In [3]:
```
py_file_location= r"C:\GWAR\Project\dataset\annotations"
sys.path.append(os.path.abspath(py_file_location))
```

In [2]:
```
import sys
import os
```

In [4]:
```
import os
import json
from itertools import islice
import random

from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
import cv2
from show_annotations import draw_bboxes, draw_landmarks
```

In [6]:
```
def collect_annotations(data_dir="C:/GWAR/dataset/classification_frames"):
    """Collect annotations in the individual files"""
    all_annotations = {}
    for seq in os.listdir(data_dir):
        if not os.path.isdir(os.path.join(data_dir, seq)):
            continue
        seq_annots = json.load(open(os.path.join(data_dir, seq, "annotatio
        for frame, frame_label in seq_annots.items():
            all_annotations[os.path.join(data_dir, seq, frame)] = frame_la
    len(all_annotations)

    json.dump(all_annotations, open(os.path.join(data_dir, "annotations_al
    return all_annotations

# all_annotations = collect_annotations()
all_annotations = json.load(open("C:/GWAR/dataset/classification_frames/an
```

In [7]:

```
TEST_SEQS = [
    "P1042762_720",
    "P1043086_720",
    "P1043081_720",
    "P1042780_720",
    "P1043106_720"
]

def construct_holdout_set(all_annotations, data_dir="C:/GWAR/dataset/class
    """Construct the holdout set."""
    holdout_annotations = {}
    train_annotations = {}

    for k, v in all_annotations.items():
        if k.split("/")[2] in TEST_SEQS:
            holdout_annotations[k] = v
        else:
            train_annotations[k] = v

    json.dump(holdout_annotations, open(os.path.join(data_dir, "annotation
    json.dump(all_annotations, open(os.path.join(data_dir, "annotations_tr

construct_holdout_set(all_annotations)
```

In [8]:

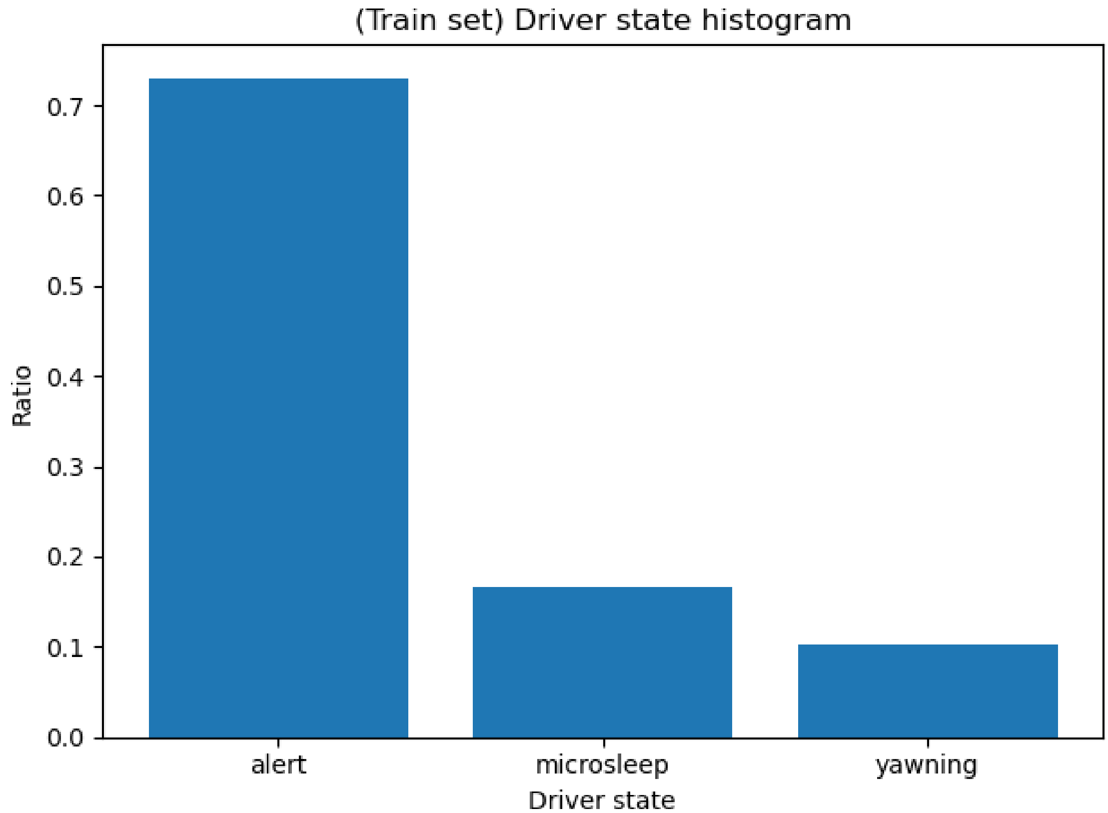
```
def val_test_split(val_ratio=0.5, data_dir="C:/GWAR/dataset/classification
    """Split the data into a validation and test set."""

    all_annotations = json.load(open(os.path.join(data_dir, file_name)))

    keys = list(all_annotations.keys())
    random.shuffle(keys)
    all_annotations_s = {}
    for key in keys:
        all_annotations_s[key] = all_annotations[key]
    n_val = int(val_ratio * len(all_annotations_s))

    annotations_val = dict(islice(all_annotations_s.items(), n_val))
    annotations_test = dict(list(all_annotations_s.items())[n_val: ])
    print(f"n_val: {len(annotations_val)}, n_test: {len(annotations_test)}

    json.dump(annotations_val, open(os.path.join(data_dir, "annotations_va
    json.dump(annotations_test, open(os.path.join(data_dir, "annotations_t

val_test_split()
```

```
n_val: 2502, n_test: 2502
```

In [10]:

```
def driver_state_distribution(annotations_file):
    """Compute distribution of driver states"""
    annotations = json.load(open(annotations_file))
    driver_states = {"alert": 0, "microsleep": 0, "yawning": 0}
    for frame, label in annotations.items():
        driver_states[label["driver_state"]] += 1
    driver_states_count = driver_states.copy()

    val_sum = sum(driver_states.values())
    for key, val in driver_states.items():
        driver_states[key] = val / val_sum
    print(driver_states)
    return driver_states_count, driver_states

driver_states_count, driver_states = driver_state_distribution("C:/GWAR/da
plt.bar(list(driver_states.keys()), list(driver_states.values()))
plt.xticks(list(driver_states.keys()))
plt.xlabel('Category')
plt.title("(Train set) Driver state histogram")
plt.xlabel("Driver state")
plt.ylabel("Ratio");
plt.tight_layout()
plt.savefig("C:/GWAR/dataset/image/train_set_distribution.pdf")
print(driver_states_count)
class_weights = list(1 / count for count in driver_states_count.values())
print(f"class_weights: {class_weights}")
```

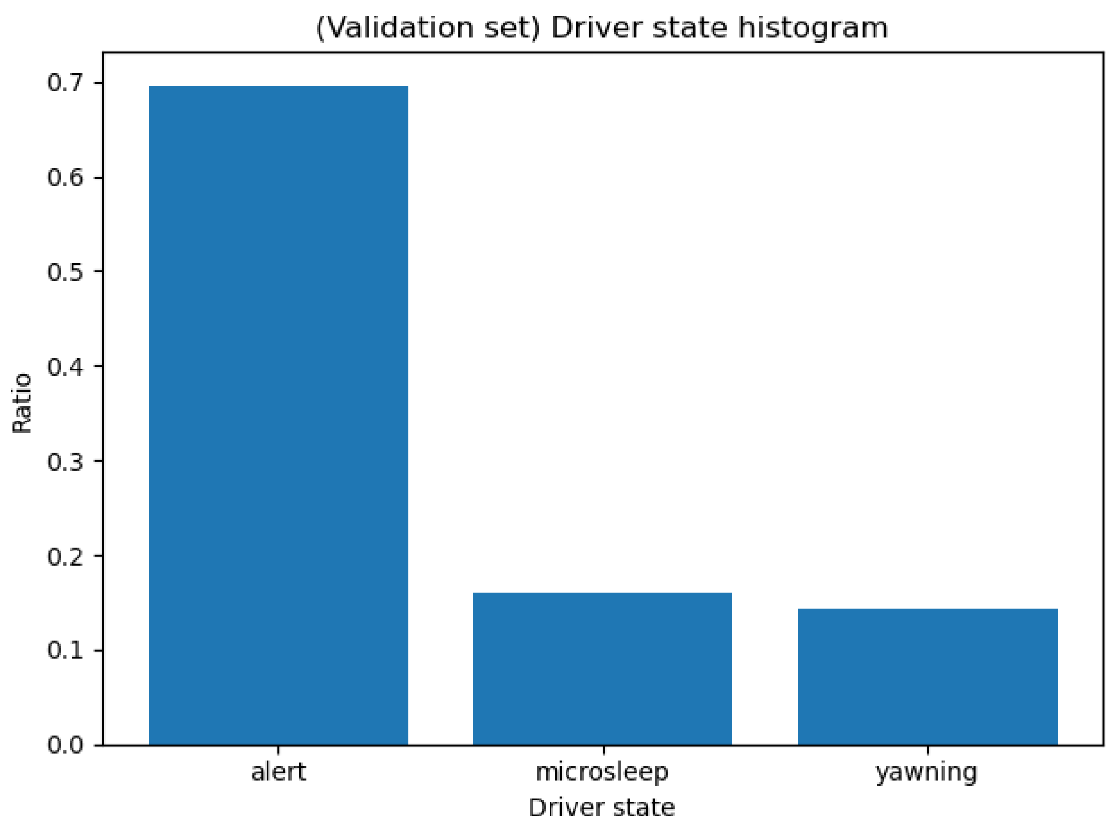
```
{'alert': 0.7304194558511935, 'microsleep': 0.16650728469370535, 'yawnin
g': 0.10307325945510117}
{'alert': 38954, 'microsleep': 8880, 'yawning': 5497}
class_weights: [2.567130461570057e-05, 0.00011261261261261261, 0.00018191
740949608878]
```


(Train set) Driver state histogram
Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
alert
microsleep
yawning
Driver state

In [11]:

```
driver_states_count, driver_states = driver_state_distribution("C:/GWAR/da
plt.bar(list(driver_states.keys()), list(driver_states.values()))
plt.xticks(list(driver_states.keys()))
plt.xlabel('Category')
plt.title("(Validation set) Driver state histogram")
plt.xlabel("Driver state")
plt.ylabel("Ratio");
plt.tight_layout()
plt.savefig("C:/GWAR/dataset/image/val_set_distribution.pdf")
print(driver_states_count)
class_weights = list(1 / count for count in driver_states_count.values())
print(f"class_weights: {class_weights}")
```

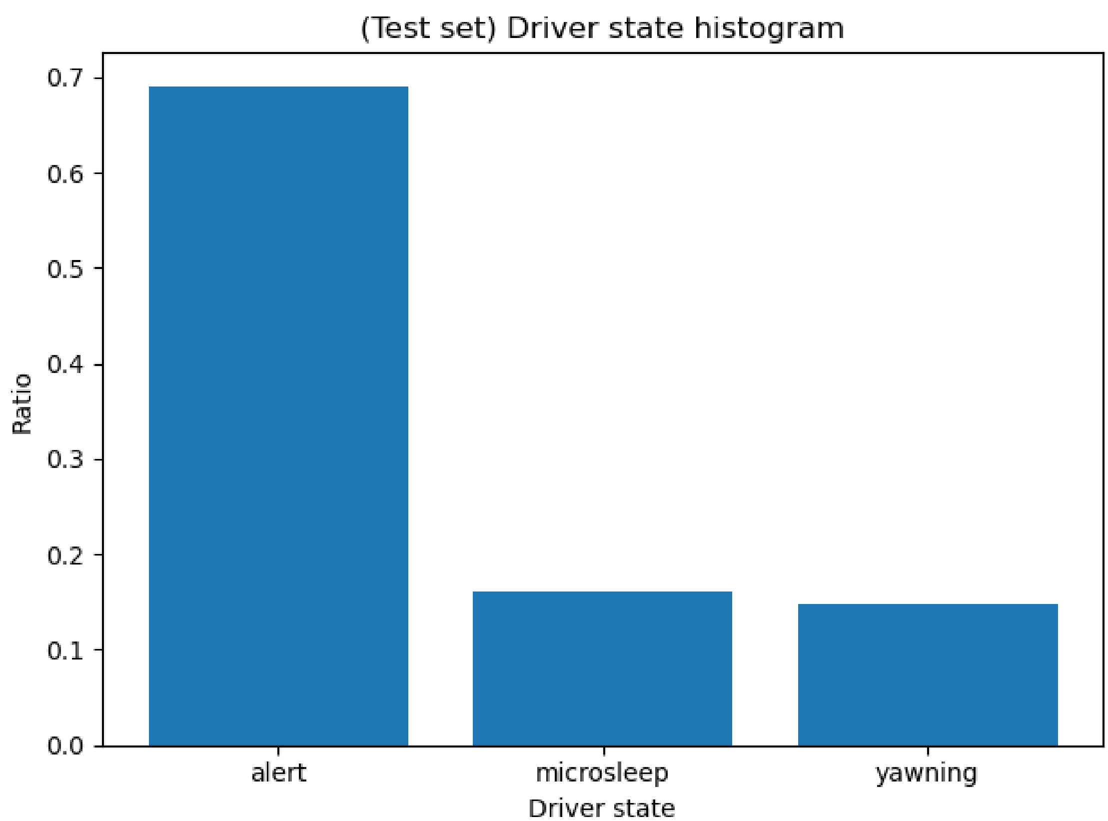
```
{'alert': 0.6962430055955235, 'microsleep': 0.1606714628297362, 'yawnin
g': 0.14308553157474022}
{'alert': 1742, 'microsleep': 402, 'yawning': 358}
class_weights: [0.000574052812858783, 0.0024875621890547263, 0.0027932960
89385475]
```



In [12]:

```
driver_states_count, driver_states = driver_state_distribution("C:/GWAR/da
plt.bar(list(driver_states.keys()), list(driver_states.values()))
plt.xticks(list(driver_states.keys()))
plt.xlabel('Category')
plt.title("(Test set) Driver state histogram")
plt.xlabel("Driver state")
plt.ylabel("Ratio");
plt.tight_layout()
plt.savefig("C:/GWAR/dataset/image/test_set_distribution.pdf")
print(driver_states_count)
class_weights = list(1 / count for count in driver_states_count.values())
print(f"class_weights: {class_weights}")
```

```
{'alert': 0.6910471622701838, 'microsleep': 0.16147082334132695, 'yawnin
g': 0.1474820143884892}
{'alert': 1729, 'microsleep': 404, 'yawning': 369}
class_weights: [0.0005783689994216310, 0.0024752475247524753, 0.0027100271
002710027]
```



In [59]:

```
import cv2
import numpy as np
import os
from tqdm import tqdm

# Function for smoothing images
def smooth_image(image_path, output_path):
    # Read the image
    frame_data = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded successfully
    if frame_data is None:
        print(f"Could not read the image: {image_path}")
        return

    # Define the smoothing kernel
    kernel = np.ones((3, 3), np.float32) / 9  # 3x3 averaging kernel

    # Perform data smoothing using the filter2D function
    smoothed_frame_data = cv2.filter2D(frame_data, -1, kernel)

    # Display the original and smoothed images using matplotlib
    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(frame_data, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title('Smoothed Image')
    plt.imshow(smoothed_frame_data, cmap='gray')
    plt.show()

# Specify the folder path containing images
folder_path = 'C:\GWAR\dataset\classification_frames\P1042751_720'  # Repl
folder_path1 = 'C:\GWAR\dataset\classification_frames_smooth\P1042751_720'

# Iterate over the files in the folder with tqdm
for filename in tqdm(os.listdir(folder_path)):
    if filename.endswith(('.jpg', '.png', '.jpeg')):
        image_path = os.path.join(folder_path, filename)
        output_path = os.path.join(folder_path1, f"smoothed_{filename}")
        smooth_image(image_path, output_path)
```

