Project 2: Image Processing

Tong Gia Huy [1⋆]
[1] *Faculty of Information Technology, VNUHCM-University of Science, Vietnam*
⋆ *Student ID: 21127307, E-mail: tghuy212@clc.fitus.edu.vn*

Abstract

Ever since the invention of the first camera in the 1800s, photographs have become an inseparable part of our lives. We took pictures to tell stories of our society, our way of life. Not only that, we used those images to study phenomenons around us: animals, geography, from small things like atoms and microbes to celestial bodies in the depth of space far from our reach. It is not common to find a field of science where photography is not present. As technology for capturing light progressed, new methods of processing images were also created. From developing photographs in darkrooms, now we are able of performing intricate operations on individual pixels of digital images with the help of softwares like Adobe Lighroom or Photoshop. In this project, I will study some basic and commonly used image processing techniques to grasp a deeper understanding of how digital images are manipulated in those programs.

1 INTRODUCTION

Nowadays, digital image is not only a form of art, it is also a medium containing information valuable to science. Modern cameras were developed to overcome the biological limits of our eyes. Through their lenses, we are able to open new windows and observe things we have never been able to see. In the field of astronomy, for example, massive telescopes with powerful cameras helped scientists *zoom in* on objects light years away and not visible with our naked eyes. Through photos captured by observatories, astronomers are able to study the motion, characteristics, and formation of those bodies. Therefore, capturing and processing images is a vital part in many researches.

The field of image processing is extremely vast with many methods and algorithms used to enhance and extract data from images. In this report, I will present some simple procedures commonly used in most photo editing softwares. In Project 1, I established a basic understanding of digital image as well as *NumPy* (Harris et al. 2020) calculations. In the next sections of this report, I will demonstrate ways to manipulate digital pixels in order to produce the desired outcome.

2 PROCESSING PROCEDURES

The procedures present in this section include: brightness and contrast adjustment, orientation flipping, grayscale and sepia conversion, blur and sharpen, cropping from center, apply circular and elliptical mask. The Notebook file already includes fully implemented functions corresponding to each procedure, therefore in this report, I will give some brief explanations of each function, any unnecessary technical details will be skipped.

In some procedures, I will make use of histograms to display the distribution of each channel's intensity from 0 to 255. The histogram is widely used in photography, alternative versions of this type of graph are present in well-known softwares like Adobe Lightroom as well as built-in features of digital cameras, which is useful to observe the characteristics of lighting and colour in each picture.

2.1 Brightness adjustment

Brightness adjustment is one of the most simple operation. The purpose of this is to increase the intensity of the pixel towards 255 which will make the image appear *brighter* to the human eye since it approaches

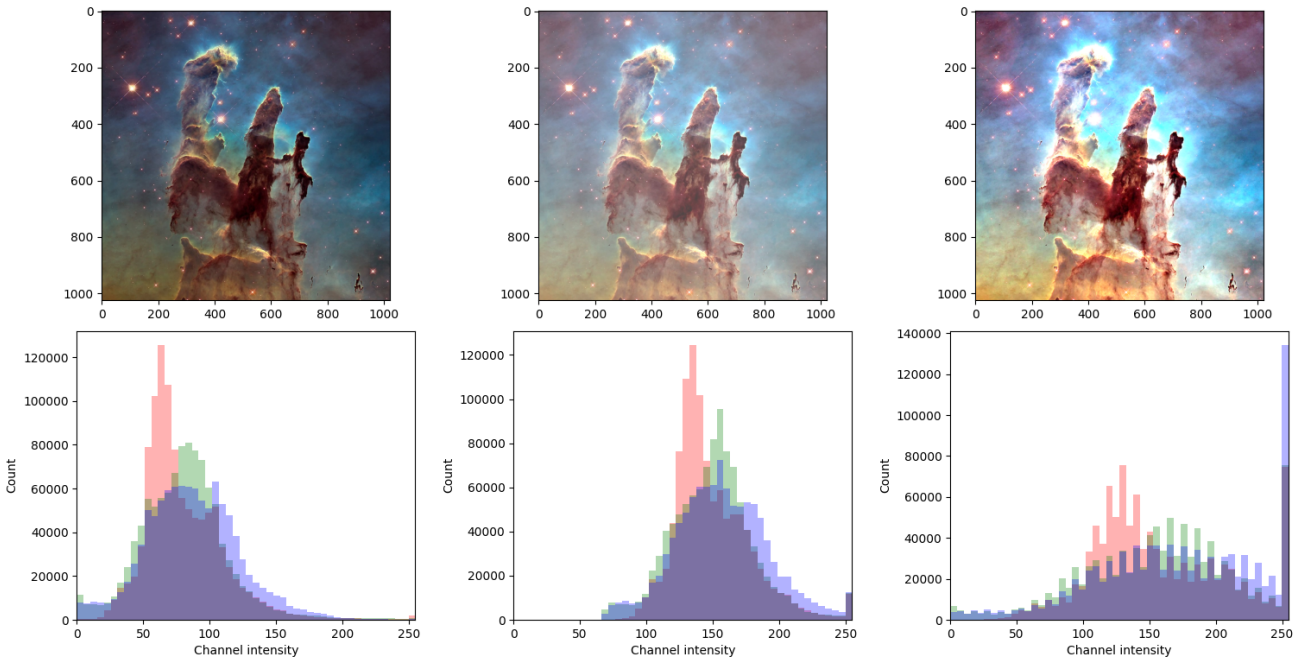

Figure 1. Outputs of brightness adjustment using additive method with an amount of 70 (middle) and multiplicative with a factor of 2 (right), in comparison with the original *Pillars of Creation* photo. The histograms on the bottom row display the distribution of colour channel intensities, 255 intensity values are divided into 50 bins.

the colour white, or decrease it towards 0 to darken it. There are many ways to achieve this effect, including adding the original image with a factor of a constant (usually 255), or multiplying the image with a factor (Ransom 2012).

Each method can be explained with some simple equations. Let us define the brightness factor as α, and the pixel intensity as x, the functions for the additive and multiplicative approach would be, respectively:

$$f(x) = x + \alpha \tag{1}$$

$$f(x) = \alpha x \tag{2}$$

For Eq. 1, our brightness transformation is simply linearly shifting the values up or downwards. Whereas in Eq. 2, we are changing the slope of the original line. Either methods can be utilized to alter the brightness, however after further testing, I prefer the multiplicative method due it being more dynamic in *biasing* brighter values thus giving a more appealing result. Figure 1 shows the differences of the mentioned brightness adjustment methods, the effects on the colour histogram are also visible: the distribution is shifted rightwards when increasing brightness with additive method, on the other hand, the distribution is stretched when using multiplicative. I also tested Adobe Lightroom's exposure setting and found out it produced the same scaling effect, although I am uncertain of the exact algorithm used by the software.

For the Python script, I only implemented Eq. 2 into function **`adjust_brightness(...)`**, which take in the original image as *NumPy* array of pixel values and a factor inputted by the user then return the result of the multiplication, which is the adjusted image.

2.2 Contrast adjustment

Contrast is defined as the difference between the maximum and minimum pixel intensity value, and *contrast adjustment* is an operation aimed to reduce or extend the gap of those values.

