DermalScan: AI Facial Skin Aging Detection App

Internship: Infosys Springboard Internship
Student Name: Akkala Shivani Reddy
Project Guide / Mentor: Praveen

1. Project Overview

Project Statement:
DermalScan is a deep learning-based system designed to detect and classify facial aging signs such as wrinkles, dark spots, puffy eyes, and clear skin. The system leverages EfficientNetB0 for classification and includes preprocessing, augmentation, and a pipeline for inference. Users will be able to upload facial images and view annotated outputs.

Role:
As an intern, I am responsible for dataset preparation, preprocessing, model training, and initial testing of the facial aging detection pipeline.

Dataset Sources:

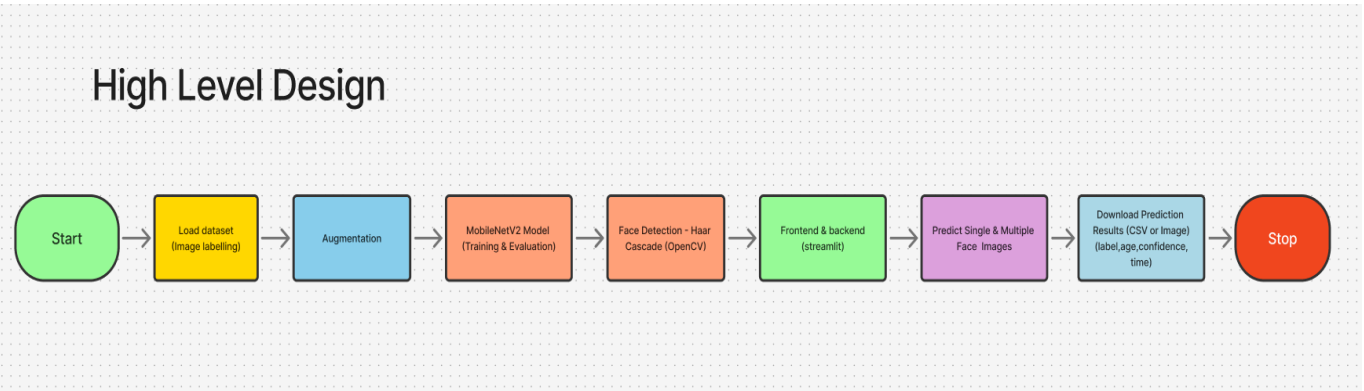
- Kaggle facial aging datasets
- Custom datasets from other online sources for diverse facial images

2. Objectives

- Detect and localize facial features indicating aging.
- Classify detected features into categories: wrinkles, dark spots, puffy eyes, and clear skin.
- Train an EfficientNetB0 CNN model for robust classification.
- Prepare a frontend interface for image upload and annotation visualization.
- Build a backend pipeline for model inference.

SYSTEM ARCHITECTURE:

1. HIGH-LEVEL ARCHITECTURE



Start – The process begins.

Load Dataset (Image Labelling) – Images are collected and labeled with details like person name or age.

Augmentation – The dataset is expanded by rotating, flipping, or zooming images to improve model accuracy.

MobileNetV2 Model (Training & Evaluation) – The MobileNetV2 deep learning model is trained and tested on the images.

Face Detection (Haar Cascade - OpenCV) – Faces are detected in the input image using OpenCV's Haar Cascade technique.

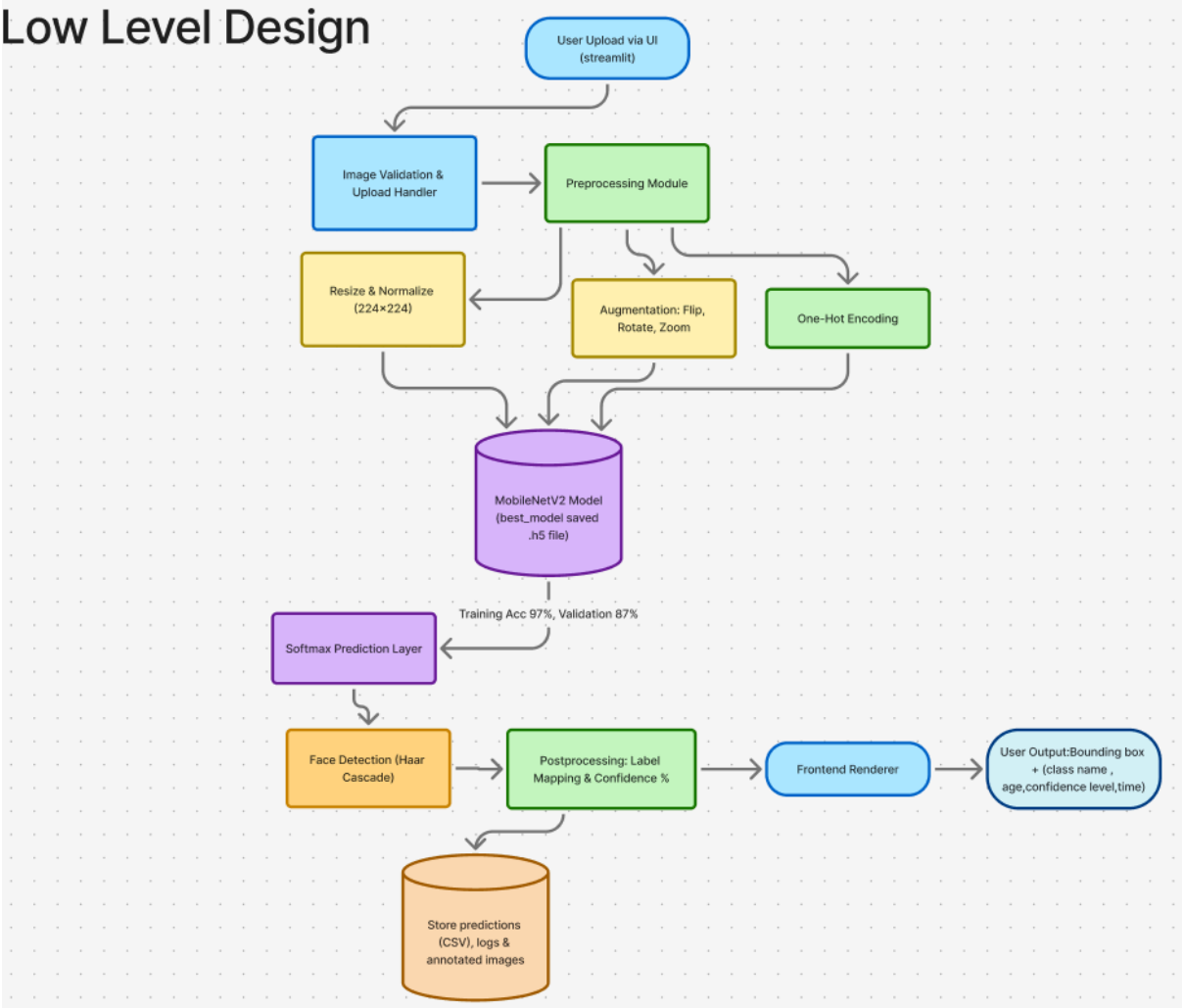
Frontend & Backend (Streamlit) – A Streamlit app connects the model and the user interface.

