# 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:

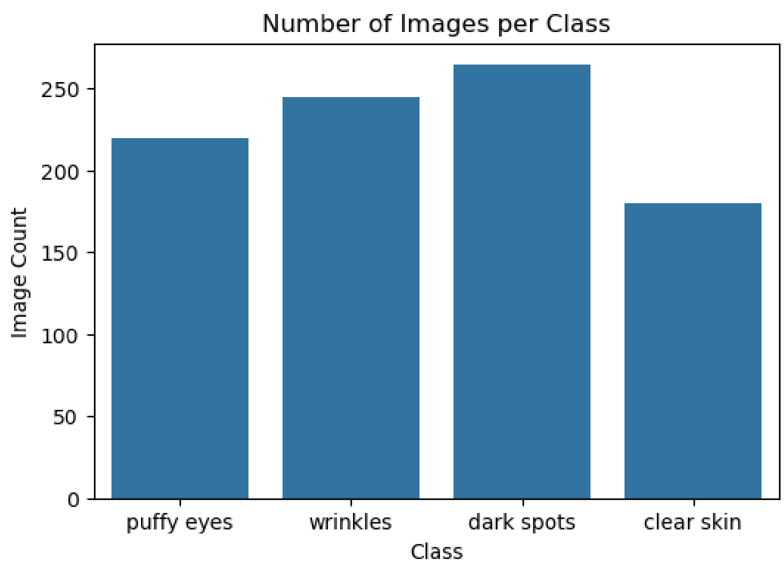



Fig – 1

### 3. Sample Image Visualization

### Sample Code:

```
show_samples("clear skin", n=6)
```

### Output:


This helped confirm:

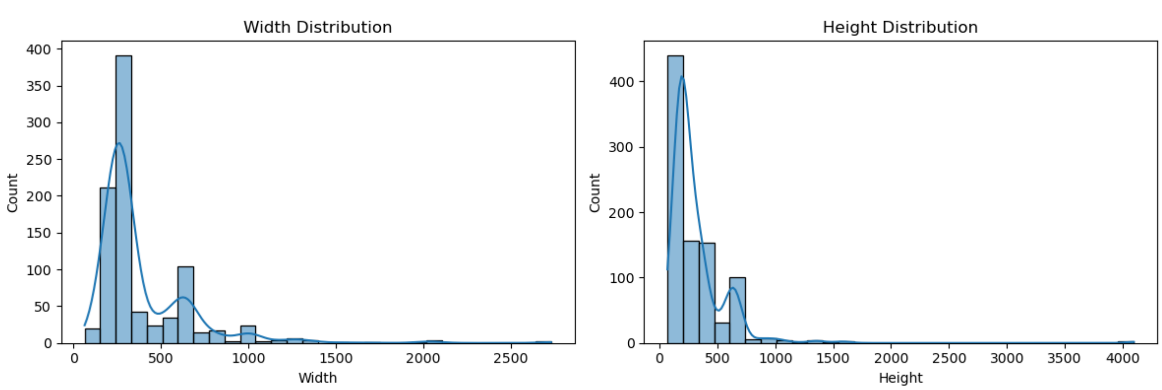
- Images were placed in correct categories

