Fall 2023 BBM 415 - AIN 432: Image Processing Lab Assignment 1 Report

Yağız Buğra Boz
2200356009
19 Kasım 2023

Brief Overview of The Problem

We need to write code that takes an image as input and produces a cartoon-like version of it. To do this, filtering methods must be used. Here is my solution for this.

Explanation of Code

```
inputimg = "input.jpg"
smoothedimg = "smoothed.jpg"
edgesimg = "edges.jpg"
quantizedimg = "quantized.jpg"
finalimg = "final.jpg"
```

File names for each step are defined in this part.

```
image = Image.open(inputimg).convert('RGB')
filterType = "gaussian"
sigma = 1
if filterType == 'gaussian':
    smoothed = image.filter(ImageFilter.GaussianBlur(radius=sigma))
    smoothed.save(smoothedimg)
elif filterType == 'median':
    smoothed = image.filter(ImageFilter.MedianFilter(size=sigma))
    smoothed.save(smoothedimg)
```

Smoothing: In this part, the image is opened and it is converted to RGB format. Then the smoothing format is chose either 'gaussian' or 'median' and sigma value is assigned. The image is blurred and is saved it to ‘smoothedimg’.

```
sigma1 = 1.4
sigma2 = 1.6
epsilon = 50
img = Image.open(smoothedimg).convert('L')
blurredImg1 = np.array(img.filter(ImageFilter.GaussianBlur(radius=sigma1)))
blurredImg2 = np.array(img.filter(ImageFilter.GaussianBlur(radius=sigma2)))
dog_image = blurredImg1 - blurredImg2
thresholdedDog = np.where(dog_image >= epsilon, 255, 0).astype('uint8')
Image.fromarray(thresholdedDog, mode='L').save(edgesimg)
```

Edge detection: 'sigma1', 'sigma2' for two gaussian filter and 'epsilon' for DoG filter are defined.Then the smoothed image is opened and converted to grayscale by using '.convert('L')'. Two different gaussian blur for two different sigma value are applied to the grayscale image. The result is stored as NumPy arrays. DoG is calculated by subtracting the blurred images. Values greater than or equal to 'epsilon' are set to 255, and others to 0. At the end NumPy array is converted to an image and the grayscale image is saved.

```
numColors = 32
image = Image.open(smoothedimg)
#Convert to numpy
imgArray = np.array(image)
#Convert to Lab
labImage = image.convert('LAB')
#Quantize the L
quantizedL = (np.array(labImage.split()[0]) // numColors) * numColors
L, a, b = labImage.split()
quantizedLab = Image.merge( mode: 'LAB',  bands: (Image.fromarray(quantizedL), a, b))
#Convert to RGB
quantizedRGB = quantizedLab.convert('RGB')
quantizedRGB.save(quantizedimg)
```

Quantization: The desired number of colors for quantization 'numColors' is set. Then smoothed image is opened. The image is converted to NumPy array and is converted to Lab color space separately. The L channel is extracted from the Lab image, quantized by rounding down to the nearest multiple of 'numColors', and then scaled back up. The result is stored as 'quantizedL'. Then the quantized L channel is merged with the original a and b channels to create a new Lab image 'quantizedLab'. The quantized Lab image is converted to RGB and saved.

```
edgeImage = Image.open(edgesimg)
quantizedImage = Image.open(quantizedimg)
#Convert to numpy
edgeArray = np.array(edgeImage)
quantizedArray = np.array(quantizedImage)
#Convert to grayscale
quantizedLuminance = np.array(quantizedImage.convert('L'))
invertedEdge = 255 - edgeArray
resultArray = quantizedArray * (invertedEdge / 255.0).reshape((*invertedEdge.shape, 1))
resultImage = Image.fromarray(resultArray.astype('uint8'))
resultImage.save(finalimg)
```

Final: The edges image and the quantized image are opened and they are both converted to NumPy array. The quantized image is converted to grayscale and result is stored as 'quantizedLuminance'. The edges are inverted by subtracting each pixel value from 255, resulting in 'invertedEdge'. The final result array is calculated by multiplying the quantized array with the normalized inverted edges, and the result is reshaped to include an additional dimension. At the end the result array is converted back to image 'resultImage' and saved as ‘finalimg'.

