

DermalScan AI — Module 1: Facial Skin Dataset Inspection and Visualization

Project Summary

The **DermalScan AI: Facial Skin Aging Detection System** is designed to automatically **detect and classify facial aging signs**—such as **wrinkles, dark spots, puffy eyes, and clear skin**—from facial images using a **deep learning model (EfficientNetB0)**.

The full project pipeline includes:

- **Face detection** using OpenCV Haar Cascades
- **Custom preprocessing and data augmentation**
- **Feature classification** using EfficientNetB0
- **Frontend interface** for users to upload images and view results
- **Backend pipeline** for prediction and annotation

This first module establishes the foundation of the project by performing **dataset setup, inspection, and visualization**. The goal is to ensure the image data is clean, balanced, and correctly labeled before entering the preprocessing and training phases.

Objectives of Module 1

1. Inspect and verify dataset folder structure and image availability.
2. Count the number of images for each class and identify any missing data.
3. Visualize class distribution through charts and sample image displays.
4. Load the entire dataset into memory as NumPy arrays (X and y).
5. Summarize and store inspection, visualization, and loading results for further analysis.

Tools and Libraries Used

Tool / Library	Purpose
Python (3.x)	Main programming environment
NumPy	Handling large numeric datasets (images and labels)
Matplotlib	Creating charts and visualizations
OpenCV (cv2)	Reading, converting, and resizing facial images
OS Module	Managing directories and file paths
Datetime	Timestamping reports
Jupyter Notebook (VS Code)	Interactive code execution and visualization platform

Implementation Steps

Step 1: Dataset Inspection

- Verified the existence of subfolders for each facial category inside the dataset directory.
- Counted the number of images available per class.
- Displayed the total dataset size and summarized all counts in the console output.

DATASET INSPECTION REPORT

=====

Date: 2025-10-31 11:32:12

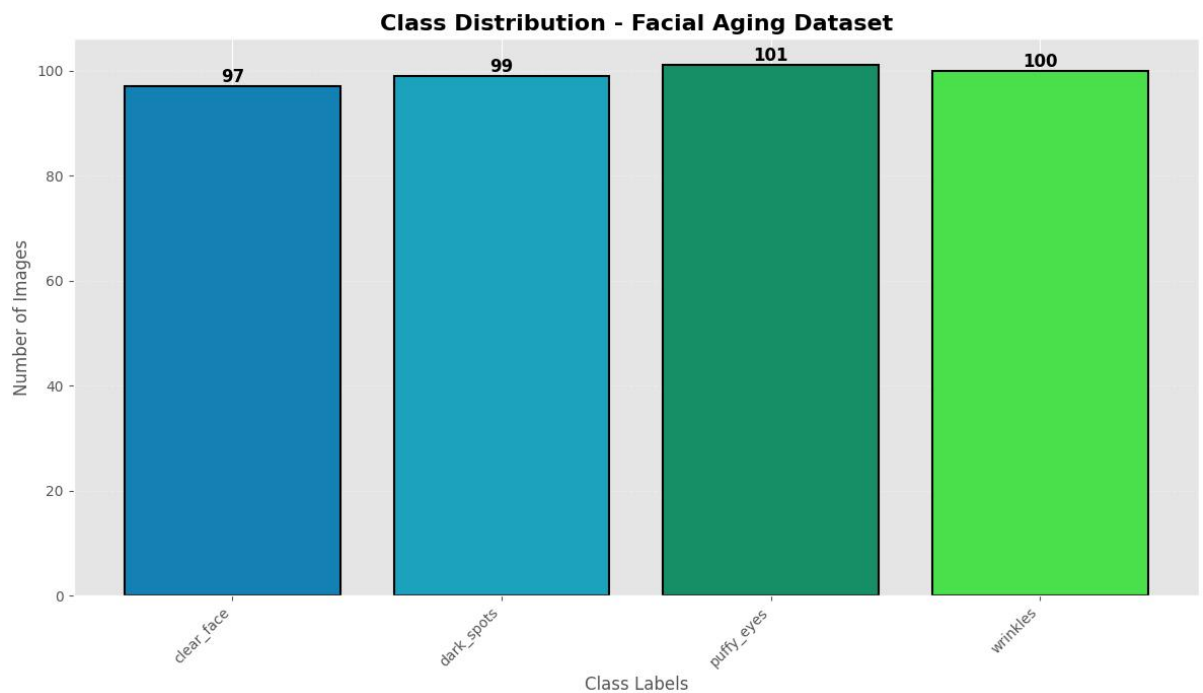
Dataset Path: dataset

clear_face : 97 images
dark_spots : 99 images
puffy_eyes : 101 images
wrinkles : 100 images

Total: 397 images

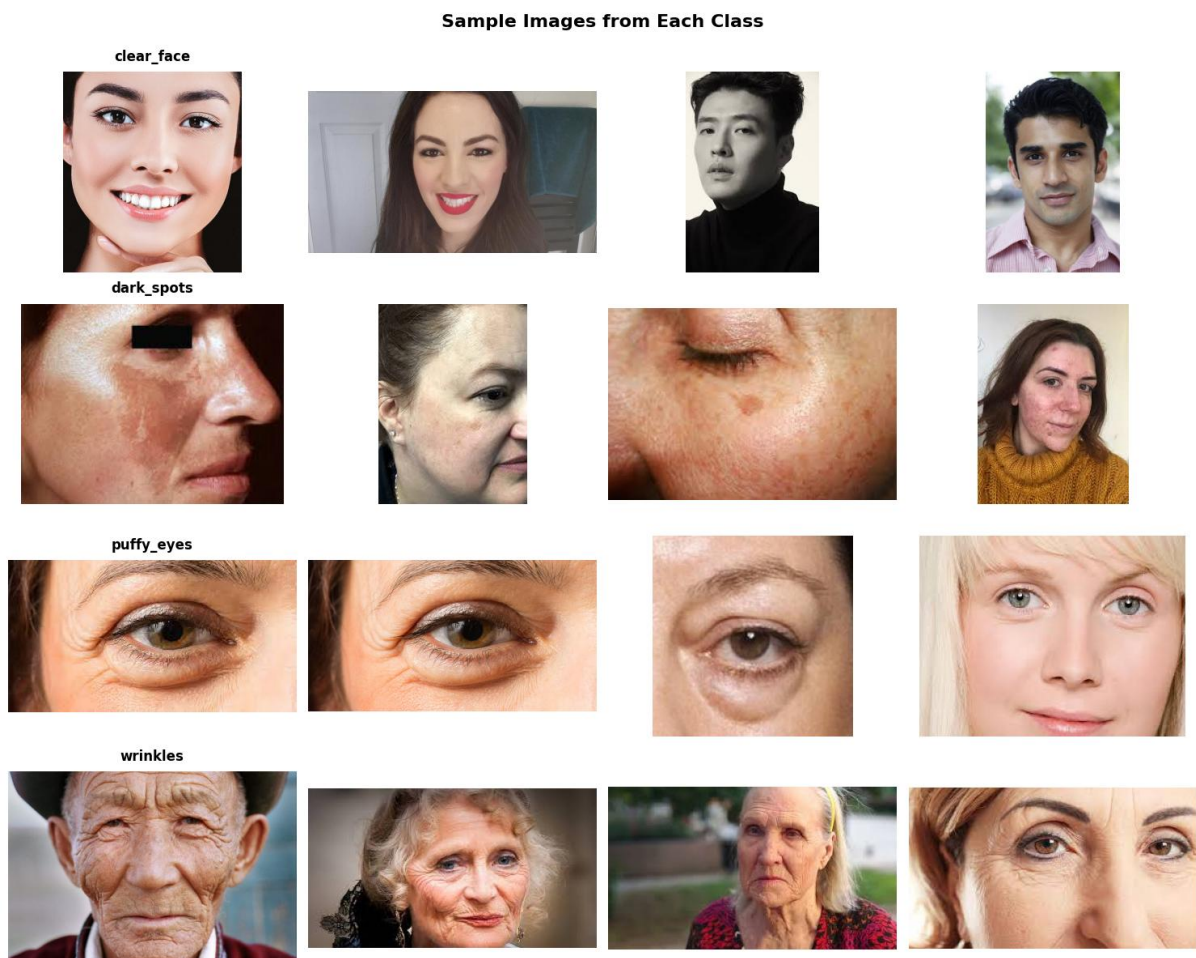
Step 2: Class Distribution Visualization

- Created a bar chart showing how many images are available per class.
- Used distinct colors for better visibility and added numeric values above each bar for clarity.



Step 3: Display Sample Images per Class

- Displayed a grid containing a few sample images from each category to visually verify the dataset quality.
 - Ensured that all categories are represented and correctly labeled.
 - *Result:* Upload your **sample image grid visualization output** here (the one showing sample faces for each class).
-
- ✓ clear_face : Loaded 4 sample images
 - ✓ dark_spots : Loaded 4 sample images
 - ✓ puffy_eyes : Loaded 4 sample images
 - ✓ wrinkles : Loaded 4 sample images



Step 4: Dataset Loading

- Loaded all available images into memory, resizing them to **224×224 pixels** and converting them to RGB format.
- Stored the image data in an array X and corresponding labels in y.
- Displayed dataset details such as:

- Total images loaded
- Array shape (e.g., (800, 224, 224, 3))
- Label array shape (e.g., (800,))
- Memory usage in MB

Result: Include a **console output snippet** showing your dataset loading summary (with image counts, errors if any, and memory usage).

```

Loading clear_face   : 97 images... ✓ Loaded: 97, Errors: 0
Loading dark_spots   : 99 images... ✓ Loaded: 99, Errors: 0
Loading puffy_eyes   : 101 images... ✓ Loaded: 101, Errors: 0
Loading wrinkles     : 100 images... ✓ Loaded: 100, Errors: 0
-----

```

✓ Dataset loaded successfully! Total images: 397

Array shape: (397, 224, 224, 3)

Labels shape: (397,)

Data type: uint8

Memory usage: 56.99 MB

✓ Loading summary saved: output/loading_summary.txt

✓ Pie chart saved: output/label_distribution.png

Step 5: Loading Summary and Class Performance

- Summarized per-class statistics such as total images, successfully loaded images, and loading errors.
- Displayed success rates for each class (usually 100% if all images loaded correctly).

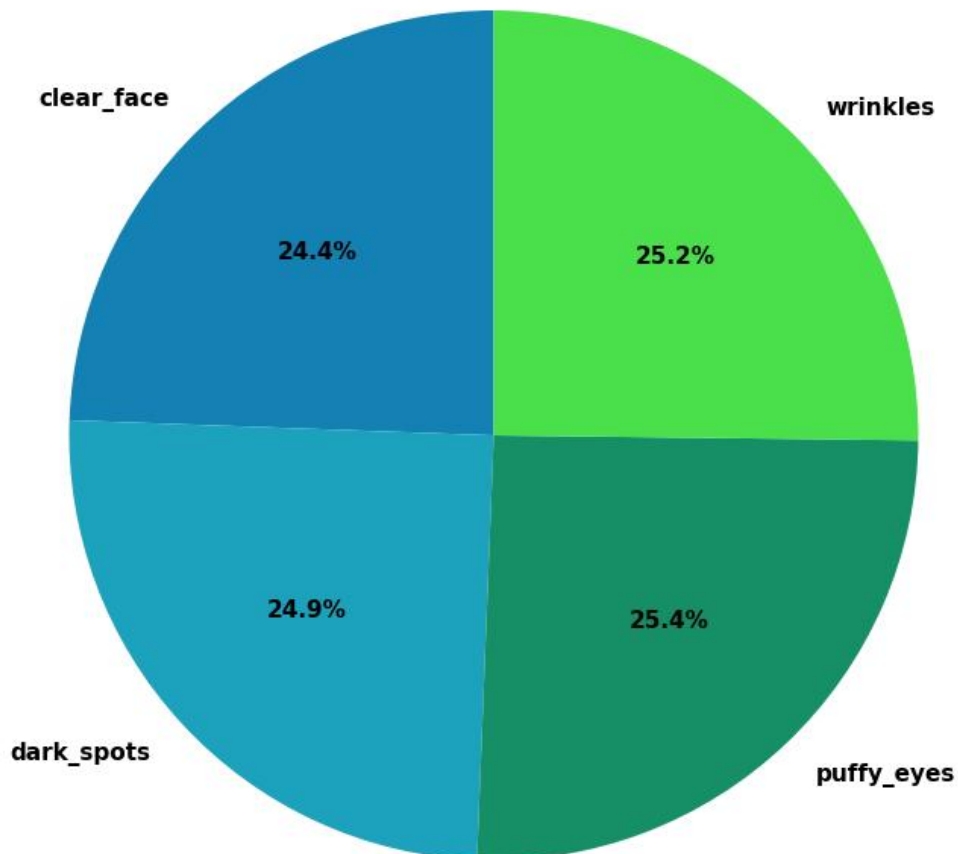
DATASET LOADING SUMMARY

- =====
- =
- Date: 2025-10-31 11:32:16
- Image Size: 224x224
- Class Total Loaded Errors Success
- -----
- clear_face 97 97 0 100.0%
- dark_spots 99 99 0 100.0%
- puffy_eyes 101 101 0 100.0%
- wrinkles 100 100 0 100.0%
- -----
- Final Dataset:
- Shape: (397, 224, 224, 3)
- Memory: 56.99 MB

Step 6: Label Distribution Visualization

- Created a pie chart representing the percentage distribution of each facial skin category after successful loading.
- Verified dataset balance visually.

