Face Classification System using Machine Learning and Deep Learning Models

Author: Research Team

Date: April 7, 2025

Table of Contents

1. Introduction
2. Dataset and Feature Extraction
   2.1 Handmade Features (LBP, HOG, GLCM, HU Moments)
   2.2 Deep Learning Features (VGG19)
3. Machine Learning Models
   3.1 Support Vector Machine (SVM)
   3.2 Random Forest
   3.3 XGBoost
   3.4 Naïve Bayes
   3.5 Decision Tree
4. Deep Learning Model (CNN- Convo 4 Block 5)
5. Experimental Results
   5.1 Performance Comparison
   5.2 Accuracy Analysis
6. Conclusion

Introduction

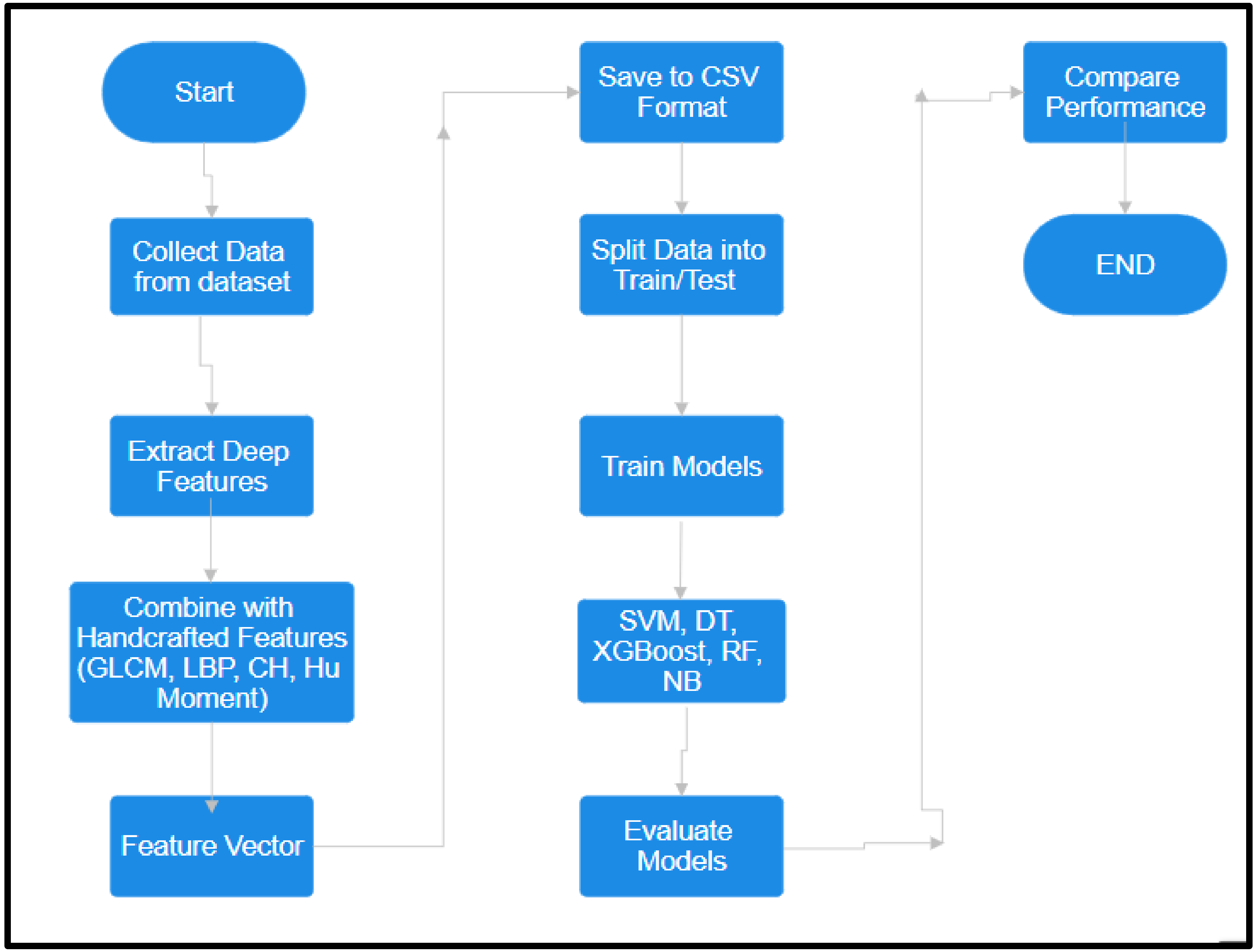
Face classification is a crucial task in computer vision with applications in security, biometric verification, and human-computer interaction.

This project explores machine learning and deep learning-based techniques for enhancing the classification accuracy.

We explore classic models like Support Vector Machine (SVM), Random Forest, XGBoost, Naïve Bayes and Decision Tree using both handcrafted (LBP, HOG, GLCM & HU) and deep learning-based (VGG19) feature extraction. The research also uses a Convolutional Neural Network (CNN) model consisting of four convolutional blocks and five layers for improved classification performance.

Experimental outcomes reveal that ensemble techniques such as XGBoost and Random Forest perform better than traditional methods with the best accuracy of 87%. The results point to the importance of feature extraction techniques and model choice in improving classification performance.

