Project: Building a Face Detection System

Build a face recognition system using the Histogram of Oriented Gradients (HOG) for feature extraction and Support Vector Machine (SVM) as the primary classification model. Additionally, a tree-based method will be used for comparison and accuracy evaluation. The system will be trained and tested on the BioID Face Detection Database dataset. The project includes visualizing some images with bounding boxes and labels to demonstrate the system's effectiveness.

Step-by-Step Implementation

1. Dataset: BioID Face Detection Database

- **Description**: The LFW dataset contains over 1,521 images of faces collected from the web, labeled with the names of the individuals. It is widely used for face detection tasks.
- **Usage**:
  - Load the dataset using Python libraries like OpenCV, scikit-learn or dlib.
  - Preprocess the data to ensure consistent image sizes and grayscale conversion.

2. Feature Extraction: Histogram of Oriented Gradients (HOG)

- **Purpose**: HOG is a feature descriptor used to capture the structural information of an image by analyzing gradients and orientations.
- **Implementation**:
  1. Divide the image into small regions (cells).
  2. Compute the gradient orientation and magnitude for each pixel.
  3. Create histograms of oriented gradients for each cell.
  4. Normalize the histograms to improve invariance to lighting and contrast.
- **Tools**: Use OpenCV, scikit-image or dlib for HOG feature extraction.

3. Classification Models

3.1. Support Vector Machine (SVM)

- **Role**: SVM will serve as the primary classifier to distinguish between different individuals.
- **Implementation**:
  - Use scikit-learn's SVC class.
  - Choose a kernel (e.g., linear or RBF) based on experimentation.
  - Tune hyperparameters like C and gamma using grid search or cross-validation.

3.2. Tree-Based Method

- **Role**: To compare accuracy and provide additional insights.
- **Options**: Use decision trees, random forests, or gradient-boosted trees.
- **Implementation**:
  - Use scikit-learn's DecisionTreeClassifier or RandomForestClassifier, XGBoost or AdaBoost
  - Evaluate feature importance and accuracy.

4. Model Training and Evaluation

- **Data Splitting**:
  - Split the dataset into training and testing sets (e.g., 80% training, 20% testing).
- **Metrics**:
  - Accuracy, precision, recall, and F1-score.
  - Use a confusion matrix to analyze misclassifications.
- **Evaluation**:
  - Compare the performance of SVM and the tree-based model.
  - Perform k-fold cross-validation to ensure robust evaluation.

5. Visualization

- **Bounding Boxes and Labels**:
  - Use a sliding window approach with the trained model for face detection to locate faces in images.
  - Draw bounding boxes around detected faces and annotate them with predicted labels.
- **Tools**: Use matplotlib or OpenCV for visualization.

Submission Guidelines

1. **Code Files**:
   - Submit all Python source files used in the project.
   - Include a `requirements.txt` file listing all necessary libraries and their versions.
2. **Documentation**:
   - Provide a detailed README file explaining the project structure, how to run the code, and dependencies.
   - Include inline comments in the code for clarity.
3. **Report**:
   - Submit a project report summarizing the methodology, results, and insights.
   - Include tables, graphs, or images to support your findings.
4. **Evaluation Results**:
   - Provide confusion matrices and classification reports for both models.
   - Include screenshots of visualized results (bounding boxes and labels).
5. **Testing Instructions**:
   - Ensure the code runs without errors on a standard Python environment.

    - Specify any additional setup required.
6. **Submission Format**:
    - Compress all files into a single ZIP archive named `FaceRecognition_Project_<YourName>.zip`.
7. **Deadline**:
    - Ensure submission by the specified deadline to avoid penalties.