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 model for feature extraction and classification, combined with DNN-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 and ensures stable compatibility with TensorFlow and all dependent libraries.)

Hardware Requirements

The system is designed to run on standard consumer hardware using CPU-based inference. No dedicated GPU is required. GPU acceleration, if available, can improve performance but is optional.

Storage: At least 5 GB free disk space (for model files, images, and logs)

Software Dependencies

- **Backend:**
  - Flask – Web framework
  - TensorFlow / Keras – Deep learning model inference
  - OpenCV (cv2) – Image processing and DNN face detection
  - NumPy – Numerical computations
  - Matplotlib – Image annotation and visualization
- **Frontend:**
  - HTML5 – Structure and layout
  - CSS3 – Styling and responsive design
  - JavaScript (Vanilla JS) – Client-side logic and API communication
  - Font Awesome – UI icons

Model & Assets

- Pre-trained CNN model file (.h5)
- OpenCV DNN face detection files:
  - deploy.prototxt (Download Link)
  - res10_300x300_ssd_iter_140000.caffemodel (Download Link)

Browser Requirements

- Google Chrome (Recommended)
- Mozilla Firefox
- Microsoft Edge

Additional Notes

- The application runs entirely on a local or hosted Flask server.
- No external APIs are required for inference.
- Internet connection is only needed for initial dependency installation or deployment.

Milestone 1 of the Project

<u>Module 1: Dataset Setup and Image Labelling</u>

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.

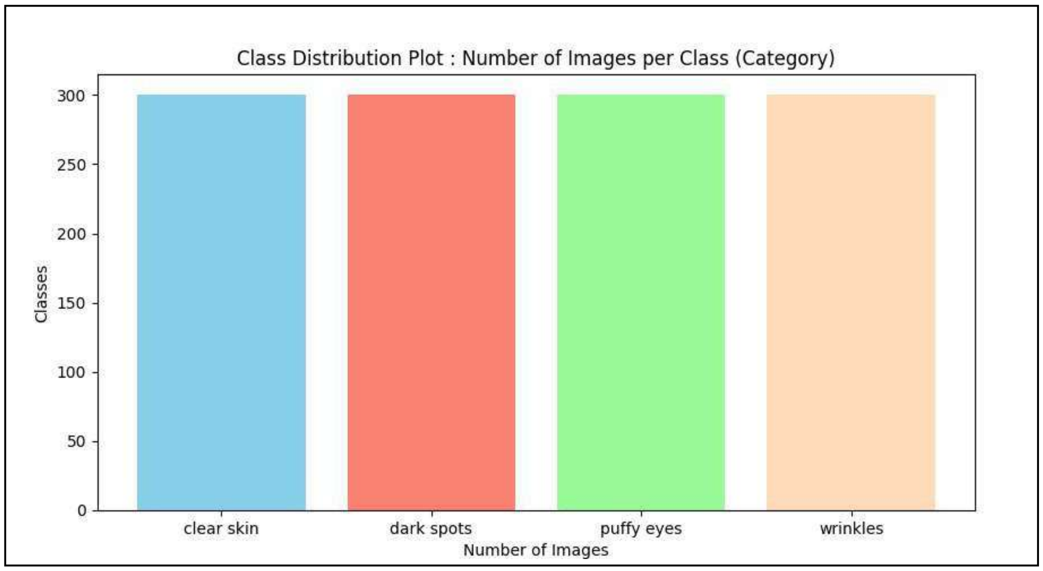


Figure 1: Class Distribution Plot of the Dataset

<u>Module 2: Image Preprocessing and Augmentation</u>

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.

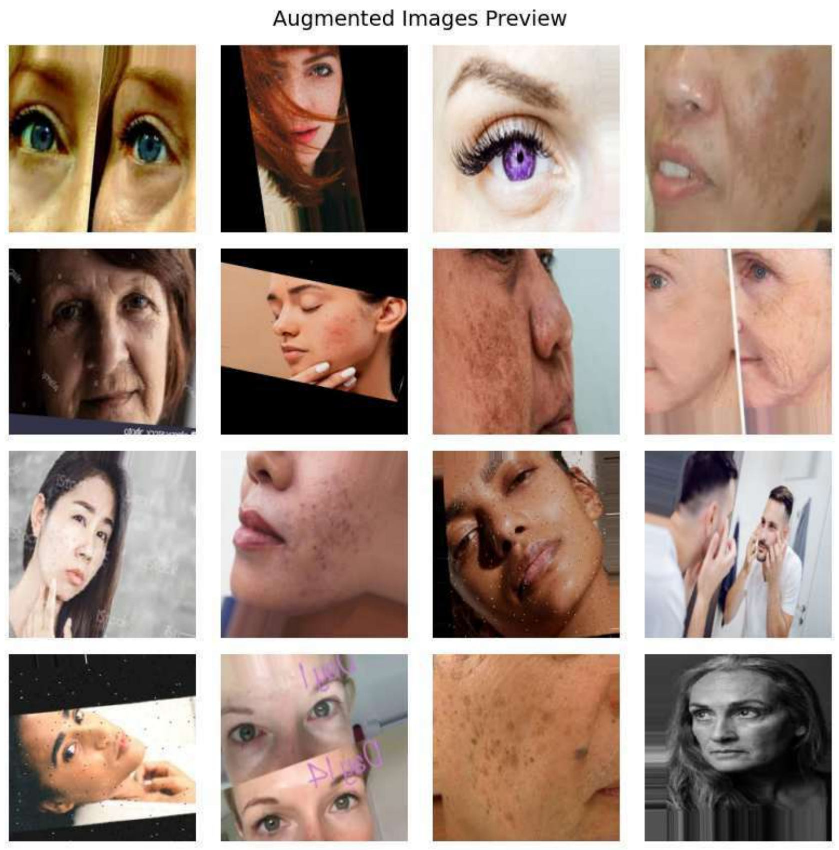


Figure 2: A Preview of the Augmented Images (16 images)