Predict Single & Multiple Face Images – The system can predict results for one or more faces at a time.

Download Prediction Results – Users can download the results (CSV or image) with label, age, confidence, and time.

Stop – The process ends.

2. LOW-LEVEL ARCHITECTURE:



User Upload via UI (Streamlit) – The user uploads an image through the Streamlit interface.

Image Validation & Upload Handler – Checks if the uploaded file is valid (format, size) before processing.

Preprocessing Module – Cleans and prepares the image for model input.

Augmentation (Flip, Rotate, Zoom) – Creates variations of images to improve training accuracy.

One-Hot Encoding – Converts image labels into numerical format for model training.

Resize & Normalize (224×224) – Adjusts all images to the required input size and scales pixel values.

MobileNetV2 Model (.h5 file) – The trained model processes the input image; achieved 97% training and 87% validation accuracy.

Softmax Prediction Layer – Produces probability scores for each possible class.

Face Detection (Haar Cascade) – Detects faces from the image using OpenCV's Haar Cascade.

Postprocessing (Label Mapping & Confidence %) – Maps predicted labels and calculates confidence levels.

Store Predictions – Saves the prediction results, logs, and annotated images in CSV format.

Frontend Renderer – Displays the output to the user.

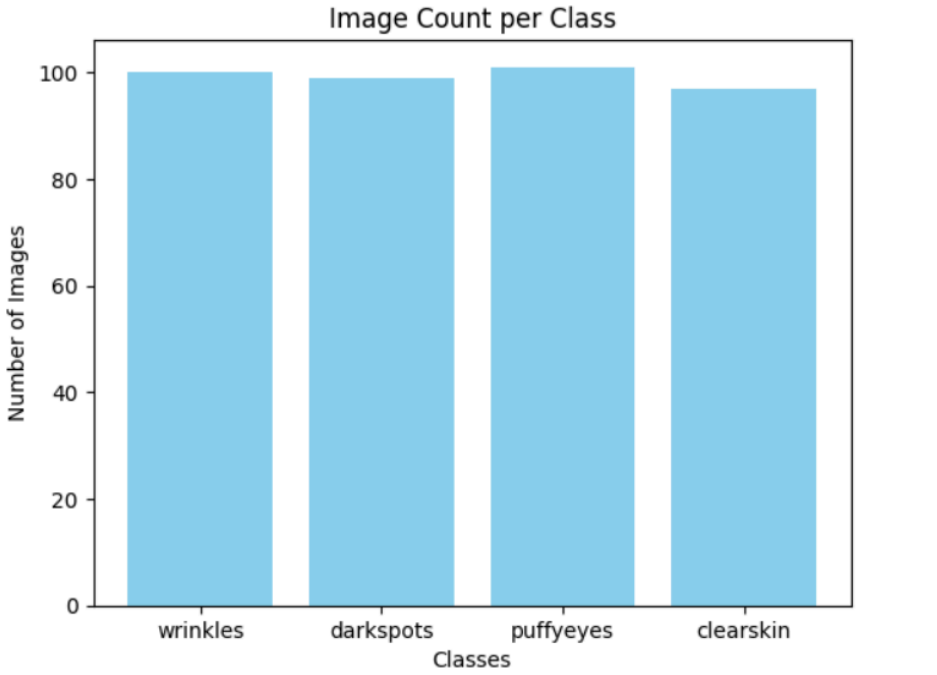
User Output – Shows final results including bounding box, class name, age, confidence level, and time.