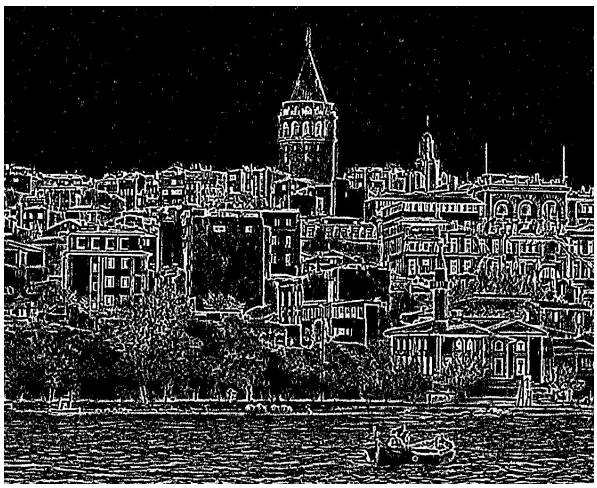
Experiments

Gaussian and median filter for image 1:

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.4, 1.6, 50, ‘gaussian’

Input image

Smoothed image



Edge detection

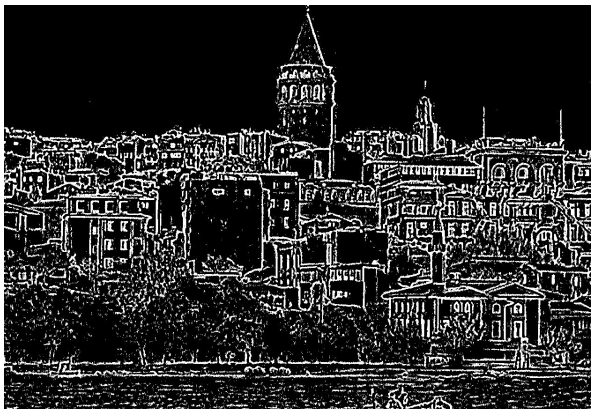
Quantized image

Final image

sigma, sigma1, sigma2, epsilon, filter type = 5, 1.4, 1.6, 50, ‘median’

Input image

Smoothed image



Edge detection

Quantized image

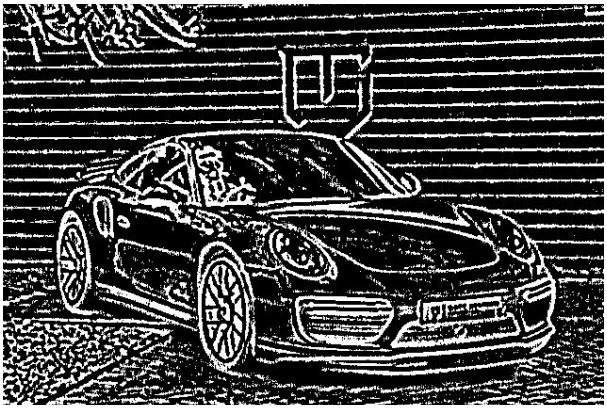
Final image

Gaussian and median filter for image 2:

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.4, 1.6, 50, ‘gaussian’

Input image

Smoothed image



Edge detection

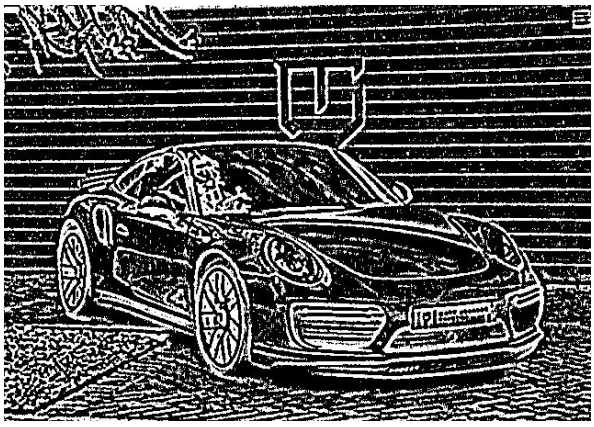
Quantized image

Final image

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.4, 1.6, 50, ‘median’

Input image

Smoothed image



Edge detection

Quantized image

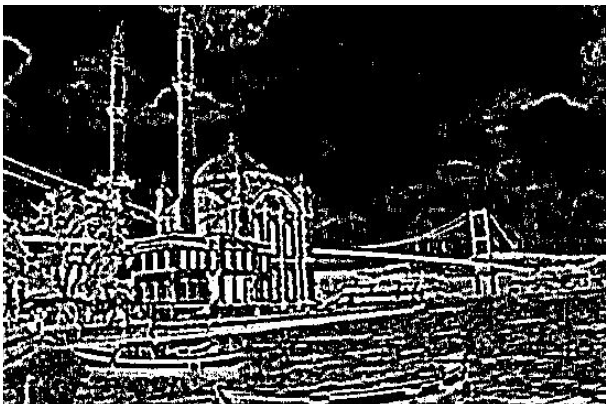
Final image

Gaussian and median filter for image 3:

sigma, sigma1, sigma2, epsilon, filter type = 0.1, 1.4, 1.6, 50, ‘gaussian’

Input image

Smoothed image



Edge detection

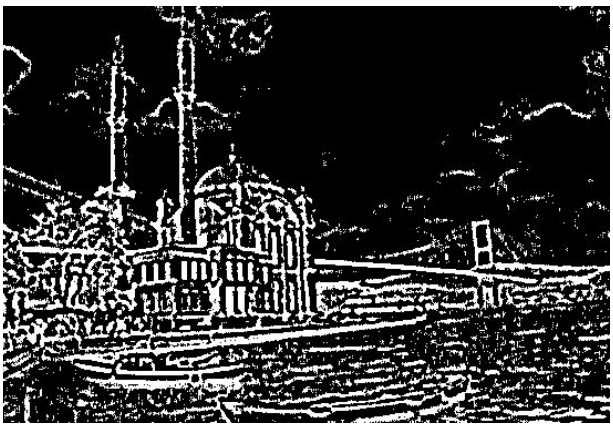
Quantized image

Final image

sigma, sigma1, sigma2, epsilon, filter type = 3, 1.4, 1.6, 50, ‘median’

Input image

Smoothed image



Edge detection

Quantized image

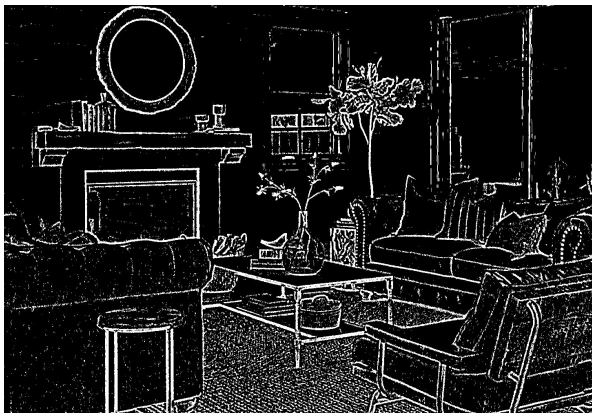
Final image

Gaussian and median filter for image 4:

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.4, 1.6, 50, ‘gaussian’

Input image

Smoothed image



Edge detection

Quantized image

Final image

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.4, 1.6, 50, ‘median’

Input image

Smoothing image

Edge detection

Quantized image

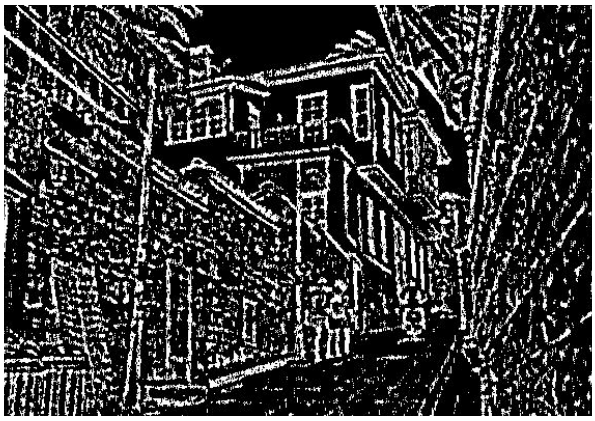
Final image

Gaussian and median filter for image 5:

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.46, 1.54, 50, ‘gaussian’

Input image

Smoothed image



Edge detection

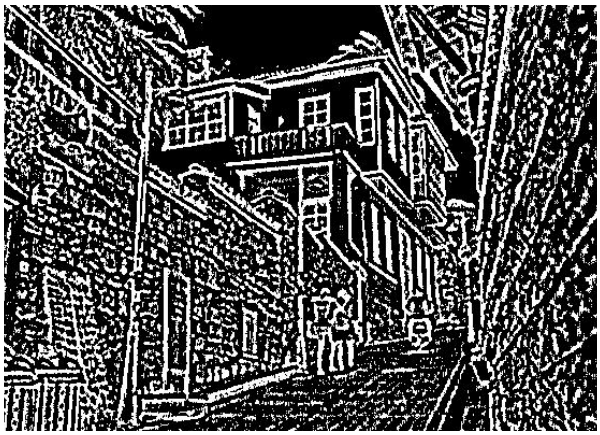
Quantized image

Final image

sigma, sigma1, sigma2, epsilon, filter type = 1, 1.42, 1.58, 50, ‘median’

Input image

Smoothed image



Edge detection

Quantized image

Final image

Conclusion

As a result of these and many other experiments, I observed that the Median filtering method gives better results. Images filtered with the median filter are softer and at the same time clearer, with distinct color transitions. This provides a better cartoon effect.

The ‘sigma’ we use for smoothing usually gives the best result when we set the value '1'. In cases where we use a median filter, sigma of 3 or 5 can give better results, but the value most often used is 1. In our third image, I got the best result with the Gaussian filter at a value of '0.1', which is different than expected. This value made the details in the image more visible.

‘sigma1’ and ‘sigma2’, which we used for edge detection, gave the best results with values of '1.4' and '1.6'. In some cases (such as our 5th image), small changes allowed us to obtain better results. Apart from this, the best results in all experiments were obtained with values of '1.4' and '1.6'.

The ‘epsilon’ value did not make any radical changes in the image, except for extreme values. That's why I used 50, which I found to be the optimum value.