Digital Image Color Manipulation

CST 205

Review: Grayscale images (mode = `'L'`)

- Our *Mona Lisa* was a grayscale image. 
  - It used one channel with values between 0 and 255.

  - We used one byte for each channel.
    - 1 byte = 8 bits and represents 256 values (0 — 255)

Review: RGB images

- An RGB image, like the one used in the homework, uses three color channels.

- **Note:** In reality, often 4 bytes are used.

    - The last byte represents the alpha channel.

        - The alpha channel is transparency (or α)

    - These images are referred to as RGBA.

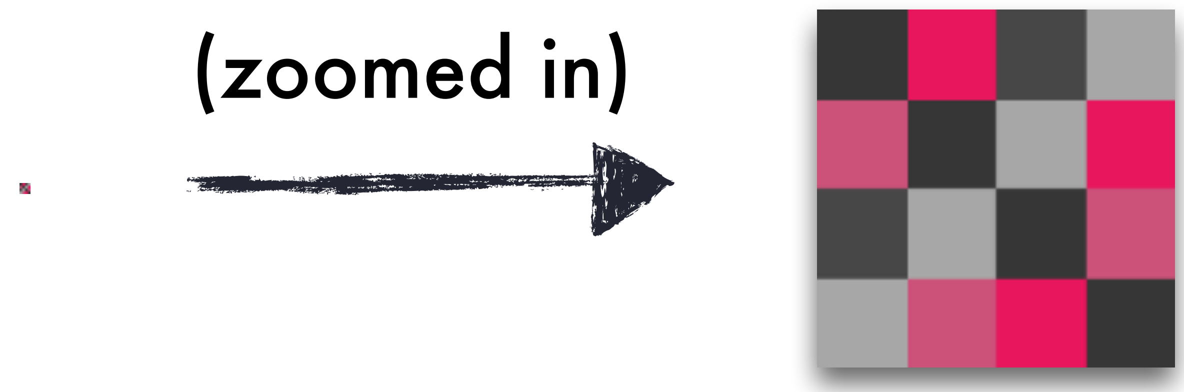
Review: `getpixel()` and `putpixel()`

Both methods are used on Pillow `Image` objects.

Pillow's `getdata()` method

- If we don't need to access specific pixel coordinates, we can get a **flat** list of every pixel in the image.
    - To do so, we use Pillow's `getdata()` method

`getdata()` example




```
from PIL import Image
import pprint

im = Image.open('images/colorbox.png')

pixel_list = im.getdata()

print(f'The number of pixels is {len(pixel_list)}.')

pprint.pprint(list(pixel_list))
```

Review: Python lists

- Remember, lists preserve order
- Using `getdata()`, we can alter pixel values in the list as long as we don't start rearranging the order.

Decrease red in a picture

We can decrease the red intensity by 50%.

```
from PIL import Image

im = Image.open('images/meadow-flowers-a.jpg')

def decrease_red(picture):
    new_list = []
    for p in picture.getdata():
        temp = (int(p[0]*.5), p[1], p[2])
        new_list.append(temp)
    picture.putdata(new_list)
    picture.save('images/new_pic.jpg')

decrease_red(im)
```

Python's `map()` function

- `map()` *maps* functions over sequences.
- `map()` takes as input a function and a sequence.
  - `map()` applies the function to each input in the sequences and returns the result.

```
def hello(someone):
    return f'Hello, {someone}!'

map(hello, ['Janet', 'Jason'])
```

Decrease red using `map()`

```
from PIL import Image

im = Image.open('images/meadow-flowers-a.jpg')

def map_red(pixel):
    return (int(pixel[0]*.5), pixel[1], pixel[2])

new_list = map(map_red, im.getdata())

im.putdata(list(new_list))

im.save('images/new.jpg')
```

Whoa, where’d the loop go?

Lambda expressions (λ)

Nameless functions that provide functionality right where you need it.

```
lambda arg1, arg2, ..., argN : expression using arguments
```

```python
from PIL import Image

im = Image.open('images/meadow-flowers-a.jpg')

new_list = map(lambda a : (int(a[0]*.5), a[1], a[2]),  im.getdata())

im.putdata(list(new_list))

im.save('images/new.jpg')
```

List comprehension

- Python also provides **lists comprehensions** as a concise way to create lists *from* lists.

