Skin Detection using image processing and classification

Course: Image Processing

Dr: Alaa Hamdy

Name: Roaa Khaled Salah

ID: 2022/05885

Contents

Introduction

Problem Statement

Skin detection is a crucial task in various applications, including medical diagnosis, augmented reality, and image editing. The objective of this project is to develop a system that accurately classifies pixels of an image as skin or non-skin based on various image processing techniques.

Project Objectives

- To implement a comprehensive pipeline for skin detection.
- To evaluate the performance of different algorithms in classifying skin areas.
- To visualize the results of each stage to better understand the processing.

Folder Structure

Overview of Files

The project is organized into the following files:

- **image_acquisition.py:** Handles loading and resizing images from specified directories.
- **preprocessing.py:** Contains functions for image enhancement and restoration.
- **segmentation.py:** Implements algorithms for segmenting skin areas from images.
- **ml_models.py:** Includes machine learning models for feature extraction and classification.
- **main.py:** The main script that has the entire pipeline and executes all steps.

Methodology

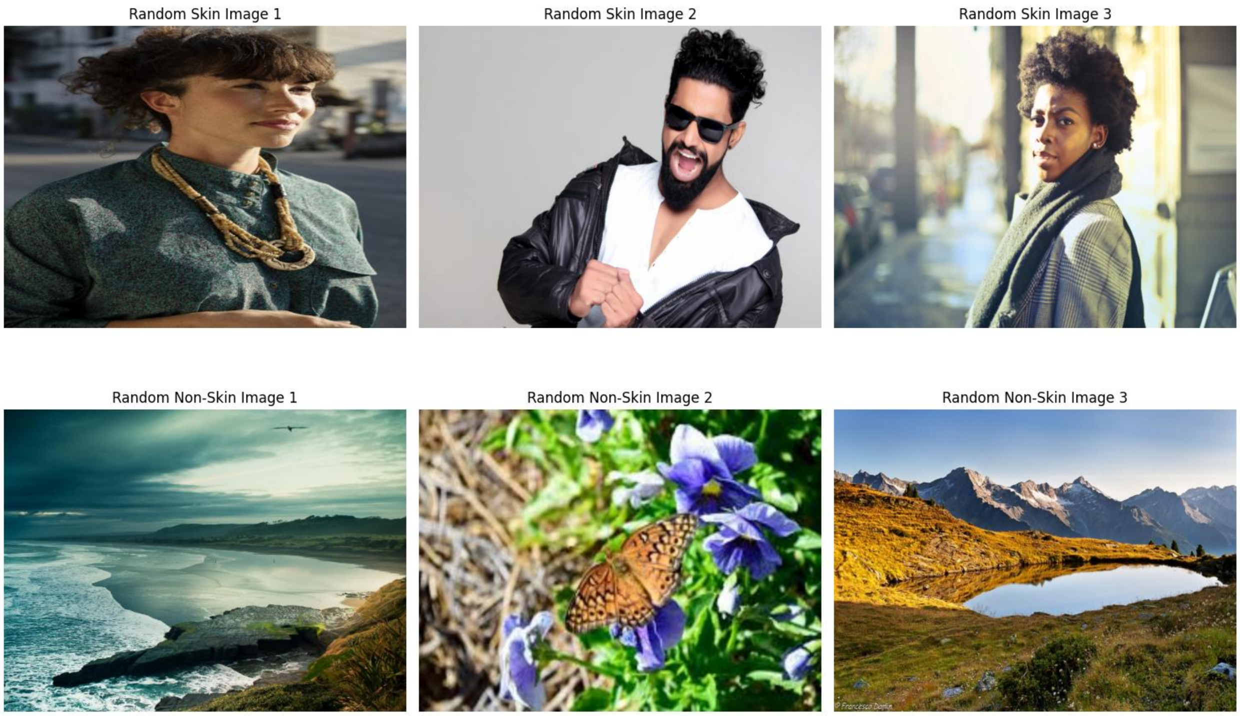
Image Acquisition

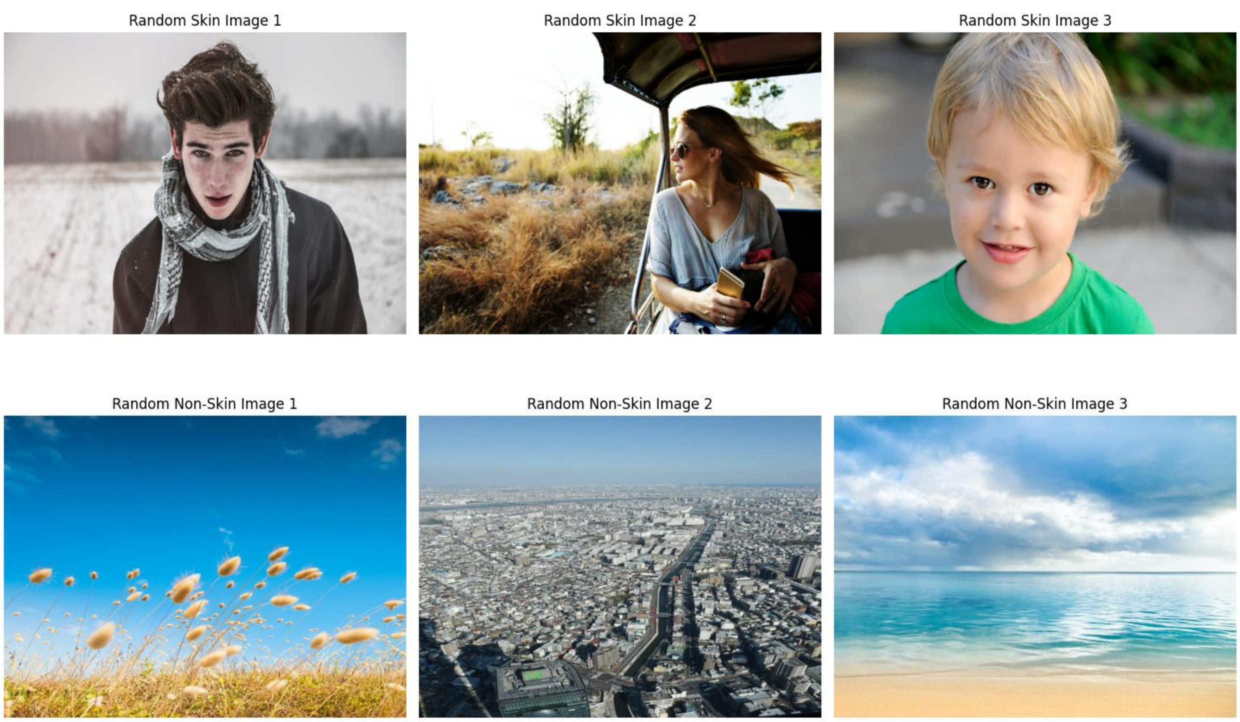
Images were acquired from two directories: one containing skin images and the other containing non-skin images. The ***load_images_from_folder*** function was implemented to load and resize images to a uniform dimension of 800x600 pixels. The function also ensured that grayscale images were converted to RGB format.

Code

```
image_acquisition.py > load_images_from_folder
Run Cell | Run Below | Debug Cell
# %% image_acquisition.py
import os
import cv2

Run Cell | Run Above | Debug Cell
#%%
def load_images_from_folder(folder_path, resize_dim=(800, 600)):
    images = []
    labels = []
    loaded_count = 0
    failed_count = 0

    for filename in os.listdir(folder_path):
        img_path = os.path.join(folder_path, filename)
        img = cv2.imread(img_path)
        if img is not None:
            if len(img.shape) == 2 or img.shape[2] != 3:
                img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
            resized_img = cv2.resize(img, resize_dim)
            images.append(resized_img)
            print(f"Loaded image: {filename}")
            loaded_count +=1
            labels.append(1 if "Skin" in folder_path else 0)
        else:
            print(f"Failed to load image: {filename}")
            failed_count +=1
    print(f"Total images loaded: {loaded_count}")
    print(f"Total images failed to load: {failed_count}")

    return images, labels
```

Visuals


Random Skin Image 1
Random Skin Image 2
Random Skin Image 3
Random Non-Skin Image 1
Random Non-Skin Image 2
Random Non-Skin Image 3


Random Skin Image 1
Random Skin Image 2
Random Skin Image 3
Random Non-Skin Image 1
Random Non-Skin Image 2
Random Non-Skin Image 3

Preprocessing

Image preprocessing was conducted to improve the quality of the images before segmentation. The following techniques were applied:

Gaussian Blur:

- This step reduces noise in the image.
- We use a Gaussian blur to smooth the image, which helps in making skin detection more accurate.

Gamma Correction: enhance the image's brightness and contrast. This helps highlight the skin areas better.

Code

```python
# %% preprocessing.py
import cv2
import numpy as np

#%%
# Gaussian blur
def restore_image(image):
    return cv2.GaussianBlur(image, (5, 5), 0)

#%%
# Gamma correction
def enhance_image(image, gamma=1.5):
    look_up_table = np.array([((i / 255.0) ** gamma) * 255 for i in np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, look_up_table)

```

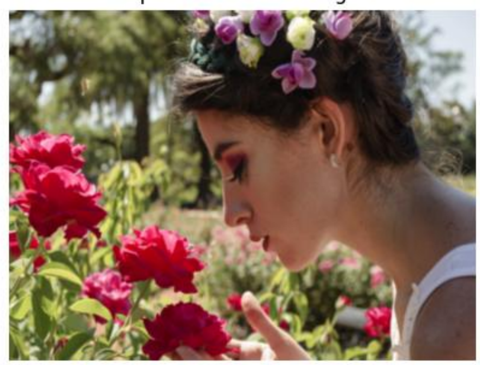
Visuals

Original Skin Image 1

Preprocessed Skin Image 1

Original Skin Image 2

Preprocessed Skin Image 2



Original Skin Image 3

Preprocessed Skin Image 3

Segmentation

The segmentation process isolated skin areas from the images. The *segment_image* function utilized color space transformation (from BGR to HSV) and morphological operations to create a binary mask that highlights skin regions

- **Color Space Conversion:** Converts the image from BGR to HSV color space, enhancing the ability to identify skin tones.
- **Skin Tone Range:** Applies a specified range for typical skin tones in HSV values, with defined minimum and maximum values for hue, saturation, and value.
- **Mask Production:** Produces a binary mask where skin pixels are white (255) and non-skin pixels are black (0) using cv2.inRange.
- **Morphological Operations:** After generating the mask, morphological operations are applied to remove noise and fill small holes in the detected skin areas.
- **Applying the Mask:** Finally, the mask is applied to the original image using *cv2.bitwise_and*, allowing skin regions to retain their original color while setting non-skin areas to black.

segmentation.py > ...

```
Run Cell | Run Below | Debug Cell
# %% segmentation.py
import cv2
import numpy as np

Run Cell | Run Above | Debug Cell
#%% RGB -> HSV + skin tone range + apply mask + morph
def segment_image(image):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_hsv = np.array([0, 20, 70], dtype=np.uint8)
    upper_hsv = np.array([25, 150, 255], dtype=np.uint8)
    mask_hsv = cv2.inRange(hsv_image, lower_hsv, upper_hsv)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    mask_hsv = cv2.morphologyEx(mask_hsv, cv2.MORPH_CLOSE, kernel)
    mask_hsv = cv2.morphologyEx(mask_hsv, cv2.MORPH_OPEN, kernel)
    return cv2.bitwise_and(image, image, mask=mask_hsv)

```

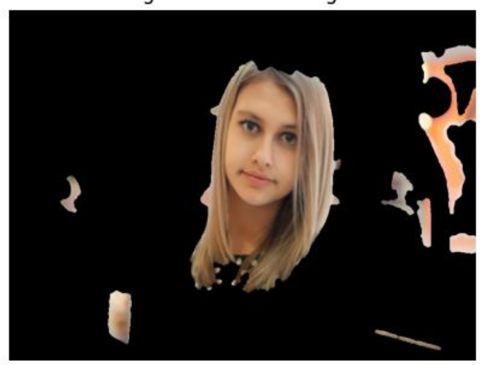
Visuals

Original Skin Image 1

Segmented Skin Image 1

Original Skin Image 2

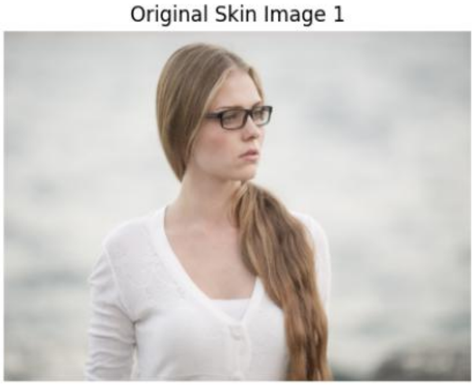
Segmented Skin Image 2



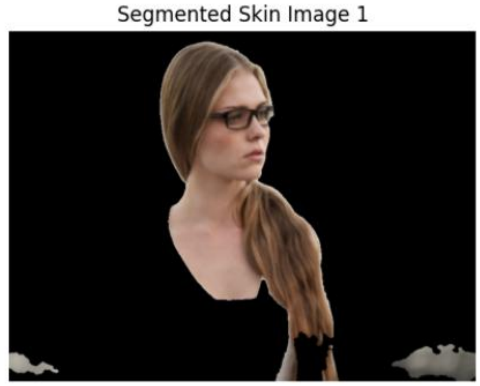
Original Skin Image 3

Segmented Skin Image 3

Original Skin Image 1



Segmented Skin Image 1



Feature Extraction

Features were extracted from the segmented images, which included:

- **Color Space Conversion:** Converting the image from RGB to HSV color space using the *cvtColor* function, which is essential for more effective color analysis.
- **Histogram Calculation:** Computing the histogram for each channel of the HSV (Hue, Saturation, Value) to understand the distribution of colors within the image.
- **Mean Intensity:** Calculating the average brightness value of the image, which provides insight into overall lighting conditions.
- **Standard Deviation:** Representing the variation in brightness, indicating the contrast level of the image.
- **Canny Edge Detection:** Analyzing pixel intensities to identify edges. The algorithm detects rapid changes in intensity, marking edges as white (255) and non-edges as black (0) in the resulting binary image.

Code

Run Cell | Run Above | Debug Cell

```
#%% Feature Extraction Function
def extract_features(image):
    # Convert to HSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Calculate histograms
    hsv_hist = [cv2.normalize(cv2.calcHist([hsv_image], [i], None, [256], [0, 256]), None).flatten() for i in range(3)]

    # Calculate mean and standard deviation
    mean_intensity = np.mean(image)
    std_intensity = np.std(image)

    # Use Canny edge detection for edge count
    edges = cv2.Canny(image, 100, 200)
    edge_count = np.sum(edges)

    return np.concatenate(([mean_intensity, std_intensity, edge_count], *hsv_hist))
```

Model Training and Evaluation

In the ml_models.py file, the model training and evaluation step is crucial for assessing the performance of different machine learning algorithms—specifically, Logistic Regression (LR) and Random Forest (RF)—on the extracted features. This process involves splitting the dataset into training and testing sets, training the models, making predictions, and evaluating their accuracy.

- Data Splitting: The dataset is split into training (70%) and testing (30%) sets using *train_test_split* to ensure a separate evaluation dataset.
- Model Training: using Random Forest and Logistic Regression to train the data set and predicting for the unseen test images
- Evaluate function: displaying confusion matrix, ROC and classification report

Code

```
Run Cell | Run Above | Debug Cell
# %% Model Training and Evaluation
def train_and_evaluate(X, y):
    # Split into Train --> 70% Test --> 30%
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)

    # Random Forest Model
    rf_model = RandomForestClassifier(n_estimators=10, random_state=40)
    rf_model.fit(X_train, y_train)
    rf_predictions = rf_model.predict(X_test)
    rf_probabilities = rf_model.predict_proba(X_test)[:, 1]
    rf_accuracy = accuracy_score(y_test, rf_predictions)

    print("Random Forest Accuracy:", rf_accuracy)
    print("Random Forest Classification Report:\n", classification_report(y_test, rf_predictions))

    # Logistic Regression Model
    lr_model = LogisticRegression(max_iter=100)
    lr_model.fit(X_train, y_train)
    lr_predictions = lr_model.predict(X_test)
    lr_accuracy = accuracy_score(y_test, lr_predictions)

    print("Logistic Regression Accuracy:", lr_accuracy)
    print("Logistic Regression Classification Report:\n", classification_report(y_test, lr_predictions))

    return rf_model, rf_accuracy, lr_model, lr_accuracy, X_train, X_test, y_train, y_test, rf_predictions, lr_predictions

Run Cell | Run Above | Debug Cell
#%% Evaluation Function (Confusion matrix + ROC + classification report)
def evaluate_model(y_test, y_pred, y_prob):
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Non-Skin", "Skin"]).plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()
    auc = roc_auc_score(y_test, y_prob)
    print(f"AUC: {auc:.3f}")
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")
    plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
    plt.title("ROC Curve")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()
```

Testing and Results

Testing Procedures

The models were evaluated using a subset of images that were not part of the training set. Each test image underwent the following steps:

1. **Preprocessing**: The images were prepared for analysis by adjusting their format and size.
2. **Segmentation**: Relevant features were extracted to isolate skin and non-skin areas.
3. **Feature Extraction**: Key characteristics of the images were quantified to inform the classification process.
4. **Classification**: The extracted features were fed into the trained models to classify the images.

Results and Visualizations

1. Random Forest Classifier

```
Random Forest Accuracy: 0.9941089837997055
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       345
           1       1.00      0.99      0.99       334

    accuracy                           0.99       679
   macro avg       0.99      0.99      0.99       679
weighted avg       0.99      0.99      0.99       679
```

.

```
Random Forest Predictions vs Actual Labels:
Actual: 1, Predicted: 1, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 1, Predicted: 1, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 1, Predicted: 1, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 1, Predicted: 1, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 0, Predicted: 0, Status: Correct
Actual: 1, Predicted: 1, Status: Correct
```

2. Logistic Regression

```
Logistic Regression Accuracy: 0.9455081001472754
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       345
           1       0.94      0.96      0.95       334

    accuracy                           0.95       679
   macro avg       0.95      0.95      0.95       679
weighted avg       0.95      0.95      0.95       679
```

Predictions

RF Prediction: Skin | LR Prediction: Skin | Actual: Non-Skin

RF Prediction: Skin | LR Prediction: Skin | Actual: Non-Skin

RF Prediction: Skin | LR Prediction: Non-Skin | Actual: Skin

RF Prediction: Skin | LR Prediction: Skin | Actual: Skin