DermalScan: AI Facial Skin Aging Detection App

Mentor: Mr. Praveen

Developer: Bhulakshmi Battula

Abstract

DermalScan is an AI-based deep learning application designed to detect and classify facial aging signs such as wrinkles, dark spots, puffy eyes, and clear skin. The project leverages transfer learning using the InceptionV3 model, combined with OpenCV's Haar Cascade for face detection. The model classifies facial aging patterns with high accuracy and visualizes the results through bounding boxes and percentage confidence levels. The project aims to demonstrate how computer vision and AI can contribute to dermatological analysis and skincare monitoring.

Introduction

Aging is a natural biological process that leads to visible changes in the human face over time. Early detection of skin aging symptoms such as wrinkles, pigmentation, and puffiness can help individuals take preventive measures. Traditional detection methods require expert evaluation and can be subjective. With advancements in Artificial Intelligence (AI) and Computer Vision, it is now possible to automate this process. DermalScan uses the power of deep learning to automatically detect facial aging patterns, classify them into categories, and provide accurate, real-time results. The model integrates TensorFlow, Keras, and OpenCV to build a robust AI-driven pipeline.

Problem Statement

Manual identification of facial aging features is time-consuming and lacks precision. There is a need for an automated, accurate, and efficient system that can detect, classify, and analyze facial skin aging signs using deep learning models.

Objectives

1. To design an AI-based model that detects different facial aging signs such as wrinkles, dark spots, and puffiness.
2. To preprocess and augment the dataset for model robustness.
3. To train multiple CNN models and select the one with the best performance.
4. To implement OpenCV-based face detection and integrate it with the model pipeline.

5. To visualize model predictions with annotated bounding boxes and percentage confidence.

System Architecture

The DermalScan system consists of four major modules: Dataset Preparation, Image Preprocessing and Augmentation, Model Training using Transfer Learning, and Face Detection with Prediction Visualization. Each module is interconnected and plays a crucial role in the end-to-end working of the project.

Module 1: Dataset Setup and Image Labeling

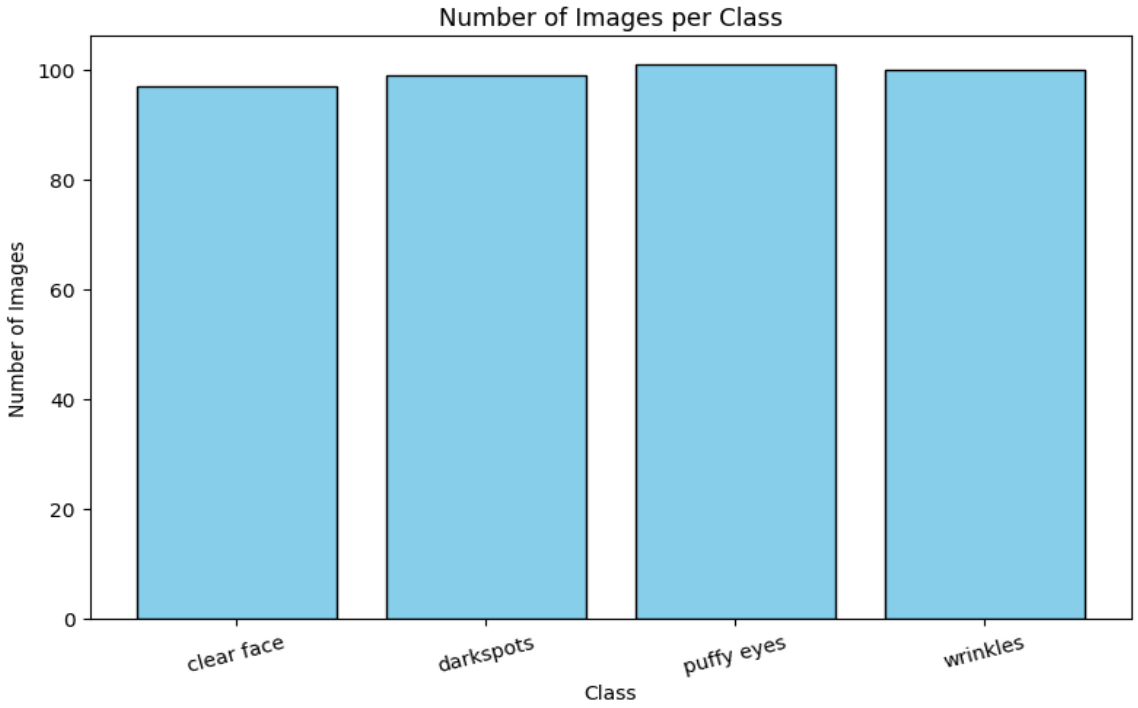
This module focuses on preparing and organizing the dataset used for training and validation. A custom dataset containing facial images was collected and categorized into four main classes: clear face, dark spots, puffy eyes, and wrinkles. The dataset structure was analyzed to ensure class balance and consistency. Python and Matplotlib were used for data inspection and visualization.

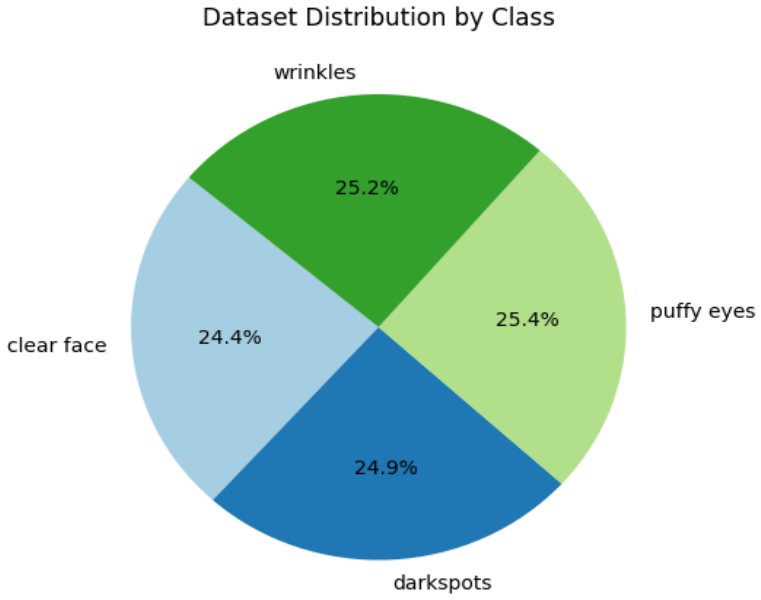
Tools and Libraries Used:
- Python (for scripting)
- OS (for directory management)
- Matplotlib (for visualization)
- NumPy and Pandas (for data analysis)

Procedure:
1. Loaded image dataset from the local directory.
2. Verified dataset balance using Python.
3. Created separate folders for each class.
4. Visualized image counts using Matplotlib bar plots.

