Lab 1: Image Transformation

Course: CS195 - Computer Vision (Spring 2022)
Instructor: Md Alimoor Reza, Assistant Professor of Computer Science, Drake University
Due: Tuesday February 15, 11:59PM

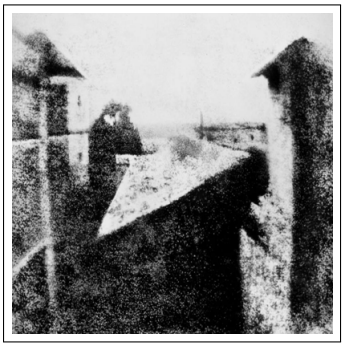
Getting up to speed with **`Pillow`**

Python does not have native support for image processing so we encourage students to make use of the **`Pillow Library`**, supported at https://colab.research.google.com/. You could also install this library in your machines. If you wish to install it into your own system, simply do `pip install Pillow`. Below we show some sample code that loads a grayscale image, checks some of its properties, and creates a new color image based on its pixel values.

```
from PIL import Image
from PIL import ImageFilter

# random number generator
import random

# for drawing text over image
from PIL import ImageDraw

if __name__ == '__main__':
    # load the image
    im = Image.open('first_photograph.png')

    #Check it's  width ,  height, and number of  color channels
    print('Image is %s pixels  wide. '%im.width)
    print('Image is %s  pixels high. '%im.height)
    print('Image mode  is %s.'% im.mode) #(8-bit pixels, black and white)

    #pixels are accessed via a (X,Y) tuple
    print('Pixel value is %s '% im.getpixel((10 ,10)))
    #pixels can be modified by specifying the coordinate and RGB value
    im.putpixel((10 ,10), 20)
    print('New pixel value is %s'%im.getpixel((10 ,10)))


# Create a new blank color image the same  size as the input
colorim = Image.new('RGB', (im.width , im.height), color =0)
# Loops over the new color image and
# fills in brighter area that was white first grayscale image we loaded with red colors!
# Basically we transformed the gray colored first ever photographh into a red colored one!
for x in range(im.width):
    for y in range(im.height):
        grayscale_val = im.getpixel((x,y))
        if ( grayscale_val >= 190):
            colorim.putpixel((x,y), (grayscale_val,0,0))
        else:
            colorim.putpixel((x,y), (grayscale_val, grayscale_val, grayscale_val))

colorim.show()
colorim.save('output.png')

# adding text on top of the image at a random position
```

```
colorim_with_text = ImageDraw.Draw(colorim)
# generate a random X-coordinate and a random Y-coordinate
text_x_coord = random.randint(0 , colorim.width//2)
text_y_coord = random.randint(0 , colorim.height)
colorim_with_text.text((text_x_coord , text_y_coord),
                       'View from the Window at Le Gras',(255,255,255))
colorim.save('output_with_text.png')
```

We will be manipulating a very famous photo, which has a little bit of history behind it. We will be manipulating the "**first ever photo captured by a camera**" back in 1827 (or 1826) by French inventor Joseph Nicephore Niepce. It shows the outside view from his window. This photograph is famously known as "**View from the Window at Le Gras**".



Input

Output

Output (with text)

This example should provide you with everything you will need to build on for the projects and while it's not the most impressive thing in the world ... **it's something**.

This week's Lab

For this week, we want to get everyone comfortable in the programming environment and to get you used to manipulate pixel values of images. There will be two tasks. You will be doing various per-pixel transformation to some images and see how they turn out.

Histogram Generation

Before you start applying the per-pixel transformation, it would be a good practice to visualize the histogram of the image first. Write a few lines of python code to visualize the histogram first.

Task 1: Whitening

You will be adjusting the contrast of the image. Your goal is to transform the image so that the resulting image has a zero mean and unit variance. Denote the image as $I(.)$ which is a 2D array of pixel values. Its width and height are N and M pixels respectively. Also $I(x, y)$ denotes the pixel value at 2D location (x, y). You can compute the mean and unit variance of the gray-scale image $I(.)$ as follows:

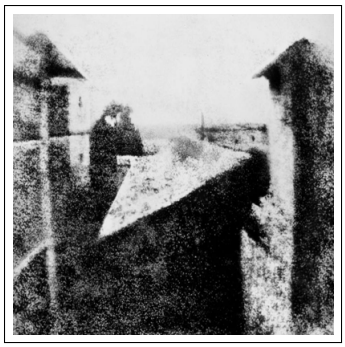
$$\mu = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} I(x, y)}{N * M} \tag{1}$$

$$\sigma = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} (I(x, y) - \mu)^2}{N * M} \tag{2}$$

Now, you can transform each pixel value separately using the above two computed statistics μ and σ as follows:

$$I^{'}(x,y) = \frac{I(x,y) - \mu}{\sigma} \tag{3}$$

Apply this *Whitening Transformation* on the provided input image to see how it affects. The following figures show the effect of applying *Whitening Transformation* on the first-ever photograph – "**View from Window at Le Gras**".



Input

'Whitening' Transformed

Task 2: Histogram Equalization

Let's try another type of contrast transformation called *Histogram Equalization* on the input image. You may need to do it in multiple steps.

Step 1: you need to generate the histogram of intensity values. The simplest way to find the histogram of intensity values is to make use of the **`Pillow Library`** functionalities (HINT: There is a function called 'histogram()' in Pillow). Alternatively, if you want, you could write few lines of python code and build your own histogram of intensity values. There will be at most 255 intensity values. Let's denote $hist_b$ is 1D vector denoting the histogram of intensity values. b denotes a particular bin, and its value ranges from 0 to 255. Now, count how many pixels in image $I(.)$ have the intensity value equal to b. Put that total count into the respective bin location, i.e., $hist_b$. You can do it for all the intensity values one after another, starting from $b = 0$ up until $b = 255$.

Step 2: Now you need to normalize the histogram $hist$ such that the sum of this new histogram is equal to 255. Denote this new histogram as $hist^{'}$. For each bin b, you can compute it as follows:

$$hist_b^{'} = \frac{hist_b}{N * M} \tag{4}$$

$$hist_b^{'} = hist_b^{'} * 255 \tag{5}$$

Recall that the image width and height are N and M pixels respectively.

Step 3: Compute the cumulative sum of the $hist^{'}$. Denote this histogram as $histcum$.

Step 4: Use the cumulative histogram $histcum$ as a lookup table to transform the value of each pixel location as follows:

$$I^{'}(x,y) = histcum(I(x,y)) \tag{6}$$

where $I^{'}(x,y)$ denotes the histogram equalized image value at pixel location (x,y). Apply your newly implemented *Histogram Equalization* on the provided input image to see how it affects. The following figures show the result on another input image called "**Himalaya**".

Input

Histogram Equalized

What to turn in:

Please submit a zip file containing your source code and the image(s) that you used for this lab. If you used any external libraries other than Pillow, please include a README file explaining why you did so. Again, if you have any questions, please ask or email me.