INFOSYS SPRINGBOARD VIRTUAL INTERNSHIP 6.0

ARTIFICIAL INTELLIENCE DOMAIN

PROJECT DOCUMENTATION

DERMALSCAN: AI_FACIAL SKIN AGING DETECTION APP

Submitted By

S. Kanishka

Under the Guidance of Mentor

Mr. Praveen Bhargav

INTRODUCTION

Problem Statement

Facial aging indicators such as wrinkles, dark spots, puffy eyes, and overall skin clarity are important in understanding skin health, yet these features are difficult to assess consistently through manual observation. Variations in lighting, skin tone, facial expressions, and image quality make the visual evaluation of aging signs unreliable and time-consuming. Currently, there is no automated system that can accurately identify and categorize multiple facial aging signs from a single image in a standardized way. This creates a gap for applications in skincare analysis, cosmetic recommendations, and dermatology support, where consistent and objective assessment of facial aging is essential.

Objective of the Project

The project focuses on developing an automated deep learning–based system capable of detecting and classifying facial aging signs such as wrinkles, dark spots, puffy eyes, and clear skin. The system will utilize a pretrained EfficientNetB0 model for feature extraction and classification, combined with Haar Cascade–based face detection for accurate localization of facial regions. The workflow includes custom preprocessing techniques, data augmentation for improved generalization, and percentage-based prediction outputs for each identified aging sign. A user-friendly web interface will be developed to allow users to upload facial images and receive annotated results with bounding boxes, labels, and confidence scores. The solution aims to provide consistent, objective, and real-time analysis of facial aging indicators.

System Requirements

Python Version

Recommended: **Python 3.10** [This version is used in this project]
(Supports TensorFlow 2.15 and the listed libraries without compatibility issues)

Milestone 1 of the Project

Module 1: Dataset Setup and Image Labelling

In this module, the dataset is prepared and organized for further processing. The dataset is downloaded and stored locally within the project directory, ensuring easy access during model development. To handle the dataset structure, **Python's os module** is used, which allows directory traversal, file listing, and automated reading of class folders.

Each class in the dataset represents a specific facial aging category. The project includes four primary classes:

- Clear Skin
- Dark Spots
- Puffy Eyes
- Wrinkles

Why this module is important in Deep Learning?

Proper **categorical labelling** of the dataset into these classes is crucial because deep learning models rely on accurately labelled data to learn distinguishing features for each category. **Mislabelled or mixed images can lead to confusion during training and result in poor classification performance.** By organizing images into separate folders corresponding to each facial aging sign, the model can effectively learn the unique characteristics of each class.

After organizing and labelling the dataset, it is essential to ensure **class balance** by checking the number of samples in each category. Imbalanced datasets can lead to **model bias**, where the network disproportionately favors classes with more examples, reducing generalization and prediction accuracy for underrepresented categories. To evaluate this, a **class distribution analysis** is performed, often visualized using bar charts or histograms, providing a clear overview of sample counts per class.

Based on this analysis, techniques such as **data augmentation**, **oversampling**, or **stratified sampling** may be applied to achieve a **balanced dataset**. Ensuring proper labelling, organization, and balance forms a solid foundation for subsequent **preprocessing, feature extraction, and deep learning model training**, ultimately improving the accuracy and robustness of the facial aging classification system.

To create a bar graph of the class distribution:

```
plt.bar(labels,counts)

plt.title("Class Distribution Plot: Number of Images per Class")

plt.show()
```

The variable labels specify the classes and variable count specifies the number of images.