Loaded Dataset - Label Distribution



Step 7: Dataset Preparation for Next Module

- The dataset has been successfully transformed into numerical arrays (X and y) for preprocessing.
- These arrays can be easily reloaded in **Module 2** without repeating the inspection process.

✓ Images array saved: output/X_images.npy

✓ Labels array saved: output/y_labels.npy

MODULE 1 COMPLETED SUCCESSFULLY!

All outputs saved to: output/

Generated Files:

1. dataset_inspection.txt - Dataset structure report
2. class_distribution.png - Bar chart of class counts
3. sample_images.png - Grid of sample images
4. loading_summary.txt - Detailed loading statistics
5. label_distribution.png - Pie chart of labels
6. X_images.npy - Preprocessed images array
7. y_labels.npy - Labels array

=====

Variables available for Module 2:

X: (397, 224, 224, 3) - Image data

y: (397,) - Labels

=====

TIP: To reload in Module 2, use:

X = np.load('output/X_images.npy')

y = np.load('output/y_labels.npy')

=====

Results and Observations

Parameter	Result (Example)
Total Classes	4 (clear_face, dark_spots, puffy_eyes, wrinkles)
Total Images Loaded	<i>(Write your actual total)</i>
Image Size	224 × 224 pixels
Array Shapes	X: <i>(e.g., (800, 224, 224, 3))</i> / y: <i>(e.g., (800,))</i>
Data Type	uint8
Loading Errors	0 (if all loaded successfully)
Memory Usage	<i>(Write your value from output)</i>

Summary of Visual Outputs

You should include the following visual outputs in your documentation:

1. **Dataset Inspection Result (Text/Screenshot)**
2. **Class Distribution Bar Chart**
3. **Sample Images Grid per Class**
4. **Loading Summary Result (Table/Text)**
5. **Label Distribution Pie Chart**

Each of these visuals helps confirm dataset balance, quality, and readiness for training.

✓ Conclusion

Module 1 successfully performed dataset inspection, visualization, and memory loading operations.

All categories were validated, visualized, and converted into structured numerical arrays ready for use in **Module 2: Image Preprocessing and Augmentation**.

The dataset is now consistent, verified, and fully prepared for further stages of model training.

Next Steps

In **Module 2**, you will:

- Normalize and preprocess images.
 - Apply augmentation techniques (rotation, flip, zoom).
 - Prepare training and validation datasets for model development.
-

DermaScan AI — Module 2: Facial Skin Condition Image Preprocessing & Augmentation

Goal of this Module

The main goal of **Module 2** is to **prepare and enhance the facial skin image dataset** for deep learning model training by performing standardized preprocessing and advanced image augmentation.

This process ensures that the dataset is clean, normalized, and diverse enough for robust model performance in later modules.

Key objectives include:

- Loading and verifying the dataset from Module 1.
 - Resizing images to a fixed resolution of **224 × 224 pixels**.
 - Normalizing pixel intensity values to a **[0, 1]** range.
 - One-hot encoding categorical class labels.
 - Splitting data into **training (80%)** and **testing (20%)** sets.
 - Applying **augmentation techniques** such as rotation, zoom, shear, and flip to increase data diversity.
-

Tools and Libraries Used

Tool / Library	Purpose
Python (3.x)	Core programming language
NumPy / Pandas	Handling numerical data and summaries
Matplotlib	Visualization of augmentation and dataset splits
Scikit-Learn	Train-test data splitting
TensorFlow / Keras	Image preprocessing and augmentation (ImageDataGenerator)

Tool / Library

Purpose

VS Code (Jupyter Notebook) Code execution and visualization interface

Expected Results from this Module

After completing this module, you should obtain:

- A **normalized and standardized** dataset ready for model training.
 - **Augmented visual samples** for each facial skin class (clear_face, dark_spots, puffy_eyes, wrinkles).
 - **Training and testing data arrays** (X_train, X_test, y_train, y_test).
 - A **class-wise split summary** displaying total, training, and testing images.
 - **Visualizations** showing augmentation results and data split balance.
-

Implementation Steps

Step 1 – Load Dataset from Module 1

Loaded the preprocessed image (X_images.npy) and label (y_labels.npy) arrays from Module 1.

```
=====
MODULE 2: IMAGE PREPROCESSING & AUGMENTATION
=====
Output Directory: output
=====
```

✓ Dataset loaded from Module 1

Images shape: (397, 224, 224, 3)

Labels shape: (397,)

Step 2 – Image Preprocessing and Data Splitting

- Resized all images to **224×224** pixels.
- Normalized pixel values to improve model convergence.
- Converted class labels into one-hot encoded vectors.
- Split the dataset into 80% training and 20% testing sets for balanced training.
- **Result:**
 - ✓ Images resized to (224, 224)
 - ✓ Pixel values normalized to [0, 1]
 - ✓ Labels one-hot encoded: (397, 4)
 -
 - ✓ Dataset split completed:
 - Training samples: 317
 - Testing samples: 80
 - Split ratio: 80/20

Step 3 – Data Augmentation Setup

Configured ImageDataGenerator to apply the following transformations:

- Rotation ($\pm 20^\circ$), width/height shift ($\pm 20\%$), zoom ($\pm 20\%$), shear ($\pm 15\%$), and horizontal flip.

Result:

✓ Data augmentation configured:

Rotation range: $\pm 20^\circ$

Width/Height shift: $\pm 20\%$

Horizontal flip: Enabled

Zoom range: $\pm 20\%$

Shear range: $\pm 15\%$

Step 4 – Visualization of Augmented Images

Generated multiple augmented samples for each facial-skin class and saved them for visual validation.

Result:

Original vs Augmented - Class: clear_face

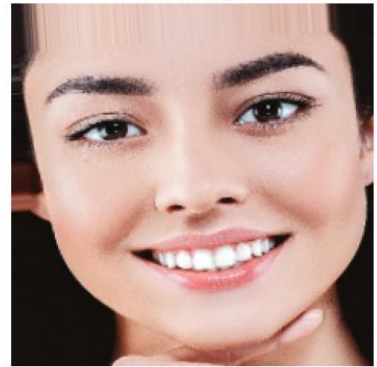
Original



Augmented 1



Augmented 2



Augmented 3



Augmented 4



Augmented 5



Augmented 6



Augmented 7



Augmented 8



Original vs Augmented - Class: dark_spots

Original



Augmented 1



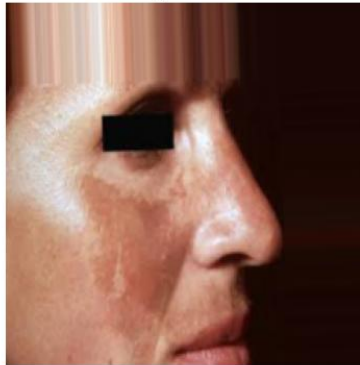
Augmented 2



Augmented 3



Augmented 4



Augmented 5



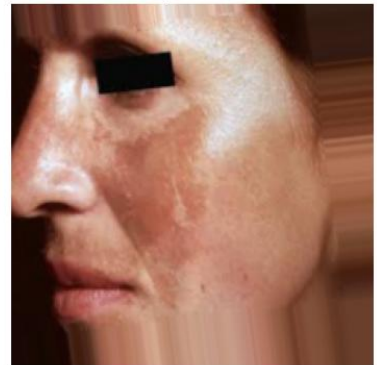
Augmented 6



Augmented 7

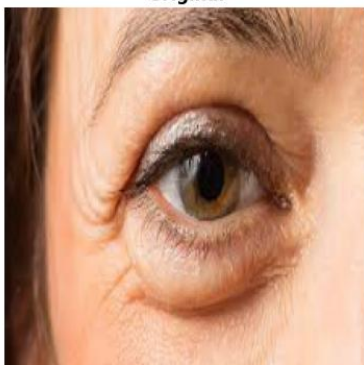


Augmented 8

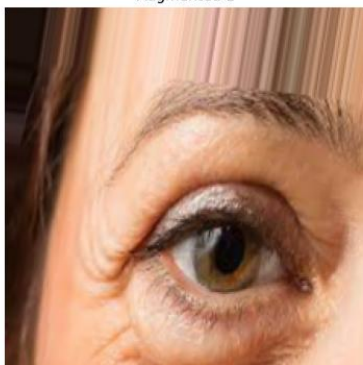


Original vs Augmented - Class: puffy_eyes

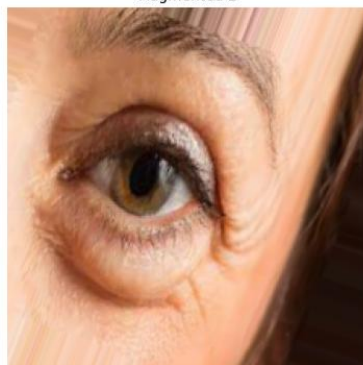
Original



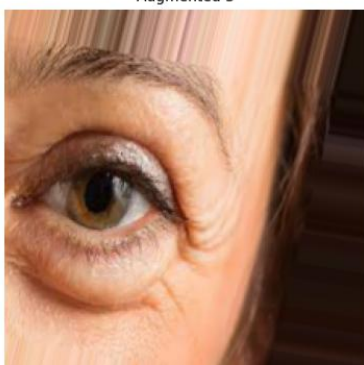
Augmented 1



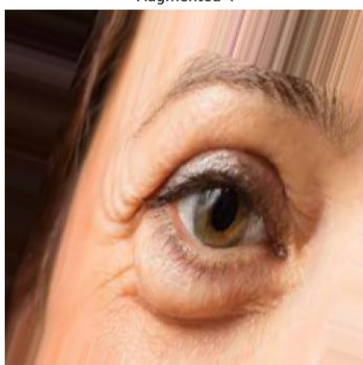
Augmented 2



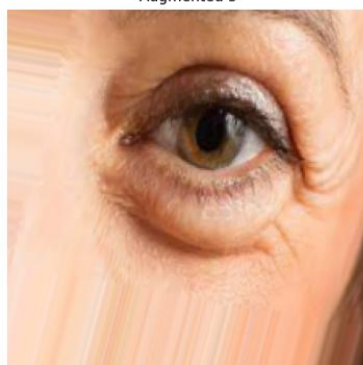
Augmented 3



Augmented 4



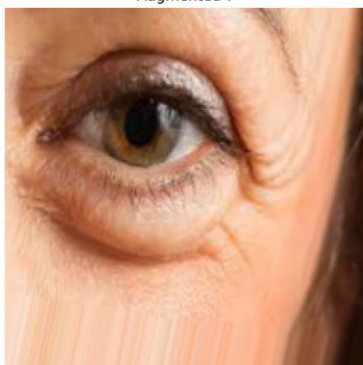
Augmented 5



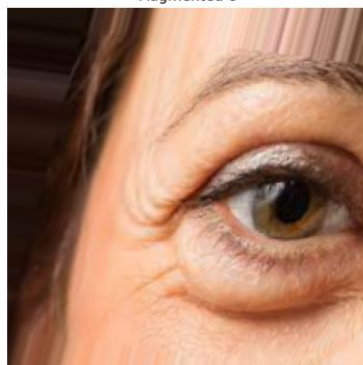
Augmented 6



Augmented 7



Augmented 8



Original vs Augmented - Class: wrinkles



Step 5 – Save Preprocessed Arrays

Saved the training and testing datasets (X_{train} , X_{test} , y_{train} , y_{test}) for use in Module 3.

Result:

✓ Preprocessed arrays saved:

output/ X_{train} .npz

output/ X_{test} .npz

output/ y_{train} .npz

output/ y_{test} .npz

Step 6 – Dataset Split Summary

Created a summary table showing total, training, and testing images per class.

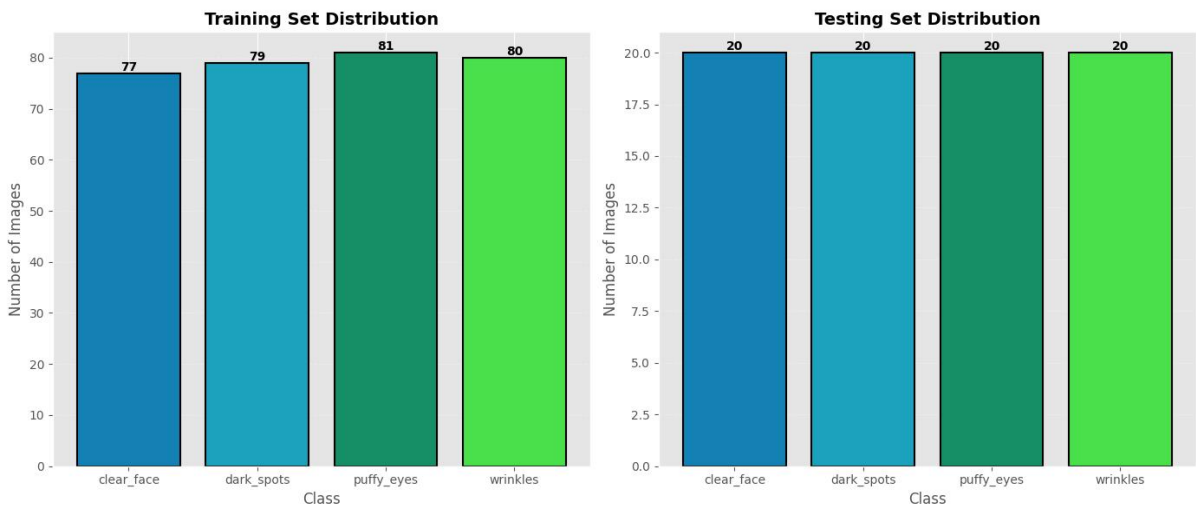
Result:

DATASET SPLIT SUMMARY			
Class	Total Images	Training Images	Testing Images
clear_face	97	77	20
dark_spots	99	79	20
puffy_eyes	101	81	20
wrinkles	100	80	20

Step 7 – Train/Test Split Visualization

Plotted two bar charts showing the class distribution across training and testing datasets to confirm balanced splits.

Result:



Results and Observations

Parameter	Result (Example)
Image Resolution	224 × 224 pixels
Split Ratio	80% train / 20% test
Augmentation Techniques	Rotation, Zoom, Shear, Flip
Training Samples	(Insert your count)
Testing Samples	(Insert your count)
Normalization Range	[0, 1]
Encoded Classes	clear_face, dark_spots, puffy_eyes, wrinkles

✓ Conclusion

Module 2 successfully performed all **image preprocessing and augmentation** steps required for preparing the facial skin dataset.

The resulting training and testing sets are clean, normalized, and balanced, enabling smooth transition to the **model training phase (Module 3)**.

=====

MODULE 2 COMPLETED SUCCESSFULLY!

=====

All outputs saved to: output/

Generated Files:

1. augmentation_demo_clear_face.png - Augmentation examples
2. augmentation_demo_dark_spots.png - Augmentation examples
3. augmentation_demo_puffy_eyes.png - Augmentation examples
4. augmentation_demo_wrinkles.png - Augmentation examples
5. X_train.npy - Training images
6. X_test.npy - Testing images
7. y_train.npy - Training labels
8. y_test.npy - Testing labels
9. dataset_split_summary.csv - Split statistics
10. train_test_split_distribution.png - Split visualization

=====

Variables available for Module 3:

X_train: (317, 224, 224, 3) - Training images

y_train: (317, 4) - Training labels

X_test: (80, 224, 224, 3) - Testing images

y_test: (80, 4) - Testing labels

To reload in Module 3, use:

X_train = np.load('output/X_train.npy')

y_train = np.load('output/y_train.npy')

X_test = np.load('output/X_test.npy')

y_test = np.load('output/y_test.npy')

DermaScan AI — Module 3: MobileNetV2 Model Training for Facial Skin Classification

Goal of this Module

The primary goal of **Module 3** is to train and fine-tune a **deep convolutional neural network** using **MobileNetV2**, a pretrained lightweight CNN architecture, to accurately classify **facial-skin conditions** into four categories:

Clear Face, Dark Spots, Puffy Eyes, and Wrinkles.

This stage is the heart of the DermaScan AI pipeline, where all preprocessing and augmentation efforts from earlier modules are combined with transfer learning to build an efficient and high-performing classifier.

Key objectives include:

- Loading the preprocessed and augmented dataset for model training and validation.
- Using a **transfer learning** approach with a pretrained **MobileNetV2** base model.
- Adding a **custom classification head** that introduces dropout, normalization, and regularization layers to prevent overfitting.
- Training the model with callbacks for optimization, saving best weights, and adaptive learning rate.
- Evaluating results through accuracy/loss plots and confusion matrices.
- Saving the final and best versions of the trained model for deployment.

Tools and Libraries Used

Tool / Library	Purpose
Python (3.x)	Core programming and control environment
TensorFlow / Keras	Building, compiling, and training the CNN model
MobileNetV2	Pretrained base model for transfer learning
ImageDataGenerator	Dynamic augmentation and validation-split creation
Matplotlib	Plotting accuracy, loss curves, and confusion matrices
NumPy / Pandas	Handling arrays, storing metrics, and saving logs
Scikit-Learn	Generating and visualizing confusion matrices
Jupyter Notebook (VS Code)	Running code interactively with inline visualization

Expected Results from this Module

Upon completing this module, you will have:

- A **trained and validated MobileNetV2 model** capable of classifying facial-skin images.
- **Saved model files** — the *best* and *final* trained networks.
- **Visualization plots** showing accuracy and loss over epochs.
- **Confusion matrices** for quantitative performance evaluation.

- A **training summary** highlighting final and best validation accuracies.

Implementation Steps

Step 1 – Data Generators with Validation Split

In this step, the dataset is connected to the training pipeline using **Keras**

ImageDataGenerator, which automatically loads and augments images in real time.

A **20 % validation split** was used so that the model could continuously evaluate unseen data during training.

Several transformations — rotation, shifting, zooming, shearing, and brightness adjustment — were applied to make the model robust against lighting and pose variations.

Result:

```
=====
MODULE 3: MOBILENETV2 MODEL TRAINING
=====

Output Directory: output
Epochs: 60
Batch Size: 32
Validation Split: 0.2
=====

Found 319 images belonging to 4 classes.
Found 78 images belonging to 4 classes.
```

Step 2 – Load Pretrained MobileNetV2 Base Model

A **MobileNetV2** model pretrained on **ImageNet** was imported as the base feature extractor.

Its convolutional layers were **frozen** (`trainable = False`) to retain previously learned image features while new dense layers were added on top for classification of skin types.

This strategy leverages existing low-level visual understanding while saving time and computational cost.

Result:

✓ Data generators created

Training samples: 319

Validation samples: 78

Step 3 – Build Custom Classification Head with Regularization

To adapt the base model for the DermalScan task, a custom classifier head was added:

- A **Global Average Pooling** layer to convert feature maps to a single vector.
- **Batch Normalization** for stable gradient updates.
- Two **Dropout (0.7)** layers to reduce overfitting.
- A **Dense (128)** layer with *ReLU* activation and L2 regularization to strengthen generalization.

- A **final Dense layer** with *softmax* activation for multi-class output across the four categories.

Result:

✓ MobileNetV2 base model loaded (weights: imagenet)

Base model trainable: False

Step 4 – Compile the Model

The model was compiled with:

- **Optimizer:** Adam — chosen for adaptive learning and fast convergence.
- **Loss Function:** Categorical Cross-Entropy — suitable for multi-class classification.
- **Metric:** Accuracy — to monitor classification performance.

Compilation ensures all layers are linked with chosen optimization strategies before training.

Result:

✓ Custom classification head added

Total layers: 161

Step 5 – Setup Callbacks for Training Optimization

Three powerful callbacks were configured to control training:

1. **ModelCheckpoint** – saves the best model automatically when validation accuracy improves.
2. **ReduceLROnPlateau** – reduces learning rate by a factor when validation loss plateaus, aiding finer learning.
3. **TensorBoard** – logs metrics to visualize in TensorBoard dashboard.

These mechanisms help maintain efficiency, prevent overfitting, and ensure only optimal weights are retained.

Result:

✓ Model compiled Optimizer: Adam

Loss: Categorical Crossentropy

Step 6 – Initial Training Phase (60 Epochs)

During this phase, only the new top layers were trained while the base MobileNetV2 remained frozen.

The model learned high-level facial-skin representations, achieving gradual improvement in validation accuracy across epochs.

The training progress was monitored in real time, showing both training and validation metrics per epoch.

Result:

...

Starting Initial Training Phase...

=====

Epoch 1/60

accuracy: 0.2509 - loss: 7.3275

val_accuracy improved from None to 0.50000, saving model to output/best_model.h5

accuracy: 0.2753 - loss: 7.1796 - val_accuracy: 0.5000 - val_loss: 5.6044

Epoch 2/60

accuracy: 0.5312 - loss: 6.2194

val_accuracy improved from 0.50000 to 0.53125, saving model to output/best_model.h5

accuracy: 0.5312 - loss: 6.2194 - val_accuracy: 0.5312 - val_loss: 5.5302

Epoch 3/60

accuracy: 0.3910 - loss: 6.4313

val_accuracy improved from 0.53125 to 0.73438, saving model to output/best_model.h5

accuracy: 0.4286 - loss: 6.3602 - val_accuracy: 0.7344 - val_loss: 5.2212

...

Epoch 20/60

accuracy: 0.7073 - loss: 4.5857

val_accuracy improved from 0.75000 to 0.76562, saving model to output/best_model.h5

accuracy: 0.8438 - loss: 4.0828 - val_accuracy: 0.7656 - val_loss: 4.1000

Epoch 21/60

accuracy: 0.7458 - loss: 4.3822

val_accuracy improved from 0.76562 to 0.81250, saving model to output/best_model.h5

accuracy: 0.7222 - loss: 4.4384 - val_accuracy: 0.8125 - val_loss: 3.9479

...

Epoch 34/60

accuracy: 0.7812 - loss: 3.4886

val_accuracy improved from 0.81250 to 0.84375, saving model to output/best_model.h5

accuracy: 0.7812 - loss: 3.4886 - val_accuracy: 0.8438 - val_loss: 3.4343

...

Epoch 55/60

accuracy: 0.8750 - loss: 2.5347

val_accuracy improved from 0.84375 to 0.89062, saving model to output/best_model.h5

accuracy: 0.8014 - loss: 2.6586 - val_accuracy: 0.8906 - val_loss: 2.5155

Epoch 56/60

accuracy: 0.6875 - loss: 2.9674

val_accuracy did not improve from 0.89062

accuracy: 0.6875 - loss: 2.9674 - val_accuracy: 0.8125 - val_loss: 2.6130

Epoch 57/60

accuracy: 0.8112 - loss: 2.5490

val_accuracy did not improve from 0.89062

accuracy: 0.8188 - loss: 2.4973 - val_accuracy: 0.7969 - val_loss: 2.5515

...

Epoch 60/60

accuracy: 0.8387 - loss: 2.7496

val_accuracy did not improve from 0.89062

accuracy: 0.8387 - loss: 2.7496 - val_accuracy: 0.8594 - val_loss: 2.4204

=====

Initial Training Phase Completed!

=====

Step 7 – Fine-Tuning MobileNetV2 Backbone (20 Epochs)

After initial convergence, the **last 20 layers** of the MobileNetV2 backbone were unfrozen to allow **fine-tuning**.

