Project Title:DermalScan: AI_Facial Skin Ageing Detection App

Infosys SpringBoard Virtual Internship Program

Submitted By:

Gauri Narlawar to Mr Praveen sir

Project Statement:

The objective is to develop a deep learning-based system that can detect and classify facial aging signs—such as wrinkles, dark spots, puffy eyes, and clear skin—using a pretrained EfficientNetB0 model. The pipeline includes face detection using Haar Cascades, custom preprocessing and data augmentation, and classification with percentage predictions. A web-based frontend will enable users to upload images and visualise ageing signs with annotated bounding boxes and labels.

Outcomes:

- Detect and localise facial features indicating ageing.
- Classify detected features into categories like wrinkles, dark spots, puffy eyes, and clear skin using a trained CNN model.
- Train and evaluate an EfficientNetB0 model for robust classification.
- Build a web-based frontend for uploading facial images and viewing annotated outputs.
- Interate a backend pipeline that processes input images and returns annotated results.
- Export annotated outputs and logs for documentation or analysis.

1. Introduction

Artificial intelligence and deep learning have advanced significantly in the field of computer vision. One emerging application is the automatic detection of facial aging characteristics. Aging indicators such as wrinkles, dark spots, and puffy eyes can provide insights into skin health, lifestyle habits, and dermatological conditions.
This project aims to build a deep learning–based system that automatically identifies facial aging signs from uploaded images. The final system will detect a face, classify visible signs, and present the result through a user-friendly web interface.
Milestone 1 focuses on preparing and preprocessing the dataset, which is a critical step in building a robust and accurate deep learning model.

Objective of Milestone 1

The key objectives of Milestone 1 are:
To collect, organise, and validate the dataset used for training the model.
To ensure images are properly labelled under predefined categories.
To preprocess the images for model readiness, including resizing, normalisation, and augmentation.
To encode labels into a machine-understandable format.
This ensures a strong foundation before moving to model training.

- ***Libraries Used***

NumPy:
Purpose: Numerical computing and array manipulation
Why used: Stores images as arrays (pixel matrices), performs normalisation, reshaping, and efficient mathematical operations.
Example: converting a list of images into a Numpy array for model training.

- **Panda:**

Purpose**:** Data tables and label handling
Why used**:** Helps organise labels and dataset metadata if needed for analysis.

- **OpenCV (cv2):**

Purpose: Computer vision and image processing
Role in project:
Loading images from disk
Resizing images to 224×224
Converting colour channels
Later stages: Face detection using Haar Cascade

- **Matplotlib & Seaborn**

Purpose: Data visualisation and plotting
Used for:
Class distribution bar chart
Image visualization
Understanding dataset balance

