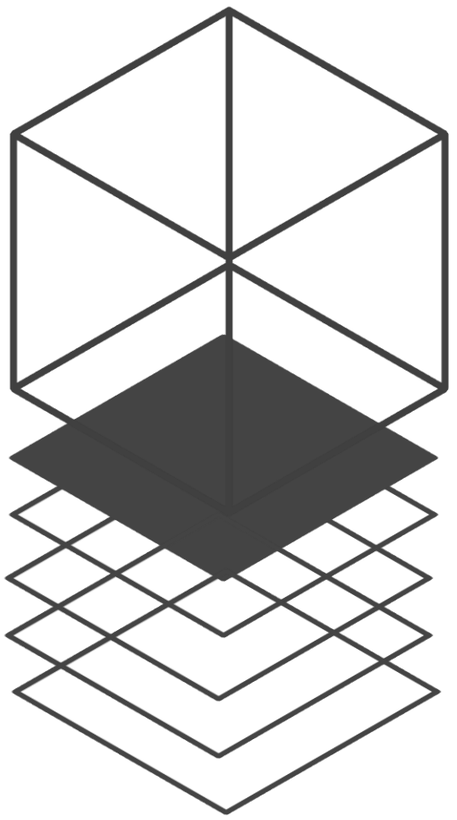
DIGITAL IMAGE FORMATION AND ENHANCEMENT

IMAGE AND VIDEO PROCESSING - MODULE 1

NINO PHILIP RAMONES | GITHUB
2020 - 05616
MARCH 10, 2023

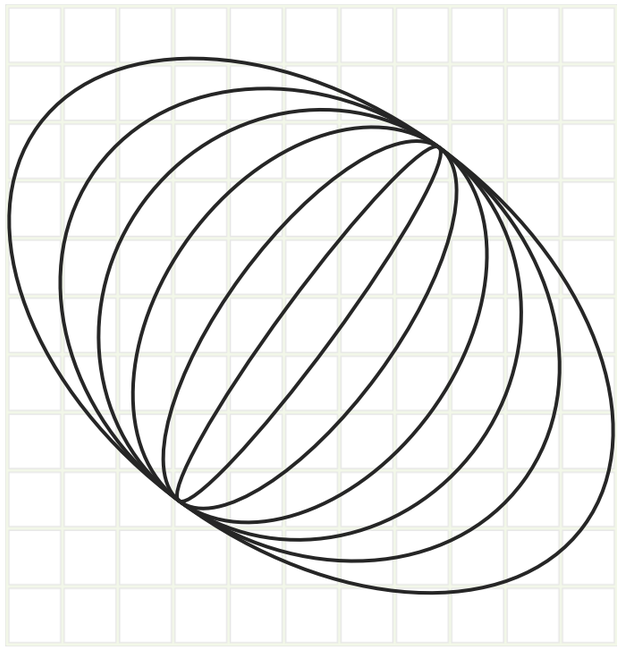
OBJECTIVES

- Mathematically **create** images in Python
- **Save** an image in an appropriate file format
- Use the **back projection technique** to manipulate the histogram of an image to a desired cumulative distribution
- Improve the appearance of gray level and color images by implementing **white balancing algorithms**



KEY TAKEAWAYS

- **File formats** of images to be used in analysis and modelling do matter!
- Manipulating the **histogram** of an image affects its grayscale or color values for a certain range and ultimately its contrast and overall appearance
- A colored digital image is formed from the **overlay** of three matrices representing the **red**, **green**, and **blue** channels

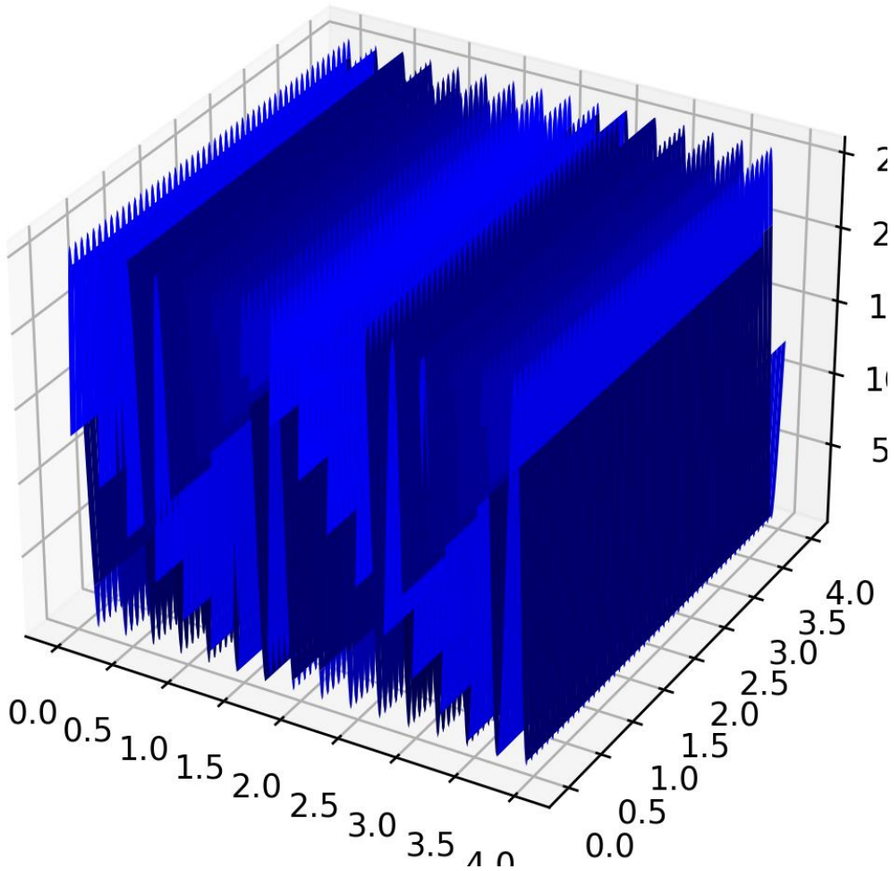


SOME PITFALLS

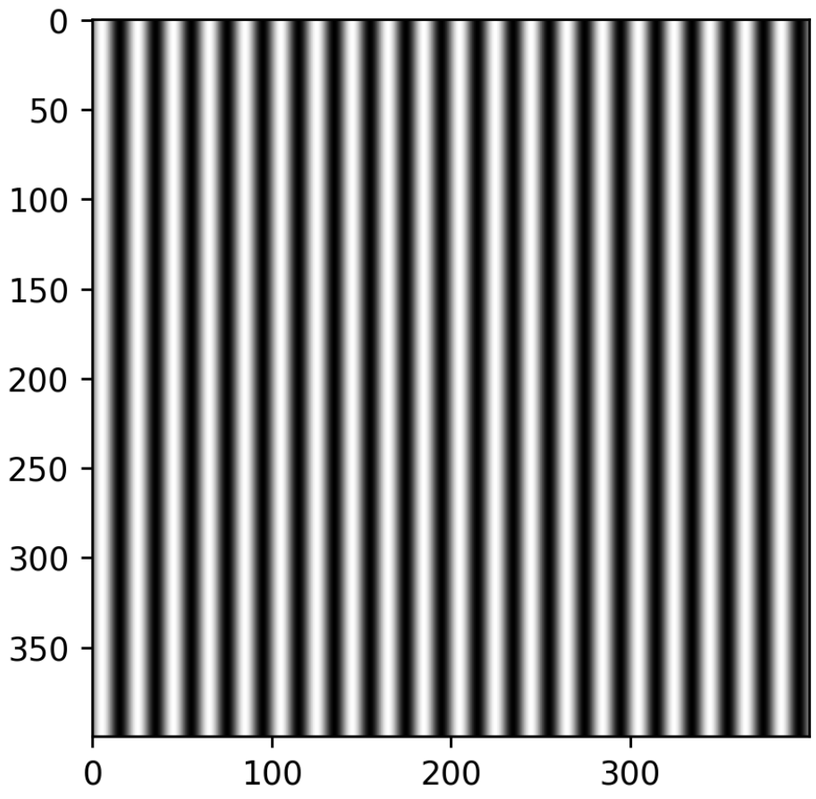
- Processing an RGB image in **float type** (0 to 1) at some cases resulted to an overflow of pixels compared to an integer **0 to 255** representation
- This was circumvented through clipping or **normalization** of values

IMAGE DIY

The first part of the activity involves the creation of synthetic images through mathematical representations. A series of 4 cm by 4 cm (not to scale) **optical elements** confined in a 400 pixel by 400 pixel were made as shown.