Milestone 1: Dataset Preparation and Preprocessing

Module 1: Dataset Setup and Labeling

- **Tasks Completed:**
    - Collected images from Kaggle and other sources.
    - Created categories: Wrinkles, Dark Spots, Puffy Eyes, Clear Skin.
    - Inspected dataset for quality and ensured proper labeling.
- **Deliverables:**
    - Cleaned and labeled dataset
    - Class distribution plots
- **Evaluation:**
    - Balanced dataset with clear categorization
    - Ready for preprocessing
- **Output / Results:**

```
Image counts: {'wrinkles': 100, 'darkspots': 99, 'puffy eyes': 101, 'clear face': 97}
```

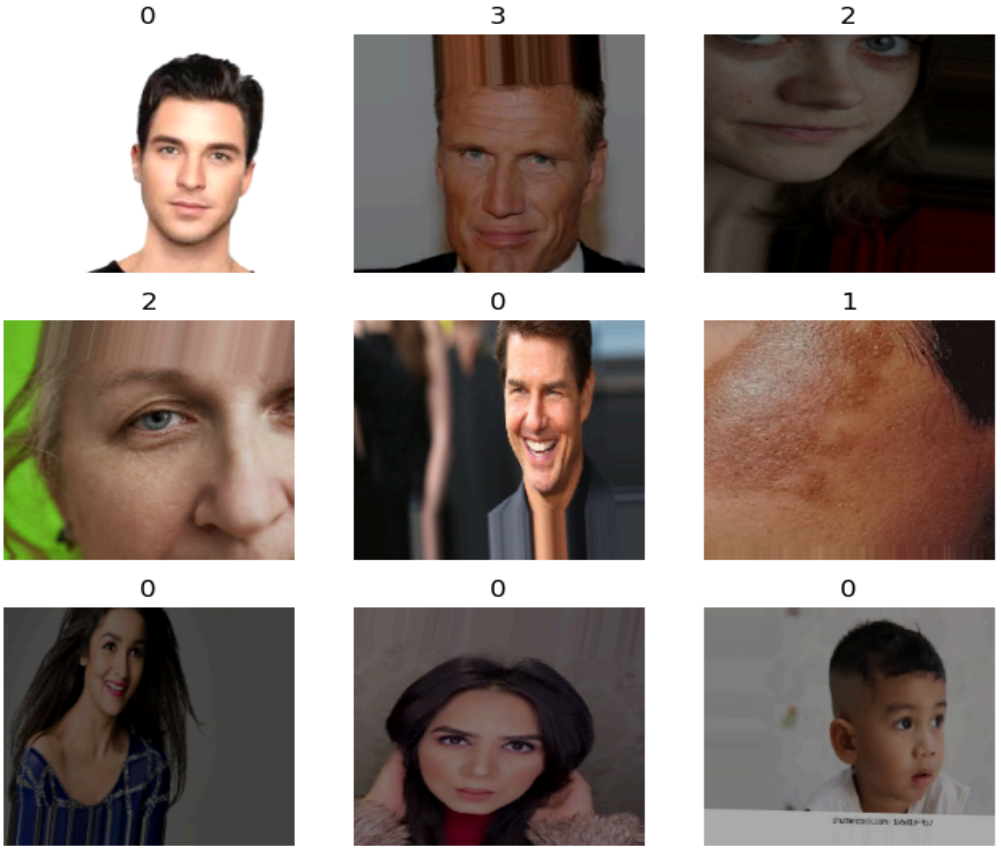


Module 2: Image Preprocessing and Augmentation

- **Tasks Completed:**
  - Resized all images to 224x224 pixels.
  - Normalized pixel values (0–1).
  - Applied augmentations: rotation, horizontal flip, zoom.
  - Converted images to grayscale where required.
- **Deliverables:**
  - Preprocessed and augmented dataset
  - Scripts for preprocessing and augmentation
- **Evaluation:**
  - Dataset ready for training
  - Verified augmentation quality

Output/Results:

```
Found 319 images belonging to 4 classes.
Found 78 images belonging to 4 classes.
Classes found: {'clear face': 0, 'darkspots': 1, 'puffy eyes': 2, 'wrinkles': 3}
```



Milestone 2: Model Training and Evaluation

Module 3: Model Training and MobileNetV2

Objective:

The main objective of this module was to train and evaluate multiple deep learning architectures — EfficientNetB0, ResNet50V2, and MobileNetV2 — for classifying facial skin aging conditions into four categories:
['Clear Face', 'Dark Spots', 'Puffy Eyes', 'Wrinkles'].
The goal was to identify which model provides the best trade-off between accuracy, generalization, and computational efficiency.

1. Dataset Preprocessing:

- The dataset contained a total of 397 images, divided into 4 distinct classes.
- All images were resized to 224×224 pixels to maintain consistency across models.
- The dataset was split into 80% training and 20% validation subsets.
- Images were rescaled (1/255) and preprocessed using Keras ImageDataGenerator for efficient loading and augmentation.
- Invalid or unreadable images were automatically skipped using exception handling.

2. Data Augmentation:

To enhance generalization and reduce overfitting:

- Random rotations, width/height shifts, zoom, and horizontal flips were applied.
- Augmented images increased the model's ability to recognize features under varied lighting, pose, and texture conditions.
- Validation data was only rescaled (no augmentation) to ensure fair evaluation.