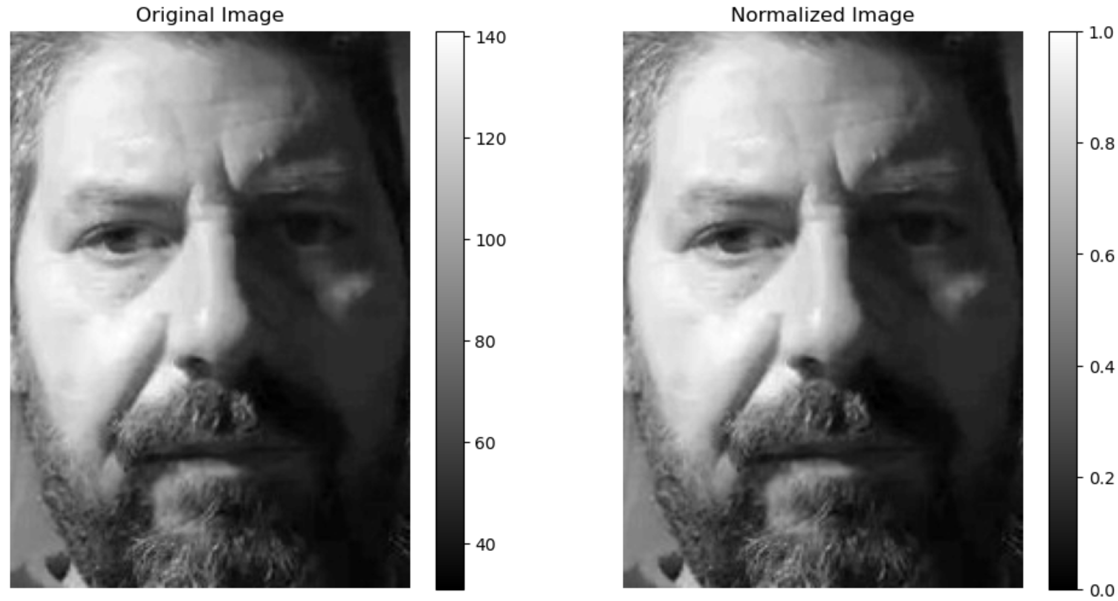
```
  0%|
| 0/3060 [00:00<?, ?it/s]
```


Original Image
0
50


Smoothed Image
0
50

In [65]:

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

# Function for image normalization
def normalize_image(frame_data):
    # Convert the image to float32 format
    frame_data = frame_data.astype(np.float32)

    # Normalize the image to the range [0, 1]
    normalized_frame_data = (frame_data - np.min(frame_data)) / (np.max(fr

    return normalized_frame_data

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Normalize the image
normalized_frame_data = normalize_image(frame_data)

# Display the original and normalized images using matplotlib
plt.figure(figsize=(12, 6))

# Plot the original image
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(frame_data, cmap='gray')
plt.colorbar()
plt.axis('off')

# Plot the normalized image
plt.subplot(1, 2, 2)
plt.title('Normalized Image')
plt.imshow(normalized_frame_data, cmap='gray')
plt.colorbar()
plt.axis('off')

plt.show()

# Print pixel values
print("\nOriginal Image Pixel Values:")
print(frame_data)
print("\nNormalized Image Pixel Values:")
print(normalized_frame_data)
```




```
Original Image Pixel Values:
[[ 97 100 101 ...  50  51  51]
 [ 85  96  96 ...  51  51  51]
 [ 76  86  80 ...  51  51  51]
 ...
 [ 65  65  64 ...  37  37  37]
 [ 64  64  63 ...  37  37  37]
 [ 65  65  64 ...  37  37  37]]

Normalized Image Pixel Values:
[[0.6        0.6272727  0.6363636  ... 0.17272727 0.18181819 0.18181819]
 [0.4909091  0.59090906 0.59090906 ... 0.18181819 0.18181819 0.18181819]
 [0.4090909  0.5        0.44545454 ... 0.18181819 0.18181819 0.18181819]
 ...
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]
 [0.3        0.3        0.29090908 ... 0.05454545 0.05454545 0.05454545]
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]]
```

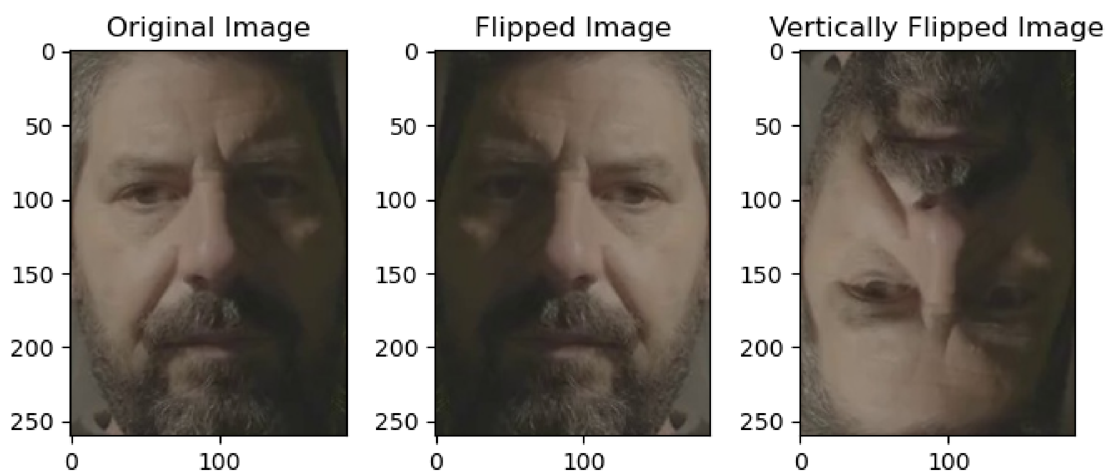
In [67]:

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

# Function for image normalization
def normalize_image(frame_data):
    # Convert the image to float32 format
    frame_data = frame_data.astype(np.float32)

    # Normalize the image to the range [0, 1]
    normalized_frame_data = (frame_data - np.min(frame_data)) / (np.max(fr

    return normalized_frame_data

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Normalize the image
normalized_frame_data = normalize_image(frame_data)

# Display the pixel values and the shape of the original image
print("Original Image:")
print("Shape:", frame_data.shape)
print("Pixel values:")
print(frame_data)

# Display the pixel values and the shape of the normalized image
print("\nNormalized Image:")
print("Shape:", normalized_frame_data.shape)
print("Pixel values:")
print(normalized_frame_data)

# Display the original and normalized images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Normalized Image')
plt.imshow(normalized_frame_data)
plt.show()
```

```
Original Image:
Shape: (260, 186, 3)
Pixel values:
[[[ 86  97 101]
  [ 89 100 104]
  [ 90 100 107]
  ...
  [ 41  49  56]
  [ 40  51  55]
  [ 40  51  55]]

 [[ 74  85  89]
  [ 85  96 100]
  [ 85  95 102]
  ...
  [ 42  50  57]
  [ 40  51  55]
  [ 40  51  55]]
```