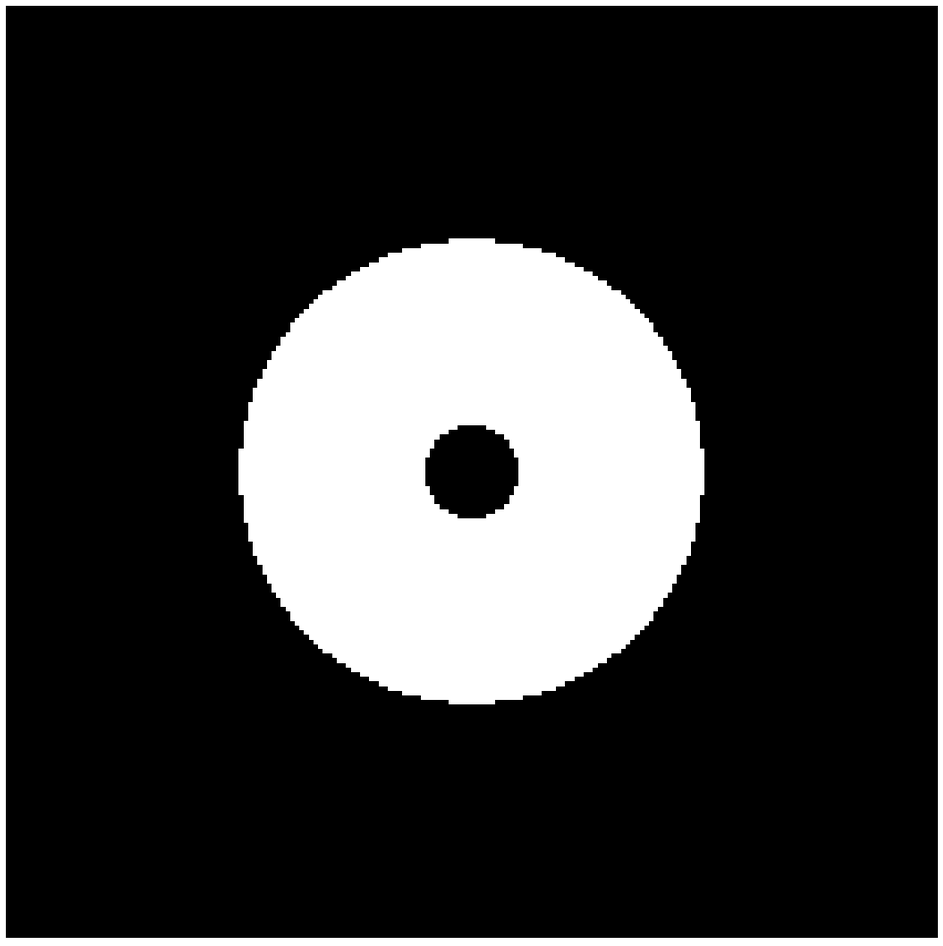


Sinusoid along the x-direction with frequency of 4 cycles per cm

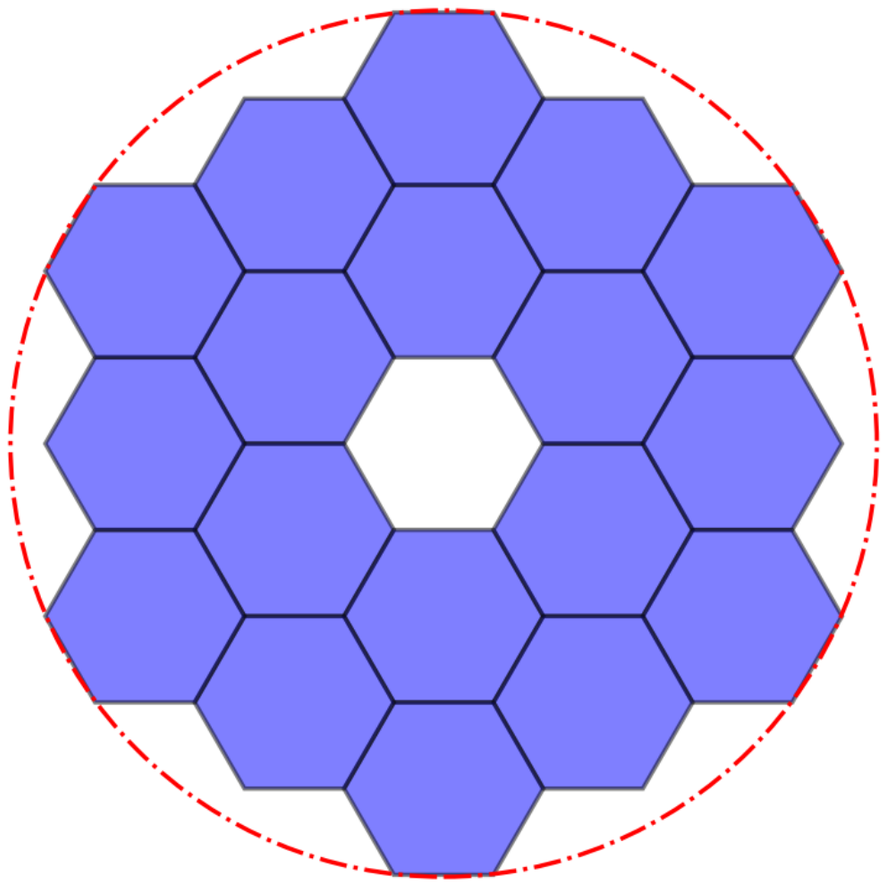


Grating with frequency of 5 lines per cm

The **Hubble's primary mirror** and **James Webb space telescope** were also synthetically created using Python and some of its built-in libraries for polygon creation.



Circle with a hole in the middle (annulus)

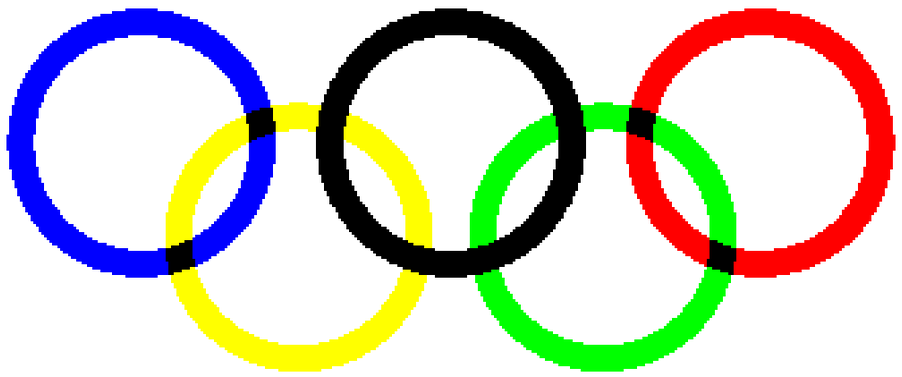


Hexagonal array circumscribed by a circle

The code in generating the inner hexagon array was patterned from Stackoverflow in courtesy of user named tmdavidson. The rest was executed by trial and error for centers.