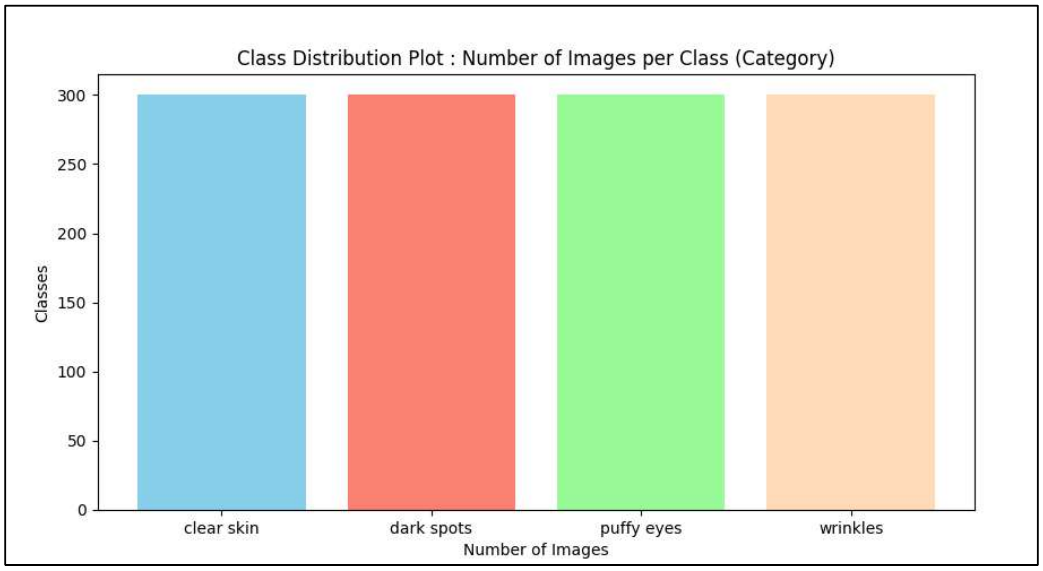

Figure 1: Class Distribution Plot of the Dataset

Module 2: Image Preprocessing and Augmentation

In this module, all images in the dataset are prepared for ingestion by the deep learning model. Since neural networks operate on standardized input dimensions, each image is resized to 224×224 pixels. Following resizing, images are normalized to scale pixel values to a consistent numerical range, typically between 0 and 1. This normalization stabilizes gradient updates and enhances the training efficiency of the model.

img_size = (224, 224)

rescale=1./255

To improve the dataset's robustness and prevent overfitting, **image augmentation** techniques are applied. Augmentation introduces controlled **variations such as horizontal flipping, rotation, and zoom transformations.** These variations simulate real-world changes in facial orientation and lighting, enabling the model to learn more generalized features. Each augmentation preserves essential facial characteristics, ensuring that class-specific features like wrinkles or dark spots remain identifiable.

The dataset is split into **training** and **validation** sets to ensure the model learns from one portion of the data while being evaluated on unseen samples. This helps monitor overfitting and improves the model's ability to generalize. 80% of data is for training and 20% of data is for validation.

In addition to preprocessing, the categorical labels of the dataset are converted into **one-hot encoded vectors**. This encoding transforms each class label into a binary vector representation suitable for multi-class classification, enabling the model to process labels mathematically during training.

Creation of a generator object using the ImageDataGenerator class from the tensorflow.keras.preprocessing.image module. This object is defined with all the parameters for performing the data augmentation.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
```

Performing the data augmentation on the actual dataset and applying one-hot encoding on the dataset

