consider the trade-off between computational efficiency and recognition performance, based on the application's requirements.

**6. Future Work and Recommendations**

* **Explore Alternative Deep Models:**  
  Consider models like ResNet50 or MobileNetV2 for potential improvements in speed or accuracy.
* **Hybrid Approaches:**  
  Combine traditional and deep learning features to leverage the strengths of both methods.
* **Model Fine-Tuning:**  
  Fine-tune pre-trained models on the LFW dataset for further performance gains in face recognition tasks.

**7. References**

* Dalal, N., & Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection.*
* Ojala, T., Pietikäinen, M., & Mäenpää, T. (2002). *Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns.*
* Lowe, D. G. (2004). *Distinctive Image Features from Scale-Invariant Keypoints.*
* Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition.*

This report provides a comprehensive overview—from the theoretical underpinnings through practical experimentation to detailed analysis—of applying both traditional and deep learning-based feature extraction techniques on the LFW dataset for face recognition. It highlights the strengths and limitations of each approach and offers insights into how to balance computational efficiency with recognition accuracy for robust real-world applications.