Image Proceesing Project:

Enhancements of images captured from security cameras

Project Idea:

The project's idea is to enhance images from security cameras using image processing techniques.

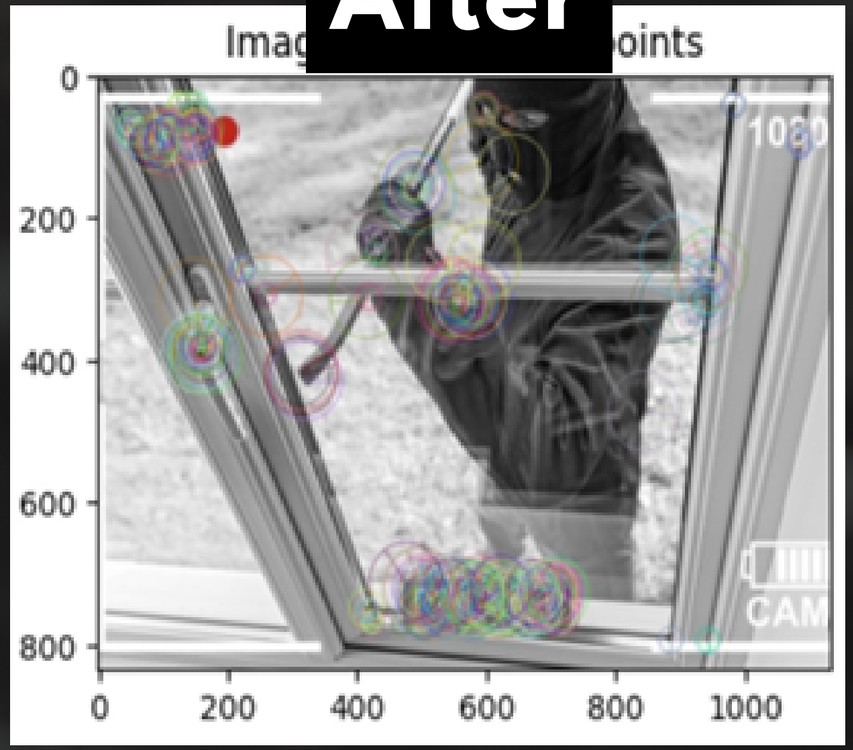
Project Goal:

The goal is employ image processing techniques in security cameras to enhance security camera footage to prevent and discourage criminal activities. Clearer and more detailed images enable early detection of potential threats, allowing security personnel to take quick actions and avoid risks effectively.

1

Before

After



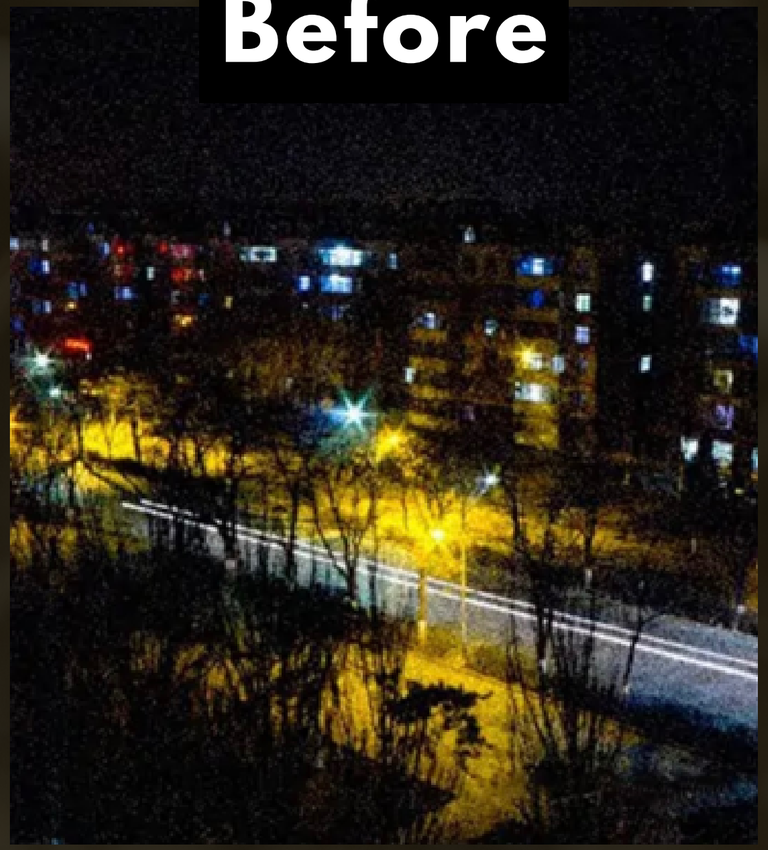
Methods used:

- cv2.ORB_create()
- orb.detectAndCompute()
- cv2.drawKeypoint()

```python
# this line to convert the  image to grayscale for feature detection
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize the ORB detector
orb = cv2.ORB_create()

# Detect ORB keypoints and compute descriptors
keypoints, descriptors = orb.detectAndCompute(gray_image, None)

# Draw the keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

2

Before



After

Methods used:

- bilateral filter for noise reduction
- Laplacian
- Gamma correction to make image more contract

```
# this line to apply bilateral filter for noise reduction
noiseReduc_image = cv2.bilateralFilter(image, 9, 75, 75)

# Apply image sharpening using a kernel
kernel = np.array([[-1, -1, -1],
                   [-1, 9, -1],
                   [-1, -1, -1]])
sharpened_image = cv2.filter2D(noiseReduc_image, -1, kernel)

# this line to apply laplacian sharpening
laplacian = cv2.Laplacian(sharpened_image, cv2.CV_64F)
sharpened_image_laplacian = sharpened_image - 0.3 * laplacian

# Convert the Laplacian-sharpened image to 8-bit unsigned integer
sharpened_image_laplacian = cv2.convertScaleAbs(sharpened_image_laplacian)

# Apply gamma correction
gamma = 0.9
gamma_corrected = np.clip(sharpened_image_laplacian ** (1/gamma), 0, 255).astype(np.uint8)
```

Before

After

3

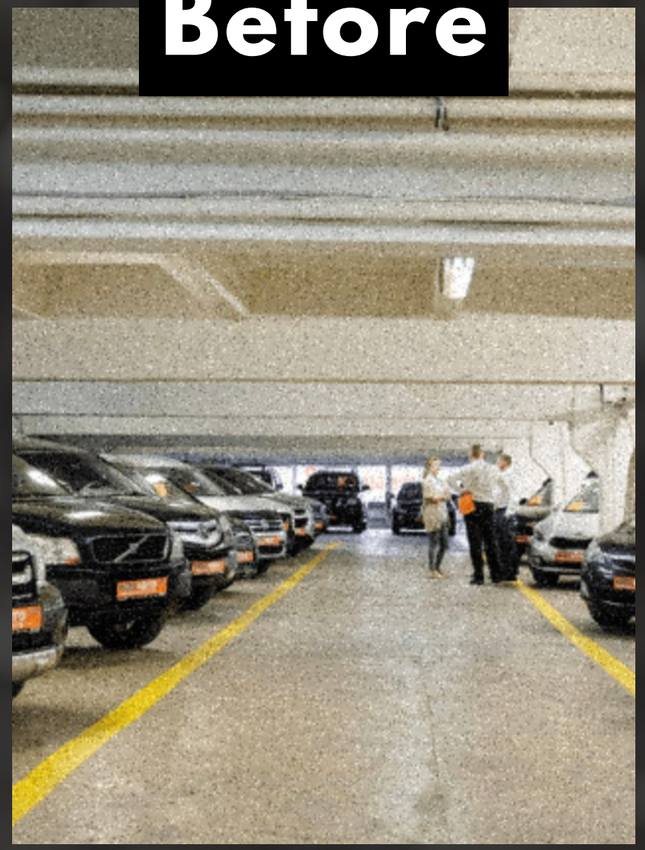
Methods used:

- bilateral filter for noise reduction
- Adjust contrast to make image brightness
- Gaussian Blur
- Histogram equalizationto better contract

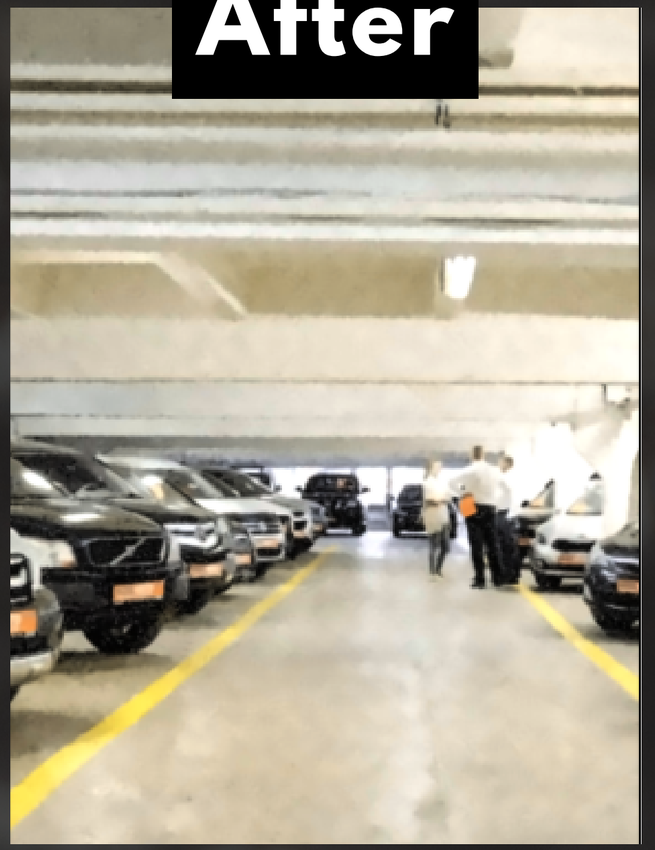
```python
# Apply bilateral filter for noise reduction
noiseReduc_image33 = cv2.bilateralFilter(image33, 9, 75, 75)

