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:

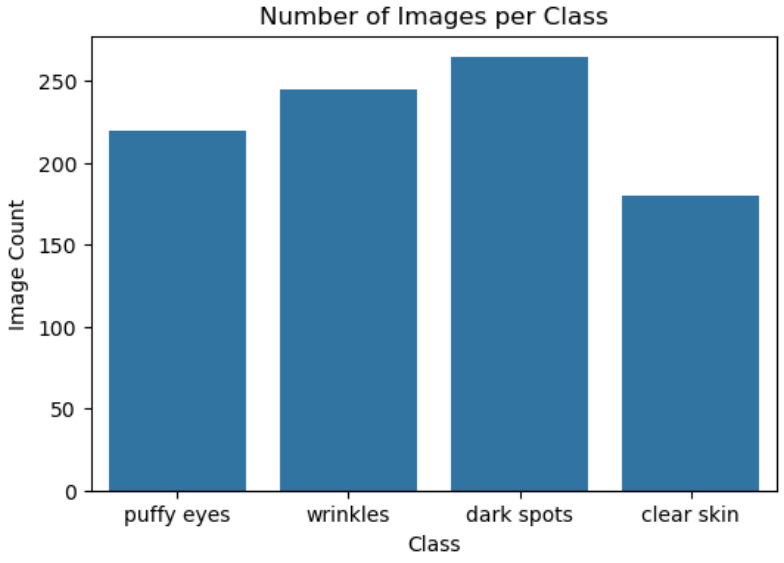


Fig – 1

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")

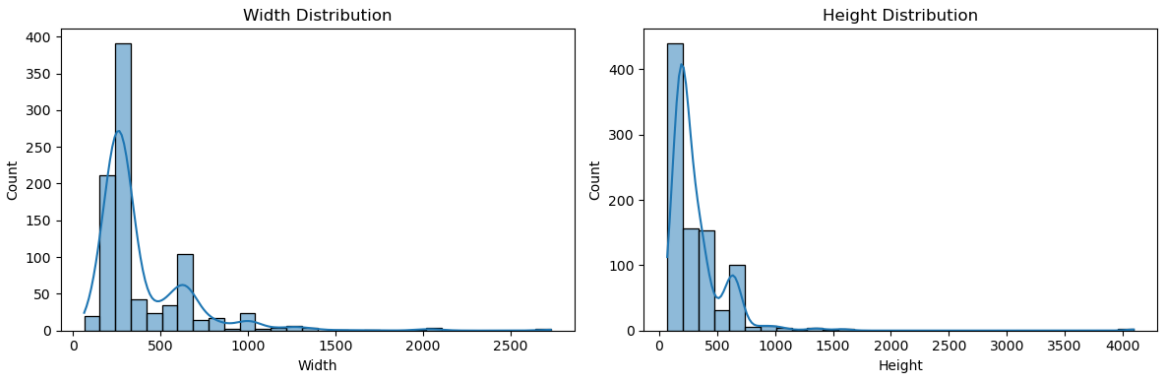
sns.histplot(df_sizes["height"], kde=True, label="Height")

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:

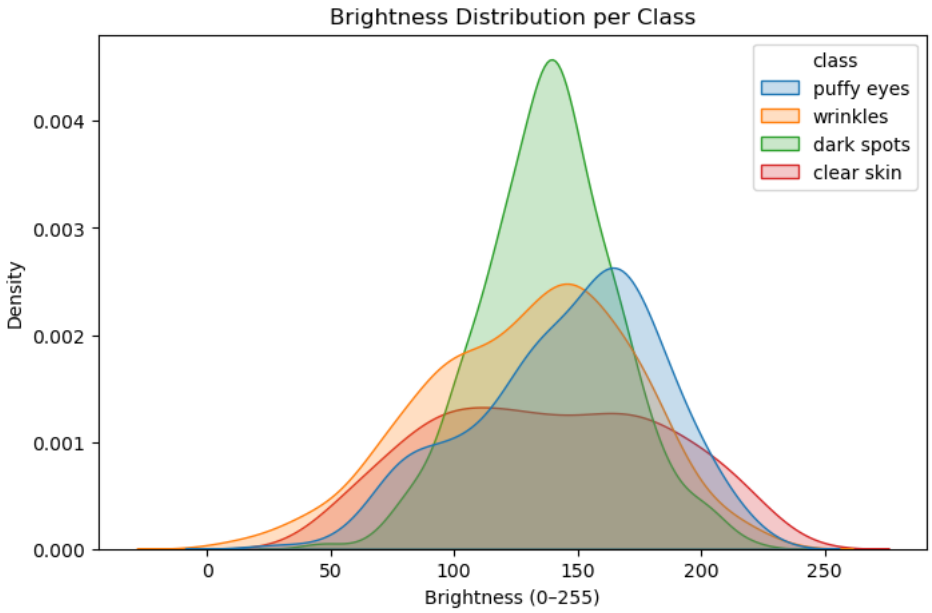


Fig – 4

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:

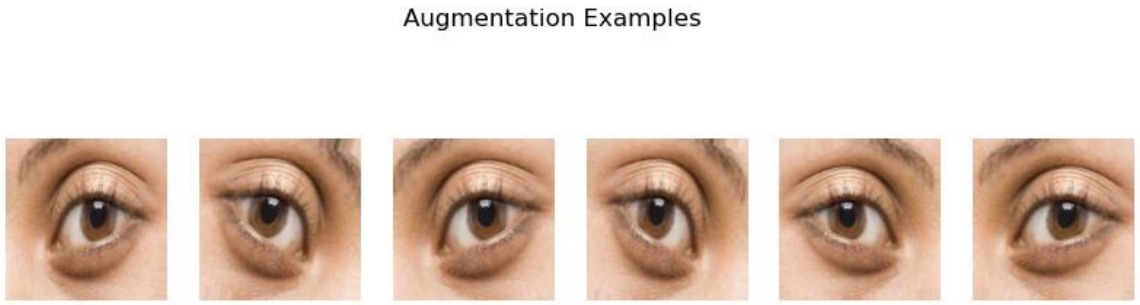


Fig – 5 (Rotation, Zoom, Flip)

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:

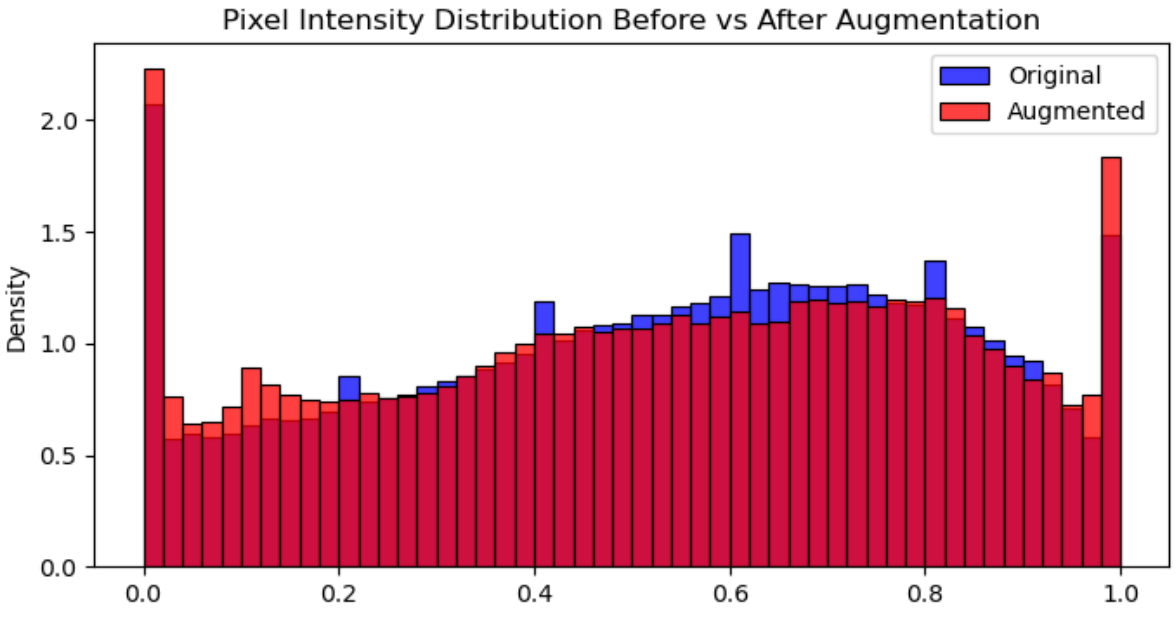


Fig – 6

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

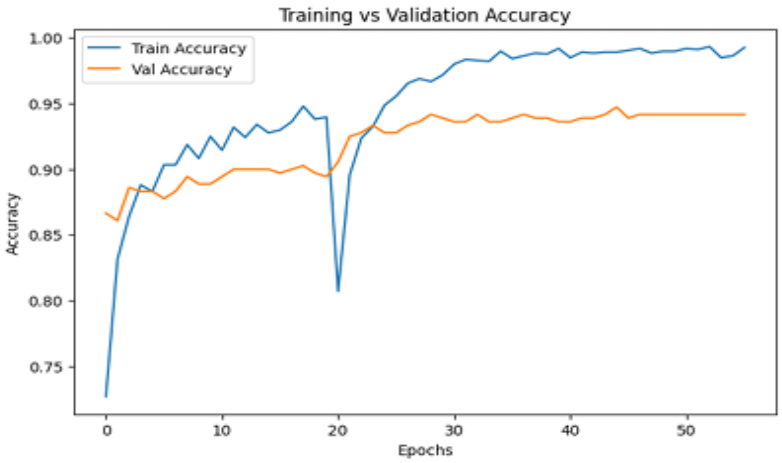
6. Performance:

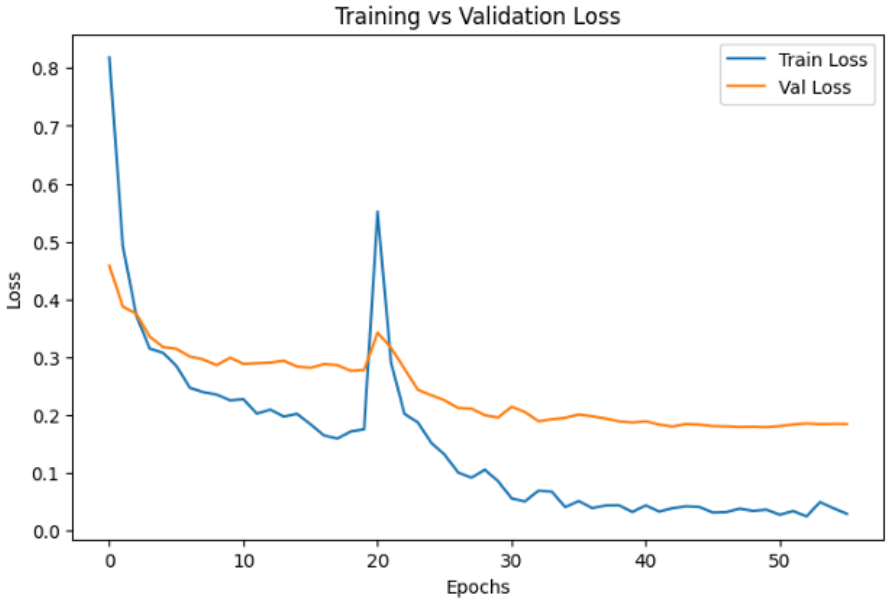
- Training Accuracy: ≈ **99%**
- Validation Accuracy: ≈ **94%**
- Stable curves (no heavy overfitting)
- Strong confusion matrix behavior