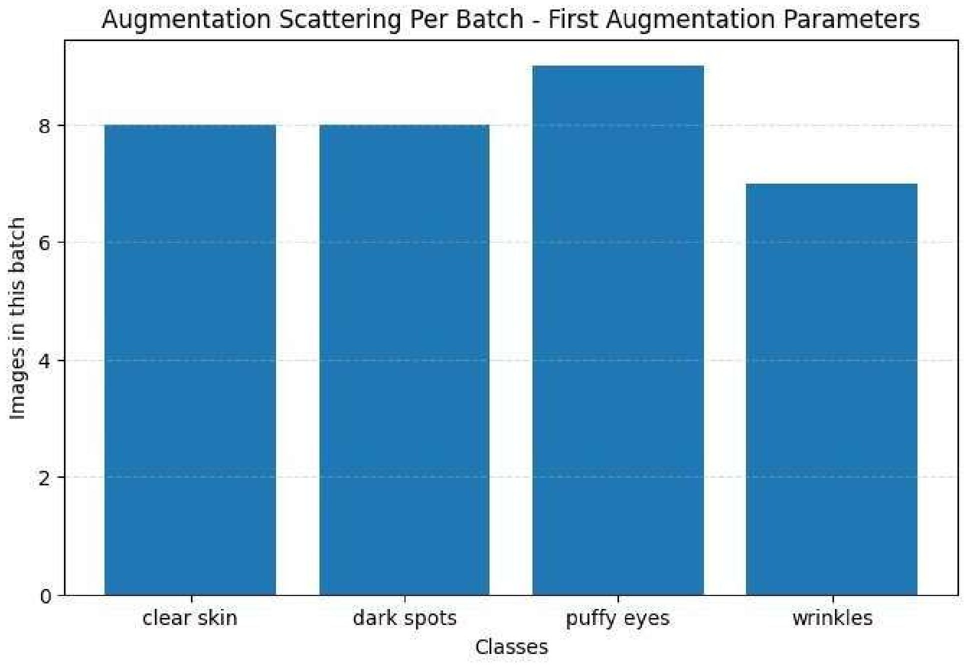


Figure 3: Class-Wise Augmentation Scattering Plot

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

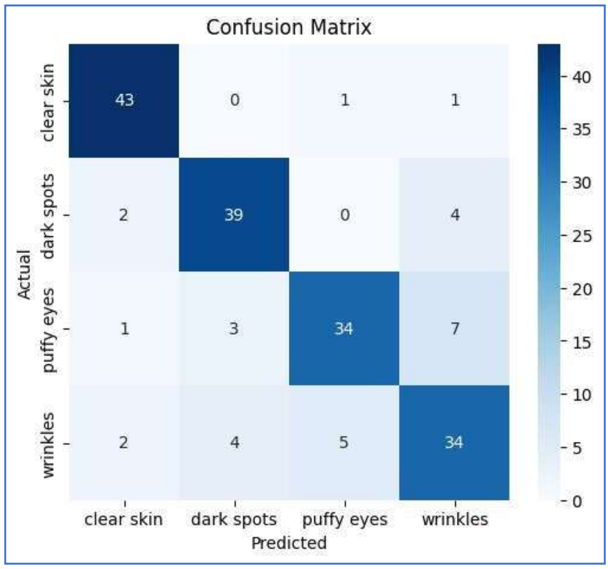


Figure 4: Confusion Matrix – Model 1

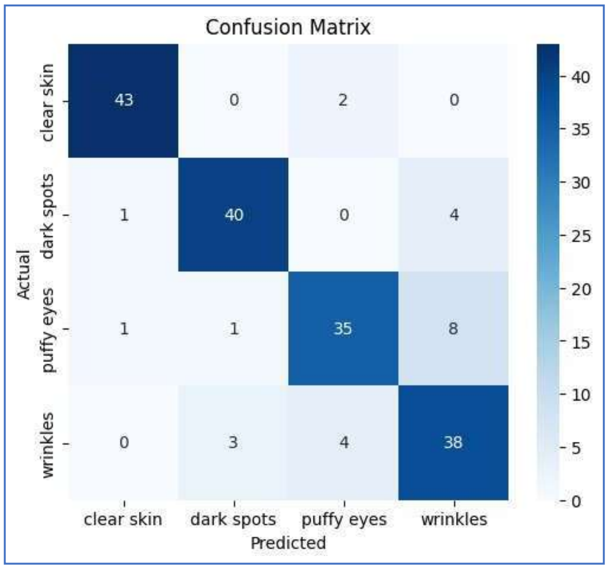


Figure 5: Confusion Matrix – Model 2

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

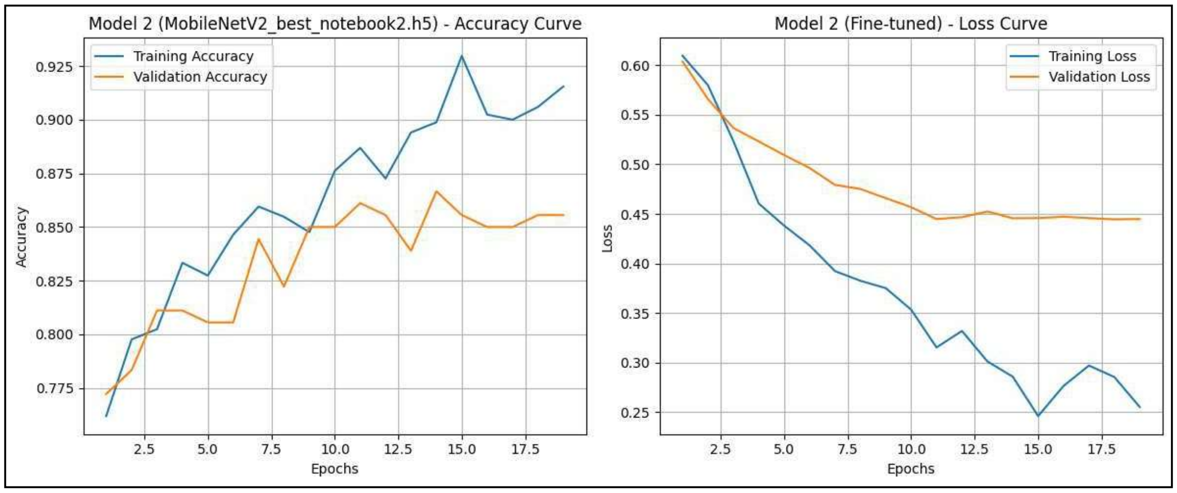


Figure 6: Accuracy and Loss Curves of the Final Model (MobileNetV2_Model2_Final.h5)

Module 4: Face Detection and Prediction Pipeline

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

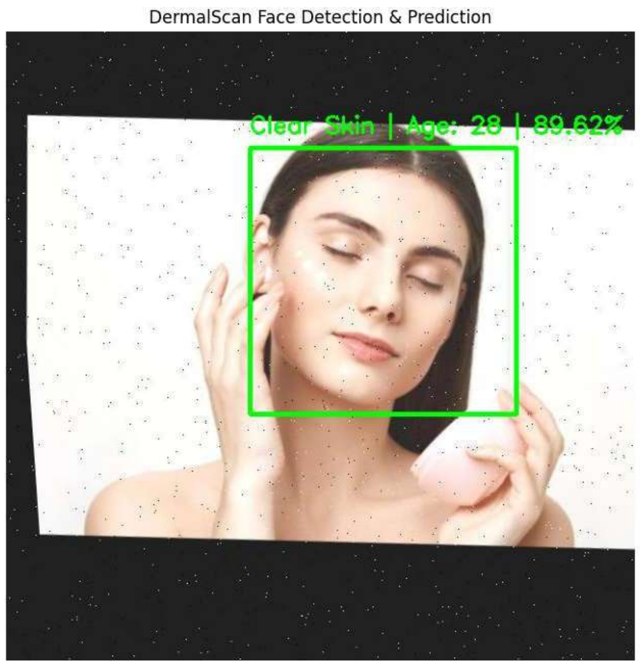

DermalScan Face Detection & Prediction
Clear Skin | Age: 28 | 89.62%

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.

Milestone 3 of the Project

Module 5: Web UI for Image Upload and Visualization

Objective

The objective of Module 5 is to design and implement a responsive, user-friendly web interface that allows users to upload facial images, visualize detected faces with annotations, and view prediction results clearly without UI lag. User should also be able to download the annotated image and the predictions as a csv file.

Technologies Used

- **HTML5**: Page structure and layout
- **CSS3**: Styling, responsiveness, and UI enhancements
- **JavaScript (Vanilla JS)**: Client-side logic and asynchronous communication
- **Flask (Frontend Rendering)**: Serving HTML templates and static assets

Frontend Architecture Overview

The frontend consists of a single-page interface rendered by Flask, supported by JavaScript for asynchronous operations. The UI dynamically interacts with the backend inference pipeline through a REST API.

Tasks Implemented

1. Frontend Development

A minimal UI was designed using HTML and CSS to ensure clarity and ease of use. The interface is divided into clearly defined sections:

- Image upload section
- Uploaded image preview
- Annotated image visualization
- Analysis summary section
- Prediction table with CSV download option

The layout ensures a logical flow of interaction from image upload to result analysis.

2. Image Upload and Preview

- Implemented an image upload input that supports common image formats such as JPG, PNG and JPEG.
- The uploaded image is displayed instantly on the UI without a page reload.
- Handles both single-face and multi-face image uploads seamlessly.
- Utilizes asynchronous requests (Fetch API) to avoid UI blocking.