# Adjust contrast using cv2.multiply to make img Brightness
contrast_factor = 1.5
contrast_image33 = cv2.multiply(noiseReduc_image33, np.array([contrast_factor]))

# Apply Unsharp Masking for detail enhancement
def unsharp_mask(image, sigma=1.0, strength=1.5):
    blurred = cv2.GaussianBlur(image, (0, 0), sigma)
    sharpened = cv2.addWeighted(image, 1.0 + strength, blurred, -strength, 0)
    return sharpened

sharpened_image33 = unsharp_mask(contrast_image33)

# Convert the sharpened image to grayscale
sharpened_gray = cv2.cvtColor(sharpened_image33, cv2.COLOR_BGR2GRAY)

# Apply histogram equalization for better contrast
equalized_image33 = cv2.equalizeHist(sharpened_gray)
```

4

Before



After



Methods used:

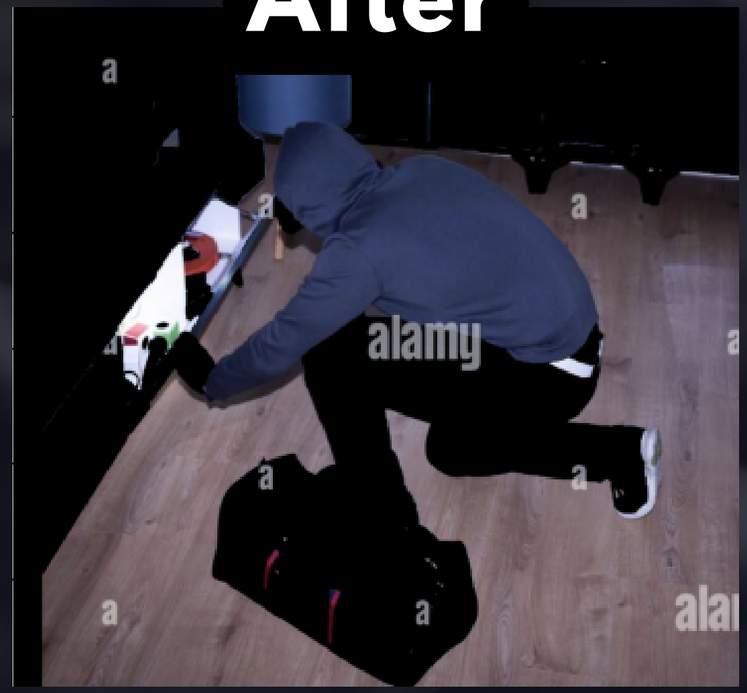
- cv2.fastNlMeansDenoisingColored
- cv2.multiply