Flowchart


Start
Collect Data from dataset
Extract Deep Features
Combine with Handcrafted Features (GLCM, LBP, CH, Hu Moment)
Feature Vector
Save to CSV Format
Split Data into Train/Test
Train Models
SVM, DT, XGBoost, RF, NB
Evaluate Models
Compare Performance
END

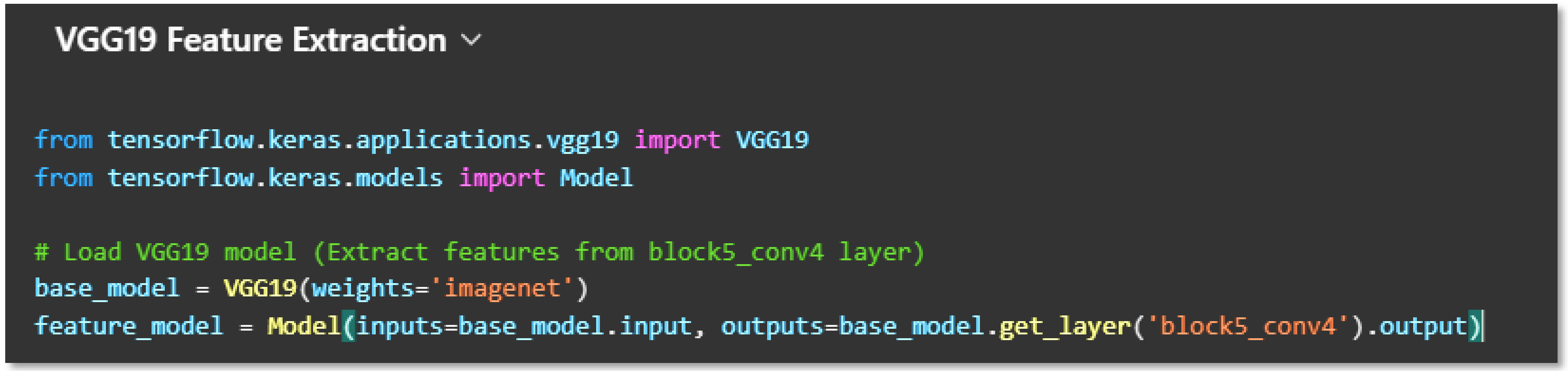
Deep Learning Model (CNN- Convo 4 Block 5)

The CNN structure comprises 4 convolutional blocks and 5 layers to extract hierarchical facial features and improve classification accuracy.

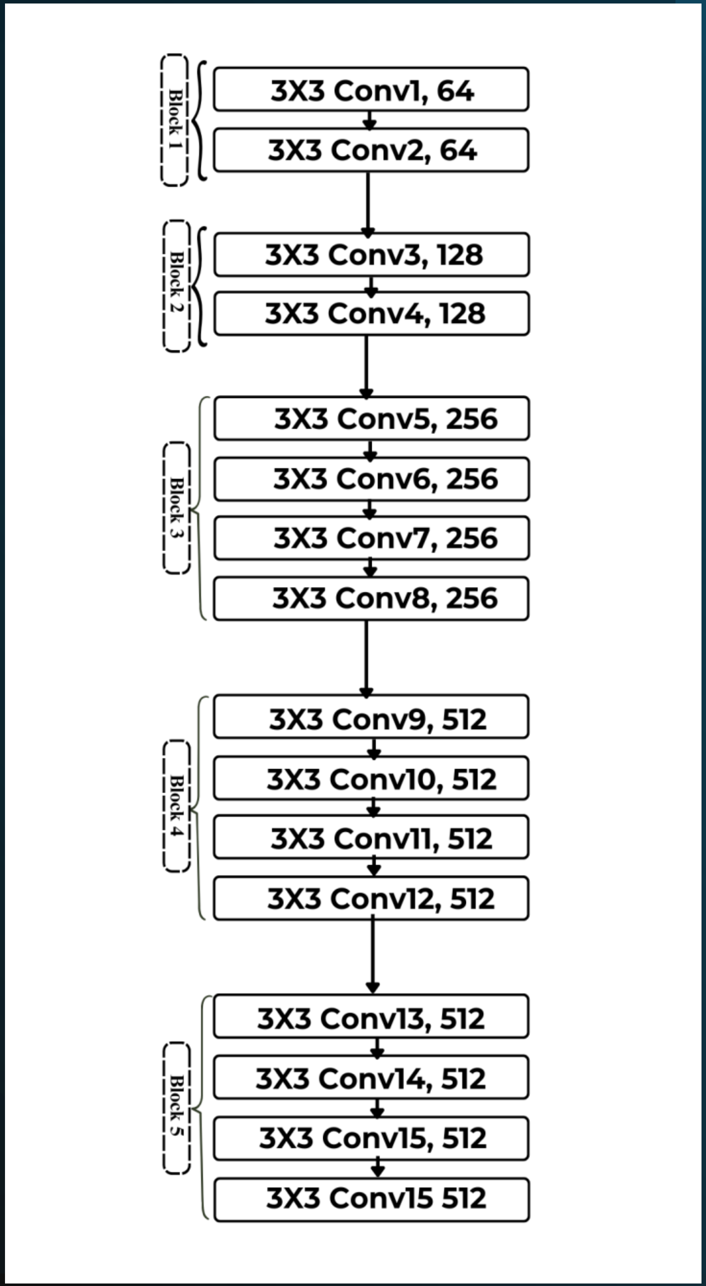
The structure involves:

1. Convolutional Layers: Extract spatial patterns and edge features from face images.
2. Pooling Layers: Downsample dimensionality and preserve important information.
3. Fully Connected Layers: Classify the extracted features into corresponding face categories.