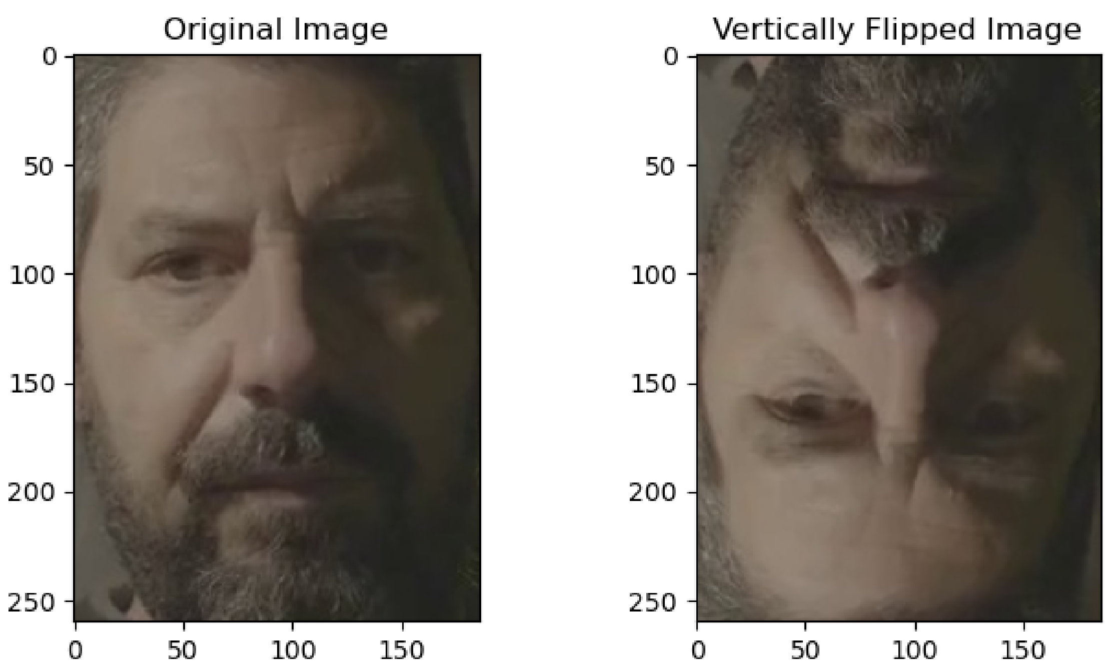
In [146]:

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

# Function for flipping images
def flip_image(frame_data):
    # Flip the image horizontally
    flipped_frame_data = np.fliplr(frame_data)

    return flipped_frame_data

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Apply image flipping
flipped_frame_data = flip_image(frame_data)
flipped_vertical = cv2.flip(frame_data, 0)

# Display the original and flipped images using matplotlib
plt.figure(figsize=(8, 3))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB))
plt.subplot(1, 3, 2)
plt.title('Flipped Image')
plt.imshow(cv2.cvtColor(flipped_frame_data, cv2.COLOR_BGR2RGB))
plt.subplot(1, 3, 3)
plt.title('Vertically Flipped Image')
plt.imshow(cv2.cvtColor(flipped_vertical, cv2.COLOR_BGR2RGB))
plt.show()
```



In [136]:

```
import cv2
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Flip the image vertically
flipped_vertical = cv2.flip(frame_data, 0)

