A black and white logo

AI-generated content may be incorrect.



**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY – CHENNAI**

**AD23402COMPUTER VISION**

A PROJECT REPORT

# JANUNDICE DETECTION

**SUBMITTED TO**

Mrs. M. Hemalatha

Assistant Professor (Sr. Gr.)

Submitted By:

1. Madhu Prasad (2023510009)
2. S. M. Navarasan (2023510017)

M. Akash (2023510046)

M. Jagadeesh (2023510052)

*TABLE OF CONTENTS*

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENT** | **PG.NO** |
| 1 | AIM | 3 |
| 2 | PROBLEM STATEMENT | 3 |
| 3 | PROJECT PROPOSAL | 3 |
| 4 | EXPLANATION | 5 |
| 5 | RESULT | 17 |
| 6 | SAMPLE OUTPUT | 17 |

1. **AIM:**

To develop an intelligent jaundice detection system that combines machine learning-based medical feature analysis and computer vision-based eye image processing to assist in early identification and risk assessment of jaundice in individuals.

1. **PROBLE STATEMENT:**

Jaundice is a common yet potentially serious condition that often goes undiagnosed in its early stages due to lack of awareness or limited access to diagnostic tools. Traditional diagnosis relies heavily on lab reports or visible symptoms like yellowing of the eyes, which may not be accurately interpreted by the untrained eye. There is a critical need for a reliable, accessible, and non-invasive screening tool that can assist in the early detection of jaundice.

This project aims to solve this problem by building a web-based application that integrates:

* **Machine learning models** trained on patient medical records to predict jaundice risk based on clinical features.
* **Computer vision algorithms** to detect scleral discoloration from uploaded eye images.
* A combined decision-making system to provide users with a clear, personalized risk assessment.

By merging both data-driven and image-based approaches, the system offers a comprehensive and user-friendly solution to support preliminary jaundice screening and raise timely health alerts.

1. **PROJECT PROPOSAL:**

**Jaundice Risk Detection System using Medical & Visual Cues**

**Objective:**  
 To develop a web-based application that detects the risk of jaundice by analyzing both *medical parameters* and *eye images* using machine learning and image processing techniques.

**Modules & Key Components:**

1. **Frontend Interface (HTML with Flask templates):**
   * Users input their medical parameters (e.g., bilirubin levels, ALT, AST, etc.).
   * An image upload option is provided for uploading an eye photograph.
   * Output is rendered on a results page with a risk classification and health advice.
2. **Flask Backend Logic:**
   * **Image Upload & Processing**:
     + Accepts and saves uploaded eye images securely.
     + Converts the image to a NumPy array after decoding from base64.
     + Passes the image to analyze\_eye\_image() for sclera color-based analysis.
   * **Medical Feature Extraction**:
     + Parses user-submitted medical data into a structured numerical feature list.
     + Uses these features to predict jaundice risk via a trained model using predict\_jaundice\_ml().
3. **Risk Assessment Logic:**
   * Combines machine learning prediction with eye image analysis for a more reliable result.
   * Provides color-coded and descriptive messages based on combined findings (e.g., high bilirubin but no eye yellowing).
4. **API Support:**
   * A /predict endpoint allows external applications (like mobile apps) to access the system via JSON.
   * Accepts base64-encoded images and health data.
   * Returns a structured JSON response with risk level and recommendations.

**Core Technologies Used:**

* **Python (Flask)** – For server-side logic and API.
* **OpenCV & PIL** – For image decoding and preprocessing.
* **Machine Learning Model** – For predicting jaundice from clinical data.
* **HTML/CSS** – For user interface rendering.
* **Base64 Encoding** – For in-memory image transfer.

**Proposed Enhancements (Future Work):**

* Improve eye analysis using a trained deep learning segmentation model for better accuracy.
* Add user authentication and history tracking.
* Support additional symptoms and lab parameters for improved predictions.
* Deploy the system on cloud platforms like AWS or Heroku for wider access.