Through the use of deep learning, the CNN model proved to have strong performance in face classification, topping ensemble algorithms like XGBoost and Random Forest.


VGG19 Feature Extraction

```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.models import Model

# Load VGG19 model (Extract features from block5_conv4 layer)
base_model = VGG19(weights='imagenet')
feature_model = Model(inputs=base_model.input, outputs=base_model.get_layer('block5_conv4').output)
```

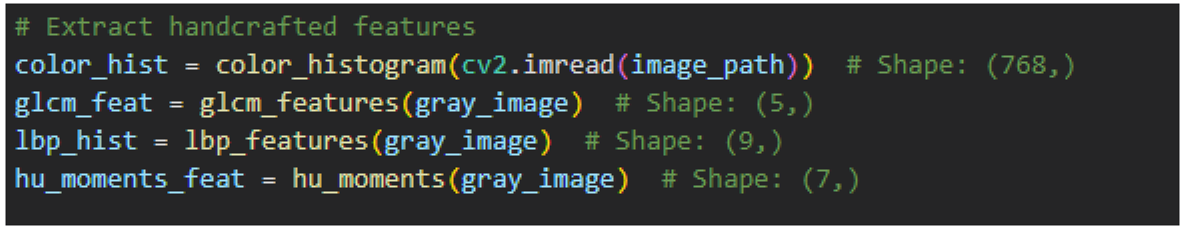


Dataset and Feature Extraction

For training and testing the face classification models, a diverse dataset of facial images from various demographics was utilized.

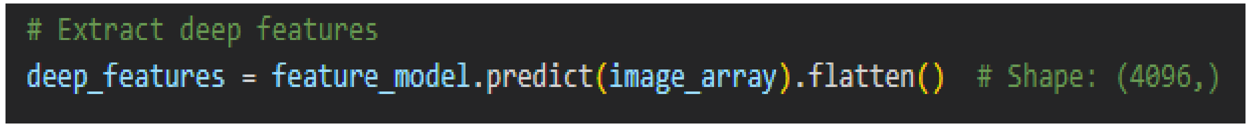
Two types of features extraction were investigated:

1. Handmade Features: Local Binary Pattern (LBP), Histogram of Oriented Gradients (HOG), Gray Level Co-occurrence matrix(GLCM) & Hu Moment invariants were used for texture-based feature extraction.

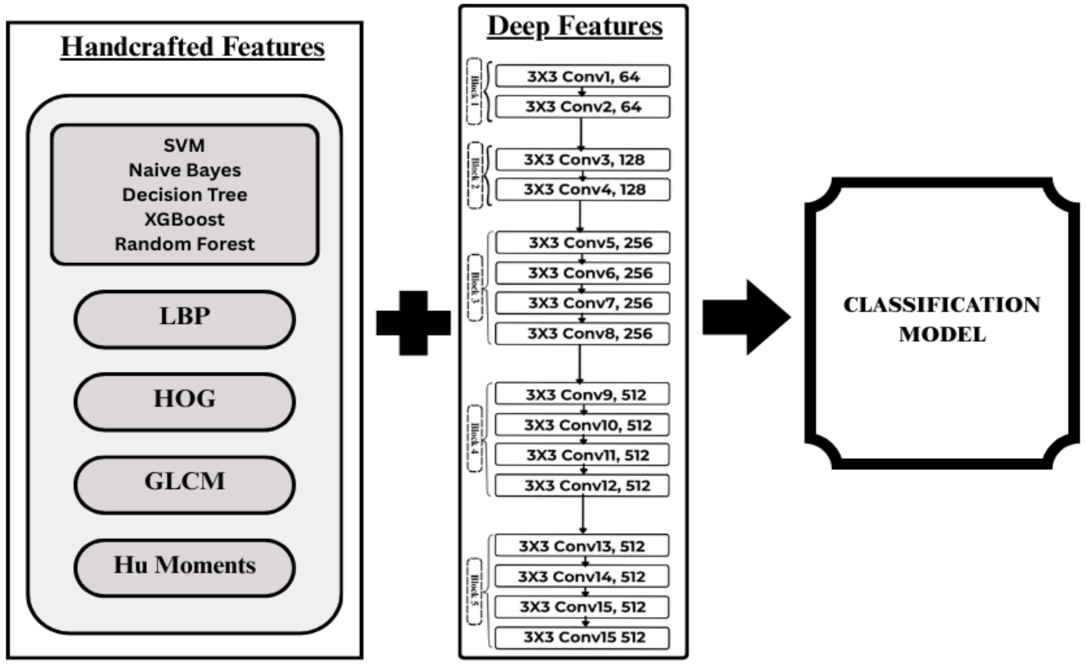


```
# Extract handcrafted features
color_hist = color_histogram(cv2.imread(image_path))  # Shape: (768,)
glcm_feat = glcm_features(gray_image)  # Shape: (5,)
lbp_hist = lbp_features(gray_image)  # Shape: (9,)
hu_moments_feat = hu_moments(gray_image)  # Shape: (7,)
```

2. Deep Features: The VGG19 pre-trained network was leveraged to extract high-dimensional representations, capturing complex facial structures.

