Face Mask Detection Project Documentation

1. Project Overview

The Face Mask Detection Project aims to develop a computer vision system capable of detecting whether individuals in images or live video feeds are wearing face masks. This system leverages machine learning and image processing techniques to classify faces as either "With Mask" or "Without Mask." The project includes scripts for preprocessing images, extracting features, training machine learning models, and performing real-time detection using a webcam or processing static images. The primary applications include public health monitoring and automated surveillance in environments where mask-wearing is required.

The project consists of four main Python scripts:

- `preprocess.py`: Handles data preprocessing, feature extraction, and PCA transformation.
- `train.py`: Trains and compares multiple machine learning models, saving the best-performing model.
- `detect.py`: Processes static images to detect and classify faces with respect to mask usage.
- `PREDICT.PY`: Performs real-time face mask detection using a webcam feed.

2. Dataset Description

The dataset is expected to be stored in a folder named `dataset` with two subfolders:

- `with_mask`: Contains images of individuals wearing face masks.
- `without_mask`: Contains images of individuals not wearing face masks.

Each image is in a standard format (e.g., `.jpg` or `.png`). The dataset is used to extract features for training machine learning models. The labels are binary:

- `0`: Without Mask
- `1`: With Mask

The dataset size is not explicitly specified, but the preprocessing script (`preprocess.py`) processes all images in the subfolders, indicating flexibility in handling varying dataset sizes. The assumption is that the dataset contains sufficient samples to train robust models, with balanced representation of both classes.

3. Data Preprocessing

Data preprocessing is handled by the `preprocess.py` script and is critical for preparing images for feature extraction. The preprocessing steps include:

- **Image Loading and Grayscale Conversion**: Images are loaded using OpenCV (`cv2.imread`) in grayscale mode to reduce dimensionality and focus on intensity information.
- **Resizing**: Images are resized to a uniform size of 64x64 pixels to ensure consistency in feature extraction.
- **Normalization**: Pixel values are normalized to the range [0, 255] using `cv2.normalize` with `NORM_MINMAX` to standardize intensity values.
- **Quality Check**: Images with low standard deviation (indicating low contrast or invalid images) are skipped.

For real-time detection (`PREDICT.PY`), additional preprocessing includes:

- **Padding for Resizing**: The `resize_with_padding` function ensures that face regions maintain their aspect ratio by adding padding to fit the 64x64 target size.

4. Feature Extraction

Feature extraction is a core component of the project, implemented in `preprocess.py`. Three types of features are extracted from each preprocessed image:

1. **Histogram of Oriented Gradients (HOG)**:
    - Computes gradients using differences in pixel intensities (`compute_gradients`).
    - Generates histograms of gradient orientations in 8x8 pixel cells with 9 bins (`compute_histograms`).
    - Normalizes features over 2x2 cell blocks to reduce sensitivity to illumination changes (`normalize_blocks`).
    - Produces a feature vector capturing edge and shape information.
2. **Chain Code**:
    - Applies binary thresholding to the image and extracts contours (`cv2.findContours`).
    - Computes chain codes based on 8-directional movements between contour points (`compute_chain_code`).
    - Generates an 8-bin histogram of direction codes, normalized to sum to 1.
    - Captures boundary shape information.
3. **Gray-Level Co-occurrence Matrix (GLCM)**:

    - Resizes the image to 64x64 and quantizes pixel values to 16 levels.
    - Computes GLCMs for distances [1] and angles [0, π/4, π/2, 3π/4] using `graycomatrix`.
    - Extracts five statistical features: contrast, dissimilarity, homogeneity, energy, and correlation (`graycoprops`).
    - Captures texture information.

The features are concatenated into a single feature vector per image. Principal Component Analysis (PCA) is applied to reduce dimensionality to 100 components (`apply_pca`), improving computational efficiency and reducing noise.

5. Machine Learning Algorithm

The project employs a Support Vector Machine (SVM) with a radial basis function (RBF) kernel as the primary classifier, implemented in `train.py`. The SVM is chosen for its effectiveness in handling high-dimensional data and non-linear classification tasks. The model is trained on PCA-transformed features, with the following configuration:

- **Kernel**: RBF
- **C**: 1.0 (regularization parameter)
- **Gamma**: 'scale' (automatically set based on feature variance)

Additionally, `train.py` compares the SVM with three other classifiers to evaluate performance:

- **Logistic Regression**: A linear classifier with a maximum of 1000 iterations.
- **Random Forest**: An ensemble method with 100 trees.
- **K-Nearest Neighbors (KNN)**: A distance-based classifier with 5 neighbors.