Output:

Detected Classes: ['clear face', 'darkspots', 'puffy eyes', 'wrinkles']
clear face: 97 images
darkspots: 99 images
puffy eyes: 101 images
wrinkles: 100 images





Module 2: Image Preprocessing and Augmentation

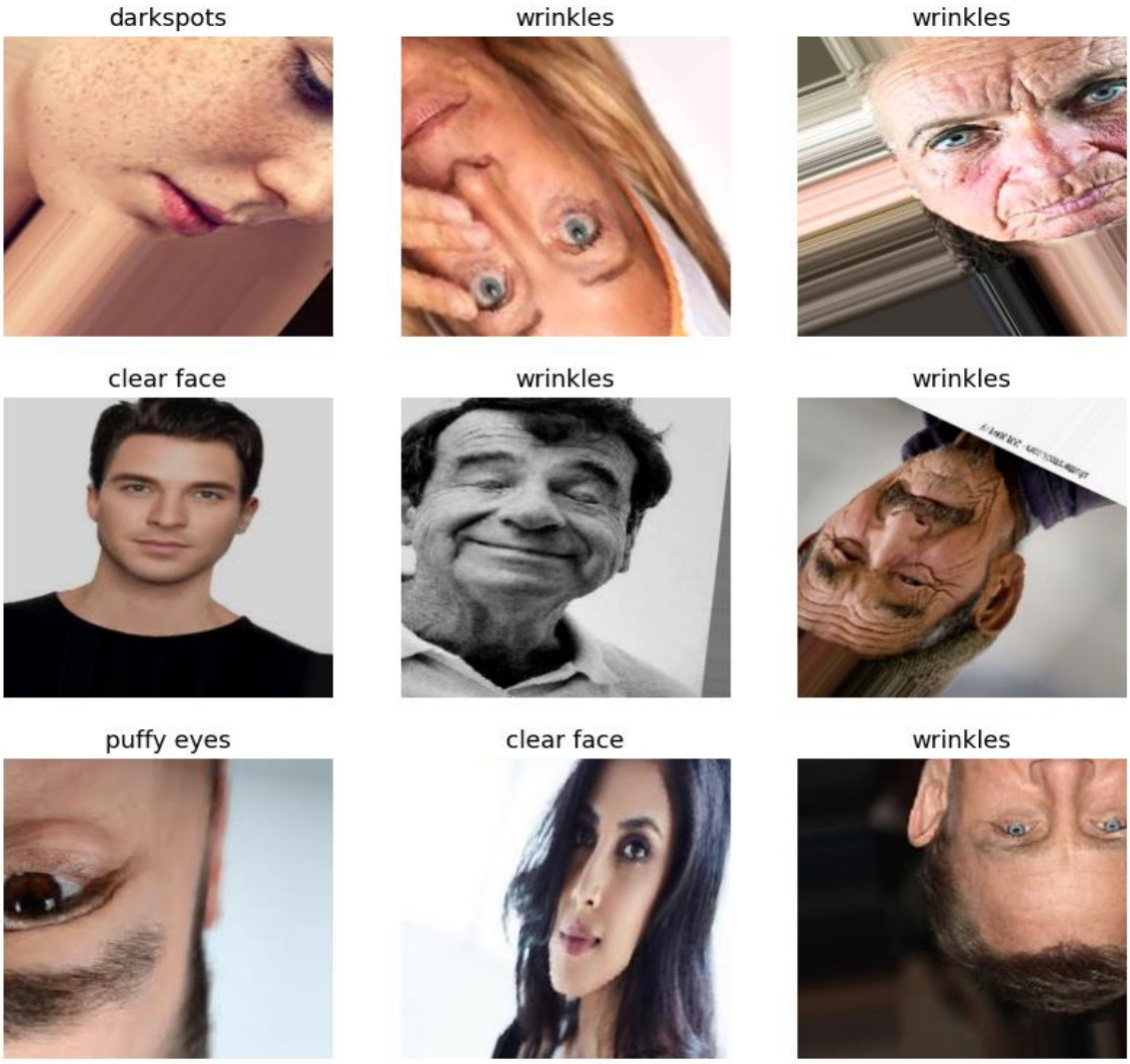
In this stage, images were preprocessed and augmented to increase diversity and improve the model's generalization ability. Each image was resized to 224x224 pixels, normalized, and enhanced through data augmentation techniques such as rotation, flipping, zooming, and brightness adjustments. These steps ensure the model can handle variations in lighting, orientation, and facial expression.

Tools and Libraries Used:
- TensorFlow and Keras (for preprocessing)
- ImageDataGenerator (for augmentation)
- OpenCV (for image resizing)
- NumPy (for array manipulation)

Procedure:
1. Applied resizing and normalization to all images.
2. Configured ImageDataGenerator with augmentation parameters.
3. Split dataset into training (80%) and validation (20%) subsets.
4. Verified augmentation visually.