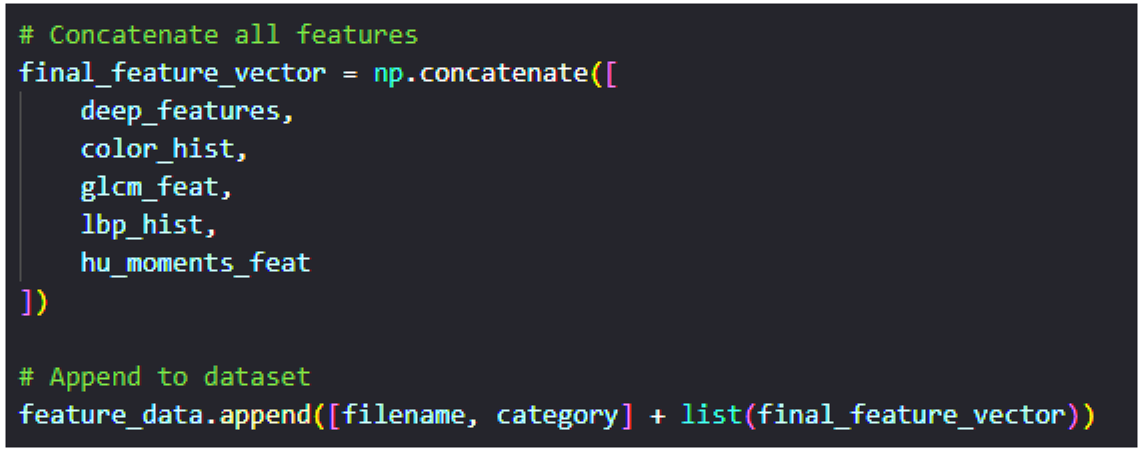


```
# Extract deep features
deep_features = feature_model.predict(image_array).flatten()  # Shape: (4096,)
```



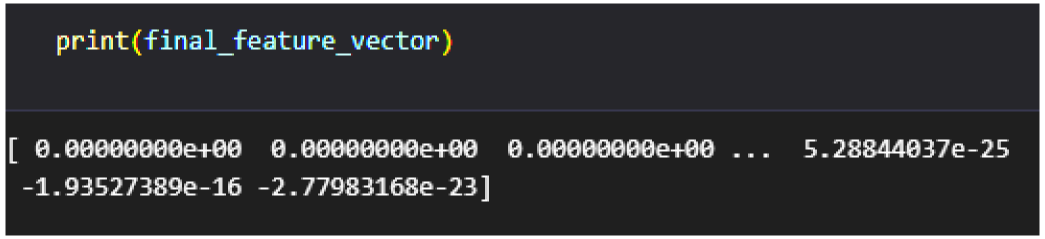
Dataset and Feature Extraction

By comparing these feature extraction methods, we analyzed the impact on different classification models.



```python
# Concatenate all features
final_feature_vector = np.concatenate([
    deep_features,
    color_hist,
    glcm_feat,
    lbp_hist,
    hu_moments_feat
])

# Append to dataset
feature_data.append([filename, category] + list(final_feature_vector))
```

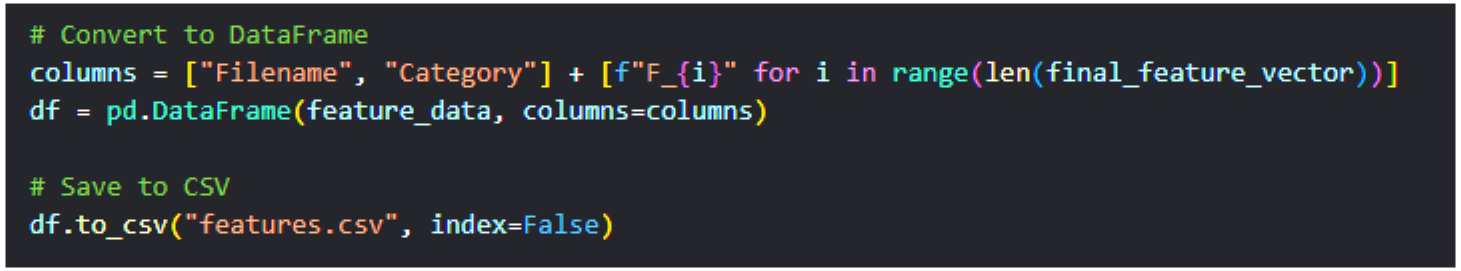
We then print the final feature vector



```
print(final_feature_vector)
```

