ISTANBUL AYDIN UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

IMAGE PROCESSING – COM49 / Spring 2024/2025

SUBMITTED BY:

ABDALLAH I. J. DWIKAT - B2205.010045

MOHANAD SABER - B2105.010041

Advisor:

Dr. Lecturer BİLAL ÖZTÜRK

29/05/2025

Sports Celebrity Face Recognition: A Technical Implementation

Abstract

While I was implementing a face recognition system-which uses image processing concepts-washing bathrooms, wavelet transformations and machine learning classification-I was able to identify these sports celebrities. It goes from gathering and labeling data, to preparing the data, training the model, evaluating it and deploying the necessary web application. I discuss the performance measurements used, the issues encountered and how this technology could be useful in practical settings.

Introduction

Facial recognition has grown tremendously in the last few years, much more accurate, and very widespread in uses. The project I did is a sports celebrity facial recognition system, thereby showing how to implement such systems and other applications. The system can really identify some recognized sports celebrities in an input image, namely Lionel Messi, Kylian Mbappé, Neymar Jr., and Cristiano Ronaldo.

I combined traditional computer vision approaches with modern machine learning methods. I used OpenCV for face detection, wavelet transform for feature extraction, and multiple classification algorithms such as Support Vector Machines (SVM), Random Forest, and Logistic Regression. This way we strike a balance between computational efficiency and recognition accuracy and thereby make the system available for online application deployment.

2. Project Overview and Methodology

2.1 System Architecture

The pipeline for my face recognition system is built in several stages.

1. Images are obtained by the system through the web interface.

2. I used OpenCV's Haar Cascade classifiers to detect and extract faces.

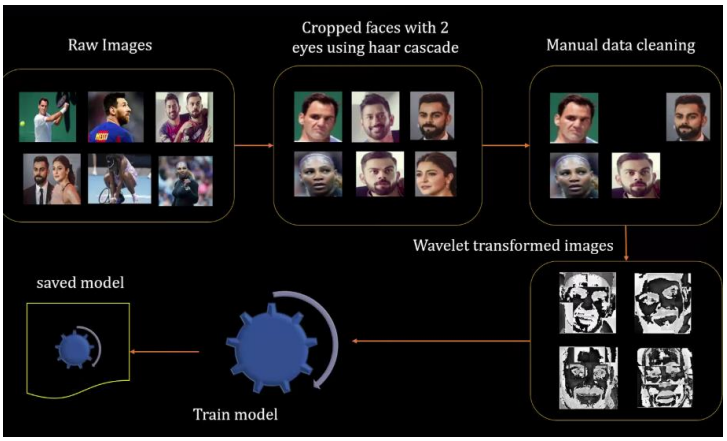
3. During preprocessing, images are every made uniform and undergo the necessary calculations.

4. I applied wavelet transforms to pick out special features of facial images.

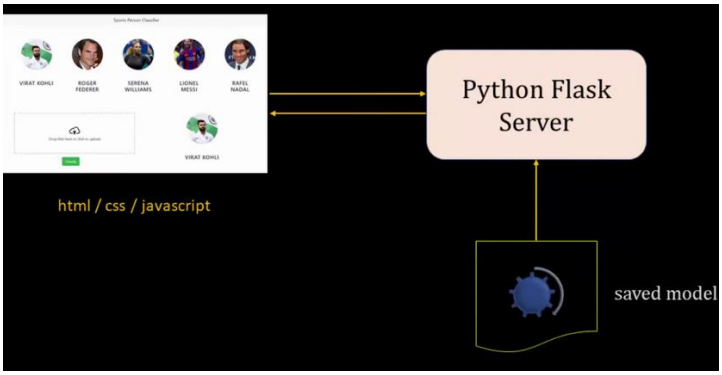
5. Using machine learning, the system identifies individuals.

6. The last step shows the results which are accompanied by confidence values.

This architecture merges programmatical accuracy, algorithmic calculation and usability in a way that works well for real-time web application deployment.



