Project Analysis Document

This document provides a detailed breakdown of the "AI Face Mask Detection System," analyzing its architecture, components, and data flow based on the provided project files.

1. Project Overview

The project is a real-time face mask detection system. It uses a deep learning model to process a live webcam feed, identify human faces, and classify them as either "wearing a mask" or "not wearing a mask."

The system is built on a client-server architecture:

- **Backend:** A **Flask** (Python) server handles the AI-powered video processing and provides a REST API for control.
- **Frontend:** A modern **HTML/CSS/JavaScript** single-page application provides the user interface for monitoring the video feed and viewing statistics.
- **AI Model:** A **TensorFlow/Keras** model, built using transfer learning on **MobileNetV2**, performs the core image classification.

2. Core Components & File Analysis

Component 1: The AI Model (The "Brain")

- **Files:** train.py, face_mask_detector.h5, training_history.png
- **train.py (The Training Script):**
    - **Purpose:** This script is used to create, train, and save the AI model. It is not run during normal operation of the web app.
    - **Process:**
        1. **Data Loading:** It loads images from data/with_mask (labeled as 0) and data/without_mask (labeled as 1).
        2. **Preprocessing:** It uses OpenCV (cv2) to resize all images to 224x224 pixels, the input size required by the MobileNetV2 model.
        3. **Data Augmentation:** It uses ImageDataGenerator to create new variations of the training images (rotation, zoom, flips). This makes the model more robust and prevents overfitting.
        4. **Model Architecture:** It uses **transfer learning**.
            - It loads the MobileNetV2 model, pre-trained on the massive ImageNet dataset.
            - It "freezes" the original MobileNetV2 layers so they are not retrained.
            - It adds a new "head" on top: a GlobalAveragePooling2D layer, a Dropout(0.5) layer (to prevent overfitting), and a final Dense(2, activation='softmax') layer to output the probabilities for the two classes (mask/no-mask).
        5. **Training & Evaluation:** It trains the new "head" on the mask dataset. After training, it evaluates the model's performance and prints a classification report.

6. **Output:** It saves the final, trained model as face_mask_detector.h5 and plots the training/validation accuracy and loss, saving it as training_history.png.

- **face_mask_detector.h5 (The Trained Model):**
  - This is the final, serialized output of the train.py script. It contains the learned weights and architecture of your AI model, ready to be loaded by app.py for performing predictions.
- **training_history.png (The "Report Card"):**
  - This image (which I can see from the upload) shows that the model trained very well. The validation accuracy is high (around 98-99%) and closely follows the training accuracy, indicating that the model is highly accurate and not overfitted.

Component 2: The Backend Server (The "Engine")

- **File:** app.py (The Flask Server)
- **Purpose:** This is the main application file. It runs a web server that performs two primary jobs: serving the frontend and processing the video.
- **Key Functions:**
  1. **Model Loading:** It loads the face_mask_detector.h5 model (if it exists) and the OpenCV Haar Cascade file (haarcascade_frontalface_default.xml) used for fast face detection.
  2. **Webpage Serving:** The @app.route('/') endpoint serves the index.html file to the user's browser.
  3. **Video Streaming (/video_feed):** This is the core of the application.
     - It uses a "generator" function (gen_frames) to stream video one frame at a time as a Motion JPEG (MJPEG).
     - Inside its loop, it reads a frame from the webcam.
     - It calls detect_and_predict_mask which:
       - Converts the frame to grayscale.
       - Uses the **Haar Cascade** to find the coordinates (x, y, w, h) of all faces.
       - For each face, it extracts the face Region of Interest (ROI).
       - It preprocesses this face ROI (resize to 224x224, normalize) and feeds it to the loaded mask_model.
       - The model predicts (mask_prob, no_mask_prob).
       - It draws a **green** box (for "Mask") or **red** box (for "No Mask") directly onto the video frame.
     - It then yields this processed frame to the browser.
  4. **API Endpoints:** It provides a REST API to control the app from the frontend:
     - /api/toggle-detection: Starts/stops the detection process.
     - /api/set-sensitivity, /api/set-threshold, /api/capture-snapshot: Provide hooks for controlling settings (though some are placeholders).
     - /api/detection-status: Reports the server's current settings.