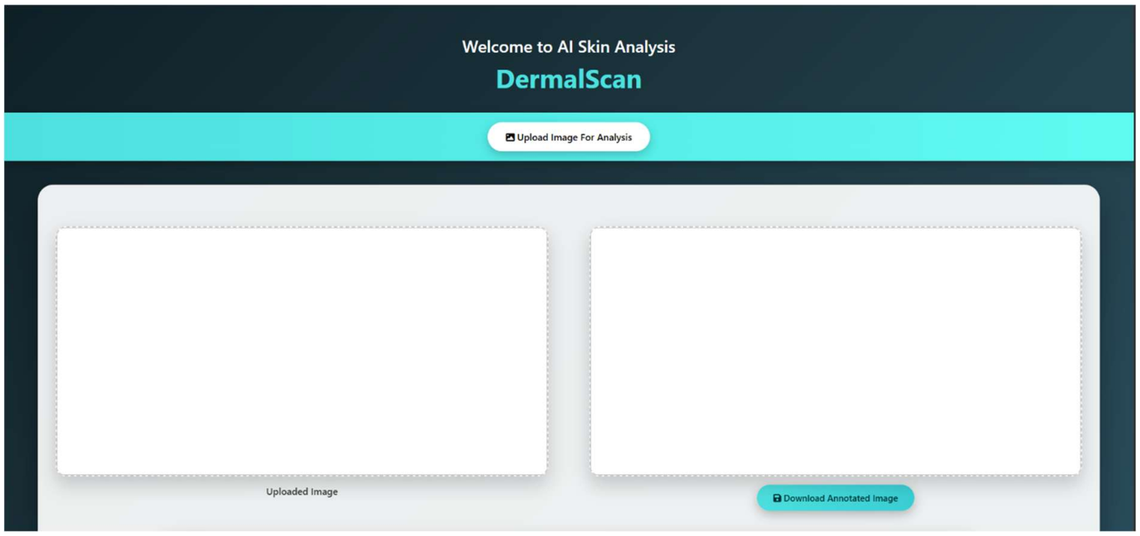
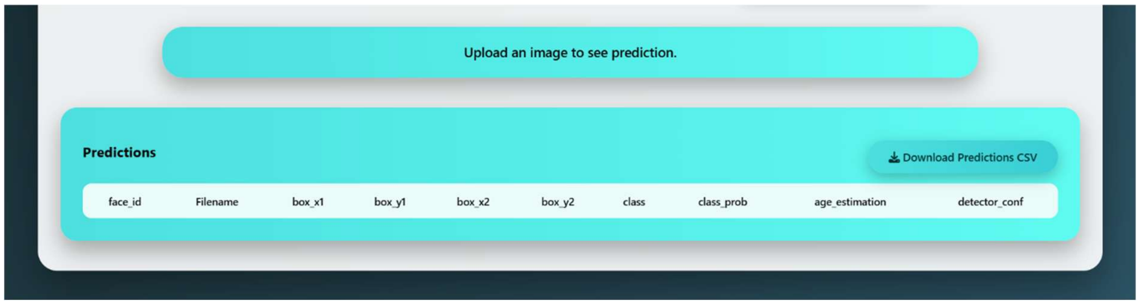


Figure 7: User-Interface of DermalScan AI Skin Analysis - 1



face_id	Filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_estimation	detector_conf

Figure 8: User-Interface of DermalScan AI Skin Analysis - 2

3. Visualization and Annotation Display

- The annotated image returned from the backend is displayed dynamically.
- Bounding boxes are drawn around detected faces.
- Each face annotation includes:
    - Skin condition label (Predicted Class)
    - Estimated age
    - Confidence score (percentage)

This ensures visual clarity and easy interpretation of results.

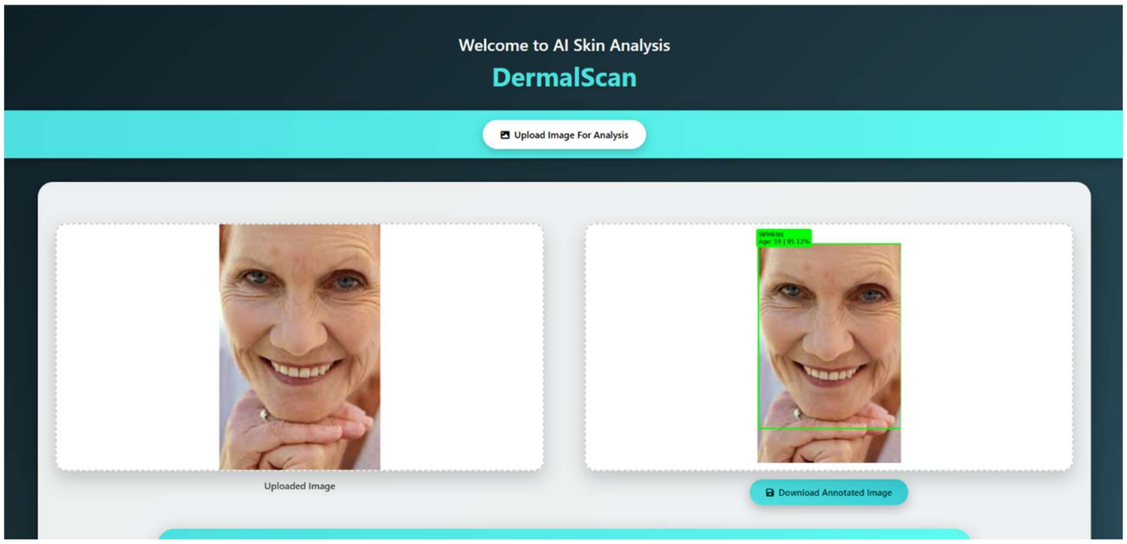
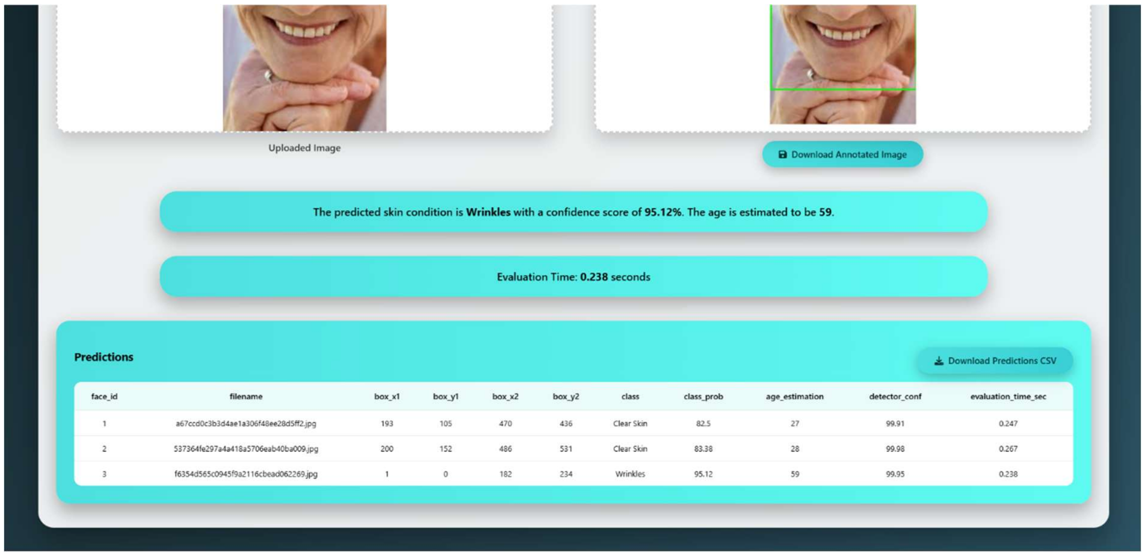


