Face Mask Detection Project — Clean Report

> **Project Summary:** A real-time face mask detection system that runs either as a Flask web app with an MJPEG video stream or as a standalone OpenCV desktop script. Faces are detected with an OpenCV Haar cascade; a Keras/TensorFlow model classifies each face as **Mask** or **No Mask**.

---

1) Executive Summary

This project implements a lightweight, camera-based mask detector. The system captures frames from a webcam, detects faces, pre-processes each face to a 100×100 RGB tensor scaled to [0,1], runs a trained Keras model to estimate the probability of "Mask," and overlays color-coded labels on the live video. The Flask app exposes a browser-viewable stream; a health endpoint reports model/cascade load status. A second entry point (`detect_mask.py`) provides a desktop window for quick, local testing.

Key Features - Real-time face detection (OpenCV Haar cascade) - Binary mask classification (Keras model, 0.5 threshold) - Two runtimes: web (Flask streaming) and local (OpenCV window) - Health check endpoint for debugging - Clear overlays: **green** = Mask, **red** = No Mask, **yellow** = model missing (web app fallback)

---

2) Project Structure

```
project_root/
├── app.py                          # Flask web app (video stream + health)
├── detect_mask.py                  # Standalone OpenCV tester
├── haarcascade_frontalface_default.xml  # Face detector (OpenCV cascade)
├── mask_detector.keras             # Trained Keras/TensorFlow model (binary)
└── templates/
    └── index.html                  # Required by Flask index route (add this)
```

> Note: `templates/index.html` is expected by the web app; include a simple page embedding the `<img src="/video_feed">` stream.

---

3) How It Works (System Design)

Data Flow 1. **Frame Capture:** Read frames from default webcam (device 0). The web app resizes frames to 640×480 for throughput. 2. **Face Detection:** Convert to grayscale; detect faces via Haar cascade (`detectMultiScale(scaleFactor=1.1, minNeighbors=5)`). 3. **Preprocessing:** Crop each face, resize to 100×100, cast/scale to float in [0,1]. Add batch dimension. 4. **Inference:** Keras model outputs a probability p for "Mask." 5. **Decision/Overlay:** If $p > 0.5 \rightarrow$ label **Mask** (green box); else **No Mask** (red box). If

the model isn't available (web app fallback), faces are boxed in **yellow** with "Model Missing." 6. **Display/Serve:** - **Web app:** Encodes frames as JPEG and streams via multipart MJPEG to the browser at `/video_feed`. - **Standalone:** Renders annotated frames in a native window; press **Esc** to quit.

Simple Sequence (Web App)

```
Camera → OpenCV capture → Resize (640×480) → Gray → Haar faces
  ↳ For each face: crop → resize(100×100) → scale → model.predict
       → label + color → draw rectangle/text → encode JPEG → stream (MJPEG)
Browser ←————————————/video_feed (multipart/x-mixed-replace; boundary=frame)
```

---

4) Core Components

4.1 Face Detection (Haar Cascade)

- Pretrained cascade detects frontal faces quickly on CPU.
- Tunable via `scaleFactor`, `minNeighbors` for performance vs. accuracy.

4.2 Mask Classifier (Keras/TensorFlow)

- Input: 100×100 (RGB) float32, normalized to [0,1].
- Output: scalar probability for "Mask."
- Threshold: 0.5 (can be adjusted to tune sensitivity/specificity).

4.3 Overlay & Alerts

- **Web app:** Green (Mask), Red (No Mask), Yellow (model missing) rectangles and labels.
- **Standalone:** Same overlay logic and additionally prints `ALERT: No Mask Detected!` to console when a violation is found.

---

5) Web Application (Flask)

Endpoints - `GET /` → renders `templates/index.html` (embed the video feed). - `GET /video_feed` → returns a multipart MJPEG stream (content type: `multipart/x-mixed-replace; boundary=frame`). Place this URL as an `<img src>` for live video. - `GET /health` → JSON with flags: `model_exists`, `model_loaded`, `haarcascade_exists`, `haarcascade_loaded`.

Runtime Behavior - On startup, attempts to load `mask_detector.keras`; if missing or fails, continues and draws yellow "Model Missing" on detected faces. - Haar cascade is loaded from `haarcascade_frontalface_default.xml`; logs if empty. - Uses a background generator to yield encoded JPEG frames to the stream.