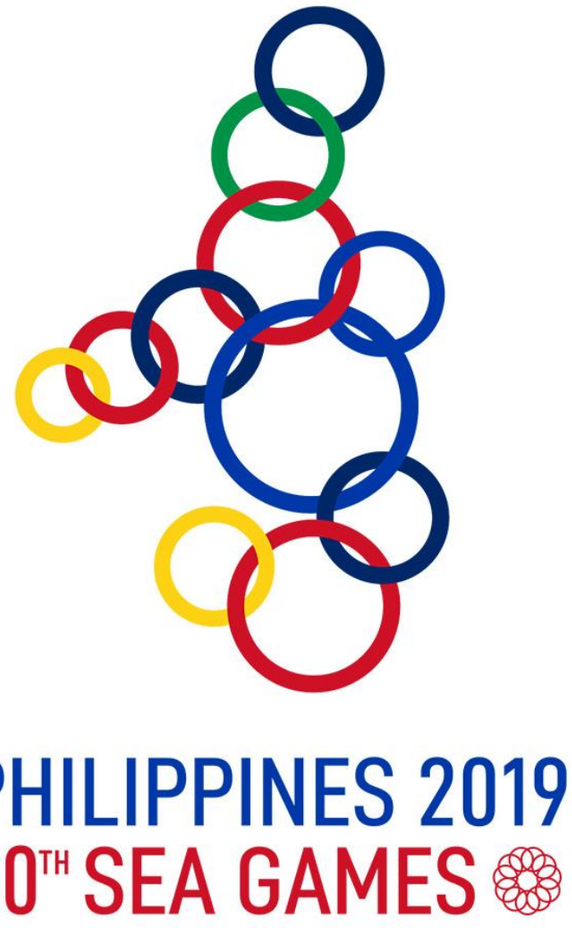
COLOR IMAGE

Using the principle of color subtraction and RGB channel overlaying, the Olympic logo was generated. Color subtraction was used as the value that is part of the ring becomes zero, otherwise, it retains its original value – using the **np.where**( ) function.

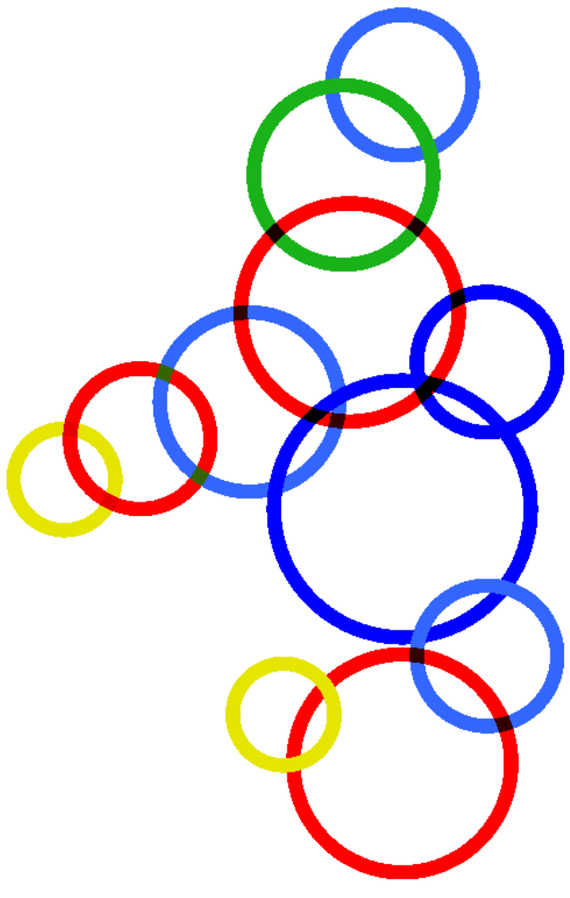


Mathematically created Olympic logo

Upon feeling inclined in doing the Olympic logo, the Philippine sea games logo was also recreated in codes. Using the same method and algorithm from the previous activity, the circles also vary in centers, radii, and colors as well.



Original sea games logo from Esquiremag.



Mathematically created sea games logo

COLORED TO GRAYSCALE IMAGE

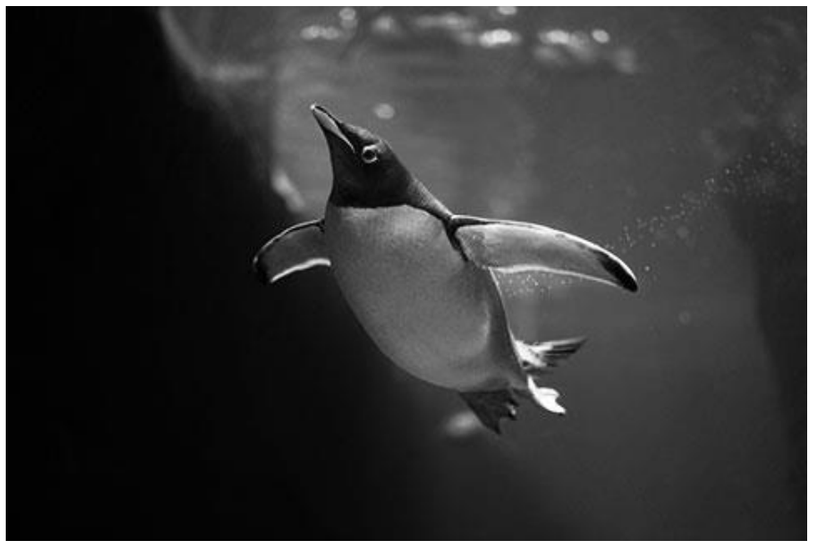
The working image was initially imported to the notebook and was converted to grayscale image. The original image has a maximum value of 1, which was then scaled to 255 for the grayscale while accounting for the **8-bit representation** of channels.

Original image (0 to 1)

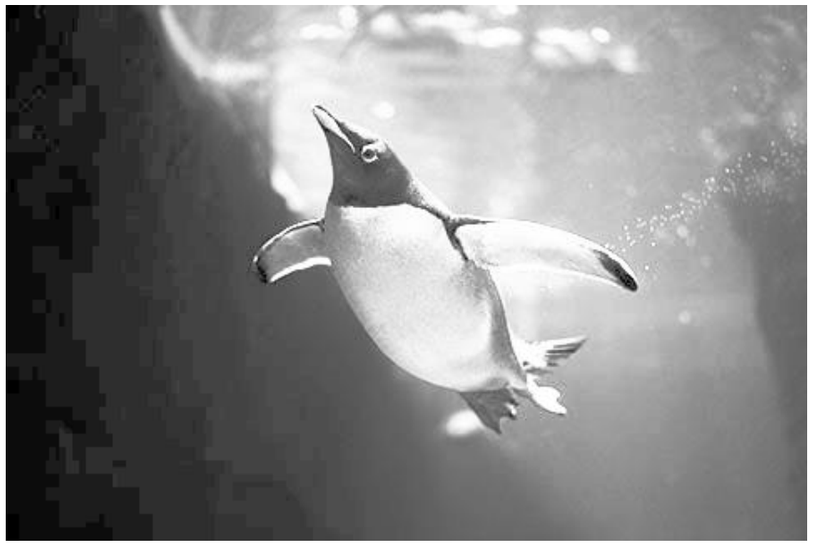
Original grayscale image (0 to 255)

An image of a cute, modest penguin swimming in the ocean was the test subject for activity. **PIL library** was also used in converting the image to grayscale as well mapping the values to 8-bit.