Figure 9: Visualization of annotated Image with Output label



The predicted skin condition is **Wrinkles** with a confidence score of **95.12%**. The age is estimated to be **59**.

Evaluation Time: **0.238** seconds

Predictions

face_id	filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_estimation	detector_conf	evaluation_time_sec
1	a67ccd0c3b3d4ae1a306f48ee28d5ff2.jpg	193	105	470	436	Clear Skin	82.5	27	99.91	0.247
2	537364fe297a4a418a5706eab40ba009.jpg	200	152	486	531	Clear Skin	83.38	28	99.98	0.267
3	f6354d565c0945f9a2116cbead062269.jpg	1	0	182	234	Wrinkles	95.12	59	99.95	0.238

Figure 10: Result Text, Time taken for Analysis and Prediction Table Output

4. Multi-Face Handling in UI

The UI automatically adjusts based on the number of detected faces:

- Single face images
    - Full detailed annotation
    - Detailed prediction text with skin condition, confidence score and the estimated age.
- Multiple face images
    - Compact annotations using Face ID + class name
    - Reduces text overlap and visual congestion
    - Maintains clean and professional visualization

Multi-face Handling Logic for Result Text

```
# Summary Response
  total_faces = len(results)
  if total_faces == 1:
    r = results[0]
    response_text = (
    f"The predicted skin condition is <b>{r['class']}</b> "
    f"with a confidence score of <b>{r['class_prob']}%</b>. "
    f"The age is estimated to be <b>{r['age_estimation']}</b>."
    )
multi_face = False
  else:
    # For multiple faces
    summary = defaultdict(list)
    for r in results:
      summary[r["class"]].append((r["class_prob"], r["age_estimation"]))
      summary_parts = []
      for cls, values in summary.items():
        probs = [v[0] for v in values]
        ages = [v[1] for v in values]
        summary_parts.append(f"<b>{len(values)}</b> <b>{cls}</b> "
                        f"(average confidence <b>{sum(probs)/len(probs):.1f}%</b>) "
                        f"with an average age <b>{sum(ages)//len(ages)}</b>")
        response_text = (f"<b>Analysis Summary:</b><br>"
                       f"Detected <b>{total_faces}</b> faces: " +
                       ", ".join(summary_parts))
    multi_face = True
```

Module 6: Backend Pipeline for Model Inference

Objective

The objective of Module 6 is to build a robust backend pipeline capable of performing:

- Face detection
- Skin condition classification
- Age estimation
- Annotation rendering
- Seamless communication with the frontend

All results must be returned within an acceptable response time.

Technologies Used

- **Python:** Main language for backend logic
- **Flask:** Backend framework
- **TensorFlow / Keras**: Deep learning model inference
- **OpenCV (DNN Module)**: Face detection
- **Matplotlib:** Annotation rendering
- **NumPy:** Numerical operations
- **CSV & Flask Session Storage**: Prediction logging and export

Backend Architecture Overview

The backend pipeline follows a structured flow:

1. Input Image Handling
2. Face Detection using DNN
3. Preprocessing for Model Inference
4. Skin Condition Classification & Age Estimation
5. Post-processing & Annotation Rendering
6. Response Packaging for Frontend

Key Backend Components

1. DNN-Based Face Detection

A Deep Neural Network (DNN) face detector is a CNN-based model trained on large-scale datasets to detect faces accurately under varying conditions.

In this project we have used:

- OpenCV’s DNN module
- SSD (Single Shot Detector) architecture
- ResNet-10 backbone

The detector predicts the:

- Face presence
- Bounding box coordinates
- Detection confidence score

```
# DNN Face Detector

PROTO = "deploy.prototxt"

WEIGHTS = "res10_300x300_ssd_iter_140000.caffemodel"

face_net = cv2.dnn.readNetFromCaffe(PROTO, WEIGHTS)
```

Why we have used DNN over traditional Haar Cascade for face detection?

Traditional Haar Cascade	DNN Face Detector
Sensitive to lighting	**Robust to lighting**
Misses faces easily	**High recall**
Poor multi-face support	**Excellent multi-face support**
Outdated	**Industry standard**

Table 4: Comparison between Haar Cascade and DNN face detector

2. Blob Creation and DNN Inference

The input image is converted into a blob, which:

- Normalizes pixel values
- Resizes the image
- Converts it into a tensor suitable for DNN inference

This ensures consistent input and reliable detection performance.

```
# Face Detection
  blob = cv2.dnn.blobFromImage(
    cv2.resize(image, (300, 300)),
    1.0,
    (300, 300),
    (104.0, 177.0, 123.0)
  )
  face_net.setInput(blob)
  detections = face_net.forward()
```

3. Multi-Face Detection with Non-Maximum Suppression (NMS)

Why **Non-Maximum Suppression** is Required?

DNN detectors often output multiple overlapping bounding boxes for the same face. Without filtering, this causes duplicate detections.

Non-Maximum Suppression (NMS):

NMS removes overlapping boxes by:

- Retaining the bounding box with the highest confidence
- Suppressing overlapping boxes beyond a defined threshold

This allows:

- Accurate multi-face detection
- Elimination of redundant detections
- Clean visualization and correct face count

```
# Non-Maximum Suppression (NMS)

indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
```

4. Face Preprocessing and Model Inference

Each detected face undergoes:

- Cropping using bounding box
- Resizing to 224×224
- Pixel normalization
- Expansion into batch format

The processed face is passed into a **trained MobileNet-based CNN model** that predicts skin conditions.

6. Annotation Rendering

Matplotlib is used to:

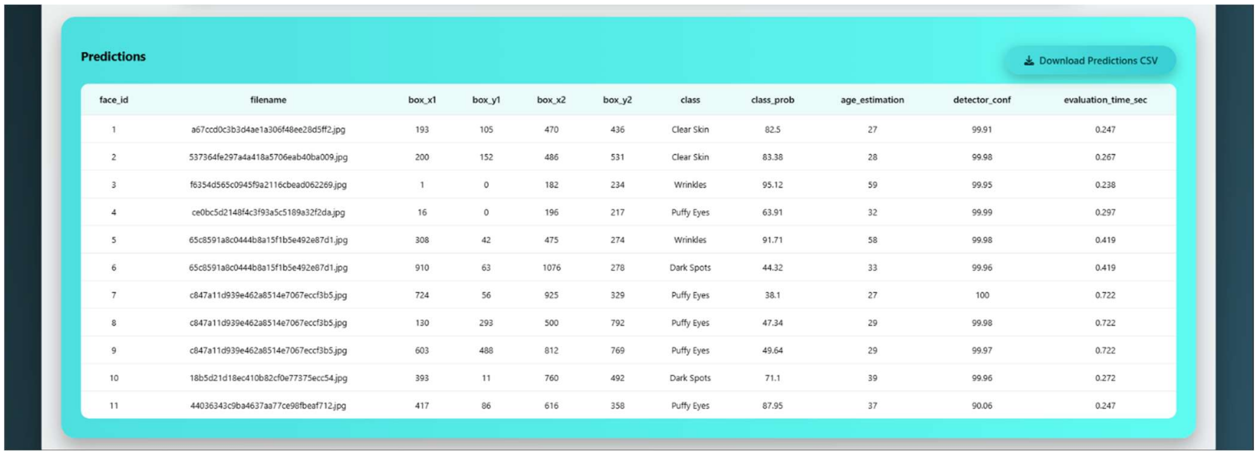
- Draw bounding boxes
- Overlay class labels, confidence, and age
- Adjust annotation detail dynamically for congested images

The annotated image is saved and returned to the frontend.

7. Session-Based Prediction Logging

- Each detected face is assigned a Face ID
- Face IDs persist throughout the session
- Predictions are logged with:
  - Face ID
  - Bounding box coordinates
  - Predicted class
  - Confidence score
  - Estimated age

This data is displayed in a table.



Predictions

face_id	filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_estimation	detector_conf	evaluation_time_sec
1	a67ccd0c3b3d4ae1a306f48ee28d5ff2.jpg	193	105	470	436	Clear Skin	82.5	27	99.91	0.247
2	537364fe297a4a418a5706eab40ba009.jpg	200	152	486	531	Clear Skin	83.38	28	99.98	0.267
3	f6354d565c0945f9a2116cbead062269.jpg	1	0	182	234	Wrinkles	95.12	59	99.95	0.238
4	ce0bc5d2148f4c3f93a5c5189a32f2da.jpg	16	0	196	217	Puffy Eyes	63.91	32	99.99	0.297
5	65c8591a8c0444b8a15f1b5e492e87d1.jpg	308	42	475	274	Wrinkles	91.71	58	99.98	0.419
6	65c8591a8c0444b8a15f1b5e492e87d1.jpg	910	63	1076	278	Dark Spots	44.32	33	99.96	0.419
7	c847a11d939e462a8514e7067eccf3b5.jpg	724	56	925	329	Puffy Eyes	38.1	27	100	0.722
8	c847a11d939e462a8514e7067eccf3b5.jpg	130	293	500	792	Puffy Eyes	47.34	29	99.98	0.722
9	c847a11d939e462a8514e7067eccf3b5.jpg	603	488	812	769	Puffy Eyes	49.64	29	99.97	0.722
10	18b5d21d18ec410b82cf0e77375ecc54.jpg	393	11	760	492	Dark Spots	71.1	39	99.96	0.272
11	44036343c9ba4637aa77ce98fbeaf712.jpg	417	86	616	358	Puffy Eyes	87.95	37	90.06	0.247

Figure 11: Predictions Table for One Session

8. Summary Generation Logic

- Single-face images display detailed individual results.
- Multi-face images generate an aggregated summary:
  - Face count per class
  - Average confidence
  - Average age per condition

This avoids UI clutter while providing meaningful insights.

