**app.py**

```python
from flask import Flask, request, jsonify, render_template, Response, send_file
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np
import cv2
import os

app = Flask(__name__)

# Load model and Haarcascade
model = load_model('model.h5')
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Emotion Labels
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# ========= Home Page =========
@app.route('/')
def home():
    return render_template('index.html')

# ========= Image Upload and Detection =========
@app.route('/upload-image', methods=['POST'])
def upload_image():
    try:
        file = request.files['file']
        if not file:
            return jsonify({'error': 'No file uploaded.'})

        # Read uploaded image
        img_bytes = file.read()
        nparr = np.frombuffer(img_bytes, np.uint8)
        frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_classifier.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

        for (x, y, w, h) in faces:
            roi_gray = gray[y:y+h, x:x+w]
            roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)
            roi = roi_gray.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)

            prediction = model.predict(roi)[0]
            label = emotion_labels[prediction.argmax()]
            label_position = (x, y - 10)
```

```python

            # Draw rectangle and label
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,255), 2)
            cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)

        # Save the output image temporarily
        output_image_path = 'static/output.jpg'
        cv2.imwrite(output_image_path, frame)

        # Return the processed image
        return send_file(output_image_path, mimetype='image/jpeg')

    except Exception as e:
        return jsonify({'error': str(e)})

# ========= Video Upload and Detection =========

# ========= Video Upload and Detection (Optimized) =========
@app.route('/upload-video', methods=['POST'])
def upload_video():
    try:
        file = request.files['file']
        if not file:
            return jsonify({'error': 'No video uploaded.'})

        temp_video_path = 'temp_video.mp4'
        file.save(temp_video_path)

        cap = cv2.VideoCapture(temp_video_path)

        # Get original video details
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = cap.get(cv2.CAP_PROP_FPS)

        out = cv2.VideoWriter('static/output_video.mp4', fourcc, fps, (width, height))

        frame_skip_rate = 5  # ✅ Process 1 frame out of every 5 frames
        frame_count = 0

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            frame_count += 1
            if frame_count % frame_skip_rate != 0:
```

```
                out.write(frame)
                continue

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_classifier.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

            for (x, y, w, h) in faces:
                roi_gray = gray[y:y+h, x:x+w]
                roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)
                roi = roi_gray.astype('float') / 255.0
                roi = img_to_array(roi)
                roi = np.expand_dims(roi, axis=0)

                prediction = model.predict(roi)[0]
                label = emotion_labels[prediction.argmax()]
                label_position = (x, y - 10)

                cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,255), 2)
                cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)

            out.write(frame)

        cap.release()
        out.release()
        os.remove(temp_video_path)

        # After video processed, show result page
        return render_template('video_result.html')

    except Exception as e:
        return jsonify({'error': str(e)})

# ========= Real-Time Webcam Streaming =========
camera = cv2.VideoCapture(0)

def generate_frames():
    while True:
        success, frame = camera.read()
        if not success:
            break
        else:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_classifier.detectMultiScale(gray, 1.1, 5)

            for (x, y, w, h) in faces:
                roi_gray = gray[y:y+h, x:x+w]
                roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)
```

```
                roi = roi_gray.astype('float') / 255.0
                roi = img_to_array(roi)
                roi = np.expand_dims(roi, axis=0)

                prediction = model.predict(roi)[0]
                label = emotion_labels[prediction.argmax()]
                label_position = (x, y-10)

                cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,255), 2)
                cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()

            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/realtime')
def realtime():
    return render_template('realtime.html')

# ========= Run Flask App =========
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)

```