1. **EXPLANATION:**
   1. **Flask Web Application (Main Interface):**

**Purpose:**

This is the core **web server** that handles user interaction. It lets users input medical data and upload an eye image, processes both, and displays the jaundice prediction results.

**Code:**

from flask import Flask, render\_template, request, redirect, url\_for, jsonify

import os

import base64

import cv2

import numpy as np

from PIL import Image

import io

from werkzeug.utils import secure\_filename

from image\_analyzer import analyze\_eye\_image

from ml\_predictor import predict\_jaundice\_ml

app = Flask(\_\_name\_\_)

# Folder for uploaded images

UPLOAD\_FOLDER = 'static/uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

@app.route('/')

def home():

    return render\_template('index.html')

@app.route('/submit', methods=['POST'])

def handle\_submission():

    form\_data = request.form.to\_dict()

    eye\_image = request.files.get('eye\_image')

    if eye\_image:

        filename = secure\_filename(eye\_image.filename)

        image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        eye\_image.save(image\_path)

        form\_data['image\_filename'] = filename

        # Read image and convert to base64

        with open(image\_path, "rb") as f:

            image\_data = f.read()

            encoded\_image = base64.b64encode(image\_data).decode('utf-8')

            eye\_image\_b64 = f"data:image/jpeg;base64,{encoded\_image}"

        # Eye image analysis

        try:

            header, b64data = eye\_image\_b64.split(",")

            image\_bytes = base64.b64decode(b64data)

            image = Image.open(io.BytesIO(image\_bytes)).convert("RGB")

            image\_np = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

            eye\_risk = analyze\_eye\_image(image\_np)

        except Exception as e:

            eye\_risk = False

    else:

        form\_data['image\_filename'] = ''

        eye\_risk = False

    # ML features

    try:

        features = [

            float(form\_data["age"]),

            1.0 if form\_data["gender"].lower() == "male" else 0.0,

            float(form\_data["total\_bilirubin"]),

            float(form\_data["direct\_bilirubin"]),

            float(form\_data["alt"]),

            float(form\_data["ast"]),

            float(form\_data["alk\_phos"]),

            float(form\_data["hemoglobin"]),

            1.0 if form\_data["fatigue"].lower() == "yes" else 0.0,

            1.0 if form\_data["stool\_color"].lower() == "dark" else 0.0,

            1.0 if form\_data["itching"].lower() == "yes" else 0.0,

            1.0 if form\_data["urine\_color"].lower() == "dark" else 0.0

        ]

    except Exception as e:

        return f"Error in form data: {e}", 400

    # ML Prediction

    model\_result = predict\_jaundice\_ml(features)

    form\_data["risk"] = "High" if model\_result == "Jaundice Detected" or eye\_risk else "Low"

    # --- Custom logic for result message ---

    model\_flag = "Jaundice" if model\_result == "Jaundice Detected" else "No"

    eye\_flag = "Jaundice" if eye\_risk else "No"

    if model\_flag == "Jaundice" and eye\_flag == "Jaundice":

        result\_message = 'High risk of jaundice'

    elif model\_flag == "No" and eye\_flag == "No":

        result\_message = 'No jaundice detected. You are normal.'

    elif model\_flag == "Jaundice" and eye\_flag == "No":

        result\_message = 'Your eyes are normal but your medical reports are not good. You may have some liver disorder.'

    elif model\_flag == "No" and eye\_flag == "Jaundice":

        result\_message = 'Your medical reports are clear, but your eyes appear yellow — could be due to other reasons.'

    else:

        result\_message = 'Inconclusive result. Please try again.'

    form\_data["model\_flag"] = model\_flag

    form\_data["eye\_flag"] = eye\_flag

    form\_data["result\_message"] = result\_message

    return render\_template("result.html", data=form\_data)

@app.route('/result')

def result\_page():

    return render\_template('result.html', data={})

@app.route('/predict', methods=['POST'])

def predict():

    data = request.json

    try:

        features = [

            float(data["age"]),

            1.0 if data["gender"].lower() == "male" else 0.0,

            float(data["total\_bilirubin"]),

            float(data["direct\_bilirubin"]),

            float(data["alt"]),

            float(data["ast"]),

            float(data["alk\_phos"]),

            float(data["hemoglobin"]),

            1.0 if data["fatigue"].lower() == "yes" else 0.0,

            1.0 if data["stool\_color"].lower() == "dark" else 0.0,

            1.0 if data["itching"].lower() == "yes" else 0.0,

            1.0 if data["urine\_color"].lower() == "dark" else 0.0

        ]

    except Exception as e:

        return jsonify({"error": f"Invalid input data: {e}"}), 400

    eye\_image\_b64 = data.get("eyeImageBase64")

    if eye\_image\_b64:

        try:

            header, b64data = eye\_image\_b64.split(",")

            image\_data = base64.b64decode(b64data)

            image = Image.open(io.BytesIO(image\_data)).convert("RGB")

            image\_np = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

            eye\_risk = analyze\_eye\_image(image\_np)

        except Exception as e:

            return jsonify({"error": f"Error processing image: {e}"}), 500

    else:

        eye\_risk = False

    model\_result = predict\_jaundice\_ml(features)

    risk = "High" if model\_result == "Jaundice Detected" or eye\_risk else "Low"

    return jsonify({

        "riskLevel": risk,

        "factors": ["Eye discoloration detected"] if eye\_risk else [],

        "recommendation": "Consult a physician for confirmation." if risk == "High" else "Maintain healthy liver habits."

    })

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Working:**

* **User Input**: A user submits a form containing clinical details (like age, bilirubin levels, etc.) and an optional eye image.
* **Image Handling**: The image is saved and encoded to base64 for processing.
* **Eye Analysis**: The image is passed to the analyze\_eye\_image() function (from the second code).
* **ML Prediction**: The clinical features are sent to predict\_jaundice\_ml() (from the third code).
* **Result Combination**: The system combines both model and image results to decide the final risk level and message.
* **Output**: It renders an HTML page showing the diagnosis result.
  1. **Computer Vision-Based Eye Analysis (image\_analyzer.py)**

**Purpose:**

To detect signs of jaundice (yellowing) in the eyes using image processing.

**Code:**

import cv2

import mediapipe as mp

import numpy as np

# Mediapipe setup

mp\_face\_mesh = mp.solutions.face\_mesh

LEFT\_EYE\_IDX = [33, 133]

RIGHT\_EYE\_IDX = [362, 263]

def crop\_eye(image, eye\_indices, landmarks):

    h, w = image.shape[:2]

    x\_coords = [int(landmarks[idx].x \* w) for idx in eye\_indices]

    y\_coords = [int(landmarks[idx].y \* h) for idx in eye\_indices]

    x\_min, x\_max = min(x\_coords), max(x\_coords)

    y\_min, y\_max = min(y\_coords), max(y\_coords)

    pad = 5

    x\_min, x\_max = max(0, x\_min - pad), min(w, x\_max + pad)

    y\_min, y\_max = max(0, y\_min - pad), min(h, y\_max + pad)

    return image[y\_min:y\_max, x\_min:x\_max]

def detect\_jaundice(image, threshold=30.0):

    image\_hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

    lower\_yellow = np.array([15, 40, 40])

    upper\_yellow = np.array([35, 255, 255])

    yellow\_mask = cv2.inRange(image\_hsv, lower\_yellow, upper\_yellow)

    yellow\_pixels = cv2.countNonZero(yellow\_mask)

    total\_pixels = image.shape[0] \* image.shape[1]

    yellow\_percentage = (yellow\_pixels / total\_pixels) \* 100

    return yellow\_percentage > threshold

def analyze\_eye\_image(image):

    with mp\_face\_mesh.FaceMesh(static\_image\_mode=True) as face\_mesh:

        rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

        results = face\_mesh.process(rgb\_image)

        if results.multi\_face\_landmarks:

            landmarks = results.multi\_face\_landmarks[0].landmark

            left\_eye = crop\_eye(image, LEFT\_EYE\_IDX, landmarks)

            right\_eye = crop\_eye(image, RIGHT\_EYE\_IDX, landmarks)