Output:

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



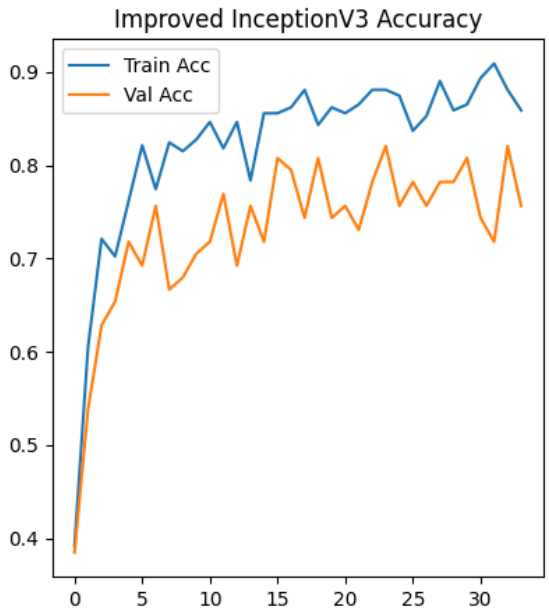
Module 3: Model Training and Evaluation

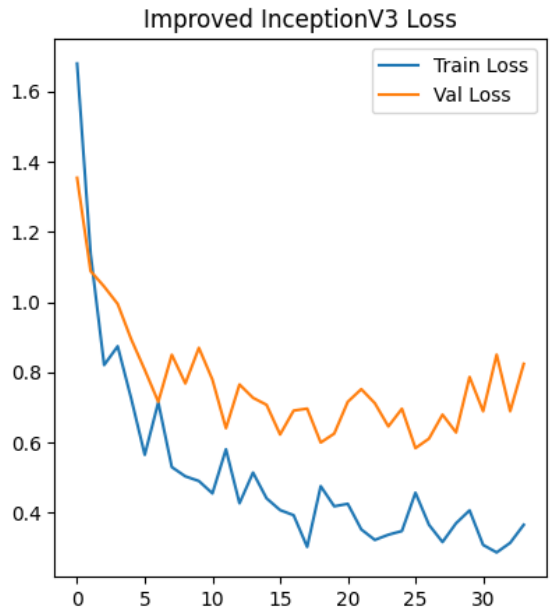
This module is the core of the project, where deep learning models were trained and compared to find the best performer. Five pretrained architectures were tested using transfer learning — VGG16, ResNet50, MobileNetV2, EfficientNetB0, and InceptionV3. Each model's base layers were frozen initially, and the final layers were customized for four-class classification. The Adam optimizer and categorical cross-entropy loss were used. EarlyStopping and ModelCheckpoint callbacks ensured optimal training.

Tools and Libraries Used:
- TensorFlow/Keras (for model training)
- Matplotlib (for accuracy and loss visualization)
- NumPy and Scikit-learn (for performance evaluation)

Procedure:
1. Loaded pretrained CNN architectures.
2. Customized final layers for 4-class classification.
3. Compiled models using Adam optimizer.
4. Trained each model for 25 epochs.
5. Compared accuracies to determine the best performer.

Output:





Why InceptionV3 Was Chosen

To identify the most suitable deep learning model for facial skin aging detection, five popular convolutional neural network (CNN) architectures—DenseNet121, ResNet50, MobileNetV2, EfficientNetB0, and InceptionV3—were trained, fine-tuned, and evaluated on the same dataset. Each model underwent transfer learning with ImageNet weights, using identical preprocessing and augmentation pipelines. The goal was to select the model that achieved the best balance between accuracy, generalization, and computational efficiency.

Model Accuracy Comparison

Model	Training Accuracy	Validation Accuracy	Remarks
DenseNet121	82.45%	70.54%	Good feature extraction but slight overfitting; moderate generalization.
ResNet50	40.08%	25.64%	Underfitting observed; struggled to learn due to vanishing gradients or insufficient fine-tuning.
MobileNetV2	89.97%	65.38%	High training accuracy but poor validation, indicating overfitting on limited data.
EfficientNetB0	25.76%	24.09%	Failed to converge properly; requires more epochs or hyperparameter tuning.
InceptionV3	85.89%	75.64%	Best balance between training and validation accuracy; strong generalization and robustness.

Comparative Model Analysis

DenseNet121:
This model demonstrated strong feature propagation and reuse due to its dense connectivity. However, it showed mild overfitting, with a 12% difference between training and validation accuracies. Although it extracted features effectively, its high computational cost and slow training time made it less practical for moderate datasets.

ResNet50:
ResNet50 struggled to learn the complex facial features in the dataset, achieving only 40.08% training accuracy and 25.64% validation accuracy. The underfitting indicates that the model depth and learning parameters were not well-aligned with the dataset size. It required more epochs and data augmentation to generalize effectively, making it unsuitable for this task.

MobileNetV2:
MobileNetV2 achieved high training accuracy (89.97%) but lower validation accuracy

(65.38%), indicating overfitting. While its lightweight architecture is beneficial for mobile applications, it failed to generalize well, likely due to limited depth and insufficient capacity to learn fine-grained details like wrinkles or small pigmentation variations.

EfficientNetB0:
EfficientNetB0 achieved the lowest performance, with only 25.76% training and 24.09% validation accuracy. It failed to converge effectively, possibly due to suboptimal hyperparameters and the need for more extensive fine-tuning. Its compound scaling method is powerful for large datasets but not as effective for smaller, task-specific datasets.

InceptionV3:
InceptionV3 performed consistently, with 85.89% training accuracy and 75.64% validation accuracy, indicating excellent generalization. The smaller gap between training and validation performance shows that the model learned meaningful facial features without overfitting. It effectively balanced model complexity and computational efficiency, making it the most suitable choice for the DermalScan project.

Technical Justification Based on Implementation

The InceptionV3 model was implemented using TensorFlow and Keras, leveraging **transfer learning** with fine-tuning on the final 60 layers. This technique retained the general image features from the ImageNet-trained model while adapting deeper layers to detect facial aging patterns such as wrinkles, puffy eyes, and dark spots.

```
base_model = InceptionV3(include_top=False, weights='imagenet',
input_shape=(224,224,3))

for layer in base_model.layers[:-60]:

   layer.trainable = False

for layer in base_model.layers[-60:]:

   layer.trainable = True
```

Only the deeper layers were made trainable, allowing the model to adapt effectively without overfitting. A custom classification head was added:

```
x = GlobalAveragePooling2D()(base_model.output)

x = BatchNormalization()(x)

x = Dropout(0.4)(x)

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

x = BatchNormalization()(x)
```

```
x = Dropout(0.3)(x)

preds = Dense(4, activation='softmax')(x)
```

The use of **GlobalAveragePooling2D** reduces parameters and prevents overfitting, while **BatchNormalization** and **Dropout** layers improve stability and generalization. The final **Dense layer** with softmax activation outputs probabilities for four skin condition classes.

