# ASSIGNMENT 1

## Giving Cartoon Effect to Colorful Images

Sara Rzayeva-2210356005

```python
def smooth_image(input_image, sigma):

    return cv2.GaussianBlur(input_image,(5,5),0)
```

**STEP 1**-The function ”smooth image” applies image smoothing using a Gaussian blur. ”cv2.GaussianBlur ” function takes input image and a parameter sigma for the Gaussian kernel.

```python
# Step 2: Edge Detection (DoG)
def edge_detection(input_image, ksize=(5, 5), ksigma=1.0, k=2.0, threshold=10):
    gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    g1 = cv2.GaussianBlur(gray_image, ksize, ksigma)
    g2 = cv2.GaussianBlur(gray_image, ksize, ksigma * k)
    dog = g1 - g2
    edge_image = cv2.threshold(dog, threshold, 255, cv2.THRESH_BINARY)[1]
    return edge_image
```

**STEP 2**-The function ”edge_detection” applies difference of Gaussians method. It converts the input image to grayscale, applies g1 and g2 which are two Gaussian blurs. then DoG kernel as the differences of two Gaussian functions. Lastly function thresholds the result .

```python
# Step 3: Image Quantization
def quantize_image(input_image, channels_to_quantize=['L']):
    lab_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2Lab)
    if 'L' in channels_to_quantize:
        lab_image[:, :, 0] = (lab_image[:, :, 0] // 12) * 10  # Quantize the L channel
    quantized_image = cv2.cvtColor(lab_image, cv2.COLOR_Lab2BGR)
    return quantized_image
```

**STEP 3**- The function ”quantize_image” quantizes the L channel by dividing its values( In this picture its 12 but use different values depending on the picture) ,then rounding down to the nearest integer , after e multiply it by 10. (but for different pictures i play with this values to get the best version of the picture).

```python
# Step 4: Combine Edge and Quantized Image
def combine_images(quantized_image, edge_image):
    inverse_edges = 255 - edge_image
    combined_image = cv2.bitwise_and(quantized_image, quantized_image, mask=inverse_edges)
    return combined_image
```

**STEP4**-The function ”combine image” combines the quantized image and the edge image. It substracs edge image from 255 and creates a mask ”inverse_edges” and performs a bitwise . And operation between the quantized image and the mask.

```python
# Apply image abstraction step
smoothed = smooth_image(input_image, sigma=5)
edges = edge_detection(smoothed, ksize=(1, 5), ksigma=2.4, k=2.5, threshold=10)
quantized = quantize_image(smoothed, channels_to_quantize=['L'])
result = combine_images(quantized, edges)
```

**Lastly,** I applied the image abstraction steps. First we smooth the input image, then we perform edge detection, quantize image, then we combine quantized image with the edge image.

# RESULTS OF MY ALGHORITHM

## FIRST IMAGE

## INPUT

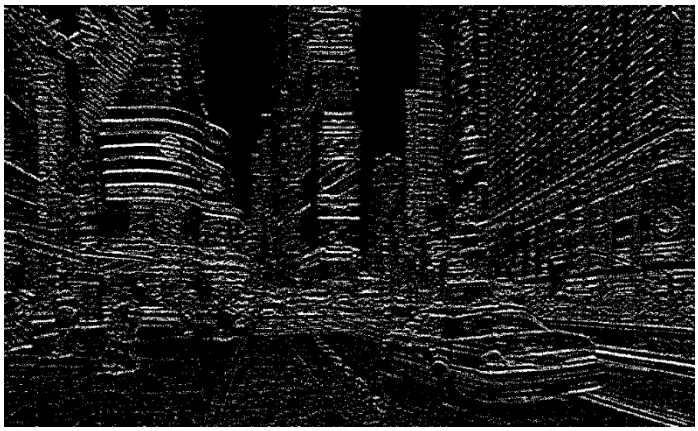
I chose this image because there is a lot of details like skyscrapers, people walking and the taxi.