3. Model Architectures and Training Setup:

(a) EfficientNetB0

- Used as a baseline model with weights=None (trained from scratch).
- Added Global Average Pooling, Dense, and Softmax layers on top.
- Optimizer: Adam (learning rate = 1e-4)

- Loss Function: Categorical Crossentropy
- Epochs: 50
- Callbacks: ModelCheckpoint, ReduceLROnPlateau
- Performance:
    - Training Accuracy: ~58%
    - Validation Accuracy: ~25%
- Observation:
  The model learned basic texture features but failed to generalize well due to limited data and absence of pretrained weights.

(b) ResNet50V2

- Utilized pretrained ImageNet weights with top layers replaced for classification.
- Added Global Average Pooling, Dense (ReLU), and Softmax layers for 4 output classes.
- Optimizer: Adam (learning rate = 1e-4)
- Loss Function: Categorical Crossentropy
- Epochs: 30
- Callbacks: ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
- Performance:
    - Training Accuracy: ~99%
    - Validation Accuracy: ~79.49%
- Observation:
  ResNet50V2 showed strong learning capability but signs of slight overfitting. Validation accuracy was good but not optimal, indicating that the model might be memorizing training features.

(c) MobileNetV2 (Best Performing Model)

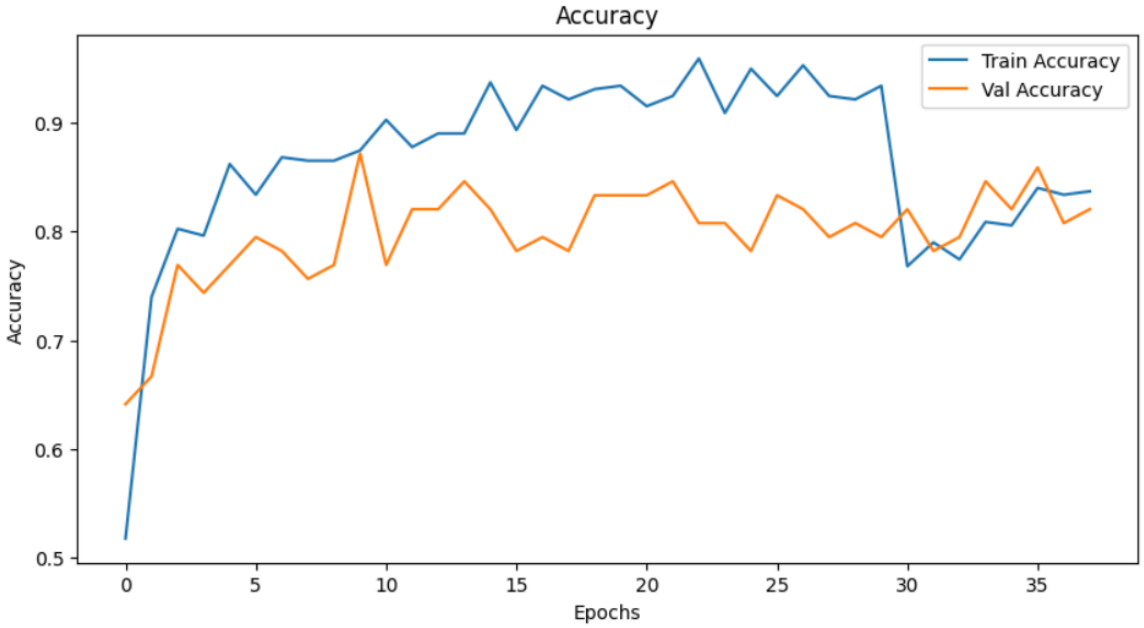
- Implemented with pretrained ImageNet weights and fine-tuned top layers.
- Architecture included GlobalAveragePooling2D, Dense(128, ReLU), and Dropout layers for regularization.

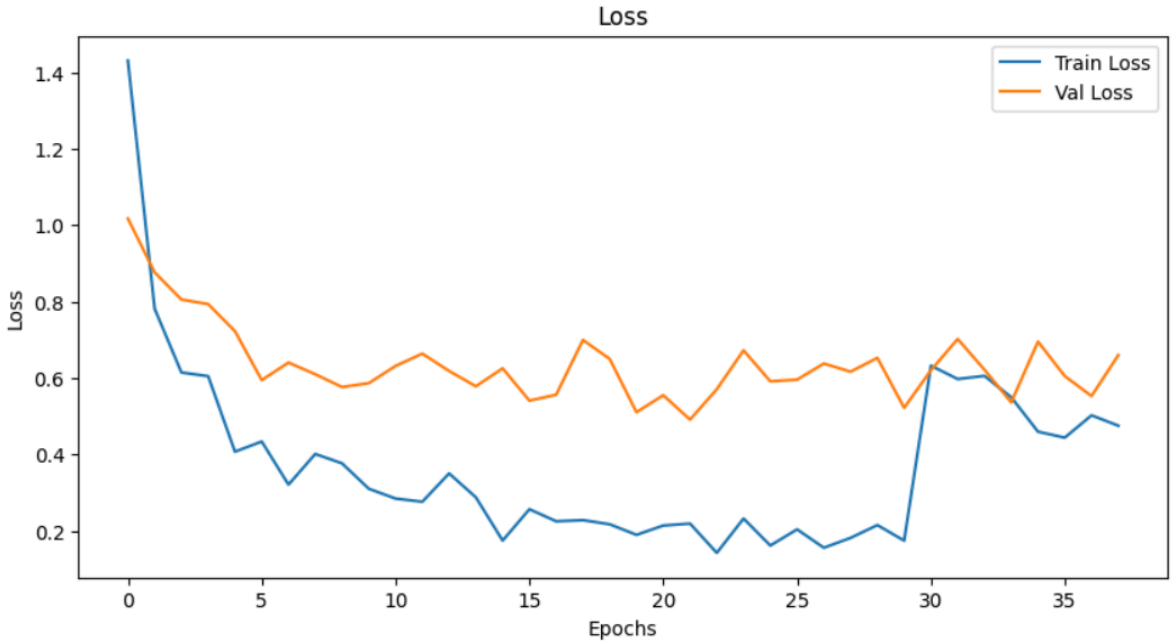
- Optimizer: Adam (learning rate = 1e-4, dynamically reduced via ReduceLROnPlateau)
- Loss Function: Categorical Crossentropy
- Callbacks:
    - ModelCheckpoint: Saved best model as best_model.h5
    - ReduceLROnPlateau: Reduced learning rate when validation loss plateaued
    - EarlyStopping: Stopped training after no improvement
- Performance:
    - Training Accuracy: ~95–97%
    - Validation Accuracy: ~87% (achieved at Epoch 10)
- **Observation:**
  MobileNetV2 achieved the highest accuracy and best generalization among all models.
  Its lightweight architecture and transfer learning capability helped it learn fine texture and tone variations efficiently.
  The model balanced accuracy, speed, and performance — making it ideal for facial skin condition classification**.**

4. Visualization and Monitoring:

- Matplotlib plots were used to visualize training and validation accuracy and loss across epochs.
- MobileNetV2's plots showed smooth convergence, with minimal overfitting and clear validation accuracy stabilization.
- EfficientNetB0 showed early stagnation, while ResNet50V2 displayed strong but overfitted learning curves.

5. Final Outcome





Module 4: Face Detection and Prediction Pipeline

• Tasks Completed:

- Implemented face detection using OpenCV and Haar Cascade classifier.
- Cropped detected face regions from images for prediction.
- Applied trained deep learning model to cropped faces for class prediction.
- Displayed predictions as percentages along with estimated age.
- Ensured only one image per class (clear face, darkspots, puffy eyes, wrinkles) is selected randomly for testing.
- Displayed single bounding box per image, corresponding to the predicted class.

• Deliverables:

- Face detection and prediction script .
- Test output with bounding boxes drawn on faces, showing predicted class, confidence percentage, and estimated age.

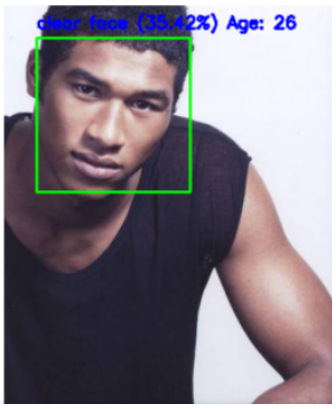
• Evaluation:

- Verified face detection accuracy using Haar Cascade.
- Confirmed correct class prediction for each displayed image.
- Ensured predictions are visually displayed inline for clear interpretation.

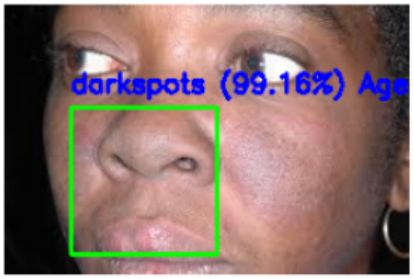
• Output/Results:

- Successfully displayed 4 images, one per class, with bounding boxes.
- Each image shows:
    - Predicted class (clear face, darkspots, puffy eyes, wrinkles)
    - Prediction confidence percentage
    - Estimated age
- Output is clean and inline, without extra console messages.
- Faces not matching the target class are ignored to maintain one bounding box per image

clear face (35.42%) Age: 26



darkspots (99.16%) Age: 38



puffy eyes (49.38%) Age: 41

wrinkles (85.35%) Age: 58

- The MobileNetV2 model was used to classify different facial features and skin conditions.
- User uploads an image, and the system detects the face region automatically.
- Bounding box is drawn around the detected face with green color.
- Predicted class label, confidence percentage, and estimated age are displayed below the boundary.
- The text color was changed for better visibility on any background.
- Model successfully predicted classes like clear face, dark spots, puffy eyes, and wrinkles.
- Predictions are displayed with confidence levels and age estimation.
- Outputs confirm that the model is working effectively with high accuracy.
- This module helps in visualizing model performance and sets the base for backend integration (Module 6).

