Project Title:DermalScan: AI_Facial Skin Ageing Detection App

Infosys SpringBoard Virtual Internship Program

Submitted By:

Brijesh Rath to Mr. Praveen sir

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.

Objectives:

- Detecting and locating facial characteristics which indicate the presence of age.
- Categorise the characteristics into 4 specific categories (i.e. Wrinkled, Dark Spot, Puffy Eyes and Clear Skin) using a Convolutional Neural Network (CNN).
- Create a user-friendly web based front end by allowing users to upload images with the expected annotated outcomes and percentage predictions.
- Connect the pipeline with a backend to process uploaded images, make inferences and return annotated results.

System Requirements:

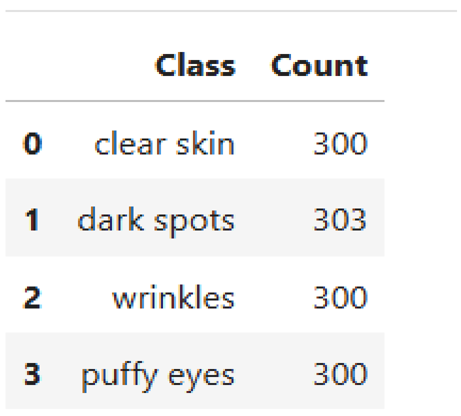
- **Python (v3.10.1):** The primary language used for all data processing and model building.
- **Pandas(2.3.3):** Used for creating DataFrames, managing the dataset, and handling CSV files (import pandas as pd).
- **Matplotlib(3.10.7):** It is the foundational plotting and visualization library for Python.
- **Seaborn(0.13.2):** It simplifies complex statistical plots (like heatmaps or count plots) into single lines of code and makes them look better automatically.
- **Scikit-learn(1.7.2):** Used for splitting the data into training and testing sets (train_test_split).
- **Numpy(2.2.6):** It is a fundamental dependency for Pandas, Scikit-Learn, and TensorFlow, and is required for numerical operations.
- **Jupyter Notebook:** Used for interactive coding and ensuring the code runs step by step

Milestone 1 of the Project

➢ Module 1: Dataset Setup and Image Labeling

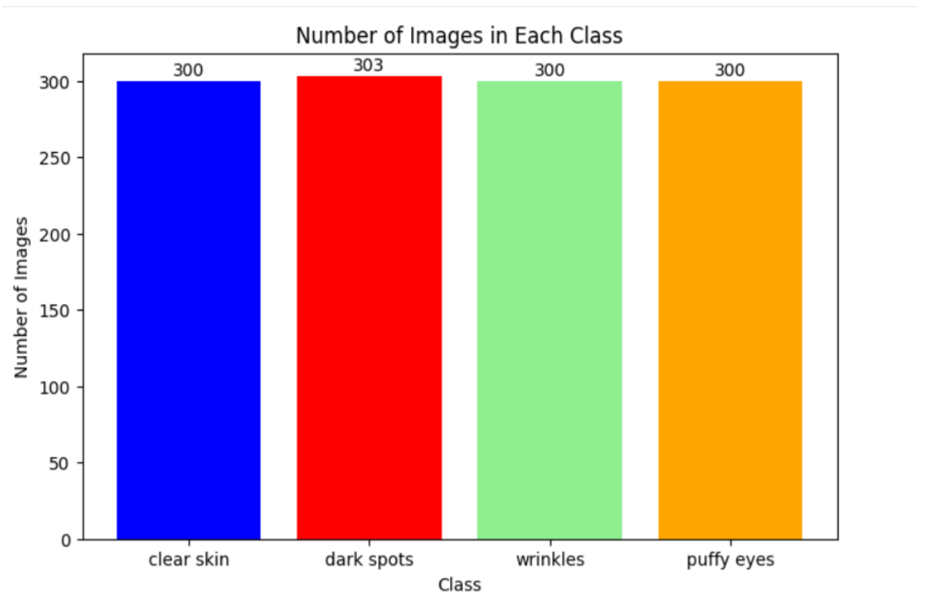
- ➢ Aggregated facial images into four distinct classes: Wrinkles, Dark Spots, Puffy Eyes, and Clear Skin.
- ➢ Manually filtered out poor-quality samples and ensured equal representation across all classes to prevent model bias.
- ➢ Produced a finalized, labeled dataset directory and a class distribution plot confirming data balance.

➢ Output:



	Class	Count
0	clear skin	300
1	dark spots	303
2	wrinkles	300
3	puffy eyes	300

[Fig 1. Dataset Class Distribution Summary]



[Fig 2. Distribution of Images Across Skin Condition Categories]

➢ 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.

```
target_size= (224, 224)
img = img.astype("float32") / 255.0  #normalize
```

- 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. 75% of data is for training, 15% of data is for testing and 15% 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 multiclass classification, enabling the model to process labels mathematically during training.

- **Creation of a generator object using the method RandomFlip,RandomZoom and etc. from the tensorflow. keras.Sequential module,**

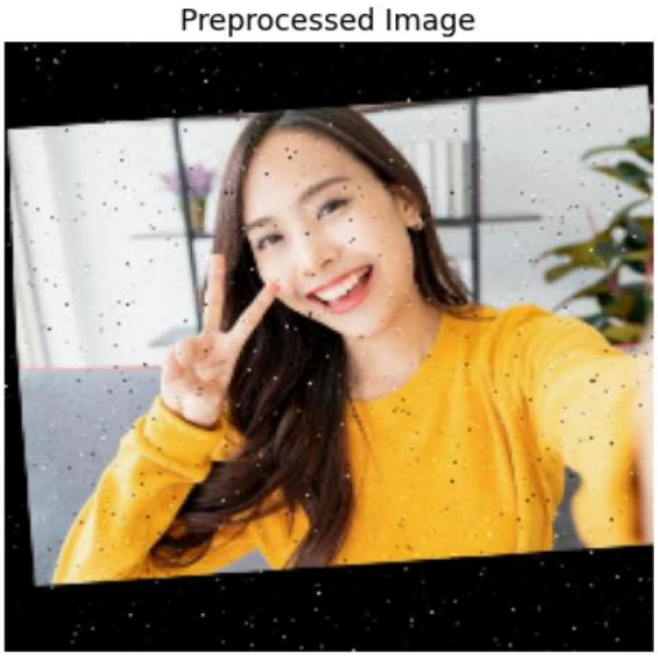
```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.1),
    tf.keras.layers.RandomZoom(0.1),
    tf.keras.layers.RandomContrast(0.2)
])
```

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

```
def load_dataset(path, img_size=224, batch_size=32):
    dataset =
tf.keras.preprocessing.image_dataset_from_directory(
        path,
        image_size=(img_size, img_size),
        label_mode="categorical",  # one-hot encoding
        shuffle=True
    )
```

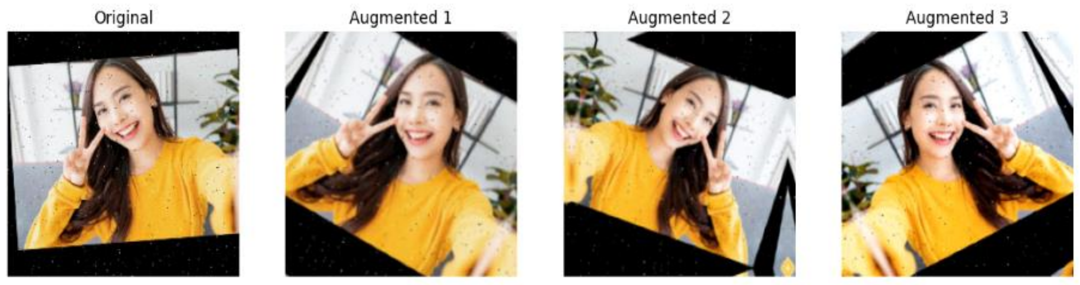
- This module results in a fully pre-processed, augmented, and model-ready dataset. It ensures that images maintain consistent dimensions, classes are represented with increased diversity, and the dataset becomes more resilient to variations, ultimately enhancing the performance and generalization of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

➢ Preprocessed Image Preview

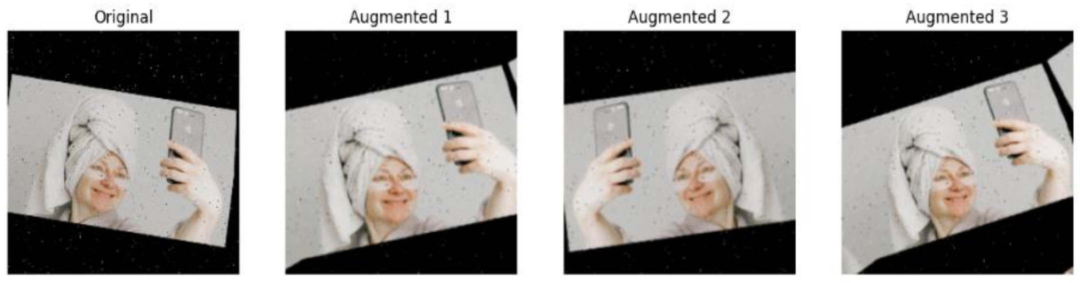


[Fig 3. Preprocessed Image]

➢ Augmented Image Preview

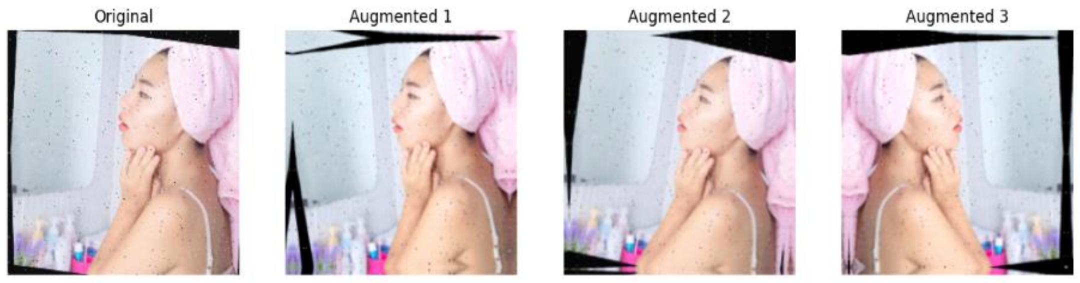


```
Processing: /Users/brijesh/Desktop/AI_DermalScan/data/raw/clear skin/clear_skin_003.jpg
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.0411072].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.012044].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..1.0363963].
```

```
Processing: /Users/brijesh/Desktop/AI_DermalScan/data/raw/clear skin/clear_skin_004.jpg
```



[Fig 4. Augmented Image]

- Additionally, a preview grid of 3 augmented images with the original image.

Milestone 2 of the Project

Overview of Milestone 2:

- Milestone 2 focuses on developing, training, evaluating, and deploying a deep learning–based image classification model using transfer learning. This milestone is divided into two main modules:

- Module 3: Model Training and Evaluation
- Module 4: Face Detection and Prediction Pipeline

- The objective is to train a robust CNN model using a pretrained architecture, validate its performance using appropriate metrics, and finally deploy the trained model in a real-time face detection and prediction system.

Module 3: Model Training and Evaluation

Objective:

- The goal of Module 3 is to build and train a deep learning image classification model using transfer learning, evaluate its performance on training, validation, and test datasets, and analyze model behavior using accuracy and loss curves.

Dataset Description:

- The dataset is organized into three separate directories:

- Training dataset
- Validation dataset
- Testing dataset

- Each directory contains subfolders representing different classes. The images are resized to a fixed resolution of 224 × 224 pixels, matching the input requirements of the pretrained network.

Data Preprocessing and Augmentation:

- To improve generalization and prevent overfitting, image augmentation techniques were applied to the training dataset:

- Rescaling pixel values to the range [0, 1]
- Random rotations
- Width and height shifts
- Zoom augmentation
- Horizontal flipping
- Brightness adjustment

- Validation and test datasets were rescaled only, without augmentation, to ensure fair evaluation.

Model Architecture:

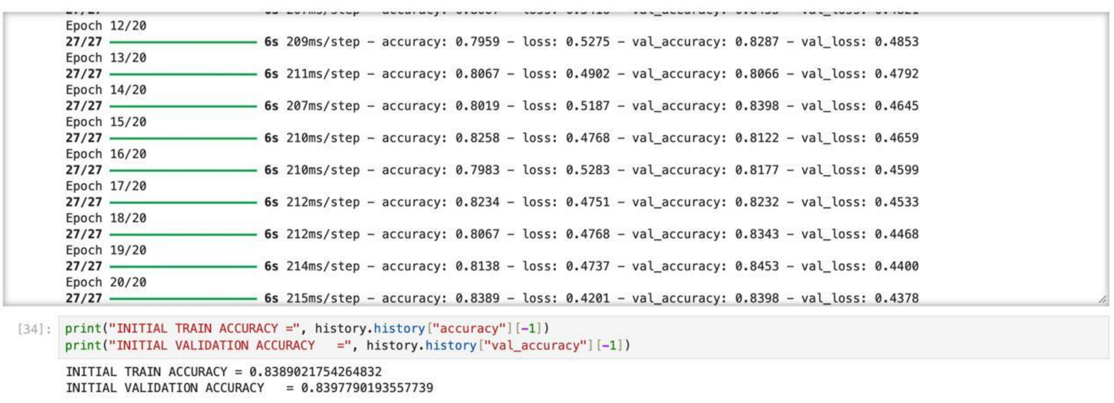
- A MobileNetV2 pretrained model was used as the backbone for transfer learning.
- Architecture Details
    - Base Model: MobileNetV2 (ImageNet weights)
    - Top layers removed: Fully connected layers excluded
    - Custom classification head added:
        - Global Average Pooling layer
        - Fully connected Dense layer with ReLU activation
        - Dropout layers for regularization
        - Final Dense layer with Softmax activation for multi-class classification

Initial Training Phase:

- During the first training phase:
    - All layers of the pretrained base model were **frozen**
    - Only the custom classification head was trained
    - Optimizer used: **Adam**
    - Loss function: **Categorical Crossentropy**
    - Metric monitored: **Accuracy**

- Training Configuration:
    - Epochs: 20
    - Batch size: 32

```
history = model.fit(    #first TRAINING
    train_data,
    validation_data=val_data,
    epochs=20,
    verbose=1
)
```



```
Epoch 12/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 209ms/step - accuracy: 0.7959 - loss: 0.5275 - val_accuracy: 0.8287 - val_loss: 0.4853
Epoch 13/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 211ms/step - accuracy: 0.8067 - loss: 0.4902 - val_accuracy: 0.8066 - val_loss: 0.4792
Epoch 14/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 207ms/step - accuracy: 0.8019 - loss: 0.5187 - val_accuracy: 0.8398 - val_loss: 0.4645
Epoch 15/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 210ms/step - accuracy: 0.8258 - loss: 0.4768 - val_accuracy: 0.8122 - val_loss: 0.4659
Epoch 16/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 210ms/step - accuracy: 0.7983 - loss: 0.5283 - val_accuracy: 0.8177 - val_loss: 0.4599
Epoch 17/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 212ms/step - accuracy: 0.8234 - loss: 0.4751 - val_accuracy: 0.8232 - val_loss: 0.4533
Epoch 18/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 212ms/step - accuracy: 0.8067 - loss: 0.4768 - val_accuracy: 0.8343 - val_loss: 0.4468
Epoch 19/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 214ms/step - accuracy: 0.8138 - loss: 0.4737 - val_accuracy: 0.8453 - val_loss: 0.4400
Epoch 20/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 215ms/step - accuracy: 0.8389 - loss: 0.4201 - val_accuracy: 0.8398 - val_loss: 0.4378
```

```
[34]: print("INITIAL TRAIN ACCURACY =", history.history["accuracy"][-1])
      print("INITIAL VALIDATION ACCURACY   =", history.history["val_accuracy"][-1])

      INITIAL TRAIN ACCURACY = 0.8389021754264832
      INITIAL VALIDATION ACCURACY   = 0.8397790193557739
```

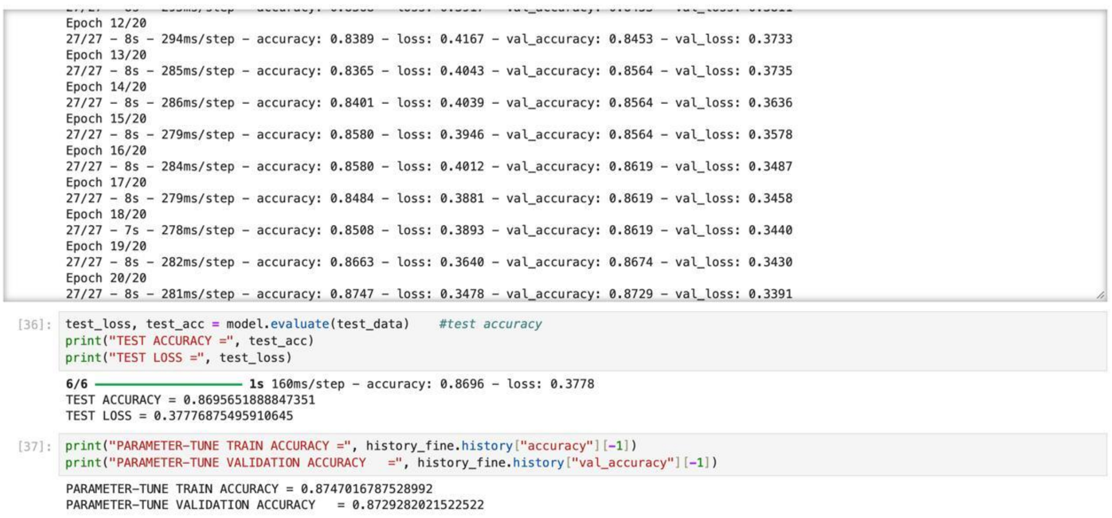
[Fig 5. Training & Validation Accuracy – Initial Training]

Parameter-Tuning Phase:

- ➢ To further improve performance, fine-tuning was performed:
  - The last 30 layers of the base model were unfrozen
  - A lower learning rate was used to prevent large weight updates
  - This allowed the model to learn task-specific features

- ➢ Parameter-Tuning Configuration
  - Optimizer: Adam
  - Learning rate: 1e-5
  - Epochs: 20

```
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

history_fine = model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    verbose=2
)
```



```
Epoch 12/20
27/27 - 8s - 294ms/step - accuracy: 0.8389 - loss: 0.4167 - val_accuracy: 0.8453 - val_loss: 0.3733
Epoch 13/20
27/27 - 8s - 285ms/step - accuracy: 0.8365 - loss: 0.4043 - val_accuracy: 0.8564 - val_loss: 0.3735
Epoch 14/20
27/27 - 8s - 286ms/step - accuracy: 0.8401 - loss: 0.4039 - val_accuracy: 0.8564 - val_loss: 0.3636
Epoch 15/20
27/27 - 8s - 279ms/step - accuracy: 0.8580 - loss: 0.3946 - val_accuracy: 0.8564 - val_loss: 0.3578
Epoch 16/20
27/27 - 8s - 284ms/step - accuracy: 0.8580 - loss: 0.4012 - val_accuracy: 0.8619 - val_loss: 0.3487
Epoch 17/20
27/27 - 8s - 279ms/step - accuracy: 0.8484 - loss: 0.3881 - val_accuracy: 0.8619 - val_loss: 0.3458
Epoch 18/20
27/27 - 7s - 278ms/step - accuracy: 0.8508 - loss: 0.3893 - val_accuracy: 0.8619 - val_loss: 0.3440
Epoch 19/20
27/27 - 8s - 282ms/step - accuracy: 0.8663 - loss: 0.3640 - val_accuracy: 0.8674 - val_loss: 0.3430
Epoch 20/20
27/27 - 8s - 281ms/step - accuracy: 0.8747 - loss: 0.3478 - val_accuracy: 0.8729 - val_loss: 0.3391
```

```
[36]: test_loss, test_acc = model.evaluate(test_data)    #test accuracy
      print("TEST ACCURACY =", test_acc)
      print("TEST LOSS =", test_loss)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 160ms/step - accuracy: 0.8696 - loss: 0.3778
TEST ACCURACY = 0.8695651888847351
TEST LOSS = 0.37776875495910645
```

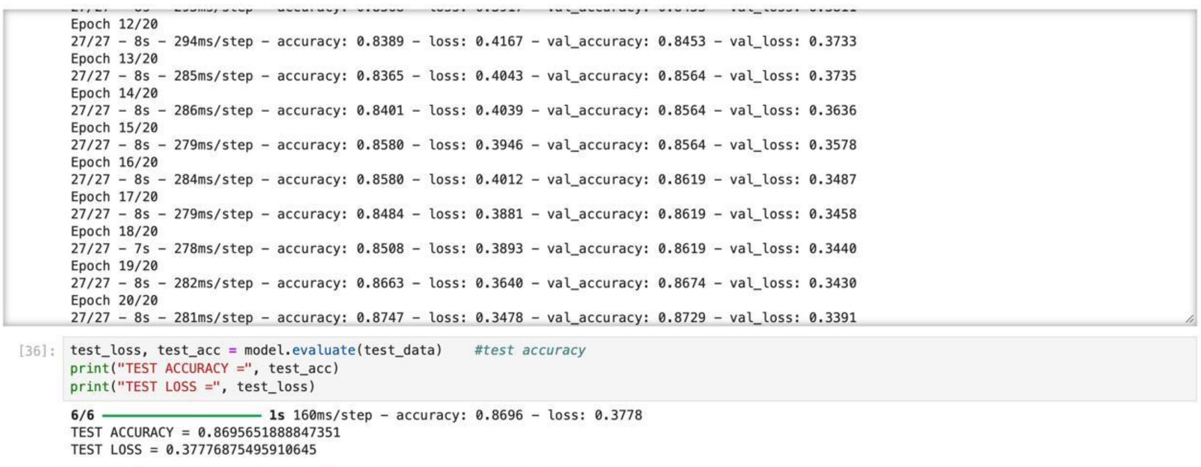
```
[37]: print("PARAMETER-TUNE TRAIN ACCURACY =", history_fine.history["accuracy"][-1])
      print("PARAMETER-TUNE VALIDATION ACCURACY   =", history_fine.history["val_accuracy"][-1])
```

```
PARAMETER-TUNE TRAIN ACCURACY = 0.8747016787528992
PARAMETER-TUNE VALIDATION ACCURACY    = 0.8729282021522522
```

[Fig 6. Training & Validation Accuracy – Parameter-Tuning Training]

Model Evaluations:

- After training, the model was evaluated on the test dataset, which was not used during training or validation.
- Evaluation Metrics
  - Training Accuracy
  - Validation Accuracy
  - Testing Accuracy
  - Loss values
- Observed Performance
  - Training accuracy showed consistent improvement
  - Validation accuracy remained stable
  - Testing accuracy confirmed good generalization

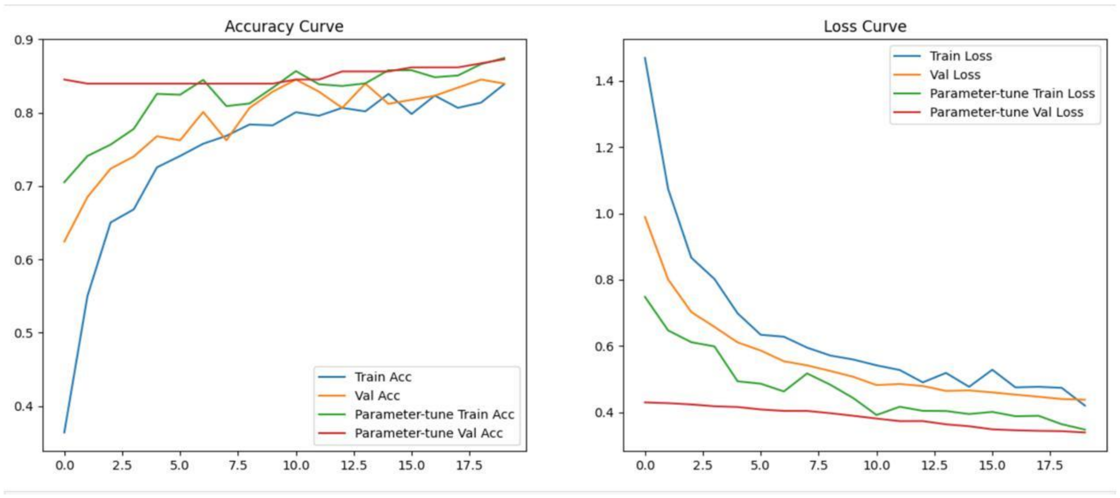


```
Epoch 12/20
27/27 - 8s - 294ms/step - accuracy: 0.8389 - loss: 0.4167 - val_accuracy: 0.8453 - val_loss: 0.3733
Epoch 13/20
27/27 - 8s - 285ms/step - accuracy: 0.8365 - loss: 0.4043 - val_accuracy: 0.8564 - val_loss: 0.3735
Epoch 14/20
27/27 - 8s - 286ms/step - accuracy: 0.8401 - loss: 0.4039 - val_accuracy: 0.8564 - val_loss: 0.3636
Epoch 15/20
27/27 - 8s - 279ms/step - accuracy: 0.8580 - loss: 0.3946 - val_accuracy: 0.8564 - val_loss: 0.3578
Epoch 16/20
27/27 - 8s - 284ms/step - accuracy: 0.8580 - loss: 0.4012 - val_accuracy: 0.8619 - val_loss: 0.3487
Epoch 17/20
27/27 - 8s - 279ms/step - accuracy: 0.8484 - loss: 0.3881 - val_accuracy: 0.8619 - val_loss: 0.3458
Epoch 18/20
27/27 - 7s - 278ms/step - accuracy: 0.8508 - loss: 0.3893 - val_accuracy: 0.8619 - val_loss: 0.3440
Epoch 19/20
27/27 - 8s - 282ms/step - accuracy: 0.8663 - loss: 0.3640 - val_accuracy: 0.8674 - val_loss: 0.3430
Epoch 20/20
27/27 - 8s - 281ms/step - accuracy: 0.8747 - loss: 0.3478 - val_accuracy: 0.8729 - val_loss: 0.3391
```

```
[36]: test_loss, test_acc = model.evaluate(test_data)    #test accuracy
      print("TEST ACCURACY =", test_acc)
      print("TEST LOSS =", test_loss)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 160ms/step - accuracy: 0.8696 - loss: 0.3778
TEST ACCURACY = 0.8695651888847351
TEST LOSS = 0.37776875495910645
```

[Fig 7. Console output showing Test Accuracy and Test Loss]



[Fig 8. Line-Graph showing Accuracy and Loss Curve]

Model Saving:

- The trained model was saved in (.h5) format for later use in deployment.
- **Saved Files:**
  - `skin_classifier_mobilenetv2.h5`
  - `class_indices.json`

- Module 3 successfully achieved its objective of training a reliable image classification model using transfer learning. The model demonstrated stable training behaviour, good validation performance, and acceptable test accuracy, making it suitable for deployment in a real-time prediction pipeline.

Module 4: Face Detection and Prediction Pipeline

Objective:

- The goal of Module 4 is to integrate the trained model into a real-time prediction pipeline that:
    - Detects faces using OpenCV
    - Crops detected face regions
    - Applies the trained model to predict the class
    - Displays predictions as percentages in real time

Face Detection Method:

- Face detection was implemented using:
    - **OpenCV**
    - **Haar Cascade Classifier**
- The Haar Cascade model detects faces in grayscale images using pre-trained features.

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
```

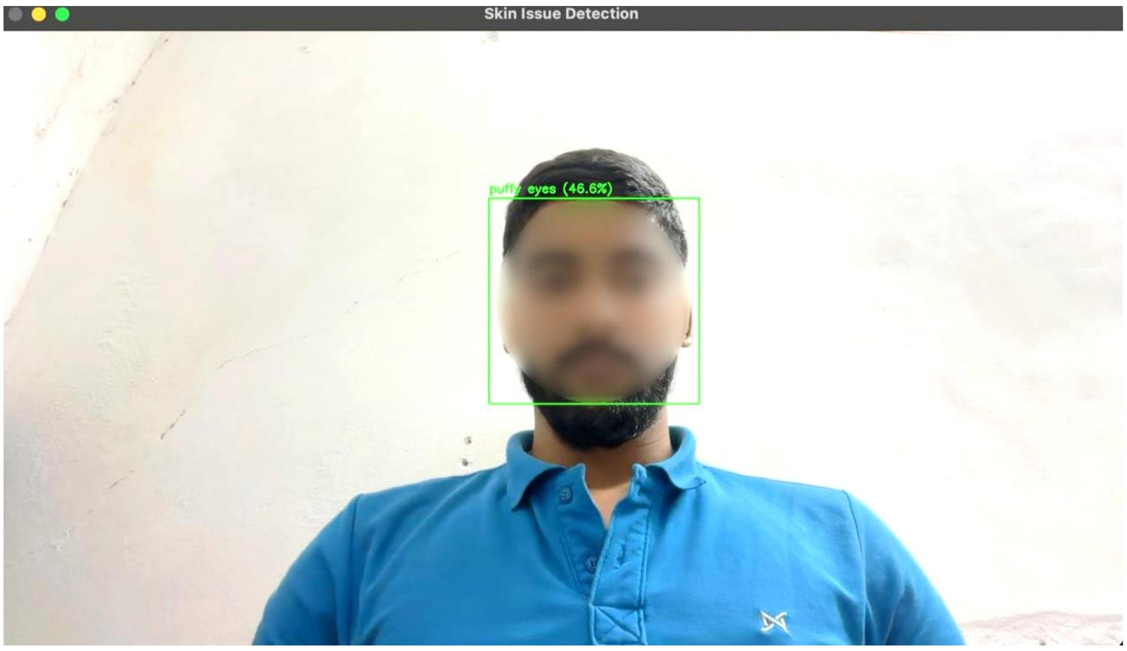
Prediction Pipeline Workflow:

- The complete prediction pipeline follows these steps:
    1. input from webcam or image
    2. Convert frame to grayscale
    3. Capture Detect faces using Haar Cascade
    4. Crop detected face region
    5. Resize cropped face to model input size (224×224)
    6. Normalize pixel values
    7. Pass the image to the trained CNN model
    8. Obtain class probabilities
    9. Display predicted class and confidence percentage

Real-Time Webcam Prediction:

- The system was tested using a webcam feed:
    - Bounding boxes drawn around detected faces
    - Predicted class label displayed
    - Confidence score shown as a percentage
    - Real-time inference achieved with stable frame rates

```
cap = cv2.VideoCapture(0)  #(0) for webcam

while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        face_roi = frame[y:y+h, x:x+w]

        class_name, confidence, probs = predict_skin_issue(face_roi)

        # Bounding box
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        # Display prediction text
        text = f"{class_name} ({confidence:.1f}%)"
        cv2.putText(frame, text, (x, y-10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)

    cv2.imshow("Skin Issue Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```



[Fig 9. Real-time webcam output showing detected face with bounding box and prediction confidence percentage]

Evaluation Criteria:

- **Face Detection Accuracy**
  - Cascade successfully detected frontal faces
  - Bounding boxes correctly enclosed face regions
- Haar **Prediction Accuracy**
  - Model predictions matched expected class labels
  - Confidence values reflected prediction certainty

- Module 4 successfully completed the deployment of the trained model into a real-time face detection and prediction pipeline. The system accurately detects faces and displays class predictions with confidence percentages, fulfilling all module requirements.