The SVM model is saved as `svm_face_mask_model.pkl` for use in detection scripts.

6. Model Training and Evaluation

The training process is implemented in `train.py`:

- **Data Loading**: Loads PCA-transformed features (`features_pca.npy`) and labels (`labels.npy`) from the `files` directory.
- **Train-Test Split**: Splits data into 80% training and 20% testing sets using `train_test_split` with a random seed of 42 for reproducibility.
- **Training**: Each model is trained on the training set, and training time is recorded.
- **Evaluation**: Models are evaluated on the test set using accuracy (`accuracy_score`).

- **Output**: Prints accuracy and training time for each model. The SVM model is saved for use in detection.

The evaluation focuses on accuracy, but additional metrics (e.g., precision, recall) could be added for a more comprehensive analysis.

7. Tools and Libraries Used

The project relies on the following Python libraries:

- **OpenCV (cv2)**: For image processing, face detection (Haar cascades), and visualization.
- **NumPy**: For numerical computations and array operations.
- **scikit-learn**: For machine learning algorithms (SVM, Logistic Regression, Random Forest, KNN), PCA, and evaluation metrics.
- **scikit-image**: For GLCM feature extraction (`graycomatrix`, `graycoprops`).
- **joblib**: For saving and loading models and PCA transformations.
- **time**: For measuring training time.

The project uses Haar cascade classifiers (`haarcascade_frontalface_default.xml`) from OpenCV for face detection.

8. Dataflow

The dataflow of the Face Mask Detection Project follows a sequential pipeline across the provided scripts, ensuring data is processed from raw images to final predictions:

1. **Dataset Input (`preprocess.py`)**:
    - Raw images are loaded from the `dataset` folder (`with_mask` and `without_mask` subfolders).
    - Images undergo preprocessing: grayscale conversion, resizing to 64x64 pixels, normalization, and quality checks.
2. **Feature Extraction (`preprocess.py`)**:
    - For each preprocessed image, HOG, chain code, and GLCM features are computed and concatenated into a feature vector.
    - PCA is applied to reduce feature dimensionality to 100 components.
    - Reduced features and labels are saved as `features_pca.npy` and `labels.npy`, and the PCA model is saved as `pca_model.pkl`.
3. **Model Training (`train.py`)**:
    - Loads PCA-transformed features and labels.
    - Splits data into 80% training and 20% testing sets.

    - Trains SVM, Logistic Regression, Random Forest, and KNN models.
    - Evaluates models on the test set and saves the SVM model as `svm_face_mask_model.pkl`.
4. **Static Image Detection (`detect.py`)**:
    - Loads images from the `images` folder.
    - Detects faces using OpenCV's Haar cascade classifier.
    - Preprocesses detected face regions (resize to 64x64, normalize).
    - Extracts HOG, chain code, and GLCM features, applies PCA transformation using the saved PCA model.
    - Uses the SVM model to predict "Mask" or "No Mask."
    - Annotates images with rectangles and labels, saving results to the `result` folder.
5. **Real-Time Detection (`PREDICT.PY`)**:
    - Captures live video frames from a webcam.
    - Detects faces in each frame using the Haar cascade classifier.
    - Preprocesses face regions with resizing (with padding) and normalization.
    - Extracts features, applies PCA transformation, and predicts using the SVM model.
    - Displays annotated frames with rectangles and labels ("With Mask" or "Without Mask") in real time.

This pipeline ensures a consistent flow from raw image input to feature extraction, model training, and final classification, supporting both static and real-time applications.

9. Results Summary

The project achieves the following outcomes:

- **Static Detection**: detect.py annotates images with mask/no-mask labels, saved in result.
- **Real-Time Detection**: PREDICT.PY displays live webcam annotations.
- **Performance**: SVM accuracy is 94.66% (0.1331s training); Logistic Regression is 89.99%.
- **Features**: HOG, chain code, GLCM ensure robust classification.
- **Scalability**: Handles varying dataset sizes and image sources.