DermalScan :AI_Facial Skin Aging Detection App

Infosys Springboard Virtual Internship Program

Submitted by

Parth Agrawal

Under the Guidance of

Mr. Praveen

Mentor, Infosys Springboard

1. Introduction

The goal of Milestone 1 is to complete all dataset-related tasks required for the AI Facial Skin Aging Detection App.
The Milestone 1 includes two modules:

- **Module 1:** Dataset Setup and Image Labelling.
- **Module 2:** Image Preprocessing and Augmentation.

This documentation describes the dataset preparation, cleaning, class distribution visualization, augmentation pipeline, and one-hot encoding implemented in your notebook.

2. Module 1: Dataset Setup and Image Labelling

2.1 Dataset Folder Structure

The dataset was organized in a directory structure where each folder represents a class label:

dataset/

├── wrinkles/

├── dark spots/

├── puffy eyes/

├── clear skin/

2.2 Dataset Scanning & Image Counting

A custom script was written to scan the dataset and count the number of images in each class:

```
def scan_dataset(dataset_dir=DATASET_DIR):
    class_counts = {}
    for cls in CLASSES:
        folder = os.path.join(dataset_dir, cls)
        count = len(os.listdir(folder))
        class_counts[cls] = count
    return class_counts
```

2.3 Duplicate Image Detection & Removal

To produce a clean dataset, a hash-based duplicate detection algorithm was implemented:

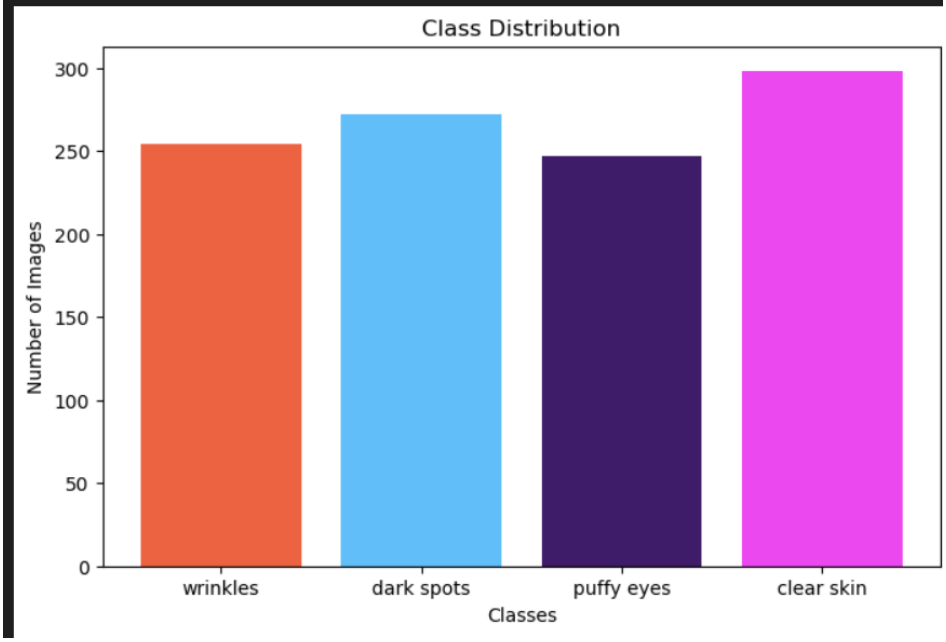
```
with open(  f:
  h = hashlib.md5(f.read()).hexdigest()
```

- If the hash already exists → image is a duplicate
- All duplicates were logged and removed automatically
- Dataset was re-scanned after cleanup

2.4 Class Distribution Visualization

```
def plot_class_distribution(counts):
 plt.figure(figsize=(8,5))
 colors = ["#FF5733", "#33C1FF", "#44196D", "#FF33F8"]
 plt.bar(counts.keys(), counts.values(),color=colors)
 plt.title("Class Distribution")
 plt.xlabel("Classes")
 plt.ylabel("Number of Images")
 plt.show()
```

```
Class Counts: {'wrinkles': 254, 'dark spots': 272, 'puffy eyes': 247, 'clear skin': 298}
```



3. Module 2: Image Preprocessing & Augmentation

As required in the project specification, Module 2 includes:

- resizing to 224×224
- normalization
- augmentation
- one-hot encoding

All four requirements were fully implemented.

3.1 Image Preprocessing with ImageDataGenerator

```
IMG_SIZE = (224, 224)
augmentation = ImageDataGenerator(
   rescale=1/255,
   rotation_range=10,
   zoom_range=0.1,
   horizontal_flip=True,
   width_shift_range=0.05,
   height_shift_range=0.05,
   validation_split=0.2
```

Preprocessing Achieved:

Requirement	Status
Resize to 224x224	Implemented
Normalize images	Using rescale=1/255
Augmentation (flip, zoom, rotation)	Implemented
Train/Validation split	80/20

3.2 Loading Data with One-Hot Encoding

```
train_gen = augmentation.flow_from_directory(
   "dataset",
   target_size=IMG_SIZE,
   batch_size=32,
   class_mode="categorical",
   subset="training"
)
```

This automatically converts each label into a **one-hot encoded vector.**

3.3 Augmentation Visualization

A 4×4 grid of augmented images was displayed:

```
for i in range(16):
   plt.subplot(4,4,i+1)
   plt.imshow(images[i])
   plt.axis("off")
```

This confirms the applied augmentations visually.

Conclusion:

Milestone 1 was successfully completed by preparing a clean and well-structured dataset, implementing preprocessing techniques, applying image augmentation, and enabling one-hot encoding for class labels. The dataset is now fully ready for model development in Milestone 2, where EfficientNetB0 will be trained using the processed inputs.

Milestone-2

In Milestone 2 we integrates **OpenCV** for face detection and **TensorFlow/ Keras** for deep learning–based skin condition classification. This approach allows the system to focus only on the facial area, reducing noise from background regions and improving prediction accuracy.

1.Objectives

The main objectives of the Dermal AI project are:

- To automatically **detect human faces** from static images using computer vision techniques
- To **draw bounding boxes** around detected facial regions for visual clarity
- To **classify facial skin conditions** using a trained deep learning model
- To display the **predicted class along with confidence percentage**, providing interpretability of results
- To **visualize model performance** using training accuracy and loss curves

These objectives collectively ensure that the system is accurate, interpretable, and suitable for further real-world enhancement.

2. Technology Stack

Programming Language

- **Python**, chosen for its extensive support for machine learning and computer vision libraries.

Libraries and Frameworks

- **TensorFlow / Keras**: Used for building, training, and saving the deep learning classification model.
- **OpenCV (cv2)**: Used for face detection, image preprocessing, and drawing bounding boxes.
- **NumPy**: Used for numerical operations and array manipulation.
- **Matplotlib**: Used for visualizing training accuracy and loss curves.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
import os
```

Development Environment

- **(Anaconda)**: Used for model training, experimentation, and visualization.
- **VS Code**: Used for editing, debugging, and running scripts efficiently.

3.Dataset Description

The dataset consists of facial images categorized into the following four skin-condition classes:

- Wrinkles
- Dark Spots
- Puffy Eyes
- Clear Skin

Key characteristics of the dataset:

- Images are resized to **224 × 224 pixels** to match the model's input requirements.
- Pixel values are normalized to improve training stability.
- The dataset is split into **training and validation sets** to evaluate the model's performance on unseen data.
- Each class is stored in a separate folder, enabling automated label generation during training.

```
DATASET_DIR = "dataset"
IMG_SIZE = (224, 224)
BATCH_SIZE = 16
EPOCHS = 50
NUM_CLASSES = 4
```

4. Model Architecture

The skin-condition classification model is based on a **Convolutional Neural Network (CNN)** using transfer learning principles.

- **Input Shape**: (224, 224, 3)

- **Base Model**: Pretrained CNN (such as MobileNetV2 / EfficientNet) trained on ImageNet
- **Custom Layers**:
    - Global Average Pooling
    - Fully Connected Dense Layers
    - Dropout (to reduce overfitting)
- **Output Layer**: Softmax activation with 4 neurons (one per class)

Training Configuration

- **Loss Function**: Categorical Crossentropy
- **Optimizer**: Adam
- **Evaluation Metric**: Accuracy

This architecture allows the model to reuse pretrained visual features while learning task-specific skin-condition patterns.

```
base_model = MobileNetV2(
  input_shape=(224, 224, 3),
  include_top=False,
  weights="imagenet"
)
base_model.trainable = False
```

5. Training Process

The model training process follows a structured and optimized approach:

- Training is performed for multiple epochs to allow the model to converge.
- Validation data is used to monitor performance and detect overfitting.
- **ModelCheckpoint** is used to automatically save the best-performing model based on validation accuracy.
- Learning curves (accuracy and loss) are tracked across epochs to analyze training behavior.

This process ensures efficient learning and stable convergence of the model.

6.Face Detection

Face detection is performed using **OpenCV's Haar Cascade classifier**:

- File used: haarcascade_frontalface_default.xml
- The classifier detects frontal faces in grayscale images.
- Detected faces are represented as bounding boxes defined by (x, y, width, height).

Face detection is a crucial step as it isolates the region of interest (face) before passing it to the classification model.

7. Prediction Pipeline

The prediction pipeline follows these sequential steps:

1. Load the input image from disk
2. Convert the image to grayscale for face detection
3. Detect facial regions using Haar Cascade
4. Crop the detected face region
5. Resize and normalize the cropped face image
6. Pass the processed image to the trained CNN model
7. Predict the skin condition
8. Display bounding box, predicted label, and confidence score

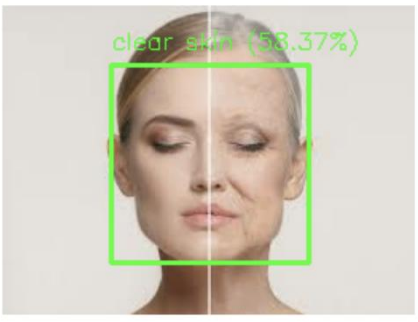
This pipeline ensures end-to-end automation from image input to final visual output.

8.Output Example

The output generated by the system includes:

- A **green bounding box** drawn around the detected face
- A **text label** indicating the predicted skin condition
- A **confidence percentage** showing the model's prediction certainty

This visual output improves interpretability and user understanding of model predictions.



9. Performance Visualization

To evaluate training quality, the following graphs are generated:

- **Accuracy vs Epoch Curve**: Shows how classification accuracy improves over time.
- **Loss vs Epoch Curve**: Indicates how prediction error decreases during training.

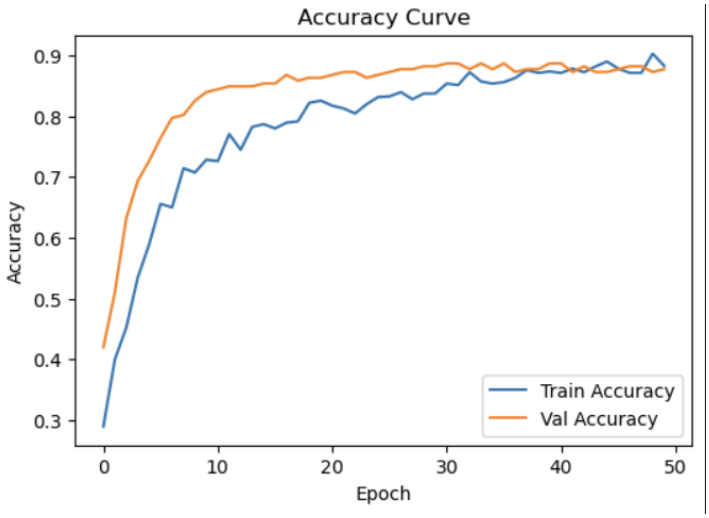
These plots help in identifying issues such as overfitting or underfitting and guide further model improvements.

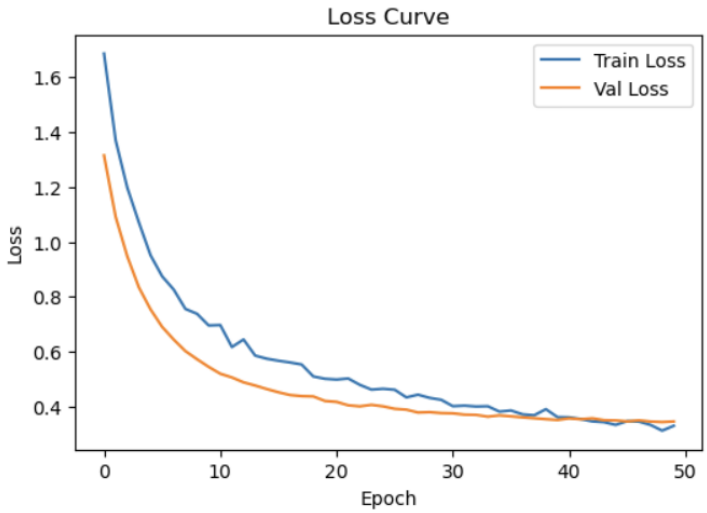
10. Performance Visualization

To evaluate training quality, the following graphs are generated:

- **Accuracy vs Epoch Curve**: Shows how classification accuracy improves over time.
- **Loss vs Epoch Curve**: Indicates how prediction error decreases during training.

These plots help in identifying issues such as overfitting or underfitting and guide further model improvements.





11. Challenges Faced

During development, several challenges were encountered:

- File path mismatches between **C drive and D drive**
- Missing or incorrectly named image files
- Oversized bounding boxes due to Haar Cascade default parameters
- Lower confidence scores caused by limited dataset size

These challenges were resolved through path validation, preprocessing improvements, and parameter tuning. I also run drive check with the help of Command Prompt and remove the corrupt file that causes the problem in running the script.