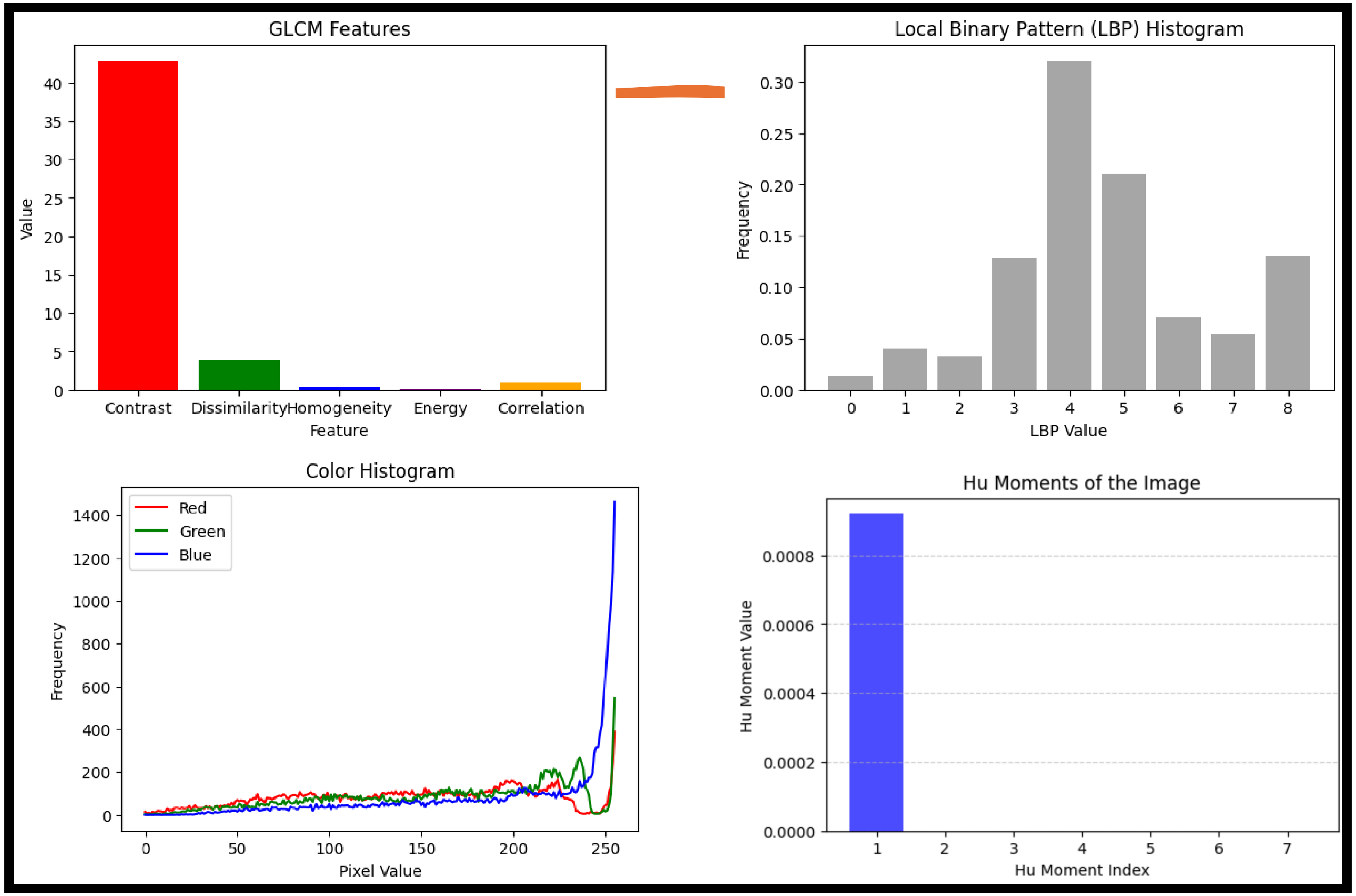
```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ...  5.28844037e-25
 -1.93527389e-16 -2.77983168e-23]
```

The code converts extracted features, along with filenames and categories, into a structured DataFrame. It then saves this data as a CSV file (`features.csv`) without including row indices.



```python
# Convert to DataFrame
columns = ["Filename", "Category"] + [f"F_{i}" for i in range(len(final_feature_vector))]
df = pd.DataFrame(feature_data, columns=columns)

# Save to CSV
df.to_csv("features.csv", index=False)
```

Handcrafted Feature Result:



Machine Learning Models

A number of machine learning algorithms were used **for face classification:**

1. Support Vector Machine (SVM): An effective classifier that maximizes margin between various classes in high-dimensional space.

```
svm_model = SVC(kernel='linear')  # You can change kernel to 'rbf', 'poly', etc.
svm_model.fit(X_train, y_train)
```

2. Naïve Bayes: A Bayes' theorem-based probabilistic model, which is widely applied in text classification but also performs well in face classification.

```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

3. XGBoost: A gradient boosting algorithm used to optimize tree-based models with high efficiency and accuracy.

```python
 xgb_model = XGBClassifier(n_estimators=100, learning_rate=0.1,
eval_metric='logloss')  # Model initialization
 xgb_model.fit(X_train, y_train)  # Training the model
```

4. Random Forest: A form of ensemble learning that involves several decision trees combined to achieve high accuracy and lower overfitting.