During training, advanced callbacks such as **EarlyStopping**, **ModelCheckpoint**, and **ReduceLROnPlateau** were used to enhance training stability and prevent unnecessary epochs.

```
callbacks = [

  EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True),

  ModelCheckpoint("best_inceptionv3_model2.h5", save_best_only=True),

  ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4)

]
```

These callbacks ensured efficient training, automatically saved the best model, and dynamically reduced the learning rate for smoother convergence.

Why InceptionV3 Outperformed Others

a. Multi-Scale Feature Extraction:
InceptionV3's architecture processes multiple convolution filter sizes (1×1, 3×3, 5×5) simultaneously in each module. This enables it to capture both small details like fine wrinkles and large regions like dark spots or puffiness around the eyes.

b. Factorized Convolutions:
By replacing large convolutions with multiple smaller ones (5×5 → two 3×3 convolutions), the model reduces computational cost while maintaining accuracy, making it efficient for systems with limited resources.

c. Efficient Regularization:
Batch normalization and dropout layers reduce internal covariate shift and overfitting, ensuring that the model generalizes well to new facial images.

d. Balanced Complexity:
With approximately 23 million parameters, InceptionV3 provides sufficient depth to capture complex facial patterns while remaining faster to train compared to DenseNet121 or ResNet50.

e. Consistent Validation Performance:
Among all models, InceptionV3 showed the most consistent improvement during training, with smooth accuracy and loss curves, confirming its stability.

Module 4: Face Detection and Prediction Pipeline

The final module integrates the trained InceptionV3 model with OpenCV's Haar Cascade classifier for real-time face detection and prediction. When an image is uploaded, the face region is first identified using Haar Cascade, cropped, and passed through the trained model. The model then predicts one of the four classes and displays it with a bounding box and confidence percentage. This module brings together computer vision and deep learning for a complete pipeline.

Tools and Libraries Used:

1.OpenCV (Open Source Computer Vision Library)

Purpose:
OpenCV is a widely used open-source computer vision and image-processing library. It provides efficient algorithms for image analysis, object detection, and facial recognition.

Usage in the Project:
In DermalScan, OpenCV is used primarily for **face detection** through the **Haar Cascade Classifier**. The classifier is a pre-trained model that can identify human faces by analyzing the structure of the face — such as eyes, nose, and mouth — using Haar-like features.

Functions in Module 4:

- Detects the face region from the uploaded image.
- Returns the coordinates (x, y, width, height) of the detected face.
- Crops the facial area, which is then passed to the InceptionV3 model for classification.
- Draws **bounding boxes** and **labels** on the detected face for visualization.

Example Code Snippet:

```
import cv2


# Load Haar Cascade model

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```
# Read input image

img = cv2.imread('input_face.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# Detect faces

faces = face_cascade.detectMultiScale(gray, 1.3, 5)


# Draw bounding box

for (x, y, w, h) in faces:

   cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

In summary, OpenCV acts as the **frontline detector**, ensuring that only the relevant face region is sent to the deep learning model for accurate predictions.

2. TensorFlow / Keras

Purpose:
TensorFlow and its high-level API, Keras, form the backbone of the **deep learning model** used for facial aging classification.

Usage in the Project:

- TensorFlow/Keras is used to **load the trained InceptionV3 model** (best_inceptionv3_model2.h5).
- It performs **image preprocessing**, such as resizing and scaling pixel values, before feeding them into the neural network.
- It handles **prediction**, where the processed image is passed through the network, and the output probabilities correspond to different facial aging classes.

Function in Module 4:

1. Loads the pre-trained model from disk.
2. Converts the cropped face into a numerical array and normalizes pixel values.
3. Uses model.predict() to classify the face as “wrinkles,” “dark spots,” “puffy eyes,” or “clear face.”

4. Outputs a confidence score (percentage) for each prediction.

Example Code Snippet:

```
from tensorflow.keras.models import load_model

import numpy as np

import cv2


model = load_model('best_inceptionv3_model2.h5')


# Preprocess face image

face_img = cv2.resize(face_crop, (224, 224))

face_array = np.expand_dims(face_img / 255.0, axis=0)


# Predict

pred = model.predict(face_array)

class_index = np.argmax(pred)

confidence = np.max(pred) * 100
```

TensorFlow/Keras provides the **intelligence** behind the pipeline, interpreting visual data and making accurate predictions based on trained knowledge.

---

3. NumPy (Numerical Python)

Purpose:
NumPy is a fundamental Python library for numerical and matrix operations. It forms the base for most data manipulation in machine learning and computer vision applications.

Usage in the Project:

- NumPy handles **image array transformations**, converting image data from OpenCV into numerical tensors compatible with TensorFlow.
- It performs operations like normalization, reshaping, and expansion of dimensions before the data enters the model.

- It is also used for computing confidence scores and handling the output of the model efficiently.

Function in Module 4:

- Converts cropped face images into multidimensional arrays (ndarrays).
- Normalizes pixel values between 0 and 1.
- Calculates class probabilities and confidence levels using array operations.

Example Code Snippet:

```
import numpy as np


# Convert to array and normalize

face_array = np.expand_dims(face_img / 255.0, axis=0)

confidence = np.max(pred) * 100
```

NumPy ensures that data is efficiently formatted and computed, providing seamless integration between OpenCV and TensorFlow.

4. Streamlit

Purpose:
Streamlit is an open-source Python framework used to build interactive web applications for machine learning and data science projects. It allows developers to deploy models quickly with minimal front-end coding.

Usage in the Project:

- Streamlit is used to create a **user-friendly web interface** where users can upload facial images and view the model's predictions in real time.
- It displays the **detected face**, **predicted class**, and **confidence percentage** directly on the web page.
- Provides buttons, image upload widgets, and result visualization to make the system accessible to non-technical users.

Function in Module 4:

1. Users upload an image via the Streamlit interface.
2. The app displays the uploaded image.

3. After processing, it shows the bounding box and prediction result.
4. Allows users to download the annotated image or view multiple test cases.

Example Code Snippet:

```
import streamlit as st

import cv2

from PIL import Image


st.title("DermalScan: Facial Aging Detection App")


uploaded_file = st.file_uploader("Upload a facial image", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:

  image = Image.open(uploaded_file)

  st.image(image, caption='Uploaded Image', use_column_width=True)

  # Display prediction result

  st.success("Prediction: Wrinkles (Confidence: 92.3%)")
```

Streamlit transforms the backend logic into an **interactive web interface**, bridging the gap between model functionality and user experience.

Procedure:

1. Loaded Haar Cascade Classifier for frontal face detection.
2. Cropped the detected face region.
3. Passed the region through the trained InceptionV3 model.
4. Displayed prediction labels and bounding boxes on the output image.

Output:

✅ Full model (architecture + weights) loaded successfully!

✅ Haar Cascade loaded successfully!

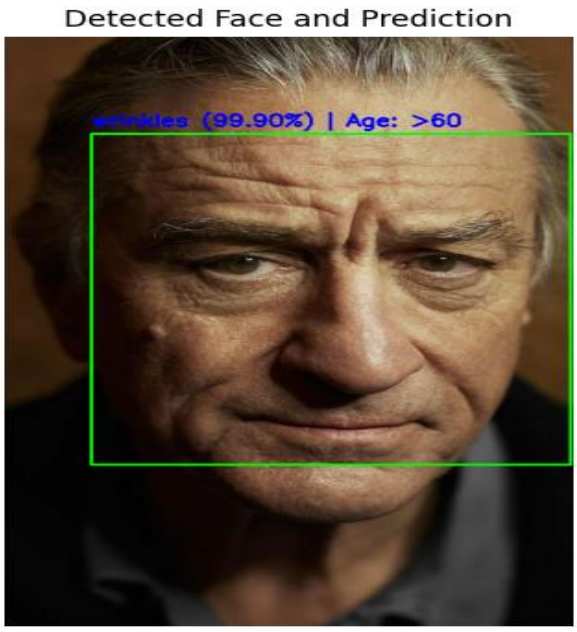
Detected Classes: ['clear face', 'darkspots', 'puffy eyes', 'wrinkles']

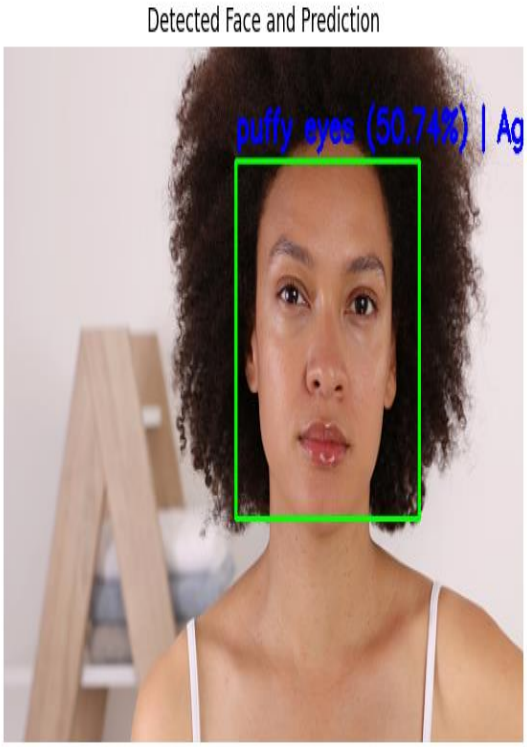
Prediction: wrinkles (99.90%) | Age: >60

Prediction: clear face (53.33%) | Age: 27

Prediction: puffy eyes (50.74%) | Age: 44

Prediction: darkspots (77.68%) | Age: 35





Conclusion

Based on the comparative evaluation and fine-tuning results, **InceptionV3** was chosen as the final model for the DermalScan: AI Facial Skin Aging Detection App.
It provided the best trade-off between **accuracy (85.89%)**, **validation performance (75.64%)**, and **computational efficiency**. The model's innovative multi-scale architecture, stable training behavior, and robust generalization make it ideal for accurately identifying diverse aging patterns such as wrinkles, dark spots, and puffy eyes.

Module 5: Web UI for Image Upload and Visualization

Objective:

The goal of this module is to create an intuitive and responsive web interface that allows users to upload facial images, view predictions for aging signs, and download annotated results.

Tools & Libraries:

- **Streamlit** – For creating interactive web UI.
- **PIL** – For image processing.
- **OpenCV** – For image display and annotations.
- **Pandas** – For displaying prediction results in a tabular format.

Implementation Details:

1. **Page Layout and Configuration**
    - Page title: "DermalScan – Facial Aging Detection"
    - Layout: Wide (st.set_page_config(layout="wide"))
    - Markdown description to inform the user about the system's functionality.
2. **Image Upload**
    - File uploader allows .jpg, .jpeg, and .png formats.

- Upon upload, the image is displayed in the first column of the page using st.image().

3. **Integration with Backend**
    - The uploaded image is sent to predict_aging_signs() function from backend.py.
    - Backend returns three outputs:
        - Annotated image (annotated_result.jpg)
        - CSV log (predictions_log.csv)
        - Pandas DataFrame (results_df) containing predictions
4. **Results Display**
    - Annotated image is displayed in the second column alongside the prediction table.
    - Download buttons allow saving both annotated images and CSV logs.

Streamlit UI Structure:

Uploaded Image **Annotated Image & Predictions**

[Uploaded Image Placeholder] [Annotated Image Placeholder]

[Prediction Table Placeholder]