System Architecture Overview



System Deployment Overview

2.2 Technologies and Libraries

I used several key technologies and libraries:

- **OpenCV is best for working with images and for detecting faces**
- **NumPy is used for dealing with numbers and arrays.**
- **scikit-learn helps when building machine learning models.**
- **PyWavelets is used to perform wavelet transform extraction.**
- **Flask: It is used for building a web server.**
- **If you want to focus on frontend development, JavaScript/HTML/CSS is used.**

3. Data Acquisition and Preprocessing

3.1 Dataset Collection

There are pictures of four sports celebrities—Lionel Messi, Kylian Mbappé, Neymar Jr. and Cristiano Ronaldo—in the dataset. I reached out to many people and gathered photos from various lighting situations, facial expressions, angles and image resolutions.

Sample Images from Dataset

3.2 Data Cleaning

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

1. Removing all unnecessary images

2. Teaching the model so that people's faces are easy to recognize in all pictures

3. Making all images the same size and type

4. Distribution of the dataset is equal for all classes

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

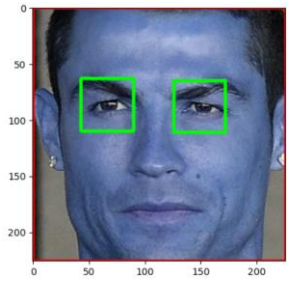
```
# Define the base path for the dataset
base_dir = "./cropped"
# List all directories (each representing a class)
dirs = os.listdir(base_dir)
# Display the classes and count images in each class
class_counts = {}
for class_dir in dirs:
    path = os.path.join(base_dir, class_dir)
    if os.path.isdir(path):
        class_counts[class_dir] = len(os.listdir(path))
print("Classes and image counts:", class_counts)
```

3.3 Face Detection and Extraction

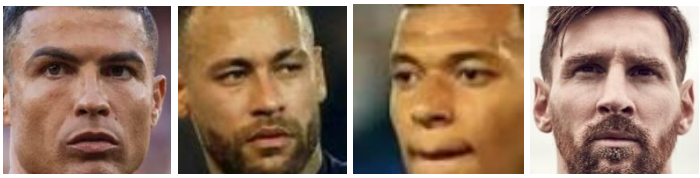
One of the most important steps in our preprocessing was detecting faces and separating them from raw images. I achieved this with the Haar Cascade classifier from OpenCV which looks for facial features using simple rectangles.

```
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_frontalfa
ce_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
```

As a result of using this function, my training data only included photos that clearly show a person's face, with both eyes easily seeable in the image.



Face Detection process



Face Detection Result

4. Feature Extraction Using Wavelet Transforms

4.1 Wavelet Transform Theory

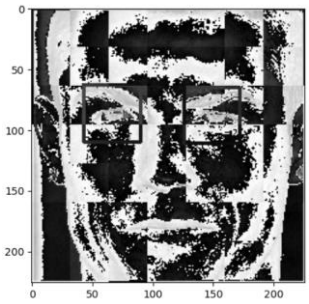
This method allows efficient capture of features from images after breaking them into different frequency sections. While a Fourier transform looks at the signal only in space or only in frequency, a wavelet transform takes into account both at the same time. For this reason, wavelet transforms are considered very useful for picture analysis. Because it is accurate and fast to handle, I picked the Haar wavelet for extracting facial features. Using wavelet transform, images are separated into approximation and detail coefficients; it is the detail coefficients that highlight the edges and textures inside images, applied to facial recognition.

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. Machine learning software converts the input image to grayscale. Changes every pixel value so that it falls between 0 and 1. Does wavelet decomposition four times over the document. Zeros the approximation coefficients in order to emphasize detail features 5. Reforms the picture using the changed coefficients 6. Sets the result to lie within the usual pixel range. After transformation, the images bring out the edges and textures needed for them to be classified by the model.