Component 3: The Frontend Interface (The "Dashboard")

- **File:** index.html
- **Purpose:** This is the single-page web application the user sees and interacts with.
- **Key Functions:**
  1. **Layout:** It defines the HTML structure, including the video feed area, the statistics

panel (Total, Mask, No Mask, Accuracy), the control buttons, and the log panel.
2. **Styling:** It uses modern CSS (defined in the <style> tag) for a clean, responsive dashboard look.
3. **Video Display:** It uses a simple <img> tag. Its src is set to the /video_feed endpoint, which is how it displays the MJPEG stream from the Flask server.
4. **Interactivity (JavaScript):**
    - It handles button clicks to toggleDetection or captureSnapshot by sending fetch requests (POST) to the backend API endpoints.
    - It updates the UI (e.g., changing the toggle button's icon and text) based on the response.
    - It includes an "Export Data" feature that generates a JSON file of the session's logs.

Component 4: Dependencies

- **File:** requirements.txt
- **Purpose:** This file lists all the Python libraries needed to run the project. pip install -r requirements.txt would set up the environment.
- **Key Libraries:**
    - tensorflow: To load and run the .h5 model.
    - opencv-python: For webcam access, face detection (Haar), and image processing.
    - flask & flask-cors: To create the web server and API.
    - numpy: For numerical operations on image data.

3. Key Observation: Data Disconnect

There is a critical disconnect between the backend's detections and the frontend's statistics:

- **Backend (app.py):** Performs *actual* mask detection. It *draws* the results (red/green boxes) onto the video frames. However, it **does not send the statistical data** (e.g., {'status': 'No Mask', 'confidence': 0.98}) back to the frontend as JSON.
- **Frontend (index.html):** The statistics shown on the dashboard (Mask Count, No Mask Count) are **simulated**. The JavaScript has its own logic (Math.random()) to generate fake detection events to populate the dashboard.

Conclusion: The video feed is real, but the dashboard numbers are not connected to the video feed. To fix this, you would need to implement a **WebSocket** or **Server-Sent Events (SSE)** channel to push the JSON detection results from app.py to index.html in real-time.

System Blueprint (How it Works)

This blueprint is broken into two phases: **Phase 1 (The Build)**, which you do once to create the model, and **Phase 2 (The Operation)**, which is how the application runs.

Phase 1: The Build (Training the Model)

This flow describes the train.py script.

1. **Input Data:** You provide two folders: data/with_mask and data/without_mask.
2. **Execute train.py:** You run the command python train.py.

3. **Load & Preprocess:** The script loads all images, converts them to RGB, and resizes them to 224x224.
4. **Define Model:** It loads the pre-trained MobileNetV2 and adds a new custom classification "head."
5. **Data Augmentation:** An ImageDataGenerator is created to apply random transformations (zoom, rotate, etc.) to the training data.
6. **Training:** The model is trained (model.fit) using the augmented data. The script fine-tunes the new "head" to specialize in mask detection.
7. **Output:** The script saves two files:
    - models/face_mask_detector.h5 (the trained model).
    - training_history.png (the performance plot).

Phase 2: The Operation (Running the Application)

This flow describes the real-time interaction between the user, the frontend (index.html), and the backend (app.py).

1. **Start Server:** The user runs python app.py. The Flask server starts, loads the face_mask_detector.h5 model, and opens the webcam.
2. **User Access:** The user opens http://localhost:5001 in their web browser.
3. **Load Interface:**
    - **Browser:** Sends a GET request to /.
    - **Flask (app.py):** Responds by sending the index.html file.
    - **Browser:** Renders the HTML, showing the dashboard and buttons.
4. **Start Video Feed:**
    - **Browser (index.html):** The <img> tag automatically sends a GET request to /video_feed.
    - **Flask (app.py):** This request starts the gen_frames() loop.
5. **Real-time Detection Loop (Backend):** For every single frame from the webcam, the app.py server does the following:
    - **a. Capture:** Grabs one frame from cv2.VideoCapture.
    - **b. Detect Face:** Uses the fast face_cascade to find the (x, y, w, h) coordinates of a face.
    - **c. Preprocess Face:** Extracts the face ROI and resizes it to 224x224.
    - **d. Predict:** Feeds the face ROI into mask_model.predict().
    - **e. Get Result:** The model returns probabilities, e.g., [0.02, 0.98], which means 98% "No Mask."
    - **f. Draw:** The server draws a **red box** and the "No Mask" text onto the *original* frame.
    - **g. Stream:** The server encodes this modified frame as a JPEG and sends it to the browser.
6. **Display Video (Frontend):**
    - **Browser (index.html):** Receives the new JPEG frame and updates the <img> tag's content.
    - This process repeats many times per second, creating a live video stream.
7. **User Control (Example):**
    - **Browser (index.html):** User clicks the "Pause" button.
    - **JavaScript:** Sends a POST request to /api/toggle-detection.
    - **Flask (app.py):** The API endpoint receives the request, sets the detection_active

variable to false, and sends back a {'status': 'paused'} JSON response. (In the next loop iteration, step 5.b-5.f will be skipped).