```
# Load the image with salt-and-pepper noise
image_path = '/content/parking.jpg'
image = cv2.imread(image_path)

# Apply Non-Local Means Denoising
denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)

# Enhance color by increasing contrast
contrast_factor = 1.2
enhanced_color = cv2.multiply(denoised_image, np.array([contrast_factor]))
```

5

Before

After



Methods used:

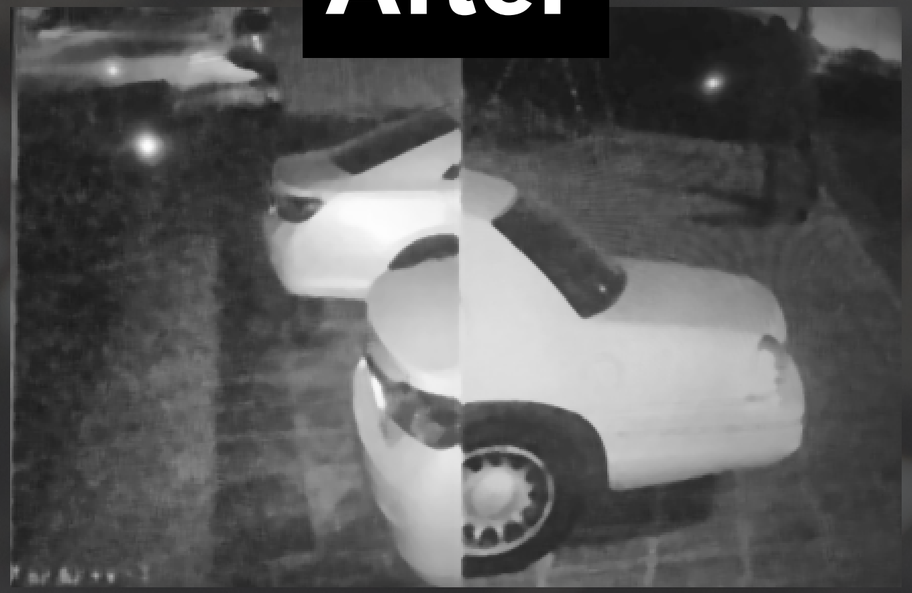
- Apply GrabCut algorithm cv2.grabCut
- Modify the mask to create a binary mask for the foreground np.where, that to cut whout loss the qulity of image

```python
def remove_background(image_path):
    # Read the image
    image = cv2.imread(image_path)

    # Create a mask (binary image) for the foreground and background
    mask = np.zeros(image.shape[:2], np.uint8)

    # Define a rectangle around the object to help GrabCut algorithm
    rect = (50, 50, image.shape[1]-50, image.shape[0]-50)

    # Initialize foreground and background models for GrabCut
    bgdModel = np.zeros((1, 65), np.float64)
    fgdModel = np.zeros((1, 65), np.float64)

