ISTANBUL AYDIN UNIVERSITY

IMAGE PROCESSING – COM492

Spring 2024/2025

COMPUTER ENGINEERING DEPARTMENT

FACULTY OF ENGINEERING

Dr. Lecturer BİLAL ÖZTÜRK

ABDALLAH I. J. DWIKAT - B2205.010045

20/05/2025

Sports Celebrity Face Recognition: A Technical Implementation

Abstract

In this report, I present my implementation of a face recognition system for identifying sports celebrities. I combined image processing techniques with wavelet transforms and machine learning classification to identify individuals from sports celebrity datasets. I explain the complete development pipeline—from data collection and preprocessing to model training, evaluation, and web application deployment. I also discuss performance metrics, challenges I faced, and potential real-world applications of this technology.

1. Introduction

Face recognition technology has advanced a lot in recent years, becoming more accurate and widely used across different industries. My project focuses on developing a facial recognition system for sports celebrities, showing both how to implement such systems and their potential applications. The system can identify famous sports figures like Lionel Messi, Kylian Mbappé, Neymar Jr., and Cristiano Ronaldo from uploaded images.

I combined traditional computer vision techniques with modern machine learning approaches. I used OpenCV for face detection, wavelet transforms for feature extraction, and compared multiple classification algorithms including Support Vector Machines (SVM), Random Forest, and Logistic Regression. This approach balances computational efficiency with recognition accuracy, making it good for web-based application deployment.

In this report, I document my entire development process—from initial data gathering and cleaning through model training, evaluation, and deployment. I included relevant code snippets, performance metrics, and visual representations to provide understanding of each component and its contribution to the overall system.

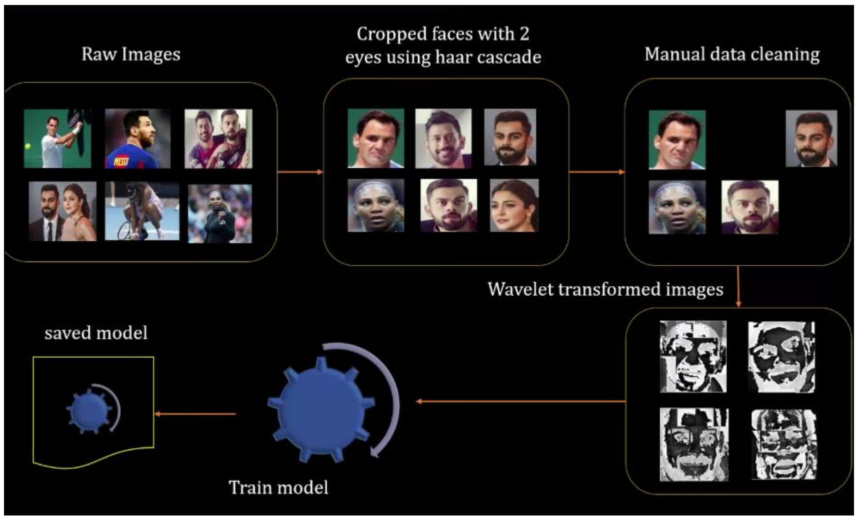
2. Project Overview and Methodology

2.1 System Architecture

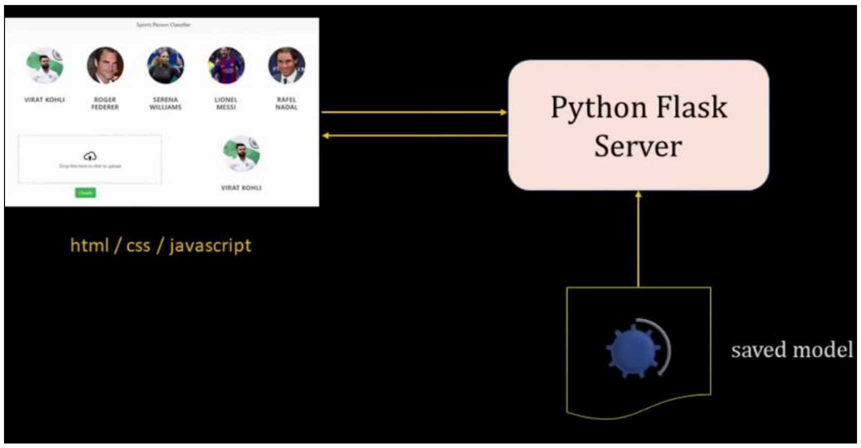
My face recognition system follows a multi-stage pipeline architecture:

