elogod-contactsheet

August 15, 2022

1 Assignment 1: Building a Better Contact Sheet

In the lectures for this week you were shown how to make a contact sheet for digital photographers, and how you can take one image and create nine different variants based on the brightness of that image. In this assignment you are going to change the colors of the image, creating variations based on a single photo. There are many complex ways to change a photograph using variations, such as changing a black and white image to either "cool" variants, which have light purple and blues in them, or "warm" variants, which have touches of yellow and may look sepia toned. In this assignment, you'll be just changing the image one color channel at a time

Your assignment is to learn how to take the stub code provided in the lecture (cleaned up below), and generate the following output image:

From the image you can see there are two parameters which are being varied for each subimage. First, the rows are changed by color channel, where the top is the red channel, the middle is the green channel, and the bottom is the blue channel. Wait, why don't the colors look more red, green, and blue, in that order? Because the change you to be making is the ratio, or intensity, or that channel, in relationship to the other channels. We're going to use three different intensities, 0.1 (reduce the channel a lot), 0.5 (reduce the channel in half), and 0.9 (reduce the channel only a little bit).

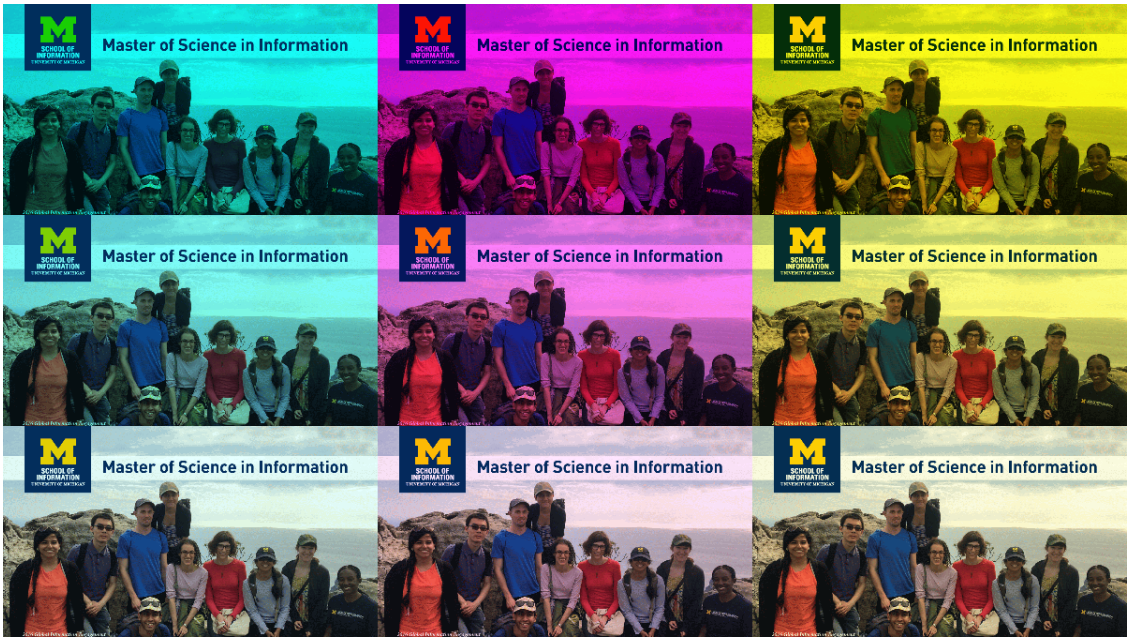
For instance, a pixel represented as (200, 100, 50) is a sort of burnt orange color. So the top row of changes would create three alternative pixels, varying the first channel (red). one at (20, 100, 50), one at (100, 100, 50), and one at (180, 100, 50). The next row would vary the second channel (blue), and would create pixels of color values (200, 10, 50), (200, 50, 50) and (200, 90, 50).

Note: A font is included for your usage if you would like! It's located in the file `readonly/fanwood-webfont.ttf`

Need some hints? Use them sparingly, see how much you can get done on your own first! The sample code given in the class has been cleaned up below, you might want to start from that.

