CS7GV1 Computer Vision

Mid-term Project 2022/23

**Part 1: Photo effects**

For each of the following photo filters, many of the processing steps are similar, such as splitting the image into its 3 channels (RGB), clipping the values between 0 and 255 and converting them to integers.

For each of these tasks I created functions that are called where needed in each filter.

The **split** function takes the img array as an input and returns three 1D arrays of the R, G and B channels.

def split(img):

    b = img[:,:,0]

    g = img[:,:,1]

    r = img[:,:,2]

    rgb = r,g,b

    return rgb

The **clipping** function takes an array as input and uses the numpy clip function to limit the array to values between 0 and 255.

def clip(r,g,b):

    b = np.clip(b,0,255)

    r = np.clip(r,0,255)

    g = np.clip(g,0,255)

    return r,g,b

The **convert\_to\_int** function takes an array as input that is typically filled with float values and returns the array converted to integers.

def to\_uint8(r,g,b):

    b=b.astype('uint8')

    g=g.astype('uint8')

    r=r.astype('uint8')

    return r,g,b

1. **Exposure Filter**

My implementation of the exposure filter is to add the original color value multiplied by a fraction of the input and 255. This ensures the blacks stay black and the brightness is increased proportionally.   
F(C,amt) = C + C(amt/255)

I think this is a reasonable implementation because it avoids the issue of faded blacks when you increase every pixel by the same amount.

    r = r[:,:] + ((r[:,:]/255) \* amt)

    g = g[:,:] + ((g[:,:]/255) \* amt)

    b = b[:,:] + ((b[:,:]/255) \* amt)

1. **Contrast Filter**

There are many different interpretations as to how a contrast filter should be applied to an image. In essence it needs to increase the distance between all the values in the image. However, this can have unintended consequences on the colour of the image and perform strangely in dark or bright regions if not implemented correctly.

To adjust the contrast, I created a function that takes the image and a value between 0.5 and 2 as input. The formula I found that gave the most reasonable results is as follows:

F(C,amt) = (C-128) \* amt + 128

    r = (r-128) \* contrast + 128

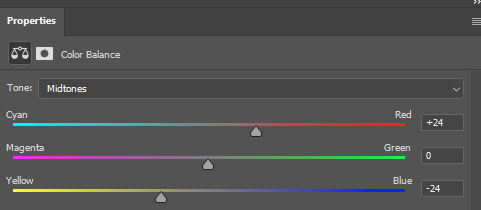
    g = (g-128) \* contrast + 128

    b = (b-128) \* contrast + 128

The resultant images resembled that of when I applied a similar amount of contrast to an image in photoshop.

1. **Saturation Filter**
2. **Temperature Filter**

To adjust the temperature I created a function that takes the image and an amount as a parameter.

The Input parameter can range from -100 to 100. Positive values will make the image look warmer and negative values will make the image look cooler.

To make the image warmer the input amount is added to each pixel in the red channel and subtracted from the blue channel. When a negative value is used as a parameter the inverse operation is performed. The blue channel is increased and red decreases.

b = b[:,:] - amt

r = r[:,:] + amt

This is typically the method used in photoshop to adjust the colour balance of an image.

1. **Tint Image**

The tint function operates similarly to the temperature filter with the same inputs of the image and an amount ranging from -100 to +100. The function has the same functionality as the middle slider in the photoshop screenshot above.

Positive values will make the image look greener and negative values will make the image look more magenta.

    g = g[:,:] + amt

1. **Binary Image filter (Thresholding)**

The function I used to create a binary image takes the image and an integer between 0 and 255 as input. It first converts the image to greyscale using the filter described above. It then loops through each pixel in the image and checks if the value of the pixel is above or below the input value.

If it is below, the value of the pixel is changed to black and if it is equal or higher it is set to 255.

1. **Solarization**

My interpretation is that the solarization filter is where in photography the image is recorded on a negative or on a photographic print is wholly or partially reversed in tone. Dark areas appear light or light areas appear dark.