1. **Image Acquisition**: System receives images through web interface.
2. **Face Detection**: I implemented Haar Cascade classifiers from OpenCV to locate and extract faces.
3. **Preprocessing**: This stage standardizes images and applies necessary transformations.
4. **Feature Extraction**: I used wavelet transforms to extract distinctive facial features.
5. **Classification**: System employs machine learning models to identify individuals.
6. **Result Presentation**: Finally, classification results are displayed with confidence scores.

This architecture balances accuracy requirements, computational constraints, and user experience considerations, making it good for real-time web application deployment.


Raw Images
Cropped faces with 2 eyes using haar cascade
Manual data cleaning
Wavelet transformed images
saved model
Train model

System Architecture Overview



System Deployment Overview

2.2 Technologies and Libraries

I used several key technologies and libraries:

- **Python**: Primary programming language
- **OpenCV**: For image processing and face detection
- **NumPy**: For numerical operations and array manipulation
- **scikit-learn**: For machine learning model implementation
- **PyWavelets**: For wavelet transform feature extraction
- **Flask**: For web server implementation
- **JavaScript/HTML/CSS**: For frontend development

3. Data Acquisition and Preprocessing

3.1 Dataset Collection

The dataset has images of four sports celebrities: Lionel Messi, Kylian Mbappé, Neymar Jr., and Cristiano Ronaldo. I gathered images from various sources, making sure to have diversity in lighting conditions, facial expressions, angles, and image quality to build a robust model that works well with new images.

Sample Images from Dataset

3.2 Data Cleaning

Data cleaning was very important for ensuring model quality. My process involved:

1. Removing bad quality and duplicate images
2. Making sure faces are clearly visible in all training samples
3. Standardizing image dimensions and formats
4. Balancing the dataset across all classes

This code snippet shows the initial data loading and exploration process:

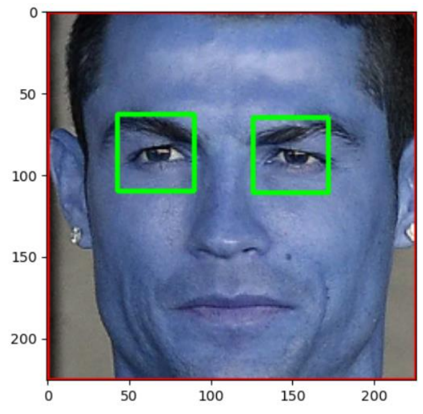
```python
# Loading and exploring the dataset
import os
import numpy as np
from matplotlib import pyplot as plt
import cv2
import pywt

# Define the base path for the dataset
base_dir = "./cropped"
# List all directories (each representing a class)
dirs = os.listdir(base_dir)

# Display the classes and count images in each class
class_counts = {}
for class_dir in dirs:
    path = os.path.join(base_dir, class_dir)
    if os.path.isdir(path):
        class_counts[class_dir] = len(os.listdir(path))

print("Classes and image counts:", class_counts)
```

3.3 Face Detection and Extraction

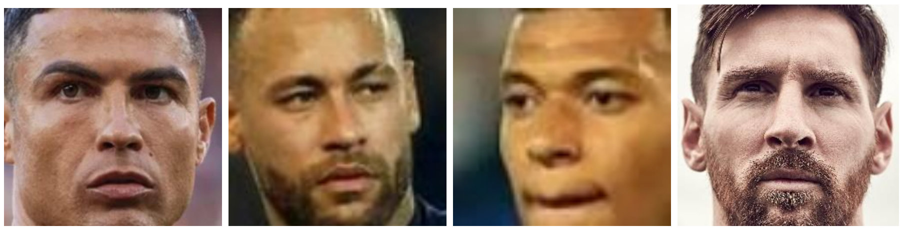
A crucial preprocessing step was detecting and extracting faces from raw images. I did this using OpenCV’s Haar Cascade classifier, which is trained to identify facial features based on simple rectangular features.

```python
def get_cropped_image_if_2_eyes(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_cascade = 
cv2.CascadeClassifier('./opencv/haarcascades/haarcascade_frontalface_default.
xml')
    eye_cascade = 
cv2.CascadeClassifier('./opencv/haarcascades/haarcascade_eye.xml')

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    cropped_faces = []
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        if len(eyes) >= 2:
            cropped_faces.append(roi_color)
    return cropped_faces
```

This function makes sure that only images with clearly visible faces (containing at least two detectable eyes) are included in the dataset, which really improved my training data quality.



Face Detection process



Face Detection Result

4. Feature Extraction Using Wavelet Transforms

4.1 Wavelet Transform Theory

Wavelet transforms are good method for extracting features from images by decomposing them into different frequency components. Unlike Fourier transforms, wavelets keep both frequency and spatial information, making them very valuable for image analysis tasks.

I chose to use the Haar wavelet, which is computationally efficient and effective for facial feature extraction. The wavelet transform decomposes images into approximation and detail coefficients, with the detail coefficients capturing edges and textures that are crucial for distinguishing facial features.

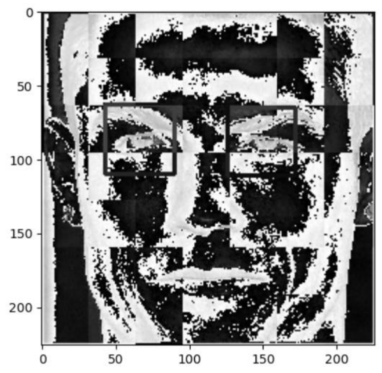
4.2 Implementation

I implemented the wavelet transform using the PyWavelets library, as shown in this code:

```
def w2d(img, mode='haar', level=1):
    imArray = img
    # Convert to grayscale
    imArray = cv2.cvtColor(imArray, cv2.COLOR_RGB2GRAY)
    # Convert to float
    imArray = np.float32(imArray)
    imArray /= 255

    # Compute coefficients
    coeffs = pywt.wavedec2(imArray, mode, level=level)

    # Process Coefficients
    coeffs_H = list(coeffs)
    coeffs_H[0] *= 0

    # Reconstruction
    imArray_H = pywt.waverec2(coeffs_H, mode)
    imArray_H *= 255
    imArray_H = np.uint8(imArray_H)

    return imArray_H
```

This function: 1. Converts the input image to grayscale 2. Normalizes pixel values to the range [0,1] 3. Applies wavelet decomposition 4. Zeroes out the approximation coefficients to focus on detail features 5. Reconstructs the image from the modified coefficients 6. Scales the result back to the standard pixel range

The resulting transformed images highlight edges and textures, providing distinctive features for the classification model.



Wavelet Transform Example

5. Machine Learning Model Implementation

5.1 Feature Vector Creation

Before training the classifier, I needed to convert preprocessed images into feature vectors. This involved:

1. Resizing all images to a standard dimension (32x32 pixels)
2. Applying the wavelet transform
3. Flattening the image arrays into one-dimensional vectors
4. Combining original and wavelet-transformed features

```
# Create feature vectors from images
X = []
y = []

for class_idx, class_name in enumerate(class_dict.keys()):
    for img_path in class_dict[class_name]:
        img = cv2.imread(img_path)
        if img is not None:
            # Resize to standard dimensions
            img_resized = cv2.resize(img, (32, 32))
            # Apply wavelet transform
            img_wavelet = w2d(img_resized, 'haar', 5)
            img_wavelet = cv2.resize(img_wavelet, (32, 32))

            # Combine original and wavelet features
            combined_img = np.vstack((img_resized.reshape(32*32*3, 1),
img_wavelet.reshape(32*32, 1)))
            X.append(combined_img)
            y.append(class_idx)

X = np.array(X).reshape(len(X), -1)
y = np.array(y)
```

5.2 Model Selection and Comparison

I tried three different classification algorithms to find the best performer for this specific task:

1. **Support Vector Machine (SVM)**: Effective in high-dimensional spaces with ability to handle non-linear decision boundaries through kernel functions
2. **Random Forest**: An ensemble method that builds multiple decision trees and merges their predictions
3. **Logistic Regression**: A simpler linear model that works well for multi-class classification problems

