Homework 1 Questions

Instructions

- Compile and read through the included MATLAB tutorial.
- 2 questions.
- Include code.
- Feel free to include images or equations.
- Please make this document anonymous.
- **Please use only the space provided and keep the page breaks.** Please do not make new pages, nor remove pages. The document is a template to help grading.
- If you really need extra space, please use new pages at the end of the document and refer us to it in your answers.

Submission

- Please zip your folder with **hw1_student id_name.zip** (ex: hw1_20201234_Peter.zip)
- Submit your homework to KLMS.
- An assignment after its original due date will be degraded from the marked credit per day: e.g., A will be downgraded to B for one-day delayed submission.

Questions

Q1: We wish to set all pixels that have a brightness of 10 or less to 0, to remove sensor noise. However, our code is slow when run on a database with 1000 grayscale images.

Image: grizzlypeakg.png

```
A = imread('grizzlypeakg.png');
[m1,n1] = size( A );
for i=1:m1
    for j=1:n1
        if A(i,j) <= 10
            A(i,j) = 0;
        end
    end
end
```

Q1.1: How could we speed it up?

A1.1: Your answer here.

By using both vectorization and logical indexing, we can speed the code more efficiently. The part of A which its brightness is 10 or less could be specified as an logical array, let us call it B. Logical array such as B functions more efficiently than using for loops. Plus, using B as a factor of the function varying the brightness, it would be more efficient using vectorization effect.

Q1.2: What factor speedup would we receive over 1000 images? Please measure it.

Ignore file loading; assume all images are equal resolution; don't assume that the time taken for one image $\times 1000$ will equal 1000 image computations, as single short tasks on multitasking computers often take variable time.

A1.2: Your answer here.

Before speeding up, using the code written below, the result of elapsed time was calculated as 34.8544587s as an average of running the program 10 times.

```
tic;
for p=1:1000

    A = imread("grizzlypeakg.png");
    [m1, n1] = size(A);

    for i=1:m1
        for j=1:n1
            if A(i,j) <= 10
                A(i,j) = 0;
            end
        end
    end
end
toc;
```

To speed up, I've written the code below. The elapsed time is 16.4487209s as an average of running the program 10 times.

```
tic;
    for p=1:1000
        A = imread("grizzlypeakg.png");
        [m1, n1] = size(A);

        B = A <= 10;
        A(B)=0;
    end
toc;
```

Q1.3: How might a speeded-up version change for color images? Please measure it.

Image: grizzlypeak.jpg

A1.3: Your answer here.

Before speeding up, using the code written below, the result of elapsed time was calculated as 7.6954872s as an average of running the program 10 times.

```
tic;
A = imread("grizzlypeak.jpg");
[m1, n1] = size(A);
for i=1:m1
    for j=1:n1
        if A(i,j) <= 10
            A(i,j) = 0;
        end
    end
end
toc;
```

To speed up, I've written the code below. The elapsed time is 0.0734264s as an average of running the program 10 times.

```
tic;
    A = imread("grizzlypeak.jpg");
    [m1, n1] = size(A);

    B = A <= 10;
    A(B)=0;
toc;
```

Q2: We wish to reduce the brightness of an image but, when trying to visualize the result, all we sees is white with some weird "corruption" of color patches.

Image: gigi.jpg

```
I = double( imread('gigi.jpg') );
I = I - 20;
imshow( I );
```

Q2.1: What is incorrect with this approach? How can it be fixed while maintaining the same amount of brightness reduction?

A2.1: Your answer here.

imshow function requires single or double values between 0 1 to show proper image. But using single or double function, it converts the image into a range of 0-255. It could be fixed by using im2single or im2double functions instead of single or double functions, which im2double would suit this case.

Q2.2: Where did the original corruption come from? Which specific values in the original image did it represent?

A2.2: Your answer here.

Since when we convert image using double function, it converts each RGB pixels into 0-255 range, what we can see as the original corruption is the pixels of original image with its brightness in a range of 20-21 since imshow function can display double values between 0 and 1.

So, in specific, it would be values between 20-21 if the original image is converted as a 0-255 format.