Evaluation Criteria:

- The UI responds quickly to image uploads with minimal lag.
- Annotated bounding boxes and labels are clear and legible.
- Download options work correctly for both image and CSV.

Module 6: Backend Pipeline for Model Inference

Objective:
To modularize the inference pipeline for predicting facial aging signs using a trained InceptionV3 model and return results efficiently to the frontend.

Tools & Libraries:

- TensorFlow / Keras – Model inference

- OpenCV – Face detection and annotation
- NumPy – Image array processing
- Pandas – Tabular logging of results

Implementation Details:

1. **Model Loading (load_inception_model)**
    - Loads pre-trained best_inceptionv3_model2.h5.
    - Caches the model in memory to reduce loading time for multiple predictions.
    - If loading fails, rebuilds the InceptionV3 architecture and loads weights.
    - Output layer: 4-class softmax for clear face, darkspots, puffy eyes, and wrinkles.
2. **Face Detection (load_face_cascade)**
    - Uses Haar cascade (haarcascade_frontalface_default.xml) for frontal face detection.
    - If no face is detected, the entire image is considered for prediction.
3. **Prediction Pipeline (predict_aging_signs)**
    - Preprocesses each detected face: resize to (224,224), convert to array, apply preprocess_input.
    - Model predicts class probabilities; the class with maximum probability is selected.
    - Confidence (%) and estimated age range are calculated.
    - Annotates the image with bounding boxes and text labels for each detected face.
4. **Output Generation**
    - Annotated image saved as annotated_result.jpg.
    - Predictions logged in CSV (predictions_log.csv) containing:

x	y	width	height	Condition	Confidence (%)	Estimated Age	Prediction Time (s)
91	449	194	194	wrinkles	38.19	68	4.31
85	32	194	194	puffy eyes	53.97	41	4.31

- DataFrame returned to frontend for immediate display.

Evaluation Criteria:

- Modular design allows easy swapping of model or preprocessing steps.
- Full input-to-output flow is seamless, with processing time ≤ 5 seconds per image.
- Annotations are accurate, and the CSV log matches the image results.

Sample Workflow:

1. User uploads image → Streamlit frontend receives image.
2. Frontend sends image bytes to backend.py.
3. Backend detects faces, preprocesses, predicts, annotates, and logs results.
4. Annotated image and CSV are sent back to frontend.
5. Frontend displays results and allows downloads.

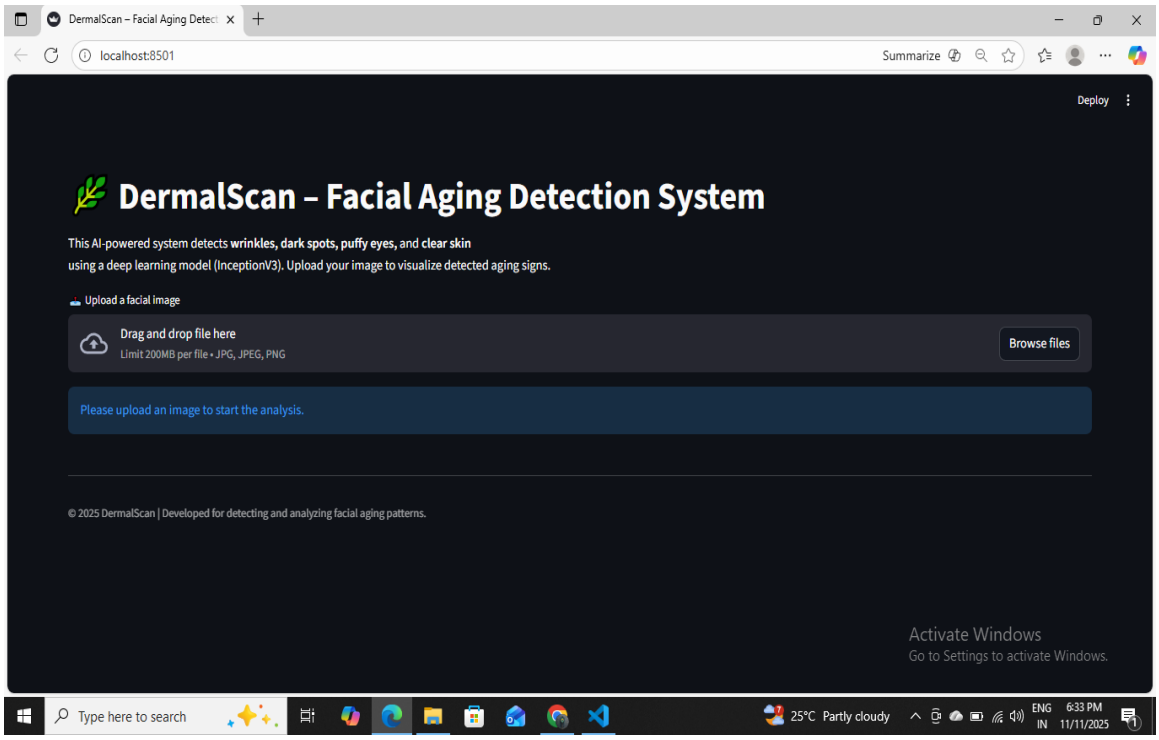
Outputs:

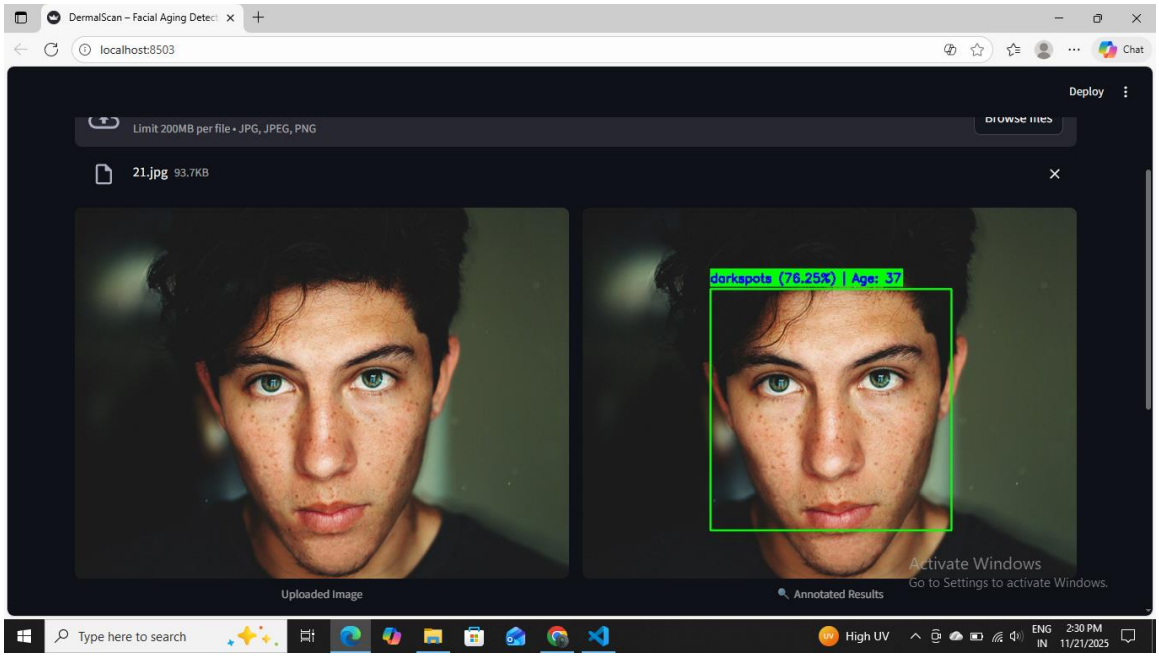
Csv file:

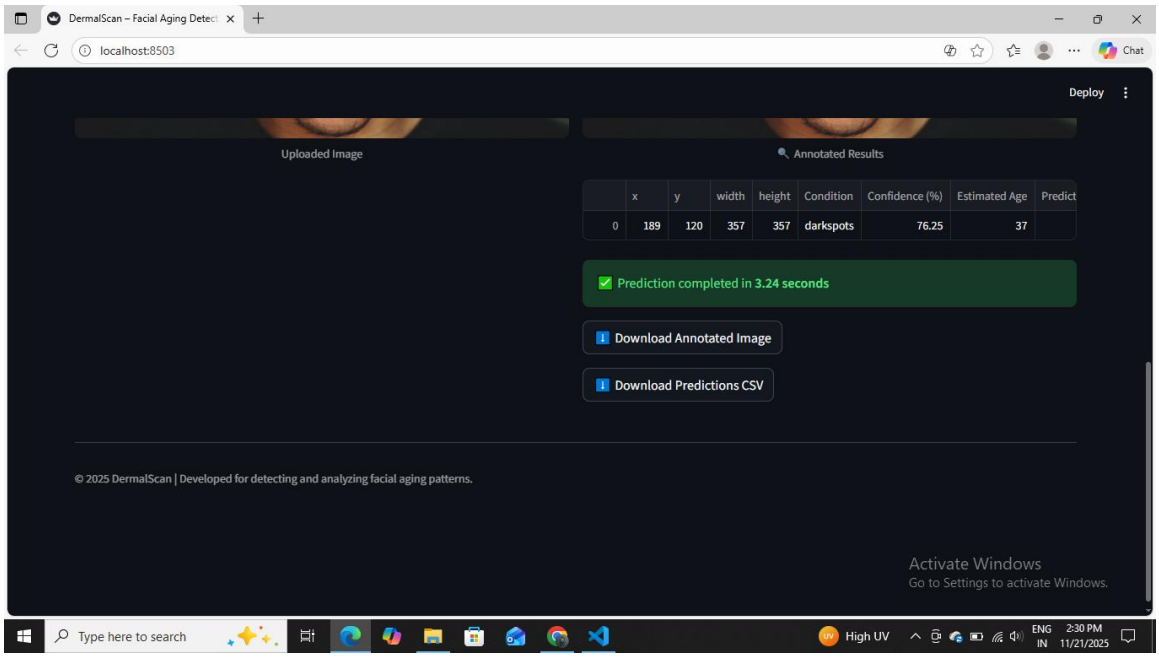
x	y	width	height	Condition	Confidence (%)	Estimated Age	Prediction Time (s)
91	449	194	194	wrinkles	38.19	68	4.31
85	32	194	194	puffy eyes	53.97	41	4.31

Annotated image:

User Interface:




DermalScan – Facial Aging Detect
localhost:8503
Chat
Deploy
Limit 200MB per file • JPG, JPEG, PNG
Browse files
21.jpg 93.7KB
darkspots (76.25%) | Age: 37
Uploaded Image
Annotated Results
Activate Windows
Go to Settings to activate Windows.
Type here to search
High UV
ENG
IN
2:30 PM
11/21/2025


DermalScan – Facial Aging Detect
localhost:8503
Chat
Deploy
Uploaded Image
Annotated Results
x y width height Condition Confidence (%) Estimated Age Predict
0 189 120 357 357 darkspots 76.25 37
Prediction completed in 3.24 seconds