1. **Image Flip Horizontal and Vertical**

I created two filters to flip in either the horizontal or vertical direction. They both create an empty array and fill it using the original image pixels in reverse order.

for i in range(img.shape[0]):

        for j in range(img.shape[1]):

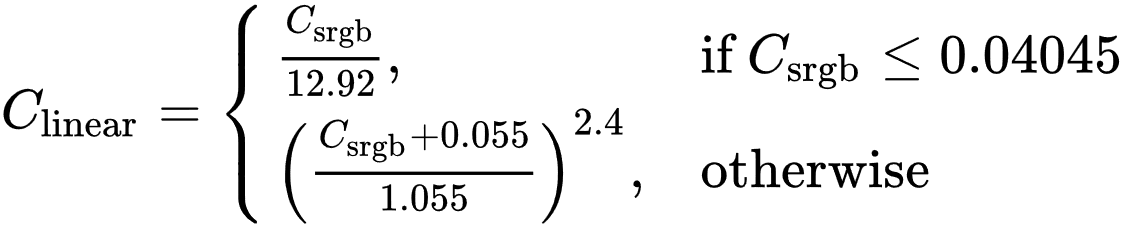
            output[i,j] = b[i,img.shape[1]-1-j], g[i,img.shape[1]-1-j], r[i,img.shape[1]-1-j]

Useful for creating mirrored images or for creating more training data for a machine learning model.

1. **Greyscale Filter**

There are many ways to convert an image to greyscale. The goal is to maintain the same luminance as the original image and to have the dark and bright areas appear as would be expected by the human eye.

The image must first be gamma corrected because the input rgb likely has a gamma compression function applied to it. This can be performed with the following formula.



    #Gamma expansion

    for i in range(r.shape[0]):

        for j in range(r.shape[1]):

            if(r[i,j] < 0.04045): r[i,j] = r[i,j]/12.92

            if(b[i,j] < 0.04045): b[i,j] = b[i,j]/12.92

            if(g[i,j] < 0.04045): g[i,j] = g[i,j]/12.92

    r = ((r+0.055)/1.055)\*\*2.4

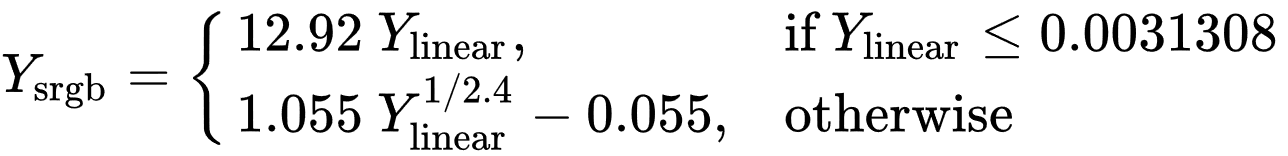
    g = ((g+0.055)/1.055)\*\*2.4

    b = ((b+0.055)/1.055)\*\*2.4

The human eye is more sensitive to certain colours more than others. Because of this each of the channels are weighted before being added together to form the final image. Green gets the highest weight at 0.7152 and Blue has a weight of 0.0722. The human eye is a lot more sensitive to green light than blue.

    Y = 0.2126\*r + 0.7152\*g + 0.0722\*b

Each pixel then needs to be gamma compressed to be converted back to the typical non-linear sRGB representation.



    #Gamma Compression

    output = np.zeros((img.shape[0],img.shape[1],1))

    for i in range(img.shape[0]):

        for j in range(img.shape[1]):

            if Y[i,j] <= 0.0031308:

                output[i,j] = 12.92\*Y[i,j]

            else:

                output[i,j] = (1.055\*(Y[i,j]\*\*(1/2.4)))-0.055

Part 2: Sharpening, blur, and noise removal

1. Softening
2. Gaussian Sharpening
3. Median Sharpening
4. Bilateral
5. Non-Linear Diffustion

Part 3: High-quality image resampling