```python
rf_model = RandomForestClassifier(random_state=42)  # Model initialization
rf_model.fit(X_train, y_train)  # Training the model
```

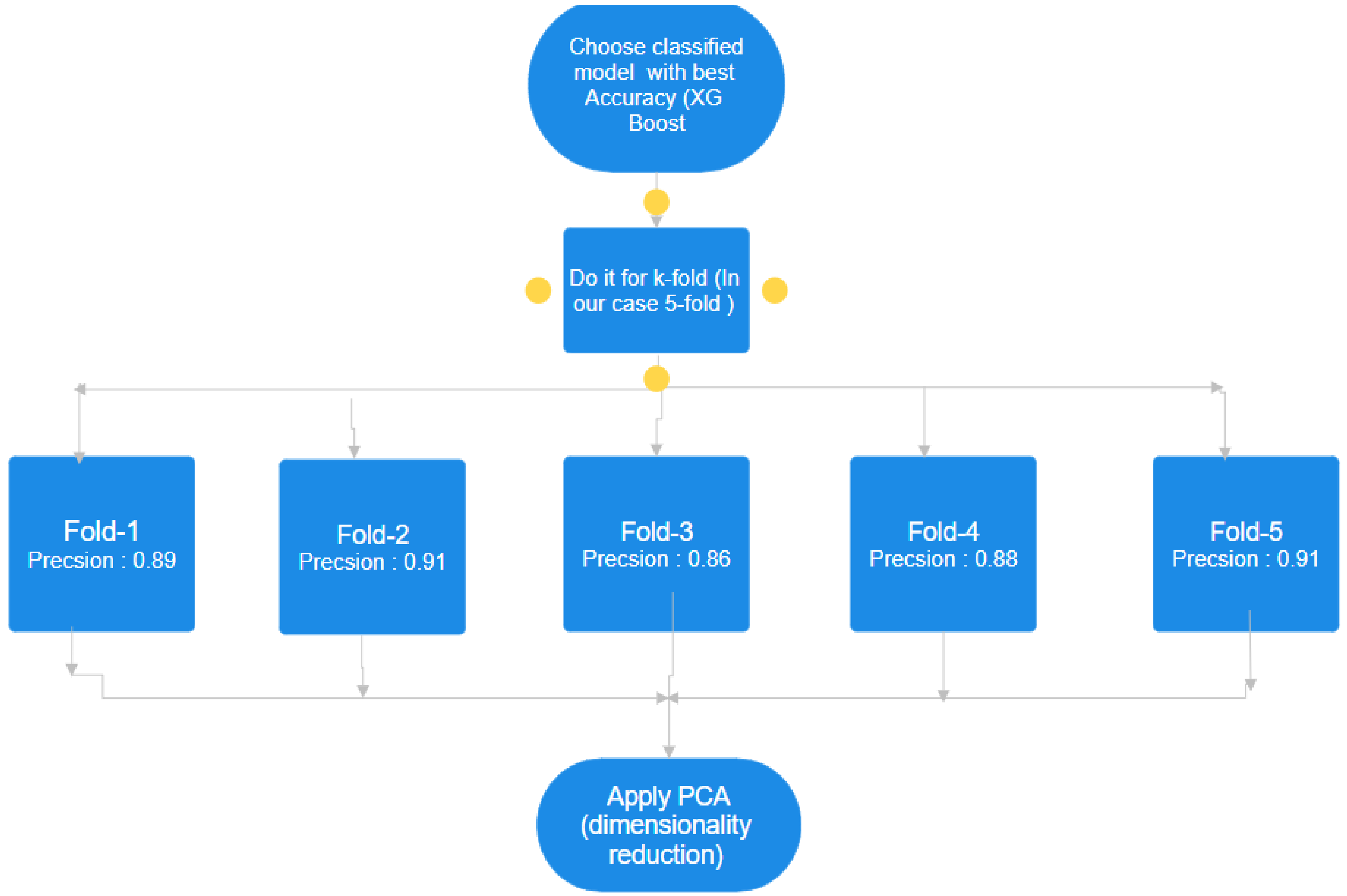
5. Decision Tree: Recursively splits data based on feature conditions, forming a tree-like structure.