    # Apply GrabCut algorithm
    cv2.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

    # Modify the mask to create a binary mask for the foreground
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

    # Apply the binary mask to the original image
    result = image * mask2[:, :, np.newaxis]

    return result
```

6

Before

After



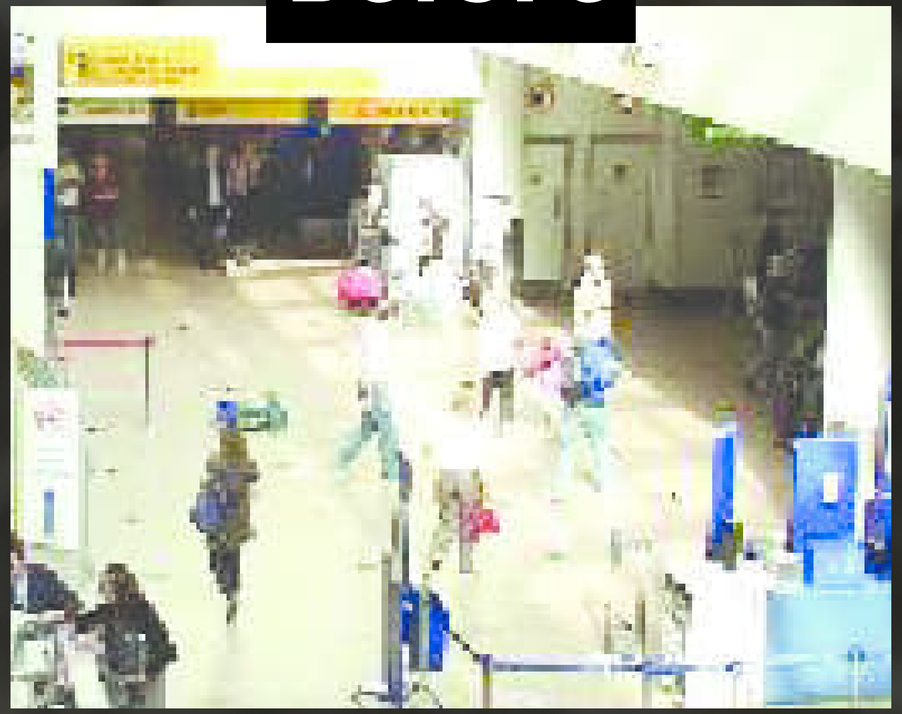
Methods used:

- median filter to remove salt and pepper noise.
- Gamma correction to adjust the brightness and contrast to make the background visible.

```
# adjust kernel size for the median filter
kernel_size=7

# Apply the median filter to remove salt and pepper noise
denoised_image = cv2.medianBlur(image1, kernel_size)

# Define the gamma value for gamma correction
gamma = 0.6  #gamma <1=lighten image, gamma >1 = darken image

# Apply gamma correction to adjust the brightness and constrast to make the background visible
gammaCorrection = Gamma_Correction(denoised_image, gamma)
```

7

Before



After

Methods used:

- bilateral filtering.
- Gamma correction to adjust the brightness and contrast.

```
smoothed_image = cv2.bilateralFilter(image2, d=11, sigmaColor=85, sigmaSpace=90)

# Convert the BGR images to RGB
smoothed_image_rgb = cv2.cvtColor(smoothed_image, cv2.COLOR_BGR2RGB)

# Define the gamma value for gamma correction
gamma = 1.3
# Enhance the colors using gamma correction
adjustLight=Gamma_Correction(smoothed_image_rgb, gamma)
```

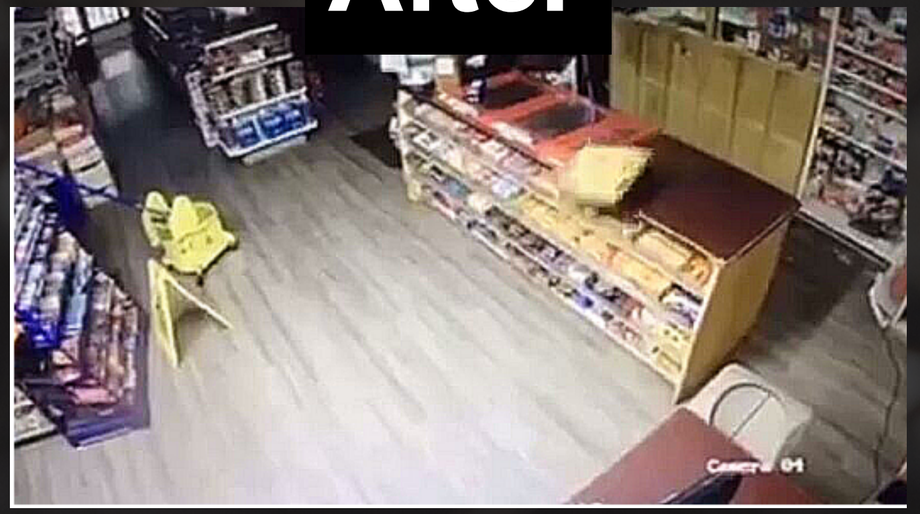
8

Methods used:

bilateral filtering.
filter2D for sharpening.

Before

After

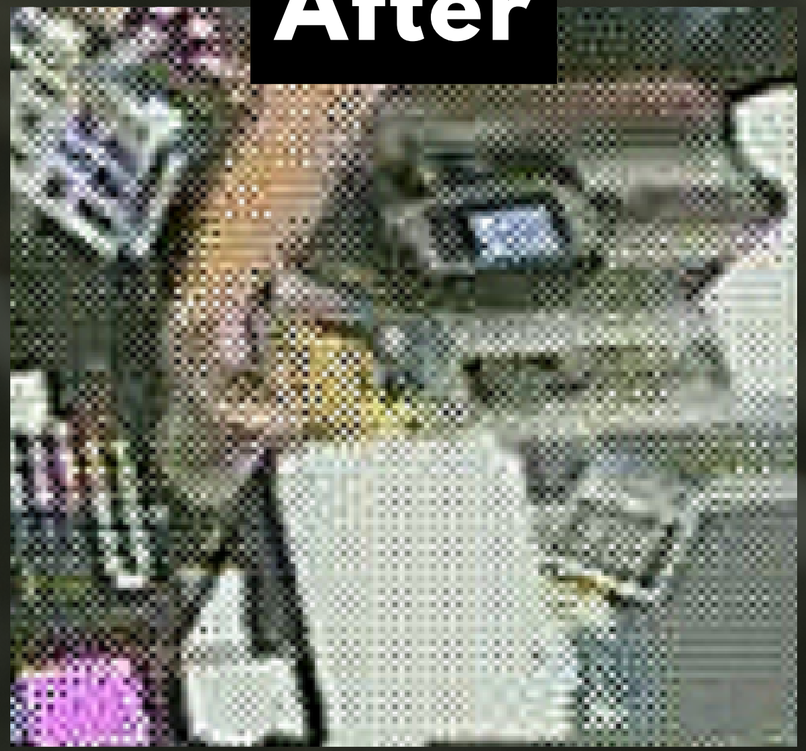


```
# Create a sharpening kernel
sharpen_filter = np.array([[-1, -1, -1],
                           [-1,  9, -1],
                           [-1, -1, -1]])

# Apply the sharpening kernel to the input image
sharp_image = cv2.filter2D(image4, -1, sharpen_filter)
sharp_image = cv2.cvtColor(sharp_image, cv2.COLOR_BGR2RGB)
```

9

Methods used:

- Filter2D for sharpening.
- Crop the image to the ROI.



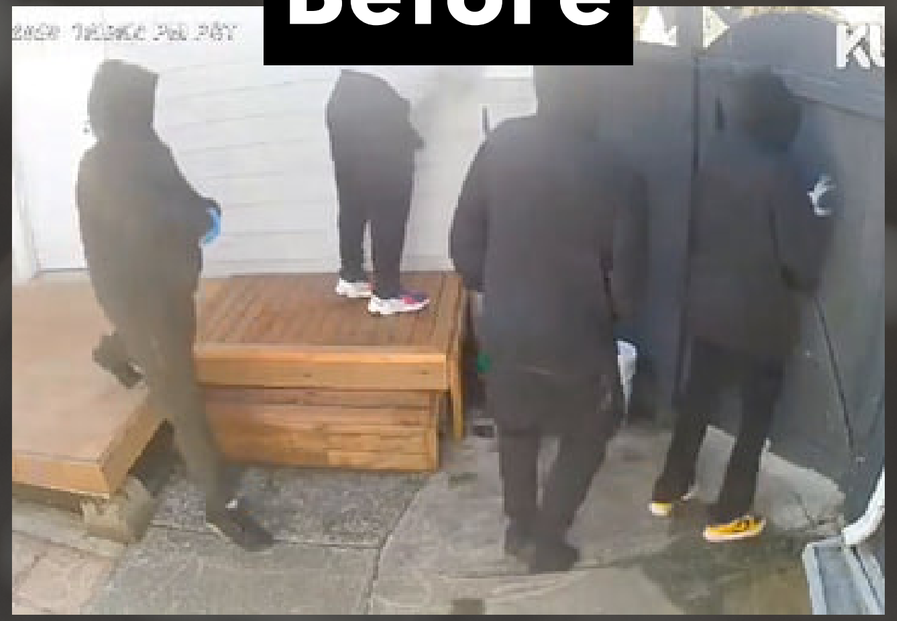

```
# Create the sharpening kernel
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