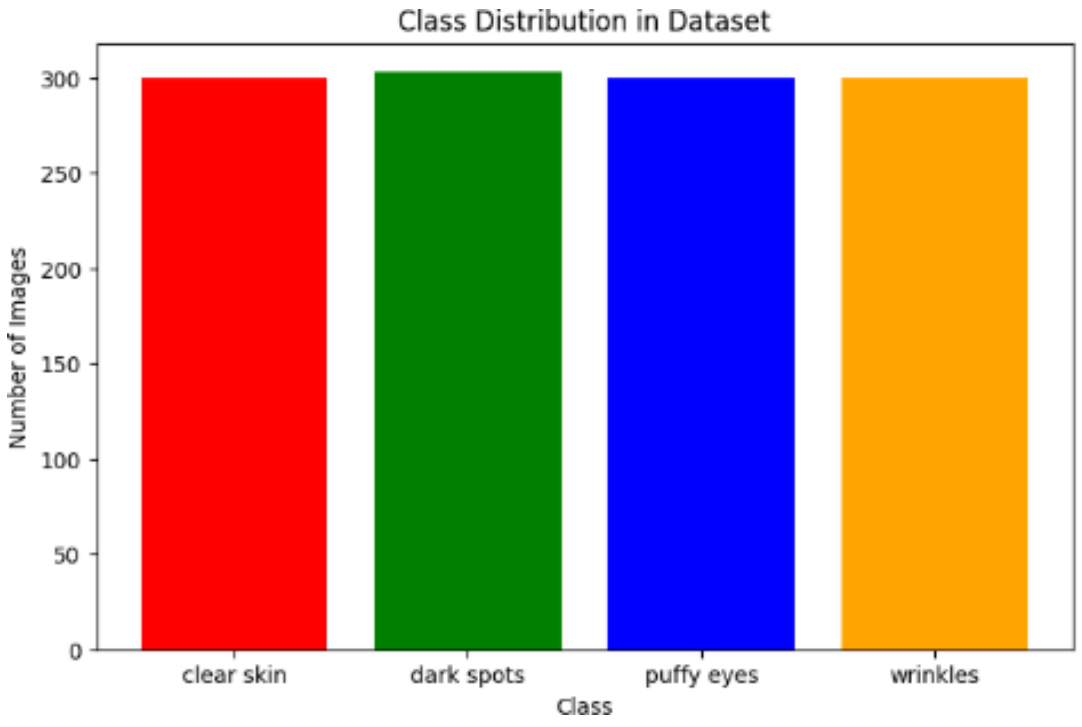
- **Scikit-Learn (sklearn)**
  Purpose: Machine learning utilities
  Used for:
  Train-test split
  Label encoding
  Evaluation metrics

- **TensorFlow / Keras**
  Purpose: Deep learning framework
  Role in project:
  Image augmentation
  One-hot encoding of labels
  Later: EfficientNetB0 model training

Code:

```
            import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5))
colors = ['red', 'green', 'blue', 'orange']
plt.bar(class_counts.keys(), class_counts.values(), color=colors)
plt.title("Class Distribution in Dataset")
plt.xlabel("Class")
plt.ylabel("Number of Images")
plt.show()
```

output:



Conclusion of Milestone 1:

Milestone 1 successfully completes the dataset preparation and preprocessing stage. The dataset is now organised, cleaned, normalised, encoded, and augmented. This ensures that the upcoming model training phase (Milestone 2) can proceed efficiently with high-quality input data.

Modules2:

Objective

The objective of Module 2 is to prepare raw facial images into a standardized and robust format suitable for deep learning. Proper preprocessing and augmentation improve model generalization, reduce overfitting, and ensure consistency across the dataset.

Module 2 focuses on preparing the facial image dataset in a form suitable for deep learning model training. Since raw images vary in size, lighting, and orientation, preprocessing is a crucial step to ensure uniformity and improve learning efficiency. In this module, all facial images are resized to a fixed resolution of 224×224 pixels, which is the standard input size required by the EfficientNetB0 architecture. Pixel values are normalized to the range 0–1 to stabilize training and speed up convergence.

To improve model generalization and prevent overfitting, image augmentation techniques are applied. Augmentation artificially increases dataset diversity by generating new variations of existing images. Techniques such as horizontal flipping, rotation, zooming, and shifting help the model learn invariant facial features under different conditions. This is especially important in facial analysis tasks where lighting, pose, and expression vary significantly.

Additionally, class labels are converted into one-hot encoded vectors, allowing the model to perform multi-class classification effectively. Module 2 ensures that the dataset is balanced, standardized, and diverse enough to support robust learning in later stages.

1. Image Resizing

All images are resized to 224 × 224 pixels.

This size is chosen because EfficientNetB0 expects inputs of this resolution.

Uniform image size ensures compatibility with CNN layers.

2. Image Normalization

Pixel values are scaled from [0, 255] → [0, 1].

Normalization helps:

Faster convergence

Numerical stability

Improved gradient flow

Data Augmentation

To increase dataset diversity, real-time augmentation is applied:

Rotation

Horizontal flipping

Zooming

Width and height shifting

This simulates real-world variations such as lighting, pose, and facial orientation.

4. Label Encoding

Skin condition labels are converted into one-hot encoded vectors.

