Project Title:DermalScan: AI_Facial Skin Ageing Detection App

Infosys SpringBoard Virtual Internship Program

Submitted By:

Gauri Narlawar to Mr Praveen sir

Project Statement:

The objective is to develop a deep learning-based system that can detect and classify facial aging signs—such as wrinkles, dark spots, puffy eyes, and clear skin—using a pretrained EfficientNetB0 model. The pipeline includes face detection using Haar Cascades, custom preprocessing and data augmentation, and classification with percentage predictions. A web-based frontend will enable users to upload images and visualise ageing signs with annotated bounding boxes and labels.

Outcomes:

- Detect and localise facial features indicating ageing.
- Classify detected features into categories like wrinkles, dark spots, puffy eyes, and clear skin using a trained CNN model.
- Train and evaluate an EfficientNetB0 model for robust classification.
- Build a web-based frontend for uploading facial images and viewing annotated outputs.
- Interate a backend pipeline that processes input images and returns annotated results.
- Export annotated outputs and logs for documentation or analysis.

1. Introduction

Artificial intelligence and deep learning have advanced significantly in the field of computer vision. One emerging application is the automatic detection of facial aging characteristics. Aging indicators such as wrinkles, dark spots, and puffy eyes can provide insights into skin health, lifestyle habits, and dermatological conditions.
This project aims to build a deep learning–based system that automatically identifies facial aging signs from uploaded images. The final system will detect a face, classify visible signs, and present the result through a user-friendly web interface.
Milestone 1 focuses on preparing and preprocessing the dataset, which is a critical step in building a robust and accurate deep learning model.

Objective of Milestone 1

The key objectives of Milestone 1 are:
To collect, organise, and validate the dataset used for training the model.
To ensure images are properly labelled under predefined categories.
To preprocess the images for model readiness, including resizing, normalisation, and augmentation.
To encode labels into a machine-understandable format.
This ensures a strong foundation before moving to model training.

- ***Libraries Used***

NumPy:

Purpose: Numerical computing and array manipulation
Why used: Stores images as arrays (pixel matrices), performs normalisation, reshaping, and efficient mathematical operations.
Example: converting a list of images into a Numpy array for model training.

- **Panda:**

Purpose**:** Data tables and label handling
Why used**:** Helps organise labels and dataset metadata if needed for analysis.

- **OpenCV (cv2):**

Purpose: Computer vision and image processing
Role in project:
Loading images from disk
Resizing images to 224×224
Converting colour channels
Later stages: Face detection using Haar Cascade

- **Matplotlib & Seaborn**

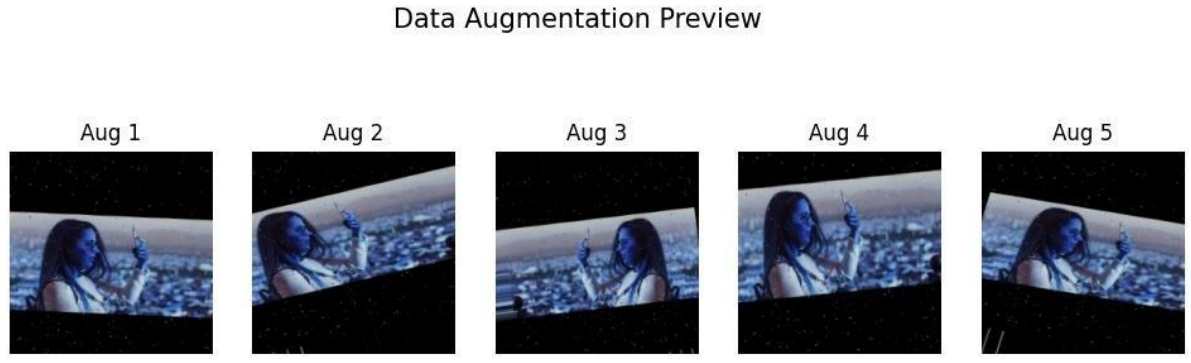
Purpose: Data visualisation and plotting
Used for:
Class distribution bar chart
Image visualization
Understanding dataset balance

- **Scikit-Learn (sklearn)**
  Purpose: Machine learning utilities
  Used for:
  Train-test split
  Label encoding
  Evaluation metrics

- **TensorFlow / Keras**
  Purpose: Deep learning framework
  Role in project:
  Image augmentation
  One-hot encoding of labels
  Later: EfficientNetB0 model training

Code:

```
import numpy as np
import random
import matplotlib.pyplot as plt
random_index = random.randrange(len(processed_images))
selected_image = processed_images[random_index]
image_array = np.expand_dims(selected_image, axis=0)
augmented_batch = [next(datagen.flow(image_array))[0] for _ in range(5)]
plt.figure(figsize=(12, 4))
for i, img in enumerate(augmented_batch):
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"Aug {i+1}")
plt.suptitle("Data Augmentation Preview", fontsize=15)
plt.show()
```



Conclusion of Milestone 1:

Milestone 1 successfully completes the dataset preparation and preprocessing stage. The dataset is now organised, cleaned, normalised, encoded, and augmented. This ensures that the upcoming model training phase (Milestone 2) can proceed efficiently with high-quality input data.