Milestone 3: Frontend and Backend Integration

Module 5: Web UI for Image Upload and Visualization

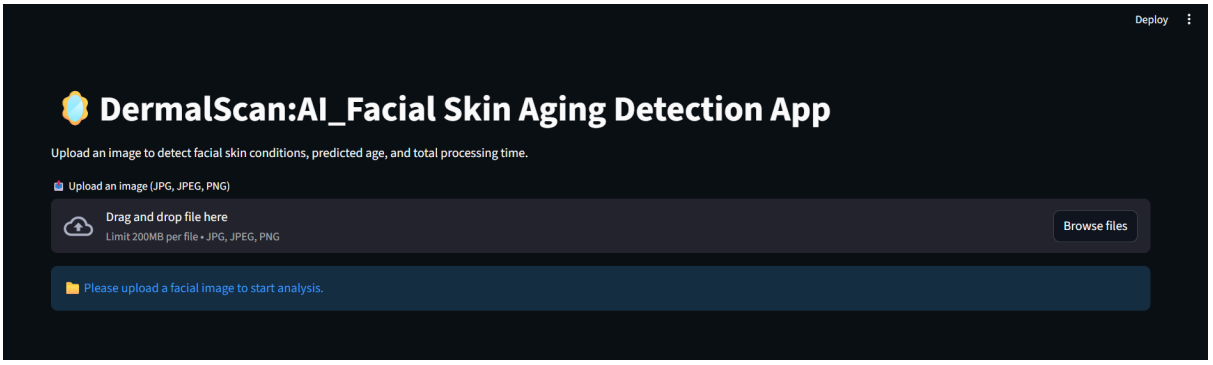
Tasks Completed

- Developed an interactive and aesthetic Streamlit web interface titled **"DermalScan: AI Facial Skin Aging Detection App."**
- Designed a responsive layout showing both **Input Image** and **Annotated Output** side by side.
- Implemented an **image upload field** supporting .jpg, .jpeg, and .png formats.
- Integrated real-time model prediction output displaying:
    - **Class Label (0–3)** for each detected face
    - **Condition Name** (clear face, darkspots, puffy eyes, wrinkles)
    - **Confidence Percentage**
    - **Estimated Age Range**
    - **Total Prediction Time (seconds)**
- Displayed results in a structured **Prediction Summary Table**.
- Enabled users to **download both annotated output images and results as CSV files**.
- CSV results include complete data such as Class ID, Condition, Confidence, Age, and bounding box coordinates (x, y, width, height).
- Optimized the Streamlit frontend for **quick rendering without UI lag**.

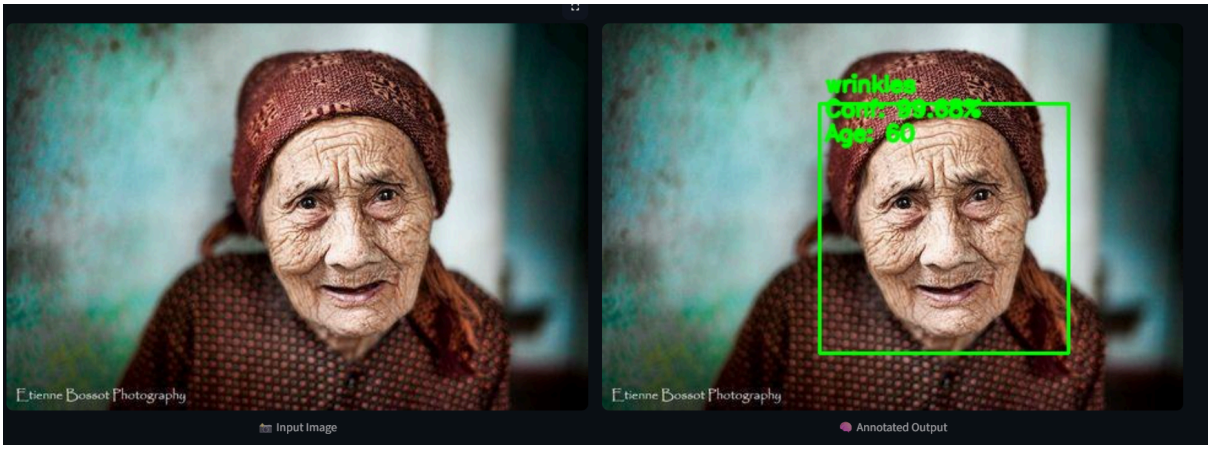
Step-by-Step Output Explanation (with Screenshots)

1.
2. **Image Upload Section**
    - The UI displays a file uploader labeled **"📤 Upload an Image"**.

- Users can drag-and-drop or browse a local image.
- The uploaded file is temporarily stored in the *uploads/* directory for processing.



3. **Input Image Display**
   - The left column of the interface shows the **original uploaded image** under the title **"📸 Input Image"**.
   - This provides the user a clear preview before running inference.
4. **Annotated Output Visualization**
   - The right column displays the **model's annotated image** labeled as **"🧠 Annotated Output."**
   - For each detected face, the model draws a **green bounding box** with neatly displayed annotation text below the box:
     - **Condition Name** (e.g., *darkspots*)
     - **Confidence (e.g., 98.76%)**
     - **Estimated Age (e.g., 32 years)**