Frontend–Backend API Integration

API Type Used: Custom REST API built using Flask

The backend returns structured JSON containing:

- Annotated image URL
- Per-face prediction data
- Dynamic summary text
- CSV-ready table data

The frontend consumes this data asynchronously for seamless updates.

Milestone 4 of the Project

Milestone 4 focuses on completing the application workflow by enabling result export, ensuring reliable logging, and preparing comprehensive documentation for end users and developers. This phase ensures that the system is not only functionally complete but also usable, maintainable, and presentation-ready.

Module 7: Export and Logging

Key Features Implemented

1. Annotated Image Export

- The system allows users to download the annotated image generated after inference.
- Each annotated image includes:
    - Bounding boxes around detected faces
    - Skin condition labels
    - Estimated age and confidence (for single-face images)
    - Face IDs and compact labels (for multi-face images)
- This ensures users can retain a visual record of the analysis for future reference.

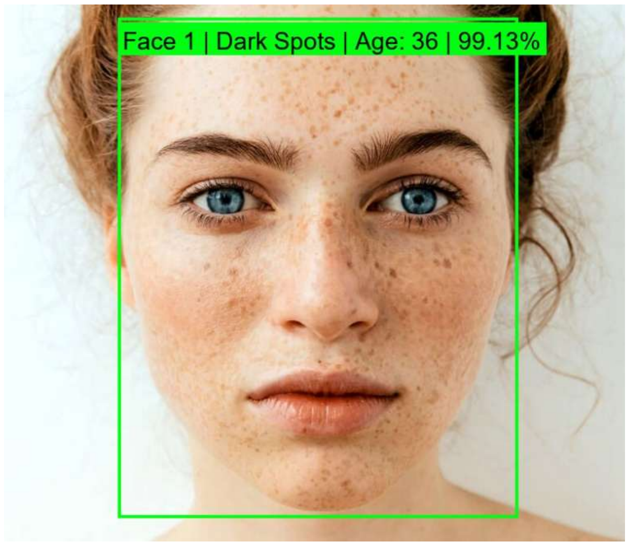
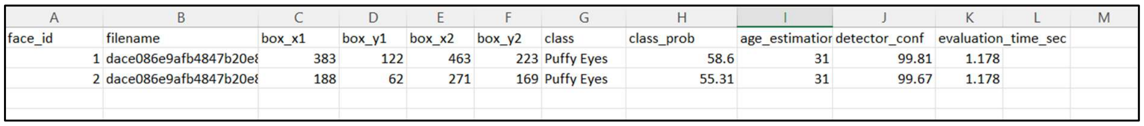


Figure 12: Annotated Image Downloaded

2. CSV Prediction Export

- A structured CSV export option is provided for logging predictions.
- Each row in the CSV contains detailed per-face information:
    - Face ID (persistent across the session)
    - Image filename
    - Bounding box coordinates
    - Predicted skin condition
    - Confidence score
    - Estimated age
    - Face detector confidence
    - Evaluation Time
- The CSV format ensures compatibility with external tools such as spreadsheets or data analysis software.



A	B	C	D	E	F	G	H	I	J	K	L	M
face_id	filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_estimatior	detector_conf	evaluation_time_sec		
1	dace086e9afb4847b20e8	383	122	463	223	Puffy Eyes	58.6	31	99.81	1.178		
2	dace086e9afb4847b20e8	188	62	271	169	Puffy Eyes	55.31	31	99.67	1.178		

Figure 13: Predictions Table exported as a CSV File

3. Testing on Diverse Images

- The system was tested on:
    - Single-face images
    - Multiple-face images
    - Images with varied lighting, age groups, and skin characteristics
- This ensured stable detection, correct annotation placement, and consistent export behaviour.

<u>**Module 8: Documentation and Final Presentation**</u>

Overview

Module 8 focuses on consolidating the complete system implementation into well-structured documentation and preparing the project for final presentation. This module ensures that both users and developers can clearly understand the system's functionality, workflow, architecture, and usage without requiring prior familiarity with the codebase. The documentation also serves as a reference for future maintenance, enhancement, and demonstration of the project.

Objectives

The primary objectives of this module are:

- To clearly document the system architecture, workflow, and design decisions
- To provide step-by-step guidance for users and developers
- To organize the project into a version-controlled repository
- To prepare the project for demonstration and reproducibility

Documentation Structure

The documentation was organized into multiple logical sections to ensure clarity and ease of navigation:

1. Project Overview

This section introduces the AI-powered skin analysis system, explaining its purpose and high-level functionality. It outlines how the system performs face detection, skin condition classification, age estimation, and result visualization through a web-based interface.

2. System Architecture

The architecture description explains the complete pipeline:

- Frontend web interface for image upload and visualization
- Backend inference pipeline built using Flask
- Face detection using a DNN-based SSD model
- Skin condition classification using a trained deep learning model
- Annotation, logging, and export mechanisms

A clear flow from user input to output generation is documented to help readers understand the end-to-end processing.

3. **README File in GitHub**

The README File is created in the GitHub repository in such a way that both an user and a developer can utilize the project repository for their intended purposes.

User Guide

This section explains how an end user interacts with the system:

- Uploading an image through the web interface
- Viewing the uploaded and annotated images
- Understanding prediction results and summaries
- Downloading annotated images and CSV prediction logs

The guide ensures that a user can operate the system without technical background.

Developer Guide

The developer guide details:

- Project folder structure
- Purpose of major files and directories
- Backend logic for face detection, inference, and annotation
- Session-based prediction handling
- CSV logging and export functionality

This section is intended to support further development, debugging, and extension of the system.