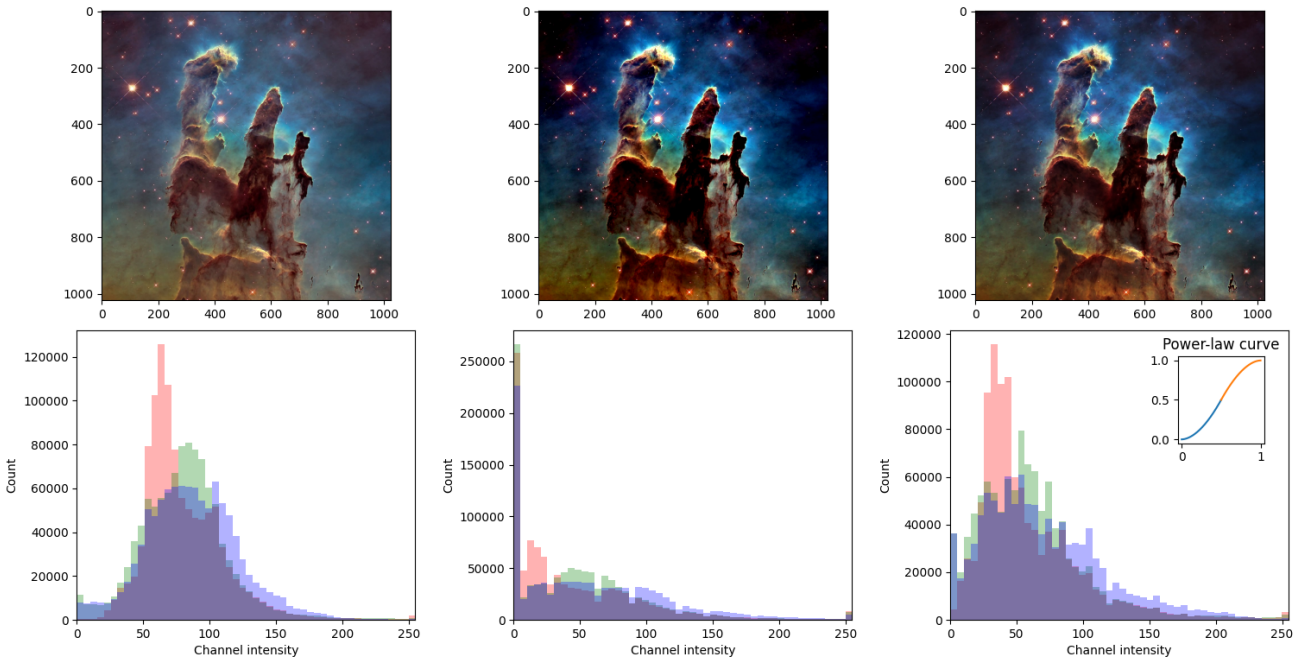

Figure 2. Outputs of contrast enhancement with a factor of 1.8 using `linear` mode (middle) and `curve` mode (right), in comparison with the original *Pillars of Creation* photo. The stretching effect of the histograms is visible for both adjustment modes.

The equation for this is relatively similar to the one in brightness adjustment, the only difference is that instead of centering the intensity values around 0, we move the center point to the middle section of the overall intensity range to create the effect of intensity distribution being stretched both ways. By doing so, *bright* values will tend to get brighter, and *dark* values will get darker, thus enhancing the contrast (Figure 2). Reduction in contrast works the same way, but instead of the *stretching* effect, the distribution is *squished* together (Figure 3). The transformation equation for this effect can be written as, with the range of x normalized to fit between 0 and 1:

$$f(x) = \alpha\left(\frac{x}{255} - 0.5\right) + 0.5 \tag{3}$$

It is important to remember to convert the result back to the original unsigned integer range in order to produce the correct image.

Apart from the method above, there is also a transformation equation that utilizes the power-law S-curve to modify the contrast even more elegantly. Schulz 2012 constructed the system of equations as:

$$f(x) = \begin{cases} 0.5\left(\dfrac{x}{0.5}\right)^{\gamma}, \text{if } 0 \le x \le 0.5 \\ 1 - 0.5\left(\dfrac{1-x}{0.5}\right)^{\gamma}, \text{if } 0.5 < x \le 1 \end{cases} \tag{4}$$

where γ is the intensity of the contrast effect, if γ is more than 1, contrast is enhanced, if it is less than 1 then contrast is reduced.

In the **`adjust_contrast(...)`** function, I implemented both Eq. 3 and 4. The function takes in the pixel array, a factor indicating the intensity of contrast and a variable specifying which formula to use, `linear` for Eq. 3 and `curve` for Eq. 4.

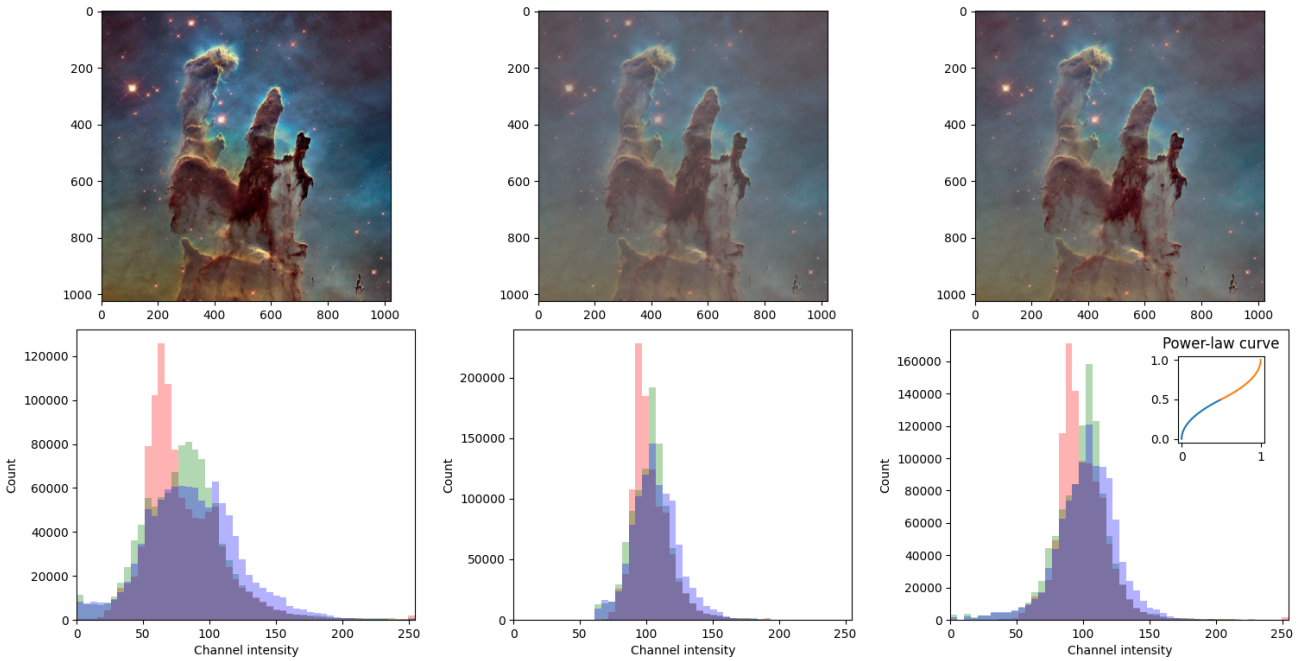

Figure 3. Outputs of contrast reduction with a factor of 0.5 using `linear` mode (middle) and `curve` mode (right), in comparison with the original *Pillars of Creation* photo. The squishing effect of the histograms is visible for both adjustment modes.

2.3 Flipping

Image *flipping* is very straightforward. In its core, it is just an operation that switches the position of the rows or columns of a 2-dimensional array in a way that resembles placing an object in front of a mirror and observe its reflection. For a vertically flipped image, the top rows are switched with the bottoms and vice versa, and for the horizontal mode, left columns are switched with right columns.

The function **`flip(...)`** in my implementation uses *NumPy*'s `flip` function to *mirror* the image on a specified axis: axis 1 for horizontal flipping, and axis 0 for vertical.