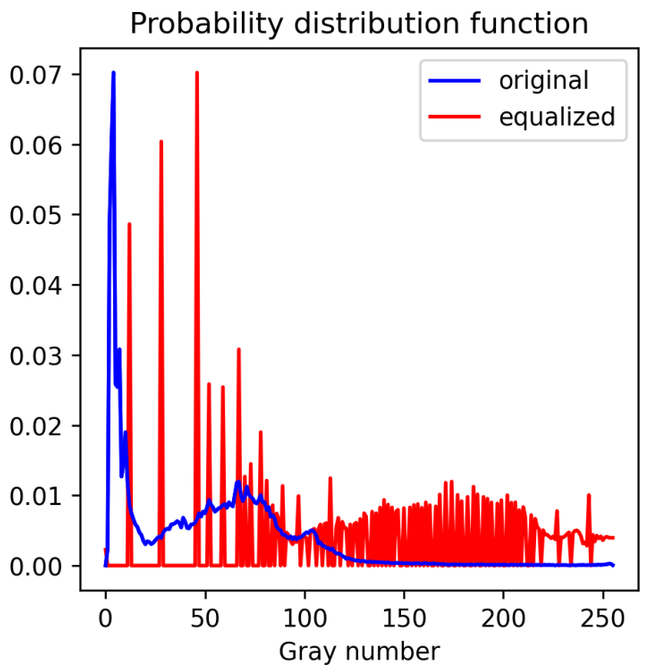
HISTOGRAM MANIPULATION – LINEAR CDF

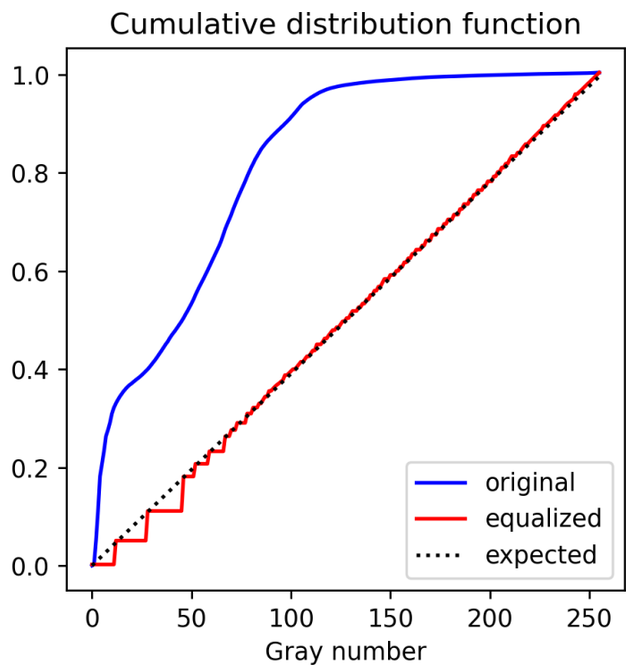


Original image



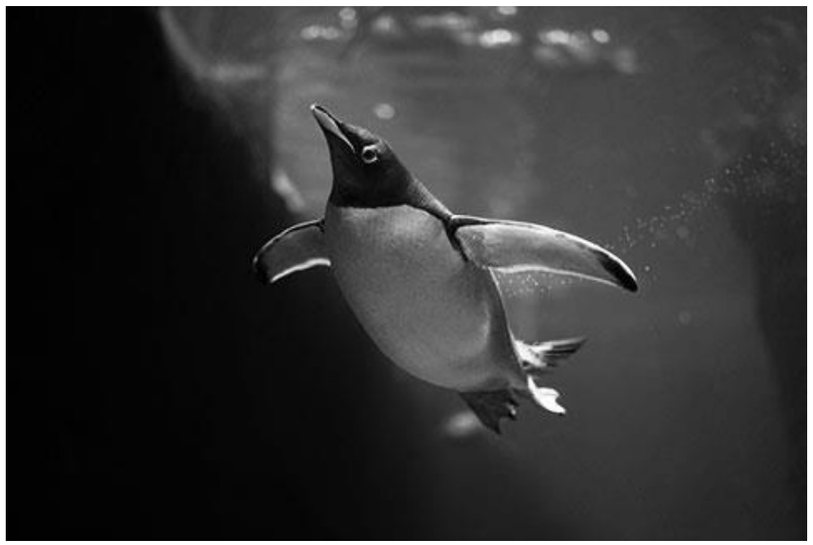
Histogram equalized image



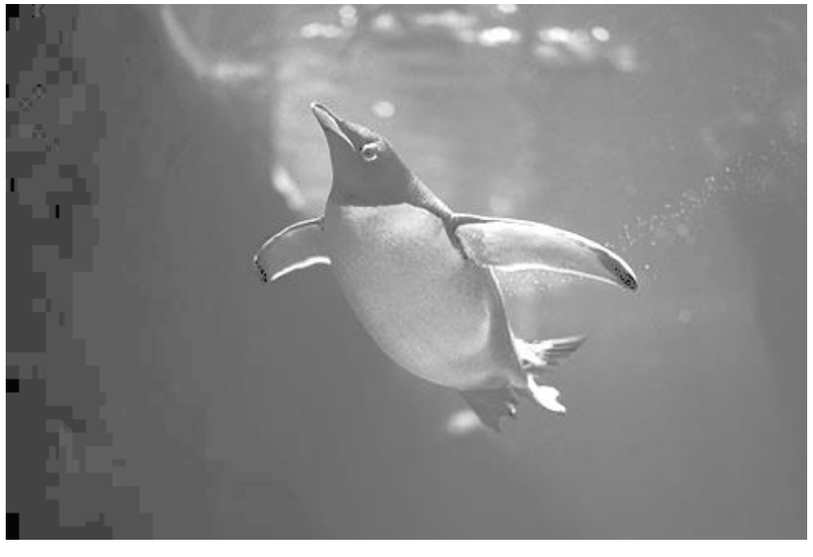


Histogram equalization of the image was performed by interpolating the initial CDF to a **linear** CDF. A contrast difference between the equalized image and the original is apparent as the gray number distribution on the histogram were modified. This modification resulted in the linearized image being much brighter than the original gray image, i.e **the most frequent gray numbers or intensity values were effectively spread throughout the range.**