```
train_generator = train_datagen.flow_from_directory(
    dataset,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',  #One-hot encoding
    shuffle=True,
    subset='training'
)
```

This module results in a **fully pre-processed, augmented, and model-ready dataset**. It ensures that images maintain consistent dimensions, classes are represented with increased diversity, and the dataset becomes more resilient to variations, ultimately enhancing the performance and generalization of the final deep learning model.

Additionally, a preview grid of 16 augmented images and a class-wise augmentation scattering plot was also generated to confirm that each batch maintains a balanced distribution of augmented samples across all four classes, ensuring consistent representation during training.

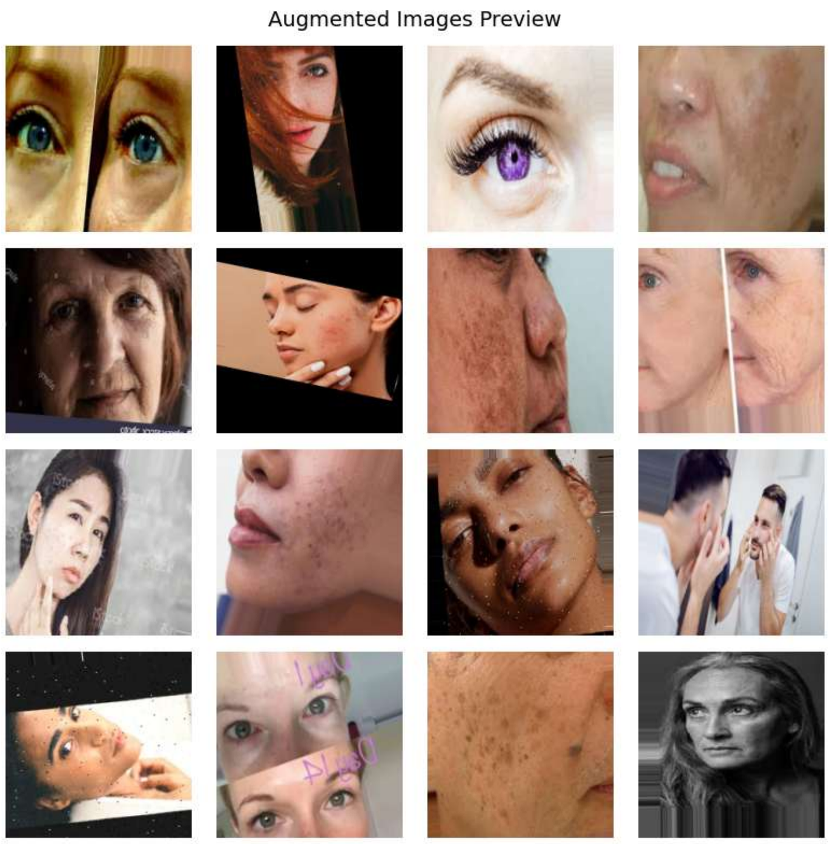

Figure 2: A Preview of the Augmented Images (16 images)

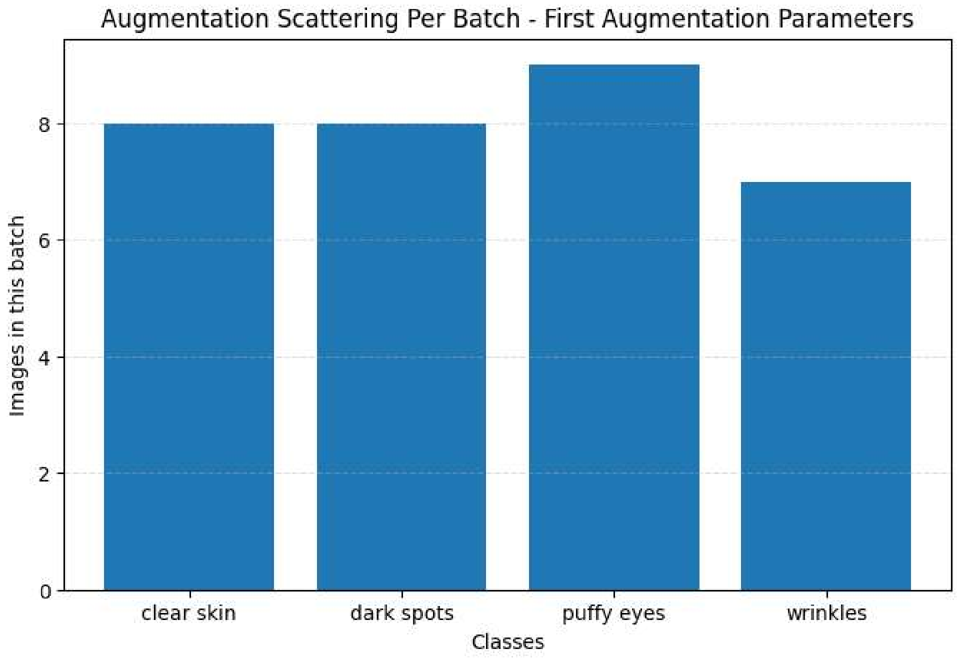

Figure 3: Class-Wise Augmentation Scattering Plot

Milestone 2 of the Project

Module 3: Model Training and Evaluation

Objective

The objective of Module 3 is to train and evaluate deep learning models for **skin condition detection and classification** using transfer learning, and to select the most reliable model based on accuracy and consistency.

Models Explored

In this module, multiple pretrained CNN architectures were experimented with to understand performance differences:

- EfficientNetB0
- ResNet50
- VGG16
- MobileNetV2

All models were initialized with ImageNet pretrained weights, and a custom classification head was added for a four class skin condition problem:

- Clear Skin
- Dark Spots
- Puffy Eyes
- Wrinkles

Training Setup

- **Loss Function:** Categorical Cross-Entropy
- **Optimizer:** Adam
- **Activation Function:** Softmax
- **Input Size:** 224 × 224
- **The dataset was split into:**
    - Training set
    - Validation set (15%)
    - Test set (15%)

- **Callbacks:**
    - EarlyStopping based on Validation Accuracy
    - ModelCheckpoint
    - ReduceLROnPlateau