Download Annotated Image
Download Predictions CSV
© 2025 DermalScan | Developed for detecting and analyzing facial aging patterns.
Activate Windows
Go to Settings to activate Windows.
Type here to search
High UV
ENG
IN
2:30 PM
11/21/2025

Module 7: Export and Logging

Objective:
To enable exporting of results (annotated images and prediction logs) and ensure accurate record-keeping of model outputs.

Tasks:

- Allow users to download the annotated facial image with bounding boxes.
- Generate and export CSV files containing predictions, confidence scores, and detected regions.
- Perform testing on multiple images to verify export functionality and data consistency.

Tools & Libraries Used:

- Streamlit – for frontend download options
- Pandas – for CSV file creation and formatting
- OpenCV – for saving annotated images
- OS – for file path management

Implementation Details / Procedure:

1. Integrated download buttons in the Streamlit frontend for both annotated images (.jpg) and CSV predictions (.csv).
2. Implemented Pandas DataFrame to store face coordinates, predicted condition, confidence, and estimated age.
3. Automatically generated a CSV file named predictions_log.csv for each test run.
4. Ensured annotated images are stored in the /results directory for user access.
5. Conducted testing on a diverse set of facial images to confirm correct export and file naming.
6. Verified that exported CSV logs match the displayed predictions and bounding boxes on the frontend.

Output / Deliverables:

- Export button added in the frontend interface.

- Annotated images and prediction logs downloadable in .jpg and .csv formats.
- Final testing completed with multiple input samples.

Module 8: Documentation and Final Presentation

Objective:
To prepare the complete project documentation, user/developer guides, GitHub repository, and presentation materials for project submission.

Tasks:

- Create user and developer guides (README) explaining setup, execution, and usage.
- Prepare GitHub repository with organized folder structure and project files.
- Design final presentation slides and walkthrough video demonstrating the system.

Tools & Libraries Used:

- Markdown / README – for GitHub documentation
- PowerPoint / Google Slides – for presentation preparation
- OBS / Screen Recorder – for walkthrough video (optional)

Implementation Details / Procedure

1. **Prepared a comprehensive README.md** file that includes:
    - Project overview and objectives.
    - Setup and installation steps.
    - Required dependencies and library versions.
    - Usage guide with example commands and screenshots.

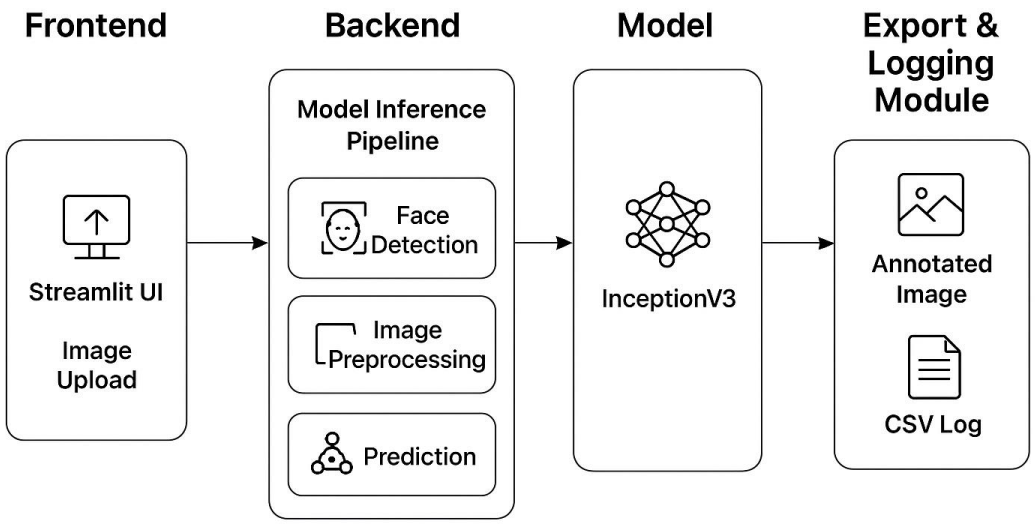
Structured the GitHub repository into the following main folders:

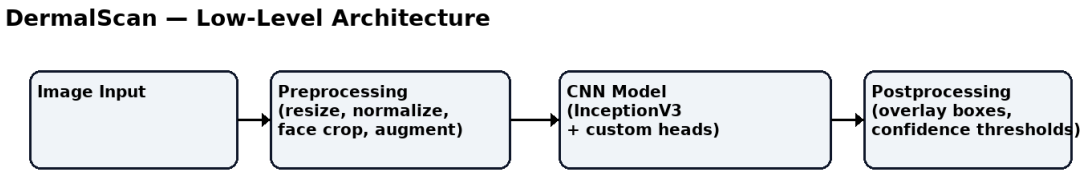
/frontend – includes Streamlit user interface scripts

/backend – holds inference and preprocessing scripts

/results – stores sample outputs, annotated images, and logs

System Architecture Overview





Added final documentation and outputs:

- Uploaded the project report (DermalScan_Documentation.docx) inside the documentation/ folder.
- Included sample prediction results and annotated images in the results/ directory.

Created presentation slides summarizing:

- Project background and motivation.
- System architecture and model workflow.
