DermalScan:AI_Facial Skin Aging Detection App

Infosys SpringBoard Virtual Internship Program

Submitted By,

Priya Ghosal

Under the guidance of Mentor **Mr.Praveen**

Project Statement:

Facial skin aging shows signs like wrinkles, dark spots, and puffiness, which can be hard to identify correctly without a specialist. As AI is becoming more common in healthcare, there is a need for an easy, reliable, and automatic system to detect and classify these aging signs. This project aims to create such a system using deep learning and computer vision.

Expected outcome:

1. **A fully structured facial skin aging dataset** organized into four classes:

- Clear Skin
- Dark Spots
- Puffy Eyes
- Wrinkles

2. **A DataFrame (df)** containing all image filepaths and corresponding labels extracted from folder structure.
3. **Successfully preprocessed images**, resized to 256×256, normalized (0–1 scaling), and stored in an array ready for deep learning models.
4. **Automated data augmentation pipeline** using Keras ImageDataGenerator to improve dataset variability and reduce overfitting.
5. **Label encoding with one-hot vectors**, enabling compatibility with classification models.
6. **Class distribution visualization** for dataset balance analysis and **Augmentation preview visualization** showing how transformations modify sample images.

Tech Stack Used

1. Programming Language

- **Python 3.10.11**

2. Deep Learning Frameworks

- **TensorFlow / Keras**
    - Image loading and augmentation
    - One-hot encoding
    - Data pipeline utilities

3. Image Processing Libraries

- **OpenCV (cv2)**
    - Image reading
    - Resizing
    - Color conversion
- **Pillow (PIL)**
    - Safe image opening
    - Handling corrupted images

5. Visualization Libraries

- **Matplotlib**
    - Augmentation preview
    - Image display
- **Seaborn**
    - Class distribution plots

6. Machine Learning Tools

- **Scikit-Learn**
    - Train/Validation split
    - Data shuffling

7. Development Environment

- **Jupyter Notebook / JupyterLab**

MILESTONE 1: Dataset Preparation and Preprocessing

MODULE 1: Dataset Setup & Image Labeling

Module 1 focuses on preparing the dataset so the model can understand it later. In deep learning, data preparation is one of the most important steps because the model's accuracy completely depends on clean, well-organized data.

1. **Dataset Setup**
   The dataset was structured into four separate class folders — **Wrinkles**, **Dark Spots**, **Puffy Eyes**, and **Clear Skin** — each containing images representing that skin condition.

   ### Code Snippet

   ```
   base_dir = "data"
   classes = ["Wrinkles", "Dark Spots", "Puffy Eyes", "Clear Skin"]
   ```

   Folder structure is like:
   data/
   ├── Wrinkles
   ├── Dark Spots
   ├── Puffy Eyes
   └── Clear Skin

2. **Building a DataFrame of Filepaths & Labels**

   We create a function to walk through each class folder, pick up image paths, and store them with their correct label.

   This helps convert the raw dataset into a *machine-readable table*.

3. **Creating the Final Dataset Table**

   After collecting the image filepaths and class labels from all four folders, we organized this information into a structured pandas DataFrame. This table acts as the central dataset index, where each row represents one image along with its corresponding label. The DataFrame contains two main columns:

- filepath: the complete directory path to the image file

- label: the class to which the image belongs (Wrinkles, Dark Spots, Puffy Eyes, or Clear Skin)

Code Snippet

```
df = build_df_from_folders(base_dir, classes)

print("Total images:", len(df))
print(df["label"].value_counts())
df.head()
```

Output

label	count
Dark Spots	303
Wrinkles	300
Puffy Eyes	300
Clear Skin	300

4. Class Distribution

The bar chart shows the number of images available for each skin condition category in the dataset. There are four classes:

- Wrinkles
- Dark Spots
- Puffy Eyes
- Clear Skin

Code Snippet

```
plt.figure(figsize=(8,5))

colors = ["blue", "yellow", "green", "purple"]

sns.countplot(data=df, x="label", hue="label", palette=colors,
legend=False)

plt.title("Class Distribution", fontsize=14)

plt.xlabel("Class")

plt.ylabel("Number of Images")

plt.xticks(rotation=20)

plt.tight_layout()

plt.show()
```