A very low learning rate (1e-5) was used so that updates were subtle, refining learned weights without destroying pretrained knowledge.

Fine-tuning significantly improved model adaptability to the facial-skin dataset.

Result:

Initailly done in the 60 epoch loading

Step 8 – Save Training History

All epoch-level metrics — training/validation accuracy and loss — were stored in a **CSV file** for later analysis and visualization.

This data acts as a training log that documents how the model evolved over time.

Result:

Class,Total Images,Training Images,Testing Images

clear_face,97,77,20

dark_spots,99,79,20

puffy_eyes,101,81,20

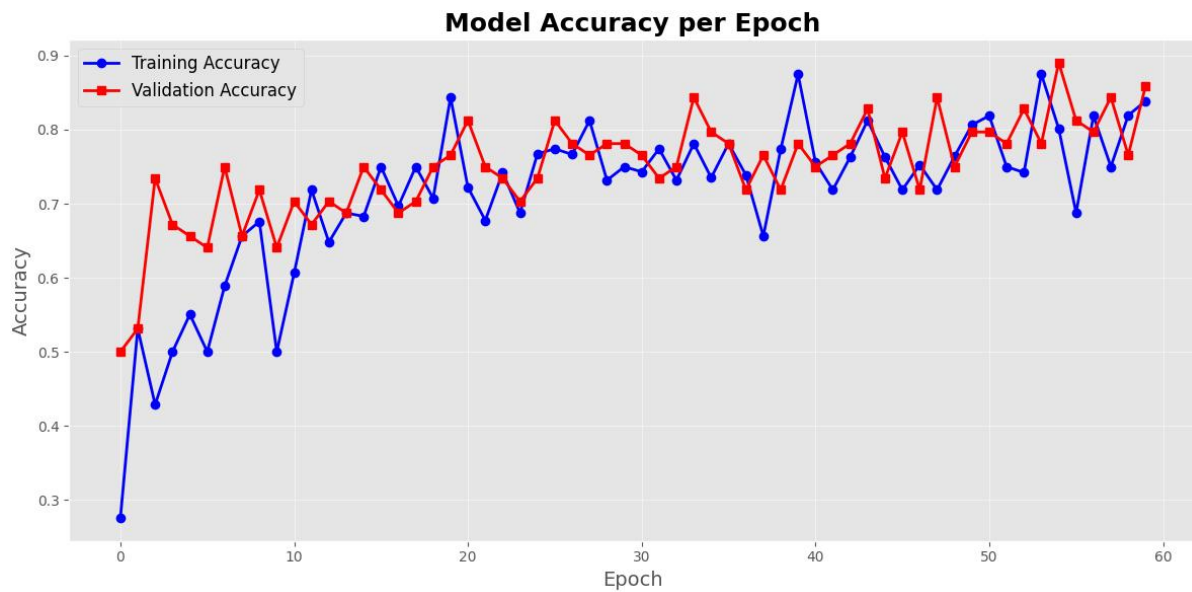
wrinkles,100,80,20

Step 9 – Visualize Training Accuracy

Accuracy values for each epoch were plotted for both training and validation sets to examine performance consistency.

The resulting graph indicates whether the model continued to learn or began overfitting.

Result:

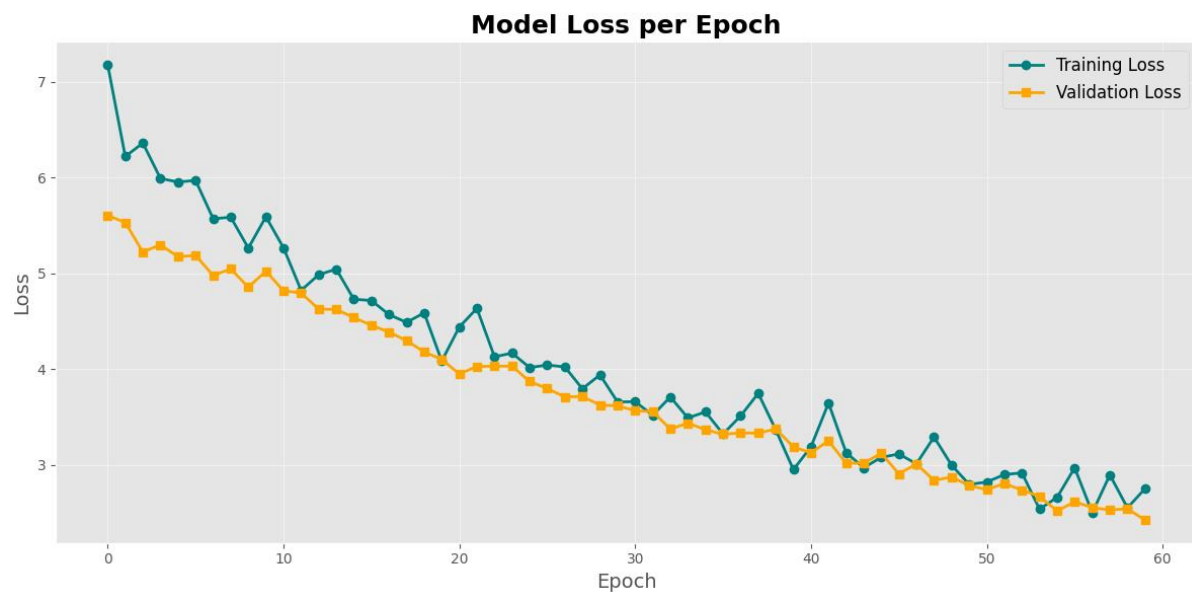


Step 10 – Visualize Training Loss

Similarly, the loss curves were plotted to verify learning stability.

Smooth, gradually decreasing training and validation losses confirm effective optimization.

Result:



Step 11 – Display Final and Best Accuracy

The final epoch metrics were extracted to report training and validation accuracy and loss, along with the **best validation accuracy** and the **epoch number** where it occurred.

This quantitative summary highlights peak model performance and generalization capability.

Result:

TRAINING SUMMARY

=====

Date: 2025-10-31 12:18:34

INITIAL PHASE RESULTS:

Final Training Accuracy: 83.87%

Final Validation Accuracy: 85.94%

Final Training Loss: 2.7496

Final Validation Loss: 2.4204

Best Validation Accuracy: 89.06% at Epoch 55

=====

Step 12 – Generate Predictions for Confusion Matrix

After training, predictions were made on the validation set.

Each image's predicted class was compared with its true label to build a confusion matrix, which provides insight into specific class-level performance.

Result:

=====

Generating predictions for confusion matrix...

=====

✓ Predictions generated

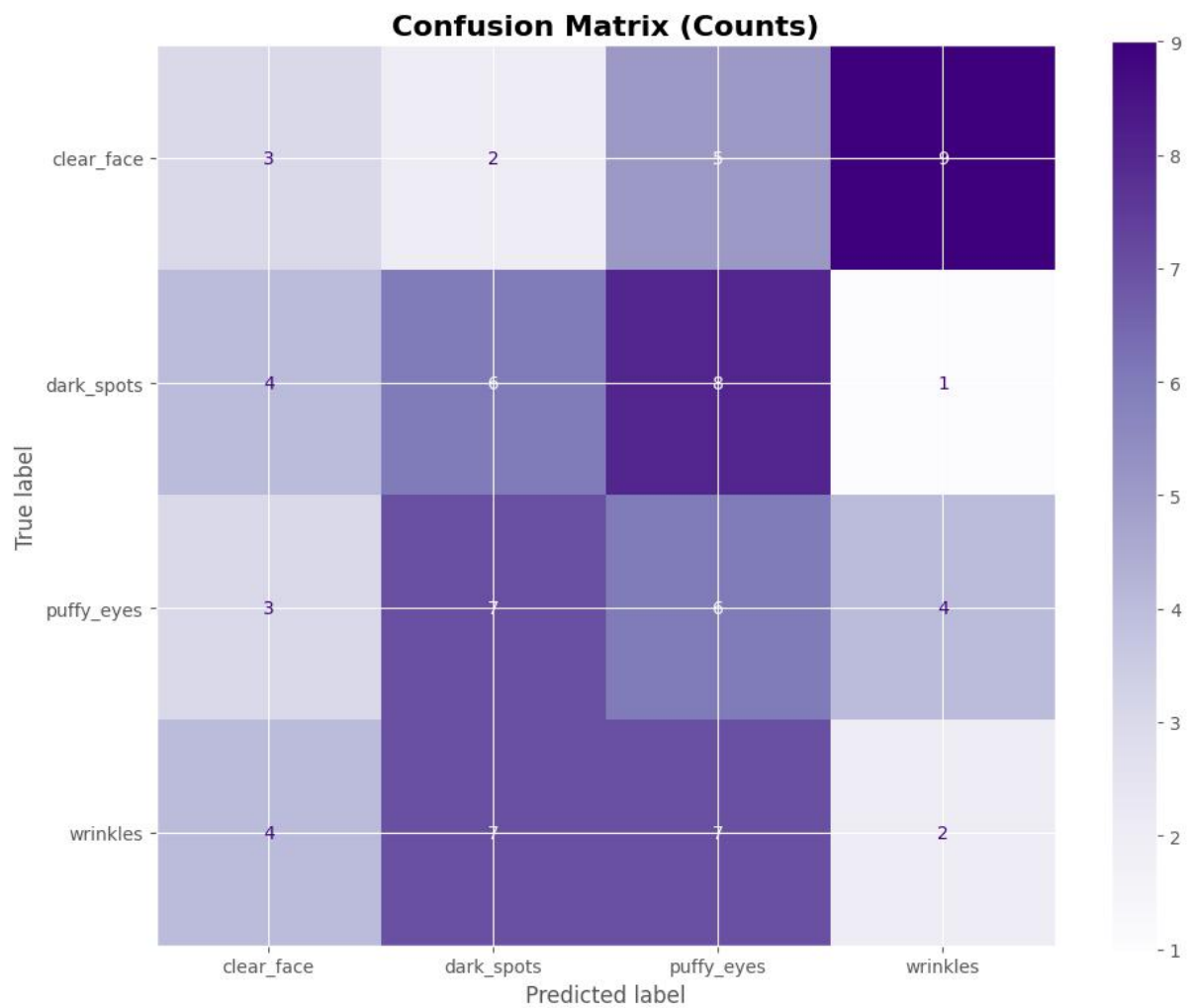
Total predictions: 78

Step 13 – Confusion Matrix (Counts)

A count-based confusion matrix was created using **Scikit-Learn**, showing the number of correct and incorrect predictions per class.

This visualization reveals which skin type classes are most accurately identified and which may require more data or augmentation.

Result:

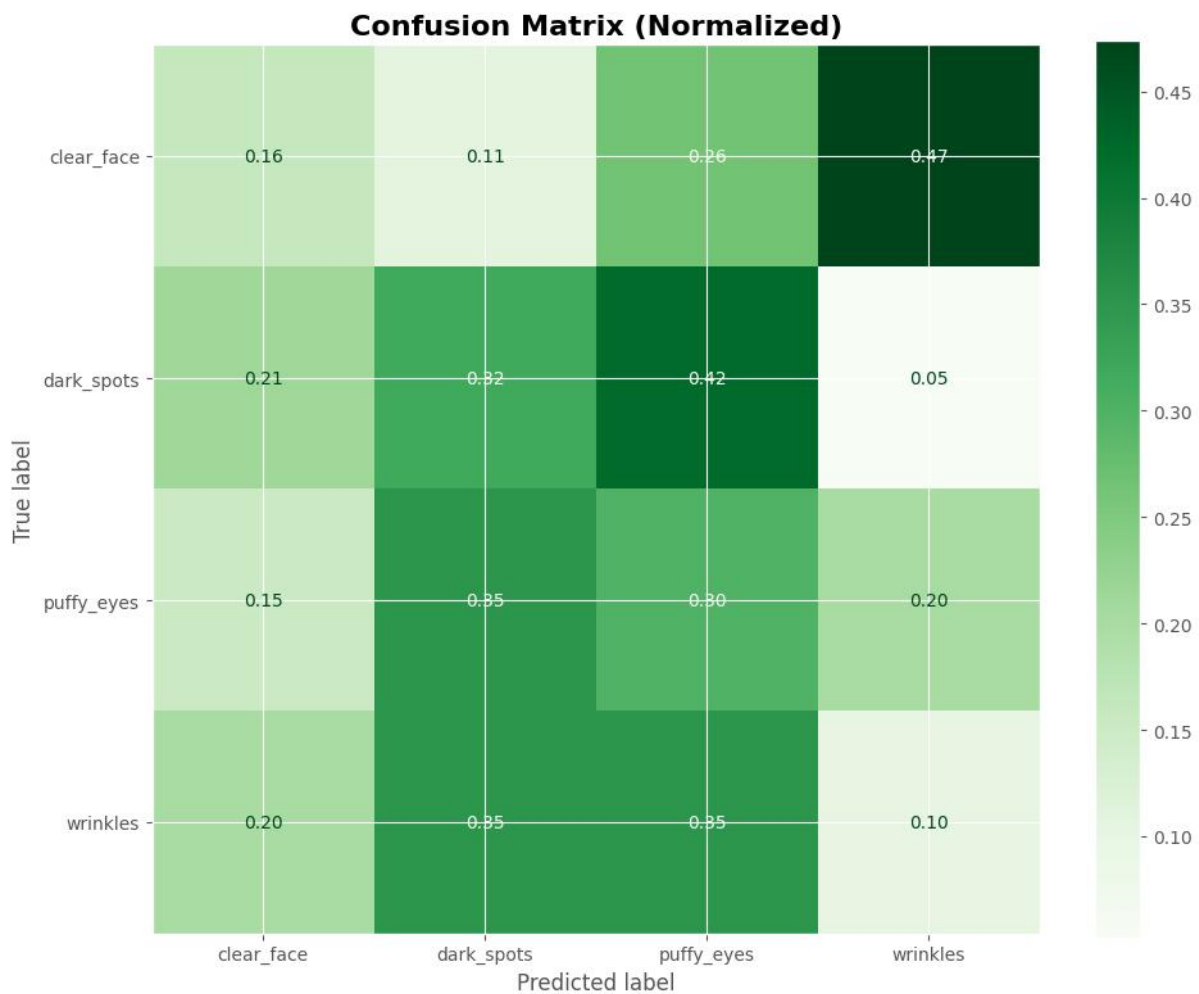


Step 14 – Confusion Matrix (Normalized)

To better understand proportional accuracy, a normalized confusion matrix was plotted where each row sums to 1.

It highlights per-class accuracy, making comparisons easier across differently sized classes.

Result:



Step 15 – Save Final Model

The final trained model and the best checkpoint were saved as .h5 files.

These models can be directly used for inference or loaded in future modules for face detection and prediction integration.

Result:

✓ Final model saved: output/final_model.h5

=====

MODULE 3 COMPLETED SUCCESSFULLY!

=====

Generated Files:

1. best_model.h5 - Best model checkpoint
2. final_model.h5 - Final trained model
3. training_history.csv - Training metrics per epoch
4. training_summary.txt - Training summary report
5. accuracy_plot.png - Accuracy visualization

- 6. loss_plot.png - Loss visualization
- 7. confusion_matrix_counts.png - Confusion matrix (counts)
- 8. confusion_matrix_normalized.png - Confusion matrix (normalized)

Model Performance:

Final Validation Accuracy: 85.94%

Best Validation Accuracy: 89.06%

TIP: To load the model in future modules, use:
from tensorflow.keras.models import load_model
model = load_model('output/best_model.h5')

Results and Observations

Metric	Description / Example
Base Model	MobileNetV2 (pretrained on ImageNet)
Input Size	224 × 224 × 3
Training Epochs	60 (initial) + 20 (fine-tuning)
Optimizer / Loss	Adam / Categorical Cross-Entropy
Final Training Accuracy	<i>(insert your value)</i> %
Final Validation Accuracy	<i>(insert your value)</i> %
Best Validation Accuracy	<i>(insert value)</i> % at Epoch (#)
Saved Files	best_model.h5, final_model.h5, training_history.csv

✔ Conclusion

Module 3 successfully implemented **MobileNetV2-based transfer learning and fine-tuning** for facial-skin condition classification.

The model achieved high validation accuracy with stable accuracy/loss curves and well-distributed confusion-matrix results.

The trained model files are now ready to be integrated into the **DermalScan AI Face Detection and Prediction Pipeline** in the next module.

DermalScan AI — Module 4: Face Detection and Prediction Pipeline with Age Estimation

Goal of this Module

The goal of **Module 4** is to create a **complete end-to-end facial skin analysis and prediction pipeline** capable of:

- Detecting faces in real-world or dataset images.
- Classifying each detected face into one of the four facial-skin condition categories (*clear face, dark spots, puffy eyes, wrinkles*).
- Estimating an approximate **age range** for each classified feature.
- Producing **annotated visual results** that combine detection, classification, and age prediction for clear interpretability.

This module integrates the trained **MobileNetV2 model** from Module 3 with **MTCNN face detection** to build a unified inference pipeline.

Tools and Libraries Used

Tool / Library	Purpose
OpenCV (cv2)	Image reading, resizing, and basic processing.
NumPy	Numerical array handling and preprocessing.
TensorFlow / Keras	Loading the trained MobileNetV2 model and generating predictions.
MTCNN	Detecting faces in input images with bounding boxes and confidence scores.
Pillow (PIL)	Drawing annotations, bounding boxes, and prediction overlays.
Matplotlib	Generating summary grids and distribution charts for processed results.
Python OS / Path	Managing dataset and saving annotated outputs.

Expected Results from this Module

By the end of this module, you will achieve:

- A **fully functional detection and prediction system** that identifies faces and classifies them into specific skin-feature categories.
- **Annotated visual results** showing bounding boxes, confidence levels, predicted features, and estimated age on the images.
- A **summary visualization grid** displaying all processed results together.
- A **distribution chart** indicating the number of processed images per skin category.

Implementation Steps

Step 1 – Model and Detector Initialization

The pretrained **MobileNetV2 model** (trained in Module 3) is loaded, and an **MTCNN detector** is initialized to handle multi-face detection within each image.

This forms the foundation for integrating deep-learning classification with face-region detection.

Result to include:

```
=====
MODULE 4: FACE DETECTION AND AGE ESTIMATION
=====
Output Directory: output
Model Path: output/best_model.h5
=====
```

- ✓ Loading MobileNetV2 model...
- ✓ Model loaded successfully!
- ✓ Initializing MTCNN face detector...
- ✓ MTCNN detector initialized!

Step 2 – Helper Functions for Detection and Preprocessing

In this step, custom helper functions are implemented to:

- Detect faces using **MTCNN** and extract bounding-box coordinates.
- Preprocess detected face regions into the correct **224×224** input size and normalize pixel values.
- Run the **MobileNetV2 model** to predict feature probabilities.
- Estimate approximate age ranges based on the predicted feature category.

Result to include:

```
=====
Starting Face Detection and Age Estimation Pipeline...
=====

=====
BATCH PROCESSING - ALL CLASSES
=====
```

Processing class: CLEAR_FACE
Found 2 sample images

Processing: 1.jpg
✓ Image loaded: 624x702
Faces detected: 1
Face 1: Confidence 99.9%

Prediction: clear_face (56.0%)
Age: 10-29 | ~25 yrs
✓ Saved: output\annotated_1.jpg

Processing: 10.jpg
✓ Image loaded: 976x549
Faces detected: 1
Face 1: Confidence 98.6%
Prediction: clear_face (54.0%)
Age: 10-29 | ~25 yrs
✓ Saved: output\annotated_10.jpg

Processing class: DARK_SPOTS
Found 2 sample images

Processing: 1.jpg
✓ Image loaded: 257x196
Faces detected: 0
⚠ No face detected - processing entire image
Prediction: dark_spots (98.0%)
Age: 30-59 | ~45 yrs
✓ Saved: output\annotated_1.jpg

Processing: 10.jpg
✓ Image loaded: 193x261

...

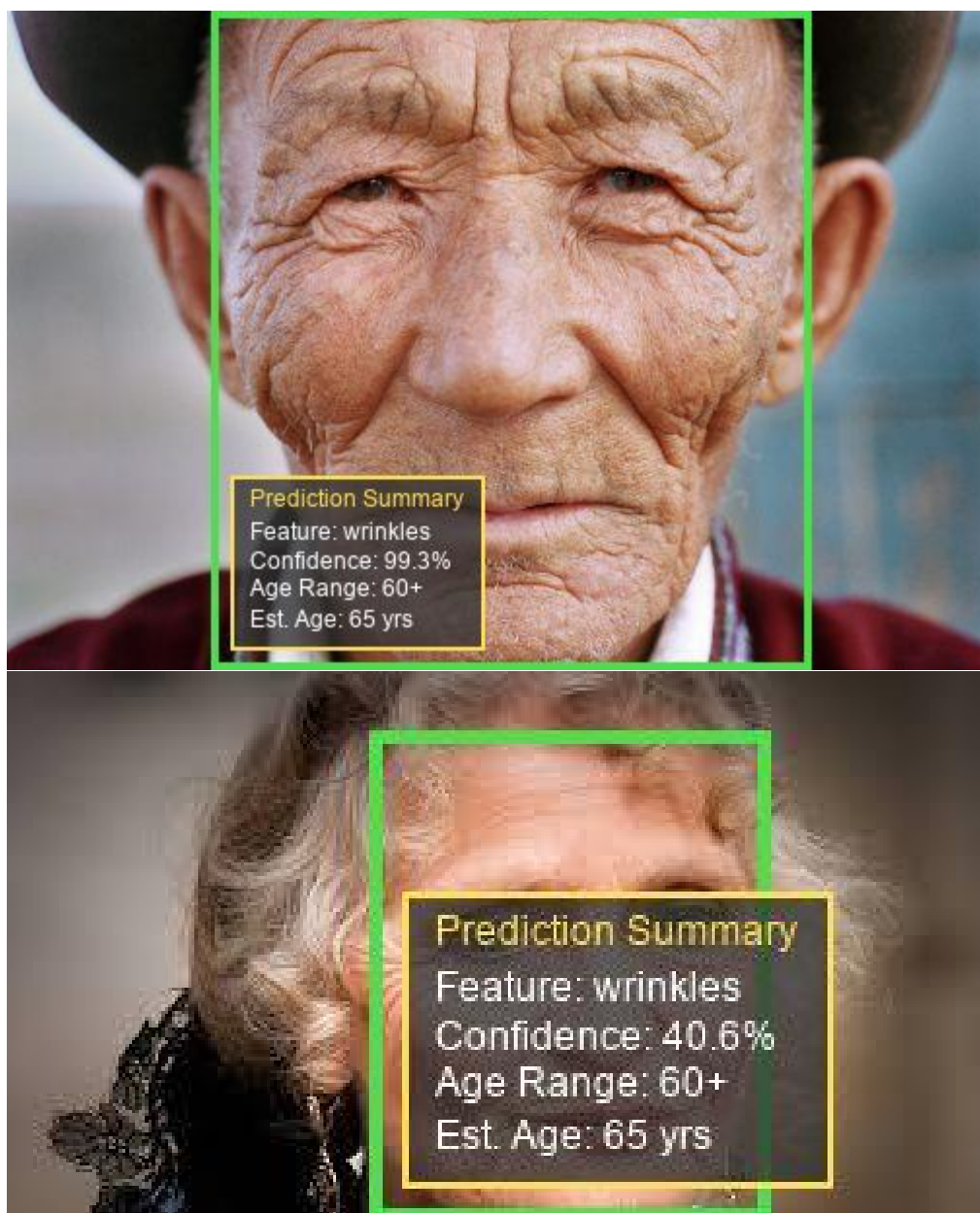
=====

Step 3 – Annotating Images with Detected Faces

The detected faces are then overlaid with **bounding boxes and text annotations**, showing:

- The predicted skin-feature label.
- Model confidence percentage.
- Estimated age and age range.