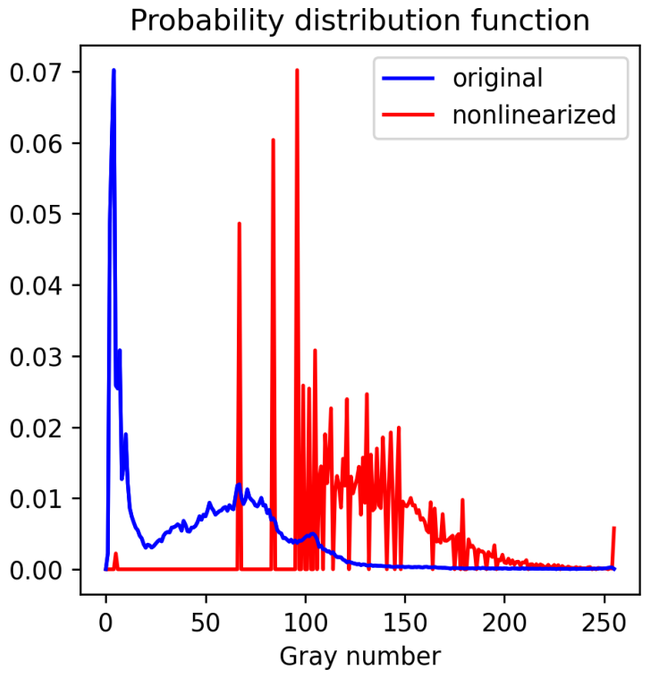
HISTOGRAM MANIPULATION – NONLINEAR CDF

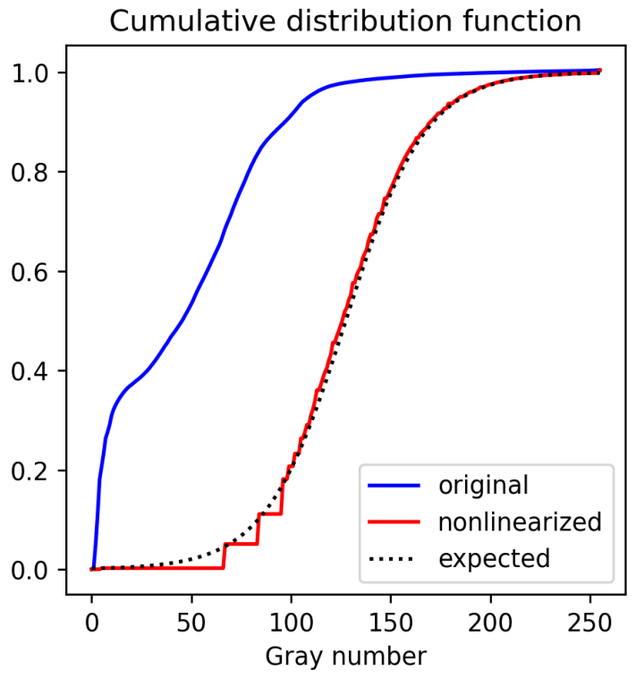


Original image



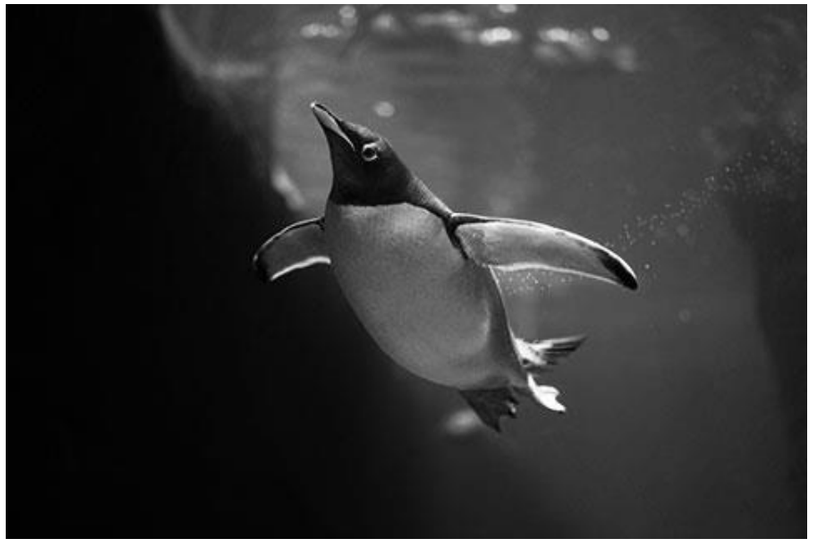
Nonlinearized image





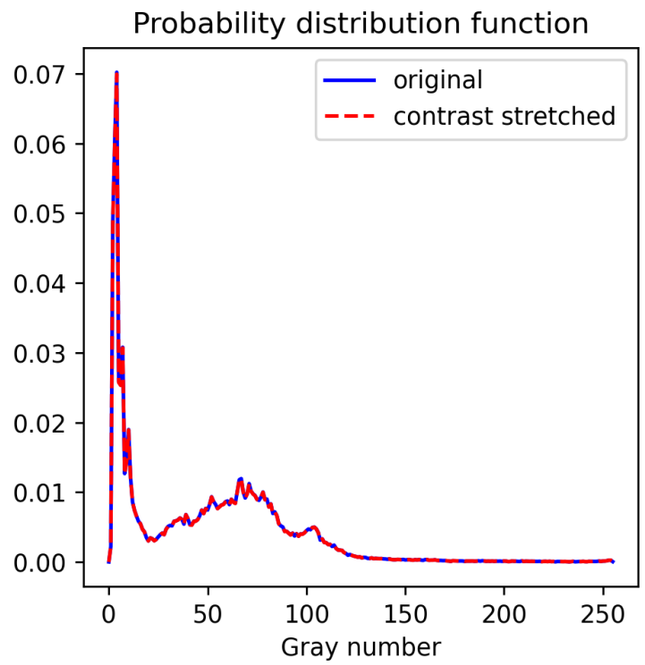
The case of a nonlinear CDF taking the form of a **sigmoid** was also examined to imitate the response of a human eye. Compared to the linear CDF, the sigmoid CDF of the resulting image tend to have an **overall gray appearance**, which is expected based on the PDF, i.e. peaks at the middle gray numbers. The resulting image also tend to be grainier on the darker areas, which can be attributed to the PDF being almost **zero** at low gray values (around 0 to 50).

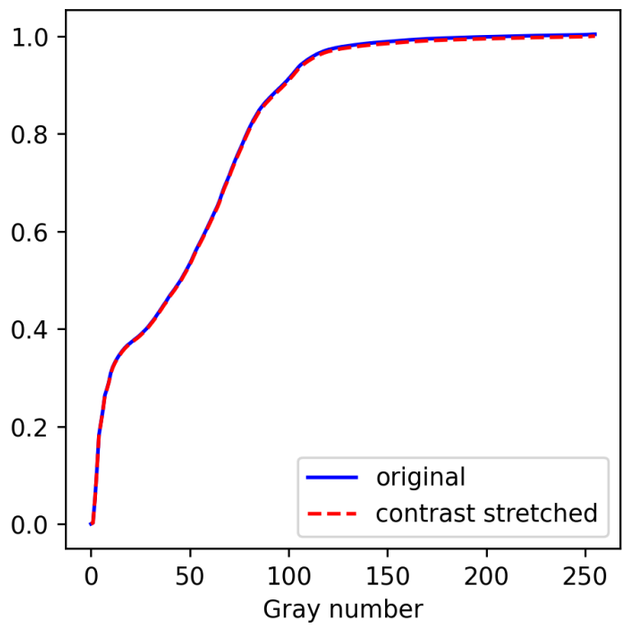
CONTRAST STRETCHING



Original image

Contrast stretched image





Upon implementing the **contrast stretch** on the original grayscale image, **no changes** were observed on the contrast stretched image. This is also apparent in the PDF and CDF plot of the resulting image, which is in agreement to

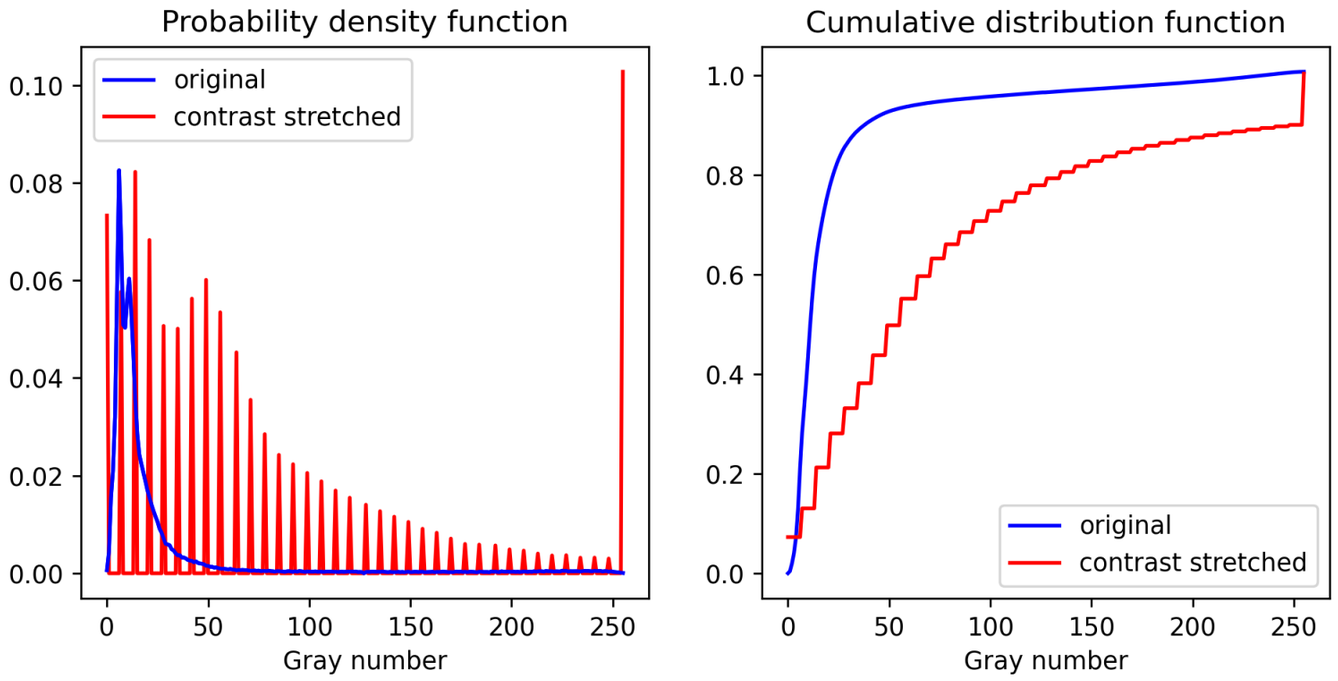
$$I_{\text{new}} = \frac{I_{\text{old}} - I_{\text{min}}}{I_{\text{max}} - I_{\text{min}}}$$

as the minimum and maximum value of a grayscale is 0 and 255, respectively.