# Display the original and vertically flipped images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Vertically Flipped Image')
plt.imshow(cv2.cvtColor(flipped_vertical, cv2.COLOR_BGR2RGB))
plt.show()
```



In [147]:

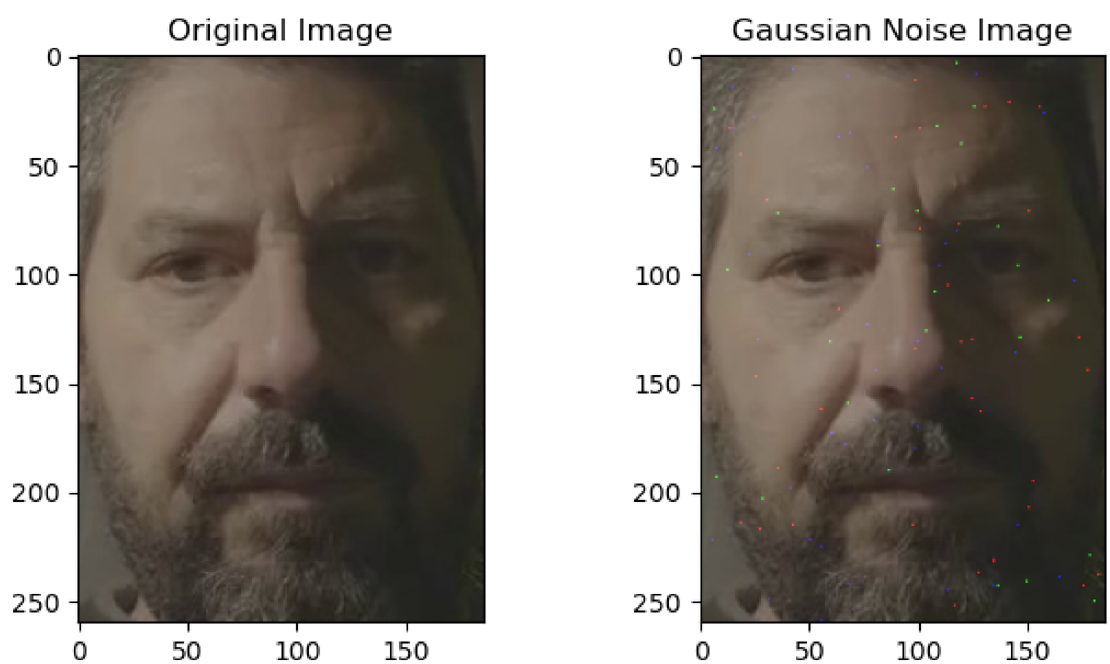
```
import cv2
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path)

# Rotate the image by 90 degrees
rows, cols = image.shape[:2]
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 20, 1)
rotated_image = cv2.warpAffine(image, M, (cols, rows))

# Display the original and rotated images using matplotlib
plt.figure(figsize=(5, 3))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Rotated Image')
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.show()
```

In [73]:

```
import cv2
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path)

# Crop the image
cropped_image = image[30:250, 30:250]

# Display the original and cropped images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Cropped Image')
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
plt.show()
```

In [151]:

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

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path)

# Generate Gaussian noise and add it to the image
mean = 0
var = 0.1
sigma = var ** 0.5
row, col, ch = image.shape
gaussian = np.random.normal(mean, sigma, (row, col, ch))
Gaussian_noisy_image = cv2.add(image, gaussian.astype(np.uint8))

# Display the original and noisy images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Gaussian Noise Image')
plt.imshow(cv2.cvtColor(Gaussian_noisy_image, cv2.COLOR_BGR2RGB))
plt.show()
```



In [152]:
```
import cv2
import matplotlib.pyplot as plt

# Read the original color image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
color_image = cv2.imread(image_path)

# Check if the image was loaded successfully
if color_image is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Convert the color image to grayscale
gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)

# Display the original color and grayscale images using matplotlib
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Color Image')
plt.imshow(cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Grayscale Image')
plt.imshow(gray_image, cmap='gray')
plt.show()
```

In [89]:

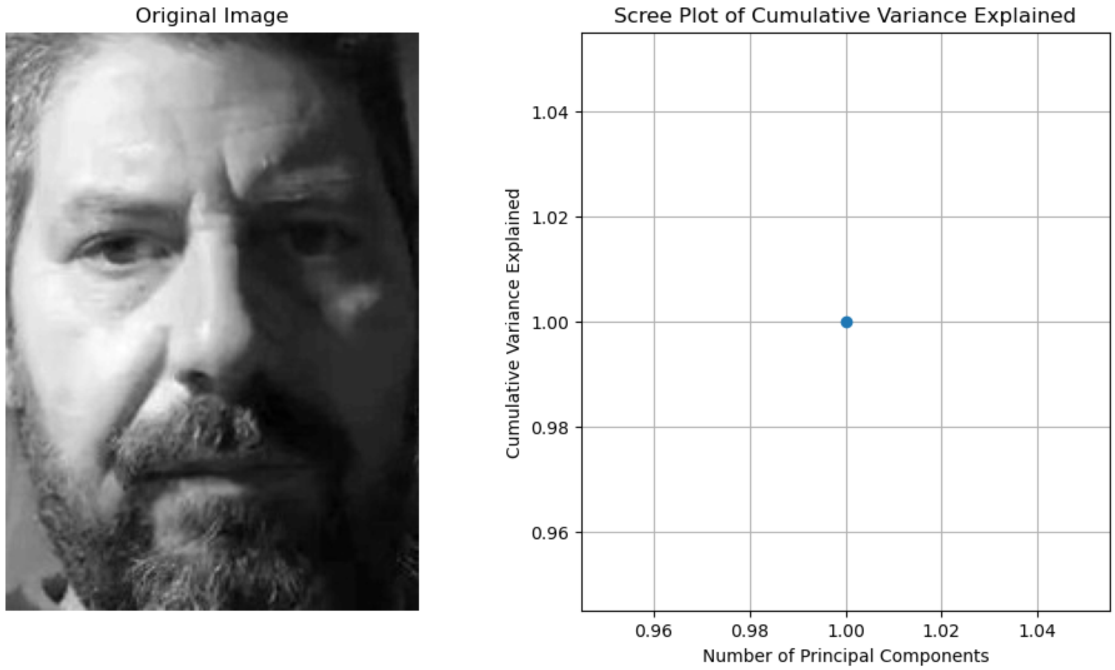
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the image into a 1D array
image_flat = image.flatten()