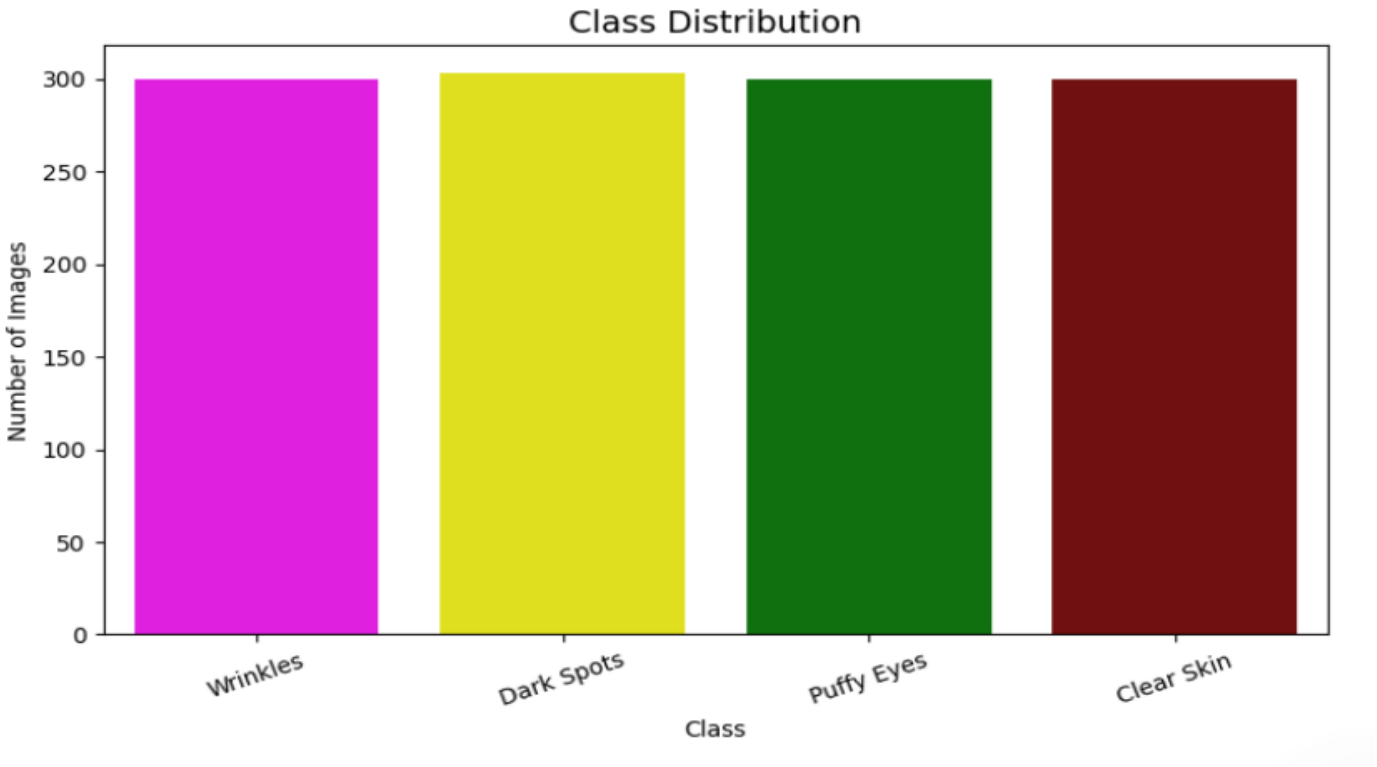

Fig 1. Class Distribution Plot

Observation From Fig. 1

- All classes contain **approximately 300 images** each.
- No class shows significant under-representation or over-representation.
- The distribution is nearly uniform across all four categories.

Importance of Balanced Distribution

- Prevents the model from becoming biased toward a class with more samples.
- Ensures fair learning and equal exposure to all facial aging conditions.
- Reduces the need for additional techniques like oversampling, undersampling, or class weighting.

Impact on Model Training

- Leads to **more stable training performance**.
- Helps improve model **accuracy, generalization, and robustness**.
- Provides a reliable foundation for preprocessing, augmentation, and later model training.

Module 2: Image Preprocessing and Augmentation

In this module, all facial images are resized to a fixed resolution of **224×224 pixels**, normalized to the range **[0, 1]**, and passed through a set of augmentation operations. These augmentations—such as rotation, flipping, shifting, and zooming—help artificially expand the dataset and introduce variability. This makes the model more generalizable and reduces overfitting.

Additionally, class labels are encoded using **one-hot encoding**, which converts categorical labels (Clear Skin, Dark Spots, Puffy Eyes, Wrinkles) into numerical vectors suitable for training a multi-class classification model.

1. Image Loading & Preprocessing

Every image from Module 1 (df) is:

- Loaded with Pillow (PIL)
- Converted to RGB (ensures color consistency)
- Resized to **224 × 224** (IMG_SIZE = 224)
- Normalized to values between **0 → 1**

2. Data Augmentation

We use **ImageDataGenerator** to apply transformations such as:

- Rotation
- Horizontal shift
- Vertical shift
- Zoom
- Horizontal flipping

Code Snippet:

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest')
```

Fig 2. Augmented Image Preview

3. Label Encoding

```
label_map = {
    "Clear Skin": 0,
    "Dark Spots": 1,
    "Puffy Eyes": 2,
    "Wrinkles": 3
}
```

Then labels are converted to **one-hot vectors**, such as:

```
Wrinkles → [0, 0, 0, 1]
```

5. Train–Validation Split

This allows testing the model on unseen data to measure performance.

Code Snippets:

```
X_train, X_val, y_train, y_val = train_test_split(
    images,
    labels_onehot,
    test_size=0.2,
    random_state=42,
    shuffle=True
)
```

Conclusion

Milestone 1 successfully prepares the dataset through structured organization, preprocessing, labeling, and augmentation. This ensures the data is clean, balanced, and sufficiently diverse to support high-performance model training in subsequent modules. Both Module 1 and Module 2 meet the milestone requirements and complete the foundation for the **DermalScan AI Facial Aging Detection System**.

MILESTONE 2: Model Training and Evaluation

Module 3: Model Training

Objective

In this module, a deep learning–based experimental framework was designed to train and evaluate multiple pretrained Convolutional Neural Network (CNN) models for facial skin aging classification. This module aims to identify the best-performing model that provides high validation accuracy.

The **Categorical Cross-Entropy loss** function was used for training all models. The **Adam optimizer** was employed for model training.

To improve training efficiency and prevent overfitting, multiple callbacks were applied during model training:

- **EarlyStopping**
- **ReduceLROnPlateau**
- **ModelCheckpoint**

Callbacks Configuration

```
callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=5,
        restore_best_weights=True
    ),
    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.3,
        patience=3,
```

```
        min_lr=1e-6
    ),
    ModelCheckpoint(
        "best_model.h5",
        monitor="val_accuracy",
        save_best_only=True
    )
]
```

Models Trained and Observations

1. MobileNetV2

MobileNetV2 was trained as a baseline model using transfer learning. The model was trained for a maximum of 50 epochs using the training and validation datasets prepared in Module 2. A moderate learning rate (1×10^{-4}) was used with the Adam optimizer.

To analyze the learning behavior of the MobileNetV2 model, line charts were plotted for training and validation accuracy and loss across all epochs.

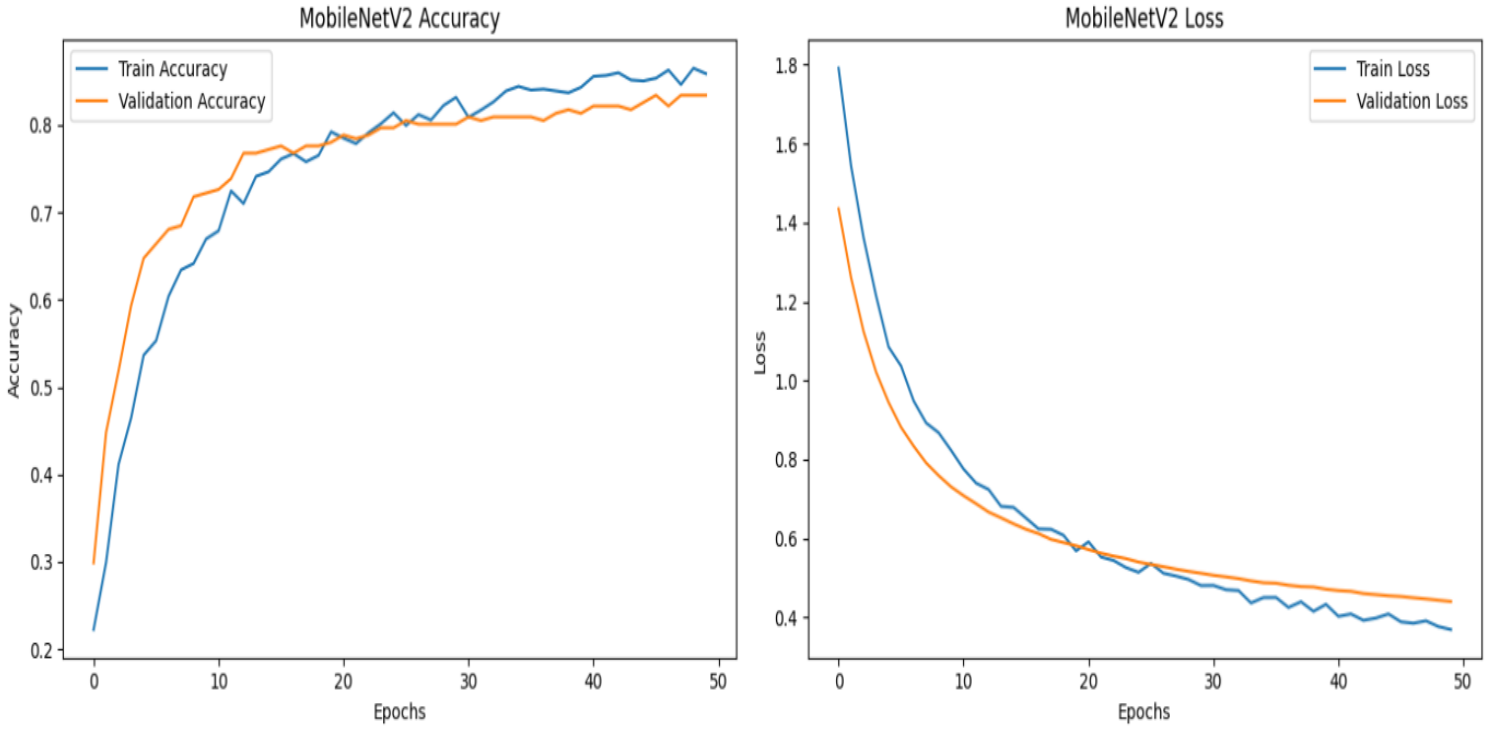

Fig 3. Line Chart of Accuracy and Loss

2. **EfficientNetB0**

EfficientNetB0 was trained using transfer learning to evaluate its performance on facial skin aging classification. The model was trained using the training and validation datasets prepared in Module 2 with a low learning rate (1e-5) and the Adam optimizer.

To study the learning behavior of the EfficientNetB0 model, line charts were plotted for training and validation accuracy and loss across epochs.

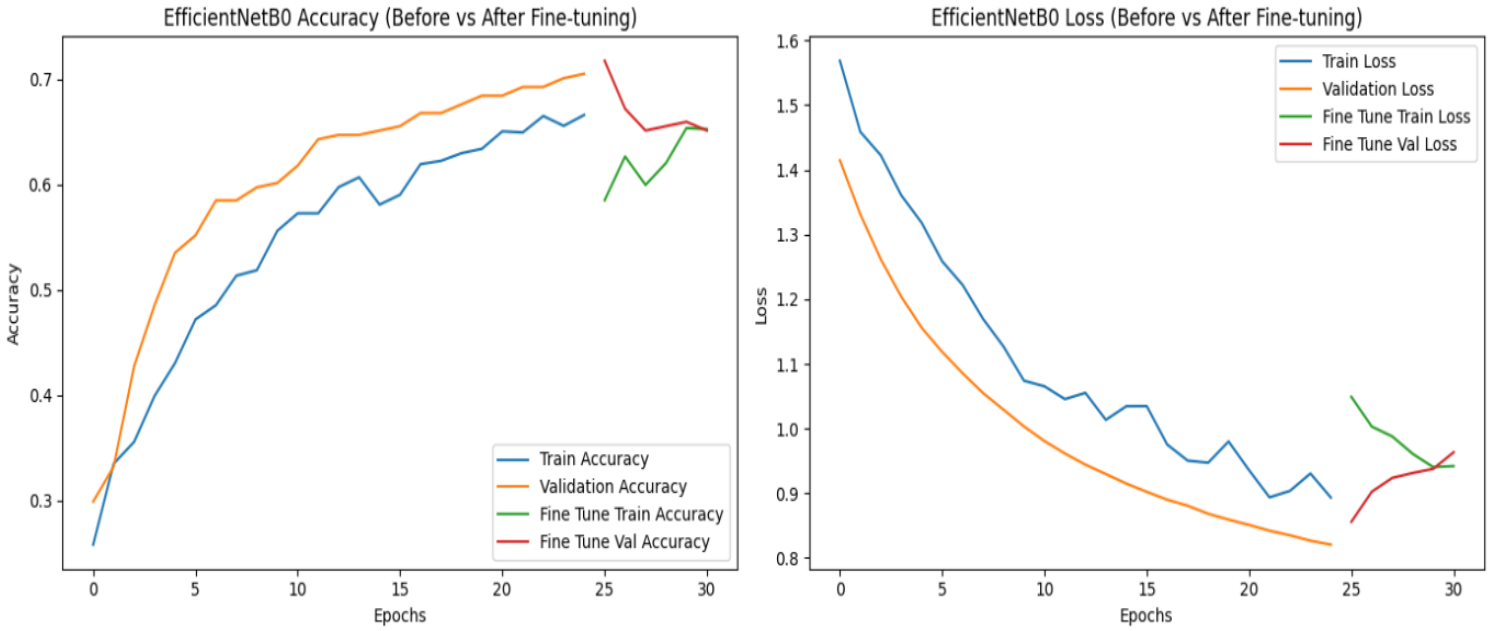

Fig 4. Line Chart of Accuracy and Loss

3. **ResNet50**

ResNet50 was trained using transfer learning due to its strong feature extraction capability and suitability for complex image classification tasks. The model was trained on the training and validation datasets prepared in Module 2 using the Adam optimizer with a very low learning rate (1e-5)

Line charts were plotted to visualize training and validation accuracy and loss for the ResNet50 model.

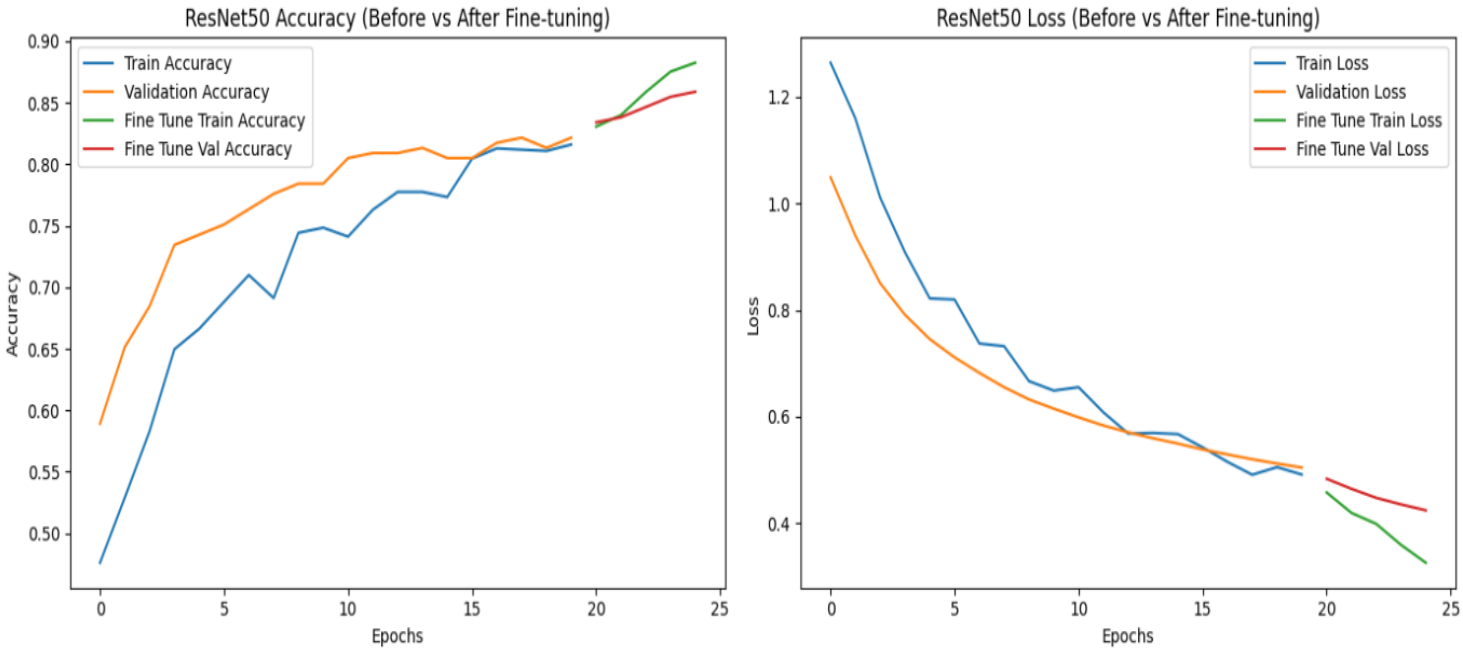

Fig 5. Line Chart of Accuracy and Loss

Model Comparison Table

Model Name	Epochs	Training Accuracy	Validation Accuracy
MobileNetV2	**50**	**86**	**83**
EfficientNetB0	**40**	**65**	**65**
ResNet50	**25**	**88**	**86**

ResNet50 demonstrated the best balance between accuracy and generalization.

Module 4: Face Detection and Prediction Pipeline

Objective

The objective of Module 4 is to perform face detection on input images, extract the detected facial region, and apply the trained deep learning model (ResNet50) to classify facial skin aging signs. The final output displays the predicted class, confidence percentage, and estimated age group using bounding boxes on the image.

Face Detection using Haar Cascade

Haar Cascade is a classical object detection technique provided by OpenCV. It is lightweight, fast, and suitable for real-time applications.

Loading Haar Cascade

```
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades +
"haarcascade_frontalface_default.xml"
)

assert not face_cascade.empty(), "Haar Cascade not loaded"
```

Class Labels and Age Mapping

The predicted class is selected based on maximum probability. An age group is assigned using rule-based mapping.

```
def estimate_age(predicted_class, confidence):
    age_map = {
        "clear skin": (18, 25),
        "dark spots": (25, 35),
        "puffy eyes": (30, 45),
        "wrinkles": (45, 60)
    }

    low, high = age_map[predicted_class]
    age = int(low + (1 - confidence / 100) * (high - low))
    return age
```

Visualization of Final Output

The final output image displays a **bounding box** around the face along with **predicted class, confidence percentage, and age group**.

```
label = class_names[class_index]
        confidence = preds[class_index] * 100
        age = estimate_age(label, confidence)

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

        cv2.rectangle(rgb, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(
            rgb, text, (x, y-10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8,
            (255, 0, 0), 2
```

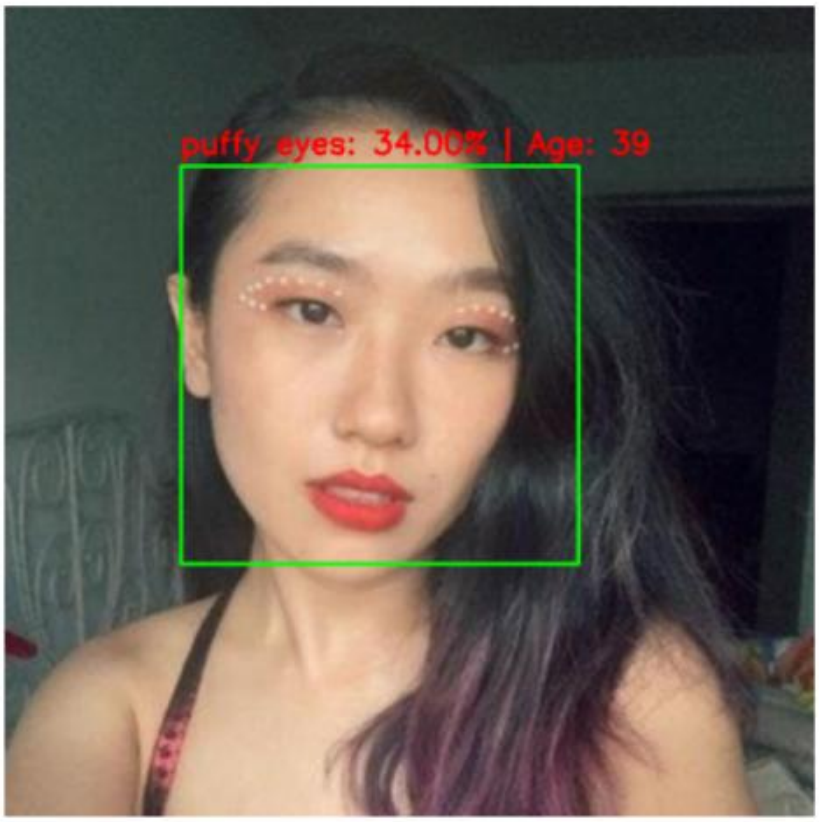

Fig 6. Predicted image

Final Output

A predicted image (annotated image)
Specifically, the function returns:

- Face bounding box
- Predicted skin-aging class
- Confidence percentage
- Estimated age

Conclusion

In this Milestone, pretrained CNN models were evaluated for facial skin aging classification, with ResNet50 achieving the best accuracy and generalization. The selected model was successfully integrated with OpenCV-based face detection to perform end-to-end prediction, displaying skin condition, confidence percentage, and age group through bounding box visualization.

Milestone 3: Frontend and Backend Integration

Overview

Milestone 3 implements a **fully integrated frontend–backend pipeline** for real-time facial skin analysis. This milestone combines user interaction, backend inference, and visualization into a single cohesive workflow.
Instead of separating frontend and backend as independent modules, this milestone emphasizes their **tight integration through a Flask-based architecture**, ensuring seamless data flow from image upload to prediction output.

System Architecture

The system operates as a **single inference pipeline** with the following layers:

- **User Interface (HTML/CSS)**
- **Flask Controller (app.py)**
- **Inference Engine (backend.py)**
- **Deep Learning Model**
- **Result Logging and Visualization**

Integrated Workflow Description

Step 1: Image Upload (Frontend)

- User uploads a facial image via the HTML interface.
- The request is handled by Flask without page reload.
- Image is stored temporarily in the static directory.

Step 2: Flask Mediation

The Flask application (app.py) acts as a **central coordinator**, performing:

- Request handling
- File storage
- Backend invocation
- Result rendering

This eliminates direct coupling between UI and inference logic.

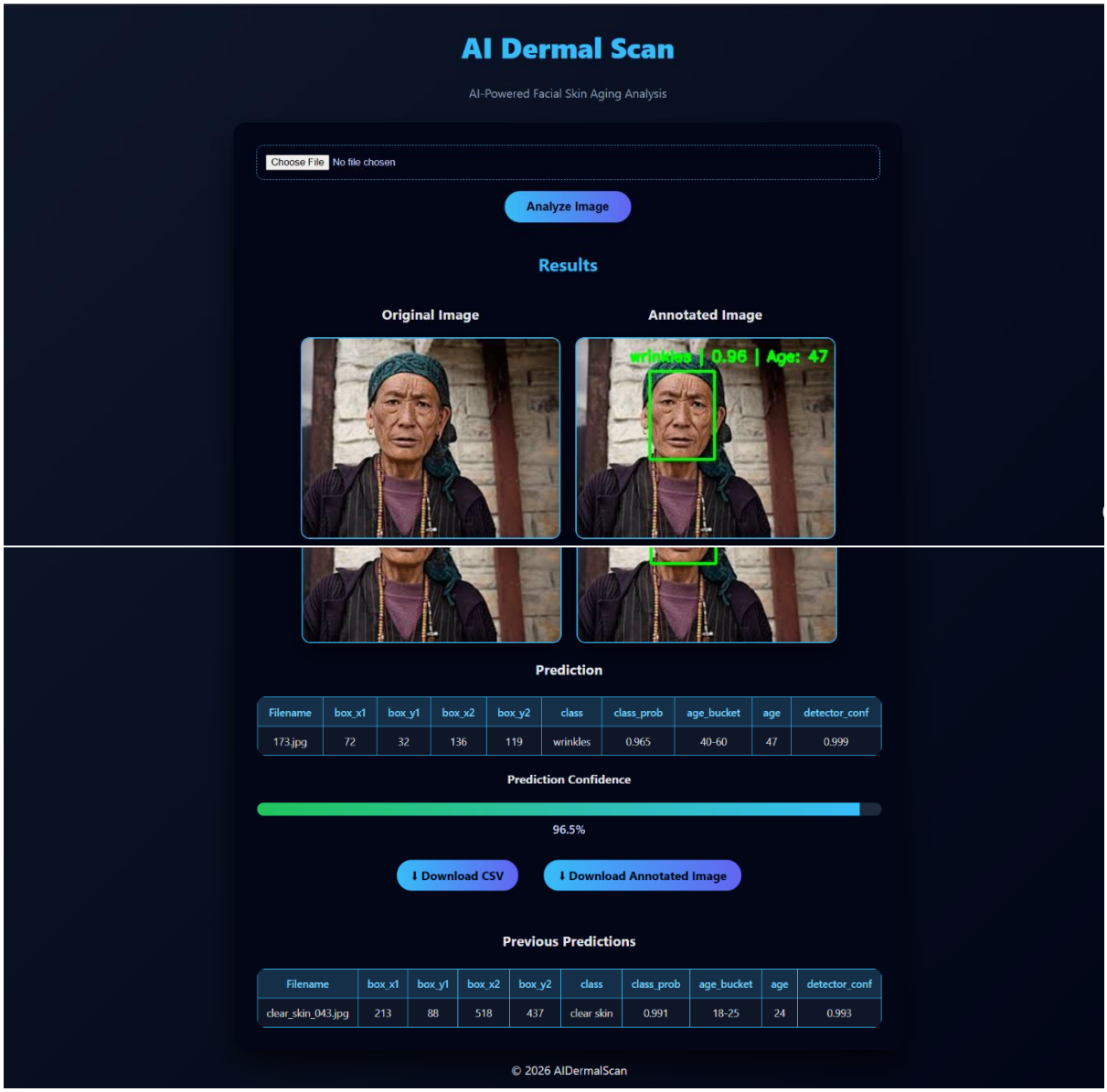

Fig 7. User Interface of AI Dermal Scan

Step 3: Face Detection and Preprocessing

- The uploaded image is passed to the backend.
- CNN-based face detection (OpenCV DNN) identifies the most confident face.
- Face region is cropped and preprocessed for inference.

Fig 8: Detected faces(wrinkles, clear skin, dark spots, puffy eyes) with bounding box.

Step 4: Model Inference

- Preprocessed face image is passed to the MobileNetV2 model.
- Softmax probabilities are generated.
- The class with highest confidence is selected.

Step 5: Age Estimation

- Age is mapped using predefined class-based ranges.
- Random age is generated within the selected bucket.
- Used strictly for visualization and academic demonstration.

Step 6: Result

- Flask renders:
    - Original image
    - Annotated image
    - Prediction confidence
    - Prediction metadata table

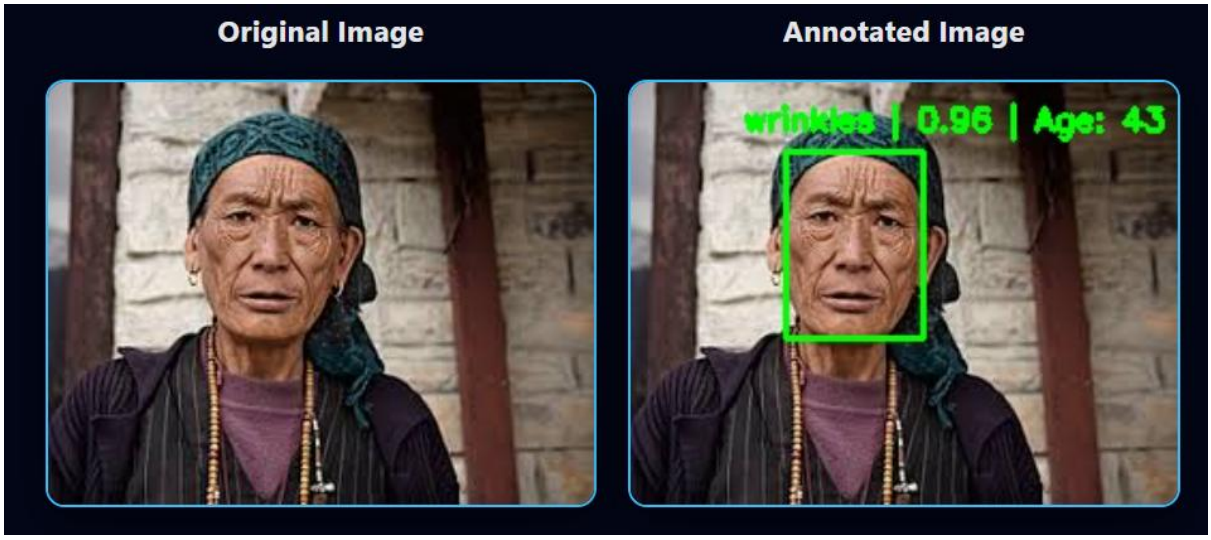

Fig 9: Result

Filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_bucket	age	detector_conf
173.jpg	72	32	136	119	wrinkles	0.965	40-60	43	0.999

Fig 10 : Prediction metadata table

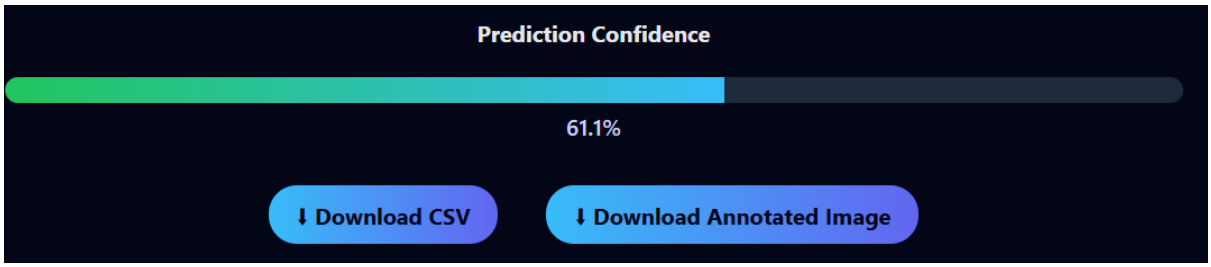

Figure 12: Prediction Confidence and Download buttons

Step 8: Prediction Logging and History

- Results are logged into a CSV file.
- Recent predictions are displayed on the UI.
- Users can download CSV and annotated images.

Filename	box_x1	box_y1	box_x2	box_y2	class	class_prob	age_bucket	age	detector_conf
6.jpg	42	28	221	153	puffy eyes	0.877	30-45	35	0.985
1.jpg	75	16	275	144	wrinkles	0.626	40-60	51	0.945
0d12df28-6795-4693-95e6-baa3c5c7d14b.jpg	19	0	237	141	dark spots	0.898	26-40	38	0.997
clear_skin_002.jpg	235	100	395	301	clear skin	0.824	18-25	22	0.999

Figure 12: Prediction history

Conclusion

The system now supports real-time image upload, automated face detection, deep learning-based skin condition prediction, annotated visualization, and structured result management—forming a solid foundation for future enhancements.

Milestone 4: Export, Logging, and Final Documentation

Overview

Milestone 4 finalizes the system by adding export and logging features, validating results on diverse images, and preparing complete documentation for demonstration.

Module 7: Export and Logging

Objective

To enable users to export prediction results and ensure consistent logging of inference data for validation, analysis, and reporting purposes.

Export Functionality

The frontend interface was enhanced to support **download options** for prediction outputs.

Implemented Export Features

- **Annotated Image Download**

  Users can download the processed image containing:

  - Face bounding box
  - Predicted skin condition
