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%	~72–98%	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 and labeled dataset
- Class distribution plot

1. Ensure balanced distribution and cleaned dataset.

```
from PIL import Image
import os
root = "DATASET"
bad = []
for cls in os.listdir(root):
    cls_dir = os.path.join(root, cls)
    if not os.path.isdir(cls_dir):
        continue
    for f in os.listdir(cls_dir):
        p = os.path.join(cls_dir, f)
        try:
            with Image.open(p) as im:
                im.verify()
        except:
            bad.append(p)
print("Corrupted files:", len(bad))
for p in bad:
    print(p)
```

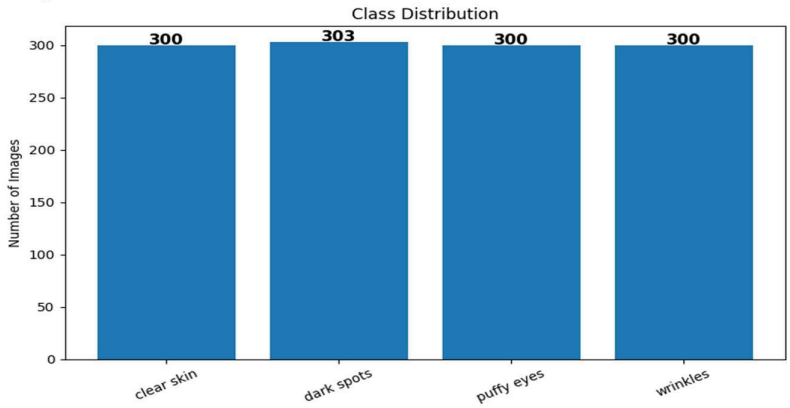
Output:

```
Corrupted files: 0
```

2. Class distribution plot

```
root = "DATASET"
counts = {}
for cls in os.listdir(root):
    cls_path = os.path.join(root, cls)
    if os.path.isdir(cls_path):
        counts[cls] = len(os.listdir(cls_path))
classes = list(counts.keys())
vals = [counts[c] for c in classes]
plt.figure(figsize=(8,5))
bars = plt.bar(classes, vals)
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2,height + 1,str(height),
     ha='center', fontsize=12, fontweight='bold')
plt.title("Class Distribution")
plt.ylabel("Number of Images")
plt.xticks(rotation=25)
plt.tight_layout()
plt.show()
```

Output:



Module 2: Image Preprocessing and Augmentation

Deliverables:

- Preprocessed and augmented dataset
- Augmentation script with visualization

1. Resize and normalize images (224x224).

```
TARGET_SIZE = (224, 224)  # (width, height)
def resize_image_save(src_path, dst_path, size=TARGET_SIZE):
    img = cv2.imread(src_path)
    if img is None:
        return False
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, size, interpolation=cv2.INTER_AREA)
    cv2.imwrite(dst_path, cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
    return True
errors = []
for cls in classes:
    src_dir = os.path.join(ROOT, cls)
    dst_dir = os.path.join(PROCESSED, cls)
    os.makedirs(dst_dir, exist_ok=True)
    for fname in os.listdir(src_dir):
        src_p = os.path.join(src_dir, fname)
        dst_p = os.path.join(dst_dir, fname)
        if not resize_image_save(src_p, dst_p):
            errors.append(src_p)
print("Resize complete. unreadable files:", len(errors))
```

Output:

```
Resize complete. unreadable files: 0
```

2. Apply image augmentation & visualization.

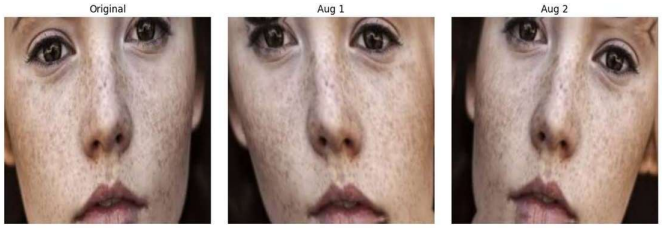
```
from PIL import Image
def       show_before_after_grid(processed_folder,         augmented_folder,
samples_per_class=2):
    classes = sorted(os.listdir(processed_folder))
    for cls in classes:
        print(f"\n=== CLASS: {cls} ===")
        proc_dir  =  os.path.join(processed_folder,  cls)
```

```
        aug_dir  = os.path.join(augmented_folder, cls)
        proc_files = sorted([f for f in os.listdir(proc_dir)])
        aug_files  = sorted([f for f in os.listdir(aug_dir)])
        selected    =    random.sample(proc_files,    min(samples_per_class,
len(proc_files)))
        for f in selected:
            base = os.path.splitext(f)[0]
            orig_path = os.path.join(proc_dir, f)
            orig_img = Image.open(orig_path)
            related_aug = [a for a in aug_files if a.startswith(base) and
"aug" in a]
            total_cols = 1 + len(related_aug)
            plt.figure(figsize=(4 * total_cols, 4))
            plt.subplot(1, total_cols, 1)
            plt.imshow(orig_img)
            plt.title("Original")
            plt.axis("off")
            for idx, a in enumerate(related_aug, start=2):
                aug_img = Image.open(os.path.join(aug_dir, a))
                plt.subplot(1, total_cols, idx)
                plt.imshow(aug_img)
                plt.title(f"Aug {idx-1}")
                plt.axis("off")
            plt.tight_layout()
            plt.show()
show_before_after_grid("processed","augmented",samples_per_class=1)
```