# Determine a reasonable value for the number of components
n_components =1  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Display the original and reconstructed images using matplotlib
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')
plt.show()
```

In [91]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the image into a 1D array
image_flat = image.flatten()

# Determine a reasonable value for the number of components
n_components = 1  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Calculate pixel-wise differences
difference = np.abs(image - image_restored)

# Display the original and reconstructed images, along with the pixel-wise
plt.figure(figsize=(16, 6))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.title('Pixel-wise Differences')
plt.imshow(difference, cmap='gray')
plt.axis('off')
plt.show()

# Print the average difference value
print(f"Average pixel-wise difference: {np.mean(difference)}")
```

```
Average pixel-wise difference: 3.6952336237317037e-16
```

In [99]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the original image into a 1D array
image_flat = image.flatten()

# Determine a reasonable value for the number of components
n_components = 1  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Calculate the absolute pixel-wise differences between the original and P
pixel_differences = np.abs(image.astype(int) - image_restored.astype(int))

# Create a scree plot showing the cumulative variance explained by each pr
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

# Plot the scree plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.plot(np.arange(1, n_components + 1), cumulative_variance_ratio, marker
plt.title('Scree Plot of Cumulative Variance Explained')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid()
plt.show()
```

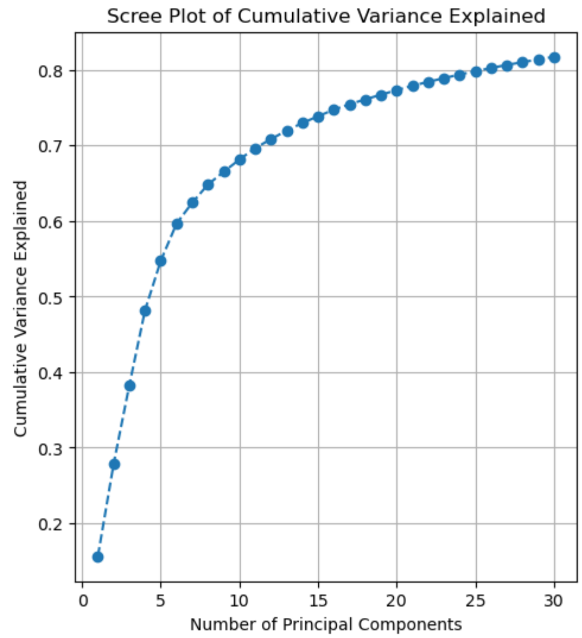

Original Image
Scree Plot of Cumulative Variance Explained
Cumulative Variance Explained
0.96
0.98
1.00
1.02
1.04
Number of Principal Components
0.96
0.98
1.00
1.02
1.04

In [105]:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Function to read images from a folder
def load_images_from_folder(folder_path):
    images = []
    for filename in os.listdir(folder_path):
        img = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_G
        if img is not None:
            img = cv2.resize(img, (100, 100))  # Resize the images to a co
            images.append(img)
    return images

# Specify the folder path containing images
folder_path = r'C:\GWAR\dataset\classification_frames\P1042756_720'  # Rep

# Load images from the folder
images = load_images_from_folder(folder_path)

# Flatten the images into a 2D array
images_flat = np.array([img.flatten() for img in images])

# Determine a reasonable value for the number of components
n_components = 30  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
images_pca = pca.fit_transform(images_flat)
images_restored = pca.inverse_transform(images_pca).reshape(images_flat.sh

# Calculate the absolute pixel-wise differences between the original and P
pixel_differences = np.abs(images_flat.astype(int) - images_restored.astyp

# Create a scree plot showing the cumulative variance explained by each pr
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

# Plot the scree plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(images[0], cmap='gray')  # Display the first image
plt.axis('off')

plt.subplot(1, 2, 2)
plt.plot(np.arange(1, n_components + 1), cumulative_variance_ratio, marker
plt.title('Scree Plot of Cumulative Variance Explained')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid()
```

```
plt.show()
```


Original Image


Scree Plot of Cumulative Variance Explained
Cumulative Variance Explained
Number of Principal Components
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30

In [107]:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from tqdm import tqdm

# Function to read images from a folder
def load_images_from_folder(folder_path):
    images = []
    for filename in tqdm(os.listdir(folder_path), desc='Loading images'):
        img = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_G
        if img is not None:
            img = cv2.resize(img, (100, 100))  # Resize the images to a co
            images.append(img)
    return images

# Specify the folder path containing images
folder_path = r'C:\GWAR\dataset\classification_frames\P1042756_720'  # Rep

# Load images from the folder
images = load_images_from_folder(folder_path)

# Flatten the images into a 2D array
images_flat = np.array([img.flatten() for img in images])

# Apply PCA with 98% variance retained
pca = PCA(0.98, svd_solver='full')
images_pca = pca.fit_transform(images_flat)
images_restored = pca.inverse_transform(images_pca).reshape(images_flat.sh

# Calculate the absolute pixel-wise differences between the original and P
pixel_differences = np.abs(images_flat.astype(int) - images_restored.astyp

# Create a scree plot showing the cumulative variance explained by each pr
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

# Print the number of components required for 98% variance
print(f"Number of components for 98% variance: {pca.n_components_}")