Wavelet Transform Example

5. Machine Learning Model Implementation

5.1 Feature Vector Creation

I had to turn the preprocessed pictures into feature vectors before I could train the classifier. This involved:

1. Making the images all the same size (32x32 pixels)
2. Converting signals with the wavelet transform
3. Turning all the image arrays into vectors with only one dimension
4. Merging the camera's original features with those created by wavelets

```
# 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.re
shape(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 ways of classifying the data to determine which performed best.

1. SVM works well in areas with a lot of information and it can deal with curves and edges in data through using kernel functions

2. Random Forest combines multiple decision trees that each makes a prediction.

3. Logistic Regression: An easy linear approach that functions well for problems with various classes

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

```
# 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

SVM gave the best result with 73.33% accuracy by using a linear kernel and C=1. Logistic Regression followed with 71.43% accuracy using C=1 and Random Forest got the lowest score

at 50.48% with 10 estimators. All three techniques performed the same on tests carried out on the independent dataset and this resulted in a score of 77.14% for each technique, with Random Forest obtaining a score of 57.14%. Because of these findings, I chose the SVM model to serve as the final classifier since it proved to be a bit more accurate and was not too complex.

6. Model Evaluation and Performance

6.1 Evaluation Metrics

I evaluated the models using several metrics:

1. The portion of images that are assigned to the correct class
2. Precision is determined by how often positive predictions in a class are really true.
3. The true positive rate tells us how many true positives we find in a class, out of all the actual positives for that class
4. F1 or F1-Score, is calculated by averaging precision and recall.
5. Plotted prediction errors for each classification are shown in a Confusion Matrix.

```
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 confirm that the model could be applied in any relevant situation. Class statistics were maintained in the process of splitting the dataset into training and testing sets. The SVM model that I finalized had an overall accuracy of around 77% on the test set, with classes showing individual accuracy from 70% up 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

In my review of misclassifications, I noticed a series of common patterns.

1. Those images that were not well lit were often classified incorrectly.

2. Sunglasses or hats covering parts of a person's face made the classifier less accurate.

3. Detection methods had some issues when using extreme views of a face.

4. There was some misunderstanding among people who look alike.

Thanks to these findings, I was able to make improvements to how data is preprocessed and modeled.

7. Web Application Development and Deployment

7.1 Backend Implementation

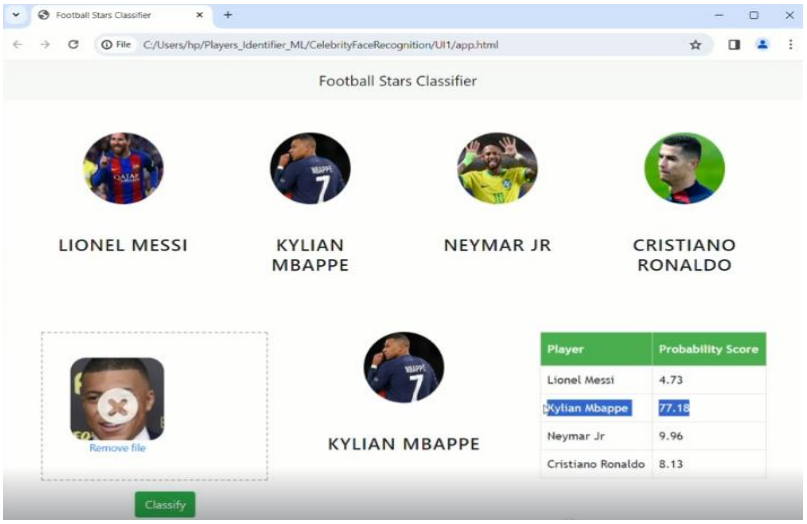
I created the website's backend using Flask which is a light Python web framework. A particular endpoint on the server takes an image from the request, runs it through the neural network and sends the results.