In [19]:
```
import PIL
from PIL import Image
#from PIL import ImageEnhance
from PIL import ImageFont
from PIL import ImageDraw

# read image and convert to RGB
image=Image.open("readonly/msi_recruitment.gif")
image=image.convert('RGB')

# adding fonts
text_font = ImageFont.truetype('readonly/fanwood-webfont.ttf',50)

# build a list of 9 images which have different brightnesses
#enhancer=ImageEnhance.Brightness(image)
images=[]
#for i in range(1, 10):
#    images.append(enhancer.enhance(i/10))

for fact in [0.1,0.5,0.9] :
    my_image = image.copy()
    t1 = 'channel 0 intensity ' + str(fact)
    draw = ImageDraw.Draw(my_image)
    draw.rectangle([0,500,1001,564],fill='black')
    draw.text((2,500),t1, font=text_font,align ="left")
    pix = my_image.load()
    for i_width in range(image.width):
        for i_height in range(image.height):
            pix[i_width,i_height] =(int(fact*pix[i_width,i_height][0]),pix[i_width,i_

    images.append(my_image)


    my_image = image.copy()
    t1 = 'channel 1 intensity ' + str(fact)
    draw = ImageDraw.Draw(my_image)
    draw.rectangle([0,500,1001,564],fill='black')
    draw.text((2,500),t1, font=text_font,align ="left")
    pix = my_image.load()
```

```
        for i_width in range(image.width):
            for i_height in range(image.height):
                pix[i_width,i_height]=(pix[i_width,i_height][0],int(fact*pix[i_width,i_hei
        images.append(my_image)


        my_image = image.copy()
        t1 = 'channel 2 intensity ' + str(fact)
        draw = ImageDraw.Draw(my_image)
        draw.rectangle([0,500,1001,564],fill='black')
        draw.text((2,500),t1, font=text_font,align ="left")
        pix = my_image.load()
        for i_width in range(image.width):
            for i_height in range(image.height):
                pix[i_width,i_height]=(pix[i_width,i_height][0],pix[i_width,i_height][1],i

        images.append(my_image)

    # create a contact sheet from different brightnesses
    first_image=images[0]
    contact_sheet=PIL.Image.new(first_image.mode, (first_image.width*3,first_image.height*
    x=0
    y=0

    for img in images:
        # Lets paste the current image into the contact sheet
        contact_sheet.paste(img, (x, y) )
        # Now we update our X position. If it is going to be the width of the image, then
        # and update Y as well to point to the next "line" of the contact sheet.
        if x+first_image.width == contact_sheet.width:
            x=0
            y=y+first_image.height
        else:
            x=x+first_image.width

    # resize and display the contact sheet
    contact_sheet = contact_sheet.resize((int(contact_sheet.width/2),int(contact_sheet.hei
    display(contact_sheet)
```



1.1 HINT 1

Check out the `PIL.ImageDraw module` for helpful functions

1.2 HINT 2

Did you find the `text()` function of `PIL.ImageDraw`?

1.3 HINT 3

Have you seen the `PIL.ImageFont` module? Try loading the font with a size of 75 or so.

1.4 HINT 4

These hints aren't really enough, we should probably generate some more.

```
In [20]: import PIL
         from PIL import ImageFont, ImageDraw
         help(ImageFont.truetype)
         print('______'*50)
         help(ImageDraw.Draw)
         print("--------------------------------------------------------------------------------
         dir(ImageDraw.Draw)
```

```
Help on function truetype in module PIL.ImageFont:

truetype(font=None, size=10, index=0, encoding='', layout_engine=None)
```

```
    Load a TrueType or OpenType font from a file or file-like object,
    and create a font object.
    This function loads a font object from the given file or file-like
    object, and creates a font object for a font of the given size.

    This function requires the _imagingft service.

    :param font: A filename or file-like object containing a TrueType font.
                 Under Windows, if the file is not found in this filename,
                 the loader also looks in Windows :file:`fonts/` directory.
    :param size: The requested size, in points.
    :param index: Which font face to load (default is first available face).
    :param encoding: Which font encoding to use (default is Unicode). Common
                 encodings are "unic" (Unicode), "symb" (Microsoft
                 Symbol), "ADOB" (Adobe Standard), "ADBE" (Adobe Expert),
                 and "armn" (Apple Roman). See the FreeType documentation
                 for more information.
    :param layout_engine: Which layout engine to use, if available:
                 `ImageFont.LAYOUT_BASIC` or `ImageFont.LAYOUT_RAQM`.
    :return: A font object.
    :exception IOError: If the file could not be read.

----------------------------------------------------------------------------------------------------
Help on function Draw in module PIL.ImageDraw:

Draw(im, mode=None)
    A simple 2D drawing interface for PIL images.

    :param im: The image to draw in.
    :param mode: Optional mode to use for color values.  For RGB
       images, this argument can be RGB or RGBA (to blend the
       drawing into the image).  For all other modes, this argument
       must be the same as the image mode.  If omitted, the mode
       defaults to the mode of the image.

----------------------------------------------------------------------------
```

```
Out[20]: ['__annotations__',
          '__call__',
          '__class__',
          '__closure__',
          '__code__',
          '__defaults__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
```

```
 '__eq__',
 '__format__',
 '__ge__',
 '__get__',
 '__getattribute__',
 '__globals__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__kwdefaults__',
 '__le__',
 '__lt__',
 '__module__',
 '__name__',
 '__ne__',
 '__new__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

In [ ]: