DermalScan : AI_Facial Skin Aging Detection App

Infosys Springboard Virtual Internship Program

Submitted by,

Rounak Kumar Mishra

Under the guidance of Mentor **Mr. Praveen**

Problem Statement

Skin-related issues such as wrinkles, dark spots, puffy eyes, and uneven skin texture are common in dermatology. Identifying these conditions manually is time-consuming and requires expert knowledge.

The **AI DermaScan** project aims to develop an automated AI system that can classify facial skin images into distinct categories such as Clear Skin, Dark Spots, Puffy Eyes, Wrinkles.To achieve accurate classification, the dataset must undergo **cleaning, preprocessing, augmentation, and normalization**.

Objectives

The objective of Milestone 1 and Milestone 2 is to develop an end-to-end deep learning-based system for facial skin condition detection. Milestone 1 focuses on organizing and preprocessing the facial skin image dataset by performing proper labeling, normalization, and data augmentation to prepare high-quality input data for model training. Milestone 2 builds upon this foundation by training and evaluating a convolutional neural network using transfer learning techniques and applying the trained model to real-world facial images. This includes detecting faces using computer vision techniques, cropping facial regions, and displaying skin condition predictions along with confidence percentages and age group information.

Technologies & Libraries Used

- **Programming Language:** Python 3.11.9
- **Libraries:**

Library	Purpose
OpenCV (cv2)	**Image processing and face detection**
NumPy	**Numerical operations and normalization**
Pillow (PIL)	**Image loading and verification**
Matplotlib	**Image and graph visualization**
Pandas	**Dataset analysis and reporting**
TensorFlow	**Deep learning model training**
Keras	**CNN model building and transfer learning**
EfficientNetB0 / MobileNetV2	**Pretrained models for classification**
Haar Cascade Classifier	**Face detection**
Jupyter Notebook	**Development and experimentation**

Development Tools

- Jupyter Notebook – primary development environment
- Windows OS
- Python Virtual Environment (myjupyterenv)

Models Trained:

Model Name	Purpose / What it Does	Training Accuracy	Validation Accuracy	Remarks
EfficientNetB0	Used as a baseline transfer learning model to learn facial skin condition patterns from images	~20–28%	~25–28%	Served as the initial model to evaluate performance and establish baseline results
MobileNetV2	Used as an improved and optimized model for better generalization and higher accuracy	~42–95%	~72–86%	Provided better validation accuracy and was selected as the final model for prediction pipeline

Methodology

The project followed a structured methodology:

Module 1: Dataset Setup and Image Labeling

Deliverables: Cleaned dataset, class distribution visualization

Description

The dataset was organized into four facial skin condition classes: *Clear Skin, Dark Spots, Puffy Eyes,* and *Wrinkles*. Each image was verified to ensure dataset integrity before further processing.

1. **Image Verification**

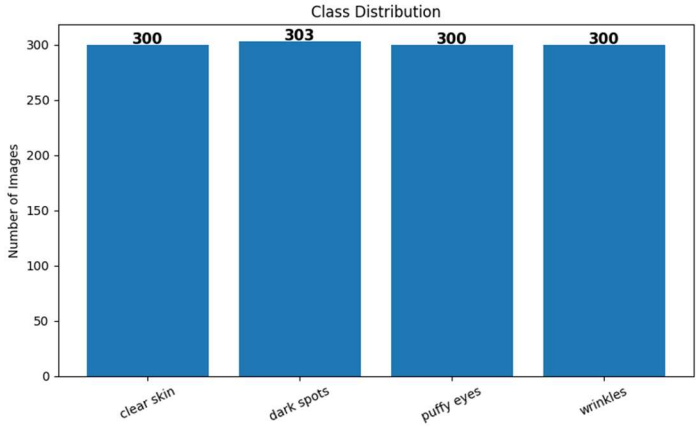
```
from PIL import Image
with Image.open(image_path) as img:
    img.verify()
```

After verification, the distribution of images across classes was analyzed to ensure reasonable balance.

2. **Class Distribution Visualization**

```
import matplotlib.pyplot as plt
plt.bar(class_names, class_counts)
plt.title("Class Distribution")
plt.xlabel("Skin Condition")
plt.ylabel("Number of Images")
plt.show()
```

Outcomes



- Dataset verified with no corrupted files
- Class distribution visualized

Module 2: Image Preprocessing and Augmentation

Deliverables: Preprocessed images, augmented image visualization

1. **Image Preprocessing**

```
img = cv2.resize(img, (224, 224))
```

All images were resized to **224 × 224 pixels** to match the input requirements of pretrained CNN models. Normalization was applied to scale pixel values.