🧾 Multiple Image Predictions

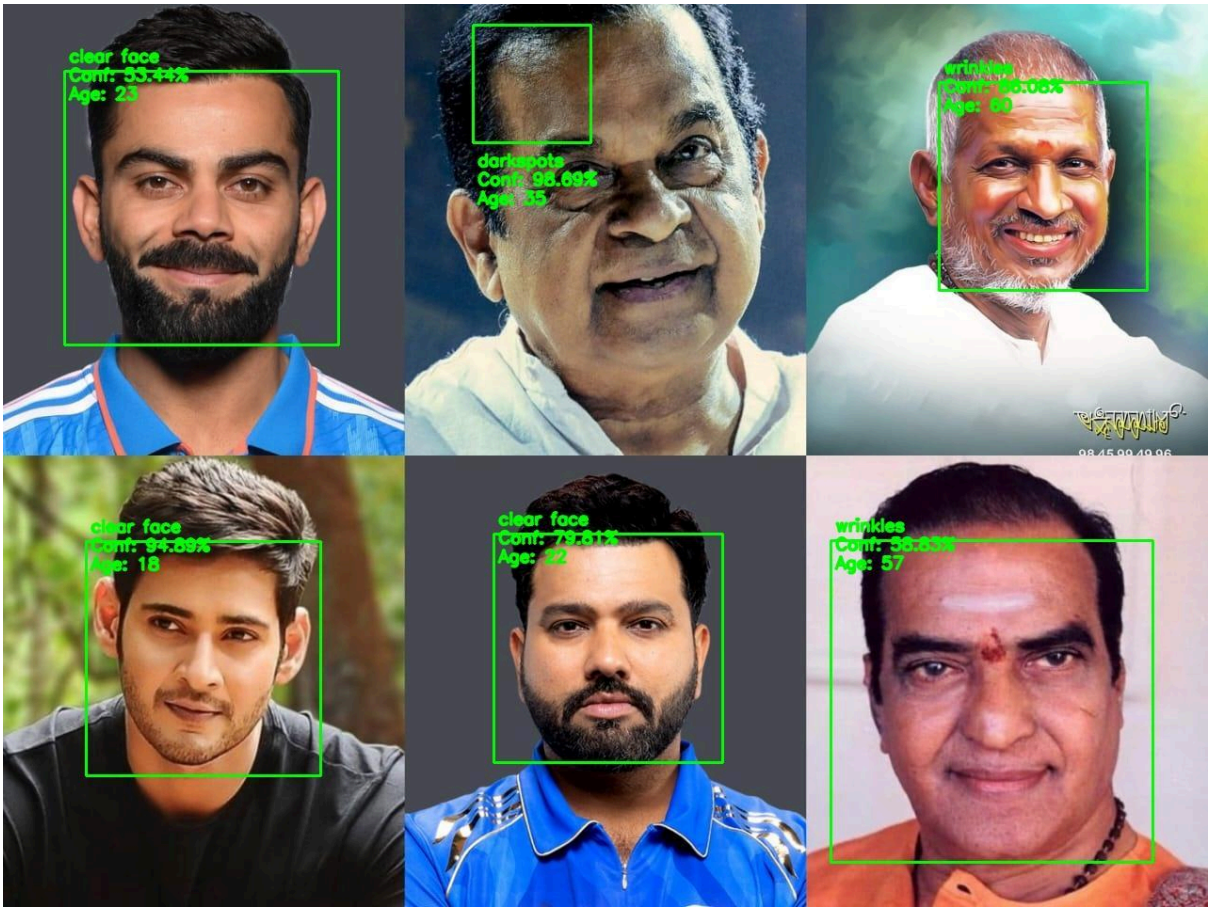
The application supports multiple image uploads or multiple faces within one frame, making it suitable for batch predictions or group photos.

Functionality Overview:

- Users can upload multiple face images simultaneously through the upload section.
- Each image is automatically processed in a loop, and predictions are generated sequentially.
- The app displays the annotated results for all uploaded images, either one below another or using a scrollable layout.
- Each image is labeled with its own results summary for clarity.

For multiple faces in one image:

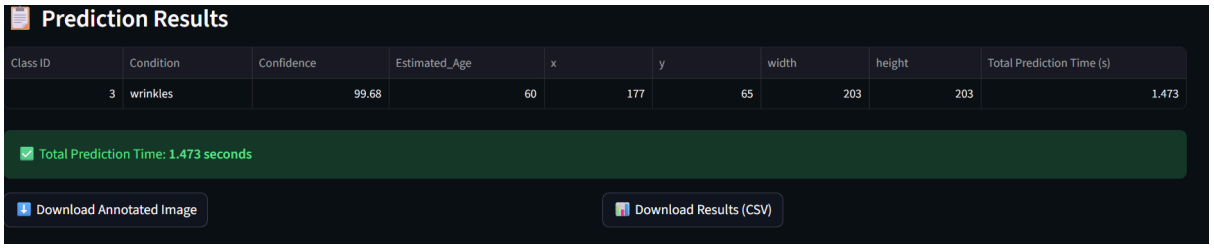
- The system detects all faces in a single frame using Haarcascade.



- For every detected face, a bounding box is drawn, and predictions are displayed beside the corresponding face.
- Each detected region is treated as a separate prediction unit, with its own:
    - Skin Type
    - Confidence (%)
    - Estimated Age

5. **Prediction Summary Table**

- Appears automatically after the model finishes inference.
- Displays all details in a structured and labeled format:



- Ensures **no extra index or empty rows** are shown.
- The class mapping is clearly defined as:
    - 0 → clear face
    - 1 → darkspots
    - 2 → puffy eyes
    - 3 → wrinkles

6. **Download Section**

- Two responsive download buttons are provided:
    - ⬇️ **Annotated Image (.jpg)**

    - 📊 **Prediction Results (.csv)**

- The CSV is compatible with Excel for easy viewing and reporting.

Deliverables

- `app.py` – Streamlit-based frontend for uploading, visualizing, and exporting results.
- Responsive web UI integrated with backend inference.

Evaluation

- Zero lag during file upload and image rendering.
- Accurate annotation display with bounding boxes and readable labels.
- Clear tabular summary and successful data export in CSV format.

Module 6: Backend Pipeline for Model Inference

Tasks Completed

- Developed modular backend (`backend.py`) for **face detection, preprocessing, and inference**.
- Integrated **MobileNetV2** model pre-trained and fine-tuned for facial condition classification.
- Achieved **87% validation accuracy** on labeled facial image dataset.
- Saved best model as **`best_model.h5`** for efficient deployment.
- Implemented image preprocessing pipeline:
    - Resize image to **224×224**
    - Normalize pixel values to **[0,1]**
    - Convert to NumPy array for model input
- Used **Haarcascade classifier** to detect multiple faces per image.

- For each detected face:
    - Extracted face ROI (Region of Interest).
    - Predicted **Condition**, **Confidence**, and **Estimated Age**.
    - Logged **bounding box coordinates (x, y, width, height)**.
    - Annotated face regions with labels and bright yellow text below the box.
- Added **total prediction time calculation** for every uploaded image.
- Returned both the annotated output image and detailed prediction logs to the frontend.

Milestone 4: Finalization and Delivery

Module 7: Export and Logging
Tasks:

- Enable users to download annotated images and CSV files containing prediction results.
- Test the system on diverse image samples to ensure stable performance and finalize results.

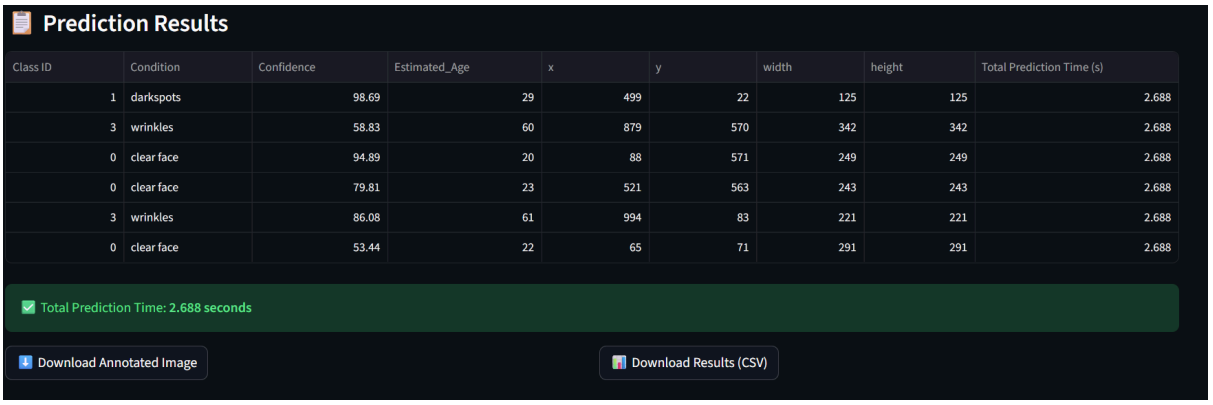
Deliverables:

- Export option integrated into the frontend interface.
- Final logs, CSV files, and annotated image outputs generated.

Evaluation Criteria:

- Accurate and consistent export functionality.
- Properly formatted CSV files with reliable log details.

Backend Log Example (from CSV/Table)



Prediction Results

Class ID	Condition	Confidence	Estimated_Age	x	y	width	height	Total Prediction Time (s)
1	darkspots	98.69	29	499	22	125	125	2.688
3	wrinkles	58.83	60	879	570	342	342	2.688
0	clear face	94.89	20	88	571	249	249	2.688
0	clear face	79.81	23	521	563	243	243	2.688
3	wrinkles	86.08	61	994	83	221	221	2.688
0	clear face	53.44	22	65	71	291	291	2.688

Total Prediction Time: 2.688 seconds

Download Annotated Image

Download Results (CSV)

	A	B	C	D	E	F	G	H	I	J	K
1	Class ID	Condition	Confidenc	Estimated_	x	y	width	height	Total Prediction Time (s)		
2	1	darkspots	98.69	29	499	22	125	125	2.688		
3	3	wrinkles	58.83	60	879	570	342	342	2.688		
4	0	clear face	94.89	20	88	571	249	249	2.688		
5	0	clear face	79.81	23	521	563	243	243	2.688		
6	3	wrinkles	86.08	61	994	83	221	221	2.688		
7	0	clear face	53.44	22	65	71	291	291	2.688		
8											
9											
10											

Deliverables

- backend.py – Modular inference and preprocessing script integrated with Streamlit.
- Logs containing all bounding box coordinates and prediction data for each face.

Evaluation

- End-to-end pipeline successfully tested and integrated.
- Predictions are displayed accurately in UI and exported as CSV.
- Bounding boxes correctly highlight face regions.

- Total processing time per image maintained under 5 seconds.

Module 8: Documentation and Final Presentation

Tasks:

- Prepare comprehensive user and developer guides (README).
- Organize project files on GitHub and create presentation slides or a walkthrough video.

Deliverables:

- Final project documentation and GitHub repository.
- Presentation slides and optional demo video for final review.

Evaluation Criteria:

- Well-structured and clear documentation.
- Demo-ready and professionally presented output.

Tech Stack Overview

Area	Tools / Libraries
Image Operations	OpenCV, NumPy, Haarcascade
Model	TensorFlow / Keras (MobileNetV2, 87% Validation Accuracy)
Dataset	Labeled Facial Skin Condition Dataset
Frontend	Streamlit (Responsive UI)
Backend	Python (Modular Inference Pipeline)
Evaluation Metrics	Accuracy, Loss, Confusion Matrix
Exporting Options	CSV, Annotated Image (JPG), Optional PDF