4. Key Technical Concepts

Important concepts used in the project are documented, including:

- Deep Neural Networks (DNN) for face detection
- Blob creation for DNN input preprocessing
- Non-Maximum Suppression (NMS) for multi-face detection
- Tensors and model inference
- Session management for prediction tracking
- Evaluation time measurement for performance analysis

These explanations help bridge the gap between implementation and theory.

5. Performance and Reliability

This section summarizes the system's runtime behaviour:

- Average inference time per image
- Handling of single-face and multi-face images
- Adaptive annotation strategy for congested images
- Stable performance under repeated uploads

Performance measurements and evaluation time logging are explained to highlight system efficiency.

6. Repository and Version Control

The project was structured into a clean GitHub repository containing:

- Source code
- Static assets
- Model files (referenced appropriately)
- Documentation files

Version control practices ensure reproducibility and organized collaboration.

Final Presentation Preparation

To support final presentation and demonstration:

- The application was verified for end-to-end functionality
- All export features (annotated image and CSV) were tested
- UI behaviour was validated for responsiveness and clarity
- Documentation was aligned with implementation details

The project is prepared for live demonstration, walkthroughs, and future enhancement.

Module 8 successfully consolidates the technical implementation into comprehensive documentation and presentation-ready material. By clearly describing system design, functionality, and usage, this module ensures that the AI-powered skin analysis system can be effectively understood, demonstrated, and extended, completing the project in a structured and professional manner.

Conclusion of the Project

DermalScan – AI Skin Analysis successfully **demonstrates the integration of computer vision and deep learning techniques** into a cohesive web-based skin analysis system. The project brings together a responsive user interface and a robust backend pipeline to enable end-to-end image processing, face detection, skin condition classification, age estimation, and result visualization within a single application.

By leveraging **a DNN-based face detector combined with Non-Maximum Suppression**, the system reliably detects and processes multiple faces in a single image while maintaining clear and uncluttered visual annotations. The adaptive annotation strategy ensures that results remain readable for both single-face and crowded multi-face scenarios. The use of a deep learning classification model allows the system to analyse facial regions at a fine level and produce meaningful condition-level insights.

The application is designed with usability and transparency in mind. Features such as real-time image preview, annotated output visualization, session-based prediction logging, downloadable results, and evaluation time reporting contribute to a smooth and informative user experience. The modular structure of the backend further improves maintainability and supports future enhancements.

Overall, DermalScan represents a complete and well-structured implementation of an AI-assisted image analysis workflow. It serves as a strong foundation for further research, feature expansion, or deployment in real-world scenarios where automated visual analysis and clear result interpretation are essential.

Key Terms

Categorical Labelling

The process of assigning images to distinct classes or categories to enable 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

A data sampling technique where samples from each class are selected proportionally to preserve the original class distribution.

Image Resizing

Adjusting image dimensions to a fixed size (e.g., 224×224 pixels) required by the neural network model.

Normalization

Scaling pixel intensity values to a standard range (commonly 0–1) to improve model stability and convergence during training.

Image Augmentation

Applying transformations such as rotation, flipping, zooming, or shifting to increase dataset diversity and reduce overfitting.

Overfitting

A condition where a model performs very well on training data but poorly on unseen or validation data due to lack of generalization.

One-Hot Encoding

A technique that converts categorical labels into binary vectors, where each class is represented by a unique position set to 1.

Transfer Learning

Using a pretrained neural network as a starting point to accelerate training and improve performance on a related task.

Pretrained Model

A neural network already trained on a large dataset (e.g., ImageNet) and reused for feature extraction or fine-tuning.

Fine-Tuning

Retraining selected layers of a pretrained model to adapt it to a new, task-specific dataset.

Categorical Cross-Entropy Loss

A loss function used in multi-class classification to measure the difference between predicted probabilities and true labels.

Adam Optimizer

An adaptive optimization algorithm that efficiently updates neural network weights using momentum and learning-rate scaling.

Validation Accuracy

The percentage of correctly classified samples from a validation dataset during model training.

Confusion Matrix

A table that summarizes correct and incorrect predictions across all classes, helping evaluate classification performance.

Classification Report

A detailed summary of precision, recall, F1-score, and support for each class in a classification task.

Precision

The proportion of correctly predicted positive samples among all predicted positives for a given class.

Recall

The proportion of correctly predicted samples among all actual instances of a given class.

F1-Score

A harmonic mean of precision and recall that provides a balanced performance metric.

Haar Cascade Classifier

A traditional OpenCV-based face detection method using handcrafted features and sliding window techniques.

Deep Neural Network (DNN)

A neural network with multiple hidden layers used in this project for robust face detection via OpenCV's DNN module.

Blob

A pre-processed image format used as input to DNN models, containing resized, normalized pixel data in a tensor form.

Non-Maximum Suppression (NMS)

A technique used to remove overlapping bounding boxes by retaining only the most confident detections.

Tensor

A multi-dimensional array used to represent data (images, feature maps, or model outputs) in deep learning frameworks.

Confidence Score

The probability value assigned by the model to a predicted class, indicating prediction certainty.

Heuristic-Based Age Estimation

A rule-based approach used to estimate age from the predicted skin condition and confidence score.

Bounding Box

A rectangular region drawn around detected objects (faces) to indicate their location in an image.

Session Storage

A server-side mechanism used to temporarily store prediction results across multiple user interactions within a session.

Evaluation Time

The total time taken (in seconds) by the backend pipeline to process an image from upload to final prediction.

Inference

The process of using a trained model to make predictions on new, unseen input data.