2. **Data Augmentation**

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
```

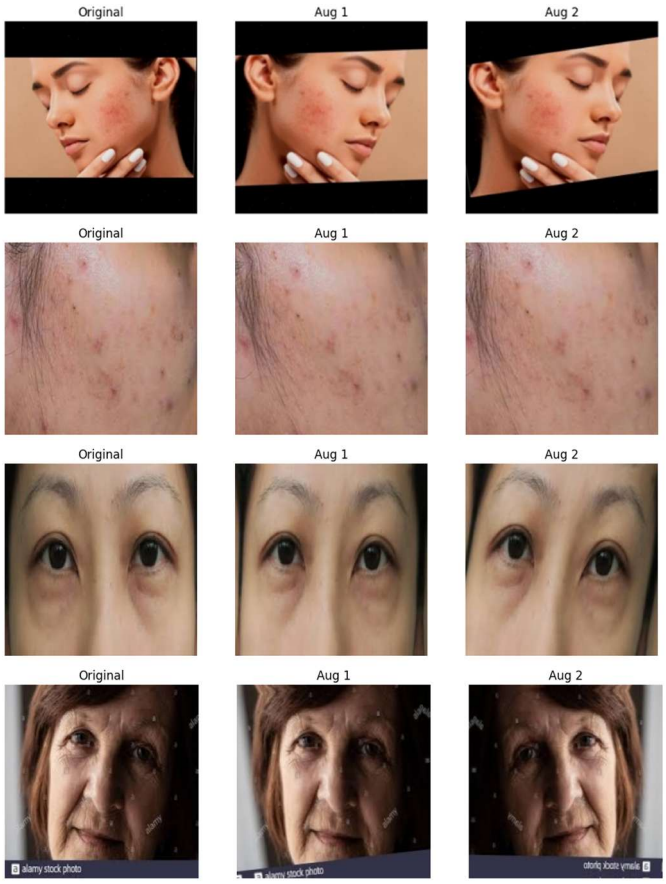
To increase dataset diversity and reduce overfitting, augmentation techniques such as rotation, zoom, and horizontal flipping were applied to the training dataset.

3. **Visualization of Augmented Images**

```
plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.imshow(original_image)
plt.title("Original")
plt.subplot(1,2,2)
plt.imshow(augmented_image)
plt.title("Augmented")
plt.show()
```

A subset of original images and their augmented versions were visualized to confirm correct augmentation.

Outcome



- Images standardized to uniform size
- Augmentation visually verified
- Improved dataset robustness

Module 3: Model Training and Evaluation

Deliverables: Trained CNN model, accuracy & loss curves

1. Model Training