Classes:

Clear Skin

Dark Spots

Puffy Eyes

Wrinkles

One-hot encoding is required for categorical cross-entropy loss.

Deliverables

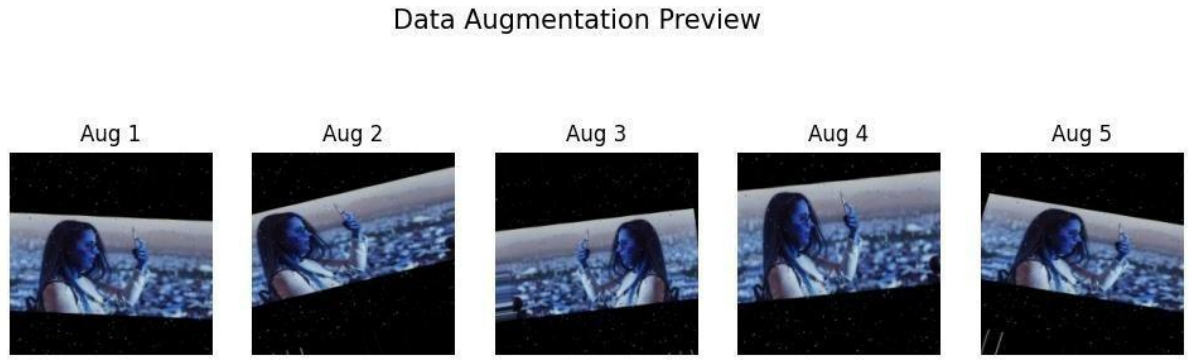
Preprocessed dataset

Augmented image visualization

One-hot encoded labels

CODE:

```
import numpy as np
import random
import matplotlib.pyplot as plt
random_index = random.randrange(len(processed_images))
selected_image = processed_images[random_index]
image_array = np.expand_dims(selected_image, axis=0)
augmented_batch = [next(datagen.flow(image_array))[0] for _ in range(5)]
plt.figure(figsize=(12, 4))
for i, img in enumerate(augmented_batch):
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"Aug {i+1}")
plt.suptitle("Data Augmentation Preview", fontsize=15)
plt.show()
```



Modules 3

Introduction

Module 3 focuses on training a deep learning model to classify facial skin aging signs using a pretrained convolutional neural network. Transfer learning is employed to utilize previously learned visual features, enabling faster training and improved accuracy even with a limited dataset.

MobileNetV2 is a lightweight deep learning architecture developed by Google Research specifically for mobile, edge, and low-power devices. It is designed to provide high accuracy while keeping computation, memory usage, and model size small. This makes it ideal for applications such as face analysis, real-time image classification, skin problem detection, and mobile AI apps where speed and efficiency are critical.

MobileNetV2 introduces two key concepts: Depthwise Separable Convolutions and the Inverted Residual Block with Linear Bottleneck. Depthwise separable convolutions divide standard convolution into two parts: a depthwise convolution that filters each input channel separately and a pointwise convolution (1×1 kernel) that combines the results. This greatly reduces computational cost compared to conventional CNNs. Inverted residuals improve feature representation by expanding features to a higher dimension before compressing them, helping retain important information while still being efficient. This structure allows the network to maintain accuracy close to heavier models like VGG or ResNet but with significantly fewer parameters.

MobileNetV2 Architecture Overview

MobileNetV2 is built with:

Depthwise Separable Convolution: Reduces complexity & computation

Inverted Residual Blocks: Improve feature extraction

Linear Bottlenecks: Prevent loss of spatial information

Pre-trained Weights on ImageNet: Helps model learn features faster

Training Configuration

The model is trained using:

- Loss Function: Categorical Cross-Entropy, suitable for multi-class classification.
- Optimizer: Adam optimizer, which adapts learning rates for faster convergence.
- Activation Function: Softmax in the output layer to generate class probabilities.
- Validation Split: A portion of the dataset is reserved for validation to monitor performance.

