Lab 01

Image Manipulations

Table of Contents

I. Tasks

Task	Description	Process
1	Load an image by reading it from a file	Done
2	Convert a color image into a grayscale image	Done
3	Change the brightness of the color image	Done
4	Change the contrast of the color image	Done
5	Filter a color image/ a grayscale image using average filter operator	Done
6	Filter a color image/ a grayscale image using Gaussian filter operator	Done
7	Show the input and output images	Done

II. User guide

For the best understanding of below `III. Implementaions`, you should read `User_Guide.pdf` first.

III. Implementations

1. Main Function

Implementing main function

```
int main(int argc, char* argv[]) { ... }
```

to read the input from terminal.

Following the instructions, `argc` must be 3 and `argv` must be an array which has these element: `Lab01.exe <image_type> <filename_path>` respectively.

Valid `<filename_path>`: `-rgb2gray`, `-brightness`, `-contrast`, `-avg`, `-gauss`

Using `imread` function to read the image from local device.

```
Mat image = imread(argv[2], IMREAD_UNCHANGED);
if (image.empty()) {
    return -1;
}
```

After reading the image, I will verify that the image is read or not, if `image.empty()` was true, the program exited.

2. Converting To Grayscale Image

Implementation: use `cvtColor` function from OpenCV library:

```
cvtColor(image, grayImage, COLOR_RGB2GRAY);
```

- `image`: original image (input image).
- `grayImage`: store output image.
- `COLOR_RGB2GRAY`: a constant representing the specific color to grayscale.

Output:

Figure 1: Grayscale Image

3. Color Image's Brightness

Implement 2 functions `brightnessModify` and `brightnessDisplay`

```
void brightnessDisplay(Mat image) {
    int initBright = 50, maxBright = 100;
    ...
    createTrackbar("Brightness", OUTPUT_IMAGE IMAGE_TYPE_2, &initBright,
maxBright, brightnessModify, (void*)&image);
    ...
}
```

- Init the brightness for first output image is `50` and maximum is `100`
- `createTrackbar` is one of the openCV function that when user interact the brightness via trackbar, this function will modify the image immediately by `brightnessModify` function:

```
void brightnessModify(int brightness, void* userData) {
    ...
    image->convertTo(brightnessImage, -1, 1.0, brightness);
    ...
}
```

- `brightnessImage` is destination image which is modified the brightness
- `-1` gives the same as the data type of the input image such as number of channels,...

- `1.0`: pixel value of output image will be linearly scaled the same as input (retain its original brightness)
- `brightness`: adjust the brightness of the image by adding a specified value to each pixel

Output:

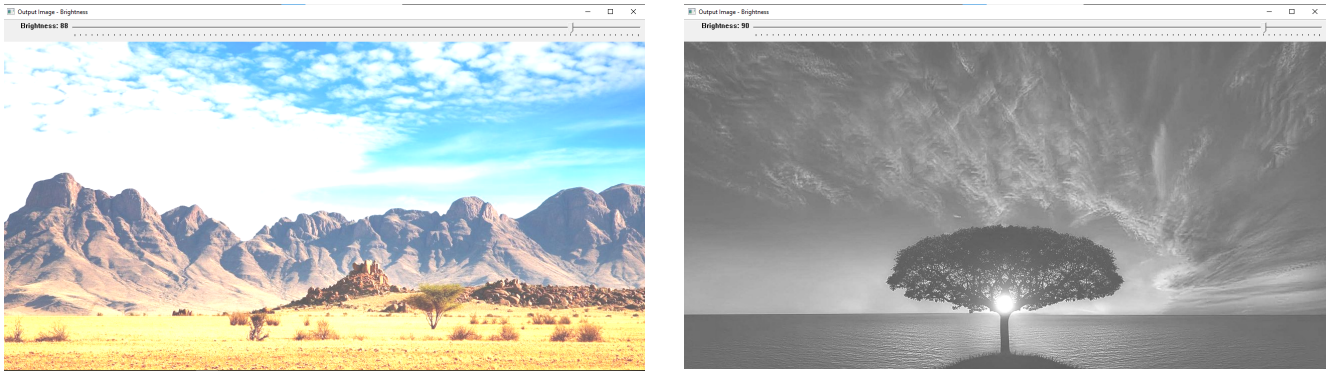

Figure 2: Brightness Image

4. Color Image's Contrast

Implement 2 functions `contrastModify` and `contrastDisplay`

```
void contrastDisplay(Mat image) {
    int initContrast = 10, maxContrast = 100;
  ...
    createTrackbar("Contrast", OUTPUT_IMAGE IMAGE_TYPE_3, &initContrast,
maxContrast, contrastModify, (void*)&image);
  ...
}
```

- Init the contrast for first output image is `10` and maximum is `100`
- `createTrackbar` is one of the openCV function that when user interact the contrast via trackbar, this function will modify the image immediately by `contrastModify` function:

```
void brightnessModify(int brightness, void* userData) {
    ...
    image->convertTo(contrastImage, -1, contrast / 50, 0);
    ...
}
```

- Similar to `section 3` implementation, instead of changing the brightness from the function `convertTo()`, I will modify the contrast which is scaled by contrast / 50.
- I set `contrast / 50` instead of `contrast` because:
    - It's hard to detect the contrast by eyes.
    - After testing different scaling factors, I think dividing by 50 is the best range for perceptual adjustment.

Output:

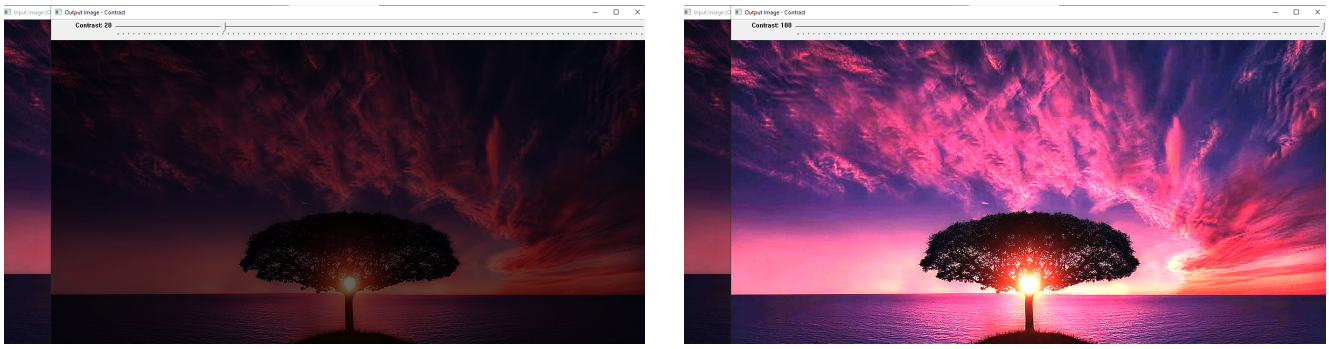

Figure 3: Contrast Image

5. Filter Average (Mean)

Filter Average (Mean) replaces each pixel value with the average value of the pixel and its neightbors within a specified kernel size. There are two functions: `filterImageAvg` and `filterDisplayAvg`

- The process of `filterImageAvg`:
  - If the kernelSize is set to 0, the function returns early without applying any filtering.
  - If the image has only 1 channel (single channel), a 2D kernel of `ones` is created, and each element is divided by the size of the kernel to get the average value. This kernel is then applied to the image using the `cv::filter2D` function.
  - Otherwise (three channels - color image), the image is split into its color channels, and the same average filter kernel is applied to each channel separately. After filtering, the channels are merged back into a color image.

```
void filterDisplayAvg(Mat image) {
  int initKernelSize = 1, maxKernelSize = 10;
  ...
  createTrackbar("KernelSize", OUTPUT_IMAGE IMAGE_TYPE_4, &initKernelSize,
maxKernelSize, filterImageAvg, (void*)&image);
  ...
}
```

I choose to set the maximum kernel size to 10 because my laptop is not powerful enough to handle larger kernel sizes.

Output:

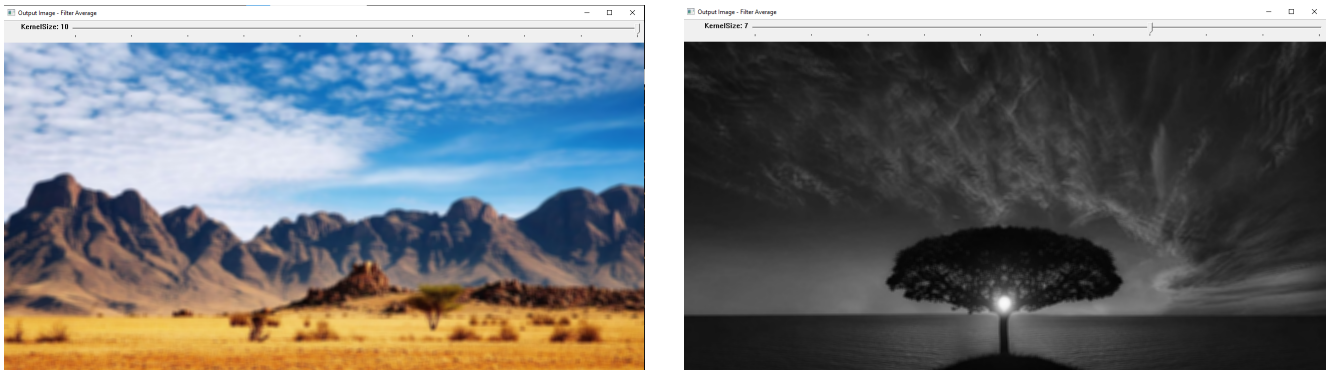

Figure 4: Average (Mean) Filter

6. Filter Gauss

Gaussian filters are widely used in image processing for tasks such as blurring and noise reduction. The code consists of two functions: `filterImageGauss` and `filterDisplayGauss`. Using `cv::GaussianBlur` function to determine output image.

```
void filterImageGauss(int kernelSize, void* userData) {
  ...
  GaussianBlur(*image, filterImage, Size(kernelSize * 2 + 1, kernelSize * 2 + 1),
0, 0);
  ...
}
```

- For `Size(kernelSize * 2 + 1, kernelSize * 2 + 1)`, this defines the size of Gaussian kernel for filtering. In this cases, kernelSize stands for the radius of the Gaussian. The entire kernel size is calculated as (kernelSize * 2 + 1) in both the width and height dimensions. +1 operator is used to ensure that the kernel size is an odd number to ensure that the center pixel of the kernel falls on the pixel being processed.
- Last two parameters `0` determied automatically based on the kernel size which is the standard deviation values for the Gaussian distribution in x and y directions respectively.

Output:

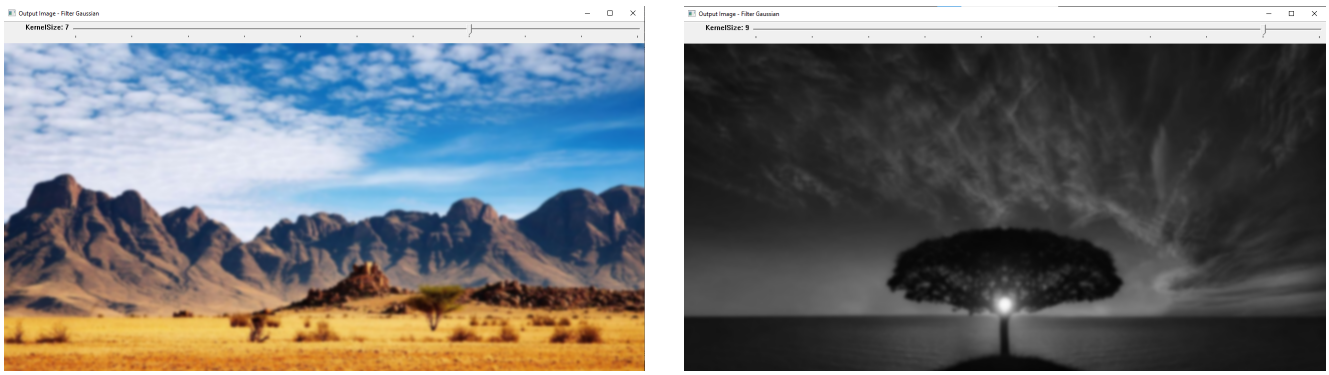

Figure 5: Gaussian Filter

7. Show Both Input And Output Images

Using 2 functions `imshow` to illustrate both original image and output image for comparison.

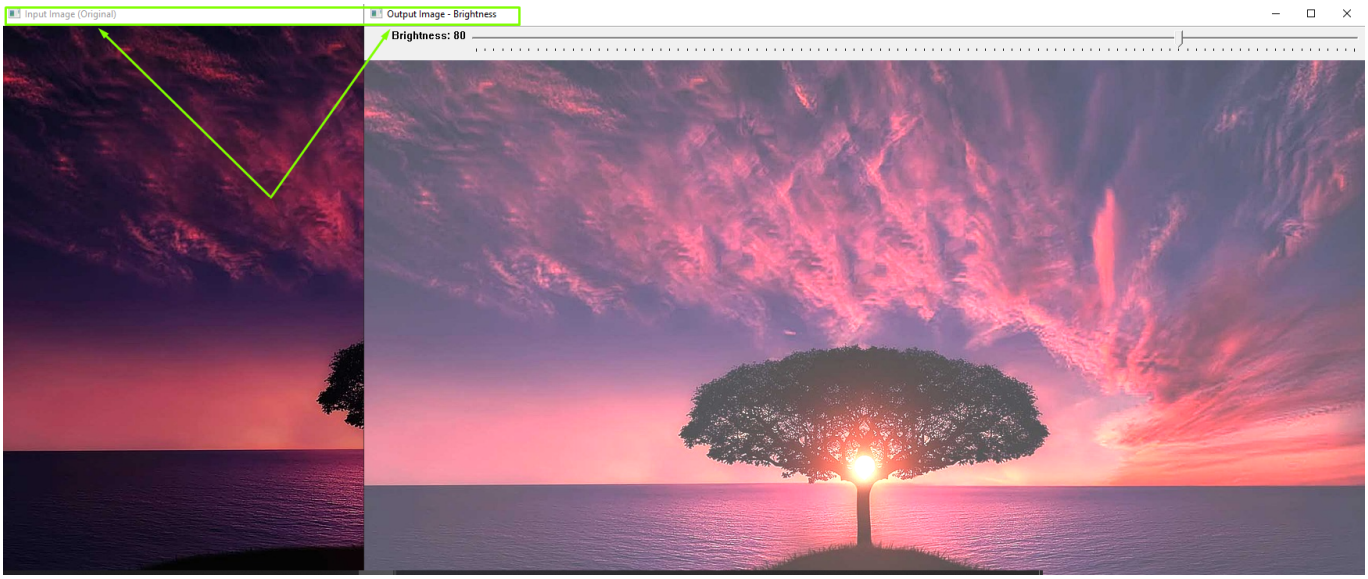

Figure 6: Show Both Input & Output