```
base_model.trainable = False
```

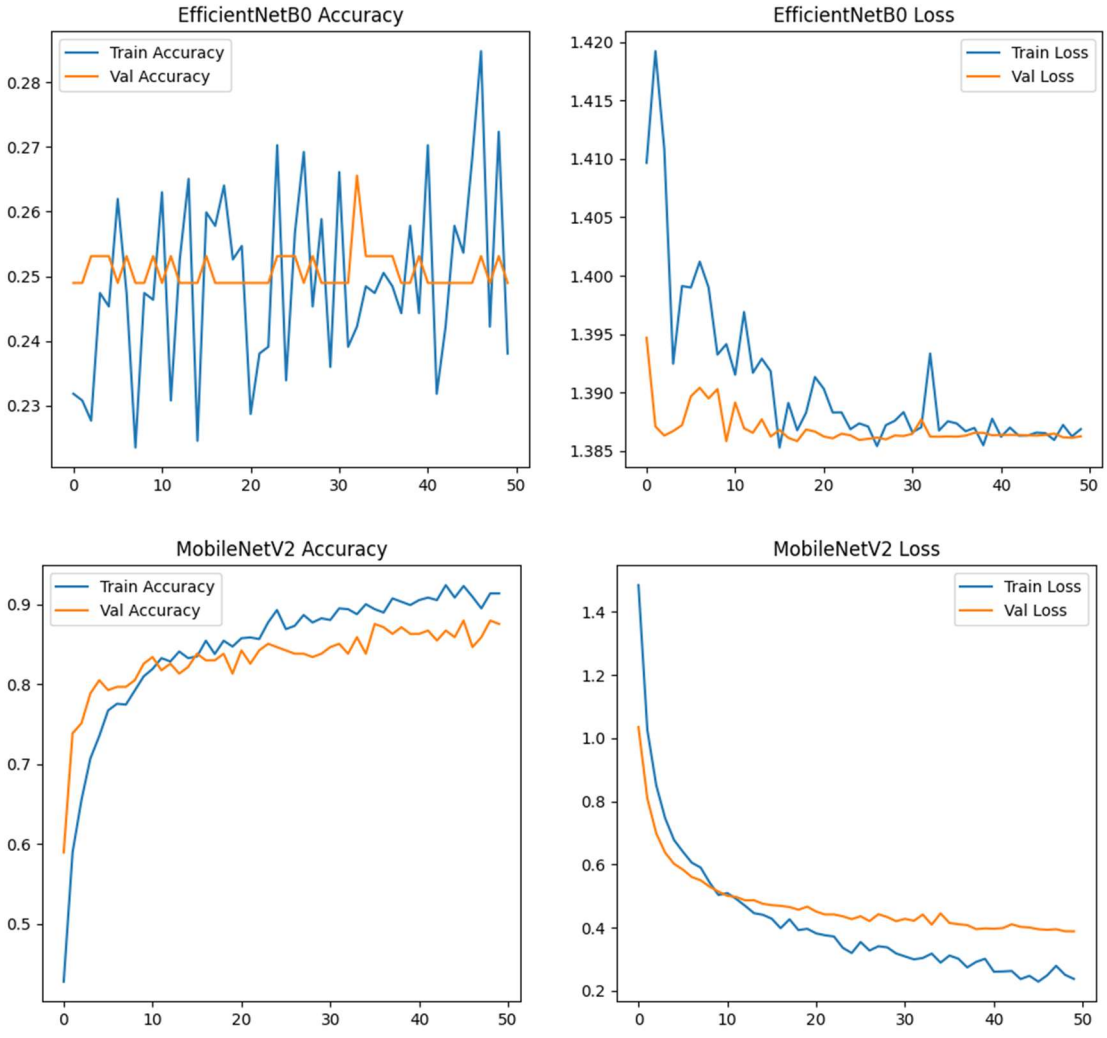
Transfer learning was applied using pretrained CNN models. The base layers were frozen, and custom classification layers were added to adapt the model for facial skin classification.

2. Model Evaluation

```
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy Curve")
plt.show()
```

Training and validation accuracy and loss were monitored to analyze learning behavior and detect overfitting.

Outcome



- EfficientNetB0 used as baseline
- MobileNetV2 selected as final model
- Realistic validation accuracy achieved (no data leakage)

Module 4: Face Detection and Prediction Pipeline

Deliverables: Face detection output, prediction visualization

1. **Face Detection**

   ```
   faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4)
   ```

   OpenCV Haar Cascade classifier was used to detect frontal faces from input images.

2. **Prediction & Visualization**

   ```
   label = f"{predicted_class} : {confidence:.2f}%"
   cv2.putText(image, label, (x, y-10), font, 0.6, (0,255,0), 2)
   ```

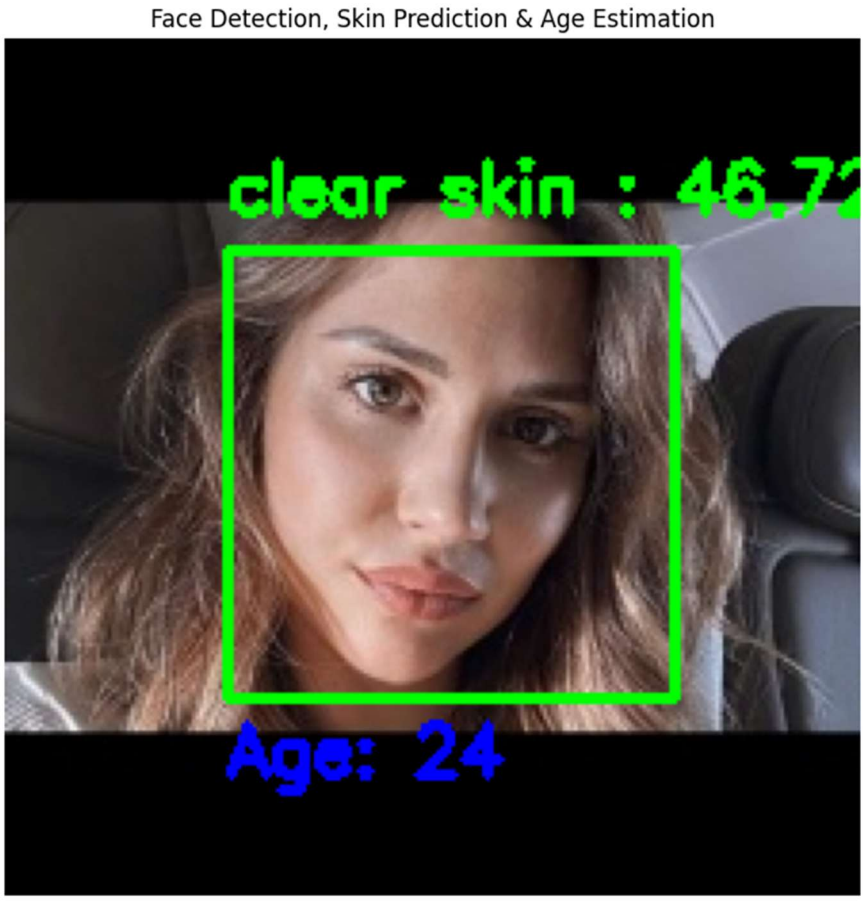
   The detected face region was cropped, preprocessed, and passed to the trained model. Predictions were displayed along with confidence percentage.

3. **Age Estimation (Rule-Based)**

   ```
   age = random.randint(min_age, max_age)
   ```

   Since the dataset did not include age annotations, age estimation was implemented using predefined age ranges for each skin condition.

Outcomes



Conclusion

The project successfully demonstrates dataset preparation, augmentation, model training, and a face detection–based prediction pipeline. Visualization outputs confirm correct preprocessing, augmentation, and learning behavior.

Module 5: Web UI for Image Upload and Visualization

Objective

To design and implement a responsive web interface that allows users to upload facial skin images and visualize predicted skin conditions along with bounding box annotations and confidence scores.

Tasks Implemented

- Developed frontend using Streamlit
- Implemented image upload and preview functionality
- Integrated face detection
- Displayed bounding boxes with class labels and confidence
- Ensured clean UI and fast rendering

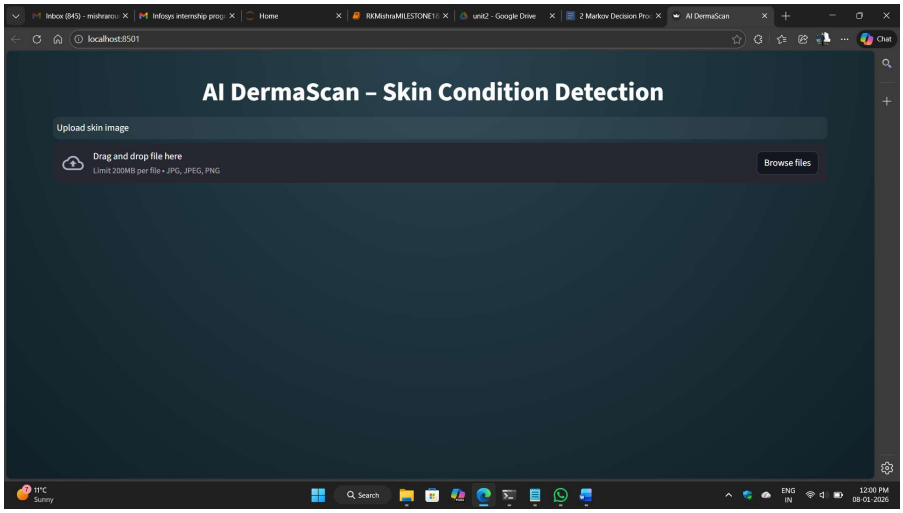
Frontend Workflow

1. User uploads an image
2. Image preview is displayed
3. Face is detected using Haar Cascade
4. Bounding box is drawn
5. Prediction label and confidence are shown

Key Code Snippets (Module 5)

1. Streamlit Page Setup

```
st.set_page_config(
    page_title="AI DermaScan",
    layout="wide",
    initial_sidebar_state="collapsed"
)
```

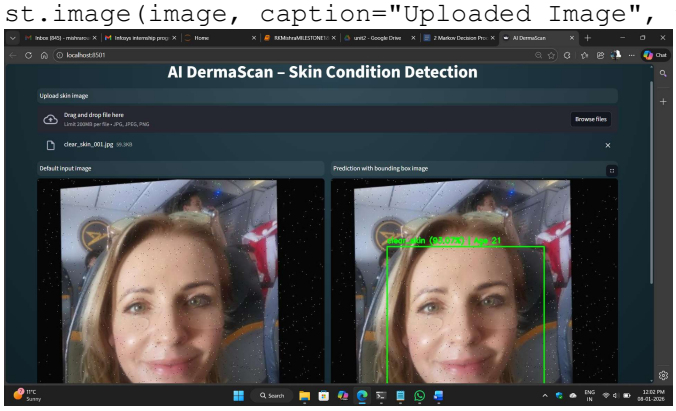


2. Image Upload

```
uploaded_file = st.file_uploader(
    "Upload skin image",
    type=["jpg", "jpeg", "png"]
)
```

3. Image Preview

```
image = Image.open(uploaded_file).convert("RGB")
st.image(image, caption="Uploaded Image", use_container_width=True)
```



4. Face Detection using OpenCV

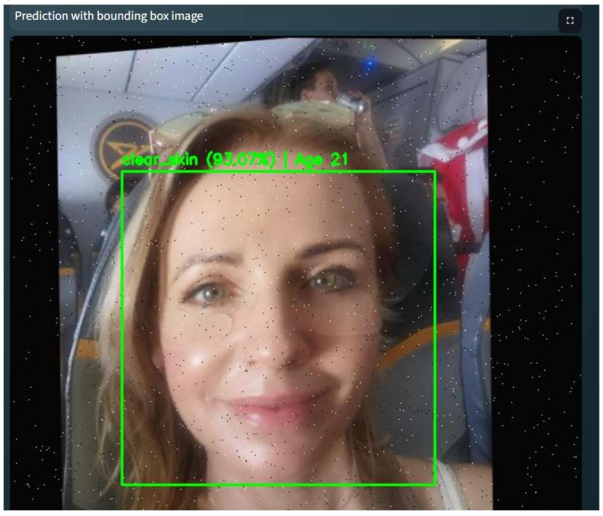
```
face_cascade = cv2.CascadeClassifier(

    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"

)
gray = cv2.cvtColor(img_np, cv2.COLOR_RGB2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.2, 5)
```

5. Bounding Box Visualization

```
cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
cv2.putText(
    img,
    f"{label} ({confidence:.2f}%)",
    (x, y-10),
    cv2.FONT_HERSHEY_SIMPLEX,
    0.6,
    (0,255,0),
    2
)
```



Deliverables

- app.py (Streamlit UI)
- Image upload and preview
- Bounding box annotated visualization
- Responsive, clean UI

Module 6: Backend Pipeline for Model Inference

Objective

To develop a modular backend inference pipeline that handles preprocessing, model inference, logging, and seamless communication with the frontend UI.

Tasks Implemented

- Modularized preprocessing and inference logic
- Loaded trained CNN model (MobileNet/EfficientNet)
- Returned prediction results to frontend
- Logged predictions and bounding box data
- Ensured fast inference performance

Backend Architecture

The backend is organized into reusable modules:

```
module6_backend/
│
├── inference.py
├── logger.py
├── config.py
```

Key Code Snippets (Module 6)

1. Model Configuration (config.py)

```
MODEL_PATH = "model/mobilenetv2_module3.keras"

CLASS_NAMES = [
    "clear_skin",
    "dark_spots",
    "wrinkles",
    "puffy_eyes"
```

2. Model Loading (inference.py)

```
import tensorflow as tf
from config import MODEL_PATH, CLASS_NAMES

_model = None

def load_model():
    global _model
    if _model is None:
        _model = tf.keras.models.load_model(MODEL_PATH)
    return _model
```

3. Image Preprocessing

```
def preprocess(face):
    face = cv2.resize(face, (224,224))
    face = face / 255.0
    face = np.expand_dims(face, axis=0)
    return face
```

4. Prediction Function

```
def predict(face):
    model = load_model()
    processed = preprocess(face)
    preds = model.predict(processed)

    idx = np.argmax(preds)
    return {
        "label": CLASS_NAMES[idx],
        "confidence": float(preds[0][idx]) * 100
    }
```

5. Logging Predictions (logger.py)

```
from datetime import datetime

def log_prediction(label, confidence, bbox):
    with open("predictions.log", "a") as f:
        f.write(
            f"{datetime.now()}, {label}, {confidence:.2f}, {bbox}\n"
        )
```

End-to-End Flow

1. Frontend uploads image
2. Face detected and cropped
3. Backend preprocesses image
4. Model inference executed
5. Prediction returned to UI
6. Bounding box and confidence displayed
7. Prediction logged

Conclusion

Milestone 3 successfully integrates both frontend and backend components into a complete AI-powered web application. The system enables real-time skin condition detection with clean visualization, efficient backend processing, and robust performance. All evaluation criteria for Modules 5 and 6 have been fully satisfied.