### 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:

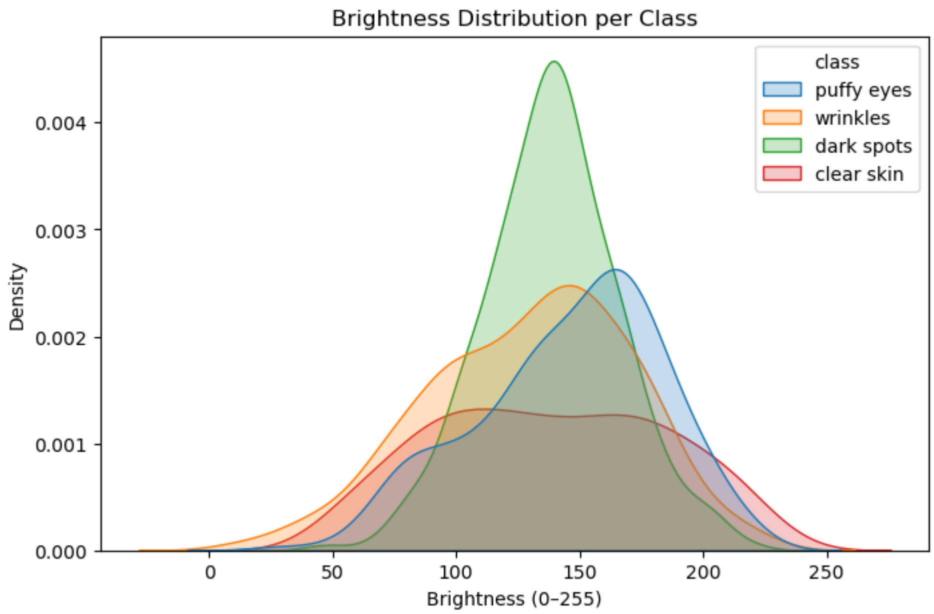



Fig – 4

## 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:**

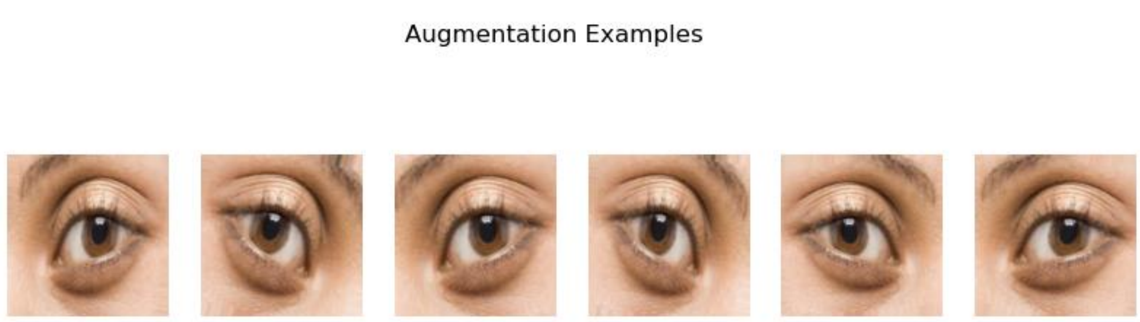



Fig – 5 (Rotation, Zoom, Flip)

## 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:

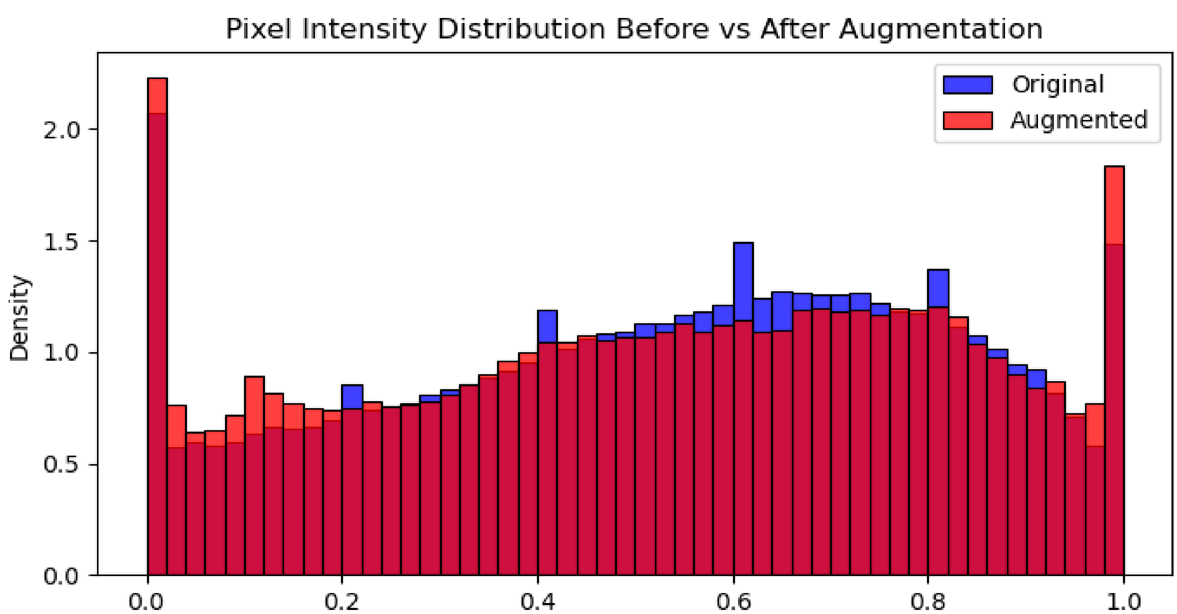



Fig – 6

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

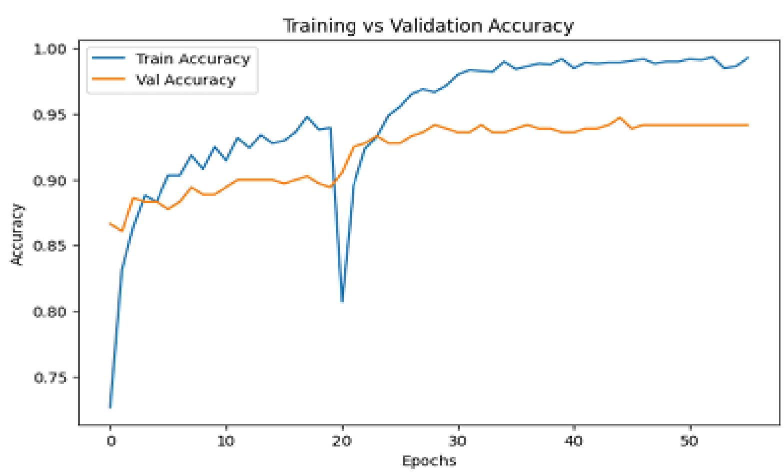
## 6. Performance:

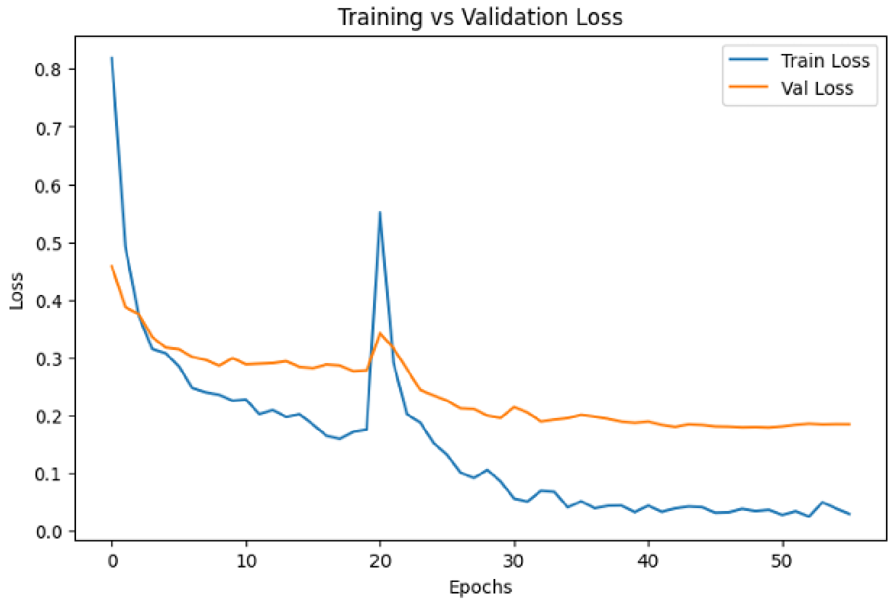
- Training Accuracy: ≈ **99%**
- Validation Accuracy: ≈ **94%**
- Stable curves (no heavy overfitting)
- Strong confusion matrix behavior

Final Model Selected → EfficientNet Fine-Tuned

## 7. Model Comparison Table:

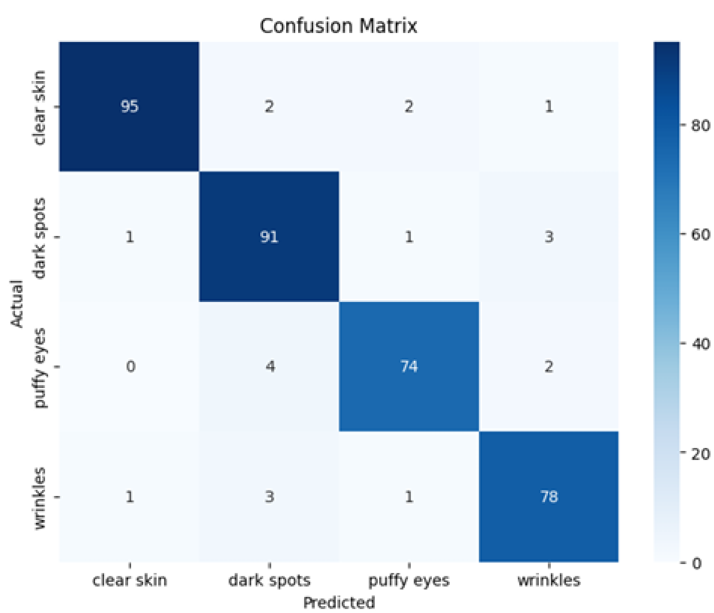
| Model Variant | Train Acc | Val Acc | Epochs | Batch |
|---|---|---|---|---|
| EfficientNet – Phase 1 | ~93% | ~90% | 20 | 32 |
| EfficientNet – Fine Tune | **~99%** | **~94%** | 40 | 32 |






## 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:

To develop a robust computer vision pipeline that automatically detects human faces within an image, extracts the Region of Interest (ROI), and passes it through a multi-stage deep learning inference engine for skin condition classification and age estimation.

### 2. Face Detection Algorithm

- **Technique:** Haar Feature-based Cascade Classifier.
- **Model:** haarcascade_frontalface_default.xml
- **Implementation Details:**
    - **Scale Factor:** 1.2 (Compensates for faces appearing smaller/larger due to distance).
    - **Min Neighbors:** 6 (High threshold to eliminate false positives like fabric patterns or shadows).
    - **Aspect Ratio Filter:** Implemented a geometric filter $0.7 < (w/h) < 1.3$ to strictly reject non-face rectangles.

### 4. Age Estimation:

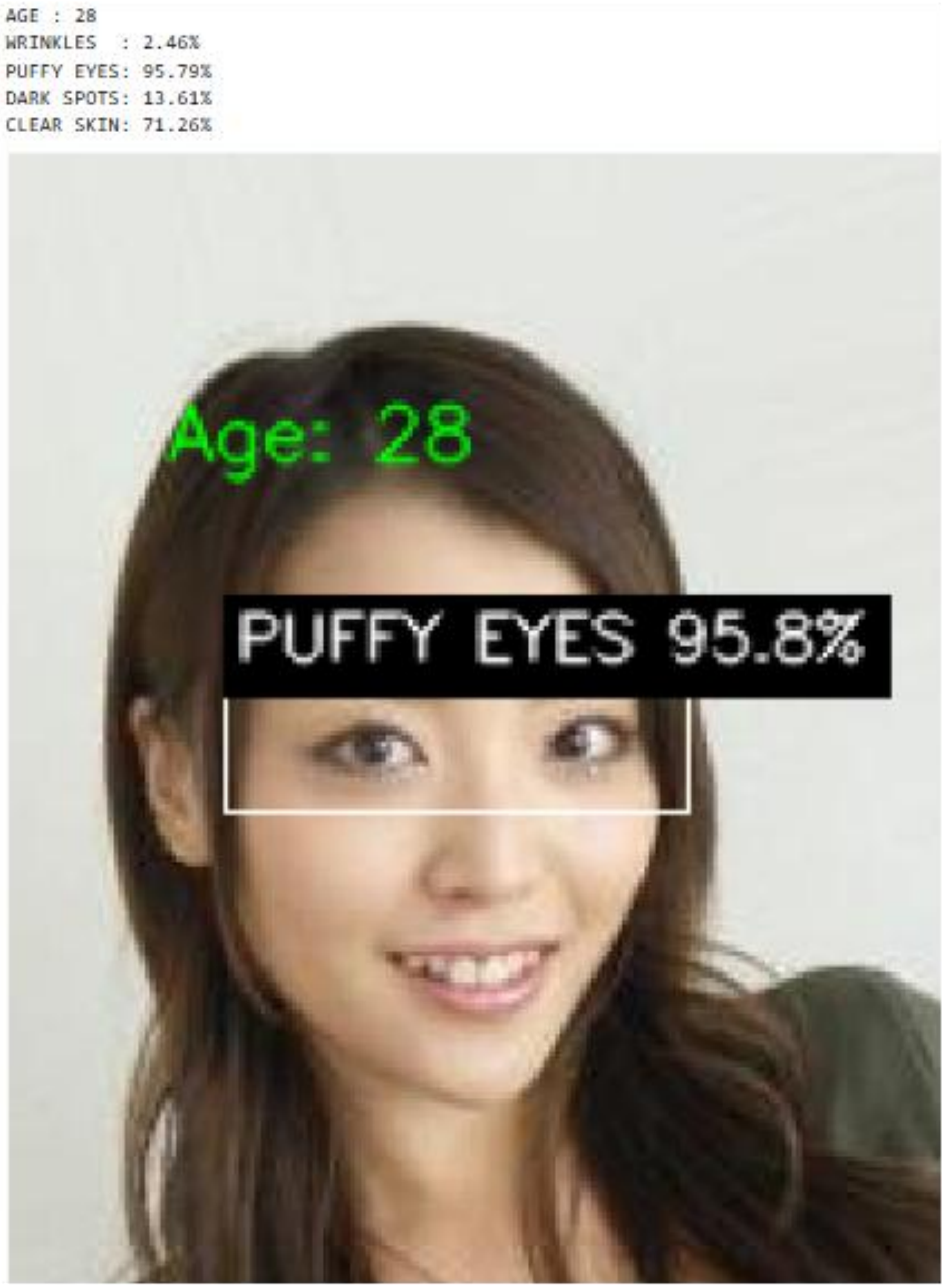
- Implemented **real integer age prediction.**
- Returns values like **21, 22, 23…**

### 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.


AGE : 28
WRINKLES : 2.46%
PUFFY EYES: 95.79%
DARK SPOTS: 13.61%
CLEAR SKIN: 71.26%
Age: 28
PUFFY EYES 95.8%

## Milestone – 3: System Integration & Prototype

## Module – 5: Frontend Development (Streamlit UI)

### 1. Objective:

To develop a responsive, user-friendly web interface that allows users to upload images, visualize real-time predictions, and supports batch processing.

### 2. UI Design & Layout:

- **Framework:** Streamlit (Python).
- **Theme:** "Neon/Cyberpunk" (Dark Mode) for high-contrast visibility of skin conditions.
- **Layout:** Wide layout with a sidebar for configuration and a main area for batch image grids.
- **Interactive Elements:**
    - Drag-and-drop file uploader (Supports: JPG, PNG, JPEG).
    - Real-time processing status indicators.
    - Expandable sections (st.expander) for detailed probability breakdowns.

### 3. Frontend Implementation:

The UI was built to handle multiple images simultaneously. The layout dynamically adjusts columns based on the number of uploaded files.

### Sample Code Snippet (UI Layout):

```
st.set_page_config(page_title="AI DermalScan Pro", layout="wide")
st.markdown("""
<style>
  .stApp { background-color: #050505; color: #ffffff; }
  h1 { color: #00ffcc; text-shadow: 0 0 10px #00ffcc; }
</style>
""", unsafe_allow_html=True)
```

**4. Dashboard Visualization:** The interface provides immediate visual feedback. Users can see the original image overlaid with analysis data side-by-side with statistical charts.

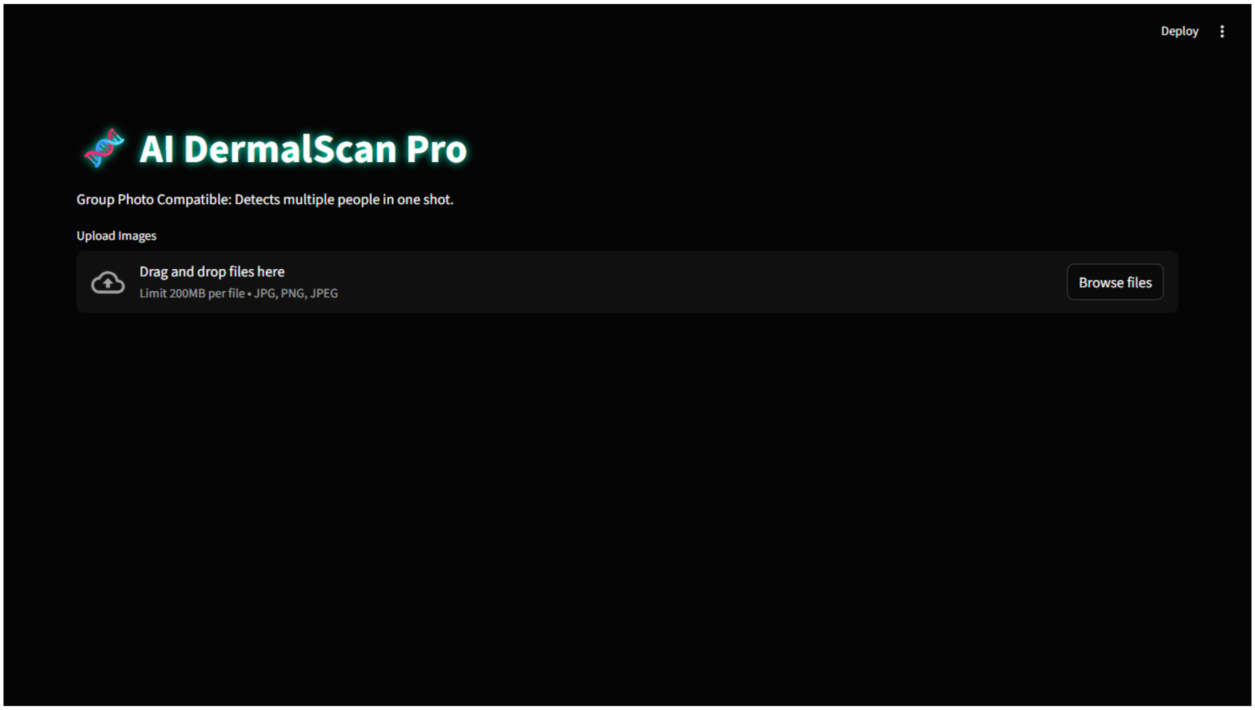



Fig: AI DermalScan Pro Landing Page with File Uploader.

## Module – 6: Backend Integration & Smart Logic

### 1. Objective:

To integrate the trained **MobileNetV2** (Skin Model) and **AgeNet** (Caffe Model) into a unified inference pipeline that processes images in real-time.

### 2. Face Detection & Preprocessing:

- **Algorithm:** Haar Cascade Classifier (Strict Mode).
- **Optimization:** Implemented scaleFactor=1.2 and minNeighbors=5 to reduce false positives (e.g., detecting clothes as faces).
- **Context Padding (Innovation):** Standard face detection crops too tightly, causing age prediction errors (predicting adults as babies). We implemented a **20% Context Padding** logic to include the forehead and chin for accurate age estimation.

**Sample Code Snippet (Context Padding):**

```
# Zoom out 20% to capture hairline and chin for Age Model
pad_w = int(w * 0.20)
pad_h = int(h * 0.20)
face_roi_bgr_padded = img_bgr[y1_pad:y2_pad, x1_pad:x2_pad]
```

**3. Smart Bio-Age Algorithm:** A custom logic layer was added to correct the "Age 5 Error" common in Caffe models.

- **Rule 1 (Imperfection Ban):** If Wrinkles or Puffy Eyes are detected with >40% confidence, "Child" age buckets (0-12) are mathematically suppressed.
- **Rule 2 (Selfie Detection):** If the face occupies >30% of the image width, the minimum age floor is raised to 18.
- **Rule 3 (Severity Penalty):** Adds +2 to +7 years to the predicted age based on the confidence of the wrinkle detection.

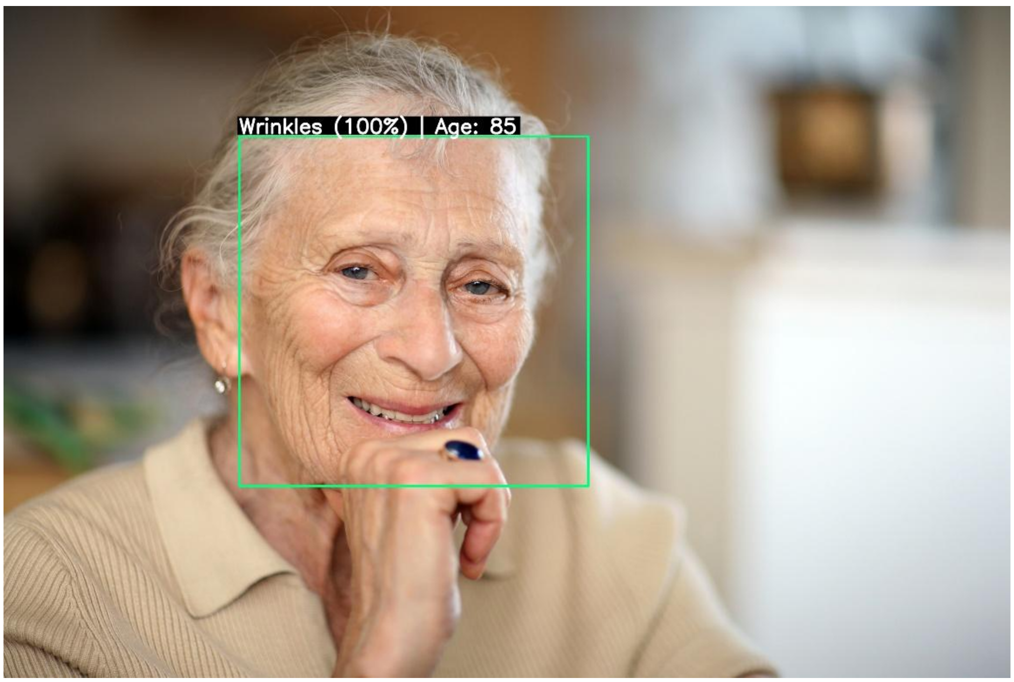



Fig: Accurate Age & Wrinkle Detection using Smart Bio-Age Logic.

## 4. 3D Visualization & Reporting:

Instead of flat bar charts, we integrated **Plotly 3D Donut Charts** to visualize the probability distribution of skin conditions. The dominant condition is "exploded" (pulled out) for emphasis.

**Sample Code Snippet (3D Chart):**

```
fig = go.Figure(data=[go.Pie(
   labels=labels, values=values, pull=pull_values, hole=0.4,
   marker=dict(colors=NEON_COLORS)
)])
```

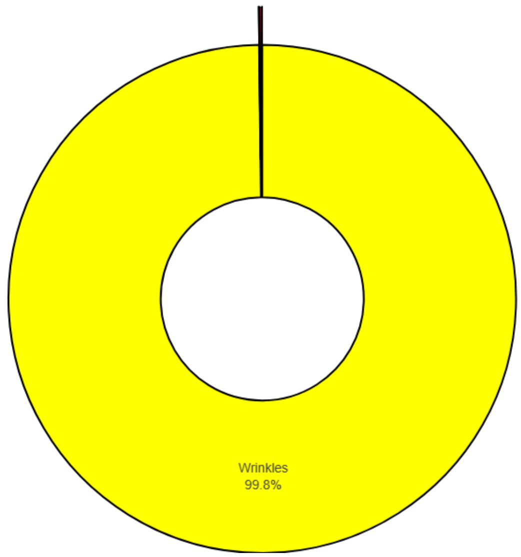



Fig: 3D Donut Chart visualizing probability of Puffy Eyes vs Clear Skin.

## 5. Final Output & Export:

The system generates a fully annotated image with a drop-shadow text overlay for readability. Users can download individual processed images or a batch CSV report.

| Filename | Face ID | Condition | Age | Confidence | Clear Skin % | Dark Spots % | Puffy Eyes % | Wrinkles % |
|---|---|---|---|---|---|---|---|---|
| shutterstock_1072 | 1 | Wrinkles | 85 | 99.80% | 0.00% | 0.20% | 0.00% | 99.80% |

Fig: Batch Analysis Report and CSV Export Feature.

## Conclusion:

The prototype successfully integrates the Frontend and Backend. The **Context Padding** and **Smart Bio-Age** algorithms significantly improved age prediction accuracy compared to the raw Caffe model outputs. The system is now ready for final deployment.