```python
 dt_model = DecisionTreeClassifier(max_depth=10, random_state=42)  #
Model initialization
 dt_model.fit(X_train, y_train)  # Training the model
```

Among all these models, ensemble methods like Random Forest and XGBoost shows better performance in classification, with an accuracy of 87%.

Experimental Results

	Accuracy	Specificity	F1 Score
SVM	0.85	0.8441	0.85
Random Forest	0.87	0.8707	0.87
Decision Tree	0.79	0.7643	0.2056
Naive Bayes	0.67	0.6046	0.67
XGBoost	0.87	0.8441	0.87


Choose classified model with best Accuracy (XG Boost
Do it for k-fold (In our case 5-fold)
Fold-1 Precsion : 0.89
Fold-2 Precsion : 0.91
Fold-3 Precsion : 0.86
Fold-4 Precsion : 0.88
Fold-5 Precsion : 0.91
Apply PCA (dimensionality reduction)

K-Fold

K-Fold Cross-Validation is a resampling technique used to evaluate the performance of a machine learning model. It helps ensure that the model generalizes well to unseen data by splitting the dataset into multiple subsets (folds).

Average Results Across 5 Folds:

Average Test Accuracy: 0.89

Average Precision: 0.8977

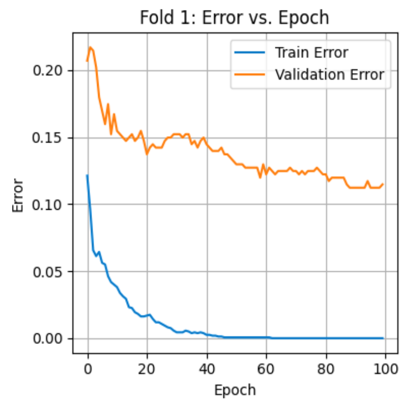
Average Recall: 0.8714

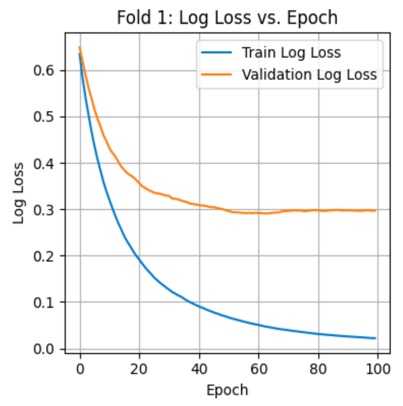
Average F1-Score: 0.8840

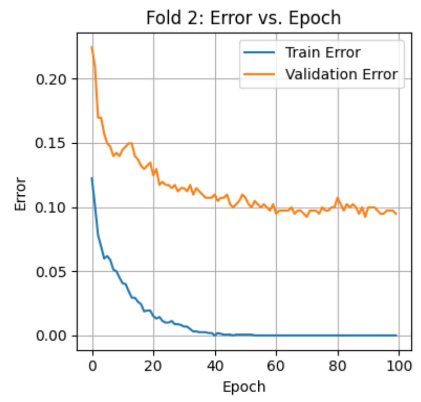
Average Training Loss: 0.0234

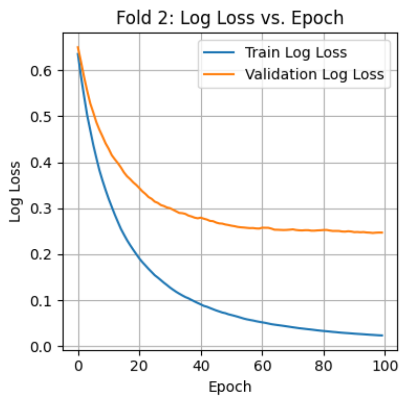
Average Test Loss: 0.2686

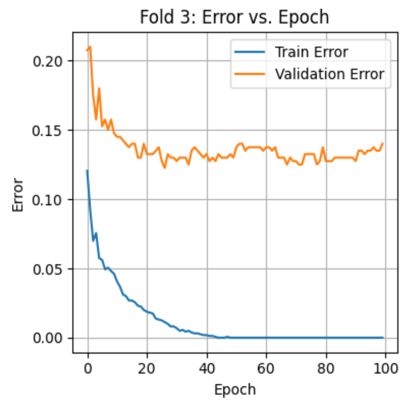
Average AUC: 0.9576

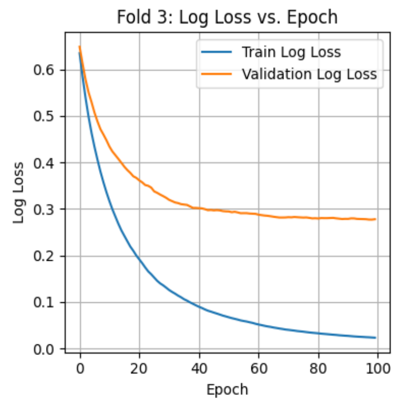


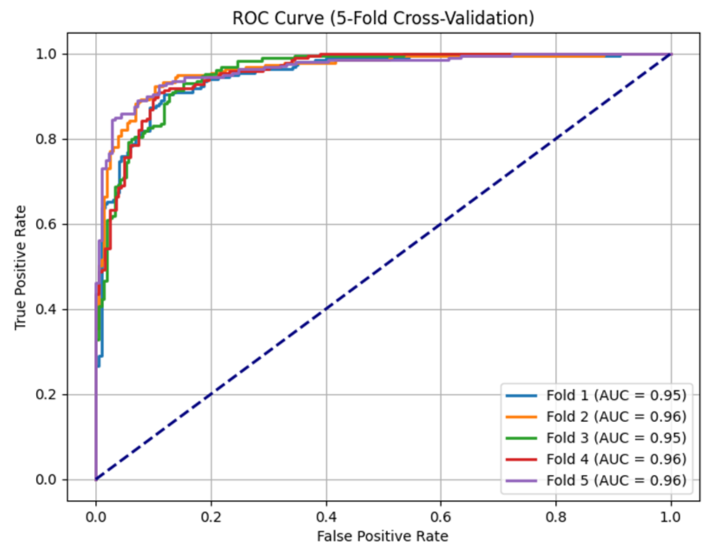












Experimental Results

- The XGBoost model showed good performance across several evaluation measures with an *accuracy* of *0.87*, i.e., 87% of the predictions were accurately classified. Also the *specificity* of *0.8441* and the model's *mean squared error (MSE)* is *0.1277*.

	Precision	Recall	F1-score	Support
Men	0.91	0.84	0.87	263
Women	0.84	0.90	0.87	238
Accuracy			0.87	501
Macro Avg	0.87	0.87	0.87	501
Weighted Avg	0.87	0.87	0.87	501

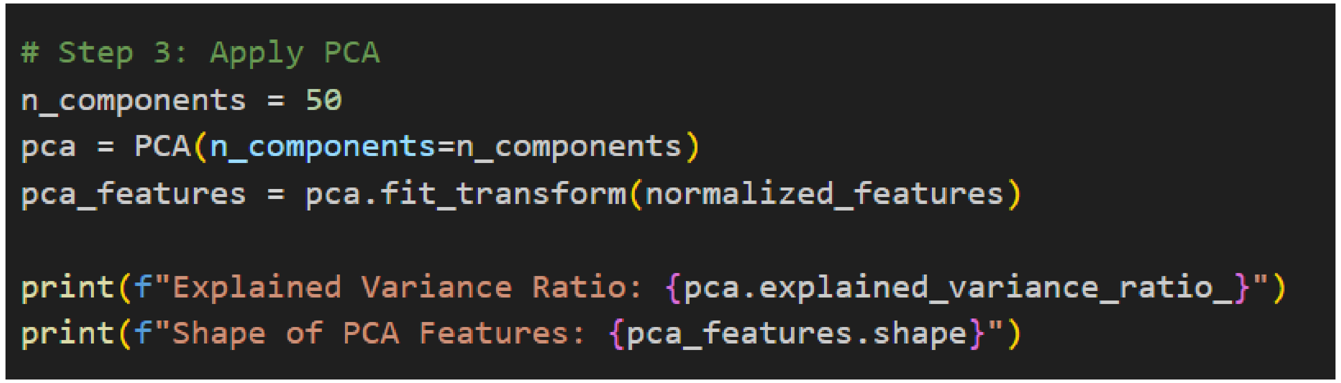
PCA

1. Preprocess the Data

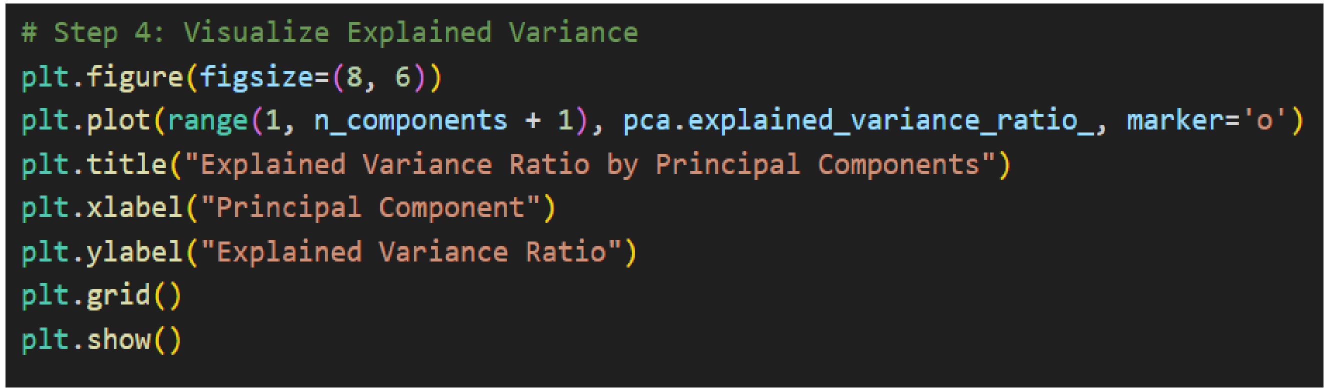
PCA requires standardized input to ensure all features contribute equally.

2. Apply PCA

Perform PCA to reduce dimensions and extract principal components.



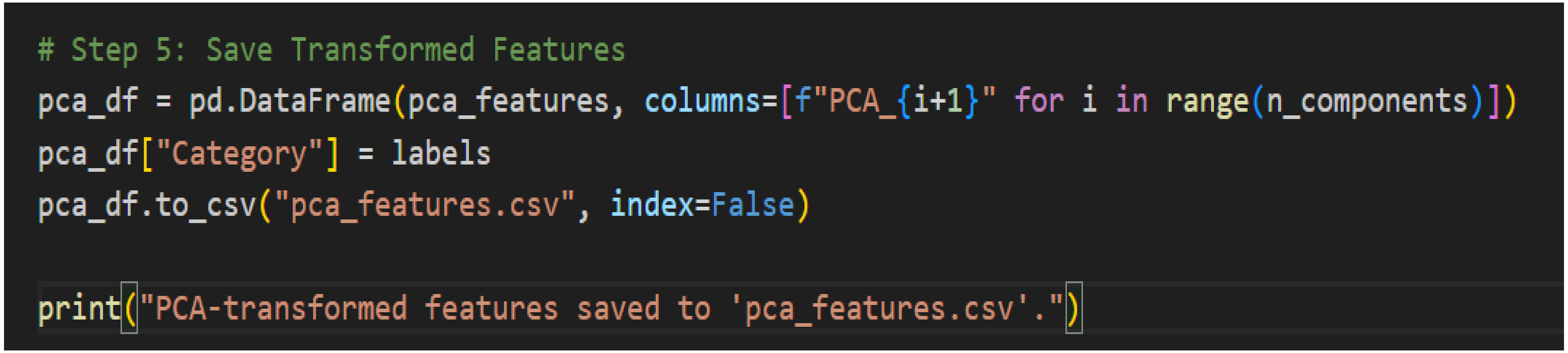
```python
# Step 3: Apply PCA
n_components = 50
pca = PCA(n_components=n_components)
pca_features = pca.fit_transform(normalized_features)

