# Internship Project Report

DermalScan: AI Facial Skin Aging Detection App

Submitted By:

**Kamsali Niharika**

Under guidance of Mentor **Mr. Praveen**

**Infosys Springboard Virtual Internship**

**Problem Statement**

Develop an AI-powered facial skin–aging detection system using EfficientNetB0 that classifies aging indicators such as **wrinkles, dark spots, puffy eyes, and clear skin** from uploaded images.
The pipeline includes:

- Face detection via Haar Cascades
- Preprocessing & augmentation
- Deep learning classification
- Web-based visualization with bounding boxes
- Prediction export and logging

**Objectives**

- Build a deep learning classifier using **EfficientNetB0** with at least **90% accuracy**.
- Detect aging regions and output **percentage-based predictions**.
- Create a **Streamlit UI** for real-time inference (≤5 seconds).
- Prepare dataset → preprocess → augment → train → evaluate → deploy.

**Milestone 1: Dataset Preparation & Preprocessing**

**Module 1: Dataset Setup and Image Labeling**

The dataset was manually curated and organized into four classes:

- **puffy_eyes**
- **wrinkles**
- **dark_spots**
- **clear_skin**

Each image was placed into the corresponding folder and renamed in a structured format (e.g., puffy_eyes_1.jpg, wrinkles_42.jpg) using an automated Python renaming script.

## 1. Image Counting per Class

### Sample Code:

```
for cls in CLASSES:
    folder = DATA_DIR / cls
    count = len(list(folder.glob("*.*")))
    print(cls, ":", count)
```

**Purpose:** ensure that the dataset is correctly loaded and classes are balanced.

## 2. Class Distribution Visualization

### Sample Code:

```
sns.barplot(x=df_counts.index, y=df_counts['count'])
plt.title("Number of Images per Class")
plt.xlabel("Class")
plt.ylabel("Image Count")
plt.show()
```

### Output:

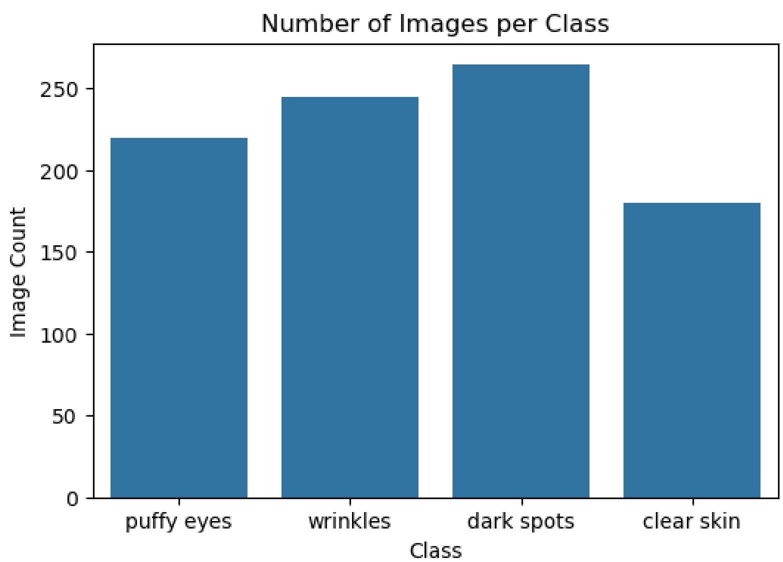



Fig – 1

## 3. Sample Image Visualization

### Sample Code:

show_samples("clear skin", n=6)

### Output:


This helped confirm:

- Images were placed in correct categories

## 4. Image Dimension Analysis

### Sample Code:

sns.histplot(df_sizes["width"], kde=True, label="Width")