  - Confidence score
  - Estimated age

- **CSV Prediction Log Download**

  Users can download a structured CSV file containing all prediction records

  These features are implemented using Flask routes and file-serving mechanisms.

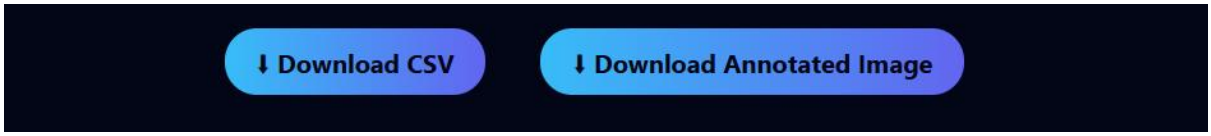

Figure 13: Export options for annotated image and CSV prediction log.

2.3 Prediction Logging

Each inference result is automatically logged into a CSV file (predictions.csv) with consistent formatting.

Logged Fields

- Image filename
- Bounding box coordinates (x1, y1, x2, y2)
- Predicted skin condition
- Class probability
- Age bucket and estimated age
- Face detector confidence

This structured logging ensures:

- Result traceability
- Easy analysis and validation
- Compatibility with spreadsheet tools

AIDermalScan – Prediction Report

Filename	Predicted_Class	Confidence(%)	Age	Age_Bucket	Detector_Conf	Time_Taken(sec)
12.jpg	puffy eyes	61.141	44	30-45	0.96	0.24
1.jpg	wrinkles	79.869	48	40-60	1.0	0.31
clear_skin_043.jpg	clear skin	99.134	21	18-25	0.99	0.25
9.jpg	dark spots	36.302	36	26-40	0.99	0.25

Figure 14: CSV prediction log with structured fields.

Module 8: Documentation and Final Presentation

Objective

To prepare clear and comprehensive documentation that explain system usage, architecture, and implementation details.

User Documentation

Basic user guidance is provided through the web interface itself and the project README.

- The UI is self-explanatory, allowing users to upload images, view predictions, and download results without external instructions.
- The README describes basic usage steps and application flow.

This ensures that even non-technical users can operate the application easily.

Developer Documentation

Developer-level understanding is supported through:

- Clear project folder structure
- Modular code design (app.py, backend.py, frontend templates)
- Inline code readability and logical separation of responsibilities
- Setup and execution instructions included in the README

This allows future developers to understand, run, and extend the project if required.

GitHub Repository Preparation

The GitHub repository was finalized with:

- Clean folder structure
- Proper commit history
- Well-structured README.md including:
    - Project overview
    - Features
    - Tech stack
    - Setup instructions
    - Usage guide
    - Screenshots

Key Outcomes of Milestone 4

- Exportable annotated images
- Consistent and reliable CSV logging
- Thorough testing on diverse inputs
- Complete project documentation
- GitHub-ready repository

Conclusion

The completion of Milestone 4 finalizes the **AI Dermal Scan** system as a fully integrated, end-to-end AI application for facial skin aging analysis. The project successfully delivers real-time prediction, clean visualization, reliable export and logging features, and clear documentation.