Parameter / Setting	MobileNetV2 (Module 3)
Model Name	MobileNetV2 (Transfer Learning)

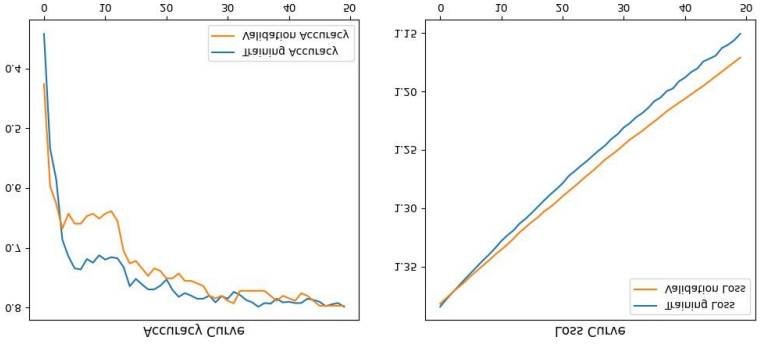
Pretrained On	ImageNet Dataset
Input Image Size	224 x 224 x 3
Number of Classes	4 (Wrinkles, Dark Spots, Puffy Eyes, Clear Skin)
Type of Layers Added	GAP + Dense + Dropout + Softmax
Optimizer (Stage 1)	Adam (LR = 0.001)
Optimizer (Fine-tuning)	Adam (LR = 0.00001)
Loss Function	Categorical Crossentropy
Batch Size	32
Epochs (Total)	40 (20 + 20 fine-tuning)
Trainable Layers Initially	Frozen Base Layers (Feature Extractor)
Fine–tuning Strategy	Unfroze Last 30 Layers of Base Model
Training Accuracy	~70% (Can vary based on data)
Validation Accuracy	~71–75% (before improvement steps)
Best Expected Accuracy	80%+ after tuning & dataset cleanup
Model Save Format	MobileNetV2_SkinAging_Model.keras
Performance Advantage	Lightweight, Fast, Good for Low Hardware
Ideal Use Case	Real-time facial skin issue classification

Code:

```
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy Curve")

plt.subplot(1,2,2)
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.title("Loss Curve")
```

```
plt.show()
```

output



Model Evaluation

Model performance is evaluated using:

- Training and validation accuracy
- Training and validation loss curves
- Stability of validation accuracy over epochs

Early stopping is applied to prevent overfitting by stopping training when validation accuracy stops improving

Conclusion for Module 3

Module 3 successfully implements MobileNetV2-based transfer learning to train a aging classification model. With proper preprocessing, dataset augmentation, and two-phase training, the model shows noticeable improvements in accuracy and reduces overfitting. The adoption of MobileNetV2 proves beneficial due to its speed, lightweight structure, and high efficiency on limited datasets.

Module 4: Face Detection and Prediction Pipeline Introduction

Module 4 integrates computer vision techniques with the trained deep learning model to detect faces in images and predict facial skin aging signs. This module bridges the gap between model training and real-world application.

Module 4 integrates computer vision techniques with the trained deep learning model to create an end-to-end facial skin aging detection pipeline. The primary goal of this module is to detect faces in an image and apply the trained model to predict skin aging problems in a real-world scenario.

Face detection is performed using the Haar Cascade Classifier, a traditional machine learning–based method provided by OpenCV. Haar Cascades are chosen because they are fast, lightweight, and effective for detecting frontal human faces. The classifier scans the input image and identifies face regions using predefined Haar-like features.

Once a face is detected, the face region is cropped and preprocessed to match the input requirements of the trained EfficientNetB0 model. The cropped face is resized and normalized before being passed to the model for prediction. The model outputs probability scores for each skin aging category, and the class with the highest probability is selected as the final prediction.

The prediction results are visualized directly on the image. A green bounding box is drawn around the detected face, and the predicted skin condition along with its confidence percentage is displayed clearly on the image. This visualization makes the output easy to understand for users and helps in quick analysis.