2.4 Grayscale and sepia conversion

2.4.1 Grayscale conversion

A *grayscale* image is an image which the value of each pixel is a single value from 0 to 255, which represents the amount of light present (Wikipedia 2023). This type of image is known for having a black-and-white kind of look.

Converting from RGB format to grayscale means combining 3 separate channels into 1 channel. It is possible to do so via calculating the mean of red, green, and blue values in a pixel. However, the result of this transformation can be unpleasant due to it not taking in consideration of human perception of colour and each colour's wavelength. Through extensive research, the International Telecommunication Union (ITU) developed a recommendation for standard-definition television and video (ITU-R BT.601), where a weighted formula for grayscale intensity conversion is derived, as in Dynamsoft 2019:

$$I_{\text{grayscale}} = 0.299r + 0.587g + 0.114b \tag{5}$$

In Python, Eq. 5 can be easily implemented in the **`to_grayscale(...)`** function. The input for this procedure is just the original image array, the grayscale version is constructed with the help of *NumPy*'s vector dot product calculation.

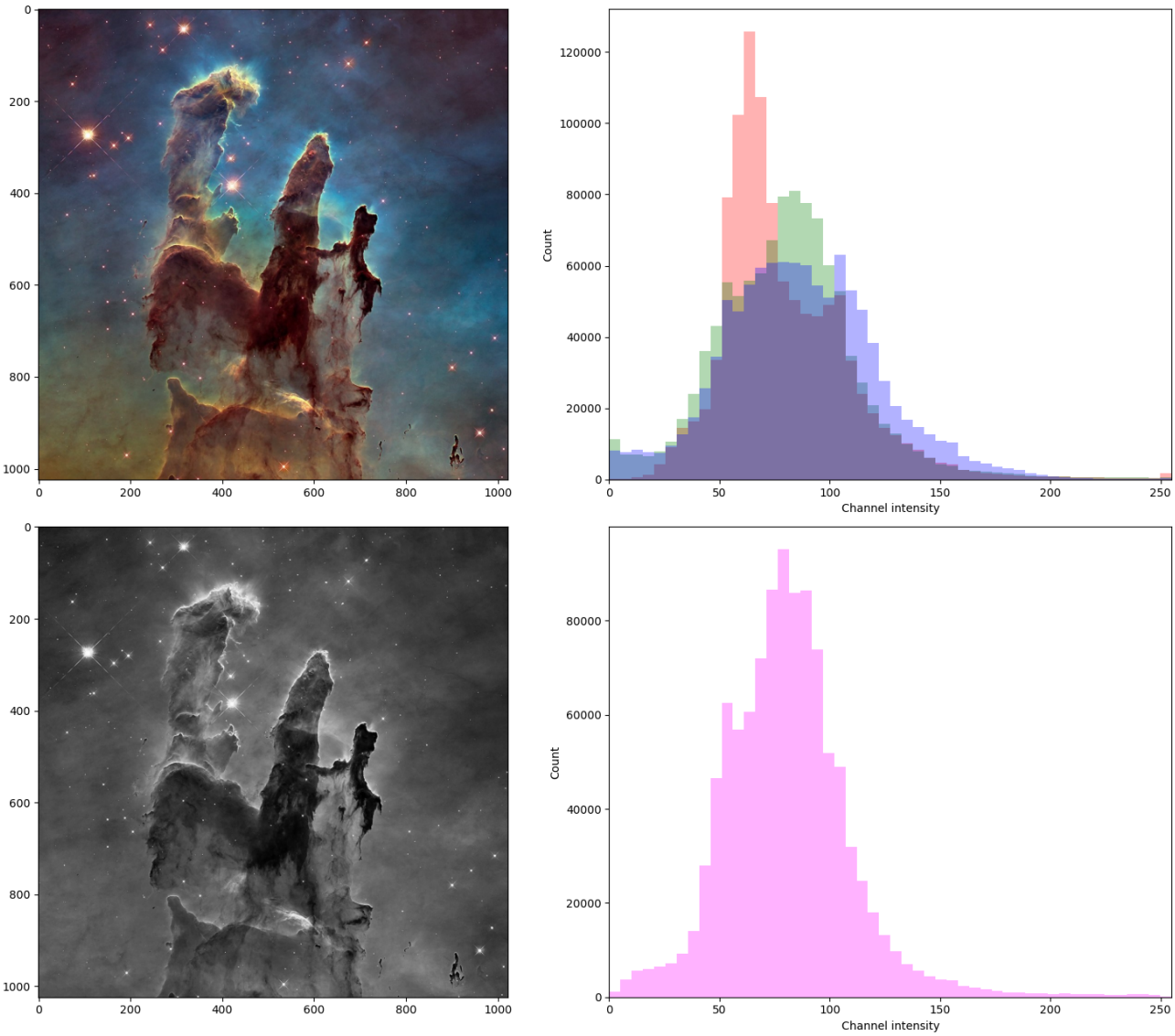

Figure 4. Output of grayscale conversion (bottom), in comparison with the original *Pillars of Creation* photo. The histogram now only displays 1 channel intensity.

2.4.2 Sepia conversion

Sepia is somewhat similar to black-and-white photos, but with a brownish or tan colour tone . This unique characteristic dates back to the days of film photography, in the darkroom. Here, images are developed by immersing photographs into a chemical compound. This chemical converts the metallic silver in the emulsion of a print into a silver sulfide compound, which makes the photo more resistant to environmental pollutants (Fisch et al. 2023).

In the function **`to_sepia(...)`**, to reproduce the sepia look, we can use a formula in Massimiliano 2012, which is again derived from ITU-R BT.709 (ITU 2015):

$$r_{\text{sepia}} = 0.393r + 0.769g + 0.189b \tag{6}$$
$$g_{\text{sepia}} = 0.349r + 0.686g + 0.168b \tag{7}$$
$$b_{\text{sepia}} = 0.272r + 0.534g + 0.131b \tag{8}$$

The intensity of the colour red is increased and blue intensity is reduced after conversion, which is visible through the histogram in Figure 5. This is what gives sepia its distinctive warm colour tone.