# Plot the scree plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(images[0], cmap='gray')  # Display the first image
plt.axis('off')

plt.subplot(1, 2, 2)
plt.plot(np.arange(1, len(cumulative_variance_ratio) + 1), cumulative_vari
plt.title('Scree Plot of Cumulative Variance Explained')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid()
plt.show()
```

```
Loading images: 100%|██████████████████████████████████████████████████| 2502/2502 [00:00<00:00, 4757.24it/s]

Number of components for 98% variance: 586
```

In [109]:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from tqdm import tqdm

# Function to read images from a folder
def load_images_from_folder(folder_path):
    images = []
    for filename in tqdm(os.listdir(folder_path), desc='Loading images'):
        img = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_G
        if img is not None:
            img = cv2.resize(img, (100, 100))  # Resize the images to a co
            images.append(img)
    return images

# Specify the folder path containing images
folder_path = r'C:\GWAR\dataset\classification_frames\P1042756_720'  # Rep

# Load images from the folder
images = load_images_from_folder(folder_path)

# Flatten the images into a 2D array
images_flat = np.array([img.flatten() for img in images])

# Apply PCA with 98% variance retained
pca = PCA(0.98, svd_solver='full')
images_pca = pca.fit_transform(images_flat)
images_restored = pca.inverse_transform(images_pca).reshape(images_flat.sh

# Select an image index for display
image_index = 0

# Display the original and reconstructed images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(images[image_index], cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(images_restored[image_index].reshape(images[image_index].shape)
plt.axis('off')
plt.show()
```

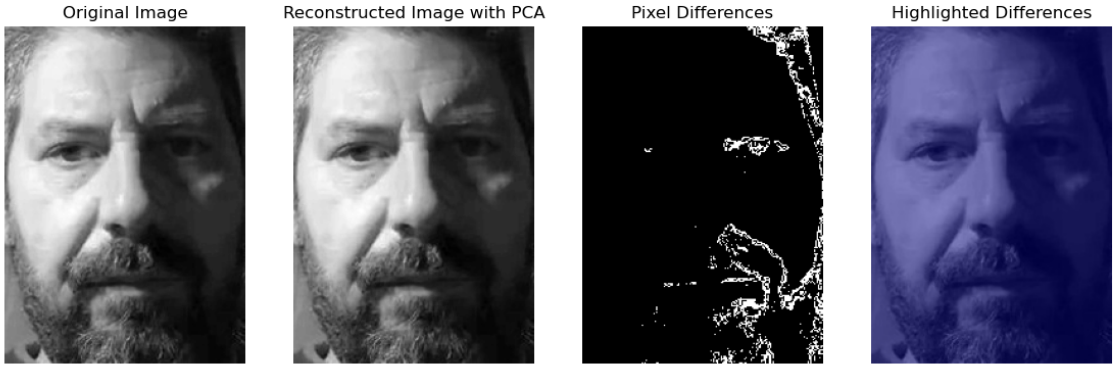
```
Loading images: 100%|██████████████████████████████████
████| 2502/2502 [00:00<00:00, 4421.47it/s]
```


Original Image
Reconstructed Image with PCA

In [111]:
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the original image into a 1D array
image_flat = image.flatten()

# Apply PCA with 98% variance retained
pca = PCA(0.95, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Display the original and reconstructed images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')
plt.show()
```

In [117]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the original image into a 1D array
image_flat = image.flatten()

# Apply PCA with 98% variance retained
pca = PCA(0.98, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Calculate the absolute pixel-wise differences between the original and r
pixel_differences = np.abs(image.astype(int) - image_restored.astype(int))

# Set a threshold for highlighting differences
threshold = 20  # Adjust as needed

# Create a mask for pixels where differences exceed the threshold
difference_mask = pixel_differences > threshold

# Display the differences between the original and reconstructed images
plt.figure(figsize=(14, 6))
plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('Pixel Differences')
plt.imshow(pixel_differences, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Highlighted Differences')
plt.imshow(image, cmap='gray')
plt.imshow(difference_mask, cmap='jet', alpha=0.5)
plt.axis('off')

plt.show()
```


Original Image
Reconstructed Image with PCA
Pixel Differences
Highlighted Differences

In [134]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Function to reconstruct an image using different numbers of principal co
def reconstruct_image(image_path, component_numbers):
    # Read the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded successfully
    if image is None:
        print(f"Could not read the image: {image_path}")
        return

    # Flatten the image into a 1D array
    image_flat = image.flatten()

    # Create subplots for the original and reconstructed images
    num_components = len(component_numbers)
    fig, axes = plt.subplots(1, num_components + 1, figsize=(15, 3))

    # Display the original image
    axes[0].imshow(image, cmap='gray')
    axes[0].set_title('Original Image')
    axes[0].axis('off')

    for i, num_components in enumerate(component_numbers):
        # Ensure that n_components is within the valid range
        num_components = min(num_components, min(image_flat.shape))

        # Apply PCA with the current number of components
        pca = PCA(n_components=num_components, svd_solver='full')
        image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
        image_restored = pca.inverse_transform(image_pca).reshape(image.sh

        # Display the reconstructed image
        axes[i + 1].imshow(image_restored, cmap='gray')
        axes[i + 1].set_title(f'{num_components} components')
        axes[i + 1].axis('off')

    plt.tight_layout()
    plt.show()

# Specify the path to the single image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j

# Specify the range of components to consider
component_numbers = [1]

# Reconstruct the image using different numbers of principal components
reconstruct_image(image_path, component_numbers)
```

Original Image

1 components

In [ ]: