Homework 1 Questions

Instructions

- Compile and read through the included Python 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

```
import cv2
import numpy as np
A = cv2.imread('grizzlypeakg.png',0)
m1, n1 = A.shape
for i in range(m1):
    for j in range(n1):
        if A[i,j] <= 10:
            A[i,j] = 0
```

Q1.1: How could we speed it up?

A1.1: We can speed it up by using logical indexing:

```
import cv2
import numpy as np
A = cv2.imread('grizzlypeakg.png', 0)
m1, n1 = A.shape
B = A <= 10
A[B] = 0
cv2.imwrite('result_1,1.png', A)
```

This results in the following picture:

Figure 1: Result after running the speedup code

Q1.2: What factor speedup would we receive over 1000 images? Write the python code for both original version in Q1 and your speedup version. Run the python code and provide the real measurement. (You may repeat operation for 1000 times on the grizzlypeakg.png)

Ignore file loading; Assume all images are equal resolution; Don't simply 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: Reducing the loops to 100 (Due to my performance problems).
The provided code needs 1138.222850561142 seconds to finish.
My speedup code needs 0.6117448806762695 seconds to finish.
So the speedup version takes just 0,05374562% of the original time and leads to an performance increase by 1.860Here you see the adjusted provided code:

```
import cv2
import numpy as np
import time
B = cv2.imread('grizzlypeakg.png',0)
start = time.time()
for n in range(100):
    A = B
    m1, n1 = A.shape
    for i in range(m1):
        for j in range(n1):
            if A[i,j] <= 10:
                A[i,j] = 0
estimated = time.time() - start
print(estimated)
```

And here my solution of the previous task adjusted by time measurement:

```
import cv2
import numpy as np
import time
original = cv2.imread('grizzlypeakg.png')
start = time.time()
for n in range(100):
    A = original
    B = A <= 10
    A[B] = 0
estimated = time.time() - start
print(estimated)
cv2.imwrite('task 1.2.png', A)
```

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

Image: grizzlypeak.jpg

A1.3: As written in the previous task, I reduced the iterations to 100, due to performance problems of my computer (with the provided code).
To keep the times comparable, I gonna use the same number of iterations in this task. I could use the same code as in task 1.2:

```
import cv2
import numpy as np
import time
original = cv2.imread('grizzlypeakg.jpg')
start = time.time()
for n in range(100):
    A = original
    B = A <= 10
    A[B] = 0
estimated = time.time() - start
print(estimated)
cv2.imwrite('task 1.3.jpg', A)
```

The speedup version for color images need 1.0416607856750488 seconds. This is an increase of 1.7!
The resulting image is:

Figure 2: Result after running the speedup code

Q2: We wish to reduce the brightness of an image but, when trying to visualize the result, we see a brightness-reduced scene with some weird "corruption" of color patches.

Image: gigi.jpg

```
import cv2
import numpy as np
I = cv2.imread('gigi.jpg').astype(np.uint8)
I = I - 40
cv2.imwrite('result.png', I)
```

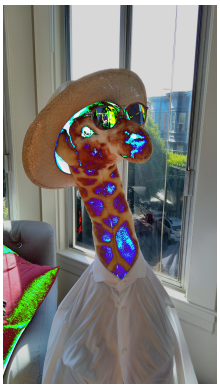
Q2.1: What is incorrect with this approach? How can it be fixed while maintaining the same amount of brightness reduction?

A2.1: The problem of the provided code is, that we work with unsigned integers (they can't get negative). Values < 40 will cause a overflow and result in an very high value! To avoid this overflow, we set all overflowed values to zero.

```
import cv2
import numpy as np
I = cv2.imread('gigi.jpg').astype(np.uint8)
I = I - 40
I[I>215] = int(0)
cv2.imwrite('result.png', I)
```

Since a uint8 max number is 255 we know that all values below 215 are caused by overflow. Setting this values to zero gives the desired result as seen in c):

(a) Original



(b) Wrong brightness

(c) Adjusted brightness

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

A2.2: Subtracting integers can cause an overflow. Since we are working with unsigned integers, the overflow leads from zero to the max value.
For instance: Consider 0 as a uint8. Subtracting 1 would cause a overflow and leads to 255 (max number represented by an uint8).
In our case, we subtract 40 from every value in our matrix data. So all values smaller then 40 will result in an overflow and appear as a corruption.