INFOSYS SPRINGBOARD VIRTUAL INTERNSHIP 6.0

ARTIFICIAL INTELLIENCE DOMAIN

PROJECT DOCUMENTATION

DERMALSCAN: AI_FACIAL SKIN AGING DETECTION APP

Submitted By

S. Kanishka

Under the Guidance of Mentor

Mr. Praveen Bhargav

INTRODUCTION

Problem Statement

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

Objective of the Project

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

System Requirements

Python Version

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

Milestone 1 of the Project

Module 1: Dataset Setup and Image Labelling

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

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

- Clear Skin
- Dark Spots
- Puffy Eyes
- Wrinkles

Why this module is important in Deep Learning?

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

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

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

To create a bar graph of the class distribution:

```
plt.bar(labels,counts)
plt.title("Class Distribution Plot: Number of Images per Class")
plt.show()
```

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

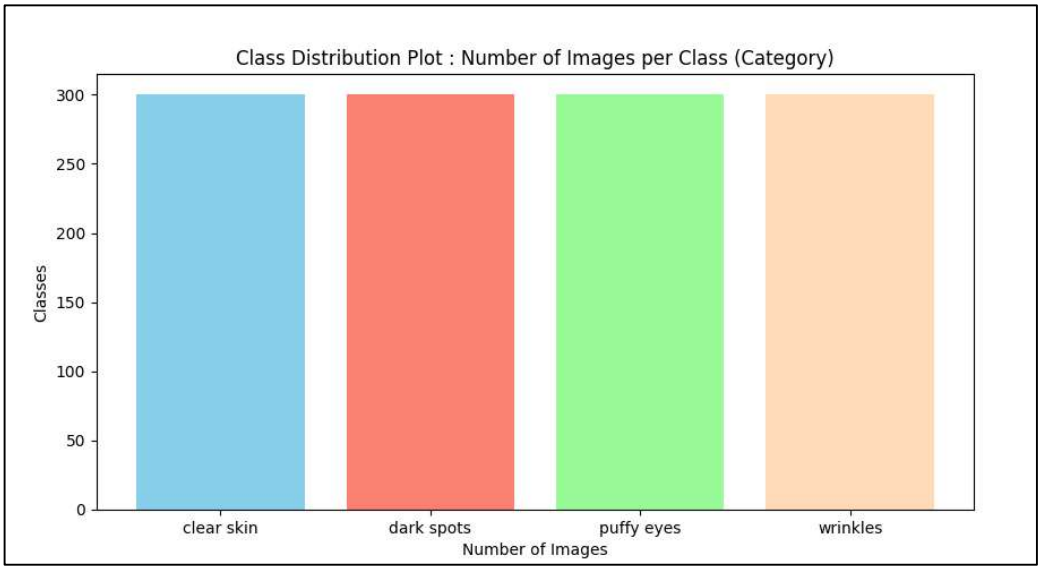

Figure 1: Class Distribution Plot of the Dataset

Module 2: Image Preprocessing and Augmentation

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

img_size = (224, 224)

rescale=1./255

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

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

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

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

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

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

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

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

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

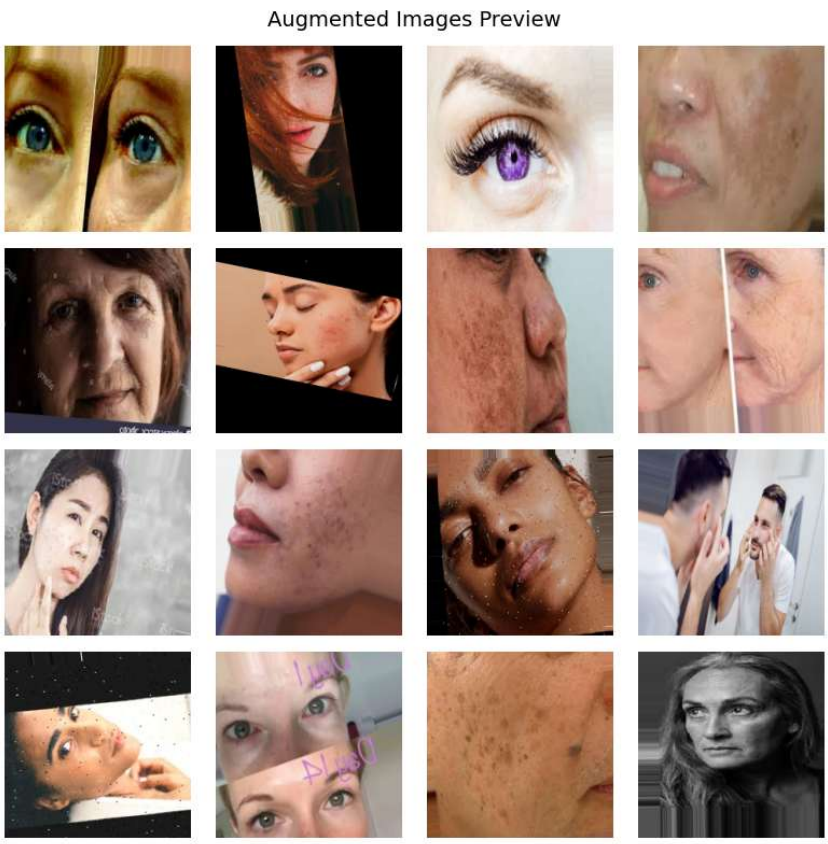

Figure 2: A Preview of the Augmented Images (16 images)

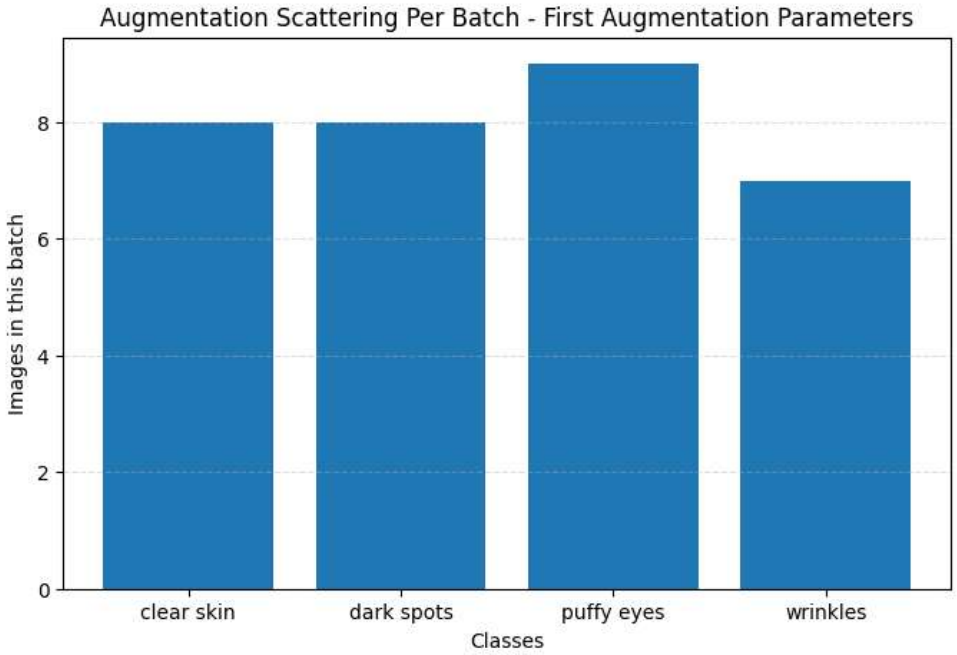

Figure 2: Class-Wise Augmentation Scattering Plot

Key Terms

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

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

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

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

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

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

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

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