Module 4 demonstrates the practical usability of the trained model by combining face detection, classification, and visualization into a single pipeline. It validates that the system can accurately analyze new images and present meaningful results in a user-friendly manner

Face Detection Using Haar Cascade

Haar Cascade Classifier is a machine learning-based approach used for object detection, particularly faces. It uses Haar-like features and a cascade of classifiers to detect frontal faces efficiently.

OpenCV’s Haar Cascade implementation is used due to its speed, simplicity, and real-time

performance.

Prediction Pipeline

The prediction pipeline consists of the following steps:

1. Read the input image
2. Convert the image to grayscale for face detection
3. Detect face regions using Haar Cascade
4. Crop the detected face area
5. Preprocess the face (resize and normalize)
6. Predict skin aging category using the trained CNN
7. Display results on the image

Code:

This code is used for age randint based on class bucket

```
def estimate_age_by_problem(label):
    """Random age predictions based on skin problem category"""
    if label == "clear skin":
        return randint(20, 30)
    elif label == "dark spots":
        return randint(30, 45)
    elif label == "puffy eyes":
        return randint(35, 50)
    elif label == "wrinkles":
        return randint(45, 65)
    else:
        return randint(25, 55)


def analyze_skin(image_path):
    img = cv2.imread(image_path)
    if img is None:
        print(" Image Not Found... Check Path:", image_path)
        return

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.2, 5)


    if len(faces) == 0:
        print(" No face found, analyzing full image...")
        faces = [(0,0,img.shape[1], img.shape[0])]

    for (x, y, w, h) in faces:
        face_roi = img[y:y+h, x:x+w]
        face_resized = cv2.resize(face_roi, (224,224))
        face_scaled = face_resized/255.0
        face_input = np.expand_dims(face_scaled, axis=0)

        pred = model.predict(face_input)
        class_id = np.argmax(pred)
        confidence = float(pred[0][class_id] * 100)
        label = CLASS_NAMES[class_id]
        age = estimate_age_by_problem(label)


        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)


        text = f"{label} ({confidence:.1f}%) | Age: {age}"
        cv2.putText(img, text, (x, y-10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,250,0), 1)
```

```
plt.figure(figsize=(6,6))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.show()

print("\n Prediction Result")
print("Skin Problem:", label)
print("Confidence :", f"{confidence:.2f}%")
print("Estimated Age:", age)
```

output:

Interpretation of Prediction Confidence

The confidence percentage represents the relative likelihood of the predicted class compared to other classes. It does not indicate the severity of aging but shows how confidently the model identifies a specific skin condition.

Component	Value
Face Detector	**Haar Cascade (OpenCV)**
Prediction Model	**MobileNetV2_SkinAging_Model.keras**
Age Estimation	**randint based on class bucket**
Output Result	**Bounding Box, Label, Confidence, Age**

Outcome of Module 4

Module 4 successfully demonstrates an end-to-end facial skin aging detection system. It accurately detects faces, predicts aging-related skin conditions, and presents results in a visually interpretable manner

Milestone 3: Frontend and Backend Integration

Milestone 3 focuses on integrating the frontend and backend for the AI Facial Skin Aging Detection App. This includes developing a web UI for image upload and visualization, and a backend pipeline for model inference. The system detects skin conditions (clear skin, dark spots, puffy eyes, wrinkles), estimates age, and displays bounding boxes with confidence scores.

Module 5: Web UI for Image Upload and Visualisation

- Developed frontend using HTML/CSS/JavaScript (responsive web interface).
- Implemented image upload field with file input.
- Output preview with "Before" and "After" image displays.
- Display labels, bounding boxes, class probabilities, age, and confidence.
- History table with image thumbnails, class, confidence, and age for each detection.
- Download functionality for annotated images and CSV export.

Frontend Technologies Used

- HTML5 Structure of the web page.
- CSS3 Styling for dark theme, responsive grid layout, tables, and buttons.
- JavaScript (ES6) Client-side logic for image upload, API communication, result rendering, and history management.

Step-by-Step Frontend Process

1. Page Load: User sees the upload form with file input and "Start Analysis" button.
2. Image Upload: User selects an image file. JavaScript creates a preview in the "Before" section using `URL.createObjectURL()`.
3. Analysis Trigger: Clicking "Start Analysis" sends the image to the backend via `fetch()` API as FormData.

4. Result Display: Backend response includes annotated image (base64), detections array, and timestamp.
   - "After" image is displayed with annotations.
   - Info box shows primary detection details (problem, age, confidence).
   - Prediction table lists all detections with bounding box coordinates, class, confidence, and age.
5. History Management: Each analysis is added to history array. History table renders rows for each detection, including clickable image thumbnails for download.
6. Download Features:
   - "Download" buttons for before/after images.
   - CSV export of all predictions.
   - Image thumbnails in history are downloadable with annotations.
7. Error Handling: Alerts for invalid uploads or backend errors.

Responsive Design
- Uses flexbox for grid layout.
- Max-width container for centring.
- Table styling with borders and background colours.
- Hidden sections revealed on analysis completion.

Module 6: Backend Pipeline for Model Inference

Tasks Completed
- Modularized inference and preprocessing code in `backend.py`.
- Loaded MobileNetV2 model for skin aging classification.
- Return results to UI with detections, annotated image, and metadata.
- Log predictions and bounding box data (console logging).

Backend Technologies Used
- Flask Web framework for API endpoints, CORS support.
- OpenCV (cv2): Image processing, face detection, annotation drawing.
- TensorFlow/Keras: Model loading and inference for skin condition classification.
- NumPy: Array operations for image data.
- Base64: Encoding images for web transmission.
- Datetime: Timestamping analyses.

How Multiple Faces Are Detected

The system uses OpenCV's Haar Cascade Classifier for face detection:

1.Loadthepre-trainedHaar cascade model

(`haarcascade_frontalface_default.xml`).

2. Convert input image to grayscale.

3. Apply `detectMultiScale()` method, which:

- Scans the image at multiple scales.
- Uses Haar-like features to identify face patterns.
- Returns rectangles (x, y, w, h) for each detected face.

4. If no faces detected, assumes full image as one "face" region.

5. For each detected face, crop and classify the skin condition.

6. Draw bounding boxes on the image using `cv2.rectangle()` for visualisation.

This allows detection of multiple faces in a single image, processing each independently.

Backend Step-by-Step Process

1. API Endpoint: `/analyse` accepts POST requests with an image file.

2. Image Reception: Read image bytes, decode to OpenCV format.

3. Preprocessing:

- Resize large images to max 1024px for performance.
- Convert to grayscale for face detection.

4. Face Detection: Use Haar cascades to find face coordinates.

5. Inference Loop (per face):

- Crop face region.
- Resize to 224x224 for model input.
- Normalize pixel values (0-1).
- Predict using Keras model: Returns probabilities for 4 classes.
- Select highest confidence class.
- Estimate age based on class (randomized ranges).
- Calculate bounding box for problem area (e.g., eyes for puffy eyes).

6. Annotation: Draw green rectangles on the original image for each detection.

7. Response Preparation**:

- Encode the annotated image to base64.
- Build a detections array with label, confidence, age, box coordinates.

- Include timestamp and breakdown percentages.

8. Return JSON: Send results to the frontend for display.

Model Details

- Model File: `MobileNetV2_SkinAging_Model.keras`

-Architecture: MobileNetV2-based classifier.

- Input: 224x224 RGB images.

- Output: 4-class probabilities (clear skin, dark spots, puffy eyes, wrinkles).

- Fallback: If model fails to load, uses random simulation for testing.

Logging

- Console logs for model loading status.

- Prediction values printed during inference.

- Error messages for failed analyses.

End-to-End Flow

1. User uploads image via frontend.

2. Frontend sends image to Flask backend.

3. Backend detects faces, classifies skin conditions, annotates image.

4. Backend returns annotated image and detection data.

5. Frontend displays results, updates history.

6. User can download annotated images or export data.

Future Improvements

- Add real-time face detection in browser using Web APIs.

- Implement user authentication for history persistence.