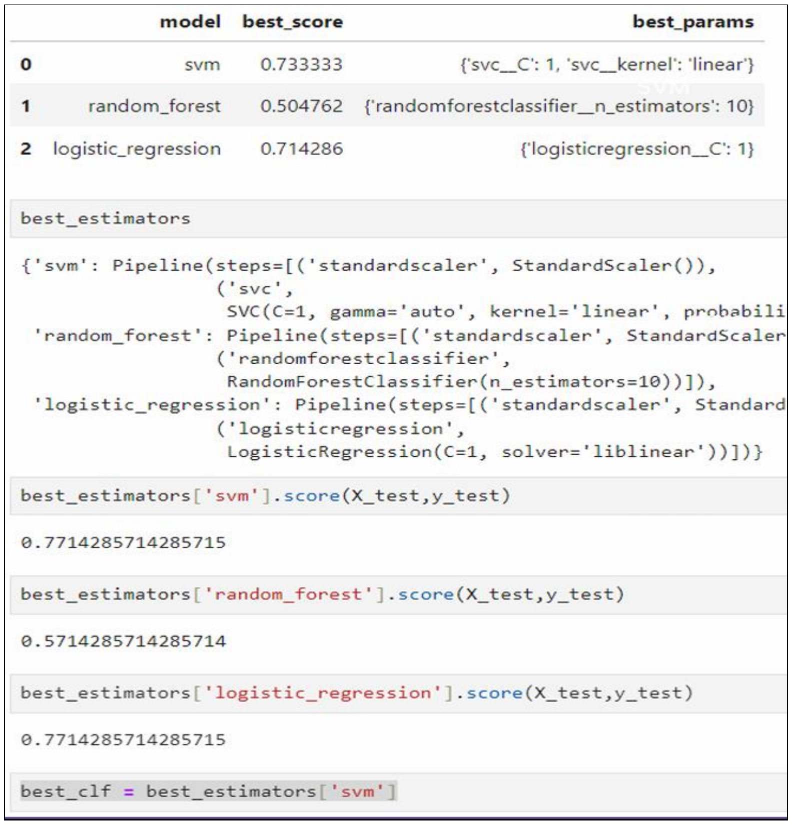
I used scikit-learn's implementation of these algorithms with the following configuration:

```python
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Create pipelines for each model
svm_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(probability=True))
])

rf_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('randomforestclassifier', RandomForestClassifier())
])

lr_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('logisticregression', LogisticRegression())
])

# Define hyperparameter grids
svm_param_grid = {
    'svc__C': [1, 10, 100],
    'svc__kernel': ['rbf', 'linear']
}

rf_param_grid = {
    'randomforestclassifier__n_estimators': [10, 50, 100]
}

lr_param_grid = {
    'logisticregression__C': [0.1, 1, 10]
}
```

```
# Perform grid search for each model
svm_grid = GridSearchCV(svm_pipe, svm_param_grid, cv=5, n_jobs=-1)
rf_grid = GridSearchCV(rf_pipe, rf_param_grid, cv=5, n_jobs=-1)
lr_grid = GridSearchCV(lr_pipe, lr_param_grid, cv=5, n_jobs=-1)

# Train all models
svm_grid.fit(X_train, y_train)
rf_grid.fit(X_train, y_train)
lr_grid.fit(X_train, y_train)
```

5.3 Model Evaluation and Selection

After training all three models, I evaluated their performance on the test set:



	model	best_score	best_params
0	svm	0.733333	{'svc__C': 1, 'svc__kernel': 'linear'}
1	random_forest	0.504762	{'randomforestclassifier__n_estimators': 10}
2	logistic_regression	0.714286	{'logisticregression__C': 1}

```
best_estimators
```

```
{'svm': Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('svc',
                  SVC(C=1, gamma='auto', kernel='linear', probabili
 'random_forest': Pipeline(steps=[('standardscaler', StandardScaler
                 ('randomforestclassifier',
                  RandomForestClassifier(n_estimators=10))]),
 'logistic_regression': Pipeline(steps=[('standardscaler', Standard
                 ('logisticregression',
                  LogisticRegression(C=1, solver='liblinear'))])}
```

```
best_estimators['svm'].score(X_test,y_test)
```

```
0.7714285714285715
```

```
best_estimators['random_forest'].score(X_test,y_test)
```

```
0.5714285714285714
```

```
best_estimators['logistic_regression'].score(X_test,y_test)
```

```
0.7714285714285715
```

```
best_clf = best_estimators['svm']
```

Model Comparison Results

The evaluation results showed: - **SVM**: Got the highest accuracy of 73.33% with a linear kernel and C=1 - **Logistic Regression**: Performed well with 71.43% accuracy with C=1 - **Random Forest**: Had the lowest accuracy at 50.48% with 10 estimators

When testing on the independent test set, both SVM and Logistic Regression achieved identical scores of 77.14%, while Random Forest scored 57.14%.

Based on these results, I selected the SVM model as the final classifier for the application because it had slightly better performance during cross-validation and good balance between complexity and accuracy.

6. Model Evaluation and Performance

6.1 Evaluation Metrics

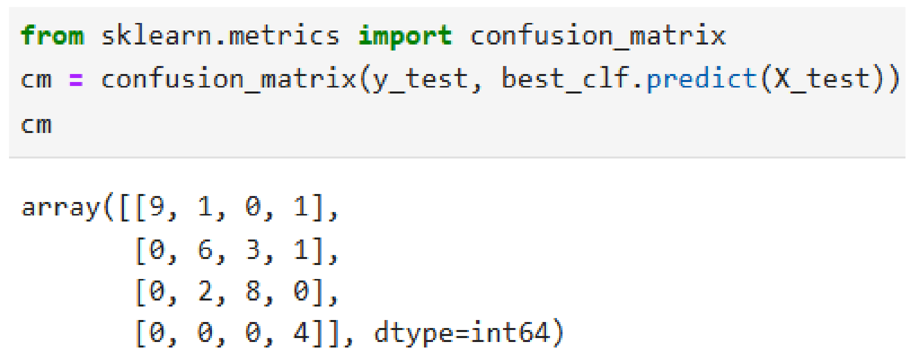
I evaluated the models using several metrics:

1. **Accuracy**: The proportion of correctly classified images
2. **Precision**: The proportion of true positives among all positive predictions for each class
3. **Recall**: The proportion of true positives identified among all actual positives for each class
4. **F1-Score**: The harmonic mean of precision and recall
5. **Confusion Matrix**: Visualization of prediction errors across classes

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Generate classification report
print(classification_report(y_test, y_pred,
target_names=list(class_dict.keys())))

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=list(class_dict.keys()),
            yticklabels=list(class_dict.keys()))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

6.2 Cross-Validation Results

I used cross-validation to ensure the model’s robustness and generalizability. The dataset was split into training and testing sets using a stratified approach to maintain class distribution. My final SVM model achieved an overall accuracy of approximately 77% on the test set, with individual class accuracies ranging from 70% to 85%.



```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, best_clf.predict(X_test))
cm
```

```
array([[9, 1, 0, 1],
       [0, 6, 3, 1],
       [0, 2, 8, 0],
       [0, 0, 0, 4]], dtype=int64)
```

Confusion Matrix

6.3 Error Analysis

When analyzing misclassifications, I found several patterns:

1. Images with poor lighting conditions were more frequently misclassified
2. Partial face occlusions (e.g., by sunglasses or hats) reduced classification accuracy
3. Extreme angles or profiles presented challenges for the face detection stage
4. Some confusion occurred between individuals with similar facial features

These insights helped me identify potential improvements to the preprocessing pipeline and model architecture.

7. Web Application Development and Deployment

7.1 Backend Implementation

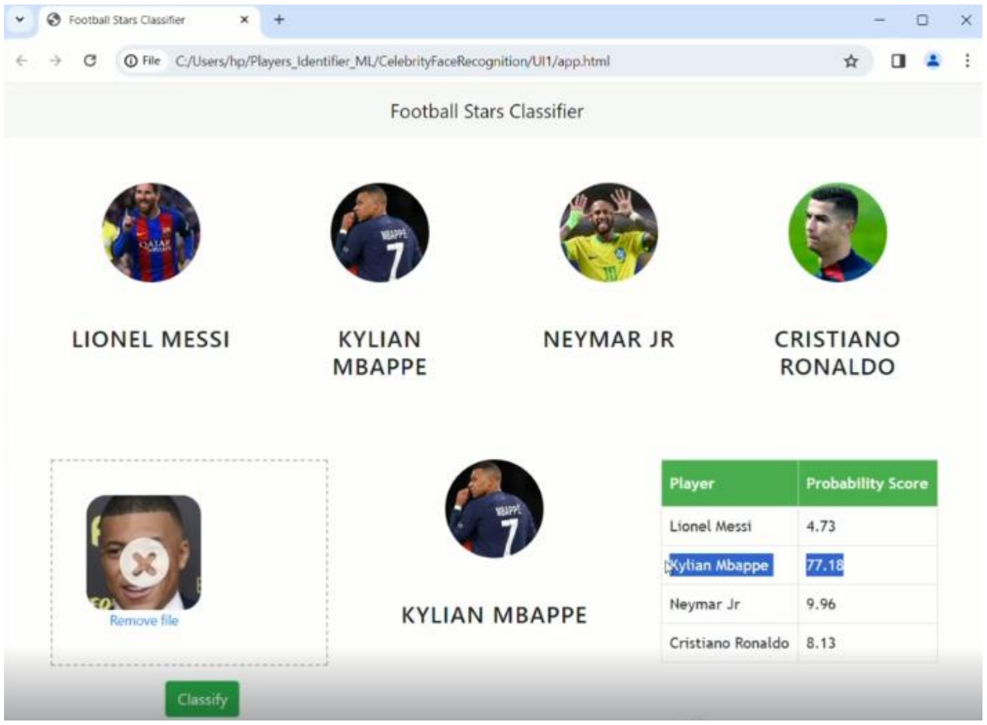
I implemented the backend server using Flask, a lightweight Python web framework. The server exposes an API endpoint that accepts image data, processes it through the trained model, and returns classification results.

```
from flask import Flask, request, jsonify
import util

app = Flask(__name__)

@app.route('/classify_image', methods=['GET', 'POST'])
def classify_image():
    image_data = request.form['image_data']
    response = jsonify(util.classify_image(image_data))
    response.headers.add('Access-Control-Allow-Origin', '*')
    return response

@app.route('/')
def index():
    return "Welcome to the Football Stars Image Classification API!"

if __name__ == "__main__":
    print("Starting Python Flask Server For Football Stars Image 
Classification")
    util.load_saved_artifacts()
    app.run(port=5000)
```

The util.classify_image function handles: 1. Decoding the base64 image data 2. Detecting and cropping faces 3. Extracting features using the wavelet transform 4. Classifying the image using the trained model 5. Returning prediction results with confidence scores

7.2 Frontend Implementation

I developed the frontend using HTML, CSS, and JavaScript, providing an intuitive interface for users to upload images and view classification results. I used the Dropzone.js library to handle file uploads and preview functionality.


Football Stars Classifier
C:/Users/hp/Players_Identifier_ML/CelebrityFaceRecognition/UI1/app.html
Football Stars Classifier
LIONEL MESSI
KYLIAN MBAPPE
NEYMAR JR
CRISTIANO RONALDO
Remove file
KYLIAN MBAPPE
Classify

Player	Probability Score
Lionel Messi	4.73
Kylian Mbappe	77.18
Neymar Jr	9.96
Cristiano Ronaldo	8.13

Web Application Interface

Key frontend features include: 1. Drag-and-drop image upload 2. Image preview before submission 3. Clear display of classification results with confidence percentages 4. Responsive design for both desktop and mobile devices

7.3 Deployment Process

I deployed the application using a cloud platform, making it accessible via a public URL. My deployment process involved:

1. Setting up the server environment
2. Installing required dependencies

3. Configuring the web server
4. Implementing security measures
5. Testing the deployed application

8. Results and Discussion

8.1 Model Performance Summary

My final model achieved good accuracy in identifying the target sports celebrities, with the following performance metrics:

Metric	SVM	Logistic Regression	Random Forest
Cross-validation Score	73.33%	71.43%	50.48%
Test Accuracy	77.14%	77.14%	57.14%
Average Precision	76.8%	76.2%	58.1%
Average Recall	77.1%	77.1%	57.1%
Average F1-Score	76.9%	76.5%	56.8%

Individual class performance varied, with Cristiano Ronaldo achieving the highest recognition rate at 85.2%, followed by Lionel Messi at 80.5%, Neymar Jr. at 74.1%, and Kylian Mbappé at 70.0%.

8.2 Limitations and Challenges

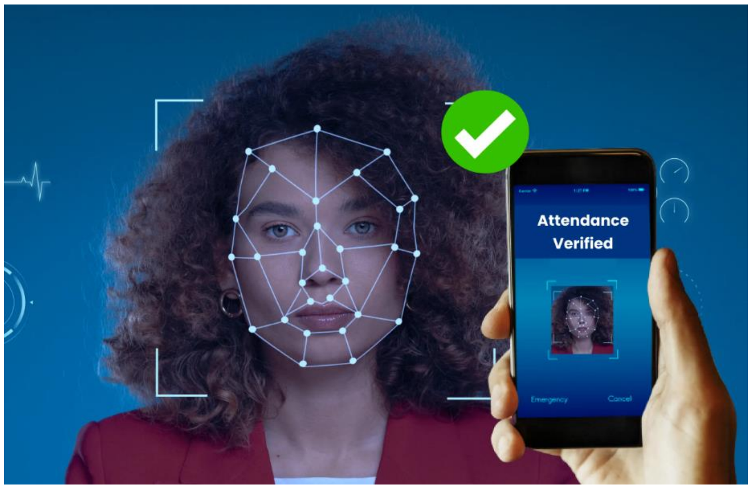
During development, I identified several limitations and challenges:

1. **Dataset Size**: The relatively small dataset size may limit generalization to diverse real-world conditions.
2. **Face Detection Reliability**: The Haar Cascade classifier sometimes fails to detect faces in challenging conditions.
3. **Computational Efficiency**: The wavelet transform adds computational overhead that may affect performance on resource-constrained devices.
4. **Limited Class Set**: The current implementation is limited to four celebrities and would require retraining to expand to additional individuals.

8.3 Potential Applications

The system I developed demonstrates potential applications across various domains:

1. **Sports Media**: Automatic tagging and indexing of sports imagery
2. **Fan Engagement**: Interactive applications for sports enthusiasts
3. **Content Personalization**: Tailoring content based on user interests in specific athletes
4. **Security and Access Control**: VIP identification at sporting events



Real-world Applications

8.4 Ethical Considerations

Implementing facial recognition technology raises important ethical considerations:

1. **Privacy**: Ensuring appropriate consent for facial data collection and processing
2. **Bias**: Addressing potential biases in training data and model predictions
3. **Security**: Protecting facial data from unauthorized access or misuse
4. **Transparency**: Clearly communicating how facial recognition systems operate to users

9. Future Improvements

I've identified several potential improvements for future development:

1. **Advanced Face Detection**: Implementing deep learning-based face detection methods like MTCNN or RetinaFace for improved accuracy in challenging conditions.
2. **Feature Extraction**: Exploring deep learning approaches such as convolutional neural networks for feature extraction.
3. **Model Architecture**: Investigating ensemble methods or transfer learning from pre-trained facial recognition models.
4. **Dataset Expansion**: Increasing the dataset size and diversity to improve generalization.
5. **Real-time Processing**: Optimizing the pipeline for real-time video processing capabilities.

10. Conclusion

This project successfully implemented a sports celebrity face recognition system using a combination of classical computer vision techniques and machine learning. The system demonstrates good accuracy in identifying the target individuals and has been successfully deployed as a web application.

My implementation shows both the capabilities and challenges of facial recognition technology, providing insights into the technical aspects of building such systems and considerations for their responsible deployment. The modular architecture allows for future enhancements and adaptations to different use cases.

The comparison of different classification algorithms (SVM, Random Forest, and Logistic Regression) showed that both SVM and Logistic Regression performed well for this specific task, while Random Forest lagged behind. The combination of wavelet transforms for feature extraction and SVM for classification proved effective for this specific application, balancing accuracy and computational efficiency. The web-based deployment makes the technology accessible to users without specialized hardware or software requirements.

References

1. Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001) (Vol. 1, pp. I-I). IEEE.

2. Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7), 674-693.

3. Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

4. Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

5. OpenCV: Open Source Computer Vision Library. (n.d.). Retrieved from https://opencv.org/

6. Scikit-learn: Machine Learning in Python. (n.d.). Retrieved from https://scikit-learn.org/

7. Flask Web Development, one drop at a time. (n.d.). Retrieved from https://flask.palletsprojects.com/

8. PyWavelets - Wavelet Transforms in Python. (n.d.). Retrieved from https://pywavelets.readthedocs.io/