            jaundice\_left = detect\_jaundice(left\_eye)

            jaundice\_right = detect\_jaundice(right\_eye)

            return jaundice\_left or jaundice\_right  # Risk if either eye shows yellow

        else:

            return detect\_jaundice(image)  # fallback

**Working:**

* **Face & Eye Detection**: Uses **MediaPipe Face Mesh** to detect facial landmarks and extract the eye regions.
* **Eye Cropping**: Uses specific eye landmarks to crop out the left and right eyes from the image.
* **Yellow Pixel Detection**:
  + Converts the cropped eye region to HSV color space.
  + Applies a color mask to detect yellow pixels.
  + Calculates the percentage of yellow pixels.
* **Risk Decision**: If the percentage of yellow pixels in either eye is above a threshold (e.g., 30%), it returns True (risk present).
  1. **Machine Learning Model Predictor (ml\_predictor.py)**

**Purpose:**

To predict whether the person has jaundice based on clinical input using trained ML models.

**Code:**

import pickle

import numpy as np

with open("models/scaler.pkl", "rb") as f:

    scaler = pickle.load(f)

model\_names = ["Random\_Forest", "SVM", "Gradient\_Boosting", "Logistic\_Regression"]

models = {name: pickle.load(open(f"models/{name}.pkl", "rb")) for name in model\_names}

with open("models/validation\_scores.pkl", "rb") as f:

    validation\_scores = pickle.load(f)

def predict\_jaundice\_ml(user\_input):

    user\_input\_scaled = scaler.transform([user\_input])

    tb, alt, itching, urine, stool = user\_input[2], user\_input[4], user\_input[10], user\_input[11], user\_input[9]

    if tb > 3.0 and alt > 120:

        selected\_model = "Random\_Forest"

    elif tb > 3.0:

        selected\_model = "SVM"

    elif itching == 1 and urine == 1:

        selected\_model = "Gradient\_Boosting"

    elif urine == 1 and stool == 1:

        selected\_model = "Logistic\_Regression"

    else:

        selected\_model = max(validation\_scores, key=validation\_scores.get)

    model = models[selected\_model]

    prediction = model.predict(user\_input\_scaled)[0]

    return "Jaundice Detected" if prediction == 1 else "No Jaundice"

**Working:**

* **Input Scaling**: Applies a pre-fitted scaler to normalize user features.
* **Model Selection**:
  + Uses logic based on key features (e.g., Total Bilirubin, ALT, itching, urine color) to dynamically select the most relevant model.
  + If none match, chooses the best-performing model based on validation accuracy.
* **Prediction**: The selected model predicts whether jaundice is present (1) or not (0).
* **Output**: Returns a textual result like "Jaundice Detected" or "No Jaundice".

1. **RESULT:**

A black line on a white background

AI-generated content may be incorrect.

1. **SAMPLE OUTPUT:**
   1. **FRONTEND ASKING DETAILS FROM THE USER for PREDICTION**

A screenshot of a medical form

AI-generated content may be incorrect.

* 1. **PREDICTED RESULT**

**In here there are four cases:**

**CASE 1 = Both the ML Model and Open cv Predicts Positive result**

**CASE 2 = ML Model predicts Positive but Open CV predicts as normal**

**CASE 3 = ML Model predicts Normal but Open CV predicts as Positive**

**CASE 4 =** **Both the ML Model and Open cv Predicts as Normal result**

**NOTE: Positive means person has jaundice, normal means the person healthy**

* + 1. **CASE 1 Both the ML Model and Open cv Predicts Positive result**

A screenshot of a medical report

AI-generated content may be incorrect.

* + 1. **CASE 2** **ML Model predicts Positive but Open CV predicts as normal** Close-up of a human eye

       AI-generated content may be incorrect.
    2. **CASE 3** **ML Model predicts Normal but Open CV predicts as Positive**

Close-up of a person's eye

AI-generated content may be incorrect.

* + 1. **CASE 4** **Both the ML Model and Open cv Predicts as Normal result**

A close-up of a brown eye

AI-generated content may be incorrect.