## OUTPUT IMAGE

### SMOOTHING

```
Cv2.GaussianBlur(input_image, (5, 5), 4).
```

**EDGE DETECTION**



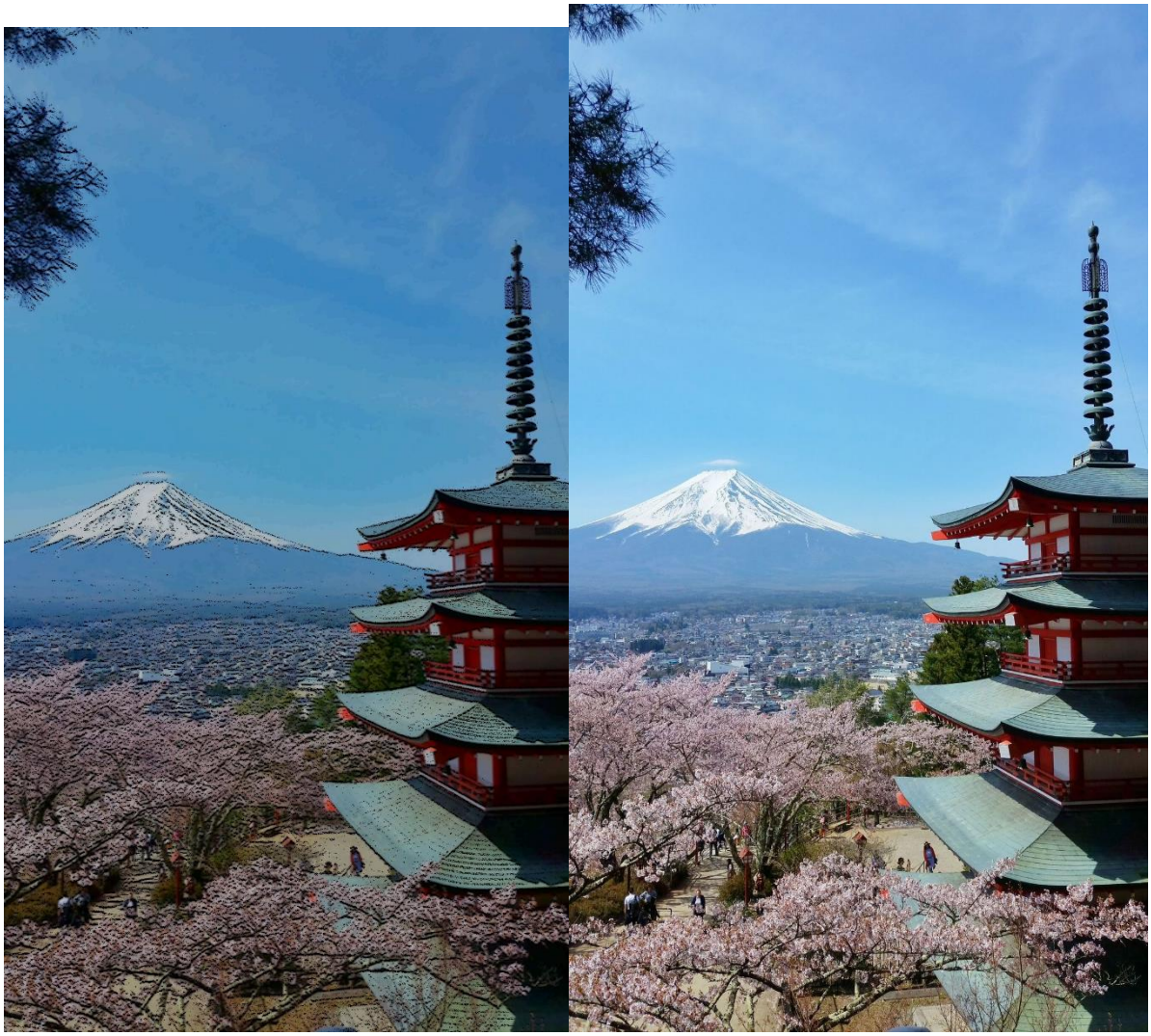
The edges are white when i apply edge detection but when we combine quantize and edge detection I inverse the edges. `ksize=(1, 5), ksigma=2.4, k=2.5, threshold=10.` These are the values that I used. When the k values are close, program gives the best result. ksize(1,5) gives smaller kernel along the x-axis and larger kernel along the y-axis when we use edge detection. For example When the ksize (5,5) I get a lot of black dots in the output image. That is why I modified the ksize to get the best version of the image.

**IMAGE QUANTIZATION**

`lab_image[:, :, 0] = (lab_image[:, :, 0] // 13) * 10.` These are the values for this picture. A smaller divisor gave me more details and nuances in luminance.

## SECOND IMAGE

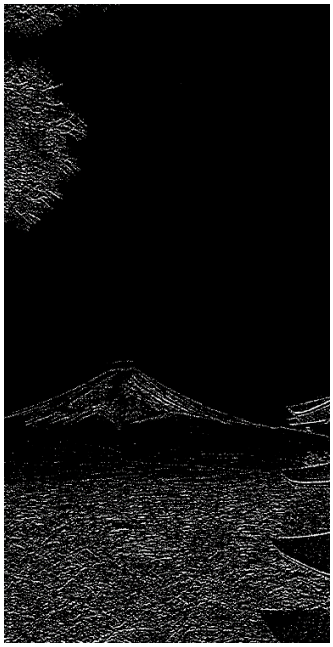
**OUTPUT** **INPUT**



**SMOOTHING**

```
return cv2.GaussianBlur(input_image, (5, 5), 5)
```
parameters I used for smoothing in this picture.

EDGE DETECTION



```
edges = edge_detection(smoothed, ksize=(1, 5), ksigma=2.0, k=2.5, threshold=20).
```
As you can see from the picture

it does not take the edges of windows. I could have fixed it by changing ksize to (5,5). But when I change ksize it takes edges of the sky. As a result in the image you can see little black dots all over the sky. For me it is the best

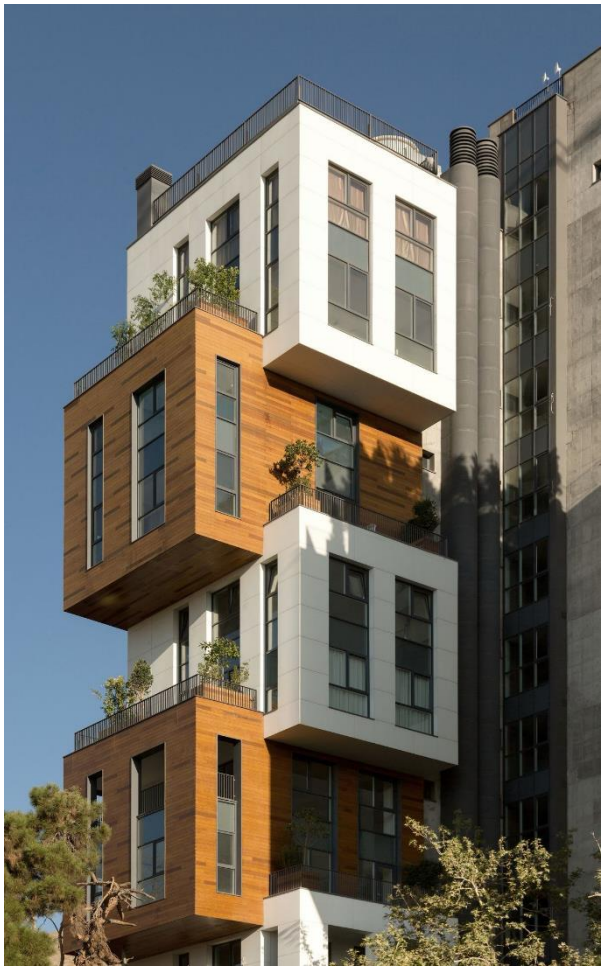
not having that dots in this image that is why I used ksize as (1,5).

## QUANTIZATION

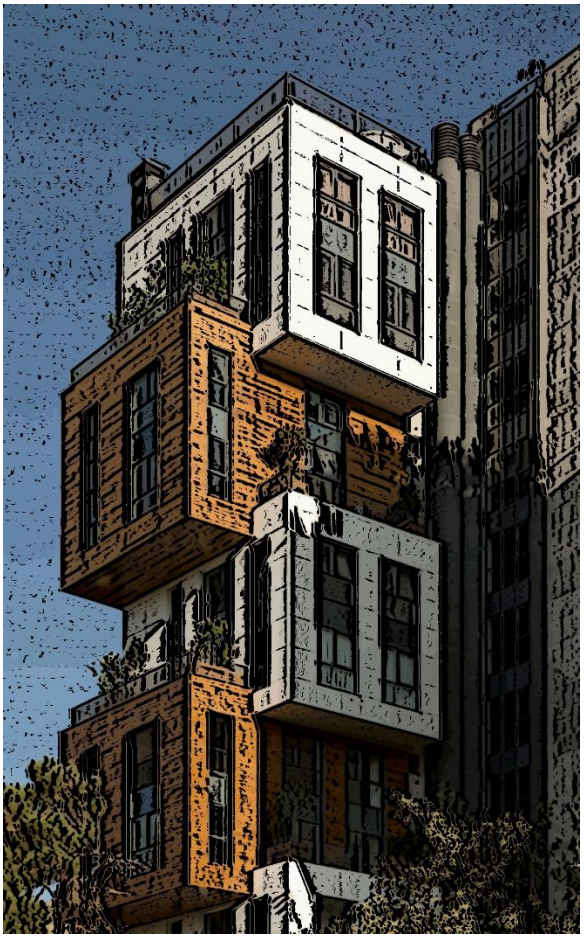
`lab_image[:, :, 0] = (lab_image[:, :, 0] // 5) * 4.`these are the values for this picture. I used even smaller divisor than before for this picture. Because when the divisor is a higher number the result was dark bands around clouds. And the image did not look good because there wasn’t smooth transitions.

## THIRD IMAGE

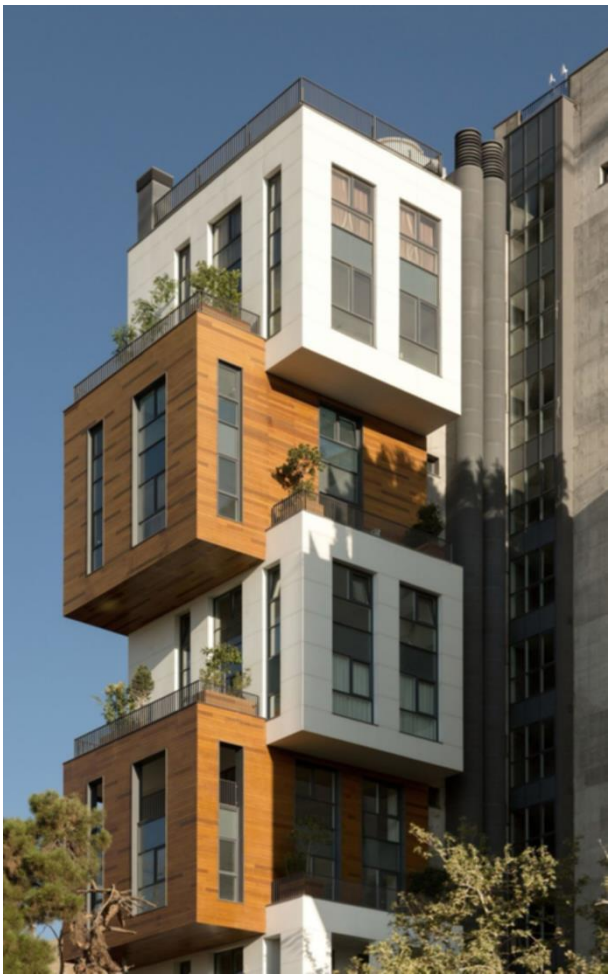
## INPUT



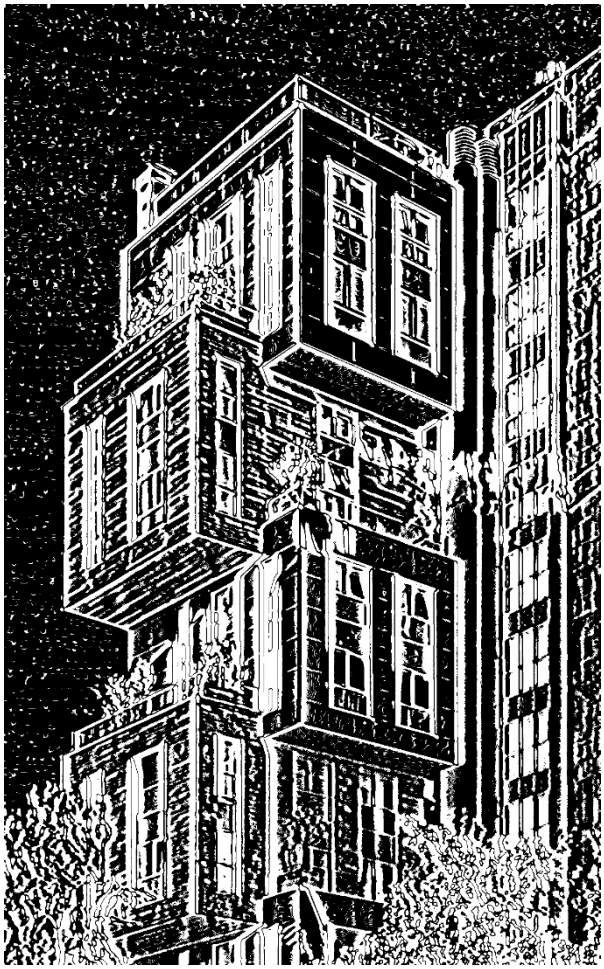
## OUTPUT



**SMOOTHING**



**EDGE DETECTION**



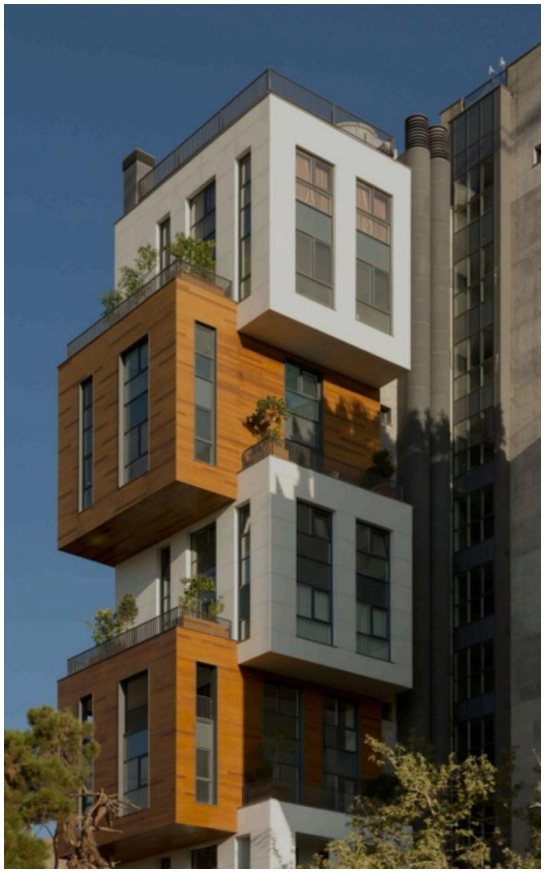
Parameters i used for edge detection-

```
edges = edge_detection(smoothed, ksize=(0, 5), ksigma=2.4, k=2.5, threshold=10).
```

This is the best result that i could get in this image. I played with ksize and threshold values. I did not wanted that black dots in the sky but when i changed the ksize to (1,5) it did not looked like cartoon effect that much. I tried to solve it with thresholding because i thought intensity,such as noise, maybe detected as edges. But it did not made noticable difference on the appearance of black dots.

## QUANTIZATION

```
lab_image[:, :, 0] = (lab_image[:, :, 0] // 5) * 4
```



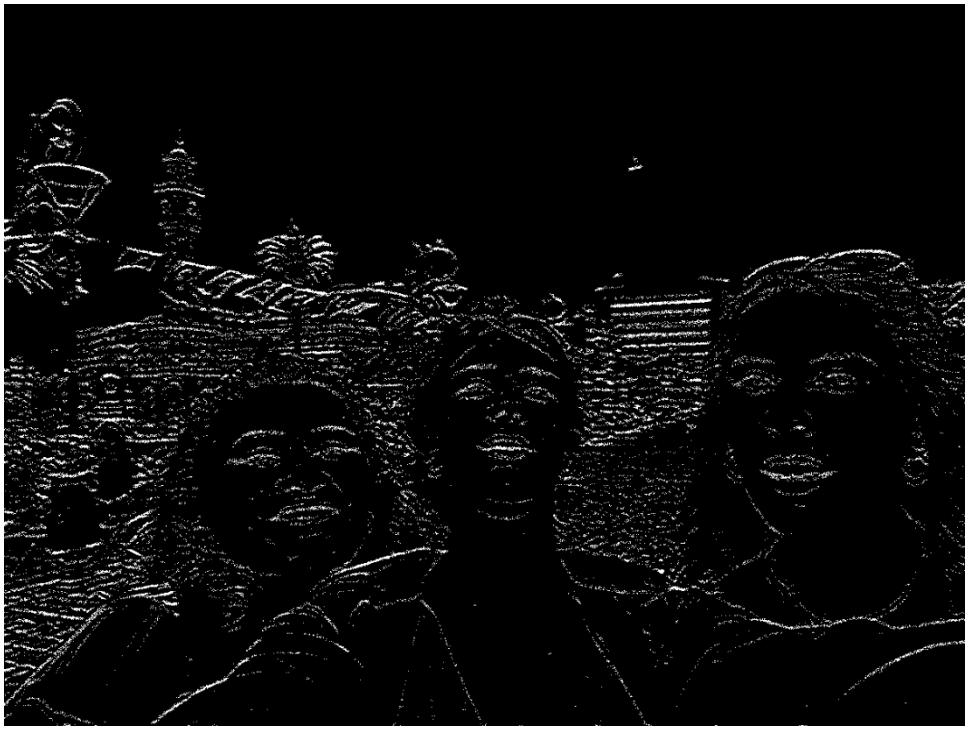
## FOURTH IMAGE

**INPUT**

**OUTPUT**

**SMOOTHING**

**EDGE DETECTION-** `edges = edge_detection(smoothed, ksize=(1, 5), ksigma=2.9, k=3.0, threshold=5)`



**QUANTIZATION-** `lab_image[:, :, 0] = (lab_image[:, :, 0] // 5) * 4`

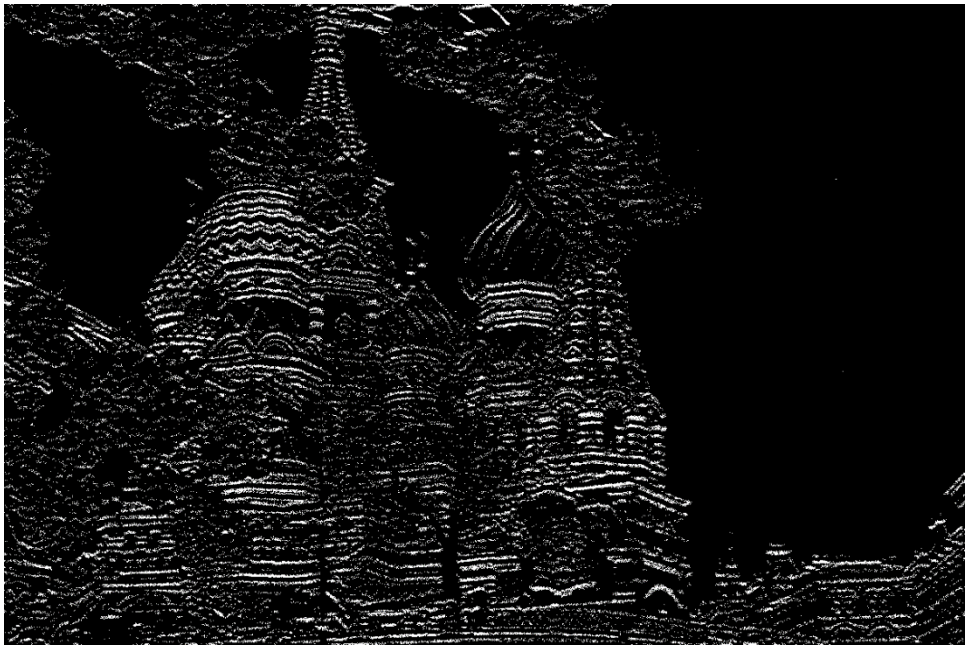
**FIFTH IMAGE**

**INPUT**

**OUTPUT**

**SMOOTHING** `smoothed = smooth_image(input_image, sigma=10)`

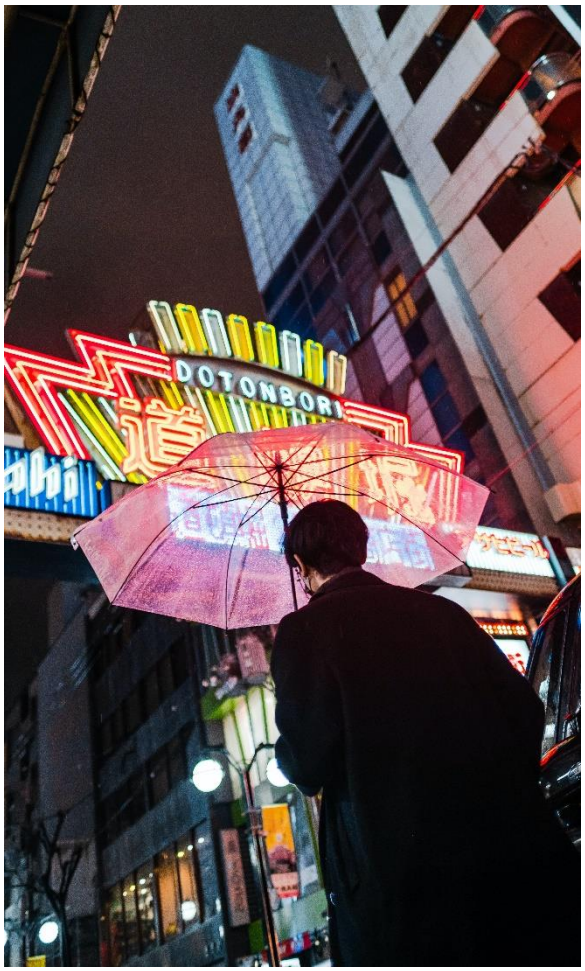
**EDGE DETECTION** `edges = edge_detection(smoothed, ksize=(1, 5), ksigma=2.3, k=2.5, threshold=10)`



**QUANTIZATION** `lab_image[:, :, 0] = (lab_image[:, :, 0] // 5) * 4`

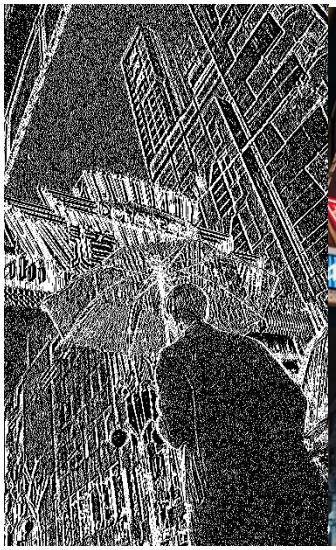
## SIXTH IMAGE

## INPUT




## OUTPUT

## EDGE DETECTION



## SMOOTHING


## QUANTIZATION

## SEVENTH IMAGE

**INPUT**

**OUTPUT**

**SMOOTHING**

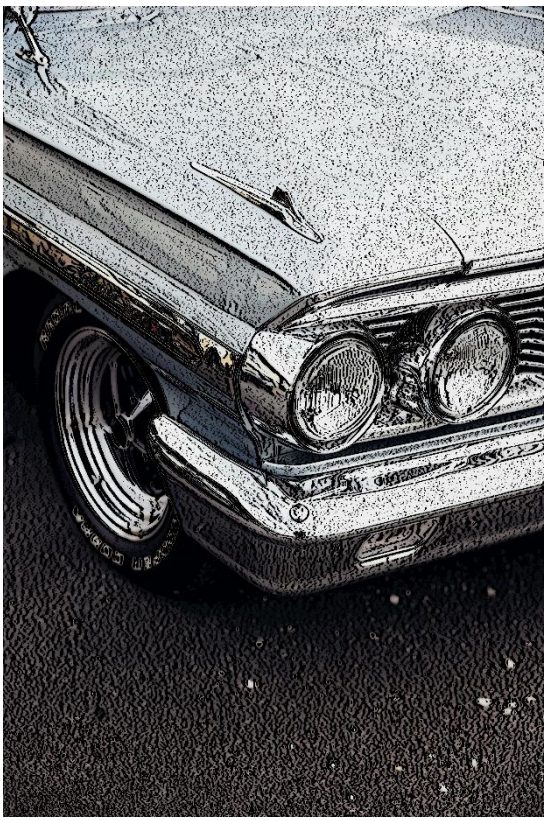
**EDGE DETECTION**

**QUANTIZATION**

## EIGTH IMAGE

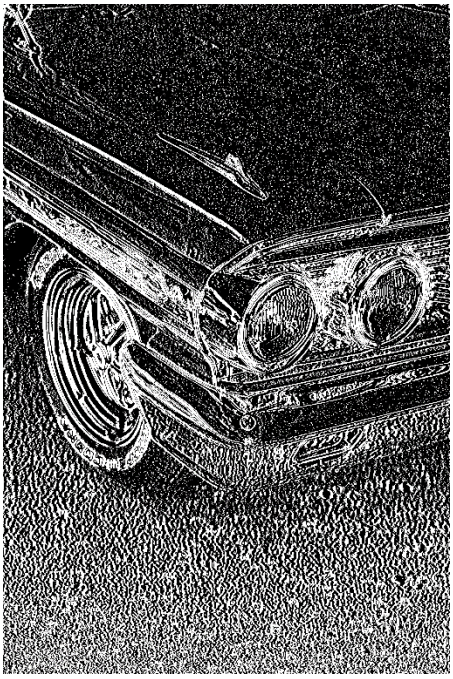
## INPUT

## OUTPUT



## SMOOTHING

## EDGE DETECTION



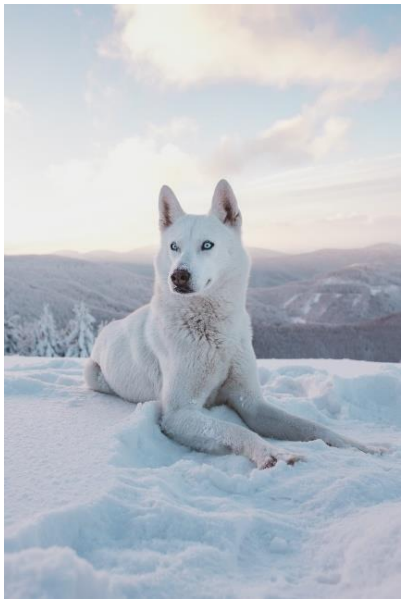
## QUANTIZATION

**NINTH IMAGE**

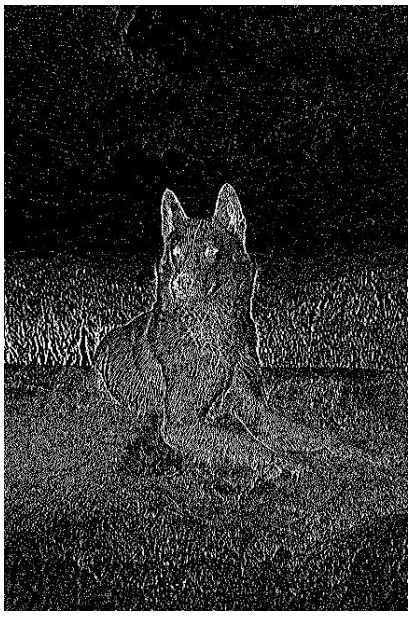
**INPUT**

**OUTPUT**

**SMOOTHING**



**EDGE DETECTION**



**QUANTIZATION**

**IN the LAST FOUR IMAGES, I USED THE SAME PARAMETERS FOR ALL FUNCTIONS  BECAUSE THE IMAGES I USED WERE HIGH-RESOLUTION. THEREFORE, THEY REQUIRED MORE CAREFUL PARAMETER TUNING TO BALANCE THE PRESERVATION OF FINE DETAILS AND THE ENHANCEMENT OF EDGE VISIBILITY.**

```
# Apply image abstraction step
smoothed = smooth_image(input_image, sigma=5)
edges = edge_detection(smoothed, ksize=(0, 5), ksigma=2.4, k=2.5, threshold=10)
quantized = quantize_image(smoothed, channels_to_quantize=['L'])
result = combine_images(quantized, edges)
```

```
# Step 3: Image Quantization
def quantize_image(input_image, channels_to_quantize=['L']):
    lab_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2Lab)
    if 'L' in channels_to_quantize:
        lab_image[:, :, 0] = (lab_image[:, :, 0] // 10) * 10 # Quantize the L channel
    quantized_image = cv2.cvtColor(lab_image, cv2.COLOR_Lab2BGR)
    return quantized_image
```