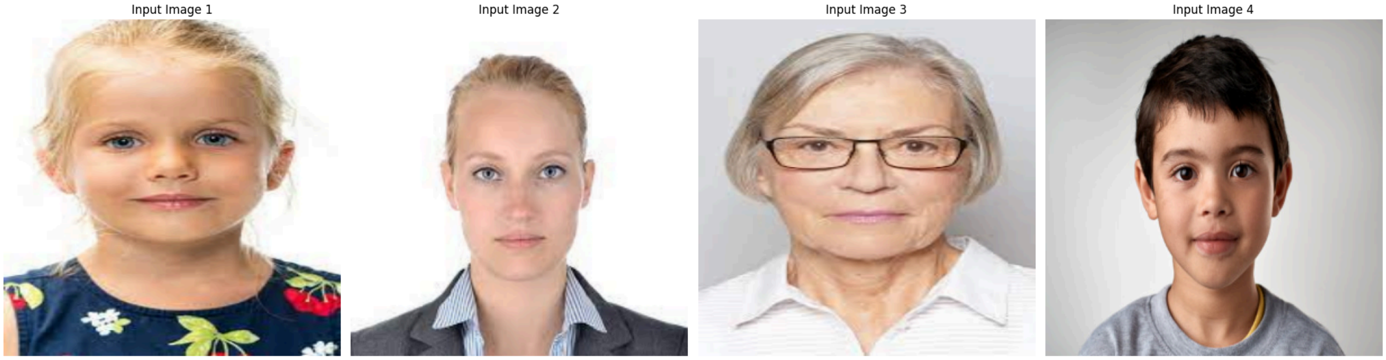
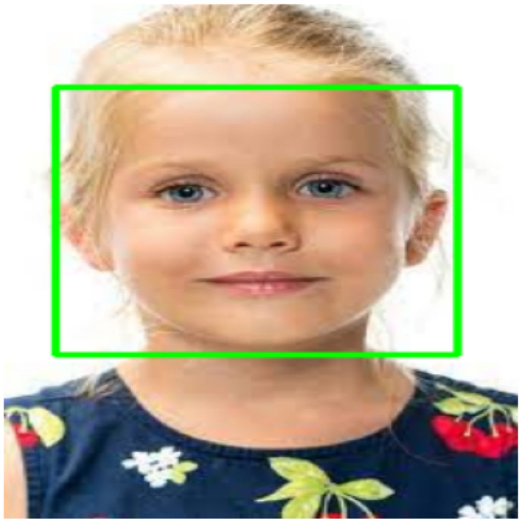
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to resize images to a fixed size
def resize_image(img, target_size=(300, 300)):
    return cv2.resize(img, target_size)

# Function to process images
def process_images(image_paths):
    plt.figure(figsize=(20, 10))

    for i, image_path in enumerate(image_paths, start=1):
        # Read the image using OpenCV
        img = cv2.imread(image_path)

        if img is None:
            print(f"Failed to load image {image_path}")
            continue

        # Resize image to a fixed size
        resized_img = resize_image(img)

        # Plot input image
        plt.subplot(1, len(image_paths), i)
        plt.imshow(cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB))
        plt.title(f'Input Image {i}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Example usage:
image_paths = ["/content/baby image.jpeg", "/content/girlimage.jpeg", "/content/grandimage.jpeg", "/content/littleboy.jpg"]
process_images(image_paths)

```

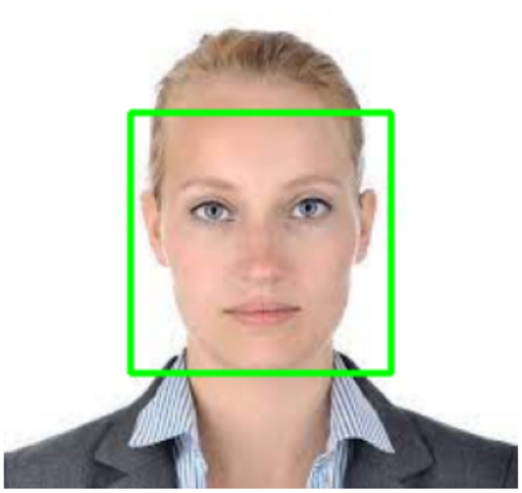


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to detect faces and draw bounding boxes
def detect_faces(image_paths, target_size=(300, 300)):
    for i, image_path in enumerate(image_paths, start=1):
        # Read the image using OpenCV
        img = cv2.imread(image_path)

        if img is None:
            print(f"Failed to load image {image_path}")
            continue

        # Convert the image to grayscale for face detection
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Load the pre-trained Haar Cascade classifier for face detection
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

        # Detect faces in the image
        faces = face_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        # Draw rectangles around the detected faces
        for (x, y, w, h) in faces:
            cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

        # Resize image to a fixed size
        resized_img = cv2.resize(img, target_size)

        # Plot input image with bounding boxes around detected faces
        plt.figure(figsize=(5, 5))
        plt.imshow(cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB))
        plt.title(f'Input Image {i} with face detection')
        plt.axis('off')
        plt.show()

# Example usage:
image_paths = ["/content/baby image.jpeg", "/content/girlimage.jpeg", "/content/grandimage.jpeg", "/content/littleboy.jpg"]
detect_faces(image_paths)

```

Input Image 1 with face detection



Input Image 2 with face detection



Input Image 3 with face detection

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to apply sharpening filter
def apply_sharpening(img):
    # Apply sharpening filter to the original image
    kernel_sharpening = np.array([[-1,-1,-1], [-1, 9,-1], [-1,-1,-1]])
    sharpened_img = cv2.filter2D(img, -1, kernel_sharpening)

    return sharpened_img

# Function to resize images to a fixed size
def resize_image(img, target_size=(300, 300)):
    return cv2.resize(img, target_size)

# Function to process images and apply sharpening
def process_images_sharpen(image_paths, target_size=(300, 300)):
    plt.figure(figsize=(20, 10))

    for i, image_path in enumerate(image_paths, start=1):
        # Read the image using OpenCV
        img = cv2.imread(image_path)

        if img is None:
            print(f"Failed to load image {image_path}")
            continue

        # Resize image to a fixed size
        resized_img = resize_image(img, target_size)

        # Apply sharpening filter
        sharpened_img = apply_sharpening(resized_img)

        # Plot sharpened image
        plt.subplot(1, len(image_paths), i)
        plt.imshow(cv2.cvtColor(sharpened_img, cv2.COLOR_BGR2RGB))
        plt.title(f'Apply Sharpen Filter for Input {i}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to apply Gaussian blur
def apply_blur(img):
    # Apply Gaussian blur to the original image
    blurred_img = cv2.GaussianBlur(img, (25, 25), 0)

    return blurred_img

# Function to resize images to a fixed size
def resize_image(img, target_size=(300, 300)):
    return cv2.resize(img, target_size)

# Function to process images and apply blur
def process_images_blur(image_paths, target_size=(300, 300)):
    plt.figure(figsize=(20, 10))

    for i, image_path in enumerate(image_paths, start=1):
        # Read the image using OpenCV
        img = cv2.imread(image_path)

        if img is None:
            print(f"Failed to load image {image_path}")
            continue

        # Resize image to a fixed size
        resized_img = resize_image(img, target_size)

        # Apply blur filter
```