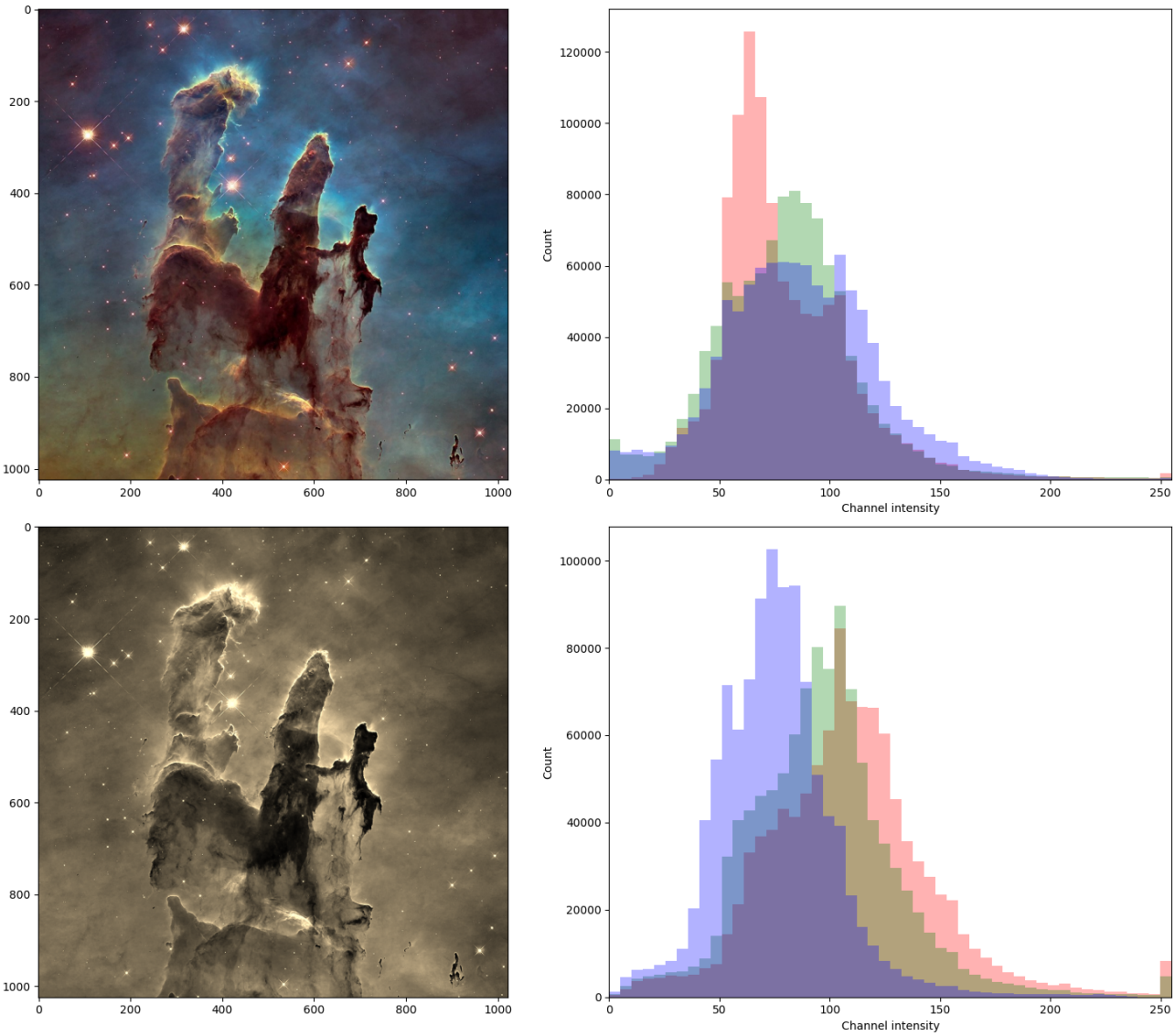

Figure 5. Output of sepia conversion (bottom), in comparison with the original *Pillars of Creation* photo.

2.5 Blurring and sharpening

The concept of image *blurring* and *sharpening* involve a technique called convolution. To summarize, it is the process of combining 2 different functions together. In the context of image processing, convolution is used to combine the original image array with a 2-dimensional kernel.

In this project, the **`blur(...)`** procedure uses the **`convolve(...)`** function with the sliding window technique visualized in Sanderson 2022 to convolve the image with a square kernel matrix. The size of this matrix is user-defined. The kernel is constructed following the 2-dimensional Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{9}$$

where σ is the standard deviation, which controls the width of the bell-shaped curve, thus, along with the kernel size, determines the blurriness of the result image.

The time complexity of this algorithm depends greatly on the kernel size and the resolution of the image because the sliding window technique works by iterating through all pixels and applying convolution individually. At the moment, I find that a kernel size of around 7 or 9 worked nicely for my

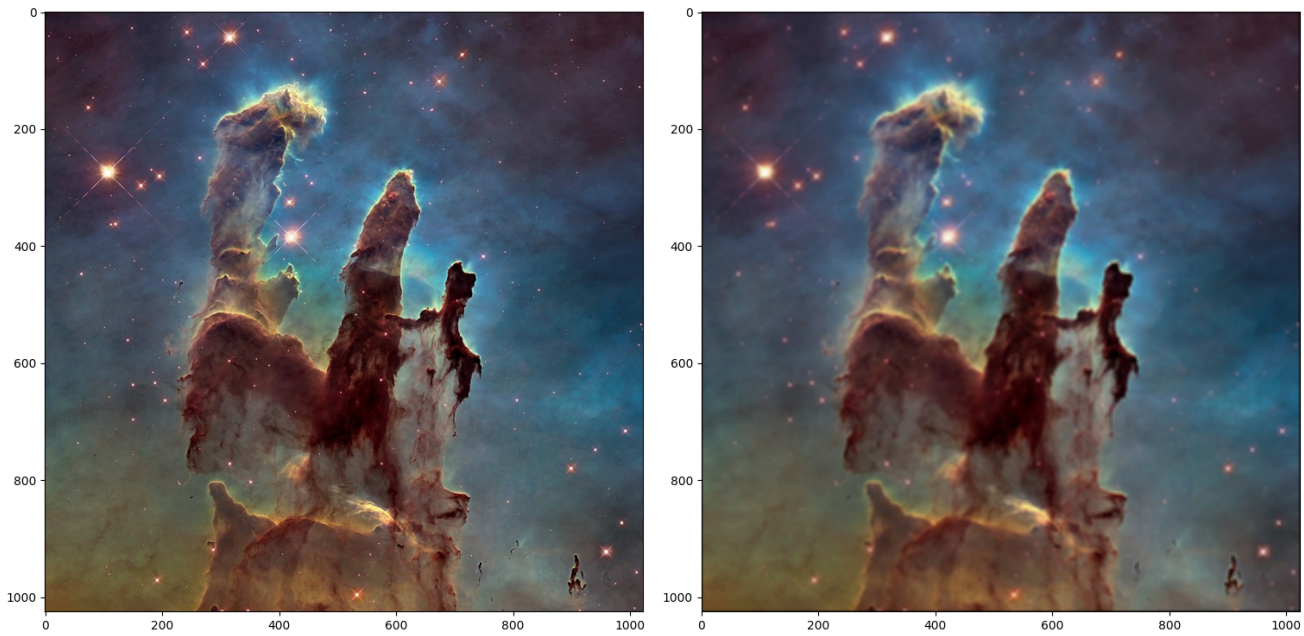

Figure 6. Result of blurring function. The original image is convolved with a 7 pixels wide 2-dimensional square Gaussian kernel, $\sigma = 50$. The convolution process finished in 10.76 seconds.

set of sample which mostly consists of images with around 1000×1000 resolution (Figure 6). It is best to choose an odd number as a kernel size for it to have a clearly defined center.

For the **`sharpen(...)`** procedure, there are many methods to implement this, including using a Laplacian kernel and repeat the same convolution process from above (Fisher et al. 2003a). However, my program uses *unsharp masking* to extract a high-pass filter by subtracting the blurred version, which is constructed with a fixed $\sigma = 100$, from the original (Fisher et al. 2003b, Cambridge 2020). This filter, which emphasizes on the presence of edges, is multiplied with an amplification amount specified by the user and added back into the original image to produce a sharpened version with enhanced edges. Figure 7 illustrates the effect of the sharpening procedure.

2.6 Cropping from the center

Cropping an image is essentially the slicing operation for arrays, which only take values of an array from within an index range and discard the rest. For the time being, the program is only able to crop from the center.

The **`crop(...)`** function works by first taking in the image and crop size entered by the user. The starting index for each axis is obtained by calculating half of the difference between the image and crop dimension. Finally, the index is used to slice the image array.