CONTRAST ENHANCEMENT

Original image

Contrast stretched image



Another image that is **much darker** than the previous image was also considered. By choosing the lower 5th and upper 90th percentile as the minimum and maximum value for the algorithm, it can be observed that the resulting image is in **high contrast** compared to the original grayscale. This is also evident in the PDF and CDF plots shown as the gray values are much now much **more spread** to the contrast stretched image.

Image of a cute pupperino basking in the sunlight from Pixabay.

RGB IMAGE - DECOMPOSED

Red channel

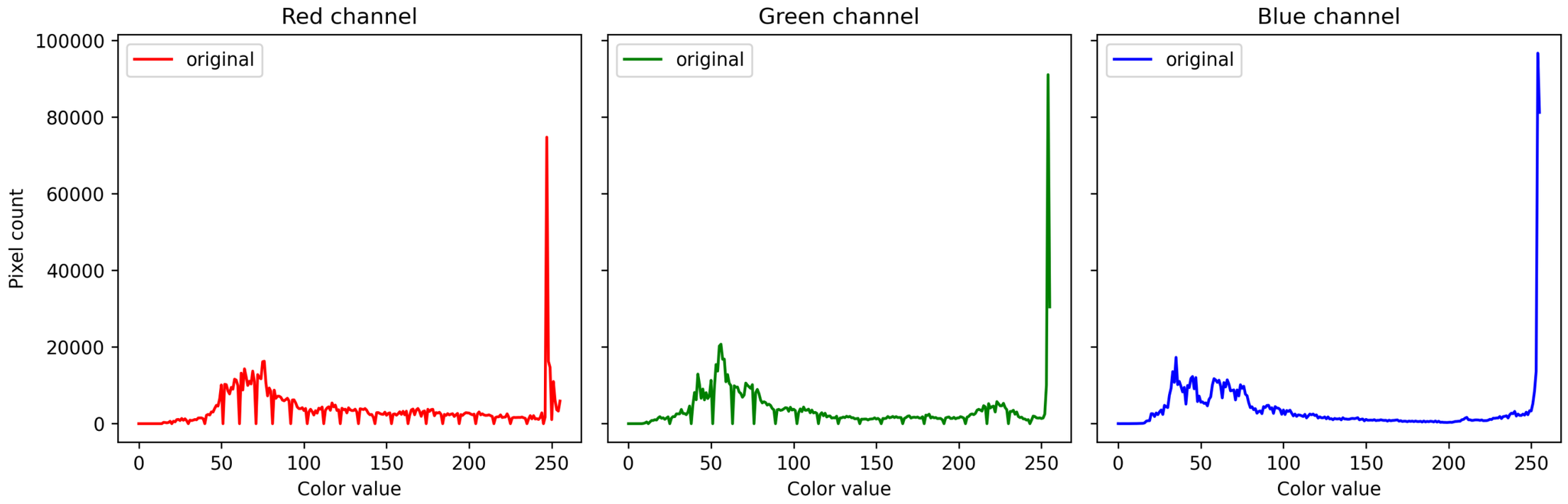
Green channel

Blue channel

We begin our analysis on colored images by initially **decomposing** it into red, green, and blue channels. Contrary to what the name of the channel implies, the corresponding decomposed channels are **grayscale** in nature as these images only have **one channel** now – with each pixel representing 0 (black) to 255 (white) gray values. These implies that colorized or RGB images in general like shown on the right are made from the **stacking** of these three channels on top of each other!

Original image from [Minecraft]

RGB IMAGE - DECOMPOSED



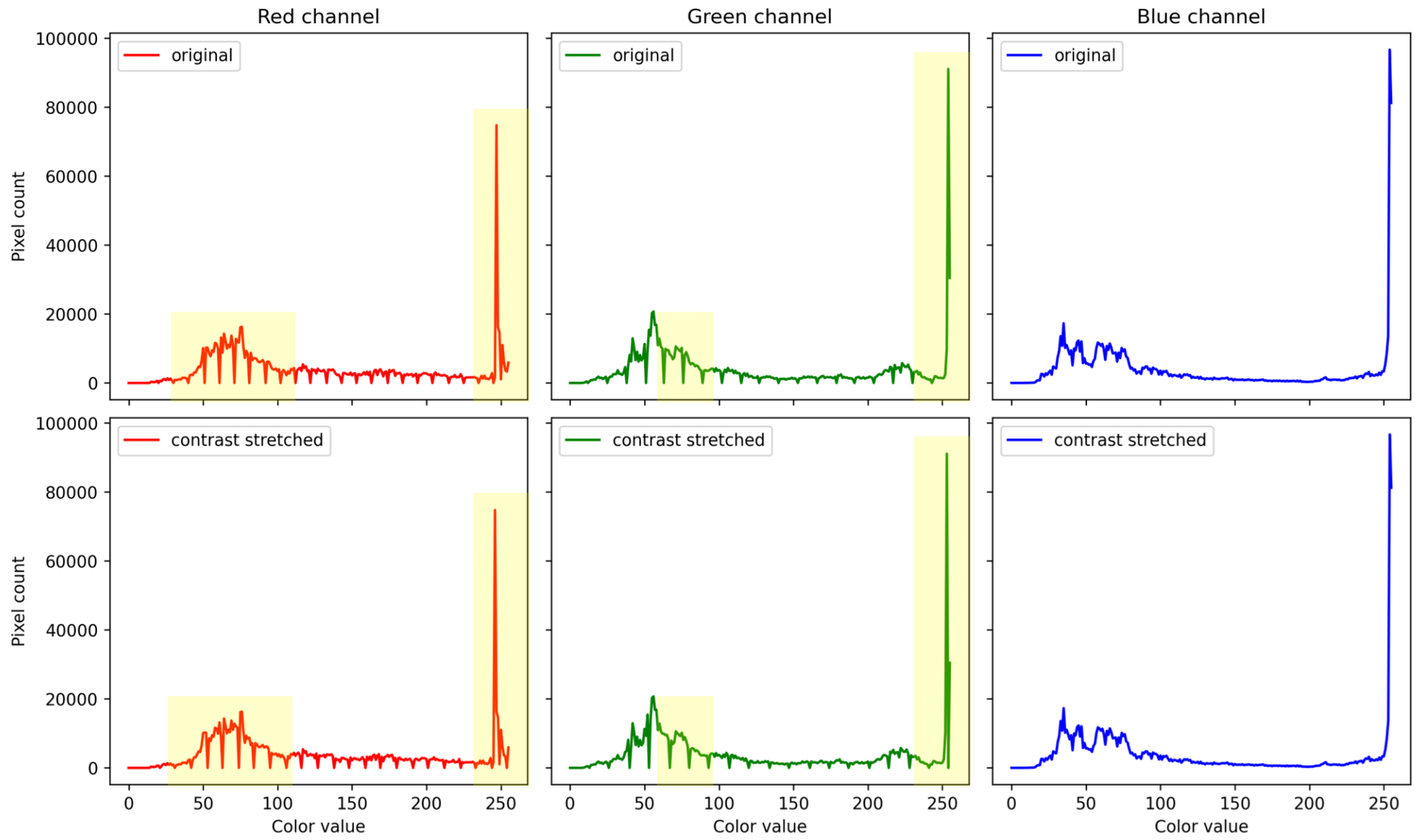
The respective histograms that account the pixel number of such color in each channel were also generated to see which color **dominates** the image. Upon taking the sum of the grayscale values of each, the **red channel** yielded a high value difference compared to the green and blue channels. This is apparent to the color of the original image as it contains wood and dirt blocks, which appears to be brownish red in in-game shaders.