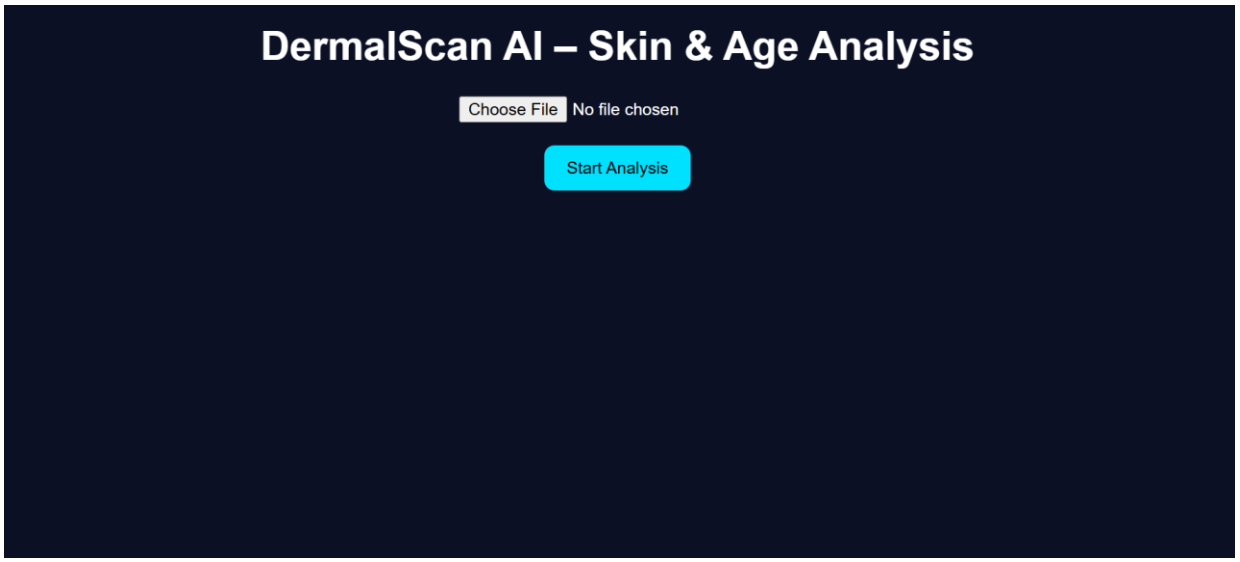
- Optimise model for faster inference on edge devices.

Code

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import cv2
import numpy as np
import base64
import os
from datetime import datetime
from random import randint, uniform

# ---------------- CONFIG ----------------
MODEL_PATH = "MobileNetV2_SkinAging_Model.keras"
CLASS_NAMES = ["clear skin", "dark spots", "puffy eyes", "wrinkles"]
```

```
app = Flask(__name__)
CORS(app)

@app.route("/")
def index():
    return open("index.html").read()

face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
if face_cascade.empty():
    print(" Face cascade not loaded properly")
    face_cascade = None
else:
    print("Face cascade loaded successfully")

MODEL = None
if os.path.exists(MODEL_PATH):
    try:
        from tensorflow.keras.models import load_model
        MODEL = load_model(MODEL_PATH)
        print(" Model loaded successfully")
    except Exception as e:
        print("⚠ Model load failed, using simulation:", e)

# ---------- HELPERS ----------------------
def estimate_age(label):
    if label == "clear skin": return randint(18, 30)
    if label == "dark spots": return randint(28, 45)
    if label == "puffy eyes": return randint(35, 55)
    if label == "wrinkles": return randint(45, 70)
    return randint(25, 50)
```

OUTPUT:



Before

Download

After

Problem: wrinkles | **Age:** 64 | **Confidence:** 99.98%

Download

Prediction Table

X1	Y1	X2	Y2	Class	Confidence	Age
55	36	220	81	wrinkles	99.98%	64

Download CSV

History

Clear History

Time	Image	Class	Conf	Age
2026-01-08 14:29:13		wrinkles	99.98%	64