# Sharpen the image
sharpened_image = cv2.filter2D(image5, -1, kernel)

# crop image
y=100
x=150
h=230
w=270
crop_image = sharpened_image[x:w, y:h]
```

10

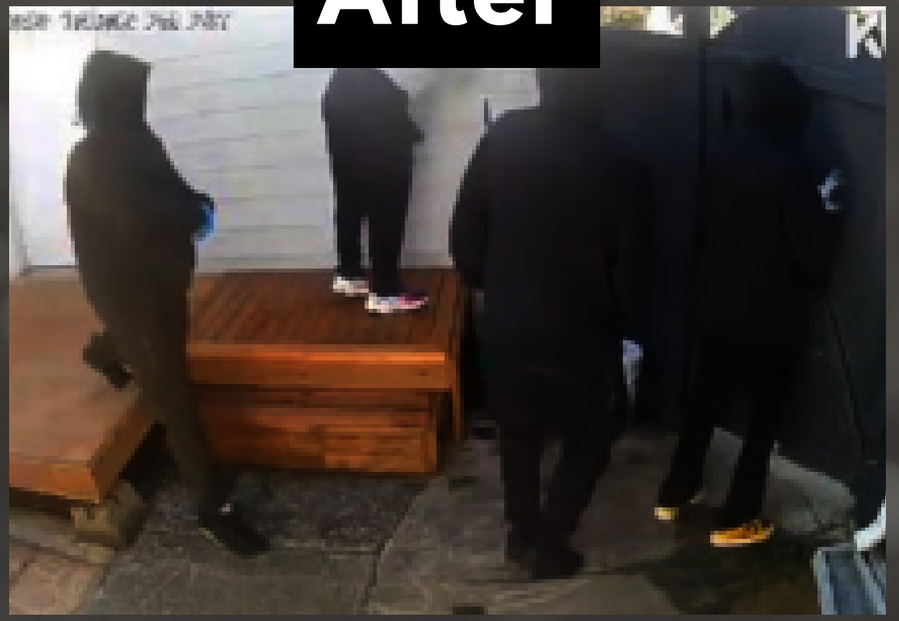
Methods used:

Gamma correction on each color channel.

Before



After



```python
# Define the gamma correction function
def gamma_correction(channel, gamma):
    return np.power(channel, gamma)

# Define the gamma value
gamma = 2.6

# Perform gamma correction on each channel
gamma_corrected_b = gamma_correction(image_float[:, :, 0], gamma)
gamma_corrected_g = gamma_correction(image_float[:, :, 1], gamma)
gamma_corrected_r = gamma_correction(image_float[:, :, 2], gamma)

# Merge the color channels back into an RGB image
gamma_corrected_image = np.dstack((gamma_corrected_b, gamma_corrected_g, gamma_corrected_r))

# Convert the image back to the original data type
gamma_corrected_image = (gamma_corrected_image * 255).astype(np.uint8)
```

11

Before

After

Methods used:

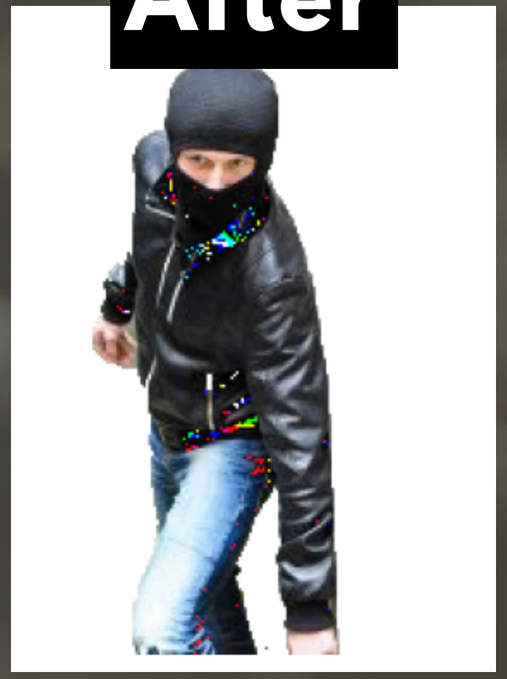
Histogram equalization.

```
# Perform histogram equalization
equalized_image = cv2.equalizeHist(image)
```

12

Before

After



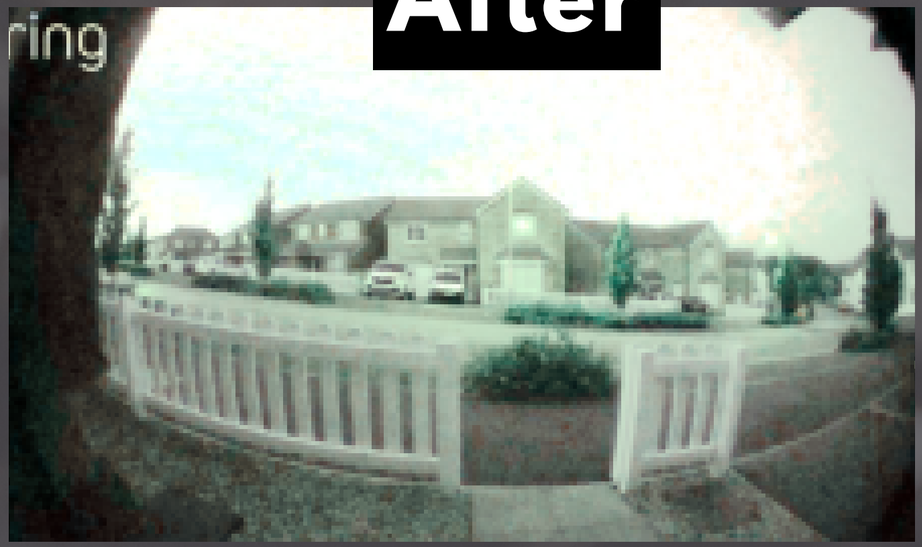
Methods used:

- Crop the image to the ROI.
- GrabCut algorithm to update the mask

```
# Define the ROI coordinates for the person
x, y, h, w = 50, 210, 300, 560

# Crop the image to the ROI
roi_image = image[x:h, y:w]

# Create a mask for the background
mask = np.zeros(roi_image.shape[:2], np.uint8)

# Define background and foreground models
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)

# Define the rectangle for GrabCut algorithm
rectangle = (10, 10, roi_image.shape[1] - 10, roi_image.shape[0] - 10)

# Apply GrabCut algorithm to update the mask
cv2.grabCut(roi_image, mask, rectangle, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

# Create a mask where the background is
background_mask = np.where((mask == 0) | (mask == 2), 0, 1).astype('uint8')

# Apply the background mask to the cropped image
background_removed = roi_image * background_mask[:, :, np.newaxis]

# Create a white background image
white_background = np.ones_like(roi_image) * 255

# Combine the background-removed image with the white background
final_image = white_background + background_removed
```

Before

After



13

Methods used:

- Histogram equalization to each channel.
- Bi-linear Interpolation to improve resolution.
- Adjust the contrast using enhance_visibility function.

```python
# Convert BGR image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Split the RGB channels
r, g, b = cv2.split(image_rgb)

# Apply histogram equalization to each channel
r_eq = cv2.equalizeHist(r)
g_eq = cv2.equalizeHist(g)
b_eq = cv2.equalizeHist(b)

# Adjust the green channel
green_adjusted = cv2.addWeighted(g_eq, 0.2, b_eq, 0.9, 0)

# Merge the adjusted channels back into an RGB image
adjusted_image_rgb = cv2.merge((r_eq, green_adjusted, b_eq))

# Apply Bi-linear Interpolation to improve resolution
upscale_factor = 2  # Adjust the upscale factor as needed
upsampled_image = cv2.resize(adjusted_image_rgb, None, fx=upscale_factor, fy=upscale_factor, interpolation=cv2.INTER_LINEAR)

# Adjust the contrast using enhance_visibility function
alpha=1
beta=1.2
enhance_contrast=enhance_visibility(upsampled_image, alpha, beta)
```

conclusion:

In conclusion, the enhancement image project successfully utilized image processing techniques to enhance the quality of images captured from security cameras. By applying denoising, deblurring, and resolution enhancement algorithms, the project effectively reduced noise, improved sharpness, and increased the level of detail in the images. These enhancements have significant implications for various sectors, including residential, commercial, and public spaces.

Thank you

Danah Bawajeeh 2115590
Ramah Alharbi 2110173