```
x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(256, activation="relu")(x)

x = Dropout(0.5)(x)

outputs = Dense(NUM_CLASSES, activation="softmax")(x)

model = Model(inputs=base_model.input, outputs=outputs)

model.compile(

optimizer=Adam(learning_rate=1e-4),

loss="categorical_crossentropy",

metrics=["accuracy"])
```

Model Performance Comparison

EfficientNetB0

- Training Accuracy: 28.02%
- Validation Accuracy: 35.42%

Observation:
EfficientNetB0 failed to learn meaningful features from the dataset. Both training and validation accuracies remained low, indicating underfitting.

VGG16

- Training Accuracy: 34.48%
- Validation Accuracy: 54.58%

Observation:
VGG16 showed improvement over EfficientNetB0 but still struggled to generalize. The gap between train and validation accuracy indicates limited feature extraction capacity for this dataset size.

ResNet50

- Training Accuracy: 80.94%
- Validation Accuracy: 78.33%

Observation:
ResNet50 learned strong representations and achieved good validation accuracy. However, the difference between training and validation curves suggested mild overfitting.

MobileNetV2 (Selected Model)

MobileNetV2 consistently outperformed other models in terms of:

- Accuracy
- Stability of validation curve
- Generalization capability

Among all evaluated models, MobileNetV2 demonstrated the best balance between training and validation performance. The second MobileNetV2 model was selected based on improved validation accuracy, stable loss curves, and a reduced gap between training and validation metrics, indicating better generalization compared to other models. Final model is saved as a '**.h5'** file.

Model	Training Accuracy	Validation Accuracy	Overall Performance
EfficientNetB0	28.02%	35.42%	Poor
VGG16	34.48%	54.58%	Moderate
ResNet50 (Fine-tuned)	80.94%	78.33%	Good
MobileNetV2 Model 2 (Final)	92.98%	86.67%	Very Good

Table 1: Transfer Learning Models Performance Comparison

MobileNetV2 – Final Models (Model 1 vs Model2)

Model 1: MobileNetV2_best1_notebook2.h5 **(renamed as MobileNetV2_Model1.h5)**

- Training Accuracy (recomputed): 94.52%
- Validation Accuracy (recomputed): 85.00%

This model showed strong performance but a larger gap between training and validation accuracy.

Model 2: MobileNetV2_best_notebook2.h5 **(renamed as MobileNetV2_Model2_Final.h5)**

- Training Accuracy (from history): 92.98%
- Validation Accuracy (from history): 86.67%

This model is selected as the final model to use in the face prediction and classification.

Base Model

```
base_model = MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)
```

For fine-tuning last 30 layers are not frozen and are made trainable