print(f"Explained Variance Ratio: {pca.explained_variance_ratio_}")
print(f"Shape of PCA Features: {pca_features.shape}")
```



```python
# Step 4: Visualize Explained Variance
plt.figure(figsize=(8, 6))
plt.plot(range(1, n_components + 1), pca.explained_variance_ratio_, marker='o')
plt.title("Explained Variance Ratio by Principal Components")
plt.xlabel("Principal Component")
plt.ylabel("Explained Variance Ratio")
plt.grid()
plt.show()
```

PCA (Continued)

- Evaluate how much variance each principal component explains and visualize.



```
# Step 5: Save Transformed Features
pca_df = pd.DataFrame(pca_features, columns=[f"PCA_{i+1}" for i in range(n_components)])
pca_df["Category"] = labels
pca_df.to_csv("pca_features.csv", index=False)

print("PCA-transformed features saved to 'pca_features.csv'.")
```

Conclusion

This research considered several machine learning and deep learning models for facial classification, including the effect of various feature extraction methods.

Key findings from the study are as follows:

1. Ensemble-based approaches (Random Forest, XGBoost) showed better performance, with highest accuracy of 87%.
2. Hand-crafted features (LBP, HOG, GLCM & Hu Moment) vs. Deep Learning Features (VGG19): Deep learning-based feature extraction showed a vast improvement in the classification outcome.
3. CNN (Convo 4 Block 5) was a strong deep learning method that utilized hierarchical feature extraction.
4. Feature subset selection and model tuning are of paramount importance for attaining optimal classification accuracy.

In future, the research can investigate even deeper deep learning architectures and mixtures of methods in order to further improve performance in face classification systems.