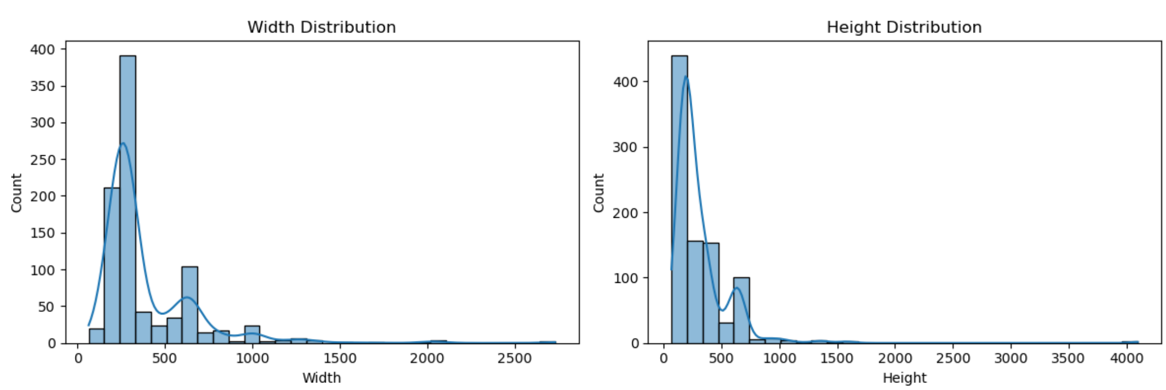
sns.histplot(df_sizes["height"], kde=True, label="Height")

plt.legend()

plt.title("Image Width & Height Distribution")

plt.show()

### Output:




Ensured consistency in image shapes before resizing.

## 5. Brightness Distribution per Class

A KDE plot was generated to analyze lighting differences between classes:

### Code:

```
sns.kdeplot(data=df_bright, x="brightness", hue="class", fill=True)
```

### Output:

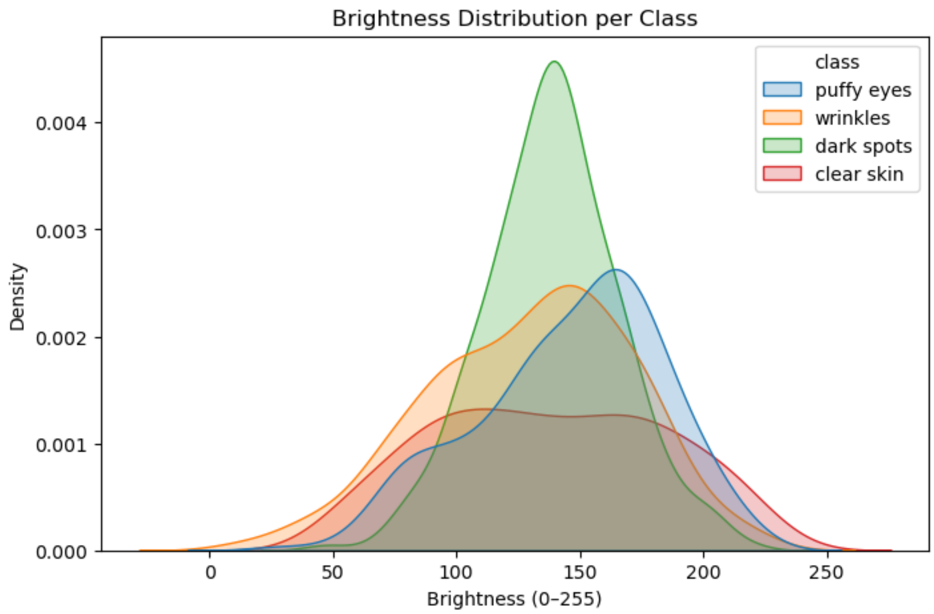



Fig – 4

## Module 2: Image Preprocessing and Augmentation

Images were resized, normalized, augmented, and label – encoded as required by EfficientNetB0.

## 1. Resizing & Normalizing (224×224):

### Sample Code:

```
img = Image.open(img_path).convert("RGB")

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

img = np.array(img) / 255.0
```

**Output:**

X shape: (908, 224, 224, 3)

y shape: (908, 4)

## 2. One-Hot Encoding of Labels

**Sample Code:**

```
y_encoded = tf.keras.utils.to_categorical(labels, num_classes=4)
```

## 3. Data Augmentation

**Sample Code:**

```
datagen = ImageDataGenerator(
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)
aug_iter = datagen.flow(sample_img, batch_size=1)
plt.imshow(next(aug_iter)[0])
```

**Output:**

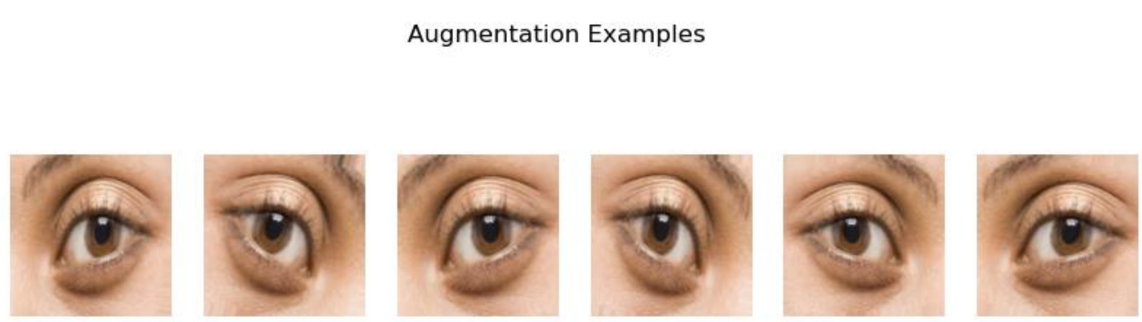



Fig – 5 (Rotation, Zoom, Flip)

## 6. Augmentation Quality Visualization

## Sample Code:

```
plt.figure(figsize=(8,4))
sns.histplot(X.ravel(), bins=50, color="blue", label="Original", stat="density")

aug_batch = datagen.flow(X, y, batch_size=100)
augmented_sample, _ = next(aug_batch)

sns.histplot(augmented_sample.ravel(), bins=50, color="red", label="Augmented",
stat="density")

plt.legend()
plt.title("Pixel Intensity Distribution Before vs After Augmentation")
plt.show()
```

## Output:

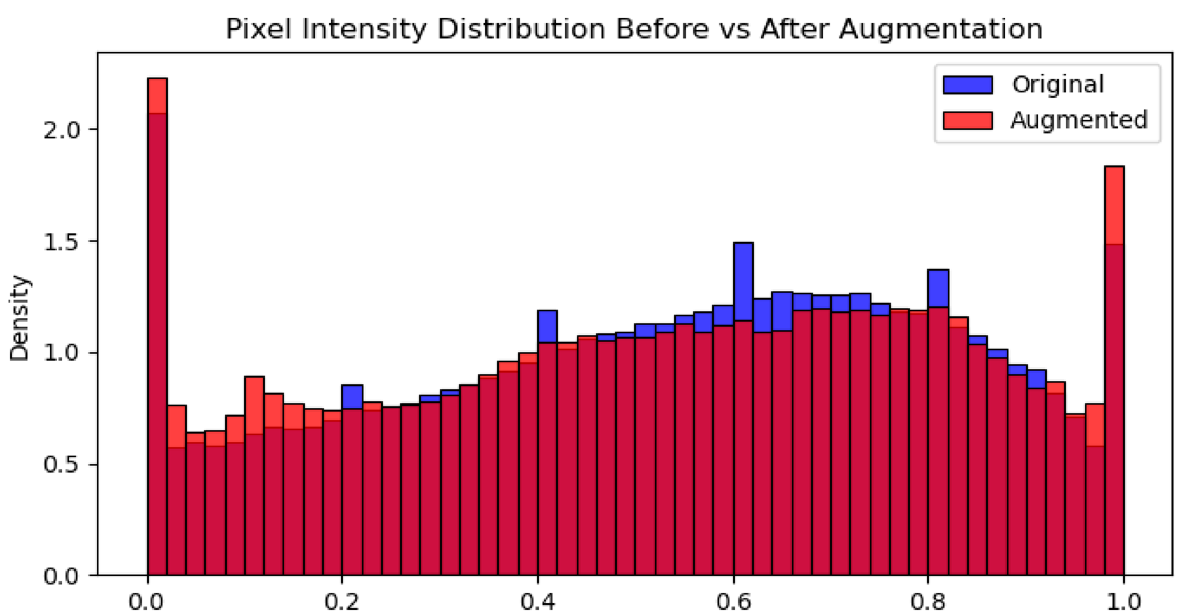



Fig – 6

After preprocessing and augmentation, the final dataset was split into training, validation, and testing subsets using the stratified train-test split method to maintain class balance.
An 80/10/10 ratio was used, which is widely accepted for deep learning tasks.
All six final arrays (X_train, y_train, X_val, y_val, X_test, y_test) were saved for efficient loading during model training.