Minimal `templates/index.html`

```
<!doctype html>
<html>
  <head><meta charset="utf-8"><title>Mask Detector</title></head>
  <body style="margin:0;display:grid;place-items:center;height:
100vh;background:#111;">
    <img src="/video_feed" alt="Video stream" style="max-width:
100%;height:auto;border-radius:12px;"/>
  </body>
</html>
```

---

6) Standalone Desktop Script

Entry Point: `python detect_mask.py`

Behavior - Opens webcam; raises a clear error if camera cannot be opened. - Resolves model path gracefully: prefers `mask_detector.keras` in project root; falls back to legacy `models/mask_detector.model` if present. - Robustly extracts scalar probability from `model.predict` output shape. - Shows a window titled **Face Mask Detector**; press **Esc** to exit. - Prints `ALERT: No Mask Detected!` to the console whenever a non-mask face is found.

---

7) Installation & Setup

7.1 Requirements (Python 3.9+ recommended)

- Flask
- opencv-python (cv2)
- tensorflow / keras
- numpy

Optional: `imutils` or `gunicorn` (not strictly required by current code). GPU support is not required.

7.2 Setup Steps

```
# 1) Create and activate a virtual environment (recommended)
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate

# 2) Install dependencies (pin versions as needed)
pip install flask opencv-python tensorflow numpy
```

```
# 3) Ensure these files are present in the project root:
#    - mask_detector.keras
#    - haarcascade_frontalface_default.xml
#    - templates/index.html

# 4a) Run the web app
python app.py
# Open http://localhost:5000 in your browser

# 4b) Or run the standalone tester
python detect_mask.py
```

---

8) Testing & Validation

- **Smoke test:** Start the Flask app and visit `/health`; verify that both the cascade and (optionally) the model are loaded as expected.
- **Functional test:** With the webcam on, present faces with and without masks; verify correct labels and color overlays.
- **Negative test:** Temporarily rename `mask_detector.keras` and confirm yellow "Model Missing" boxes appear.
- **Performance check:** Measure FPS and CPU usage while varying frame size and `detectMultiScale` parameters.

---

9) Security & Privacy Considerations

- **No persistence:** Frames are processed in memory; nothing is saved by default.
- **Local access:** The web app binds to `0.0.0.0:5000` (accessible on the local network). If exposing beyond localhost, consider reverse proxy + TLS and add authentication.
- **Model/false results:** As with any ML system, expect false positives/negatives. Do not use as a sole enforcement mechanism.

---

10) Performance Notes

- **Throughput:** Resizing frames to 640×480 keeps CPU load moderate. Further downscaling increases FPS at the cost of detection fidelity.
- **Detection params:** Tweaking `scaleFactor` and `minNeighbors` trades recall vs. precision and speed.
- **Batching:** Current design runs per-face inference sequentially; for many faces per frame, consider vectorized/batched inference.
- **Accelerators:** For heavier models, consider GPU-enabled TensorFlow or ONNX Runtime.

---

11) Limitations

- **Haar cascade sensitivity:** Works best for frontal faces with decent lighting; profile/occluded faces may be missed.
- **Model transparency:** The provided `.keras` file is a black-box in this repo (training data/metrics not included); calibrate the 0.5 threshold using a validation set pertinent to your deployment.
- **Single camera:** Code assumes one default camera at index 0. Multi-camera support would require parameterization.

---

12) Recommendations & Future Work

- Replace Haar with a modern face detector (e.g., DNN-based) for robustness.
- Log predictions and anonymized metrics to evaluate real-world performance.
- Add Dockerfile and `requirements.txt` with pinned versions.
- Provide a simple HTML UI with start/stop buttons and health status indicator.
- Add unit tests for preprocessing and an integration test that mocks `model.predict`.
- Package the model alongside a model card (training data, metrics, intended use).

---

13) Appendix

A. Dependencies (unpinned)

```
flask
opencv-python
tensorflow
keras
numpy
```

B. Environment/Hardware

- OS: Windows/macOS/Linux
- CPU-only is sufficient; 4+ GB RAM recommended for TensorFlow.
- USB or integrated webcam.

C. License Notes

- The Haar cascade XML includes the Intel Open Source Computer Vision Library license terms; ensure redistribution complies with the noted conditions.

---

14) Quick Reference (Cheat Sheet)

- **Run web app:** `python app.py` → open browser at `http://localhost:5000` → stream at `/video_feed`, health at `/health`.
- **Run desktop tester:** `python detect_mask.py` → window titled *Face Mask Detector* → press **Esc** to quit.
- **Model input:** 100×100, RGB, float32, normalized to [0,1].
- **Decision rule:** probability > 0.5 ⇒ **Mask**, else **No Mask**.
- **Overlay colors:** green (Mask), red (No Mask), yellow (model missing).