Modules2:

Objective

The objective of Module 2 is to prepare raw facial images into a standardized and robust format suitable for deep learning. Proper preprocessing and augmentation improve model generalization, reduce overfitting, and ensure consistency across the dataset.

Module 2 focuses on preparing the facial image dataset in a form suitable for deep learning model training. Since raw images vary in size, lighting, and orientation, preprocessing is a crucial step to ensure uniformity and improve learning efficiency. In this module, all facial images are resized to a fixed resolution of 224×224 pixels, which is the standard input size required by the EfficientNetB0 architecture. Pixel values are normalized to the range 0–1 to stabilize training and speed up convergence.

To improve model generalization and prevent overfitting, image augmentation techniques are applied. Augmentation artificially increases dataset diversity by generating new variations of existing images. Techniques such as horizontal flipping, rotation, zooming, and shifting help the model learn invariant facial features under different conditions. This is especially important in facial analysis tasks where lighting, pose, and expression vary significantly.

Additionally, class labels are converted into one-hot encoded vectors, allowing the model to perform multi-class classification effectively. Module 2 ensures that the dataset is balanced, standardized, and diverse enough to support robust learning in later stages.

1. Image Resizing

All images are resized to 224 × 224 pixels.

This size is chosen because EfficientNetB0 expects inputs of this resolution.

Uniform image size ensures compatibility with CNN layers.

2. Image Normalization

Pixel values are scaled from [0, 255] → [0, 1].

Normalization helps:

Faster convergence

Numerical stability

Improved gradient flow

Data Augmentation

To increase dataset diversity, real-time augmentation is applied:

Rotation

Horizontal flipping

Zooming

Width and height shifting

This simulates real-world variations such as lighting, pose, and facial orientation.

4. Label Encoding

Skin condition labels are converted into one-hot encoded vectors.

Classes:

Clear Skin

Dark Spots

Puffy Eyes

Wrinkles

One-hot encoding is required for categorical cross-entropy loss.

Deliverables

Preprocessed dataset

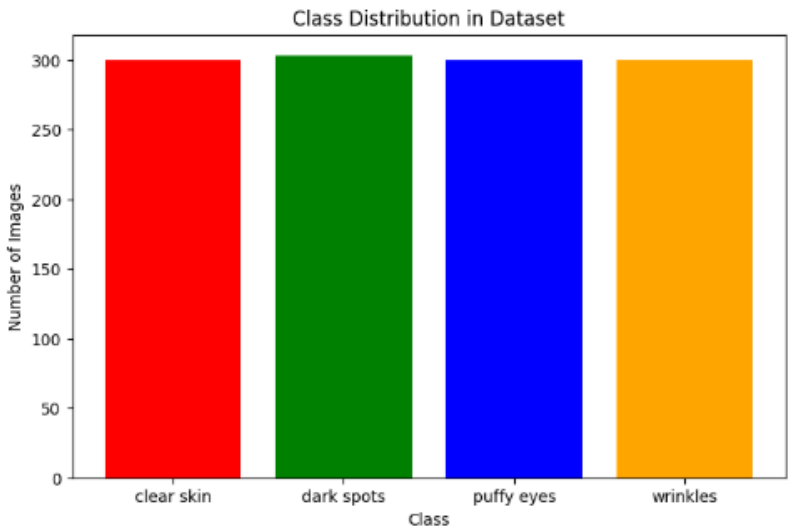
Augmented image visualization

One-hot encoded labels

Code

```
            import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5))
colors = ['red', 'green', 'blue', 'orange']
plt.bar(class_counts.keys(), class_counts.values(), color=colors)
plt.title("Class Distribution in Dataset")
plt.xlabel("Class")
plt.ylabel("Number of Images")
plt.show()
```

output:



Modules 3

Introduction

Module 3 focuses on training a deep learning model to classify facial skin aging signs using a pretrained convolutional neural network. Transfer learning is employed to utilize previously learned visual features, enabling faster training and improved accuracy even with a limited dataset.

EfficientNetB0 Architecture

EfficientNetB0 is a state-of-the-art convolutional neural network that uses compound scaling to balance network depth, width, and resolution. It is pretrained on the ImageNet dataset and can extract rich visual features such as edges, textures, and facial patterns.

The Module 3 focuses on building and training a deep learning model for facial skin condition classification using transfer learning. In this module, the EfficientNetB0 architecture pre-trained on the ImageNet dataset is used as the base model because of its high accuracy and computational efficiency. The top classification layers of EfficientNetB0 are removed, and new custom layers such as Global Average Pooling, Dropout, and a Dense Softmax layer are added to adapt the model to the skin aging dataset, which contains four classes: wrinkle, dark spots, puffy eyes, and clear skin.

The model is trained using the categorical cross-entropy loss function and the Adam optimizer, which helps in faster convergence and stable learning. During training, the dataset is split into training and validation sets to monitor the model's generalization ability. Training and validation accuracy, along with loss values, are recorded for each epoch to evaluate performance and detect issues such as overfitting or underfitting. Accuracy and loss curves are plotted to visually analyze learning behavior over epochs. Finally, the trained model is saved in .keras format so it can be reused for real-time prediction and deployment in the face detection pipeline of Module 4.

pretrained layers serve as a feature extractor, while custom classification layers are added on top to adapt the model for facial skin aging classification.

Transfer Learning Approach

Transfer learning allows the model to reuse knowledge from large-scale datasets. In this project:

- The EfficientNetB0 base model is frozen to preserve learned features.
- Custom dense layers are added for classification into four skin aging categories.
- This approach reduces overfitting and training time.

Training Configuration

The model is trained using:

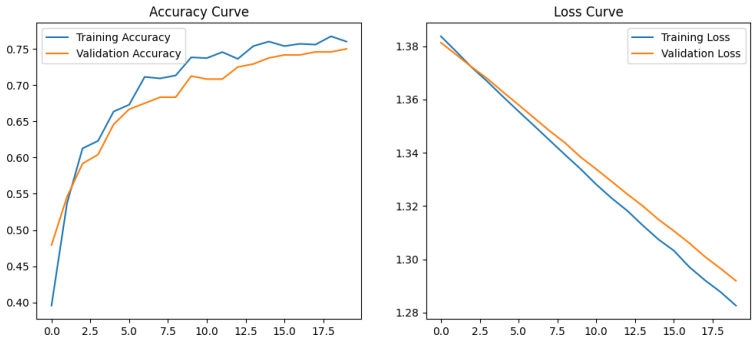
- Loss Function: Categorical Cross-Entropy, suitable for multi-class classification.
- Optimizer: Adam optimizer, which adapts learning rates for faster convergence.
- Activation Function: Softmax in the output layer to generate class probabilities.
- Validation Split: A portion of the dataset is reserved for validation to monitor performance.

Parameter	Value
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Cross-Entropy
Metrics	Accuracy
Epochs	20
Batch Size	32
Validation Split	20%

Code

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy Curve")
plt.subplot(1,2,2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.title("Loss Curve")
plt.show()
```

output



Model Evaluation

Model performance is evaluated using:

- Training and validation accuracy
- Training and validation loss curves
- Stability of validation accuracy over epochs

Early stopping is applied to prevent overfitting by stopping training when validation accuracy stops improving

Outcome of Module 3

The trained EfficientNetB0 model achieves high classification accuracy with stable validation performance. The final trained model is saved in .h5 format for reuse during inference and deployment.

Module 4: Face Detection and Prediction Pipeline

Introduction

Module 4 integrates computer vision techniques with the trained deep learning model to detect faces in images and predict facial skin aging signs. This module bridges the gap between model training and real-world application.

Module 4 integrates computer vision techniques with the trained deep learning model to create an end-to-end facial skin aging detection pipeline. The primary goal of this module is to detect faces in an image and apply the trained model to predict skin aging problems in a real-world scenario.

Face detection is performed using the Haar Cascade Classifier, a traditional machine learning–based method provided by OpenCV. Haar Cascades are chosen because they are fast, lightweight, and effective for detecting frontal human faces. The classifier scans the input image and identifies face regions using predefined Haar-like features.

Once a face is detected, the face region is cropped and preprocessed to match the input requirements of the trained EfficientNetB0 model. The cropped face is resized and normalized before being passed to the model for prediction. The model outputs probability scores for each skin aging category, and the class with the highest probability is selected as the final prediction.

The prediction results are visualized directly on the image. A green bounding box is drawn around the detected face, and the predicted skin condition along with its confidence percentage is displayed clearly on the image. This visualization makes the output easy to understand for users and helps in quick analysis.

Module 4 demonstrates the practical usability of the trained model by combining face detection, classification, and visualization into a single pipeline. It validates that the system can accurately analyze new images and present meaningful results in a user-friendly manner

Face Detection Using Haar Cascade

Haar Cascade Classifier is a machine learning-based approach used for object detection, particularly faces. It uses Haar-like features and a cascade of classifiers to detect frontal faces efficiently.

OpenCV's Haar Cascade implementation is used due to its speed, simplicity, and real-time performance.

Prediction Pipeline

The prediction pipeline consists of the following steps:

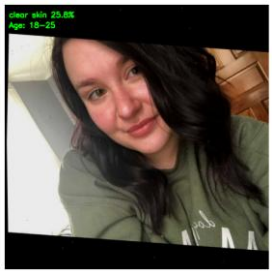
1. Read the input image
2. Convert the image to grayscale for face detection
3. Detect face regions using Haar Cascade
4. Crop the detected face area
5. Preprocess the face (resize and normalize)
6. Predict skin aging category using the trained CNN
7. Display results on the image

Code

```
def predict_aging_clean(image_path):
img = cv2.imread(image_path)
 if img is None
print("Image not found!")
return
# Convert for detection
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
 faces = face_cascade.detectMultiScale
(gray,

 scaleFactor=1.1

 minNeighbors=4,

        minSize=(80, 80)
    )# If no face detected, use whole image as fallback
    if len(faces) == 0:
        face = img.copy()
        processed_face = preprocess_face(cv2.cvtColor(face, cv2.COLOR_BGR2RGB))
        preds = model.predict(processed_face)[0]
         idx = np.argmax(preds)
        skin_label = classes[idx]
        skin_conf = preds[idx] * 10
        age_range = estimate_age(skin_label)
         text_skin = f"{skin_label} {skin_conf:.1f}%"
         text_age = f"Age: {age_range}"



        # Smaller font
        font = cv2.FONT_HERSHEY_SIMPLEX
        font_scale = 0.6
        thickness = 2
```

```
            # Put text inside top-left
            cv2.putText(img, text_skin, (10, 30), font, font_scale, (0, 255, 0), thickness)
            cv2.putText(img, text_age, (10, 55), font, font_scale, (0, 255, 0), thickness)

            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            plt.figure(figsize=(6,6))
            plt.imshow(img_rgb)
            plt.axis("off")
            plt.show()
            return

    # For each detected face
    for (x, y, w, h) in faces:
        face_region = img[y:y+h, x:x+w]
        processed_face = preprocess_face(cv2.cvtColor(face_region, cv2.COLOR_BGR2RGB))
        preds = model.predict(processed_face)[0]

        idx = np.argmax(preds)
        skin_label = classes[idx]
        skin_conf = preds[idx] * 100
        age_range = estimate_age(skin_label)

        text_skin = f"{skin_label} {skin_conf:.1f}%"
        text_age = f"Age: {age_range}"

        # Draw green bounding box
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 3)


        # Smaller text inside the box region near top-left
        font = cv2.FONT_HERSHEY_SIMPLEX
        font_scale = 0.6
        thickness = 2


        cv2.putText(img, text_skin, (x+5, y+20), font, font_scale, (0, 255, 0), thickness)
        cv2.putText(img, text_age, (x+5, y+40), font, font_scale, (0, 255, 0), thickness)


    # Show final image
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(6,6))
    plt.imshow(img_rgb)
    plt.axis("off")
    plt.show()

dataset_root = r"DATASET-20251202T142134Z-1-001/DATASET"


cls = random.choice(os.listdir(dataset_root))
img_name = random.choice(os.listdir(os.path.join(dataset_root, cls)))
test_image = os.path.join(dataset_root, cls, img_name)


predict_aging_clean(test_image)
```

OUTPUT



Interpretation of Prediction Confidence

The confidence percentage represents the relative likelihood of the predicted class compared to other classes. It does not indicate the severity of aging but shows how confidently the model identifies a specific skin condition.

Module	Face Detection & Prediction Pipeline
Face Detector	Haar Cascade (OpenCV)
Classification Model	Trained EfficientNetB0
Input Image Size	Any size (auto resized to 224×224)
Detection Method	Bounding Box on Face
Output Labels	Wrinkles, Dark Spots, Puffy Eyes ,Clear Skin
Output Display	Green Box + Class + Confidence (%)
Frameworks Used	OpenCV, TensorFlow, Keras
Inference Type	Single Image Prediction
Visualization	Matplotlib

Outcome of Module 4

Module 4 successfully demonstrates an end-to-end facial skin aging detection system. It accurately detects faces, predicts aging-related skin conditions, and presents results in a visually interpretable manner