# Milestone – 2: Model Training & Evaluation

## Module 3: Model Development & Training

### 1. Objective:

To build a reliable deep-learning model that classifies:
Wrinkles • Dark Spots • Puffy Eyes • Clear Skin

### 2. Dataset Improvement:

To ensure richer learning and better generalization:

- Increased dataset from ~300 → **~500 images per class**
- Final usable dataset ≈ **1800+ images**
- Balanced all classes
- Removed noisy, tiny & corrupt images
- Standardized input to **224×224 resolution**

### 3. Model Selection:

Chosen Model: **EfficientNet (Fine-Tuned)**
Reason:

- Excellent accuracy-efficiency balance
- Strong feature learning for skin textures
- Stable convergence

### 4. **Training Strategy**

**Phase-1:**
Freeze EfficientNet → Train Classification Head

**Phase-2:**
Unfreeze selected layers → Fine-tune at lower LR

5. **Training Configuration**

- Input Size: 224×224
- Optimizer: Adam
- Batch Size: 32
- Epochs: 50+
- Loss: Categorical Crossentropy
- Regularization: Dropout + EarlyStopping

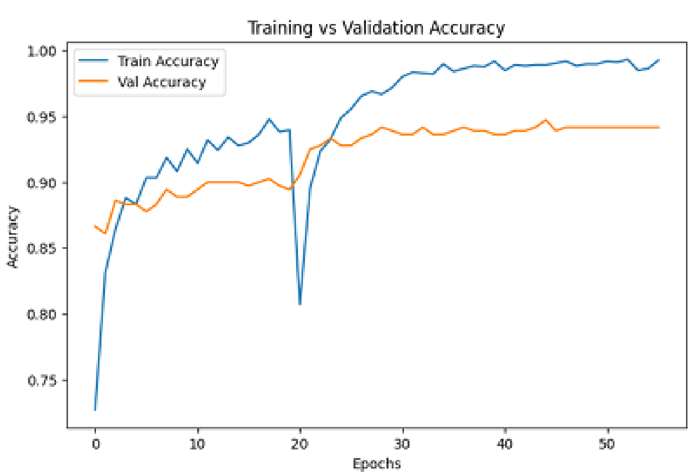
6. **Performance**

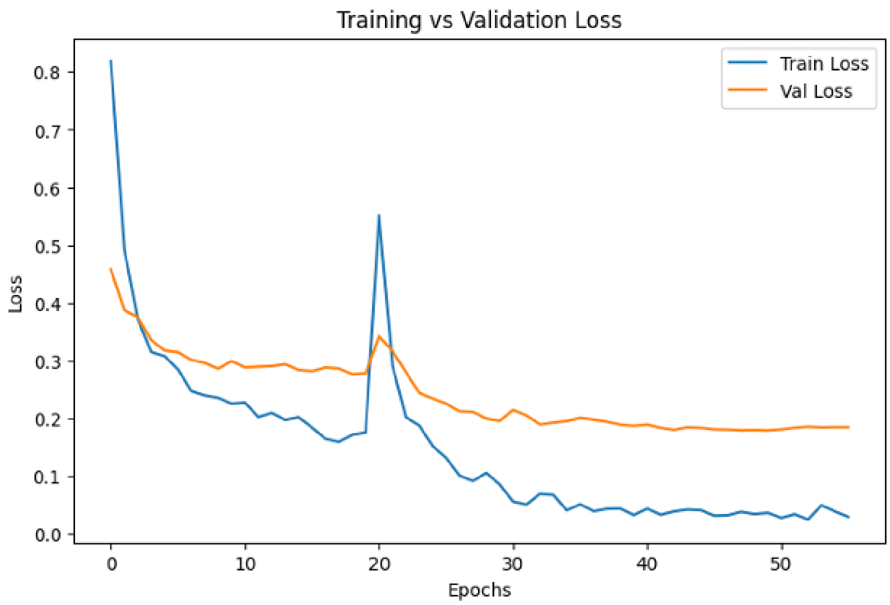
- Training Accuracy: ≈ **99%**
- Validation Accuracy: ≈ **94%**
- Stable curves (no heavy overfitting)
- Strong confusion matrix behavior

Final Model Selected → EfficientNet Fine-Tuned

## 7. Model Comparison Table

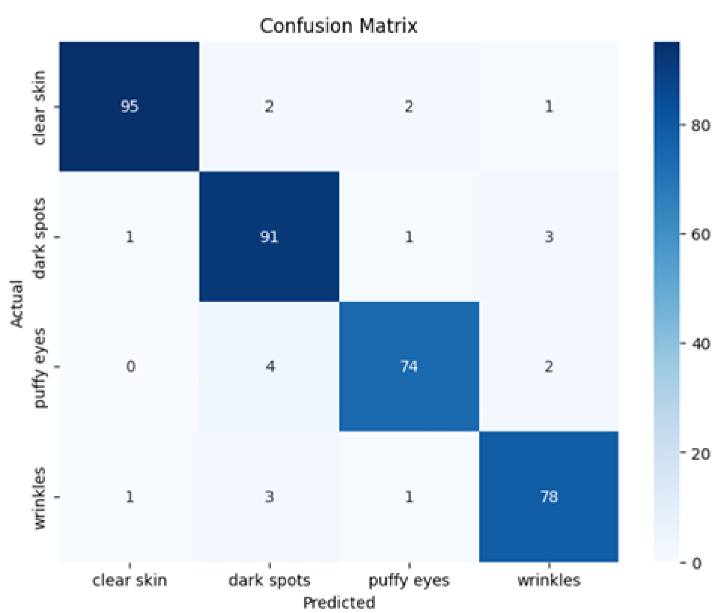
| **Model Variant** | **Train Acc** | **Val Acc** | **Epochs** | **Batch** |
|---|---|---|---|---|
| EfficientNet – Phase 1 | ~93% | ~88% | 20 | 32 |
| EfficientNet – Fine Tune | **~99%** | **~94%** | 40 | 32 |






## Classification Report:

| Label | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| clear skin | 0.98 | 0.95 | 0.96 | 100 |
| dark spots | 0.91 | 0.95 | 0.93 | 96 |
| puffy eyes | 0.95 | 0.93 | 0.94 | 80 |
| wrinkles | 0.93 | 0.94 | 0.93 | 83 |
| accuracy | | | 0.94 | 359 |
| macro avg | 0.94 | 0.94 | 0.94 | 359 |
| weighted avg | 0.94 | 0.94 | 0.94 | 359 |



## Module 4: Facial Region Detection & Prediction Pipeline

### 1. **Objective**

Integrate AI model with:

- Face Detection
- Region Identification
- Age Estimation
- Final Output Overlay

### 2. **Objective**

Integrate AI model with:

- Face Detection
- Region Identification
- Age Estimation
- Final Output Overlay

### 3. **Face & Region Detection**

Implemented:

- Haar Cascade Face Detection
- Mediapipe Landmark-Based Region Extraction:
    - Forehead
    - Left Eye
    - Right Eye
    - Left Cheek
    - Right Cheek

Works on different angles & lighting, Regions verified visually.

### 4. **Age Estimation**:

- Implemented **real integer age prediction.**
- Returns values like **21, 22, 23…**

5. **Final AI Prediction Output**

For each face system outputs:

- Wrinkles %
- Dark Spots %
- Puffy Eyes %
- Clear Skin %
- **Dominant Condition Highlighted**
- Predicted Integer Age

Displayed on image with clean black bounding box + label.