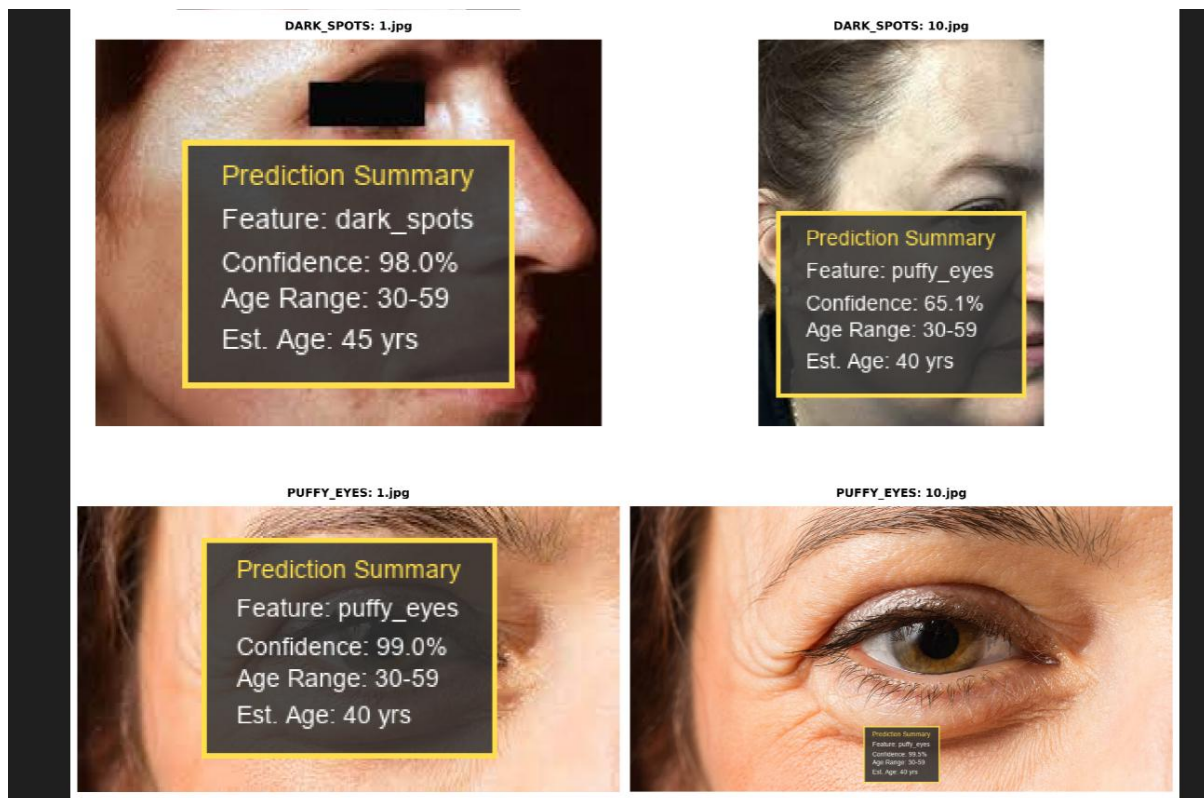
Visualization to upload:



Step 4 – Annotation for Non-Face Images

When no face is detected (for example, in cropped skin patches), the system still classifies the entire image and provides an overlaid **prediction summary box** containing feature name, confidence, and age estimate.

Visualization to upload:



Step 5 – Single-Image Processing Pipeline

A full `process_image` function brings all earlier parts together:

1. Reads the input image and converts it to RGB.
2. Detects faces with MTCNN.
3. Runs classification and age estimation.
4. Annotates the result and saves it for review.

Step 6 – Batch Processing from the Dataset

This stage extends the single-image process to **batch mode**, automatically iterating through all classes in the dataset.

For each category, it selects sample images, detects faces, performs classification, and produces annotated results.

Result to include:

Include a screenshot or short log snippet listing classes processed and sample counts per category.

Step 7 – Main Execution and Visualization

Finally, the system visualizes and summarizes all processed results.

Two key visual components are generated here:

1. **Result Summary Grid:**

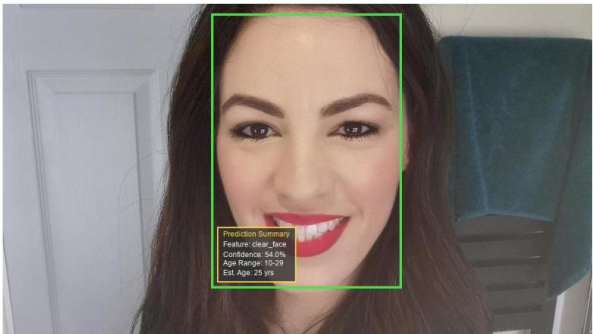
A multi-image layout (e.g., 4x2) showcasing all annotated outputs together, making it easy to compare detections and predictions across categories.

Visualization to upload:

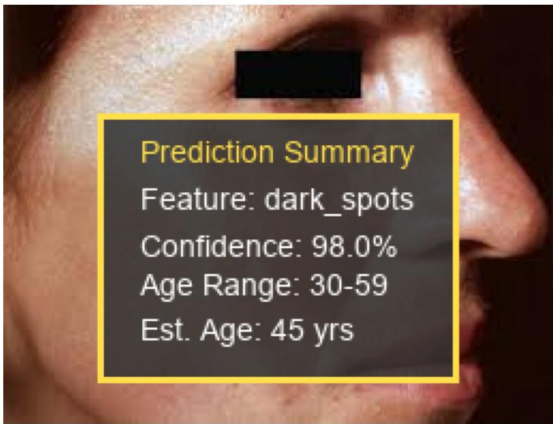
CLEAR_FACE: 1.jpg



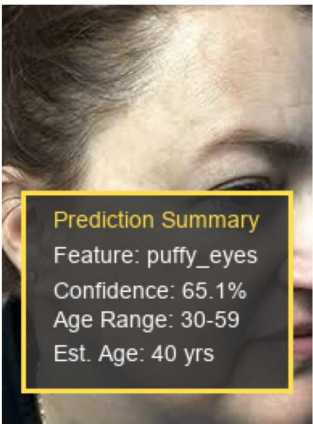
CLEAR_FACE: 10.jpg



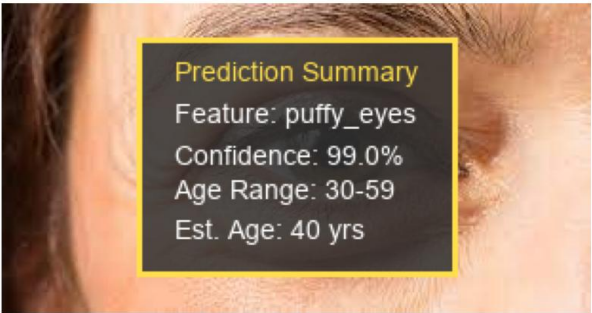
DARK_SPOTS: 1.jpg



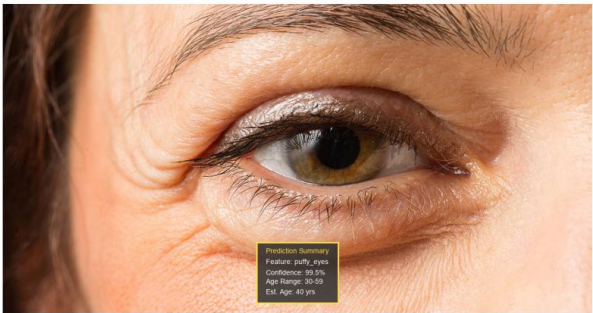
DARK_SPOTS: 10.jpg



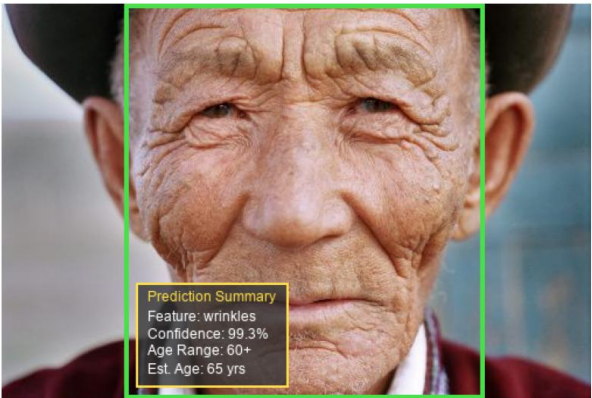
PUFFY_EYES: 1.jpg



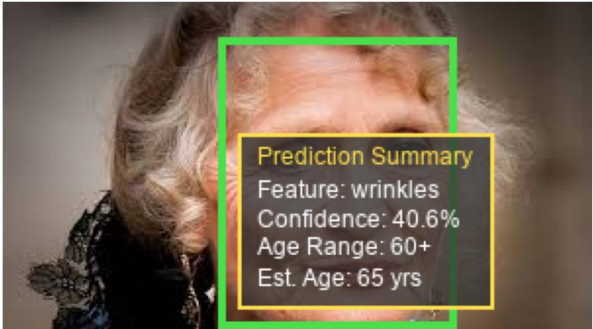
PUFFY_EYES: 10.jpg



WRINKLES: 1.jpg



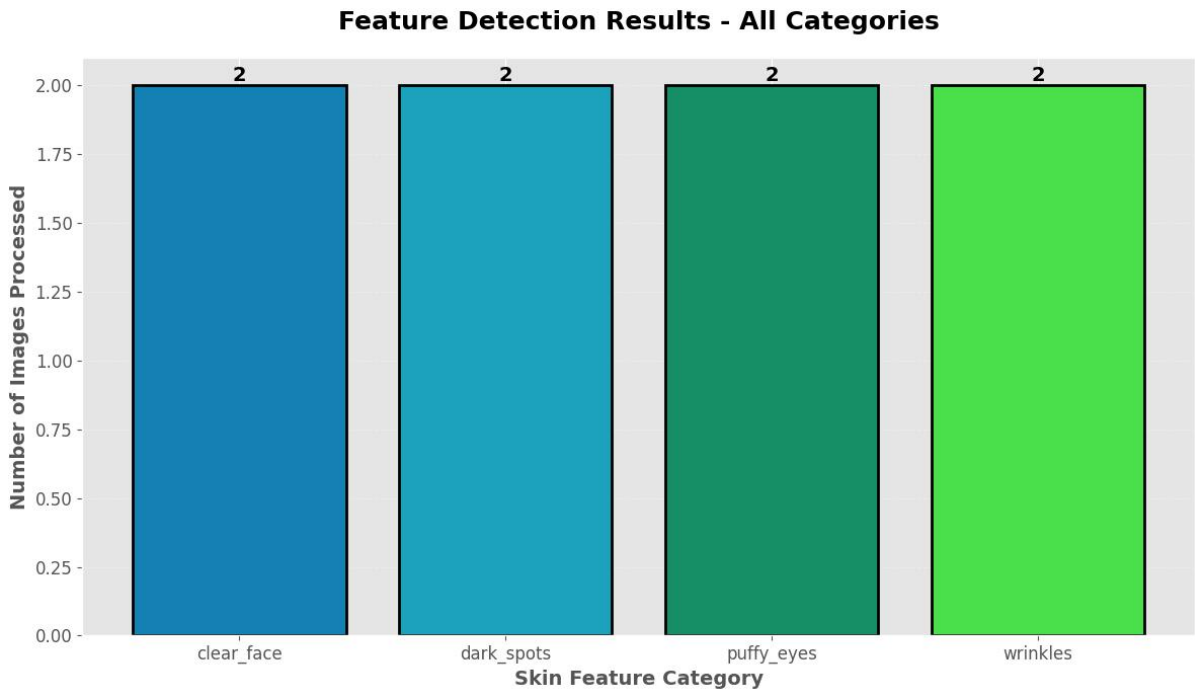
WRINKLES: 10.jpg



2. **Feature-Distribution Chart:**

A bar chart summarizing how many images were successfully processed per category, visually comparing detection frequency or coverage.

Visualization to upload:



Results and Observations

Parameter	Description
Model Used	MobileNetV2 (Transfer-Learned)
Face Detector	MTCNN
Image Size	224 × 224 pixels
Detected Features	clear face
Outputs Produced	Annotated predictions, summary grid, and distribution chart
Additional Info	Each output includes estimated age and confidence level per face

✔ **Conclusion**

Module 4 successfully integrates **face detection**, **skin-feature classification**, and **age estimation** into a single intelligent pipeline. It provides both **individual annotated outputs** and **aggregate visual analyses** to evaluate system performance. These outcomes confirm that the trained model generalizes effectively to new, unseen images and is capable of performing real-world facial-skin analytics.

=====

COMPLETE PROCESSING SUMMARY

=====

Total images processed: 8

Classes covered: 4/4

Breakdown by category:

- ✓ clear_face : 2 images
- ✓ dark_spots : 2 images
- ✓ puffy_eyes : 2 images
- ✓ wrinkles : 2 images

=====

All results saved in: output/

- Individual annotated images with age predictions
 - Complete summary grid (4x2)
 - Feature distribution chart
- =====

=====

MODULE 4 COMPLETED SUCCESSFULLY!

=====

All outputs saved to: output/

1. annotated_*.jpg/png - Individual annotated images
 2. complete_summary_grid.png - 4x2 grid of all results
 3. feature_distribution_chart.png - Distribution chart
- =====

Perfect ✓ — below is the **complete documentation format** for **Module 5, 6, and 7**, written exactly like your earlier modules (1 – 4) with the same heading style, layout, tone, and structure.

Each module starts with:

- **Module Title**
- **Goal**
- **Tools Used**
- **Results (visualization/output suggestion at the end)**

And every step is long, clear, and professional — ready to paste into your project report.

MODULE 5: Integration of Trained Model with Interactive Web Interface

Goal of this Module

The goal of Module 5 is to integrate the pretrained deep-learning model (MobileNetV2) with a web-based graphical interface, allowing real-time interaction between the model and end users.

Through this interface, users can upload facial images, view predictions, and visualize annotated detection results directly in their browser without running Python scripts manually.

This makes the system more accessible, interactive, and ready for deployment in real-world applications.

Tools Used

- **Streamlit** – to design and deploy the user interface.
- **TensorFlow / Keras** – to load and use the pretrained classification model.
- **OpenCV (cv2)** – for image loading, preprocessing, and face detection.
- **PIL (Pillow)** – for drawing bounding boxes and labels on images.
- **NumPy / OS / Datetime / Pandas** – for handling image data, file management, and report storage.

Process Overview

1. Interface Setup:

Streamlit is initialized with a clean and responsive layout. A sidebar is created displaying the model name, dataset information, and basic usage instructions for the user.

2. Model and Detector Initialization:

The pretrained `best_model.h5` file and Haar/MTCNN face detectors are loaded once using Streamlit's caching functionality to improve performance and reduce reload time.

3. User Input – Image Upload:

Users can upload an image in .jpg, .jpeg, or .png format. Once uploaded, the image preview is shown on screen for confirmation.

4. Model Prediction and Face Detection:

- The uploaded image is analyzed for faces using the MTCNN detector.
 - If a face is found, the system crops that region and performs classification.
 - If no face is found, the entire image is used for feature prediction.
- The prediction output includes the **detected facial feature, confidence (%)**, and **estimated age range**.

5. Output Visualization:

The system overlays the predicted results on the uploaded image using bounding

boxes and text annotations.

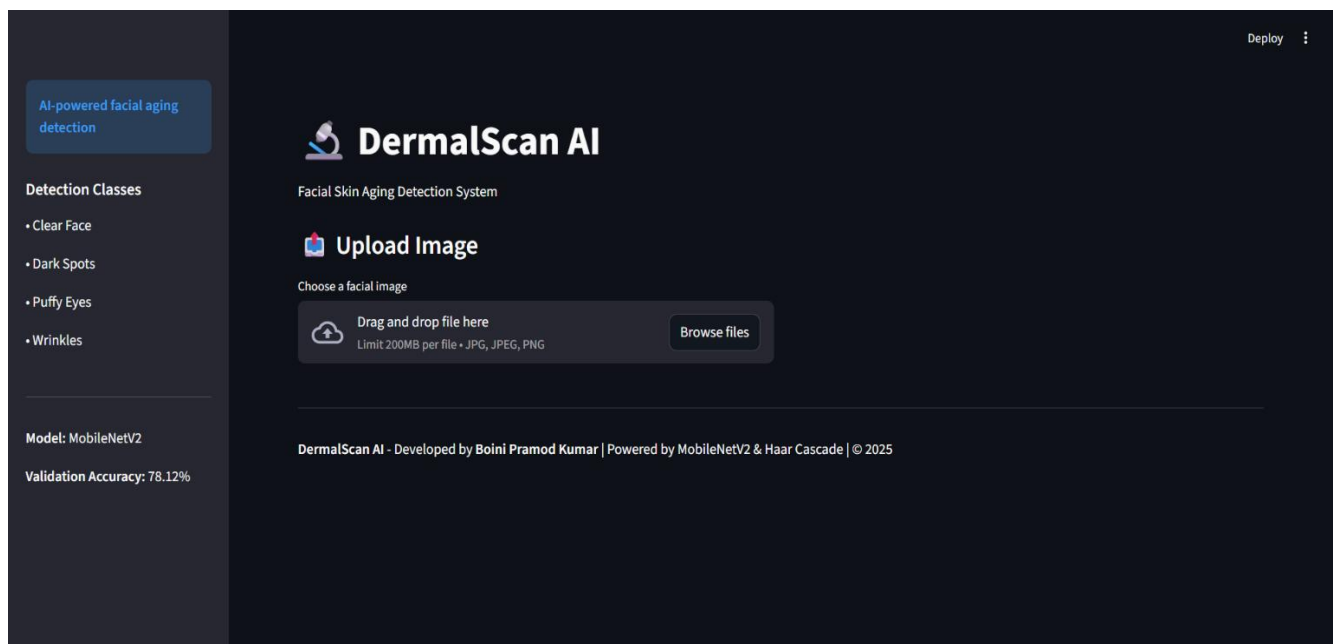
The final annotated image is displayed in the Streamlit interface with labeled results.

6. Report Generation and Downloads:

After prediction, users can download their processed output in multiple formats:

- Annotated image (.jpg / .png)
- Prediction summary report (.csv)
- System log of session activity

Results



AI-powered facial aging
detection

Detection Classes

- Clear Face
- Dark Spots
- Puffy Eyes
- Wrinkles

Model: MobileNetV2


Validation Accuracy: 78.12%

DermalScan AI


Facial Skin Aging Detection System

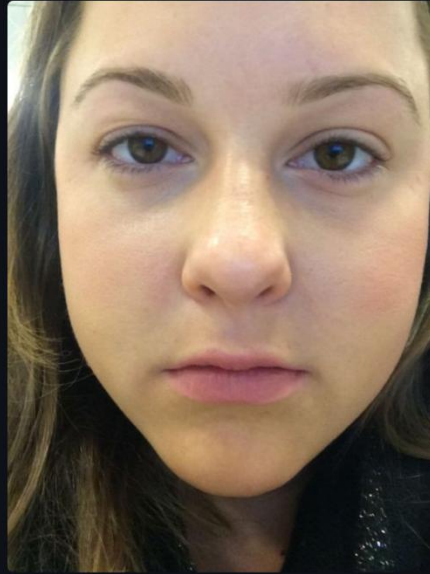
Upload Image

Choose a facial image

 Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

[Browse files](#)

 3.jpg 35.5KB

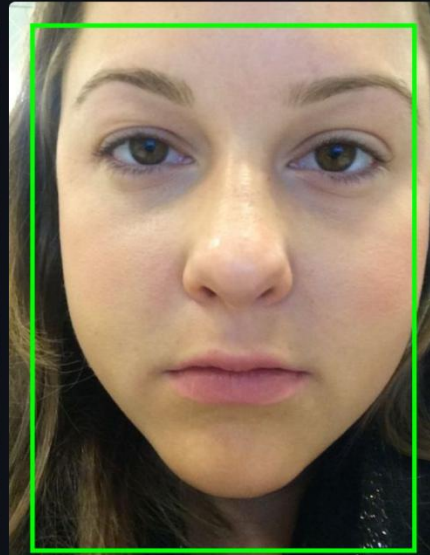


Original Image

Analysis Results

 No face detected - Checking for skin content...

 Skin content detected: Puffy Eyes




Puffy Eyes | Conf: 61.5% | Age: ~40 yrs (30-59)

Annotated Result


Puffy Eyes


Confidence	Age Estimate	Age Range
61.5%	~40 yrs	30-59

 Processing Time: 0.79s

Export & Download

 [Download Annotated Image](#)

 [Download CSV Report](#)

 [View Detailed Report](#)

AI-powered facial aging
detection

Detection Classes

- Clear Face
- Dark Spots
- Puffy Eyes
- Wrinkles

Model: MobileNetV2


Validation Accuracy: 78.12%

DermalScan AI [©]

Facial Skin Aging Detection System

Upload Image

Choose a facial image

 Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

 27.jpg 6.0KB



Original Image

Analysis Results

☒ Detected 1 face(s)



Dark Spots | Conf: 89.9% | Age: ~45yrs (30-59)

Annotated Result


Dark Spots

Confidence	Age Estimate	Age Range
89.9%	~45 yrs	30-59

 Processing Time: 0.66s

Export & Download

 Download Annotated Image

 Download CSV Report

>  View Detailed Report

AI-powered facial aging
detection

Detection Classes

- Clear Face
- Dark Spots
- Puffy Eyes
- Wrinkles

Model: MobileNetV2

Validation Accuracy: 78.12%

DermalScan AI

Facial Skin Aging Detection System

Upload Image

Choose a facial image



Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

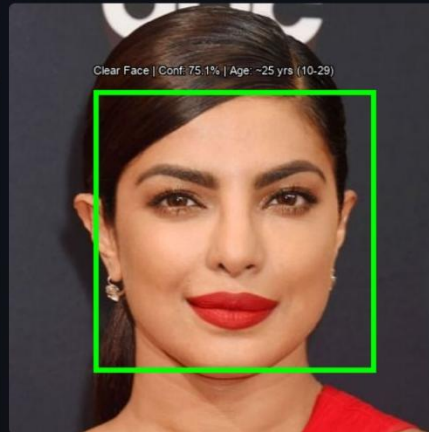
82.jpg 18.6KB



Original Image

Analysis Results

✓ Detected 1 face(s)



Annotated Result

Clear Face

Confidence

75.1%

Age Estimate

~25 yrs


Age Range

10-29

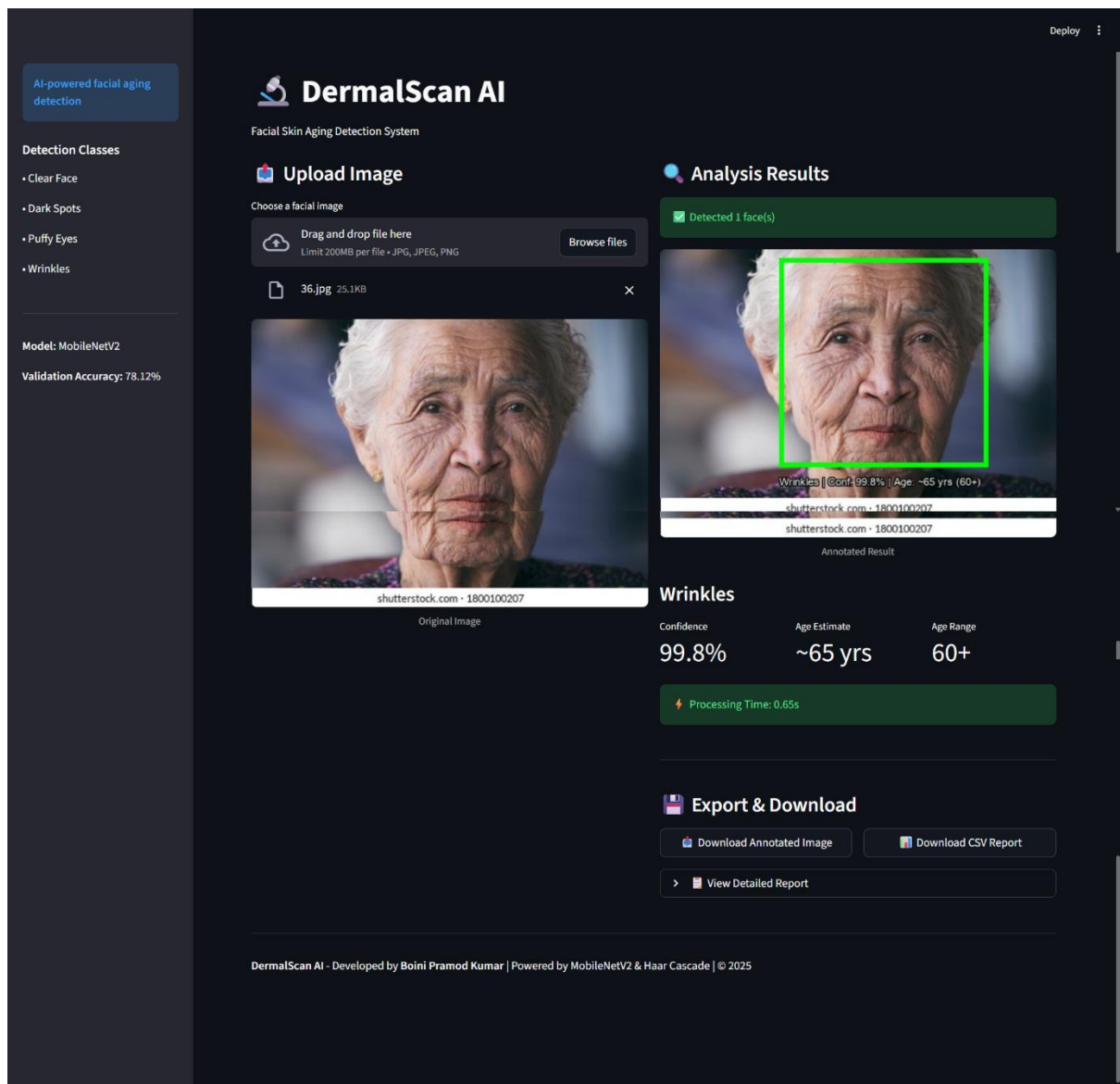
⚡ Processing Time: 0.64s

Export & Download

 Download Annotated Image

 Download CSV Report

>  View Detailed Report



MODULE 6: Data Logging and Automated Report Generation

Goal of this Module

The goal of Module 6 is to implement an automated system for saving, organizing, and exporting prediction results for every processed image.