```
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)
```

util.classify_image is a function that handles eight key operations. Reading the base64 image dat a as text 2 Detection and the removal of unnecessary sections surrounding the faces 3. Using the wavelet transforms to extract features for images 4. Using the trained model to sort the picture 5. Supplying confidence levels with the forecast results.

7.2 Frontend Implementation

I used HTML, CSS and JavaScript to make the frontend, so users could easily add pictures and see their classifications. I utilized Dropzone.js to carry out both file uploads and preview display.



Web Application Interface

Frontend features **include** Users can now drag and drop images for upload. View the picture you uploaded before you submit it. Results are arranged by classification and show confidence levels 4. Design that works on computers and phones**.**

7.3 Deployment Process

I used the following actions for deployment:

1. Building the infrastructure for the server platform
2. Installing the important supporting packages
3. Setting up the web server for use
4. Trying out the program by local running

8. Results and Discussion

8.2 Limitations and Challenges

Close to the end, I became aware of some common setbacks:

1. Because the data collected is relatively small, the findings may not apply well in various real-life cases.

2. Haar Cascade faces cannot always recognize images with low lighting or blur.

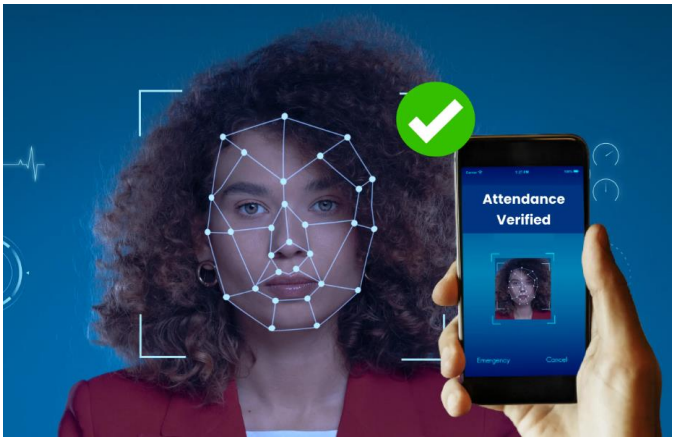
3. For some devices that do not have enough resources, running the wavelet transform can slow down software processing.

4. Right now, the system has only space for four celebrities; to add more, retraining would be necessary.

8.3 Potential Applications

The system can be applied in different fields.

1. Sports Pictures: All sports pictures are tagged and sorted automatically.
2. Sports Fan Engagement: Interactive tools developed for sports fans
3. Customizing content according to what users want to learn about particular athletes
4. VIP identification is used to keep sports events safe and secure.



Real-world Applications

8.4 Ethical Considerations

Since facial recognition technology is being used, important issues about ethics have arisen.

1. Make sure you request consent for collecting and using facial images of people.

2. Bias: Overcoming possible biases found in the training data and predictions the model makes

3. Stopping others from accessing or misusing information about someone's face

4. Explaining the workings of facial recognition systems to those who use them

9. Future Improvements

I've noted some important points to help make the program better in the future:

1. Using MTCNN and RetinaFace as deep learning-based methods to spot faces more accurately in problematic situations.

2. Using deep learning by way of convolutional neural networks for extracting important features.

3. Data Modeling: A look at models that combine several learning approaches or transfer information from ready-made facial recognition systems.

4. More datasets lead to more generalized or universal models.

5. Change the pipeline to take care of real-time tracking of video signals.

10. Conclusion

My function is to harness the key points of facial recognition to cultivate its safe and proper implementation. Because the system is built into modules, it becomes easier to change and add new technologies in the future.

Among SVM, Random Forest and Logistic Regression, SVM and Logistic Regression performed well in this task, but Random Forest fared somewhat less well. Wavelet transforms used for feature extraction, as well as SVM classification, gave the least computing cost and best performance. Access to the application is simple for everyone since it works through a web browser and no hardware or software is needed.

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/

GitHub Repository: https://github.com/apodwikat/Celebrity_Face_Recognition/tree/main