```
base_model.trainable = False
#Freeze all except last 30 layers
for layer in base_model.layers[:-30]:
    layer.trainable = True
```

Why Model 2 is chosen as the final model?

Both **MobileNetV2 Model 1** and **MobileNetV2 Model 2** were evaluated using multiple performance metrics, including **Precision**, **Recall**, **F1-Score**, and **Confusion Matrix** analysis.

While both models demonstrated strong classification capability, Model 2 consistently showed better generalization performance. This conclusion was based on the following observations:

- Higher and more stable validation accuracy compared to Model 1
- Smaller gap between training and validation accuracy, indicating reduced overfitting
- Better balance across classes observed in the confusion matrix, with fewer misclassifications
- Improved Precision, Recall, and F1-scores for most skin condition classes in the classification report

These evaluation results indicate that Model 2 not only learns the training data effectively but also performs reliably on unseen data. Therefore, **MobileNetV2 Model 2 is selected as the final model** to proceed with the skin condition prediction pipeline in Module 4.

Confusion Matrix and Classification Report of MobileNetV2 Models

Class	Precision	Recall	F1-Score	Support
Clear Skin	0.90	0.96	0.92	45
Dark Spots	0.85	0.87	0.86	45
Puffy Eyes	0.85	0.76	0.80	45
Wrinkles	0.74	0.76	0.75	45
Accuracy	—	—	0.83	180
Macro Avg	0.83	0.83	0.83	180
Weighted Avg	0.83	0.83	0.83	180

Table 2: Classification Report – Model 1

Class	Precision	Recall	F1-Score	Support
Clear Skin	0.96	0.96	0.96	45
Dark Spots	0.91	0.89	0.90	45
Puffy Eyes	0.85	0.78	0.81	45
Wrinkles	0.76	0.84	0.80	45
Accuracy	—	—	0.87	180
Macro Avg	0.87	0.87	0.87	180
Weighted Avg	0.87	0.87	0.87	180

Table 3: Classification Report – Model 2

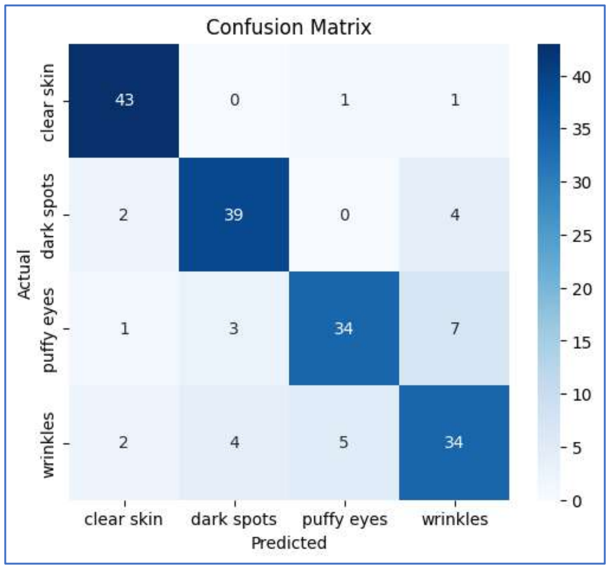

Figure 4: Confusion Matrix – Model 1

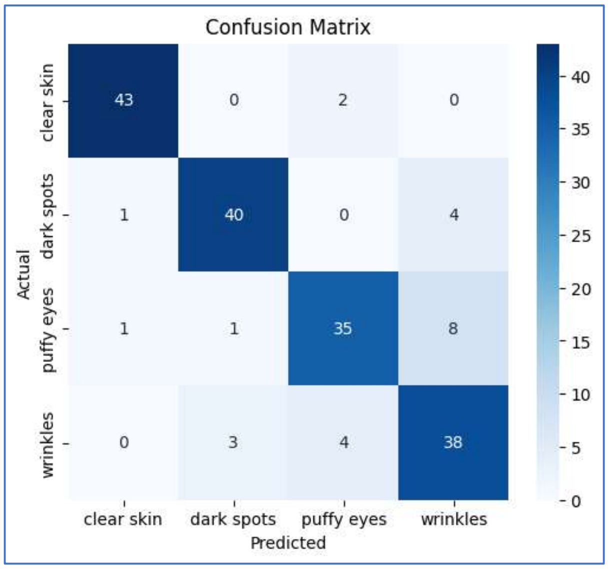

Figure 5: Confusion Matrix – Model 2

Accuracy and Loss Curves – MobileNetV2 Model 2 (Final Model)

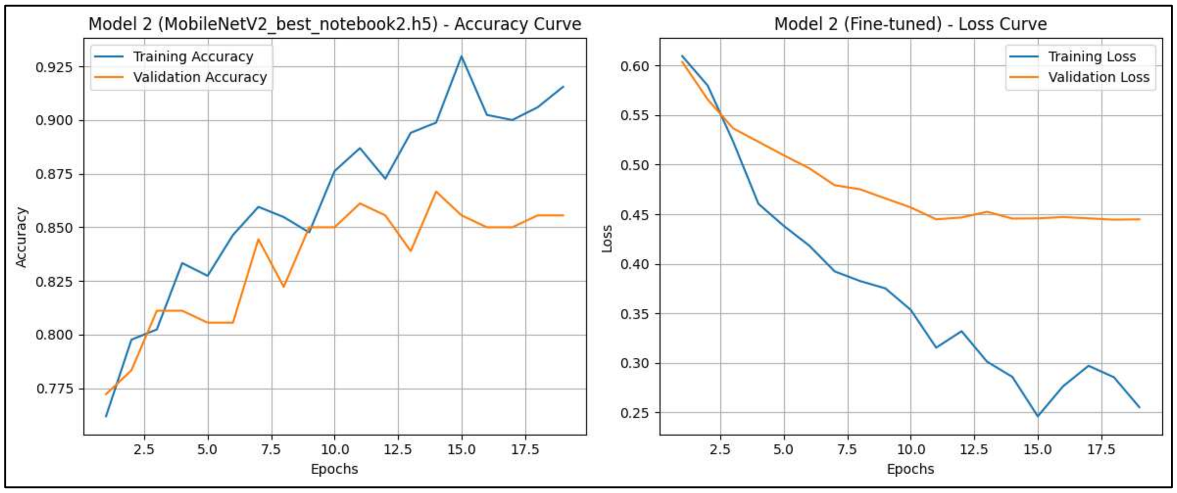

Figure 6: Accuracy and Loss Curves of the Final Model (MobileNetV2_Model2_Final.h5)

<u>Module 4: Face Detection and Prediction Pipeline</u>

This module focuses on building an end-to-end inference pipeline that integrates face detection with the final trained skin condition classification model.
Given an input image, the pipeline detects human faces, extracts the facial region, applies the trained CNN model to classify skin conditions, and displays the predicted results visually.

In addition, a heuristic-based age prediction mechanism is incorporated to provide an estimated age value alongside the skin condition prediction.

Technologies Used

Component	**Technology**
Face Detection	OpenCV Haar Cascade
Image Processing	OpenCV
Model Inference	TensorFlow / Keras
Visualization	Matplotlib
Classification Model	MobileNetV2 (Fine-tuned)

Pipeline Architecture

The pipeline follows the steps below:

Image Input

- A single image is randomly selected from the test dataset. (15% of the dataset)

Face Detection

- OpenCV's Haar Cascade classifier is used to detect faces.

