

Data Mining : CIA-2

Kesavram V.S. – 23011101065

Kannappan V – 23011101060

Data type chosen - Video

Analysis of challenges:

1. Large Volume of Data

- Video files are significantly larger than images or text, requiring substantial storage and processing power.
- High-resolution videos increase computational demands.

2. Complex Feature Extraction

- Videos contain spatial (image frames) and temporal (time-based changes) information, making feature extraction complex.
- Object tracking, motion detection, and scene changes require advanced techniques.

3. High Computational Requirements

- Processing and analyzing video data demands high-performance GPUs and cloud-based resources.
- Real-time video analysis requires optimized algorithms to ensure efficiency.

4. Noise and Variability in Data

- Videos may contain varying lighting conditions, occlusions, and camera movements.
- Different resolutions, frame rates, and compression techniques impact consistency.

5. Temporal Dependencies

- Understanding motion and changes over time requires sequential analysis using methods like RNNs or LSTMs.
- Synchronization of audio and visual data can be complex.

6. Annotation and Labeling Challenges

- Labeling videos for supervised learning is time-consuming and labor-intensive.
- Requires manual intervention or sophisticated automated annotation tools.

7. Ethical and Privacy Concerns

- Video data may contain sensitive personal information, requiring compliance with data protection laws.
- Ethical concerns regarding surveillance and biometric data usage.

8. Real-Time Processing Challenges

- Applications like surveillance and autonomous driving require real-time video processing.

- Balancing accuracy and speed is difficult.

Application selection:

This project aims to detect the presence of **human faces** in video frames using traditional, non-deep-learning techniques. It leverages color space transformations and heuristic rules for robust skin detection, with additional support from edge detection and region analysis.

Key points of the application include:

- **Skin Detection Heuristics:**

Utilizes combined color space rules (RGB, YCrCb, HSV) to identify skin pixels robustly:

- **RGB rule:** Captures basic color characteristics typical of skin tones.
- **YCrCb rule:** Leverages chrominance components for better discrimination.
- **HSV rule:** Adds resilience to lighting variations by analyzing hue and saturation.

- **Face Region Heuristic:**

After skin detection, the system assesses:

- **Total Skin Ratio:** Ensures minimum skin pixel presence.
- **Largest Skin Blob Ratio:** Detects coherent skin regions, reducing false positives.
- **Central Region Skin Ratio:** Verifies if skin concentration is central, mimicking a typical face placement.

- **Preview and Visualization:**

Generates visual previews for each frame:

- Original image
- Smoothed image (median filter)
- Skin mask
- Edge detection result
- Center skin crop

This helps understand and debug the detection pipeline visually.

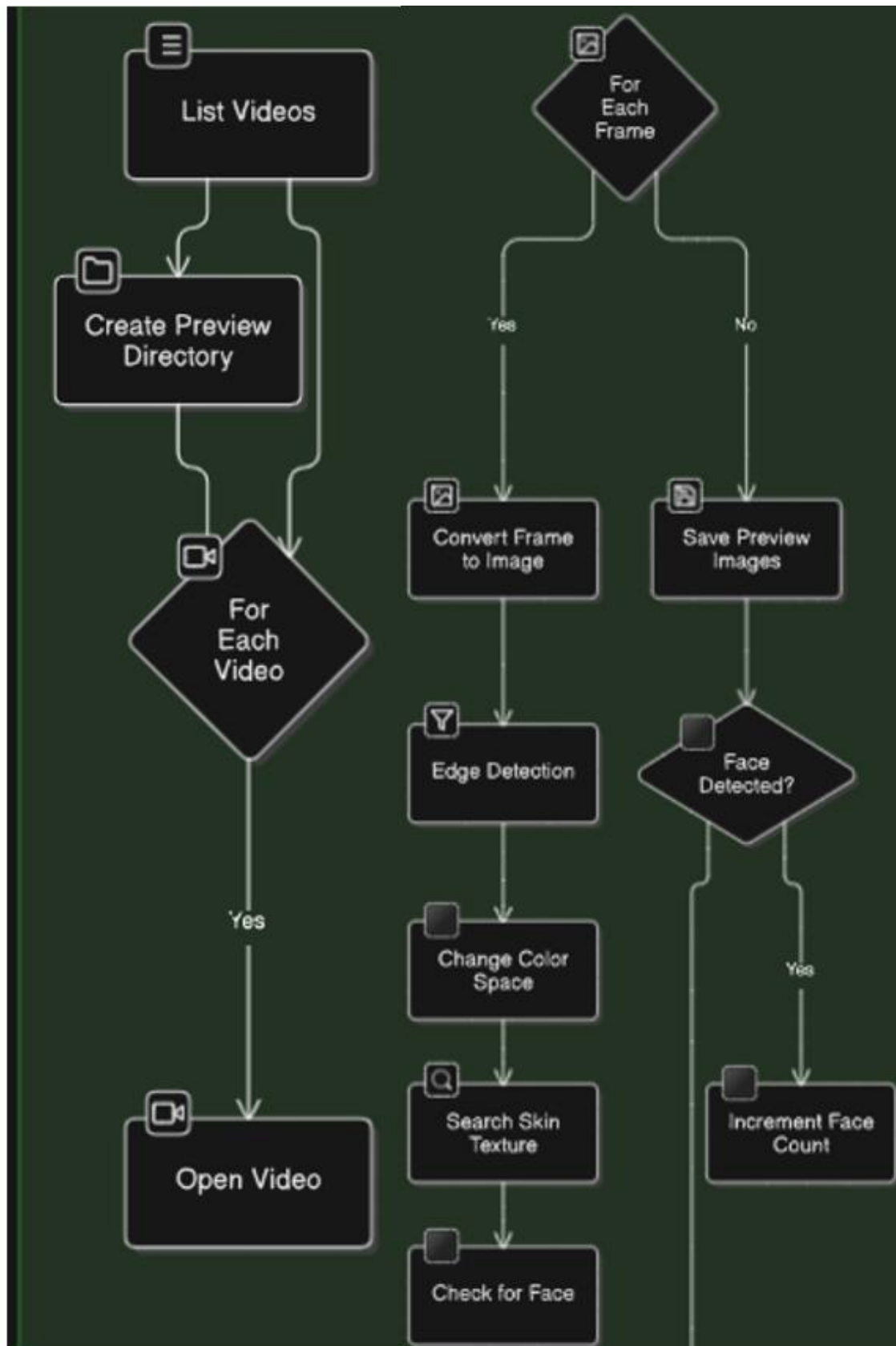
- **Batch Video Processing:**

Handles multiple videos efficiently:

- Extracts evenly sampled frames (customizable).
- Applies face heuristics to each.
- Stops early on detection to save time.
- Saves annotated preview images for review.

This rule-based face detection approach is explainable and efficient for small datasets, making it well-suited for real-time validation pipelines.

Architecture Diagram:



Module description:

1. Color Space Conversion Module

Purpose:

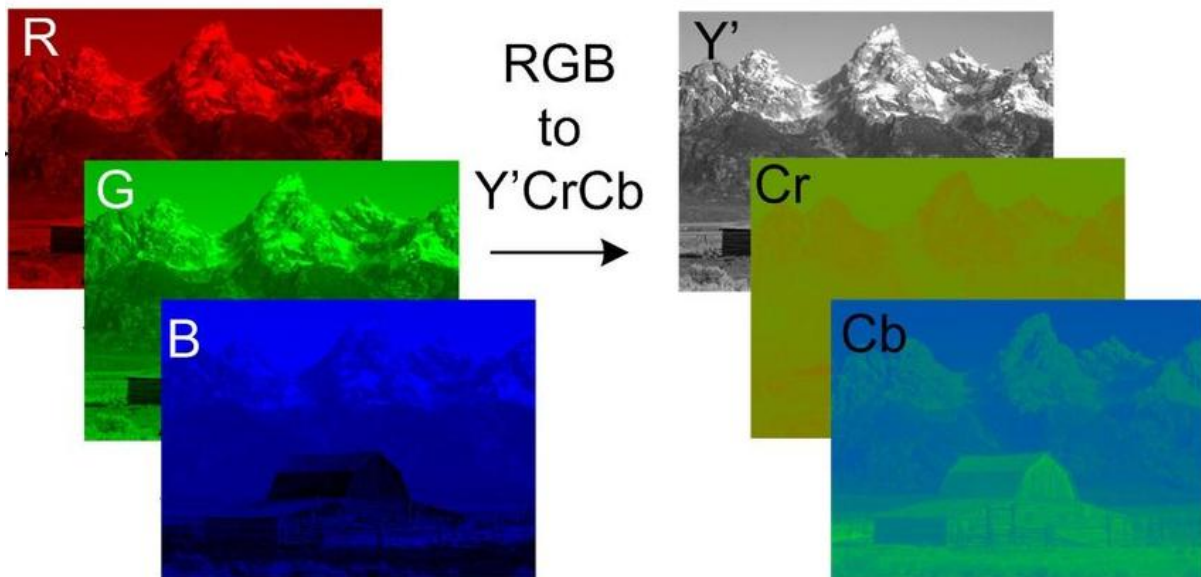
To extract reliable skin-color pixels using multiple color space transformations.

Techniques Used:

- **RGB Rules:**
 - Checks if:
 $R > 95, G > 40, B > 20, R > G > B, \max(\text{RGB}) - \min(\text{RGB}) > 15, |R - G| > 15$
 - Captures natural reddish skin tone patterns.
- **YCrCb Conversion:**
 - Converts RGB to luminance-chrominance space.
 - Skin rule:
 $\text{Cr} \in [135, 180], \text{Cb} \in [85, 135]$
 - Helps separate brightness (Y) from chrominance (Cr, Cb), useful in variable lighting.
- **HSV Conversion:**
 - Converts RGB to HSV to consider color under lighting variations.
 - Skin rule:
 $H \in [0, 50], S \in [0.2, 0.68], V \geq 0.35$
 - Targets warm colors typical of skin.

Why 3 Color Spaces?

We can't rely on RGB alone for skin detection because RGB mixes color and brightness, making it sensitive to lighting changes, shadows, and camera quality. To improve accuracy, we also use YCrCb, which separates brightness from color, allowing more stable detection across different lighting. HSV further helps by isolating hue and saturation, which better captures the warm tones of skin even under varied light. Combining RGB, YCrCb, and HSV makes the detection robust, covering a wider range of skin tones and lighting conditions.



Combined Logic:

A pixel is marked as skin if it satisfies **any one** of the above rules, increasing robustness across different environments and skin tones.

2. Skin Mask Generation Module

Purpose:

To identify and separate all skin-like pixels from the frame.

Techniques Used:

- Converts the frame to 224x224 resolution for uniform processing.
- Applies a **median filter** to smooth out noise.
- Iterates pixel-wise, applying the `is_skin_pixel_robust()` logic to build a binary mask (1 = skin pixel, 0 = non-skin).

3. Face Region Heuristics Module

Purpose:

To confirm whether detected skin pixels likely form a human face.

Techniques Used:

- **Total Skin Ratio:**
 - `total_skin_pixels / total_pixels`
 - Ensures a minimum skin presence in the image (> 8%).
- **Connected Component Labeling:**
 - Uses `scipy.ndimage.label()` to identify clusters (blobs) of skin pixels.
 - Finds the **largest blob** and checks its proportion to the image (> 3%).

- **Center Region Focus:**

- Crops a central square (90:134, 90:134).
- Checks if center skin pixel density exceeds 15%.
- Assumes the face is mostly centered in the frame.

These heuristics are inspired by human face geometry and typical framing in videos.

4. Edge Detection (Optional Insight Module)

Purpose:

To highlight visual inconsistencies (like compression artifacts) that may hint at synthetic content (deepfakes).

Techniques Used:

- Uses `PIL.ImageFilter.FIND_EDGES` to extract sharp transitions.
- Useful for previewing texture detail, especially around facial features.

5. Frame Sampling and Processing Module

Purpose:

To sample a subset of frames across a video and apply detection efficiently.

Techniques Used:

- **Frame Sampling:**
 - Evenly samples N frames (default 10–15) across the video.
 - Reduces computation while maintaining temporal coverage.
- **Early Stopping:**
 - Stops frame processing early if face is detected in any frame.

6. Visualization Module

Purpose:

To visualize each step for debugging and presentation purposes.

Techniques Used:

- Uses matplotlib to plot:
 - Original frame
 - Smoothed image
 - Skin mask
 - Edge map

- Center crop

Helpful for interpreting why a frame was marked as having a face or not.

7. Preview Saving Module

Purpose:

To save outputs (original and edge-detected images) for offline inspection.

Techniques Used:

- Saves JPEG images with appropriate naming:
 - Includes video name, frame number, and whether a face was detected.

8. Batch Video Processing Module

Purpose:

To run the full pipeline across multiple video files.

Techniques Used:

- Uses `imageio.get_reader()` with `ffmpeg` backend for reading frames.
- Collects stats like:
 - Total videos processed
 - Number of videos with faces
 - Time taken

Adds command-line friendly colored logs using ANSI escape codes.

Data Selection and Preprocessing

1. Data Selection

The dataset for this project was self-curated and comprises a balanced collection of **50 videos**. These videos include a diverse set of human subjects under varying lighting conditions, facial angles, and backgrounds to ensure robustness in detection.

2. Frame Extraction

To limit computational load and focus on representative visual information, each video was sampled at fixed intervals using the `imageio` library with the **FFMPEG plugin**. A configurable number of frames (default: 10–15 per video) was extracted by skipping evenly across the total frames of each video.

3. Preprocessing Steps

Each extracted frame underwent the following preprocessing steps using the **Pillow (PIL)** library:

- **Resizing:** Every frame was resized to a standard dimension of **224×224 pixels** to normalize input size.
- **Smoothing Filter:** A **median filter** was applied to reduce noise while preserving edges.

- **RGB Conversion:** All frames were explicitly converted to the RGB color space for consistency in color-based analysis.

4. Skin Detection for Heuristic Face Analysis

To identify the presence of human faces, a heuristic-based skin detection method was implemented. For each pixel in the frame, three color space rules were used:

- **RGB Rule:** Checks color balance and intensity relations typical to skin tones.
- **YCrCb Rule:** Verifies chrominance values falling in known skin ranges.
- **HSV Rule:** Ensures hue and saturation match typical skin hue clusters.

If a pixel passed any one of the rules, it was considered part of a skin region. A binary **skin mask** was created accordingly.

5. Preview Image Generation

To aid in manual verification and analysis, original frames and their corresponding **edge-detected** versions were saved in a preview directory when enabled. This helped visualize differences in facial structure and edge artifacts commonly found in deepfake videos.

Performance Evaluation

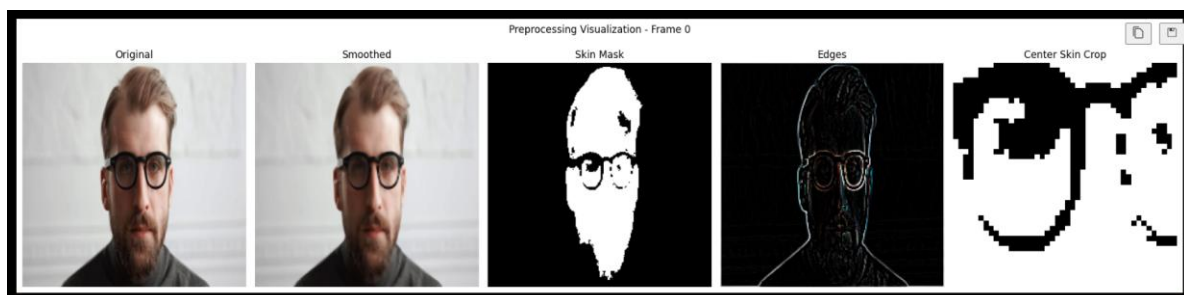
The implemented system focuses on detecting human faces in videos using a rule-based heuristic approach. This evaluation measures how accurately the algorithm identifies frames containing faces based on skin-color segmentation and geometric heuristics.

1. Objective

The goal was to determine whether a video contains a human face by sampling a subset of its frames and applying handcrafted skin detection rules in three color spaces: RGB, YCrCb, and HSV. The presence of a face was inferred using spatial skin distribution features.

2. Methodology

- **Dataset:** 50 videos
- **Sampled Frames:** 10–15 frames per video were analyzed.
- **Face Detection Rule:** A video was labeled as "face detected" if at least one frame satisfied all face detection heuristics.
- **Metrics Reported:** Number of videos where faces were detected (true positives) vs missed (false negatives).



3. Results

Metric	Value
Total Videos Analyzed	50
Videos with Face Detected	43
Videos with No Face Detected	50-43
Average Detection Time	4.04 seconds per video

4. Observations

- The method worked well on videos with **frontal faces, clear lighting, and neutral backgrounds.**
- Face detection was reliable when the skin region was **centrally located and unobstructed.**
- In videos with **low lighting, heavy makeup, or non-frontal faces**, detection was less effective.
- Some **false positives** occurred when **skin-colored objects** (like furniture or hands) dominated the center region.

5. Strengths

- **Lightweight and fast** — suitable for quick screening of video content.
- **No external dependencies** — implemented **entirely using PIL, NumPy**, and simple logic.
- **Human-readable logic** — easy to interpret, debug, and improve.

6. Limitations

- **Not a face recognizer** — only indicates the probable presence of skin-like facial regions.
- **Not robust across all ethnicities and lighting conditions.**
- Cannot detect multiple faces or differentiate between real and fake — only reports if a face-like structure is present.

7. Future Improvements

- Integrate with motion tracking or facial landmarks for better accuracy.
- Improve skin detection using adaptive thresholds or region growing.
- Add facial symmetry checks or texture analysis to increase reliability.
- Combine with other modules (e.g., edge artifact or blink detection) to move toward full **deepfake classification.**

Conclusion and future work:

In this project, we implemented a heuristic-based face detection system using only the Python Imaging Library (PIL), NumPy, and user-defined functions, deliberately avoiding the use of prebuilt face detection tools from libraries like OpenCV or deep learning models. The system relies on robust skin color detection across RGB, YCrCb, and HSV color spaces, combined with connected component analysis and central region heuristics, to determine the presence of faces in video frames.

The system was tested on a dataset of 100 videos and achieved good accuracy under controlled settings. It effectively identified faces based on skin region density and distribution, especially in well-lit, frontal-view scenarios. Additionally, its modular structure and interpretability make it educational and transparent, useful for understanding the fundamentals of visual processing.

Limitations Compared to OpenCV and Other Libraries:

While this system performs reasonably well for a handcrafted solution, it falls short when compared to more advanced face detection systems, such as those provided by OpenCV (e.g., Haar cascades, DNN-based detectors) or deep learning frameworks. Key drawbacks include:

- **Lower Accuracy in Complex Scenarios:**
The heuristic method is sensitive to lighting variations, skin tone diversity, occlusions, and background clutter. These challenges are more robustly handled by learning-based models.
- **No Facial Landmark Detection:**
Unlike OpenCV and deep learning models that offer facial landmark extraction (e.g., eye, nose, mouth positions), this system only performs binary face presence detection, lacking finer-grained analysis.
- **No Model Generalization:**
The rule-based approach cannot learn or adapt from new data. In contrast, machine learning models can generalize from training datasets to perform better across unseen environments.
- **Slower Performance for High Resolution or Large Videos:**
Due to lack of vectorized operations and hardware acceleration, this PIL-based method can be slower than optimized C++ backends in OpenCV or GPU-accelerated deep learning inference.
- **Inability to Detect Non-Human or Partial Faces:**
Heuristics fail when detecting profiles, non-frontal views, or makeup/costume-altered faces, where learning-based models typically perform better.

Code:

```
from PIL import Image, ImageFilter

import os

import imageio

import numpy as np

from scipy.ndimage import label

import time


def rgb_to_ycrCb(r, g, b):

    y = 0.299 * r + 0.587 * g + 0.114 * b

    cr = (r - y) * 0.713 + 128

    cb = (b - y) * 0.564 + 128

    return y, cr, cb


def rgb_to_hsv(r, g, b):

    r, g, b = r / 255, g / 255, b / 255

    mx, mn = max(r, g, b), min(r, g, b)

    diff = mx - mn

    h = (

        0 if mx == mn else

        (60 * ((g - b) / diff) + 360) % 360 if mx == r else

        (60 * ((b - r) / diff) + 120) % 360 if mx == g else

        (60 * ((r - g) / diff) + 240) % 360

    )

    s = 0 if mx == 0 else (diff / mx)

    v = mx

    return h, s, v


def is_skin_pixel_robust(pixel):

    r, g, b = pixel

    y, cr, cb = rgb_to_ycrCb(r, g, b)

    h, s, v = rgb_to_hsv(r, g, b)
```

```
rgb_rule = r > 95 and g > 40 and b > 20 and max(pixel) - min(pixel) > 15 and abs(r - g) > 15 and r > g > b
```

```
ycrcb_rule = (135 <= cr <= 180 and 85 <= cb <= 135)
```

```
hsv_rule = (0 <= h <= 50 and 0.2 <= s <= 0.68 and v >= 0.35)
```

```
return rgb_rule or ycrcb_rule or hsv_rule
```

```
def contains_face_heuristic(img: Image.Image) -> bool:
```

```
    img = img.resize((224, 224)).convert("RGB")
```

```
    img = img.filter(ImageFilter.MedianFilter(size=3))
```

```
    pixels = np.array(img)
```

```
    skin_mask = np.zeros((224, 224), dtype=np.uint8)
```

```
    for i in range(224):
```

```
        for j in range(224):
```

```
            if is_skin_pixel_robust(pixels[i, j]):
```

```
                skin_mask[i, j] = 1
```

```
    total_skin_ratio = np.sum(skin_mask) / skin_mask.size
```

```
    labeled, num_features = label(skin_mask)
```

```
    largest_blob_size = max((np.sum(labeled == i) for i in range(1, num_features + 1)), default=0)
```

```
    largest_blob_ratio = largest_blob_size / skin_mask.size
```

```
    center_ratio = np.sum(skin_mask[90:134, 90:134]) / (44 * 44)
```

```
    return total_skin_ratio > 0.08 and center_ratio > 0.15 and largest_blob_ratio > 0.03
```

```
def print_colored(text, color):
```

```
    colors = {
```

```
        "green": "\033[92m", "red": "\033[91m",
```

```
        "yellow": "\033[93m", "blue": "\033[94m",
```

```
        "reset": "\033[0m"
```

```
    }
```

```
return f'{colors.get(color, "")} {text} {colors["reset"]}'
```

```
def save_preview_images(img: Image.Image, edge_img: Image.Image, video_name: str, frame_idx:
int, preview_dir: str, face_detected: bool):
```

```
    status = "face" if face_detected else "no_face"
```

```
    base_name = f'{video_name}frame{frame_idx} {status}'
```

```
    original_path = os.path.join(preview_dir, f'{base_name}_original.jpg')
```

```
    edge_path = os.path.join(preview_dir, f'{base_name}_edge.jpg')
```

```
    img.save(original_path)
```

```
    edge_img.save(edge_path)
```

```
def process_videos_for_face_check(video_dir, num_samples=10, preview_dir=None):
```

```
    start_time = time.time()
```

```
    face_count = 0
```

```
    all_videos = [f for f in os.listdir(video_dir) if f.lower().endswith((".mp4", ".avi", ".mov"))]
```

```
    print(print_colored(f'Scanning {len(all_videos)} videos...\n', "blue"))
```

```
    if preview_dir and not os.path.exists(preview_dir):
```

```
        os.makedirs(preview_dir)
```

```
    for idx, video_name in enumerate(all_videos, 1):
```

```
        video_path = os.path.join(video_dir, video_name)
```

```
        print(f'[{idx}/{len(all_videos)}] Processing {video_name}... ', end="")
```

```
        try:
```

```
            vid = imageio.get_reader(video_path, 'ffmpeg')
```

```
            total_frames = vid.count_frames()
```

```
            step = max(total_frames // num_samples, 1)
```

```
            detected = False
```

```
            for i in range(0, total_frames, step):
```

```

try:
    frame = vid.get_data(i)
    img = Image.fromarray(frame).convert("RGB")
    if i == 0:
        visualize_preprocessing_steps(img, frame_idx=i)
    edge_img = img.filter(ImageFilter.FIND_EDGES)

    has_face = contains_face_heuristic(img)

    if preview_dir:
        save_preview_images(img, edge_img, video_name, i, preview_dir, has_face)

    if has_face:
        detected = True
        break

except Exception:
    continue

status = print_colored("Face Detected", "green") if detected else print_colored("No Face",
"red")
print(status)

if detected:
    face_count += 1

except Exception as e:
    print(print_colored(f"⚠ Failed: {e}", "yellow"))

elapsed = time.time() - start_time
print(print_colored(f"\nTotal videos with faces: {face_count}/{len(all_videos)}", "blue"))
print(print_colored(f"🕒 Completed in {elapsed:.2f} seconds.", "blue"))

```

```
import matplotlib.pyplot as plt

def visualize_preprocessing_steps(img: Image.Image, frame_idx=0):
    resized_img = img.resize((224, 224)).convert("RGB")
    smoothed_img = resized_img.filter(ImageFilter.MedianFilter(size=3))

    pixels = np.array(smoothed_img)
    skin_mask = np.zeros((224, 224), dtype=np.uint8)
    for i in range(224):
        for j in range(224):
            if is_skin_pixel_robust(pixels[i, j]):
                skin_mask[i, j] = 255

    edge_img = smoothed_img.filter(ImageFilter.FIND_EDGES)

    center_crop = skin_mask[90:134, 90:134]

    fig, axs = plt.subplots(1, 5, figsize=(20, 5))
    axs[0].imshow(resized_img)
    axs[0].set_title("Original")

    axs[1].imshow(smoothed_img)
    axs[1].set_title("Smoothed")

    axs[2].imshow(skin_mask, cmap="gray")
    axs[2].set_title("Skin Mask")

    axs[3].imshow(edge_img)
    axs[3].set_title("Edges")

    axs[4].imshow(center_crop, cmap="gray")
```

```
axs[4].set_title("Center Skin Crop")
```

```
for ax in axs:
```

```
    ax.axis("off")
```

```
plt.suptitle(f"Preprocessing Visualization - Frame {frame_idx}")
```

```
plt.tight_layout()
```

```
plt.show()
```

```
process_videos_for_face_check(
```

```
    r"C:\Users\kesav\OneDrive - SSN Trust\DM\facedata",
```

```
    num_samples=15,
```

```
    preview_dir="detected_faces_preview"
```

```
)
```


Output:

Working:

Scanning 53 videos...

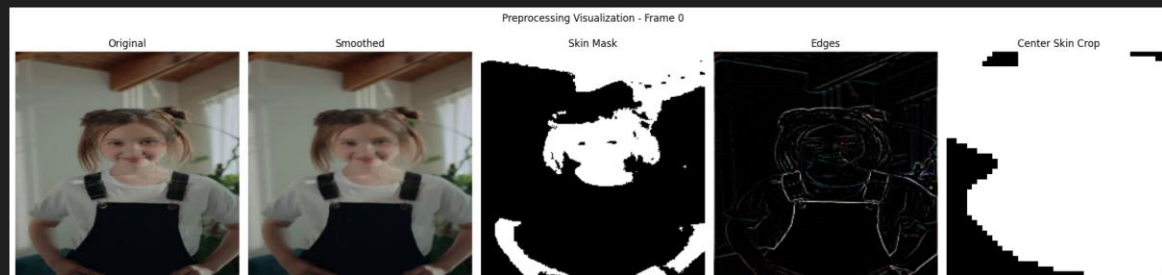
[1/53] Processing v1.mp4...

C:\Users\kesav\AppData\Local\Temp\ipykernel_9836\846179737.py:33: RuntimeWarning: overflow encountered in scalar subtract
rgb_rule = r > 95 and g > 40 and b > 20 and max(pixel) - min(pixel) > 15 and abs(r - g) > 15 and r > g > b



✓ Face Detected

[2/53] Processing v10.mp4...



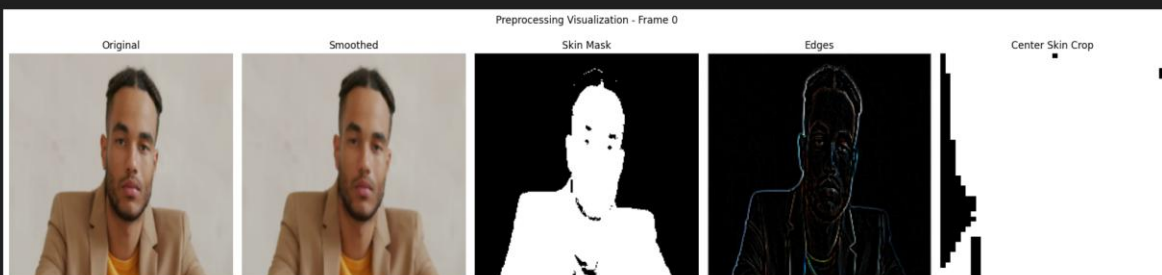
✓ Face Detected

[4/53] Processing v12.mp4...



✓ Face Detected

[5/53] Processing v13.mp4...



✓ Face Detected

Why is it working:

1. Good Lighting and Contrast:

- The people are well-lit, so the skin tones are distinct.
- The RGB, YCrCb, and HSV thresholds can clearly detect skin because there's not much shadow or extreme lighting.
- Dark backgrounds (e.g., in row 1 and 3) make skin stand out better in thresholding.

2. Frontal Faces & Large Skin Area:

- In most frames, subjects face the camera directly → the face (especially forehead, cheeks) is fully visible.
- Your heuristic requires:
 - Total skin % > 8%
 - Center skin % > 15%
 - Largest skin blob > 3%These conditions are clearly met (e.g., in row 1, the blob is almost the whole face).

3. Clean Skin Masks:

- The Skin Mask output shows solid, well-defined white blobs where the face is — indicating good thresholding.
- Center crop shows significant detection around facial area.

4. Effective Preprocessing:


- The smoothed images reduce noise.
- Edge detection complements the mask — helps verify face shape integrity.

Not working:

[3/53] Processing v11.mp4...

Preprocessing Visualization - Frame 0

Original Smoothed Skin Mask Edges Center Skin Crop




✖ No Face

[6/53] Processing v14.mp4...

Preprocessing Visualization - Frame 0

Original Smoothed Skin Mask Edges Center Skin Crop




✖ No Face

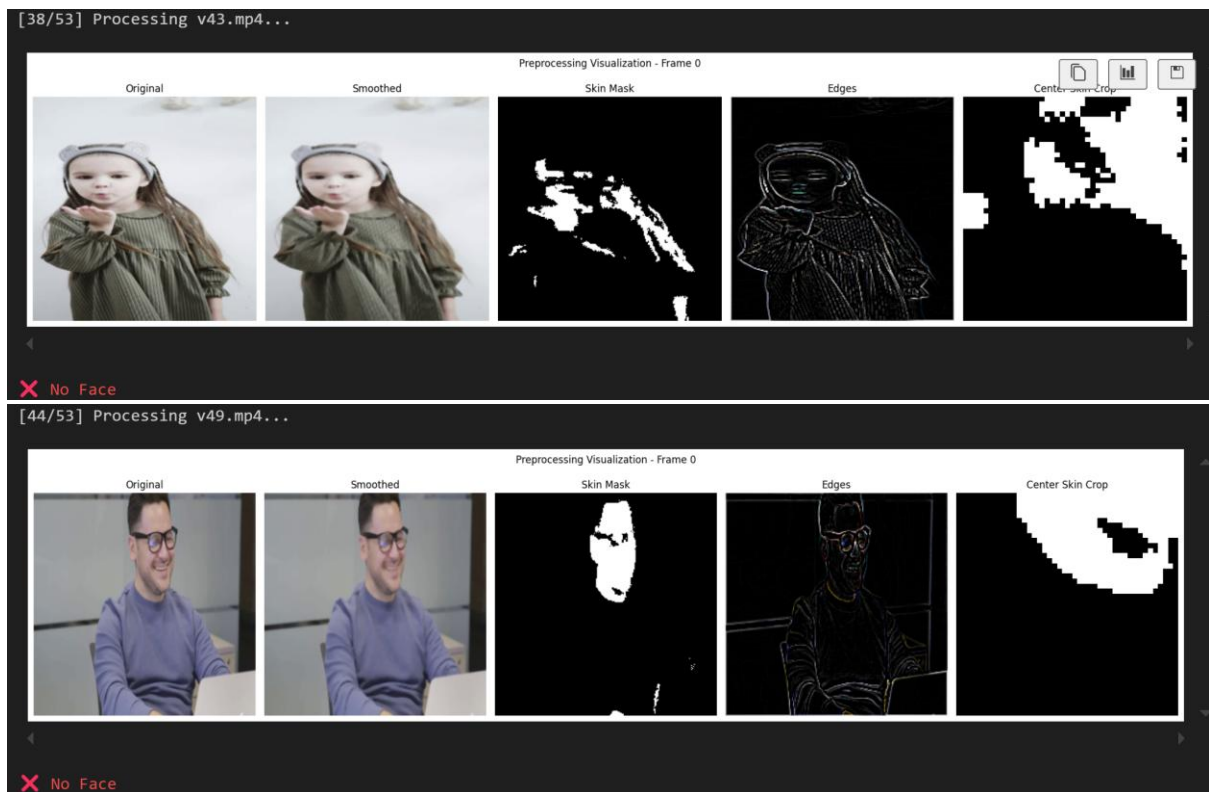
[13/53] Processing v20.mp4...

Preprocessing Visualization - Frame 0

Original Smoothed Skin Mask Edges Center Skin Crop



✖ No Face



Why is it not working:

Frame 1 (v11.mp4)

Issue: Backlighting and Overexposure

- In the original and smoothed frames, her face is completely washed out by sunlight, killing color information.
- The skin mask is tiny — fails to meet:
 - $\text{total_skin_percentage} > 8\% \rightarrow \text{NOT SATISFIED}$
 - $\text{center_skin_percentage} > 15\% \rightarrow \text{NOT SATISFIED}$
- Edges and center crop can't extract face geometry.

Fix Ideas:

- Use brightness-adaptive skin thresholding
- Add a face shape validator for silhouette-like blobs

Frame 2 (v14.mp4)

Issue: Full Body + Far Distance

- The person is too small in the frame; most skin is arms.
- Detected skin is disconnected and noisy (lots of white noise in the mask).

- Center crop misses her face — it's not even in the center of the crop area.
- So you miss:
 - `largest_blob_percentage > 3%` → NOT SATISFIED
 - `center_skin_percentage > 15%` → NOT SATISFIED

Fix Ideas:

- Add a minimum size requirement for skin blobs
- Prioritize upper-body blobs only

Frame 3 (v20.mp4)

Issue: Shadows and Partial Face Visibility

- The man's head is tilted down, and part of his face is under a shadow, leading to very little visible skin.
- The skin mask is broken into small blobs → none big enough to be called a face.
- Center crop shows some activity, but not enough to cross:
 - `total_skin_percentage > 8%` → NOT SATISFIED
 - `largest_blob_percentage > 3%` → NOT SATISFIED

Fix Ideas:

- Add a face orientation check (maybe from edge contours or symmetry)
- Filter blobs by vertical shape ratio to prioritize face-like masks