- Based on set-builder notation from mathematics:

$$S = \{.5 * x \mid x \in \{2, 4, 6, 8\}, x < 6\}$$

In Python, we could write this as:

```
s = [ x//2 for x in range(2,10,2) if x < 6]
```

List comprehension (cont.)

- Brackets `[]` containing an **expression** followed by a `for` clause, then zero or more `for` or `if` clauses.

- We can rewrite part of slide 8 as:

```
new_list = [(int(a[0]*.5), a[1], a[2]) for a in im.getdata()]
```

- **Note:** if-else in list comprehension written as:

```
new_s = [ x//2 if x < 6 else x*2 for x in range(2,10,2)]
```

How would you approach increasing the red channel by, say, 1.3?

Negative

- Say we have a red channel with intensity 10. That's very little red. The negative would be *lots* of red.
  - We can get this value, in this example, by:
    255 - 10 = 245
    - We use this same idea to negate each color component, which in turn negates the whole picture.

Code for negative

```
from PIL import Image
im = Image.open('images/meadow-flowers-a.jpg')

def negative_image(pixel):
    return tuple(map(lambda a : 255 - a, pixel))

negative_list = map( negative_image, im.getdata() )


"""
or with list comprehension,
neg_list = [(255-p[0], 255-p[1], 255-p[2]) for p in im.getdata()]

"""


im.putdata(list(negative_list))

im.save('images/negative.jpg')
```

Converting to grayscale

- If red = green = blue, we get gray
- **Luminance** — value representing the amount of darkness of the color. (Sometimes referred to as the perception of brightness.)
- One way to get the luminance is by averaging the three channels. For a single color, C_1:

$$\frac{C_{1,R} + C_{1,G} + C_{1,B}}{3}$$

Grayscale code

```
from PIL import Image
im = Image.open('images/friends.jpg')

new_list = map( lambda a : (int((a[0]+a[1]+a[2])/3),) * 3,  im.getdata() )

"""
or,
new_list = [ ((a[0]+a[1]+a[2])//3,)*3 for a in im.getdata() ]

"""

im.putdata(list(new_list))
im.save('images/gray2.jpg')
```

Lossy transformations

- Once we have converted an image to grayscale, we cannot get back the color version.
  - During the conversion process, we have *lost* information
  - We no longer know what the ratios were between the color channels
  - An example of a **lossy** transformation

A better grayscale

- In reality, we don’t perceive red, green, blue as *equal* in their amount of luminance.
  - We tend to see blue as “darker” and red as “brighter”, even if the same amount of light is coming off of each.

Grayscale based on color perception

```
from PIL import Image

im = Image.open('images/friends.jpg')

def lum_image(p):
    new_red = int(p[0] * 0.299)
    new_green = int(p[1] * 0.587)
    new_blue = int(p[2] * 0.114)
    luminance = new_red + new_green + new_blue
    return (luminance,) * 3

new_list = map( lum_image,  im.getdata() )

im.putdata(list(new_list))

im.save('images/new_gray.jpg')
```

Treating pixels differently

- Perhaps we want to search through our image for pixels of a certain shade and manipulate them.
- Say, for example, we have an individual with brown hair and we want to make their hair look red.
- Here is an idea:
    1. Estimate the shade of brown we want.
    2. Look for pixels that are *close* to that color (within a threshold).
    3. Increase the redness of those pixels (by say 50%)

Color distance

- When comparing colors, we talk about the *distance* between colors.

- No perfect way, here we discuss **Euclidean color distance**

- The Euclidean distance between two colors, C_1 and C_2 is:

$$\sqrt{(C_{1,R} - C_{2,R})^2 + (C_{1,G} - C_{2,G})^2 + (C_{1,B} - C_{2,B})^2}$$

Distance function

```
def color_distance(c1, c2):
    r_diff = (c1[0] - c2[0])**2
    g_diff = (c1[1] - c2[1])**2
    b_diff = (c1[2] - c2[2])**2
    return (r_diff + g_diff + b_diff)**(1/2)
```

Distance function using `math` module and loop

```python
import math

def color_distance2(c1, c2):
    val = 0
    for i in range(3):
        val += math.pow((c1[i]-c2[i]), 2)

    return math.sqrt(val)
```

Distance example

```
im = Image.open("images/stop.jpg")

color_to_change = (58, 71, 36)

new_list = [ (int(p[0]*1.5), int(p[1]*.5), p[2]) for p
    in im.getdata()
    if color_distance2(p, color_to_change) ]

im.putdata(new_list)
im.save("images/changed.png")
```

International Commision on Illumination (CIE)

- *Perceived* difference in color is much more complicated than what Euclidean color distance provides.

- CIE began tackling the issue of perceived color distance in 1976. They introduced dE76 (or CIE76).

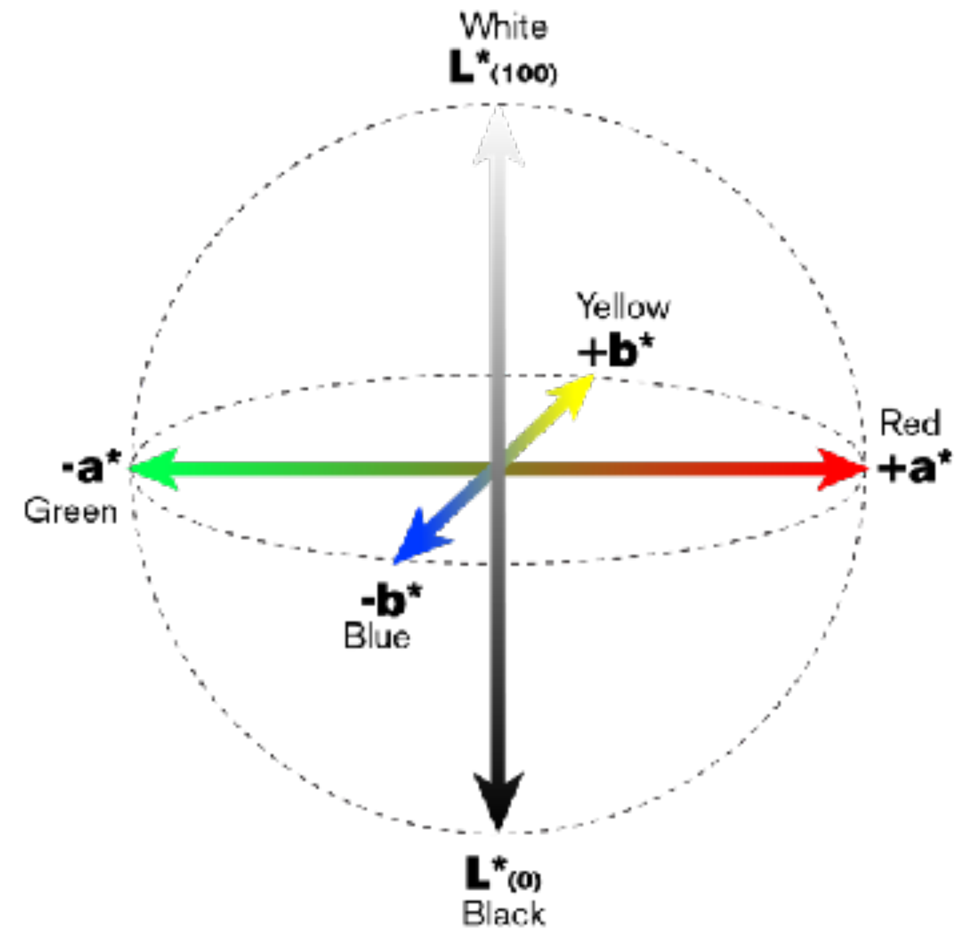
    - **d** stands for Δ or delta (i.e. change)

    - **E** stands for empfindung (German for sensation)

    - **76** stands for the year 1976

CIE L*a*b*

- dE76 introduced a new color space called CIE L*a*b*
  - L* = lightness
  - a* = red to green
  - b* = yellow to blue



$$\Delta E^*_{ab} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2}$$

Python `colormath` module

- **dE76** was superseded by much more complicated formulas **dE94** and then **dE2000** (more info here).
- Luckily, we can install the Python `colormath` module that provides all CIE dE functions (and much more).

```
from colormath.color_objects import sRGBColor, LabColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000

color1_rgb = sRGBColor(255, 0, 0, True);
color2_rgb = sRGBColor(0, 0, 255, True);

color1_lab = convert_color(color1_rgb, LabColor);
color2_lab = convert_color(color2_rgb, LabColor);

delta_e = delta_e_cie2000(color1_lab, color2_lab);

print(f'The difference is {delta_e}.')
```