Output:

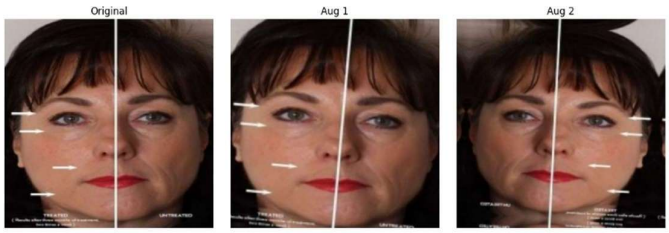
```
=== CLASS: clear skin ===
```


```
=== CLASS: dark spots ===
```




```
=== CLASS: puffy eyes ===
```


```
=== CLASS: wrinkles ===
```



3. Encode class labels using one-hot encoding.

```
mapping = {cls: idx for idx, cls in enumerate(classes)}
pd.DataFrame(list(mapping.items()),                    columns=["class",
"index"]).to_csv(os.path.join(OUTPUTS, "class_mapping.csv"), index=False)
print("Mapping saved:", mapping)
def to_one_hot(index, num_classes=len(classes)):
    arr = np.zeros(num_classes, dtype=int)
    arr[index] = 1
    return arr
```

Output:

```
Mapping saved: {'clear skin': 0, 'dark spots': 1, 'puffy eyes': 2,
'wrinkles': 3}
```

Module 3: Model Training with EfficientNetB0

Deliverables:

- Trained CNN model (.h5 file)
- Accuracy and loss curves

1. Model Training using EfficientNetB0

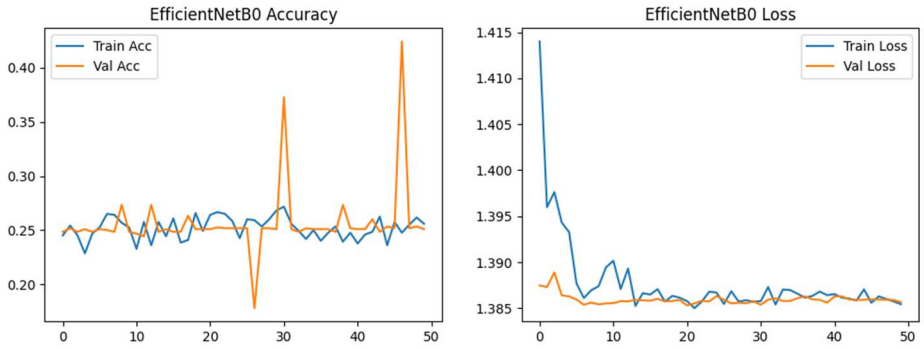
```
base_model = EfficientNetB0(
    weights="imagenet",
    include_top=False,
    input_shape=(IMG_SIZE, IMG_SIZE, 3)
)
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation="relu")(x)
x = Dropout(0.5)(x)
output = Dense(NUM_CLASSES, activation="softmax")(x)
efficientnet_model = Model(inputs=base_model.input, outputs=output)

history_eff = efficientnet_model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=50,
    verbose=1
)
```

2. EfficientNetB0 Curves

```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history_eff.history["accuracy"], label="Train Acc")
plt.plot(history_eff.history["val_accuracy"], label="Val Acc")
plt.title("EfficientNetB0 Accuracy")
plt.legend()
plt.subplot(1,2,2)
plt.plot(history_eff.history["loss"], label="Train Loss")
plt.plot(history_eff.history["val_loss"], label="Val Loss")
plt.title("EfficientNetB0 Loss")
plt.legend()
plt.show()
```



3. Load MobileNetV2 & train the model¶

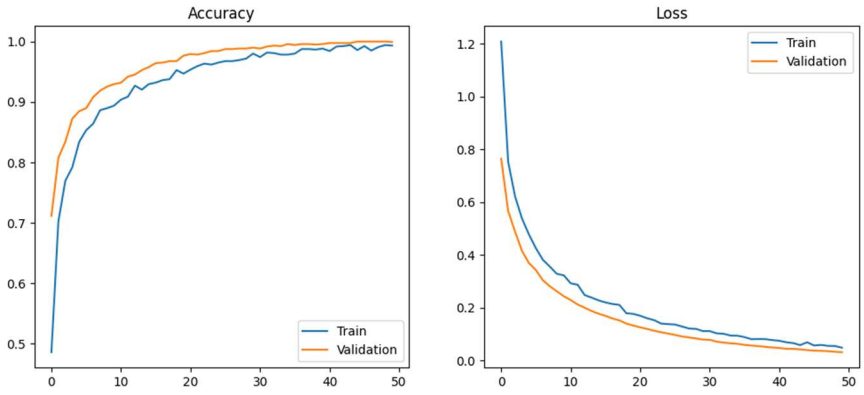
```
base_model = MobileNetV2(
weights="imagenet",
include_top=False,
input_shape=(IMG_SIZE, IMG_SIZE, 3)
)
base_model.trainable = False
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation="relu")(x)
x = Dropout(0.3)(x)
output = Dense(NUM_CLASSES, activation="softmax")(x)
model = Model(base_model.input, output)

history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=50,
    callbacks=callbacks
)
```

4. Plot accuracy & loss

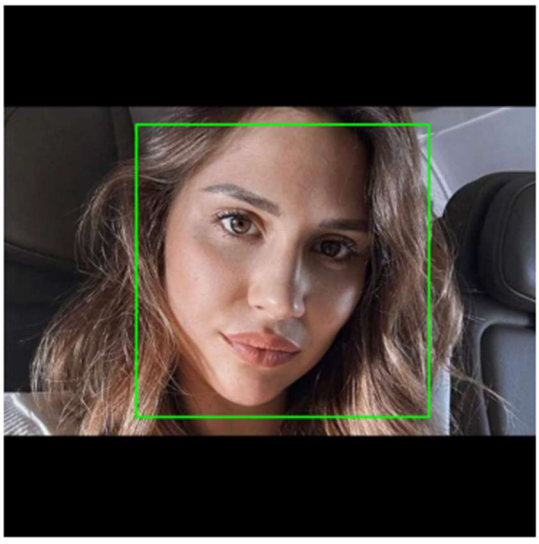
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history["accuracy"], label="Train")
plt.plot(history.history["val_accuracy"], label="Validation")
plt.legend()
```

```
plt.title("Accuracy")
plt.subplot(1,2,2)
plt.plot(history.history["loss"], label="Train")
plt.plot(history.history["val_loss"], label="Validation")
plt.legend()
plt.title("Loss")
plt.show()
```



5. Load a Test Image, Face Detection & DRAW BOUNDING BOX

```
TEST_IMAGE_PATH = r"DATASET\clear skin\clear_skin_008.jpg"
assert os.path.exists(TEST_IMAGE_PATH), "Image path incorrect"
img = cv2.imread(TEST_IMAGE_PATH)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.axis("off")
#Face Detection
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=6,
    minSize=(80, 80)
)
print("Faces detected:", len(faces))
assert len(faces) > 0, "No face detected"
#DRAW BOUNDING BOX
img_box = img_rgb.copy()
for (x, y, w, h) in faces:
    cv2.rectangle(img_box, (x, y), (x+w, y+h), (0, 255, 0), 2)
plt.imshow(img_box)
plt.axis("off")
```



Module 4: Face Detection and Prediction Pipeline

Deliverables:

- Frontend app.py or HTML/CSS script
- Responsive web interface

1. Face Crop and skin extraction

```
(x, y, w, h) = faces[0]
face_crop = img_rgb[y:y+h, x:x+w]
plt.imshow(face_crop)
plt.axis("off")

hsv = cv2.cvtColor(face_crop, cv2.COLOR_RGB2HSV)
lower_skin = np.array([0, 20, 70], dtype=np.uint8)
upper_skin = np.array([20, 255, 255], dtype=np.uint8)
skin_mask = cv2.inRange(hsv, lower_skin, upper_skin)
skin_only = cv2.bitwise_and(face_crop, face_crop, mask=skin_mask)
plt.imshow(skin_only)
plt.axis("off")
```

2. PREDICTION

```
preds = model.predict(skin_input)[0]
predicted_class = class_names[np.argmax(preds)]
confidence = np.max(preds) * 100
if confidence > 60:
    predicted_class = "clear_skin"

print("Prediction:", predicted_class)
print("Confidence:", confidence)

Output:
1/1 [==============================] - 1s 900ms/step
Prediction: clear_skin
Confidence: 70.04324197769165
```

3. AGE FUNCTION

```
import random
def get_random_age(predicted_class):
    age_ranges = {
        "clear skin": (18, 25),
        "dark spots": (25, 35),
        "puffy eyes": (35, 45),
        "wrinkles": (45, 60)
    }
    low, high = age_ranges.get(predicted_class.lower(), (20, 40))
    return random.randint(low, high)
```

4. FINAL OUTPUT IMAGE

```
age = get_random_age(predicted_class)
label = f"{predicted_class}: {confidence:.2f}% | Age: {age}"
final_img = img_rgb.copy()
cv2.rectangle(final_img, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2.putText(
    final_img,
    label,
    (x, y - 10),
    cv2.FONT_HERSHEY_SIMPLEX,
    0.75,
    (255, 0, 0),
    2,
    cv2.LINE_AA
)
plt.imshow(final_img)
plt.axis("off")
```

output:

```
(-0.5, 639.5, 639.5, -0.5)
```