CONTRAST STRETCHING IN RGB

Original image

Contrast stretched image

Since the image was **not mostly unbalanced**, i.e. the elements in the picture appears to be on their true colors in game, small changes in the contrast stretched image were observed. The **resulting image** appears to have a much richer [brownish to red] tone and darker shade in the middle to lower middle areas.



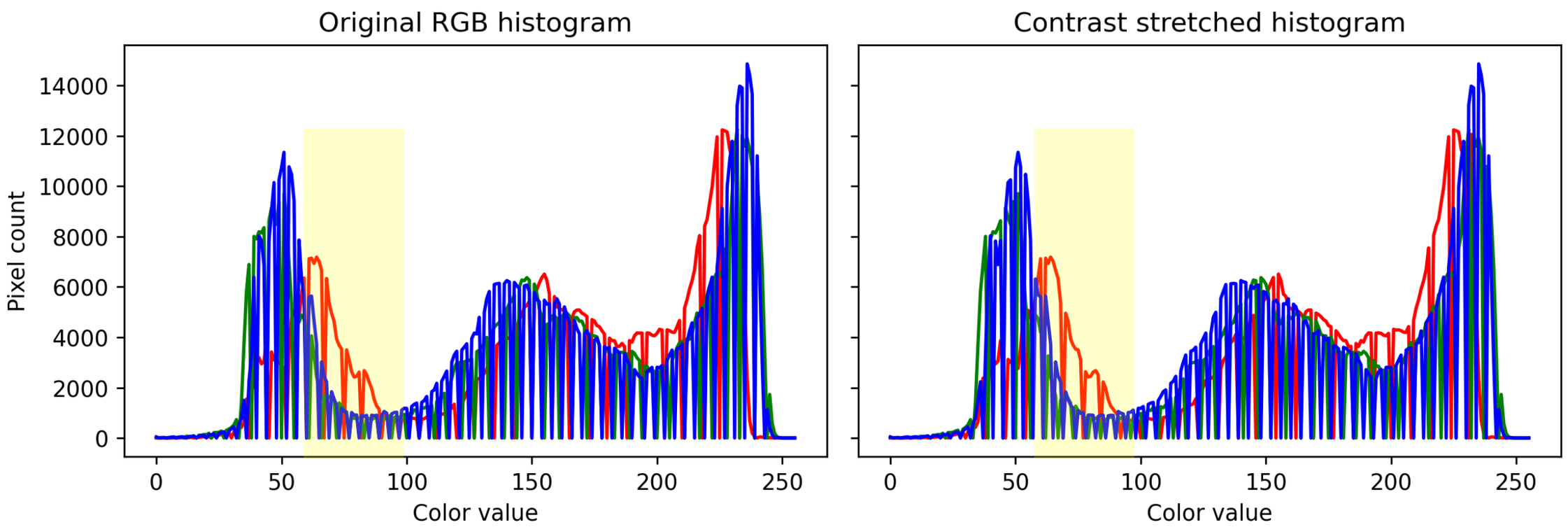
The differences in the RGB histograms of the original and resulting images were **highlighted** in yellow to emphasize the **subtle variances.** From the resulting image, we can see that the shades of the arch area (made of wood and dirt) differ to a certain extent. These can be mapped to the **histograms** shown above as there are changes in the **color value distributions** for the red and green channels.

CONTRAST STRETCHING IN RGB

Original image from Michael Collins

Contrast stretched image

Unlike the previous image, the wonder of contrast stretching algorithm is much more obvious for **unbalanced images**, i.e. the color of the skin of the people in the original image is not what it is supposed to be. The white balancing was achieved by contrast stretching each color channel using a **set of percentile** for the minimum and maximum values of the image array.



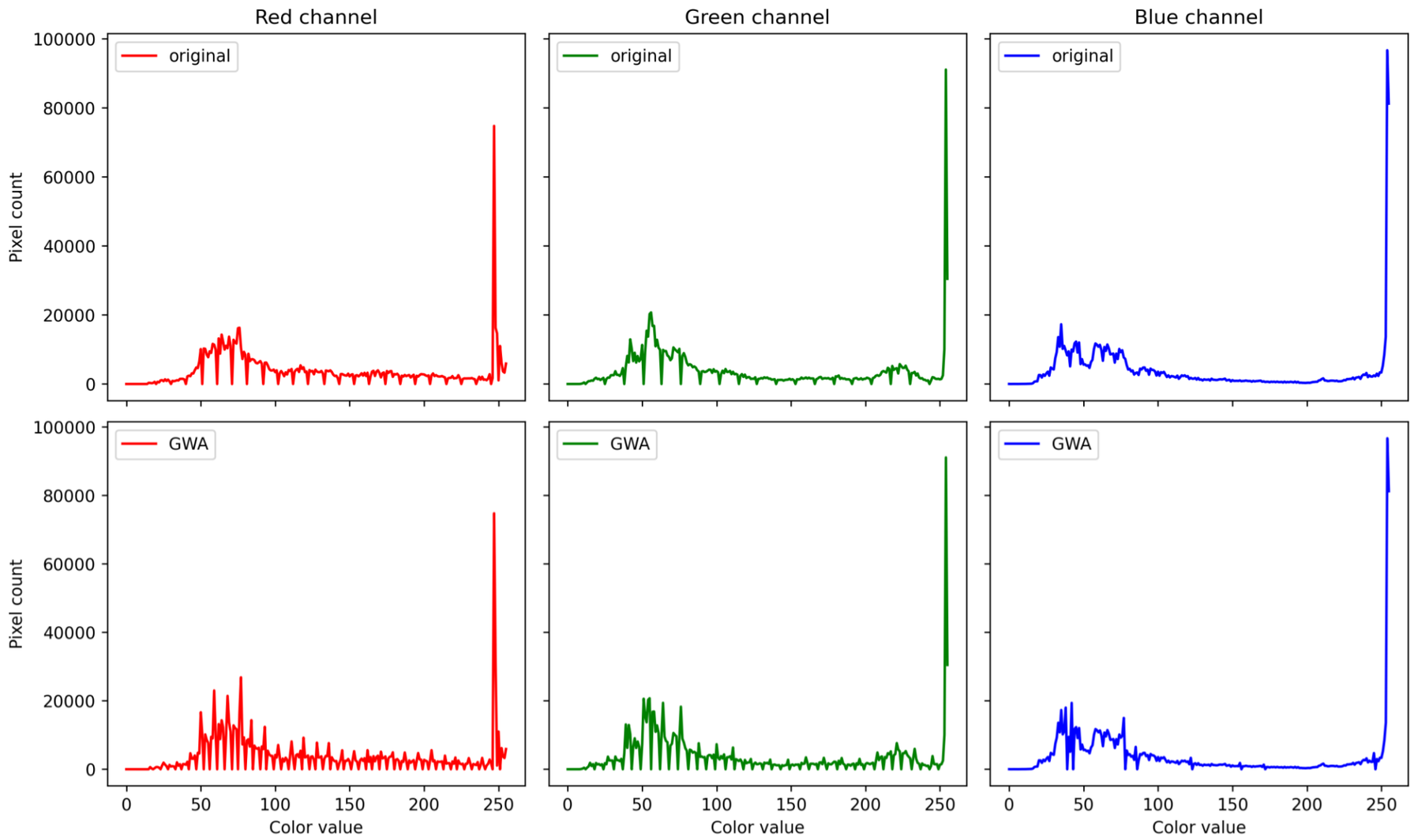
Since the original image has **dominant white pixels** (due to the astronaut suit), the resulting histograms are expected to be **spread or stretched** all throughout the range of pixel values. Slight changes on the resulting histogram were seen, but these resulted to a more **balanced image**, i.e. the yellowish hue of the image were removed, and the skin color appears to be normal compared to before. It can be hypothesized that the red channel from around [50, 100] were stretched by a small amount.