2.7 Apply circular mask

Indexing operations with boolean array is documented fairly well in *NumPy*'s guide (NumPy 2023), where a boolean array is used to filter out desired element values. Therefore, the main focus of the **`mask_cir(...)`** function is to produce a circular mask following the equation:

$$\sqrt{(x - x_{\text{center}})^2 + (y - y_{\text{center}})^2} \leq R \tag{10}$$

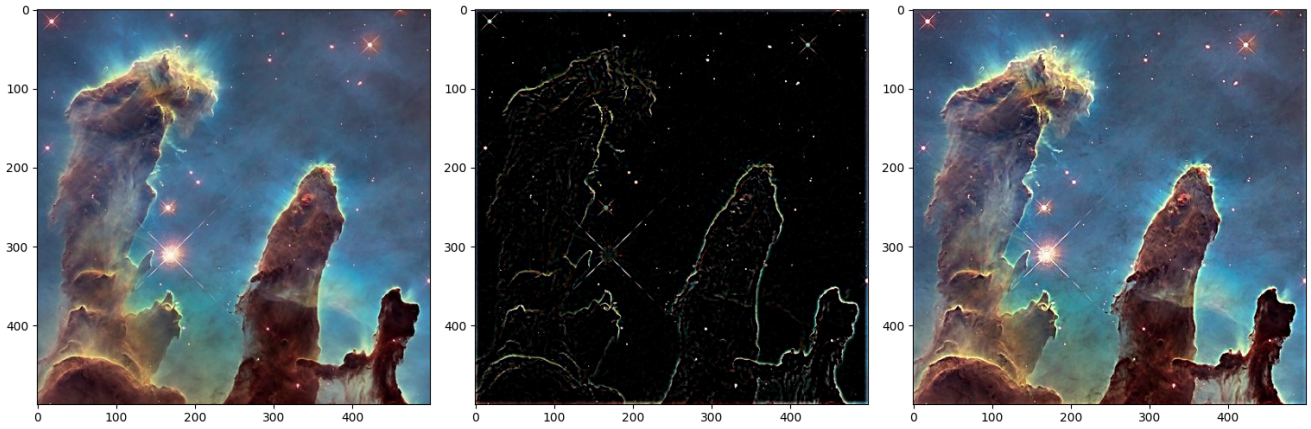

Figure 7. Result of unsharp masking on a cropped version of the *Pillars of Creation* photo to focus on the details. The center image depicts the high-pass filter with an amplification factor of 500, the brightness of the filter is increased to improve visibility. The final sharpened image is plotted on the right. The convolution process for creating a blurred version finished in 2.65 seconds.

where x_{center} and y_{center} are the coordinates of the image's center and R is the radius of the circle. In my program, the radius is calculated based on the smallest dimension of the image array.

To create the mask, the program first create a grid with matching width and height of the image using *NumPy*'s `ogrid` instance. Next, a distance matrix is calculated using the grid and the formula in Eq. 10, each element of this matrix is the Euclidean distance between that point and the center.

A mask is created by checking if each value of the distance matrix falls outside the radius. The mask is a boolean matrix, where each value can be true if the corresponding position falls outside the circle, and false if it is inside. Finally, the mask is applied by indexing the image with the boolean matrix and setting any outside pixel to the colour black. The results for this function are displayed in Section 3.

2.8 Apply elliptical mask

The **`mask_ell(...)`** function follows the same steps as in Section 2.7, with the only difference is how to establish a rotation transformation for the ellipses so that their major axes lie on the diagonals connecting 2 opposite corners of the image.

Let us first define a basic ellipse function:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1 \tag{11}$$

where a is the semi-major axis, and b is the semi-minor axis. For simplification, let us imagine that the ellipse exists on a separate plane where its axes offset the image's by an angle θ clockwise. To construct an elliptical mask where it is rotated to form a cross, we have to map the coordinates of the pixels to the ellipse's coordinate system. Figure 8 visualizes our problem: find the coordinate of a point on a rotated system of axes.

To solve the above problem and develop a suitable transformation, let us first define H as the projection of A on the y axis, and H' as the projection of A on the y' axis. We can also define the segment AH as x, OH as y, AH' as x', and OH' as y'. The OA segment divides the angle θ into 2 smaller angles α and $\theta - \alpha$. We can obtain the following systems of trigonometric equations:

$$\begin{cases} \sin\alpha = \dfrac{x}{\text{OA}} \\ \cos\alpha = \dfrac{y}{\text{OA}} \end{cases} \tag{12}$$

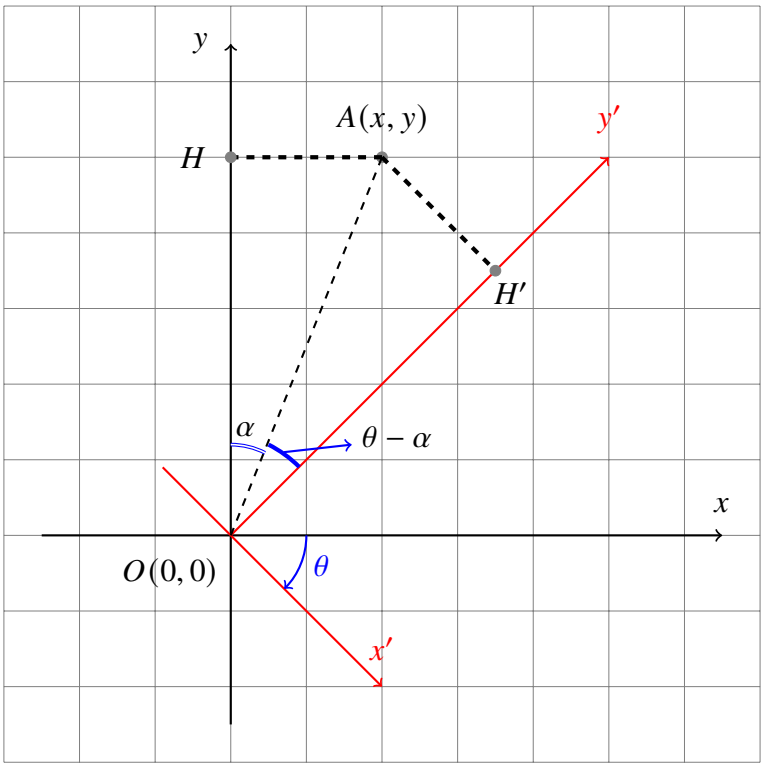

Figure 8. Simple illustration of a rotation transformation. Black axes denote the image's coordinate system (Oxy), red axes denote the ellipse's ($Ox'y'$). For example, point A have a coordinate of (x, y) on the Oxy axes. The $Ox'y'$ axes offsets Oxy by an angle θ clockwise. The problem focuses on finding the coordinate of A on the $Ox'y'$ axes.

$$\begin{cases} \sin(\theta - \alpha) = \dfrac{x'}{\mathrm{OA}} \\ \cos(\theta - \alpha) = \dfrac{y'}{\mathrm{OA}} \end{cases} \tag{13}$$

Eq. 13 can be rewritten as:

$$\begin{cases} \sin\theta\cos\alpha - \cos\theta\sin\alpha = \dfrac{x'}{\mathrm{OA}} \\ \cos\theta\cos\alpha + \sin\theta\sin\alpha = \dfrac{y'}{\mathrm{OA}} \end{cases} \tag{14}$$

Finally, we can replace some trigonometry formulas in Eq. 14 with Eq. 12, which will give us the following transformation:

$$\begin{cases} y\sin\theta - x\cos\theta = x' \\ y\cos\theta + x\sin\theta = y' \end{cases} \tag{15}$$

After performing the grid rotation via Eq. 15, the new coordinates can be passed into Eq. 11 to construct the mask through the same process as in Section 2.7. To create another intersecting ellipse, the angle is simply multiplied by -1 in the rotating process.

In my program, the angle θ is calculated from the slope formed by the height and width of the input image. The ellipse's major axis is the length of the diagonal connecting 2 opposite corners, and the minor axis is a third the length of the major axis. Results for the transformation and mask will be presented in Section 3.

3 RESULTS

The sample images in this project are the same as in the first project (Figure 9, top row). Each figure bellow will display the result images of each processing function. The images are cropped into a square to fit in the plots more nicely. Figures with necessary remarks will be presented first.

Figure 9. The original images (top row) along with brightened outputs (middle row) with a factor 2, and darkened (bottom row) with factor 0.6. The outputs are produced by the brightness adjustment function.

Figure 9 clearly illustrated the *bias* in intensity when using the multiplicative method, most noticeable in the *Crab Nebula* (middle) image. Brighter values in the center are enhanced tremendously while the black background remains the same.

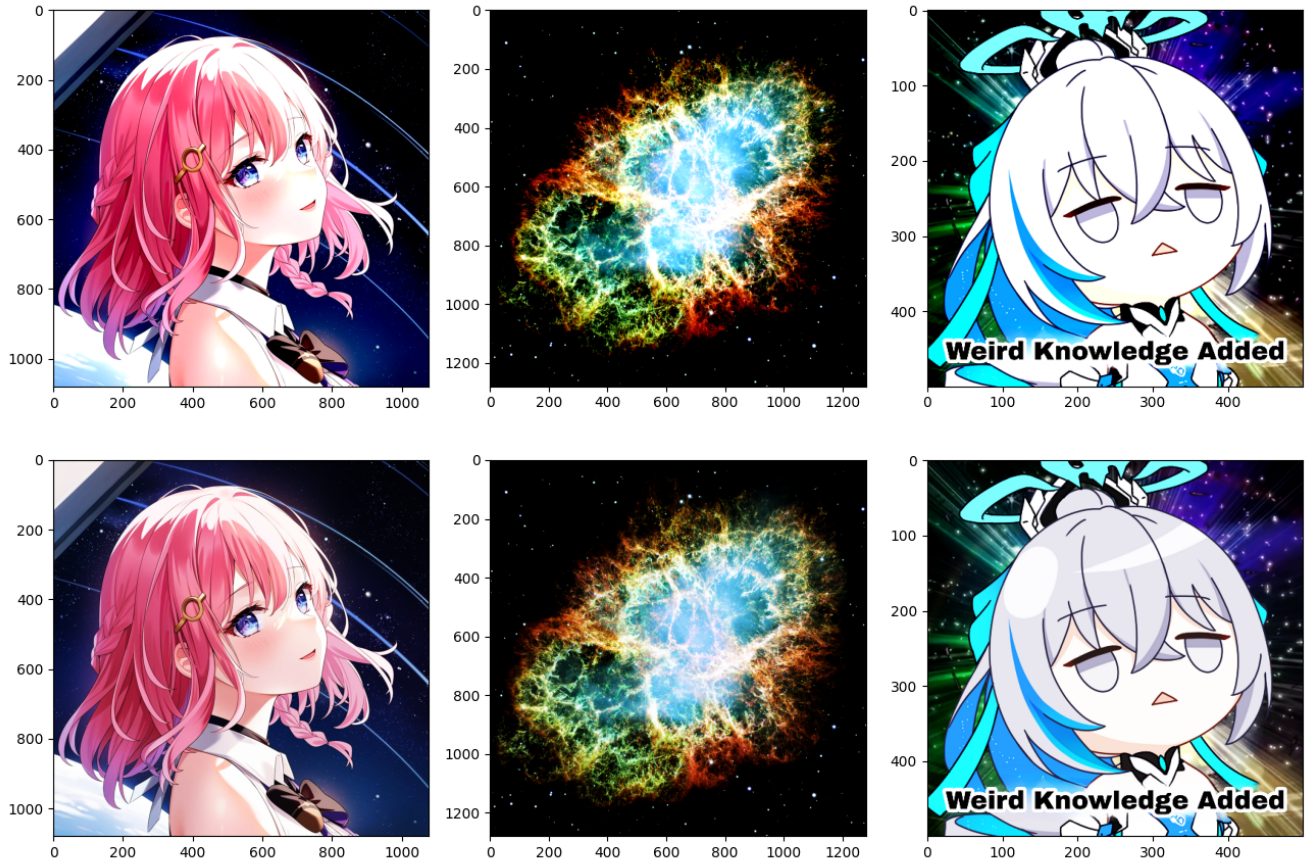

Figure 10. Results of contrast enhancement with multiplicative method (top row) and power-law method (bottom row). All runs are configured with an enhancement factor of 2.

For the contrast adjustment algorithm, both methods (multiplicative and power-law) achieved nearly the same effect without any noticeable difference. In Figure 10, the power-law configuration managed to retain some level of the detail at higher intensities. This is due to the power-law function having a smooth and continuous curve. Because of this, values near the boundaries 0 and 255 are converged slowly towards the edges instead of abruptly jumping out of range as in the linear function.

However, I believe a minor downside of the S-curve is visible in Figure 11, where contrast is reduced instead of enhanced. Because of the easing effect the curve had on the boundaries and midpoint, values around these parts receive little to no change when compared with the original values. Thus, for every maximum (255) and minimum (0) value of each channel, instead of moving closer to the middle of the range, the values become more distinct, especially in pictures with fairly uniform colour distribution (right most image). Therefore I believe that power-law work best in enhancing the contrast, and the linear function should be prioritized when performing contrast reduction.

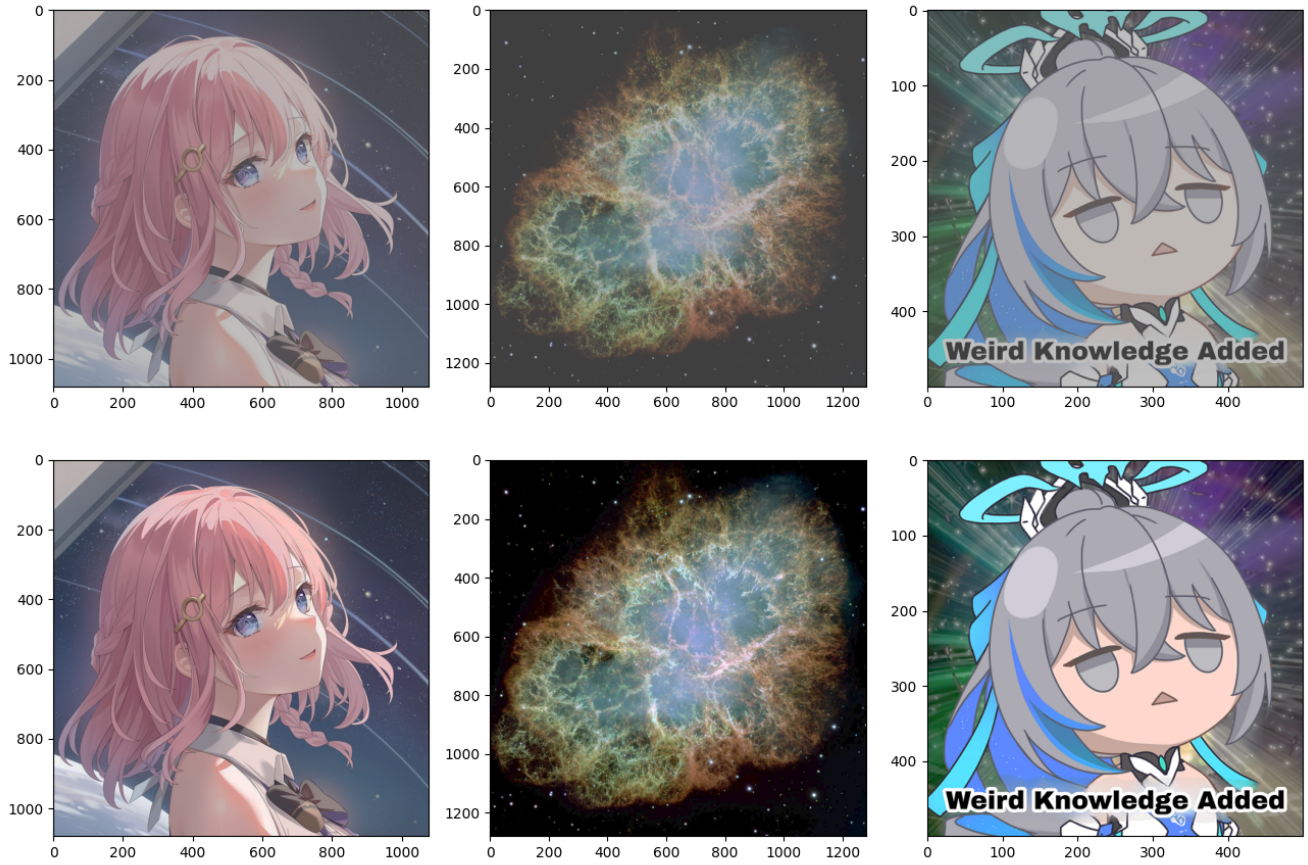

Figure 11. Results of contrast reduction with multiplicative method (top row) and power-law method (bottom row). All runs are configured with a reduction factor of 0.5. The undesirable effect of S-curve is visible for the left and rightmost output, where pixels with higher intensity in each channel stood out from the rest.

Figure 12. Results of the blurring process (top row) when convolved with a kernel size of 11 pixels and $\sigma = 100$. The results of the sharpening procedure (bottom row) and the original input image (middle row) are cropped to the resolution of 300×300 using the `crop(...)` function in Section 2.6 to better observe the enhanced edges. The sharpening run utilizes the same kernel as the blurring process, with an amplification factor of 500. The times taken for blurring process are (from left to right): 13.19s, 19.08s, 2.92s. The times taken for sharpening at full scale are: 13.82s, 19.32s, 2.93s.

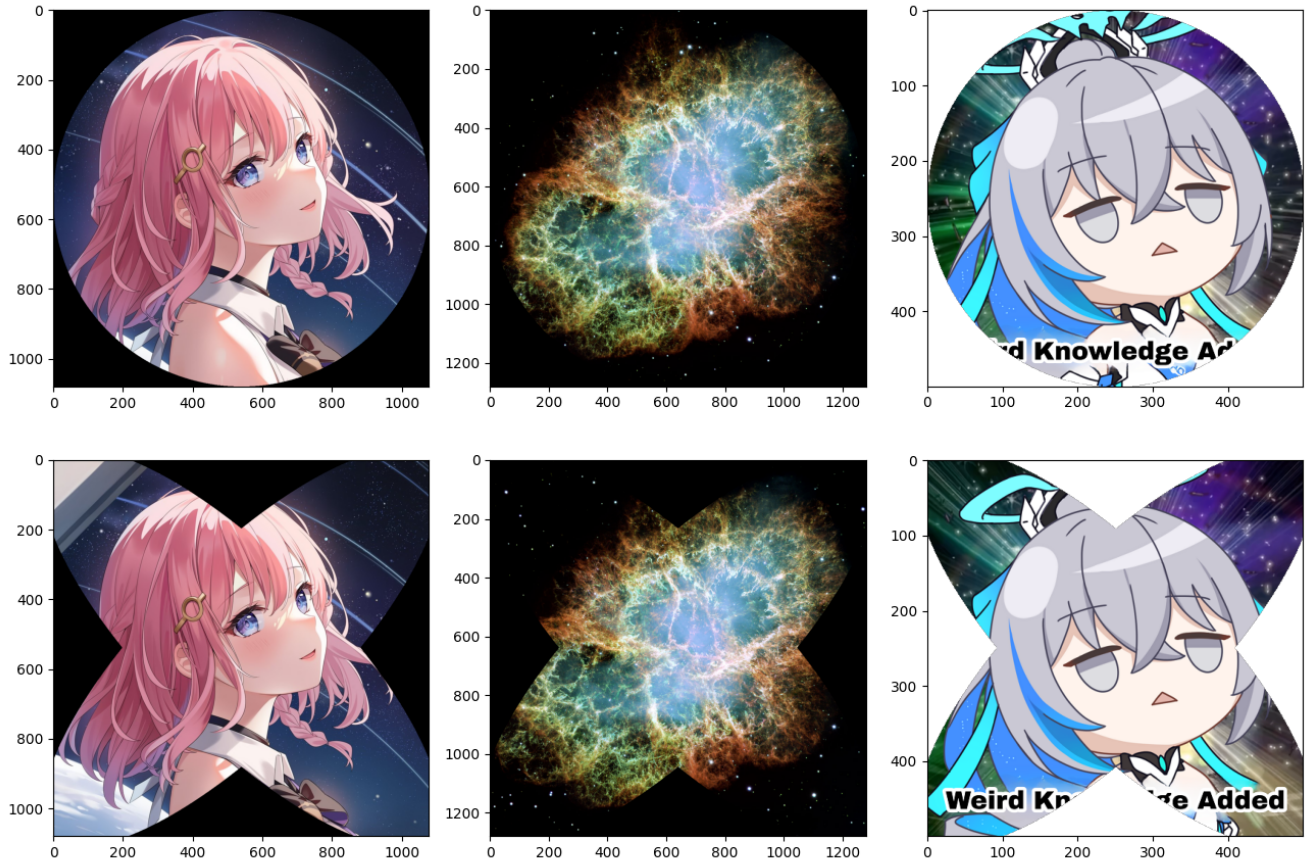

Figure 13. Results of circular masking (top row) and cross elliptical masking (bottom row). The method for creating the circular as well as rotating the ellipse is mentioned in Section 2.7 and 2.8. For the image on the right, due to it being a PNG image with a transparent background, when the mask is applied, the background will become visible through certain sections.

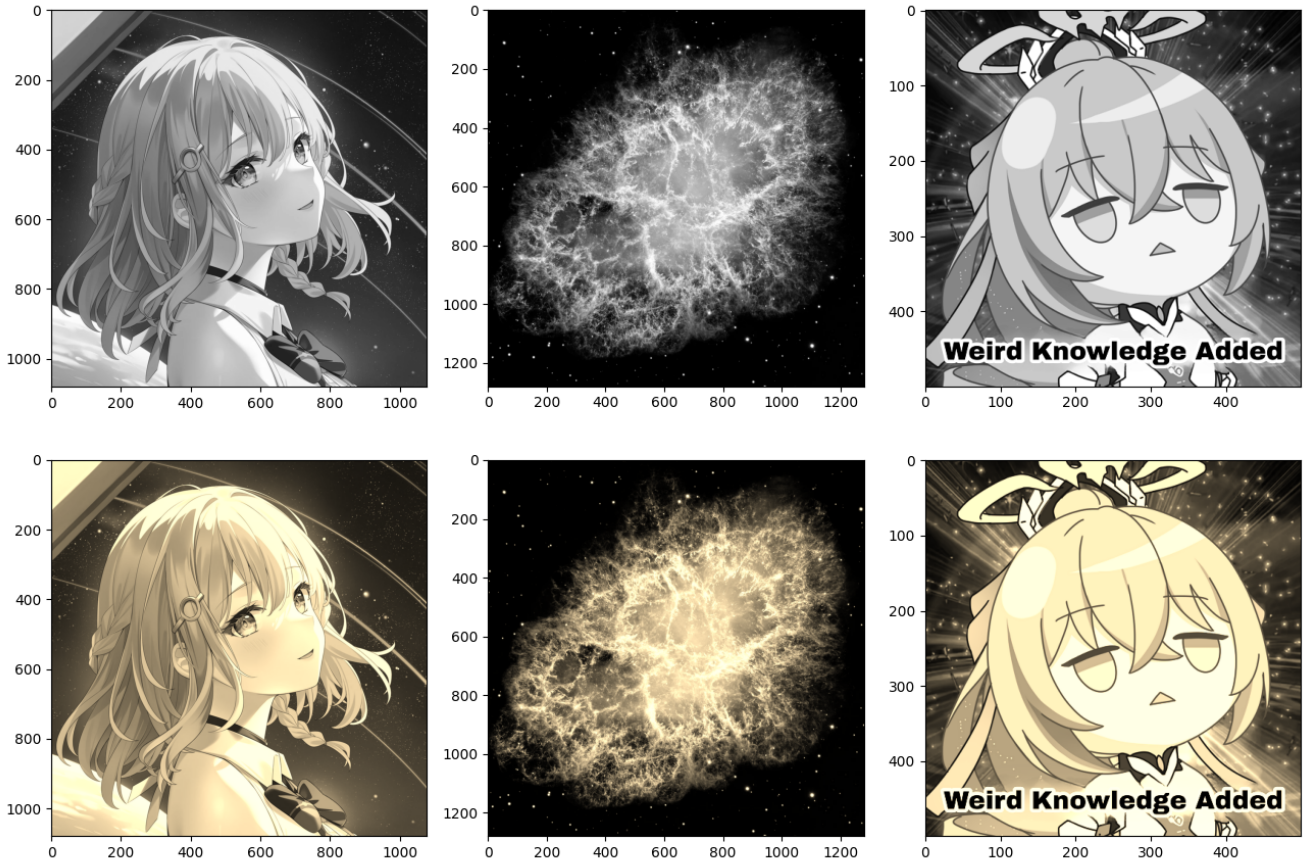

Figure 14. Results of grayscale and sepia conversion on the sample images. The function receives no other argument other than the image, therefore the results for each conversion are the same for every input.

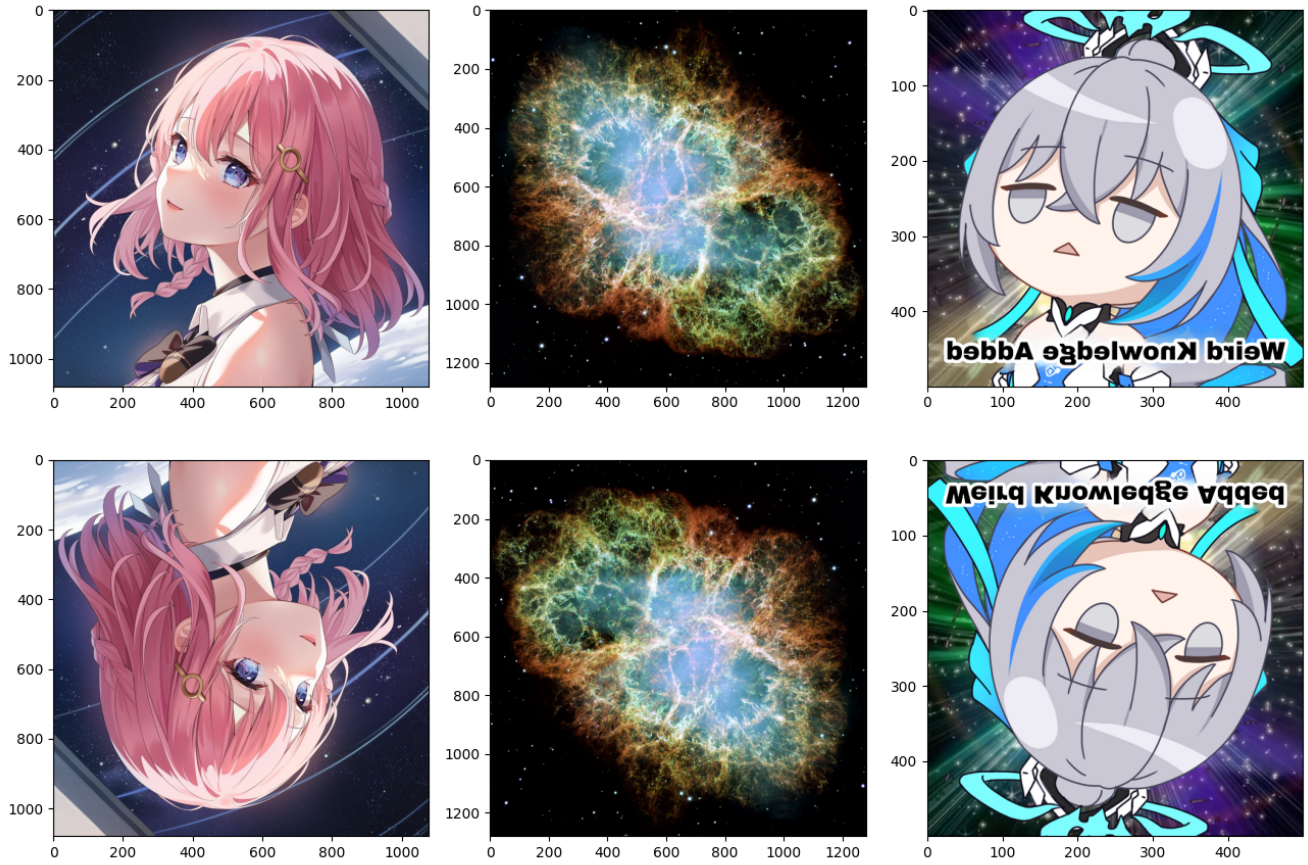

Figure 15. Results of horizontal flipping (top) and vertical flipping (bottom).

ACKNOWLEDGEMENTS

I would like to thank the lecturers of the *Applied Mathematics and Statistics* course at the Ho Chi Minh University of Science (Nguyen Dinh Thuc, Tran Ha Son, Nguyen Van Quang Huy, and Ngo Dinh Hy) for their instruction in theory and lab session as well as support during the project duration.

The testing samples used in this project are products of NASA/ESA Hubble Space Telescope, Twitter user @hitsukuya, miHoYo Co. Ltd. and unknown creator of an Internet *meme*. All data were used for educational and research purposes.

The structure and style of this report is based on the Monthly Notices of the Royal Astronomical Society (MNRAS) template and instructions for authors (https://academic.oup.com/mnras/).

Software: python 3.10 (https://www.python.org/), numpy 1.25 (https://numpy.org/), matplotlib 3.6.0 (https://matplotlib.org/), pillow 9.2 (https://python-pillow.org/).

REFERENCES

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. 2020, Nature, 585, 357–362
J. D. Hunter 2007, Computing in Science & Engineering, 9, 90-95
Mark Ransom 2012, Stack Overflow, "Algorithm to modify brightness for RGB image?"
Timothy Schulz 2012, YouTube, Timothy Schulz, "Contrast Transformation"
Wikipedia Contributors 2023, Wikipedia, "Grayscale"
Dynamsoft 2019, Dynamsoft, "Image Processing 101 Chapter 1.3: Color Space Conversion"
Ellen Fisch, Gregory Ballos 2023, Adobe, "Sepia Photography"
Massimiliano 2012, Stack Overflow, "How is a sepia tone created?"
Grant Sanderson 2022, YouTube, 3Blue1Brown, "But what is a convolution?"
R. Fisher, S. Perkins, A. Walker and E. Wolfart 2003a, HIPR, "Laplacian/Laplacian of Gaussian"
R. Fisher, S. Perkins, A. Walker and E. Wolfart 2003b, HIPR, "Unsharp Filter"
Cambridge in Colour 2020, Cambridge in Colour, "Sharpening: Unsharp Mask"
NumPy Developers 2023, NumPy v1.25 Manual, 'Indexing on ndarrays"