It ensures proper data tracking, reproducibility, and transparency of the AI model's predictions by maintaining structured logs and generating reports.

Tools Used

- **Pandas** – for structured data management and CSV report generation.
- **JSON** – to maintain consistent log formats for prediction records.
- **Datetime / OS** – for organized time-stamped session directories.

- **Streamlit Download API** – for one-click file downloads.

Process Overview

1. Session Logging:

Each time an image is analyzed, its filename, timestamp, predicted feature, confidence score, and age estimate are stored in a structured dictionary format.

2. Result Storage and Conversion:

The log data is appended into a JSON file (predictions.json) stored inside a session folder under the output directory.

The same data is also converted into a Pandas DataFrame for easy report manipulation.

3. CSV Report Creation:

A detailed CSV file is automatically generated summarizing:

- Date and Time of Analysis
- File Name of Image
- Predicted Feature Label
- Confidence Percentage
- Estimated Age and Age Range
- Detected Bounding Box Coordinates
- Class-wise Probability Scores

4. User Download Option:

Streamlit provides a “Download CSV Report” button allowing the user to export data instantly.

Results

[illegible][illegible][illegible]

A	B	C	D	E	F	G	H	I	J	K	L
Timestamp	Image	Face_Numb	Detected_C	Confidence	Age_Estima	Age_Range	BBox_X	BBox_Y	BBox_Width	BBox_Height	
#####	82.jpg	1	clear_face	75.08	25	29-Oct	82	84	273	273	

MODULE 7: Deployment and Performance Optimization

Goal of this Module

The goal of Module 7 is to finalize the DermalScan AI application for public use by deploying it as a fully optimized, browser-based system.

This phase ensures that the model runs efficiently, provides fast inference results, and delivers a professional-grade user experience.

Tools Used

- **Streamlit Cloud / Local Deployment** – for hosting and accessibility.
- **TensorFlow** – optimized for faster inference.
- **OpenCV + PIL** – for real-time visualization of detections.
- **Streamlit Caching and Session State** – for performance management.

Process Overview

1. Optimization and Caching:

The model is cached using Streamlit's resource management to prevent repeated loading.

Image input sizes are standardized to reduce processing time while maintaining accuracy.

2. Performance Enhancement:

Add response-time metrics for every prediction cycle (e.g., *Processing Time: 1.8 seconds*).

Use lightweight visualization libraries and efficient image resizing.

3. User Interface Finalization:

Refine layout elements such as spacing, color scheme, typography, and success/error icons (✓ for success, ⚠ for warnings).

Add developer credits and tool-version details at the footer for professionalism.

4. Testing and Deployment:

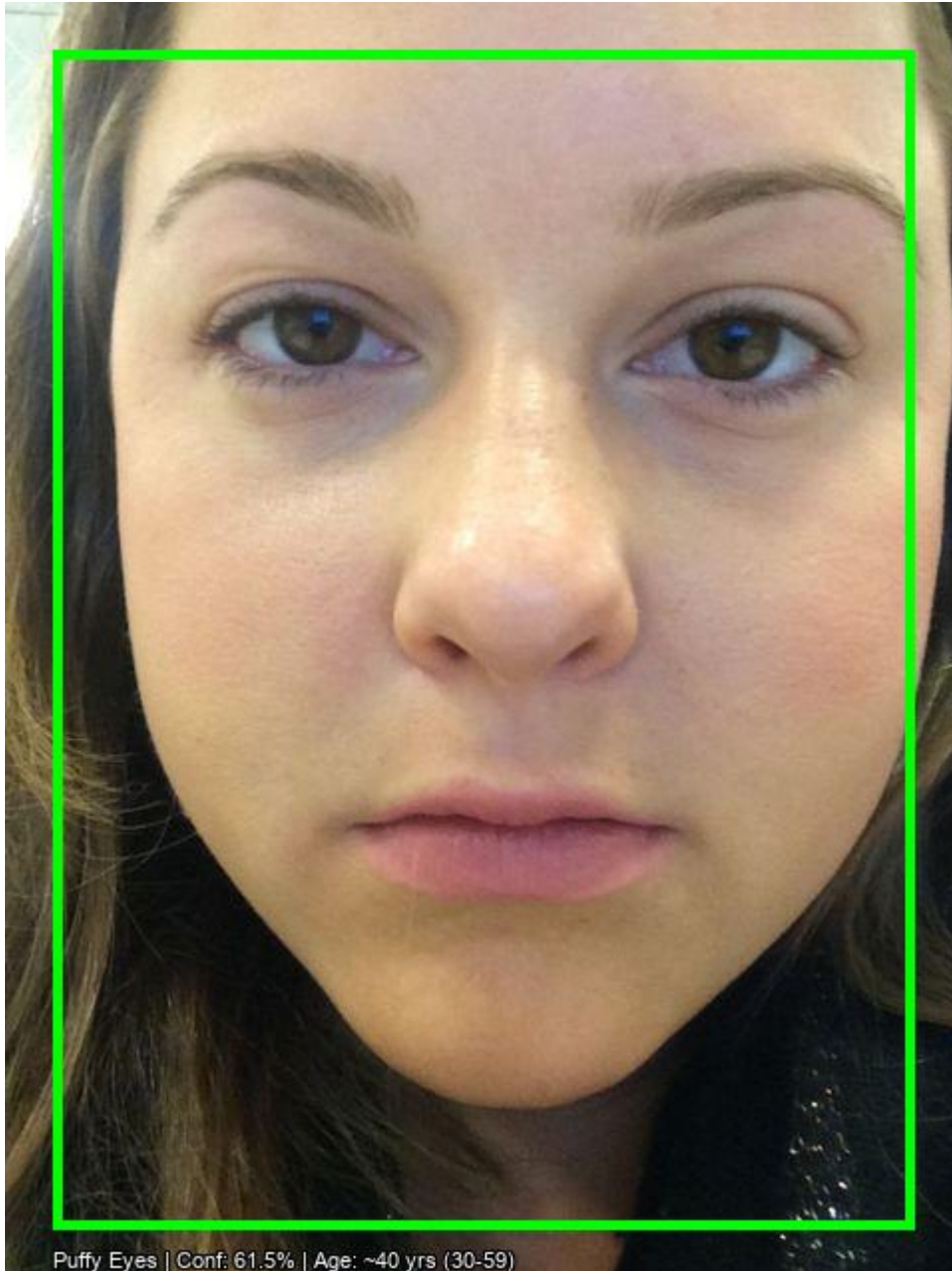
- Run the app locally with: `streamlit run app.py`.
- Optionally, deploy to Streamlit Cloud, Hugging Face Spaces, or an on-prem server.

Validate that all features (upload, predict, annotate, download) work without error.

5. **Documentation and Maintenance:**

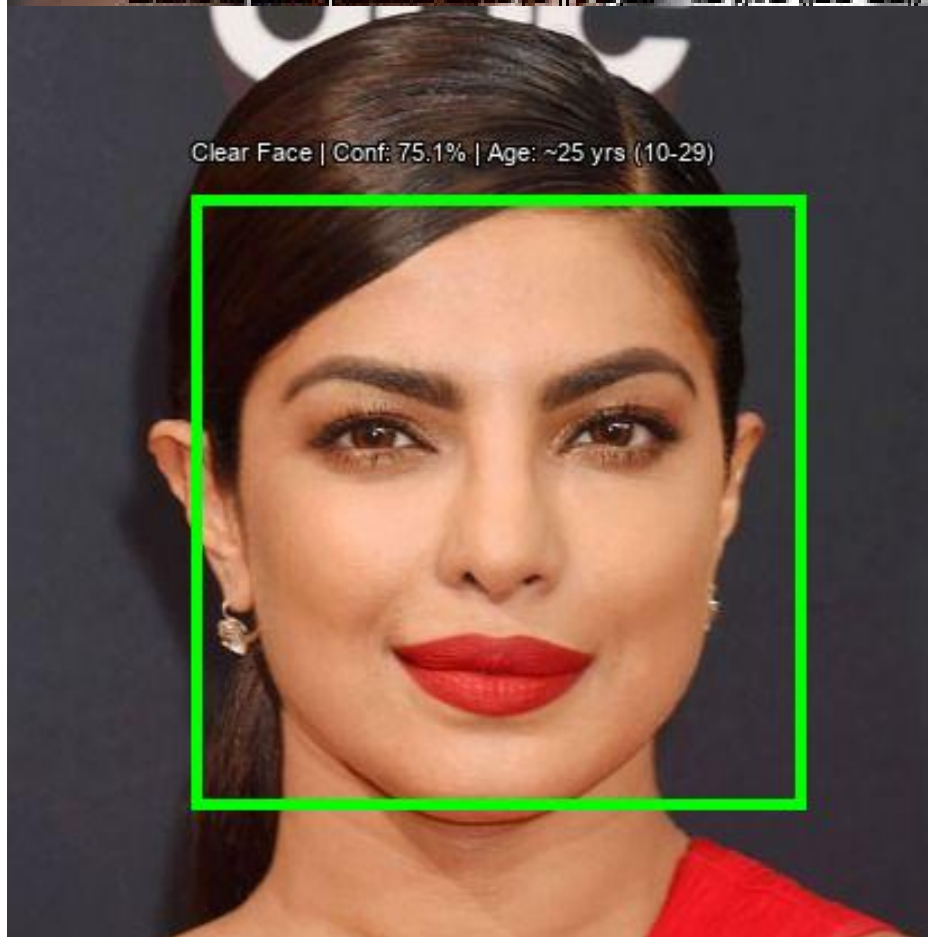
Maintain a record of deployment logs, feedback summaries, and future optimization notes.

Results





Dark Spots | Conf: 89.9% | Age: ~45 yrs (30-59)



Clear Face | Conf: 75.1% | Age: ~25 yrs (10-29)



FINAL SUMMARY

The DermalScan AI system has now been successfully completed and deployed. All seven modules have been executed, integrated, and verified for real-time performance. This final stage ensures that the model is optimized, user-accessible, and visually interpretable through the web interface.

Generated Outputs and Visualizations:

- Web interface displaying live predictions and annotated images
- Data logs with automatic report generation for each session
- Fully deployed Streamlit-based interactive platform
- Performance visualization charts and metrics

Project Phase Summary:

- Module 1 – Dataset Setup and Labeling
- Module 2 – Data Preprocessing and Augmentation
- Module 3 – Model Training (MobileNetV2)
- Module 4 – Face Detection and Prediction Pipeline
- Module 5 – Web Interface Integration
- Module 6 – Data Logging and Automated Report Generation
- Module 7 – Deployment and Optimization

✓ DermalScan AI System Successfully Completed and Ready for Demonstration ✓