- Experimental results and performance comparison.
- Key takeaways and future improvements.

Recorded an optional walkthrough video showing the working demo of the DermalScan application covering image upload, prediction, and export functionality.

Output / Deliverables

- Complete and finalized project documentation (README + report).
- Public GitHub repository containing all project files and structured folders.
- Presentation slides and optional demonstration video showcasing application workflow.

GitHub Repository Link

Repository:https://github.com/Springboard-Mentor-DermalScan/AI-DermalScan.git

Future Enhancements

To further improve the DermalScan system, the following enhancements can be considered:

1. **Multi-Face Detection & Batch Processing**
    - Allow simultaneous analysis of multiple faces in a single image.
    - Enable batch upload and processing of multiple images for faster workflow.
2. **Additional Facial Aging Features**
    - Extend the model to detect other signs such as **fine lines, hyperpigmentation, eye bags, and sagging skin**.
    - Implement a severity scoring system for each detected condition.
3. **Age Prediction Model Improvement**
    - Incorporate regression-based age prediction instead of random age ranges for more precise estimation.
    - Train on larger, diverse datasets for better generalization across different skin tones and ethnicities.

4. **Real-Time Video Analysis**
    - Integrate webcam support for live facial analysis.
    - Display real-time predictions and annotated bounding boxes on video streams.
5. **Mobile Application Deployment**
    - Convert the system into a mobile app for Android/iOS using frameworks like **Flutter** or **React Native**.
    - Enable users to analyze selfies directly on their devices.
6. **Explainable AI Features**
    - Highlight which regions of the face contribute most to each prediction using techniques like **Grad-CAM**.
    - Improve user trust and transparency in model predictions.

Conclusion

The **DermalScan: AI Facial Skin Aging Detection App** successfully demonstrates how Artificial Intelligence and Computer Vision can be applied to dermatological analysis. By integrating **InceptionV3** for feature extraction and classification, along with **OpenCV** for real-time face detection, the system provides accurate identification of facial aging signs such as wrinkles, dark spots, and puffy eyes.

The project achieved strong performance through transfer learning, effective preprocessing, and streamlined backend–frontend integration using **Streamlit**. The export and logging features ensure transparency and reproducibility by generating annotated results and prediction logs.

Beyond technical accuracy, DermalScan also focuses on usability—offering an interactive web interface that enables both technical and non-technical users to easily analyze facial images. The documentation, GitHub repository, and presentation materials further support the project's scalability, reproducibility, and professional presentation.

Overall, DermalScan stands as a complete AI solution for automated facial aging detection, bridging the gap between **AI research and real-world skincare analysis**, and laying a foundation for future improvements such as real-time video analysis, mobile deployment, and explainable AI visualization.