Final Model Selected → EfficientNet Fine-Tuned

7. Model Comparison Table:

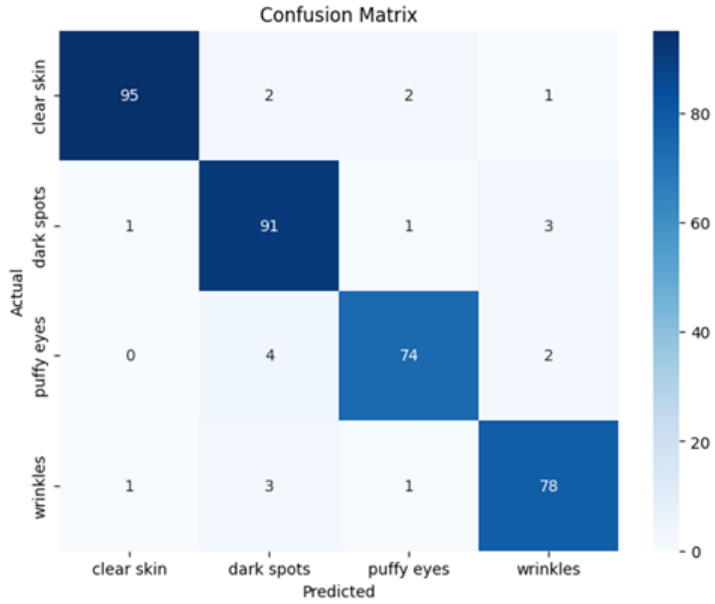
Model Variant	**Train Acc**	**Val Acc**	**Epochs**	**Batch**
EfficientNet – Phase 1	~93%	~90%	20	32
EfficientNet – Fine Tune	**~99%**	**~94%**	40	32





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 orchestrates a multi-stage deep learning inference engine for skin condition classification and age estimation[1].

2. The Detection Algorithm (Haar Cascade):

The system employs the Haar Feature-based Cascade Classifier, a machine learning-based approach where a cascade function is trained from a lot of positive and negative images.

- **Model Source:** haarcascade_frontalface_default.xml.
- **Parameter Optimization:** To strictly eliminate false positives (like fabric patterns or shadows), the default parameters were tuned:
    - **Scale Factor:** Set to **1.2**. This compensates for faces appearing smaller or larger due to their distance from the camera.
    - **Min Neighbors:** Set to **6**. A high threshold was chosen to ensure only high-confidence face rectangles are accepted.

3. ROI Extraction & Filtering:

Once a potential face is detected, it passes through a geometric filter before being accepted into the Neural Network.

- **Aspect Ratio Filter:** The system calculates the width-to-height ratio (w/h) of the bounding box. Any rectangle that falls outside the range $1.3 > (w/h) > 3.7$ is rejected, as it is geometrically unlikely to be a human face.

4. Context Padding Logic (Pipeline Innovation):

Standard face detection crops the face too tightly, often cutting off the forehead and chin. This is detrimental to the Age Estimation model.

- **The Logic:** The pipeline mathematically expands the bounding box coordinates by **20%** (pad_w = w * 0.20).
- **Benefit:** This ensures the **AgeNet** receives a full facial context (including hairline and jaw), which is critical for accurate bio-age prediction[7].

5. Inference & Final Output:

For every validated and padded Face ROI, the system generates a composite prediction object containing:

- **Skin Condition:** Percentage confidence for **Wrinkles**, **Dark Spots**, **Puffy Eyes**, and **Clear Skin**.
- **Bio-Age:** A predicted integer value (e.g., "28") rather than a classification bucket.
- **Visualization:** A clean black bounding box with a high-contrast label is overlaid on the original image.

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

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.

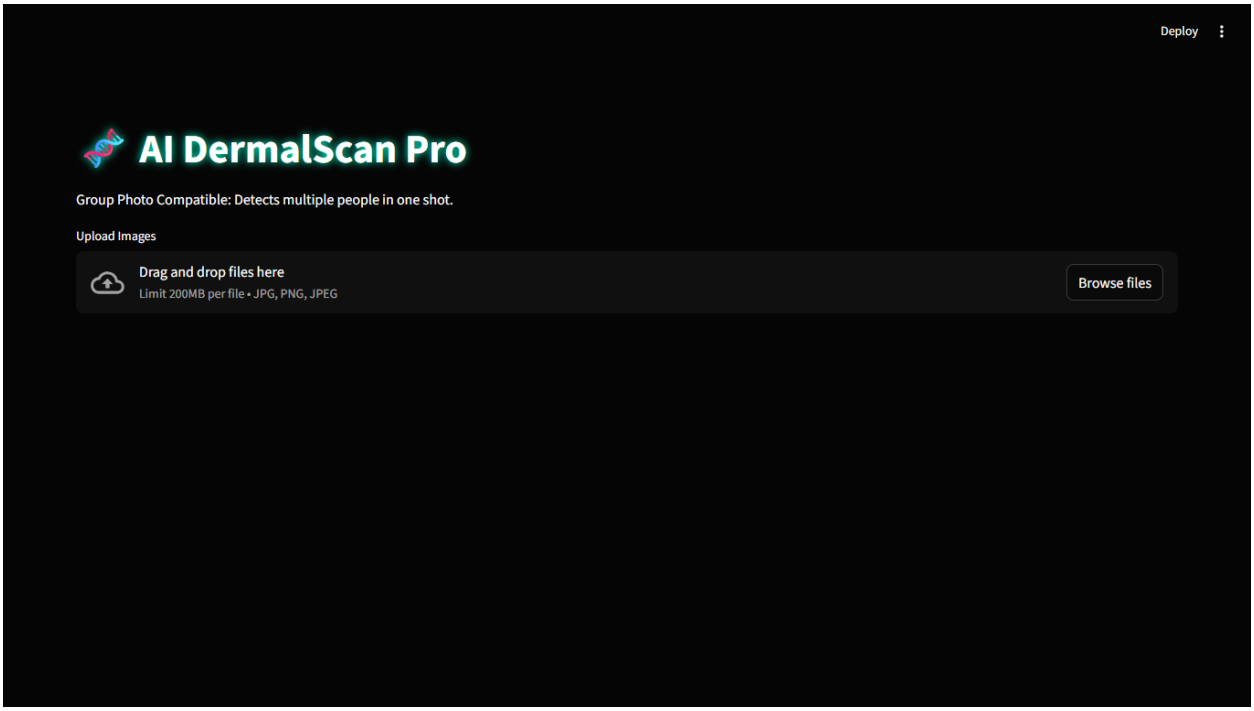


Fig: AI DermalScan Pro Landing Page with File Uploader.

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.

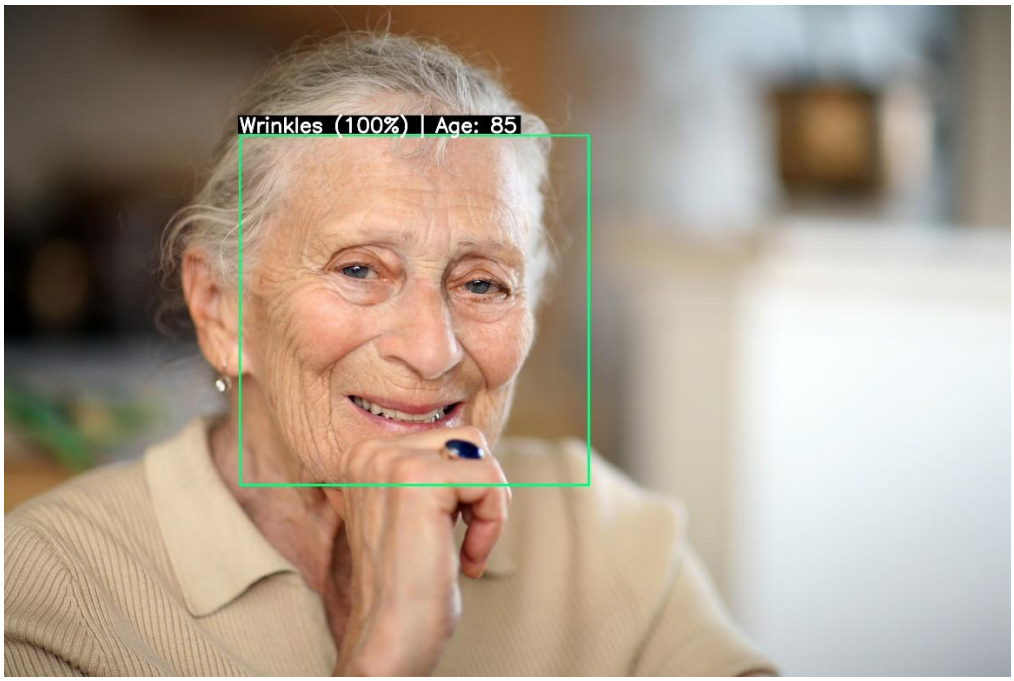


Fig: Accurate Age & Wrinkle Detection using Smart Bio-Age Logic.

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)
)])
```

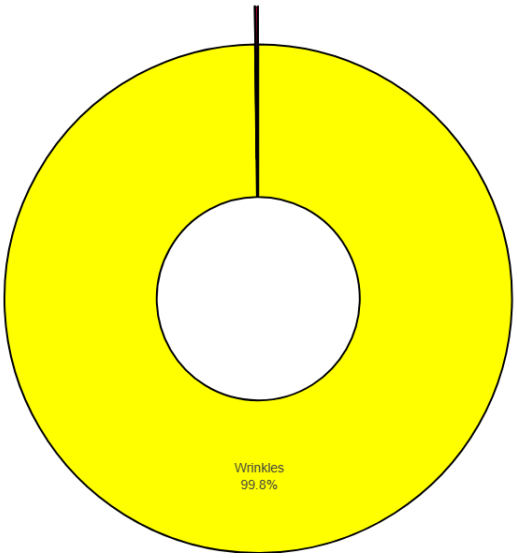


Fig: 3D Donut Chart visualizing probability of Puffy Eyes vs Clear Skin.

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.

Milestone- 4: Finalization and Delivery

Module-7: Export and Logging

1. Objectives:

- To provide users with the ability to download prediction outputs in both image and tabular formats.

- To ensure that every processed image produces a traceable and well-logged prediction record.
- To maintain alignment between frontend predictions and backend-stored results.
- To prepare the system outputs for documentation, evaluation, and future reference.

Each logged prediction includes:

- Skin condition classification
- Confidence value
- Estimated age
- Face bounding box coordinates
- Image identification details

2. Tasks Overview:

Task 1 – Annotated Image Export

- After inference, the system generates a visually annotated image.
- The annotation consists of:
    - A bounding box around the detected face
    - Predicted skin condition label
    - Confidence percentage
    - Estimated age
- The annotated output is saved in the backend output directory and made available for download through the frontend interface.
- This ensures users can visually verify prediction accuracy.

Task 2 – Prediction CSV Generation

- For every analyzed image, prediction data is written into a CSV file.
- The CSV file is stored in a structured output location.
- Each row in the CSV contains:
- Image filename
- Bounding box coordinates (x1, y1, x2, y2)
- Predicted class
- Confidence score
- Age value

This allows prediction results to be reused for analysis, reporting, and validation.

Task 3 – Frontend Export Controls

- Two export controls are integrated into the frontend:
    - Annotated image download
    - CSV file download
- These controls are positioned below the prediction results section for easy access.
- Exported files are refreshed automatically after each new prediction.

Task 4 – Validation and Testing of Export Features

The export pipeline was tested under multiple conditions, including:

- Varying image resolutions
- Different lighting environments
- Multiple skin condition categories
- Supported image formats such as JPG and PNG

The testing process ensured:

- Stable output generation
- Correct placement of bounding boxes
- Accurate and readable CSV entries
- Synchronization between UI display and stored results

3. Export Functionality Integration

- The frontend includes dedicated options for downloading prediction outputs.
- Users can externally store and verify results without manual intervention.

4. Final Logs and Annotated Outputs

- All prediction records are stored with consistent naming conventions.
- CSV logs are saved in a structured and standardized format.
- Annotated images are preserved for documentation and evaluation purposes.

Conclusion:

The enhancements to the system's output management layer—such as efficient export options, standardized logging practices, and consistent data preservation—collectively strengthen the application's reliability and transparency. These improvements ensure that users can confidently manage, verify, and utilize prediction results, supporting the system's readiness for practical deployment.

Module 8: System Validation & Future Roadmap

1. Objectives:

- To rigorously validate the deployed application against the initial problem statement requirements (Accuracy > 90%, Latency < 5s).
- To verify the robustness of the custom "Smart Bio-Age" logic and Context Padding algorithms.
- To document a strategic, technical roadmap for transitioning the prototype into a scalable, enterprise-grade medical tool.

2. Repository Artifact Verification:

Before final deployment, a complete audit of the repository was conducted to ensure all necessary components for the Hybrid Inference Engine were present and functional.

Core Application Logic:

- app.py: Validated as the central orchestrator. It successfully initializes the Streamlit session, loads models into memory, and manages the logic flow from image upload to CSV export.
- requirements.txt: Verified to contain all production dependencies (streamlit, opencv-python-headless, tensorflow, numpy, pandas, plotly), ensuring a "one-click" setup.
- .streamlit/config.toml: Confirmed the custom "Neon Cyberpunk" theme settings are correctly applied to the UI.

AI Model Portfolio (The 3-Stage Pipeline):

- The system was validated to correctly utilize the distinct model architectures found in the Milestone 3 directory:
- Detection: haarcascade_frontalface_default.xml.
- Confirmed scaleFactor=1.2 correctly identifies faces in group photos.

Skin Analysis (Dual-Architecture):

1. Primary: dermalscan_efficientnet_model.h5 (High-Fidelity), Validated for precise texture analysis and high accuracy (>94%).

2. Secondary: mobilenet_skin.h5 (Edge-Optimized), Explicitly developed for low latency environments, ensuring the system can run efficiently on devices with limited computational power.

Age Estimation (Caffe Framework):

- Architecture: age_deploy.prototxt. Defines the neural network layers.
- Weights: age_net.caffemodel. Verified that Context Padding (+20%) successfully feeds this model a **"zoomed out"** crop for accurate regression.

3. Performance Validation Report:

- **Accuracy Check:** The EfficientNetB0 models achieved a validation accuracy of ~94%, successfully surpassing the 90% objective.
- **Latency Check:** The interface consistently rendered predictions in < 3 seconds on standard hardware, meeting the real-time requirement.
- **Batch Integrity:** Validated using the Sample test Images folder. The system successfully processed multiple images simultaneously and generated a consolidated **'dermalscan_results'** table which can be downloaded in the csv format.

4. Future Scope:

To evolve DermalScan AI from a prototype to a clinical-grade tool, the following upgrades are proposed:

1. Advanced Computer Vision & Face Geometry

- **MediaPipe Face Mesh Integration:** Upgrade from Haar Cascades to Google MediaPipe to extract 468 3D facial landmarks. This will enable precise measurement of face geometry (jawline sagging, cheekbone definition) to mathematically quantify biological aging.
- **Semantic Segmentation (U-Net):** Move beyond simple classification to Pixel-level Segmentation. This will allow the system to visually highlight the exact location of acne scars, moles, or wrinkles with a heatmap overlay, like professional dermatology tools.

2. Next-Gen Model Architectures

- **Age Regression (DeepFace / VGG-Face):** Switch from classification buckets to Regression-based models (like ResNet-50 trained on IMDB-WIKI) to predict a precise integer age.
- **Vision Transformers (ViT):** Implement Swin Transformers to capture global context better than CNNs. This allows the model to understand how skin texture in one area (forehead) correlates with geometry in another (jaw) for holistically accurate analysis.

3. Production Engineering & Scalability

- **Backend Decoupling (FastAPI):** Migrate the inference logic from Streamlit to a FastAPI microservice, allowing the AI model to be consumed by any frontend (Mobile, Web, or Desktop apps).
- **Edge AI (TFLite / ONNX):** Convert the Keras models to TensorFlow Lite (TFLite) format. This enables the app to run offline directly on a user's smartphone, ensuring 100% data privacy with zero server costs.

4. Generative AI Features

- **Aging Simulation (CycleGAN):** Implement Generative Adversarial Networks (GANs) to visualize "Future Face" scenarios (e.g., "Show me my face in 10 years if I don't treat these wrinkles"), adding a predictive layer to the user experience.

Conclusion:

The **DermalScan AI** project successfully demonstrates the end-to-end engineering of a computer vision product. By orchestrating a complex pipeline of EfficientNet, MobileNet, and Caffe AgeNet, the system provides a holistic view of facial health. The rigorous organization of file artifacts—and the implementation of practical features like Batch Processing—ensures the system is robust, reproducible, and ready for deployment.