GRAY WORLD ALGORITHM

Original image

White balanced image (GWA)

By taking the **mean of each channel** to estimate the illumination of the original image, the resulting image for the gray world algorithm (GWA) appears to be **white balanced** as the yellow shade brought by the sunlight tends to appear as a white light afterwards. Better choice of image could have been chosen for this to see much more contrast in the resulting image as **GWA estimates the average [reflected] color in the image and compares it to gray**. It can also be inferred that the deviation of colors between the two pictures above can be owed to the possibility that the mean color and its comparison to gray is not that significant (Maulion, 2021).



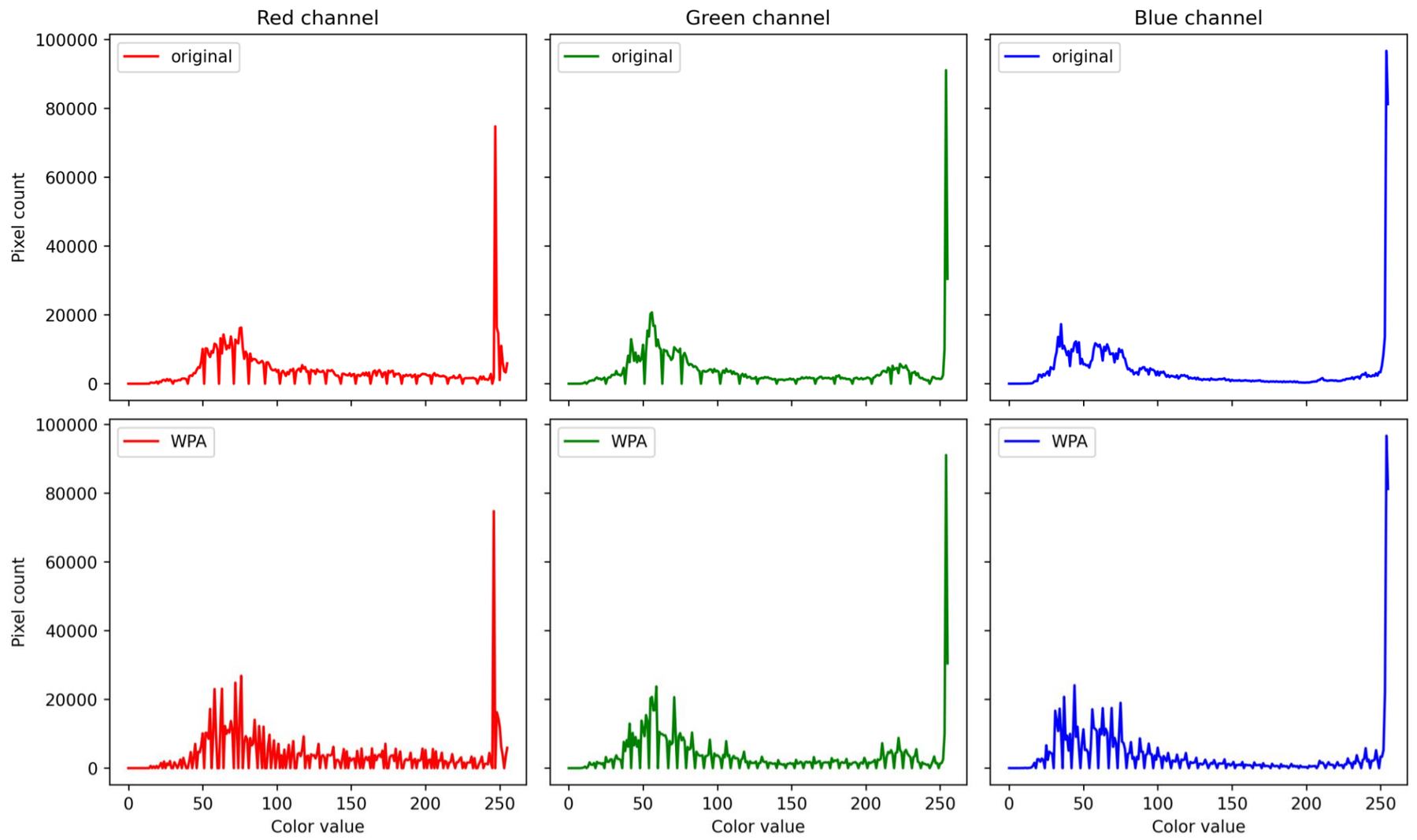
Perceptible differences in the channels were deduced as the original RGB histograms were accentuated, especially in the red and green channels. It can be hypothesized that the **increase in spike** in color values [0, 100] resulted to the sunlight to appear whiter than before as red, green, and blue channels peak at this range of the image.

WHITE PATCH ALGORITHM

Original image

White balanced image (WPA)

The **white patch algorithm** (**WPA**) was implemented by taking a white patch or region of interest (ROI) as shown on the original image. The RGB values of the clouds were averaged and was ratioed to the RGB channel values of the original image. Working through algorithm yielded an image that is **relatively brighter and more vibrant** than the original. Like the GWA, new values of the RGB were **normalized** by dividing the maximum value of the resulting image, and were scaled respectively to RGB values from 0 to 255 – a process similar to **clipping** the array of values from 0 to 1. Among the three **white balancing algorithms**, WPA reliably produced differences compared the image source.



Similar to the GWA, the **white patch algorithm** also fared in manipulating the RGB histograms. The color values in the red channel become more pronounced at [50, 100] as well as **spread all throughout the range**, so are the green and blue channels. As expected, the peaks at [50, 100] interval can be attributed to the sunlight at those certain pixel values on the left. Similarly, the rich earth or brownish red tone of the wood and dirt blocks become more **apparent** – as manifested by the prominences of the **red channel** on the rightmost range of values.

REFLECTION

The latter parts of the activity that deals with **basic image processing** were challenging due to a number of reasons. The primary challenge upon working through the codes was deciding whether to work on **float type values of 0 to 1 or RGB values of 0 to 255**.

It has come to my realization that working on images is much easier for the latter as pixels are generally encoded in an **8-bit format**, which is favored by coding algorithms and the mathematical representations in general. Perhaps, an application to what I have learned on my App Physics 155 class is to have **consistency in my codes your variables**.

I think I could have written my codes and notebooks in a much cleaner way, but, overall, I think I was able to execute the **rudimentary of image processing** in Python and even meet what is expected beyond the objectives. The **white balancing algorithms** such as GWA and WPA have its pros and cons depending on the kind of image you are using as GWA tends to favor images with gray areas while WPA needs a white patch focus to work. **Contrast stretching** can be also be used to restore unbalanced images. Overall, I believe I was able to address **some gaps and nuances** in my results as the choices of my image are somehow not fit in terms of contrast levels. Overall, I would give myself a score of **100/100**.

ACKNOWLEDGEMENT

I would like to acknowledge the **time and knowledge** Sir Rene and Sir Kenneth have shared to us during our laboratory activities when it comes to debugging and addressing some code issues. I also would not have done this without the **help** of Edneil Soriano and Julian Maypa upon resolving some mistakes I made in code structures and stuff.

REFERENCES

1. M. Soriano, Applied Physics 157 – Digital Image Formation and Enhancement, 2023
2. Creating Histograms – Image Processing with Python (datacarpentry.org)
3. Understanding image histograms with OpenCV
4. White Balancing — An Enhancement Technique in Image Processing

GITHUB REPOSITORY

npcrmns/App-Physics-157: Computational Analysis and Modelling in Physics (github.com)