Computational Photography: Homework #1

Due on September 30, 2024

Professor Cho Sunghyun

Shipilova Polina

Task 1

GAUSSIAN FILTERING

The main part of a code is *apply_gaussian_filter* function. It has such parametrs as
$image, kernel\_size, kernel\_sigma, border\_type, separable$.
At first, we need to initialize image and check is it colored or gray-scaled. For colored image we apply filter for each color channel separately. For gray-scaled image we use filter only once.
If the kernel is separable, the 2D filter can be expressed as the outer product of two 1D vectors. This reduces computational complexity since we can apply two 1D convolutions (one for rows and one for columns) instead of a full 2D convolution. A common way to check if a filter is separable is by using Singular Value Decomposition (SVD). If the rank of the matrix representing the 2D kernel is 1, it means the kernel is separable and can be written as the outer product of two vectors.

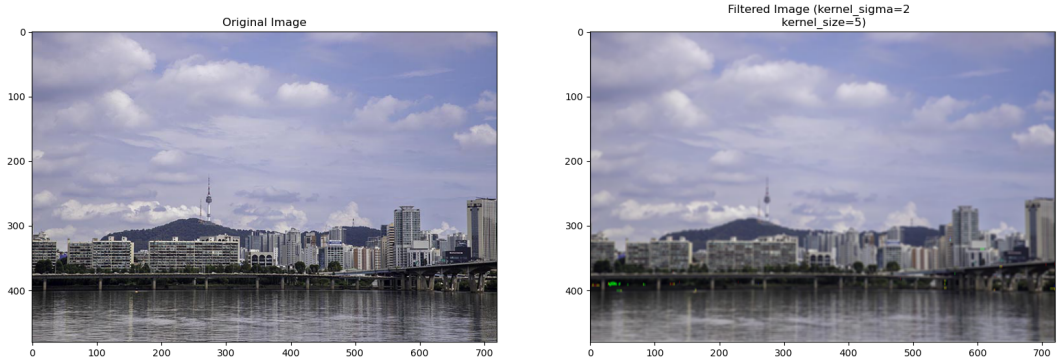
```
def check_separable(kernel):
 u, s, vt = np.linalg.svd(kernel)

# Check the number of non-zero singular values
rank = np.sum(s > 1e-10)

# If the rank is 1, the filter is separable
return rank == 1, u, s, vt
```

If filter can be separated we perform convolution for rows and columns separately. I use function pad from NumPy libruary. It pads an array with chosen numbers of elements and modes for borders could be as follows: 'constant', 'edge', 'linear_ramp', 'maximum', 'mean', 'median', 'minimum', 'reflect', 'symmetric', 'wrap', 'empty'.
For inseparable filter we should use only one convolution between an image array and a filter kernel.

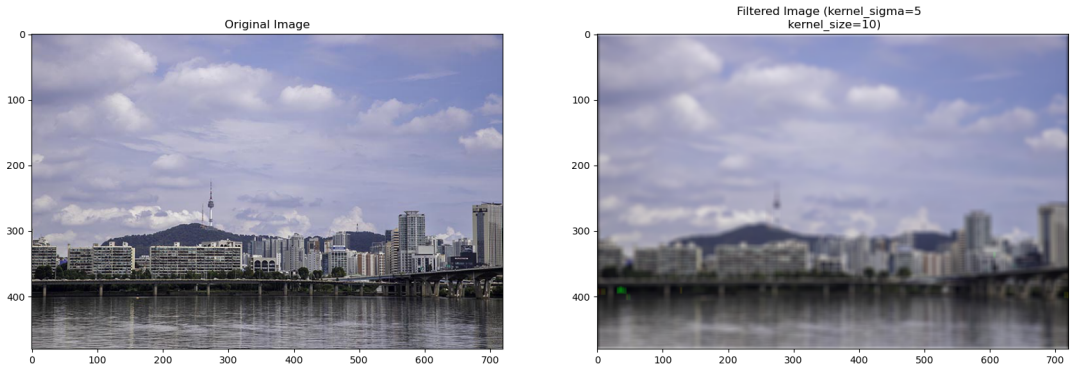
```
def convolve_2d(image, kernel, border_type):
...

# Pad the image to handle borders
padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode=border_type)

# Create an empty output array
result = np.zeros_like(image)

# Perform convolution
for i in range(image_h):
        for j in range(image_w):
                region = padded_image[i:i + kernel_h, j:j + kernel_w]
                result[i, j] = np.sum(region * kernel)
return result
```

some examples

Here is an example for $kernel\_size = 5, sigma = 2, border\_type = "constant"$.



Here is an example for $kernel\_size = 10, sigma = 5, border\_type = "constant"$.
$kernel\_size$ should be set as $2 - 3sigma$ since it is a usual place to truncate Gaussian function.
According to the picture the more is sigma, the more picture is blurred.



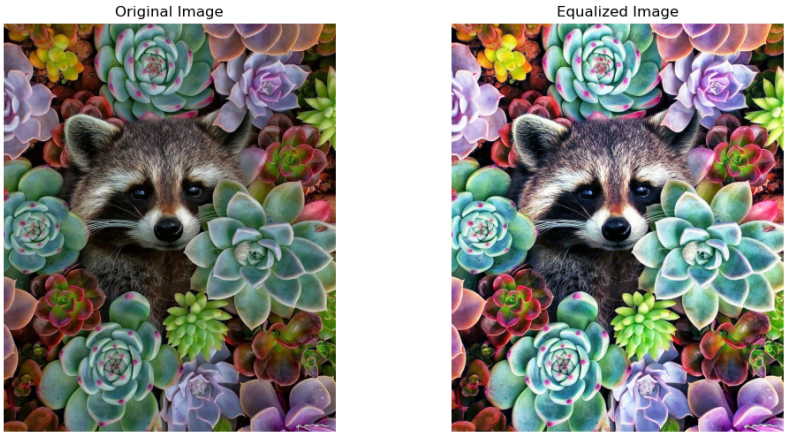
Task 2

HISTOGRAM EQUALIZATION

Before histogram equalization I load an image and divide colored one into tree different samples. After that I apply histogram calculation to flatten image.

```
histogram, _ = np.histogram(image_flat, bins=256, range=(0, 256))
```

And calculate the cumulative distribution function (CDF) for histogram. Based on the result the CDF normalized to increase image contrast. At the end the original pixel values mapped to equalized values.
Here is my example:



With this colorful image histogram equalization works well. Colors became more bright and difference between the darkest and the lightest parts became bigger. Here we can't find a lot of artifacts, hence the original picture haven't overexposed or underexposed parts.

Here we can see how blue sky became more gray, but the rainbow is brighter now.

Histogram equalization works really bad with this image. We can see a dirty sky after filtering. I think that kind of artifacts appears due to the huge amount of white sky without any texture.

On this picture we can see how all the colors became more bright after enhancement. The artifact here is a very dark regions on the ground and a castle.

This image has become more even. In my opinion, the contrast has decreased and the light and dark regions have become more uniform.

With this picture we have pretty similar problem that before with colored one. The white sky simply became a mixture of different shades of gray.

Here you can see how much the contrast has increased.

As a result, I want to say that histogram equalization method increases the contrast of the image well and makes the colors more saturated. However, for more correct image processing, we should not just normalize the histogram over the entire region, but monitor the original image so as not to make the dark areas too black, and the already light areas too light. Therefore, we should shift the histogram to those areas where there is no large peak.