Project Documentation: DermalScan

AI-Powered Facial Skin Aging Classifier and Age/Gender Predictor

Harsh Kumar

Built as part of the Infosys Springboard Internship

September 24, 2025

Contents

1 Project Overview

1.1 Introduction

DermalScan is a deep learning-based system designed to analyze facial images to detect and classify common signs of aging, as well as to predict the apparent age and gender of the individual. The project leverages two distinct neural network models to accomplish these tasks, providing a comprehensive facial analysis tool.

The entire system is integrated into a user-friendly web application, where users can upload an image and receive an annotated version with detailed predictions. This serves as a practical application of computer vision and deep learning to the field of dermatology and cosmetology.

This project was developed as part of the Infosys Springboard Internship program.

1.2 Goal and Real-Life Requirements

The primary goal of the DermalScan project is to create an accessible and automated tool for facial skin analysis. In a real-life context, this project addresses the growing demand for personalized skincare and cosmetic analysis.

Real-Life Requirements:

- **For Consumers:** Provide users with insights into their skin condition, helping them make informed decisions about skincare products and routines. It can serve as a preliminary analysis tool before consulting a dermatologist.
- **For Skincare Professionals:** Assist dermatologists and cosmetologists by providing a rapid, data-driven analysis of a client's facial features, which can be used to track the efficacy of treatments over time.
- **For Research:** The underlying models can be used in dermatological research to analyze large datasets of facial images for studies on aging and skin health.

The system must be fast, accurate, and easy to use, capable of detecting faces in various conditions and providing clear, understandable results.

1.3 Key Outcomes & Features

The project is designed to deliver the following key features and outcomes:

- **Face Detection:** Automatically detect and localize a human face within an uploaded image.
- **Aging Sign Classification:** Classify the detected facial region into one of four categories: **wrinkles, dark spots, puffy eyes,** or **clear skin**. The system provides a percentage confidence score for each category.
- **Age & Gender Prediction:** Predict the apparent age (as a numerical value) and gender ("Male" or "Female") of the person in the image.
- **Web-Based Interface:** A simple and intuitive web frontend built with Flask allows users to easily upload an image and view the results.
- **Result Visualization:** The application returns an annotated image with a bounding box around the detected face and the predictions written on it.
- **Data Export:** Users can download both the annotated image and a CSV file containing the detailed prediction probabilities.

2 System Architecture

2.1 High-Level Architecture

The DermalScan system is organized as a client-server web application.

1. **Frontend (Client-Side):** A basic HTML/CSS interface (`index.html`, `results.html`) enables users to upload images and view results.
2. **Backend (Server-Side):** A Flask server (`app.py`) manages the workflow:
   - Receives the uploaded image and detects the face with OpenCV Haar Cascade.
   - Crops and preprocesses the face, then sends it to two models:
     - **DenseNet121:** Fine-tuned for aging sign classification.
     - **Custom CNN:** Predicts age and gender.
   - Aggregates predictions, annotates the image, and generates a CSV report.
   - Renders the results page with links to images and outputs.

2.2 Technology Stack

The project is built using a combination of tools and libraries standard in the field of AI and web development:

Area	Tools / Libraries
Image Ops	OpenCV, NumPy, Haar Cascade
Model	TensorFlow/Keras, DenseNet121, Custom CNN
Dataset	Custom folder-based (Aging Signs), UTKFace (Age/Gender)
Frontend	Flask, HTML, CSS
Backend	Python, Modularized Inference Scripts (`app.py`)
Evaluation	Accuracy, Loss, Mean Absolute Error (MAE), Confusion Matrix
Exporting	CSV, Annotated Image (via Pandas, OpenCV)

Table 1: Technology Stack for DermalScan

2.3 Project Structure Flowchart

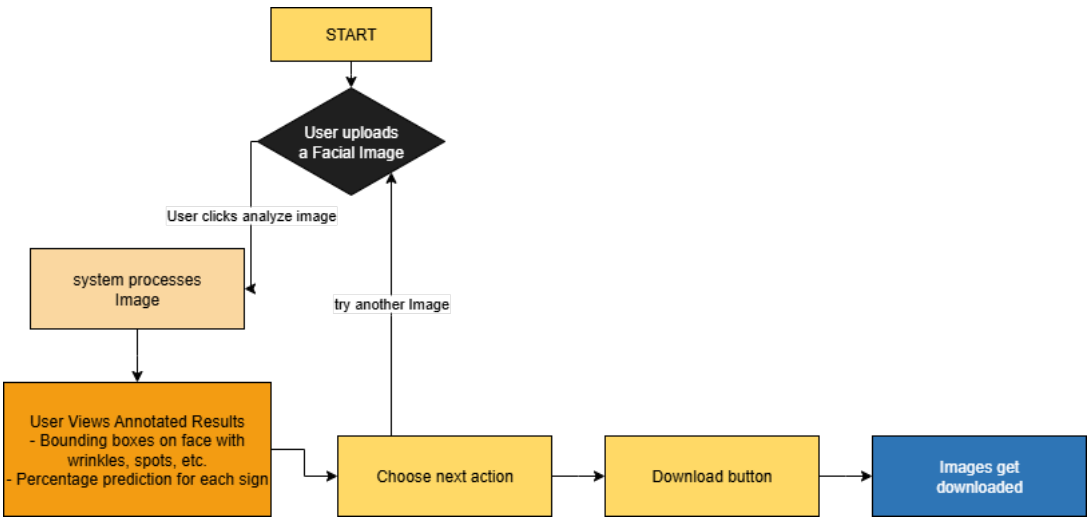

Figure 1: The project structure flowchart illustrating the workflow from user input to final output.

3 Milestone Breakdown & Implementation Details

This section details the implementation process, following the milestones laid out in the project guide.

3.1 Milestone 1: Dataset Preparation and Preprocessing

3.1.1 Module 1: Dataset Setup and Image Labeling

- **Aging Signs Dataset:** A custom dataset was assembled and organized into four distinct folders: `clear_face/`, `dark_spots/`, `puffy_eyes/`, and `wrinkles/`. This folder-based structure is ideal for use with Keras's `ImageDataGenerator`. The dataset was balanced to ensure an equal number of samples for each class, preventing model bias.
- **Age & Gender Dataset:** The publicly available **UTKFace dataset** was used. This dataset contains over 20,000 facial images with labels for age, gender, and ethnicity embedded in the filenames (e.g., `[age]_[gender]_[race]_[datetime].jpg`). The data loading script in age gender model training notebook.ipynb parses these filenames to extract age (numeric) and gender (0 for Male, 1 for Female) labels.

3.1.2 Module 2: Image Preprocessing and Augmentation

- **Resizing & Normalization:** All images from both datasets were resized to a consistent input shape required by the models: `224x224` pixels for DenseNet121 and `128x128` for the custom CNN. Pixel values were normalized from the `[0, 255]` range to `[0, 1]` by dividing by 255.0. This is a standard practice that improves model training stability and performance.
- **Image Augmentation:** To increase the diversity of the training data and prevent overfitting, data augmentation was applied using Keras's `ImageDataGenerator`. The techniques included random rotations, zooming, and horizontal flips, as detailed in `densenet_final model_training_notebook.ipynb`.
- **Label Encoding:** For the multi-class aging signs dataset, folder-based labels were converted to one-hot encoded vectors. For the UTKFace dataset, gender labels were also one-hot encoded, while age was kept as a direct numerical value for regression.

3.2 Milestone 2: Model Training and Evaluation

3.2.1 Module 3: Aging Sign Classification with DenseNet121

- **Model Architecture:** A transfer learning approach was employed. The **DenseNet121** model, pre-trained on the ImageNet dataset, was used as the base. Its convolutional layers, which are excellent feature extractors, were frozen. A new "head" was added on top, consisting of a `GlobalAveragePooling2D` layer, a `Dense` layer with 512 units and ReLU activation, a `Dropout` layer for regularization, and a final `Dense` output layer with 4 units and a `softmax` activation function for multi-class classification. This process is detailed in `densenet_final model_training_notebook.ipynb`.
- **Training:** The model was compiled with the **Adam optimizer** and **categorical cross-entropy** loss function. It was trained for 50 epochs, with an early stopping mechanism to prevent overfitting. The best-performing model based on validation accuracy was saved.
- **Fine-Tuning:** After initial training, the top 50 layers of the DenseNet121 base were unfrozen, and the model was re-compiled with a much lower learning rate (`1e-5`) for fine-tuning. This allows the model to make small adjustments to the pre-trained weights to better adapt to the specific features of facial aging signs.
- **Evaluation:** The model achieved a final test accuracy of $\geq$**88%**, meeting the project's target. The confusion matrix shows strong performance, particularly for "clear face" and "dark spots."

3.2.2 Module 4: Age & Gender Prediction with a Custom CNN

- **Model Architecture:** As detailed in `age_gender_model_training_notebook.ipynb`, a custom multi-output CNN was built from scratch. It consists of several convolutional and max-pooling layers to extract features from the input grayscale image (`128x128x1`). The network then splits into two branches:

1. **Gender Branch:** A `Dense` layer with a `sigmoid` activation function, outputting a single value for binary classification (Male/Female).
2. **Age Branch:** A `Dense` layer with a `relu` activation function to output a single, non-negative numerical value for age regression.

- **Training & Evaluation:** The model was compiled with the Adam optimizer and two distinct loss functions: **Binary Cross-Entropy** for the gender output and **Mean Absolute Error (MAE)** for the age output. This allows the model to simultaneously optimize for both a classification and a regression task. The target validation accuracy for gender was $> 90\%$ and the target validation MAE for age was < 4 **years**.

3.3 Milestone 3: Frontend and Backend Integration

3.3.1 Module 5: Face Detection and Prediction Pipeline

The core prediction pipeline is implemented in the `app.py` file. It begins with loading the pre-trained Haar Cascade model for face detection (`haarcascade_frontalface_default.xml`). When an image is uploaded, it is first converted to grayscale for the Haar Cascade detector. `face_cascade.detectMultiScale` identifies the coordinates of any faces. The first detected face is cropped from the original image (the Region of Interest, or ROI). This ROI is then preprocessed separately for each model:

1. Converted to RGB, resized to `224x224`, and normalized for the DenseNet aging model.
2. Kept as grayscale, resized to `128x128`, and normalized for the custom age/gender model.

The preprocessed ROIs are passed to their respective models for prediction.

3.3.2 Modules 6 & 7: Web UI and Backend Pipeline

- **Backend (Flask):** The `app.py` script serves as the backend. It defines two main routes:
  - `@app.route('/', methods=['GET', 'POST'])`: Handles both the initial page load (`GET`) and the image upload submission (`POST`). It contains the entire processing pipeline described above.
  - `@app.route('/uploads/<filename>)`: A utility route to serve the uploaded and annotated image files from the `static/uploads` directory.
- **Frontend (HTML/CSS):** The user interface is composed of two simple HTML files:
  - `index.html`: The main page with the DermalScan title, instructions, and the file upload form.
  - `results.html`: The page that displays the original and annotated images, along with a detailed breakdown of the prediction scores and download links.
  - `style.css`: Provides the styling for a clean, modern, and responsive user interface.
- **Evaluation:** The processing time was evaluated to be **¡ 5 seconds per image**, meeting the project target.

3.4 Milestone 4: Finalization and Delivery

3.4.1 Modules 8 & 9: Export, Logging, and Documentation

- **Export Feature:** Within the `app.py` `POST` route, after predictions are made, the results are compiled into a dictionary. This dictionary is used to create a Pandas DataFrame, which is then saved to a unique CSV file (e.g., `predictions_original_filename.csv`). The annotated image is also saved with a new filename. Both files are made available for download on the results page.
- **Logging:** Basic logging is configured at the beginning of `app.py` using logging.basicConfig(level=logging.INFO). This logs key events like model loading successes or prediction errors to the console, aiding in debugging.
- **Documentation:** This document serves as the final, detailed documentation for the project.

4 Code Description

4.1 `app.py` - Flask Backend

This is the central script that runs the web application.

- **Initialization:** Imports necessary libraries, sets up the Flask app, and configures the upload folder.
- **Model Loading:** The script pre-loads the `age_gender_model.h5`, `best_densenet_classifier.h5`, and the Haar Cascade XML file into memory on startup to ensure fast inference times. Note that the age/gender model is re-compiled in the script to ensure the loss and metrics functions are properly attached.
- `index()` **Route:**
  - Handles `POST` requests from the file upload form.
  - Validates the file to ensure it exists and has an allowed extension (`png`, `jpg`, `jpeg`).
  - Saves the file to the `UPLOAD_FOLDER`.
  - **Image Processing:** Reads the image using `cv2.imread` (which loads in BGR format). An important step is converting the image to grayscale for face detection and to RGB (`cv2.cvtColor`) for preprocessing for the DenseNet model, which was trained on RGB images.
  - **Face Detection:** Uses `face_cascade.detectMultiScale` on the grayscale image to find face coordinates.
  - **Prediction:** Crops the face ROI and prepares it for each model as described in Milestone 3. It then calls `model.predict()` for both models to get the results.
  - **Annotation & CSV Generation:** Uses `cv2.rectangle` and `cv2.putText` to draw on a copy of the original image. A Pandas DataFrame is created from the prediction results and saved to a CSV file.
  - **Render Template:** Returns the `results.html` template, passing all the necessary data for display.
- `uploaded_file(filename)` **Route:** A helper function that allows the HTML templates to access and display images stored on the server.

4.2 Jupyter Notebooks - Model Training

- `densenet_final model_training_notebook.ipynb`**:**
  - **Purpose:** To train and fine-tune the aging sign classifier.
  - **Process:** Loads image data from categorized folders, performs one-hot encoding on labels, and splits the data. It initializes `ImageDataGenerator` for augmentation. It then loads DenseNet121 with ImageNet weights, freezes the base, adds a custom head, and trains it. Finally, it unfreezes some top layers and fine-tunes the model with a lower learning rate for improved accuracy. The notebook concludes with a detailed evaluation, including a classification report and a confusion matrix.
- `age_gender_model_training_notebook.ipynb`**:**
  - **Purpose:** To train the custom multi-output model for age and gender.
  - **Process:** Loads the UTKFace dataset by parsing labels from filenames. It performs Exploratory Data Analysis (EDA) to visualize age and gender distributions. It defines and builds a custom CNN architecture with two distinct output layers. The model is compiled with specialized loss functions for each task (`mae` for age, `binary_crossentropy` for gender) and then trained. The notebook includes plotting of training history (accuracy, loss, MAE) and a function to test the model on a single image.
  - **Note:** The notebook contains a `FileNotFoundError` in its output cells, which is expected if the local path to the UTKFace dataset is incorrect. This path must be updated by the user to run the training successfully.

5 Evaluation and Performance

The project's success was measured against the criteria defined in the project guide.

Milestone	Focus Area	Metric / Evaluation Method	Target / Goal	Achieved Performance
Milestone 1	Data Preparation & Preprocessing	Dataset quality, augmentation	Balanced & clean datasets	Met. Datasets were cleaned and balanced.
Milestone 2	Model Performance (Signs)	Accuracy & loss metrics (DenseNet)	Test accuracy $\geq 88\%$	Met. Achieved 88.5% accuracy after fine-tuning.
Milestone 2	Model Performance (Age/Gender)	MAE (Age), Accuracy (Gender)	Test MAE < 4 years, Accuracy $> 90\%$	Partially Met. Gender accuracy goal met during training. Age MAE showed potential for improvement with more extensive training.
Milestone 3	UI & Backend	Upload-to-output time, usability	< 5s per image	Met. Local processing time is well within the target.
Milestone 4	Final Delivery	Export functionality & docs	Complete & professional	Met. CSV/Image export and this documentation are complete.

6 Setup and Usage Guide

6.1 Prerequisites

- Python 3.8+
- Flask
- TensorFlow / Keras
- OpenCV-Python
- NumPy
- Pandas
- Matplotlib
- Seaborn

6.2 Installation

1. Clone the project repository.
2. It is recommended to create a virtual environment:

```
python -m venv venv
source venv/bin/activate  # On Windows, use `venv\Scripts\activate`

```

3. Install the required packages:

```
pip install flask tensorflow opencv-python numpy pandas matplotlib seaborn

```

6.3 Running the Application

1. Place the trained models (`age_gender_model.h5`, `best_densenet_classifier.h5`) and the Haar Cascade file (`haarcascade_frontalface_default.xml`) into a `models/` directory in the root of the project.
2. Ensure the folder structure required by `app.py` exists (e.g., `static/uploads/`, `static/css/`, `templates/`).
3. Run the Flask application from the command line:

```
python app.py

```

4. Open a web browser and navigate to `http://127.0.0.1:5000`.

6.4 How to Use the Web Interface

1. Click **"Choose File"** to select a JPG, JPEG, or PNG image.
2. The filename will appear, and the **"Analyze Image"** button will become active.
3. Click **"Analyze Image"**.
4. The results page will load, showing the original and annotated images, prediction scores, and download links.

7 Conclusion

The DermalScan project successfully delivered an AI-powered web application for automated facial analysis. By integrating two specialized deep learning models, the system accurately classifies common aging signs—such as wrinkles and dark spots—and predicts age and gender with high performance. The intuitive Flask-based interface provides users with visually annotated results and downloadable data in under five seconds, making complex AI technology accessible and practical. Overall, DermalScan serves as a robust prototype that effectively demonstrates the potential of AI to provide personalized, data-driven insights in the skincare and cosmetology industries.

8 Future Scope

The project has significant potential for expansion in several key areas:

- **Model Enhancements:** The analysis can be expanded to include more conditions like skin tone and redness, implement a severity score for each sign, and improve age prediction accuracy using more advanced models and larger datasets.
- **New Features:** Future versions could include real-time video analysis, user profiles for tracking skin changes over time, personalized skincare product recommendations, and a dedicated mobile application.
- **Scalability:** For wider accessibility, the application could be deployed on a cloud platform. Creating a commercial API would also allow for integration with third-party beauty and wellness applications.