```
CASCADE_PATH = "haarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(CASCADE_PATH)
```

- Bounding box coordinates (x, y, w, h) are obtained for each detected face.

```
x, y, w, h = faces[0]
pad = 20
x = max(0, x - pad)
y = max(0, y - pad)
w = min(img.shape[1] - x, w + 2*pad)
 h = min(img.shape[0] - y, h + 2*pad)
```

Face Cropping & Preprocessing

- The detected face region is cropped.
- The cropped face is resized to 224 × 224.
- Pixel values are normalized to [0, 1].

Skin Condition Prediction

- The fine-tuned MobileNetV2 model predicts the skin condition.
- The class with the highest probability is selected.
- Confidence score is computed as a percentage.

```
face = img[y:y+h, x:x+w]
preds = model.predict(preprocess_face(face), verbose=0)[0]
idx = np.argmax(preds)
label = class_labels[idx]
confidence = preds[idx] * 100
```

Age Estimation (Heuristic-Based)

- A predicted age is computed using predefined age ranges for each skin condition class.
- The confidence score is used to interpolate within the age range.

Visualization

- A green bounding box is drawn around the detected face.
- Class label, predicted age, and confidence score are displayed above the bounding box.

```
cv2.rectangle( display_img, (x, y), (x+w, y+h), (0, 255, 0), 3)

text = f"{label} | Age: {predicted_age} | {confidence:.2f}%"

cv2.putText( display_img, text, (x, y - 15), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2 )
```

- The result image is displayed using Matplotlib.

Heuristic-Based Age Prediction Logic

Since the project does not use a dedicated age estimation model, a confidence-weighted heuristic is employed.

Age Range Mapping

Skin Condition	Age Range
Clear Skin	18 – 30
Dark Spots	25 – 45
Puffy Eyes	20 – 40
Wrinkles	40 – 55

Formula Used

```
min_age, max_age = age_ranges[label]

predicted_age = int(min_age + (confidence / 100) * (max_age - min_age))
```

How the age is derived?

- Each skin condition is mapped to a predefined age range (e.g., Wrinkles: 40-60).
- The predicted class determines the age range.

- The confidence score determines how deep into the range the prediction lies.
- Higher confidence shifts the predicted age closer to the upper bound of the range.
- The final output is a single integer age for display purposes, not a verified age.

Note: This is not biometric age estimation and not medically accurate. No facial landmarks, bone structure, or physiological aging features are used. If accurate age estimation is required, a separate dedicated age prediction model must be trained using labelled age datasets.

Visualization Output and Results

Each prediction output image includes:

- Bounding box covering the detected face
- Skin condition label
- Predicted age
- Confidence score (%)
- A final summary is also printed displaying the actual class of the image, predicted class, confidence score and the status of the prediction (Correct or Wrong)

Output Image

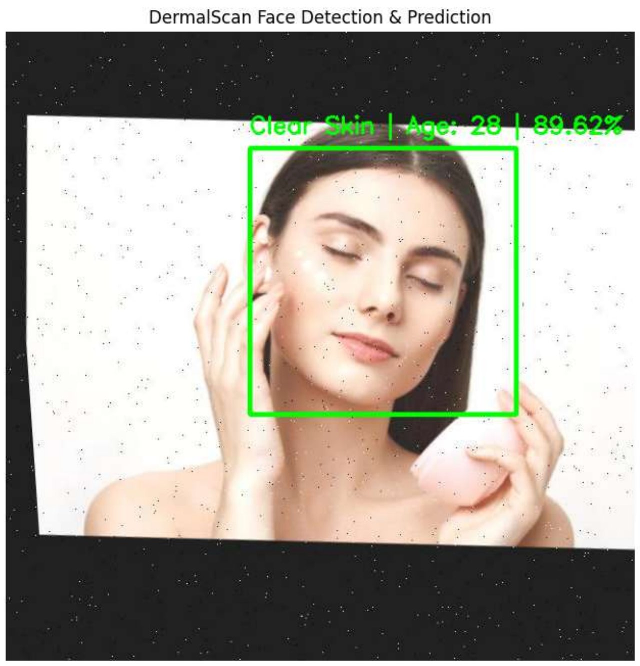
DermalScan Face Detection & Prediction
Clear Skin | Age: 28 | 89.62%

Figure 6: Face Detection and Prediction Output Image

Output Text

```
================ FINAL SUMMARY ================
Image Name    : clear_skin_226.jpg
Actual Class  : clear skin
Predicted     : Clear Skin
Confidence    : 89.62%
Predicted Age : 28
Status        : CORRECT
===============================================
```

Limitations

- Haar Cascade may fail under extreme lighting or non-frontal angles.
- Age prediction is heuristic-based, not learned from data.
- The pipeline processes one image at a time for demonstration purposes.

Key Terms

Categorical Labelling: The process of assigning images to distinct classes or categories to facilitate supervised learning.

Model Bias: A tendency of a machine learning model to favour certain classes over others, often caused by imbalanced training data.

Stratified Sampling: Selecting samples from each class proportionally to maintain the same distribution as the original dataset.

Image Resizing: Adjusting the dimensions of an image to a fixed size required by the model (e.g., 224×224 pixels).

Normalization: Scaling pixel intensity values to a standard range (e.g., 0–1) to improve model stability and convergence.

Image Augmentation: Applying transformations such as flipping, rotation, and zooming to artificially expand the dataset and reduce overfitting.

Overfitting: A scenario where a model performs well on training data but poorly on unseen data due to lack of generalization.

One-Hot Encoding: Converting class labels into binary vectors where each class has a unique position set to 1.

Transfer Learning: Using pretrained neural networks to speed up training and improve performance on a new task.

Pretrained Model: A model already trained on a large dataset (ImageNet) and reused for feature extraction.

Fine-Tuning: Retraining selected layers of a pretrained model to adapt it to the target dataset.

Categorical Cross-Entropy Loss: A loss function used for multi-class classification problems.

Adam Optimizer: An adaptive optimization algorithm used to update neural network weights efficiently.

Validation Accuracy: The percentage of correctly classified samples from the validation dataset during training.

Overfitting: A condition where the model performs well on training data but poorly on validation data.

Confusion Matrix: A table showing correct and incorrect predictions for each class.

Classification Report: A summary of precision, recall, F1-score, and support for each class.

Precision: The proportion of correctly predicted positive samples among all predicted positives.

Recall: The proportion of correctly predicted samples among all actual samples of a class.

F1-Score: A balanced metric combining precision and recall.

Haar Cascade Classifier: A traditional OpenCV-based method for detecting faces.

Confidence Score: The probability associated with a predicted class.

Heuristic-Based Age Estimation: A rule-based method for estimating age from prediction confidence and class.