

--	--	--

# Clab-1 Report

ENGN6528

Shreya Chawla  
u7195872

28/03/2021

--	--	--

### **Task-1: Matlab (Python) Warm-up. (2 marks):**

Describe (in words where appropriate) the result/function of each of the following commands of your preferred language in report. Please utilize the inbuilt help() command if you are unfamiliar with these functions.

Note: Different from Matlab, Python users need to import external libraries by themselves. And we assume you already know some common package abbreviations (e.g. numpy = np). [You only need to complete one set of questions either in matlab or Python.] (0.2 marks each)

#### **Python (Delete this part if you use Matlab)**

**(1) `a = np.array([[2, 3, 4],[5, 2, 200]])`**

A 2D np array of dimensions 2 x 3 (row x column) is stored in variable **a**.

**(2) `b = a[:, 1]`**

A slicing operation is performed on array a. All the rows (because of “:”) of column 1 (2<sup>nd</sup> column – in python the numbering starts from 0) are returned. Thus b stores all the elements of column of index 1 (i.e. 2nd column) of array a.

**(3) `f = np.random.randn(400,1)+3`**

This function returns a floating-point array shaped (400,1) of random values and each element of array is incremented by 3. Basically, a Gaussian distribution with  $\mu = 3$  and  $\sigma = 1$  (since the coefficient of `np.random.randn` = 1 here) in a column array is returned and stored in f np array.

**(4) `g = f[f > 0]*3`**

For all values in f array if they are greater than 0 then they are multiplied by 3 and only the multiplied values are stored in 1D array g.

**(5) `x = np.zeros(100) + 0.45`**

This function initializes x array with (100,) shaped array having only values 0s and each value is added by 0.45. In conclusion, x stores a (100,1) array with value of 0.45 each.

**(6) `y = 0.5 * np.ones([1, len(x)])`**

This function initializes y array with (1,100) shaped column array having only values 1s and each value is multiplied by 0.5. (100 because `len(x) = 100`) In conclusion, y stores a (1, 100) column array with value of 0.5 each.

**(7) `z = x + y`**

--	--	--

In z, x and y arrays are added, resulting in an array of dimension (1,100) each having value 0.95 (as  $0.45 + 0.5 = 0.95$ ).

**(8) `a = np.linspace(1,499, 250, dtype=int)`**

This function returns 250 evenly spaced numbers in an np array over the specified interval of 1 to 499 (inclusive by default) of type int. Here, due to choice of numbers, all numbers returned are odd integers in the range and stored in a.

**(9) `b = a[::-2]`**

Starting from the end of array a, all elements at a step of 2 (skipping one value) are stored in b. This can be useful to take a stride when convolving images.

**(10) `b[b > 50] = 0`**

This replaces the elements of b array which have value greater than 50 by 0 and stores this array in b itself.

## **Task-2: Basic Coding Practice (1 marks)**

Write functions to process an input grayscale image with following requirements, where you need to write a script to load the given image in the Lab package, apply each transformation to the input, and display the results in a figure. For Matlab, you can use subplot() function; for Python, we suggest to use matplotlib.pyplot.subplot(). Please notice that each subplot needs to be labelled with an appropriate title. (0.2 marks each)

**1. Load a grayscale image, and map the image to its negative image, in which the lightest values appear dark and vice versa. Display it side by side with its original version.**

The grayscale image “Atowergray.jpg” is read with cv2.imread() and then negative of image is found by simply subtracting it from 255 (as image dtype is np.uint8). Upon subtracting, the dark regions in image like the Eiffel tower become white and vice versa for the bright sky becoming dark. The negative image has been displayed in Fig 1 (right) alongside the original image (left) for comparison.

--	--	--

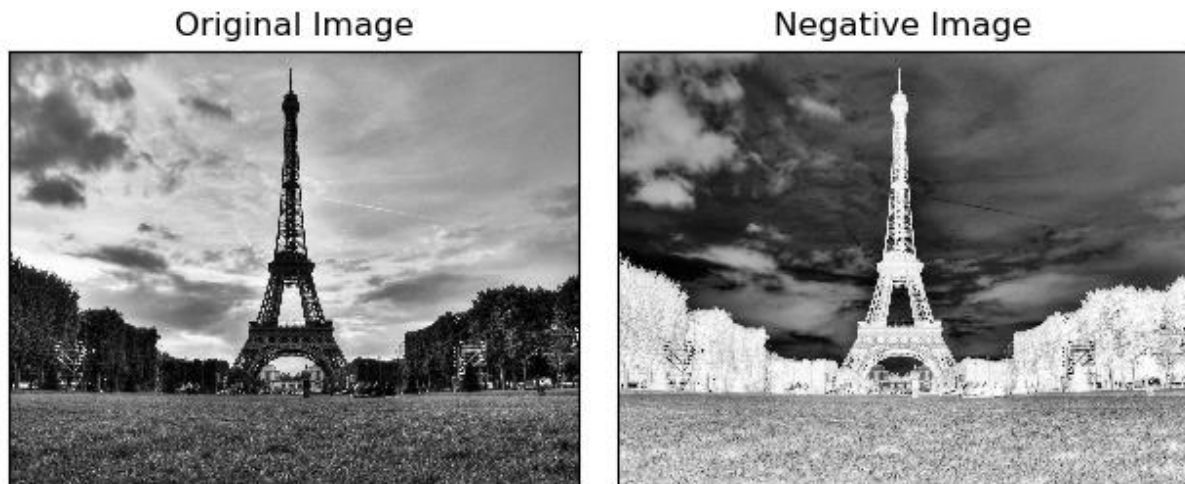


Fig 1: Comparing original grayscale image (left) and its negative (right)

## 2. Flip the image horizontally (i.e, map pixels from right to left changed from left to right).

The grayscale image “Atowergray.jpg” is flipped horizontally using the `np.flip()` method and displayed alongside original image for comparison in Fig 2.



Fig 2: Comparing original image and its horizontally flipped image that is its mirror image

## 3. Load a colour image, swap the red and green colour channels of the input.

Now “image1.jpg” is read using the same `cv2.imread()` method as above by specifying its address. The image is first converted to RGB format from the BGR using `cvtColor()` function. Next the image color channels are obtained using `cv2.split()` method and only R and G channels are to be used for this task. The color channels – red and green – are then swapped. This has been displayed in the Fig 3 below.

--	--	--

Original Image



Channel Swapped Image



Fig 3: Comparing original image (left) and image with its red and green channels swapped (right)

#### **4. Average the input image with its horizontally flipped image (use typecasting).**

The original image (Fig 2 left) is added to the image from task 1.2 (Fig 2 right) and then divided by 2 to get the average of the images. Next it is typecasted as int as division converted the values into float point. Fig 4 displays the averaged image. As can be seen, there are a lot of regions with dark or black regions. To correct this, the images' arrays can first be divided by 2 and then added. The added pixel values in this new image will not exceed the limit of [0,255].

--	--	--

Average of original and horizontally flipped images



Fig 4: Averaging the original grayscale image and its horizontally flipped image from Task 1.2 and then clipping using numpy

**5. Add a random value between [0,127] to every pixel in the grayscale image, then clip the new image to have a minimum value of 0 and a maximum value of 255. (Note: the intensity values of the original grayscale image range from 0 to 255.)**

To add random values, the `np.random.randint()` function is used. The range of noise values [0, 127] and the image dimensions are passed as parameters to get a new np array noise. This noise array is added to the image `im` to get `im_noisy`. The `im_noisy` is then clipped using `np.clip` array by specifying the min and max as 0 and 255 respectively. The noisy image is displayed as Fig 5. A lot of salt and pepper noise can be observed in the Fig 5 due to addition of random values to each pixel. Clip makes sure that the range of values does not exceed the `np.uint8` limit.



--	--	--

Noisy image



Fig 5: Adding random noise to the image of value [0,127] to each pixel and then clipping using numpy

### Task-3: Basic Image I/O (2 marks)

Note: You need to download the image1.jpg, image2.jpg, image3.jpg from wattle.

In this task, you are asked to:

1. Using image1.jpg, develop short computer code that does the following tasks:

**a. Read this image from its JPG file, and resize the image to 384 x 256 in columns x rows (0.2 marks).**

Image image1.jpg is first loaded using the `imread()` method and then resized to 384 x 256 (col x row) using the `resize()` method and stored in `im_scaled`. Then the `im_scaled` image is converted to RGB from the cv2 default BGR and then displayed without x and y ticks. The scaled image is displayed as Fig 6. It is blurrier than the original image due to loss of pixels.

--	--	--

Scaled image (384 x 256)



Fig 6: Resized RGB image to (384 x 256)

**b. Convert the colour image into three grayscale channels, i.e., R,G,B images, and display each of the three channel grayscale images separately (0.2 marks for each channel, 0.6 marks in total).**

The image is split using the `cv2.split` function into BGR values according to cv2 default color scheme. The 3 obtained channels are displayed side by side as subplots using matplotlib's `imshow` and `plot` functions with `cmap` parameter as "gray" as shown in Fig 7. The intensity of each channel is different and together as merged they form the color image.



Fig 7: Viewing the RGB channels of image as grayscale images: Blue (left), Green (middle), Red (right)



--	--	--

**c. Compute the histograms for each of the grayscale images, and display the 3 histograms (0.2 marks for each histogram, 0.6 marks in total).**

The histogram for the 3 color channels of the image are plotted using cv2's calcHist function and the parameters - the image color channel (R, G, or B), the channel it represents (1,2, or 3), with no mask, histogram range and size are passed. They are displayed using matplotlib library's subplot, and plot functions as shown in Fig 8. The histograms are colored according to color channel they represent.

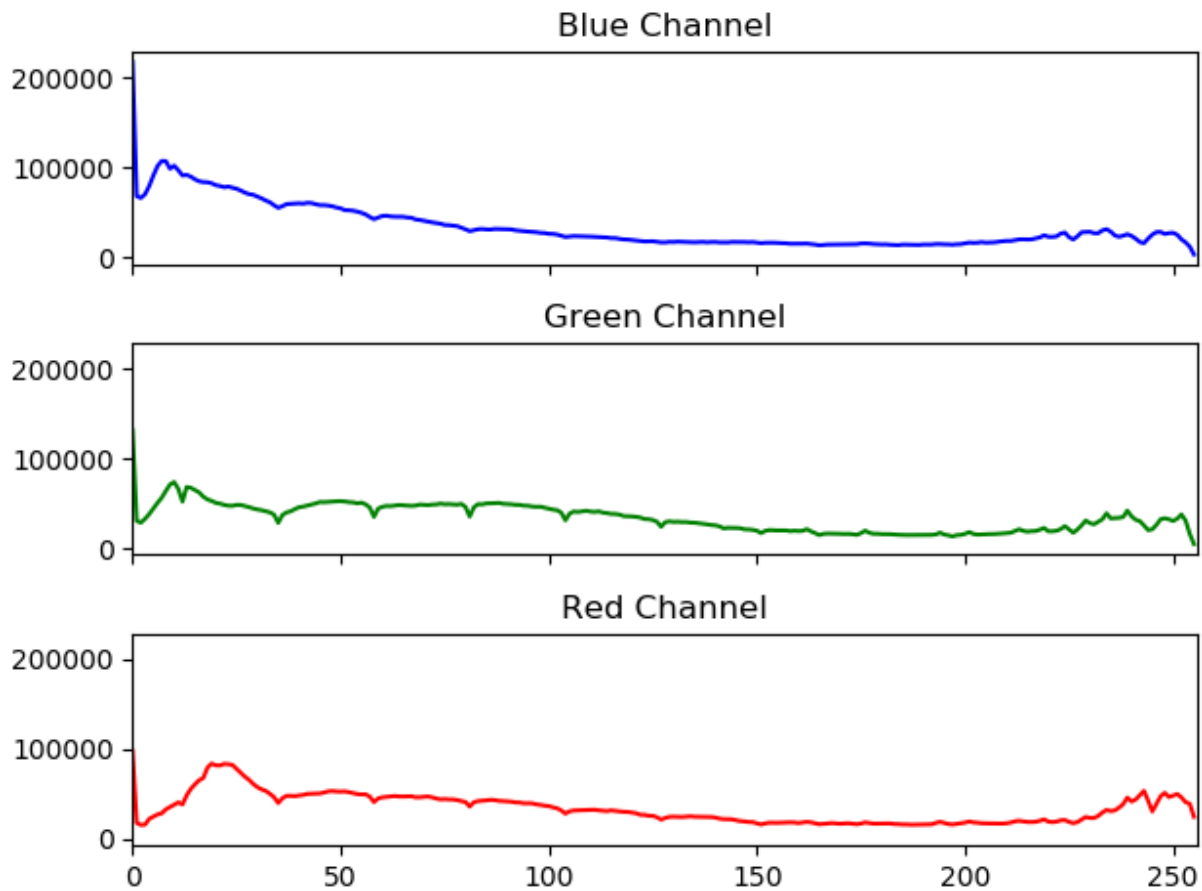
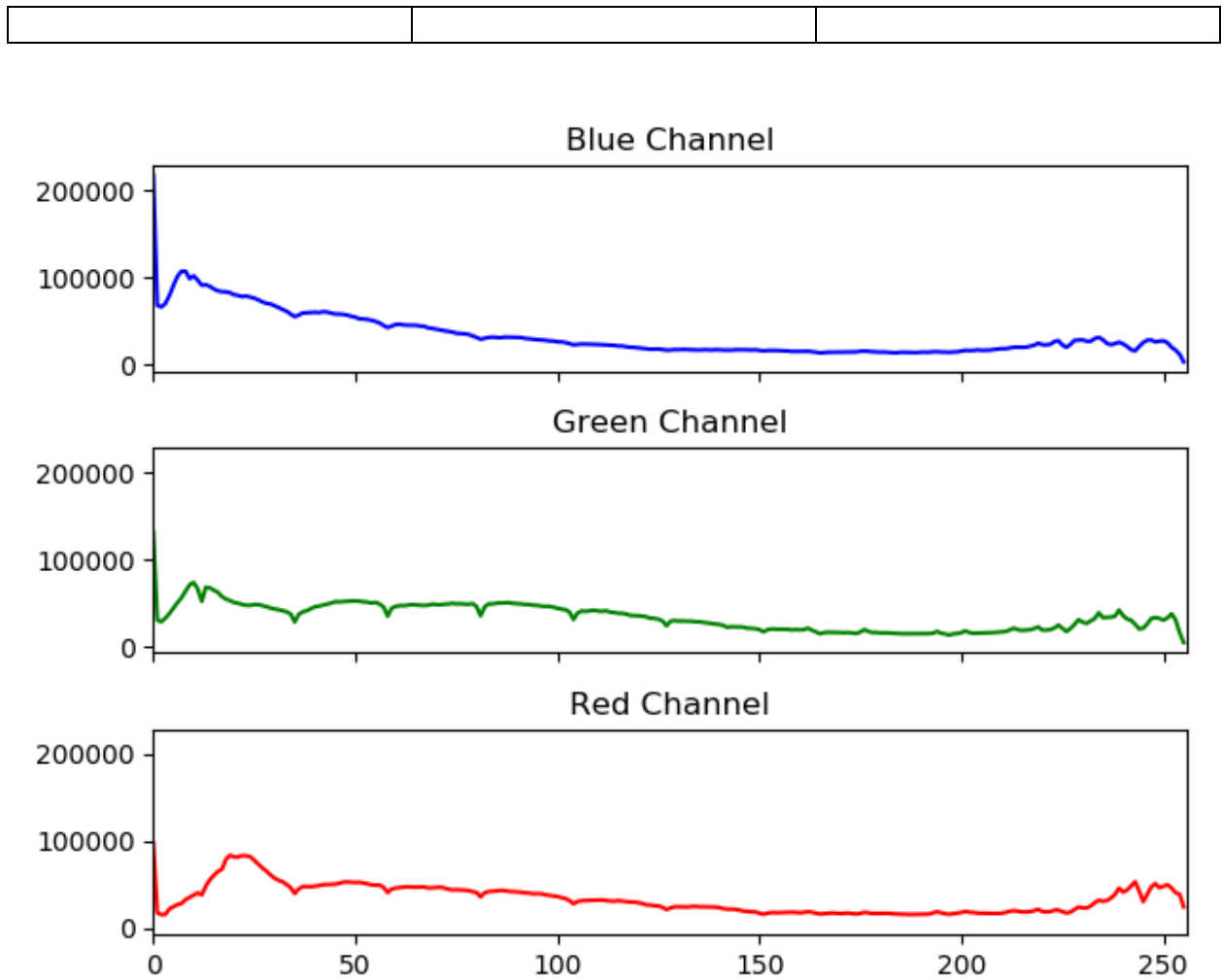


Fig 8: Plot of histogram for the RGB channels of image: Blue (top), Green (middle), Red (bottom)



**d. Apply histogram equalisation to the resized image and its three grayscale channels, and then display the 4 histogram equalization image (0.15 marks for each histogram, 0.6 marks in total). (Hint: you can use inbuilt functions for implementing histogram equalisation. e.g. `histeq()` in Matlab or `cv2.equalizeHist()` in Python).**

The color channels from Task 3 (b) are equalized to adjust contrast in image thus increasing the global contrast as can be seen in Fig 9 which displays the equalized histograms and Fig 10 from which the difference in contrast can be seen. For this task, `cv2` library's `equalizeHist` has been used.

--	--	--

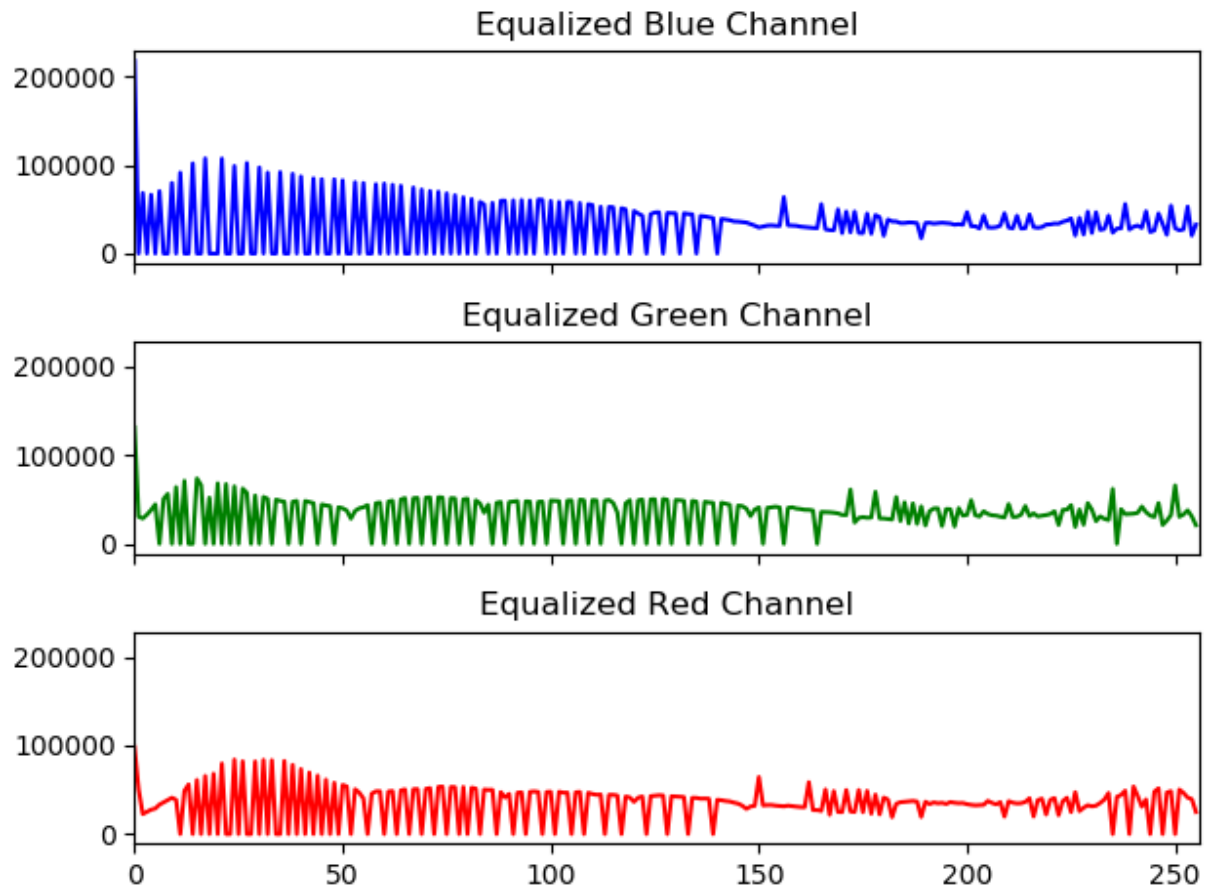
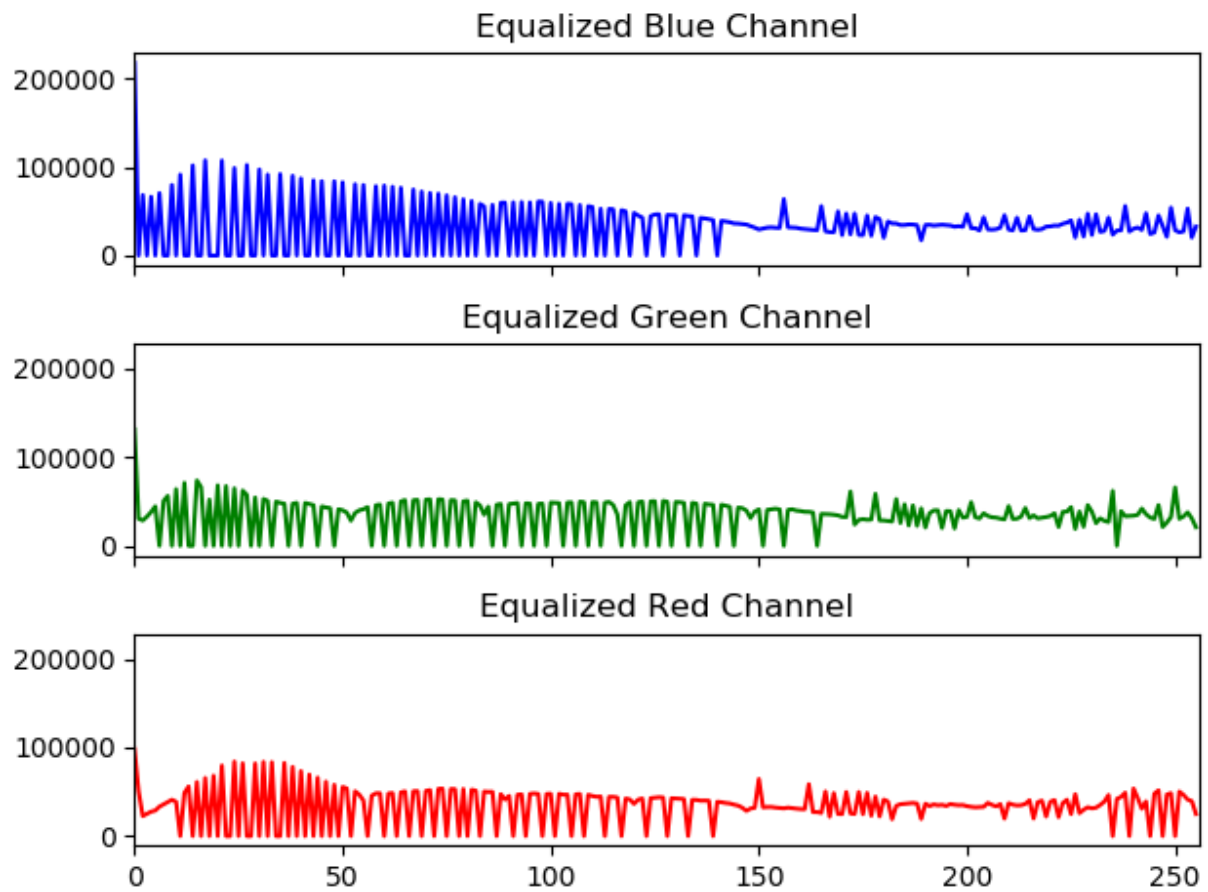


Fig 9: Plot of equalized histogram for the RGB image channels: Blue (top), Green (middle), Red (bottom)

--	--	--





--	--	--

Equalized Blue Image



Equalized Green Image



Equalized Red Image



Fig 10: Equalized RGB channels of image as grayscale images: Blue (top), Green (middle), Red (bottom)

The scaled grayscale image is converted to YUV color space using the `cv2.cvtColor()` method. The intensity values (Y channel) is then equalized using the `equalizeHist()` method of `cv2`. Then the histogram is calculated by the `calcHist()` method and displayed with the equalized image as subplots as shown in Fig 11.

--	--	--

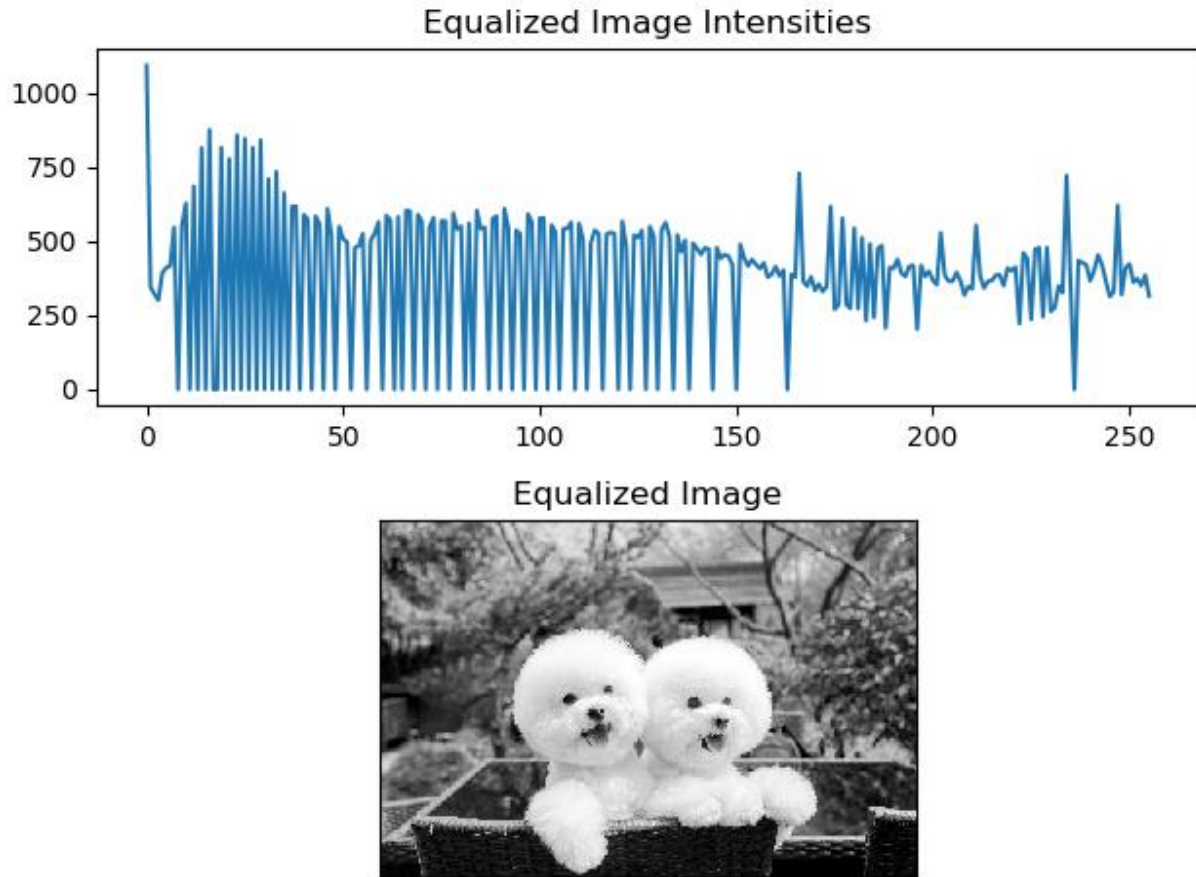
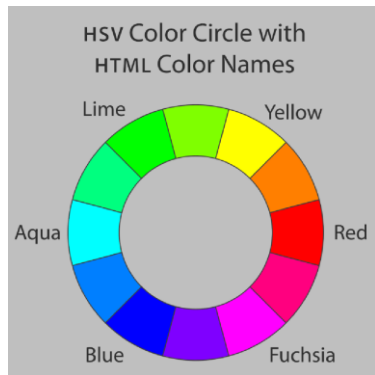


Fig 11: Equalized intensity values histogram of image (top), corresponding equalized image (bottom)

#### Task-4: Colour space conversion (3 marks)

Use the two images in Fig.2 to study colour space conversion from RGB to YUV (you can download them from the Wattle site):

--	--	--



Fig, 2 (a) Colour Wheel

(b) Palettes

**1. Based on the formulation of RGB-to-YUV conversion, write your own function `cvRGB2YUV()` that converts the RGB image to YUV colour space (0.7 marks). Read in Fig.2(a) and convert it with your function, and then display the Y, U, V channels in your report (0.1 marks for each channel, 0.3 marks in total).**

The `myRGB2YUV()` function has been defined for this task which takes image `im` in RGB color space as input and outputs the Y, U, and V channels of image. To find the channels, the formula below is used (Fig 12) (given in Piazza post). First the image was converted to RGB then passed to the function. This was done because the image loaded was in BGR color space because it was read using `cv2`. In the function, the R, G, and B channels are first split and then the below formula was applied.

BT.601 defines the following constants:

$$\begin{aligned} W_R &= 0.299, \\ W_G &= 1 - W_R - W_B = 0.587, \\ W_B &= 0.114, \\ U_{\max} &= 0.436, \\ V_{\max} &= 0.615. \end{aligned}$$

Y'UV is computed from RGB (linear RGB, not gamma corrected RGB or sRGB for example) as follows:

$$\begin{aligned} Y' &= W_R R + W_G G + W_B B = 0.299R + 0.587G + 0.114B, \\ U &= U_{\max} \frac{B - Y'}{1 - W_B} \approx 0.492(B - Y'), \\ V &= V_{\max} \frac{R - Y'}{1 - W_R} \approx 0.877(R - Y'). \end{aligned}$$

The resulting ranges of Y', U, and V respectively are  $[0, 1]$ ,  $[-U_{\max}, U_{\max}]$ , and  $[-V_{\max}, V_{\max}]$ .

Fig 12: Formula to convert RGB image to YUV color space

Image Fig.2(a) from Wattle is passed through the function to convert image into YUV color space. Fig 13 (a) is the result of `myRGB2YUV()` which is very similar visually to Fig 13 (b), the result of `cv2` library's in-built function.

--	--	--

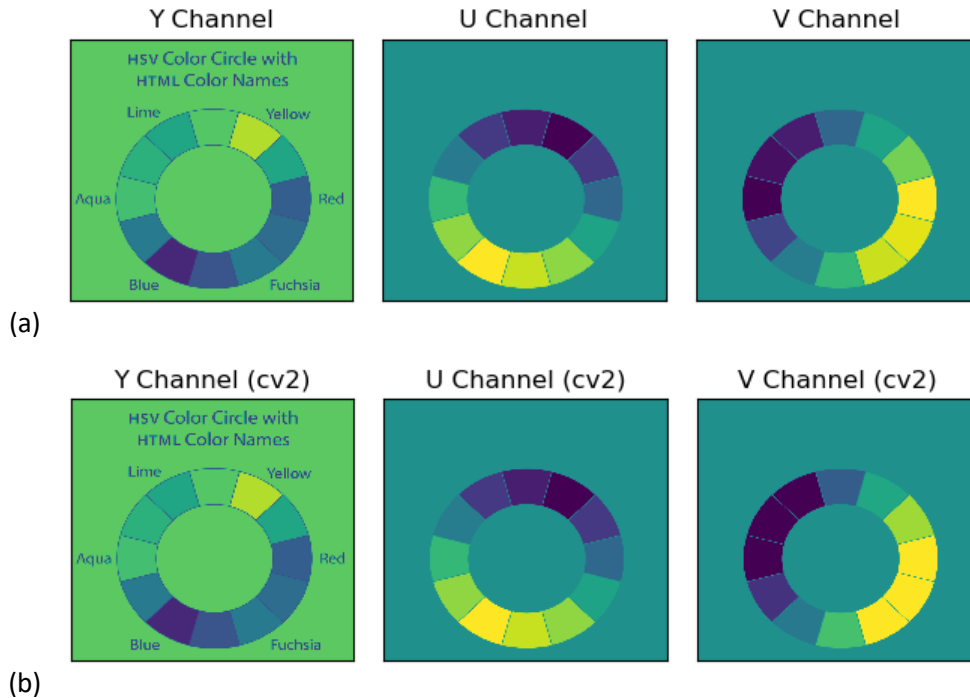


Fig 13: The YUV color space of image obtained from (a) myRGB2YUV function (top): Y channel (left), U channel (middle), V channel (right); (b) cv2 library's function (bottom): Y channel (left), U channel (middle), V channel (right)

**2. Compute the average Y values of five colour regions in Fig. 2(b) with your function and the Matlab's inbuilt function `rgb2yuv()`. Print both of them under the corresponding regions (0.1 marks for each value, 0.5 marks in total). You also need to explain how to distinguish and divide the five regions, and how to calculate the average Y value (1.5 marks, higher marks only for a smarter solution). (Note: The elements of both colormaps are in the range 0 to 1.).**

This task requires the 5 regions to first be identified and then find average Y for each region. To find the 5 regions, edges have to be found for which Canny filter has been used. This results in image below (Fig 14) stored in `im_edge` in the code "Task 4.py". This `im_edge` acts as a mask for the 5 regions to be identified. A for loop over one of the `im_edge` image's rows gives the locations for all the regions' start and end points. When an edge is detected, the next pixel in the same row would belong to the next region. Now, to find the average Y, when looping over a row in the image `im_edge`, we can simply take the value of intensity (in `im_y`) of pixel right after edge (by using location information from `im_edge`). Since for each region, the intensity Y would remain constant hence taking that value or averaging over whole region would give same results. These results are then printed.



--	--	--

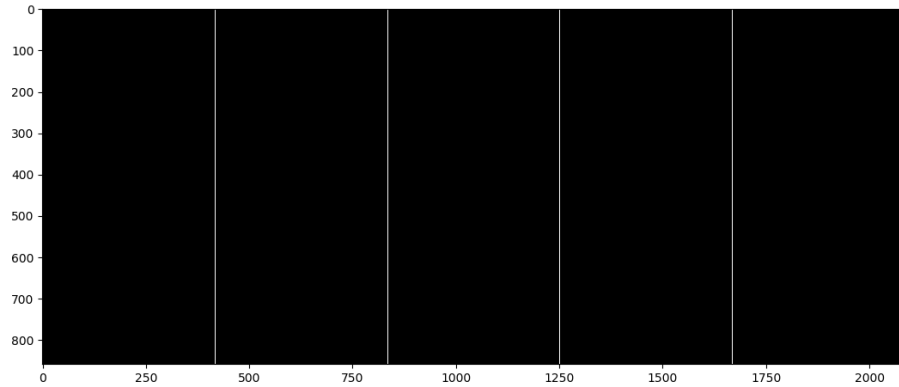


Fig 14: Edges detected from Canny filter on image Fig.2(b)

The average Y values of five color regions in image are:

1. Result of using myRGB2YUV function defined for Task 4.1:  
[116.245, 144.95900000000003, 232.51299999999998, 174.0, 122.517]
2. CV2 library's result:  
[116, 145, 233, 174, 123]

The values in the results are very close and hence the function cvRGB2YUV can be concluded to be working correctly.

### Task-5: Image Denoising via a Gaussian and Bilateral Filter (9 marks)

**1. Read in image2.jpg. Crop a square image region corresponding to the central part of the image, resize it to 512×512, and save this square region to a new grayscale image. Please display the two images. Make sure the pixel value range of this new image is within [0, 255]. Add Gaussian noise to this new 512x512 image (Review how you generate random number in Task-1). Use Gaussian noise with zero mean, and standard deviation of 15.**

**Hint:** Make sure your input image range is within [0, 255]. Kindly, you may need `np.random.randn()` in Python. While Matlab provides a convenient function `imnoise()`. Please check the default setting of these inbuilt function.

**Display the two histograms side by side, one before adding the noise and one after adding the noise (0.5 marks).**

The image is first cropped then reshaped and finally converted to grayscale as shown in Fig 16

--	--	--

Original image



Cropped, Resized and Grayscale image



Fig 15: The image is converted and displayed

The new image is then made noisy by adding gaussian noise with mean 0 and standard deviation of 15 by using this code snippet:

```
sd * np.random.randn(row, col) + mean
```

This image is displayed as Fig 16 below.

--	--	--

Gaussian noise in image (mean=0, sd=15)

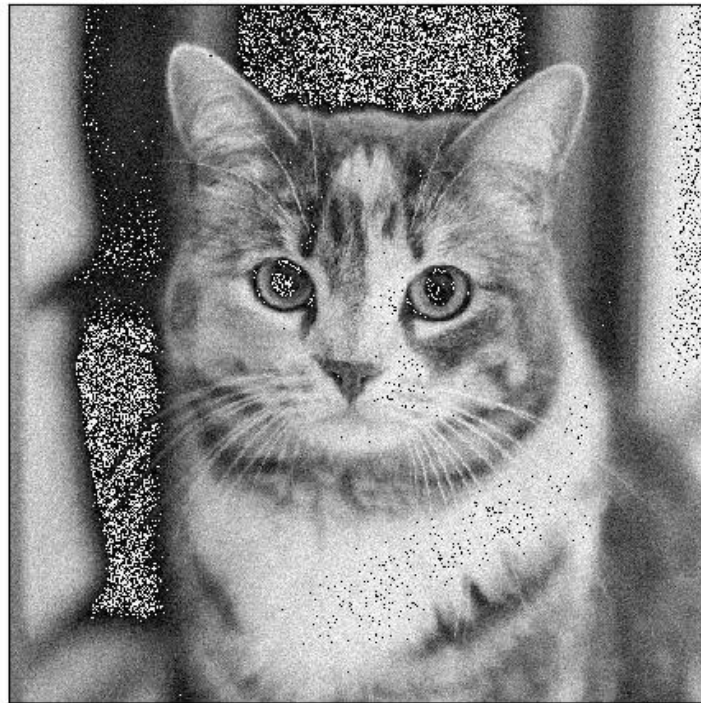



Fig 16: The image after adding Gaussian noise

Next, the intensities of the pixels of the grayscale image before and after adding noise are shown in Fig 17 

--	--	--

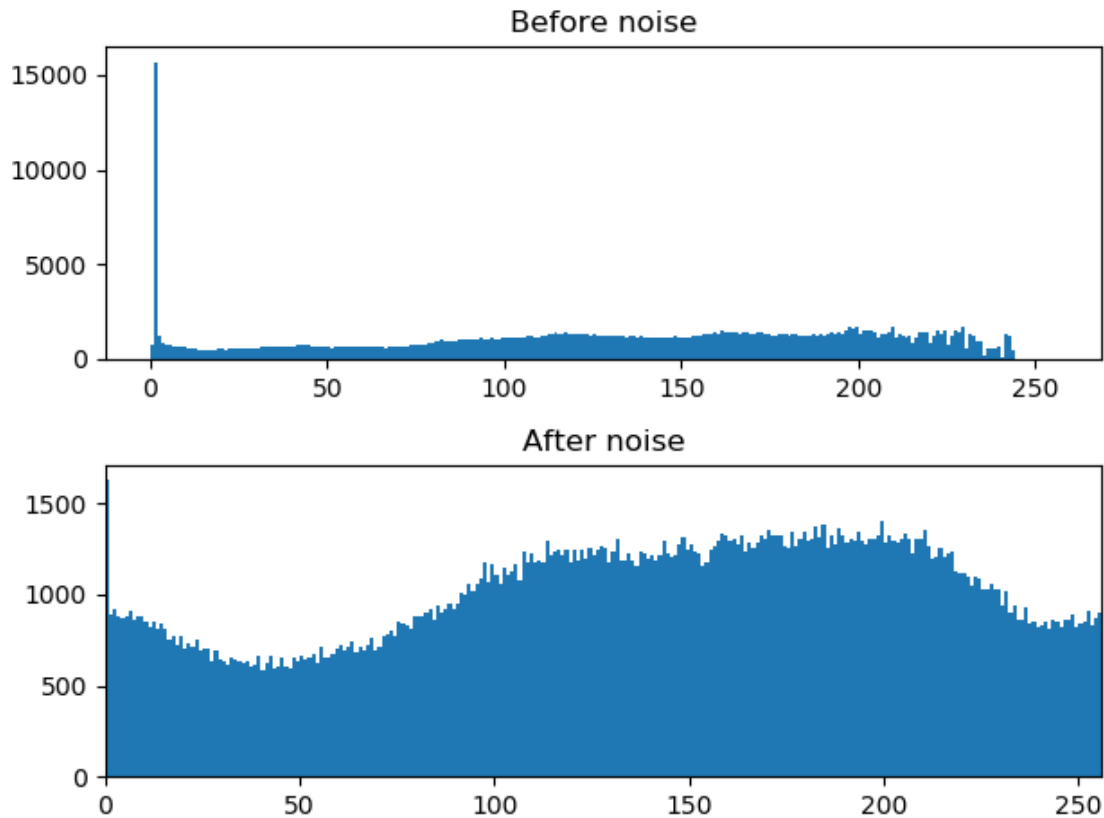


Fig 17: Intensity histogram before and after adding noise to the image

**2. Implement your own Matlab/python function that performs a 5x5 Gaussian filtering (1.5 marks). Your function interface is:**

**my\_Gauss\_filter()**

**input: noisy\_image, my 5x5 gausskernel**

**output: output\_image**

The gaussKernel is generated using inbuilt function of cv2. Which is then passed to my\_Gauss\_filter() function. This function convolves the kernel over the noisy image inputted using the convolution helper function in “Task 5.py” and outputs the smoothened image. To convolve first padding is added to image using np.pad function. Then using the formula from the lecture slides, the image is convolved by using element wise multiplication.

**3. Apply your Gaussian filter to the above noisy image, and display the smoothed images and visually check their noise-removal effects, investigating the effect of modifying the standard deviation of the Gaussian filter. You may need to test and compare different Gaussian kernels with different standard deviations (0.5 marks).**

**Note:** In doing this task, and the bilateral filter below you **MUST NOT** use any Matlab’s (or Python’s) inbuilt image filtering functions (e.g. imfilter(), filter2() in Matlab, or cv2.filter2D() in Python). In other words, you are required to code your own 2D filtering code, based on the



--	--	--

original mathematical definition for 2D convolution. However, you are allowed to generate a 5x5 sized Gaussian kernel with inbuilt functions.

The noisy image when passed through my\_Gauss\_filter() function results in the smoothing shown below. As can be seen the cat's whiskers and fur are smoothened along with the noise. For example in its eyes, the smoothing is visible.

Result of my\_Gauss\_filter (k\_size=(5,5), sigma=3)



Fig 18: Result of smoothing image using my\_Gauss\_filter

To compare the effect of sigma on smoothening, the following image (Fig 19) was generated.

--	--	--

my\_Gauss\_filter: sigma=3



my\_Gauss\_filter: sigma=7



my\_Gauss\_filter: sigma=13



--	--	--

Fig 19: Result of smoothing image using my\_Gauss\_filter using different sigma values for comparison

**4. Compare your result with that by Matlab's inbuilt 5x5 Gaussian filter (e.g. filter2(), imfilter() in Matlab, or conveniently cv2.GaussianBlur() in Python,). Please show that the two results are nearly identical (0.5 marks).**

Now the function definition of my\_Gauss\_filter() is compared with the inbuilt function (Fig 20). Both of their smoothed images have smoothed over the noise similarly. For example in the eye, they both look very similar. Also on the white fur below cat's head has been smoothed identically.

Inbuilt Gaussian Filtering (k\_size=(5,5), sigma=3)



Fig 20: Result of smoothing image using finbuilt Gaussian filter function

**Image Denoising via a Bilateral (Challenge task: this task will be difficult for most students to complete.) (5 marks)**

**1. Using your Gaussian filter as a base, implement your own Matlab/Python function that performs a 5x5 Bilateral filtering to gray-scale image. (1.5 marks)**

**Your function interface must be:**

**my\_Bilateral\_filter()**

**input: noisy\_image, my\_5x5\_gausskernel, colour\_sigma**

**output: output\_image**

--	--	--

where `colour_sigma` is the sigma applied to the intensity/gray scale part of the filter.

The same Gaussian filter as in Task 5.2 is used for `my_Bilateral_filter()` function. The weight assigned to any pixel depends on how far it is to the central pixel. It takes spatial distance and photometric distance into account.

**2. Apply your Bilateral filter to the above noisy image (greyscale version) from the last task, and display the smoothed images and visually check their noise-removal and bilateral edge preserving effects (0.5 marks in total).**

Bilateral filter smoothens the image but preserves the edge properties unlike Gaussian filter.

**In addition to the Gaussian filter, the range filter also has a standard deviation. You may need to test and compare different standard deviations for range (1.0 marks). Note: Do not use in-built functions, the same as for task 4.**

# Your description (and/or images, if needed) is here #

**3. Extend the Bilateral filter to colour images (eg. The color version of the previous grayscale image.). For this you may need to consider the CIE-Lab colour space as described in the paper. You will need to explore this for yourself. Namely, You need to generate the noisy color image and implement the bilateral filter to the color image (CIE-Lab). (2.0 marks) (Note this task is more difficult again).**

The noisy color image generated is given below (Fig)



--	--	--

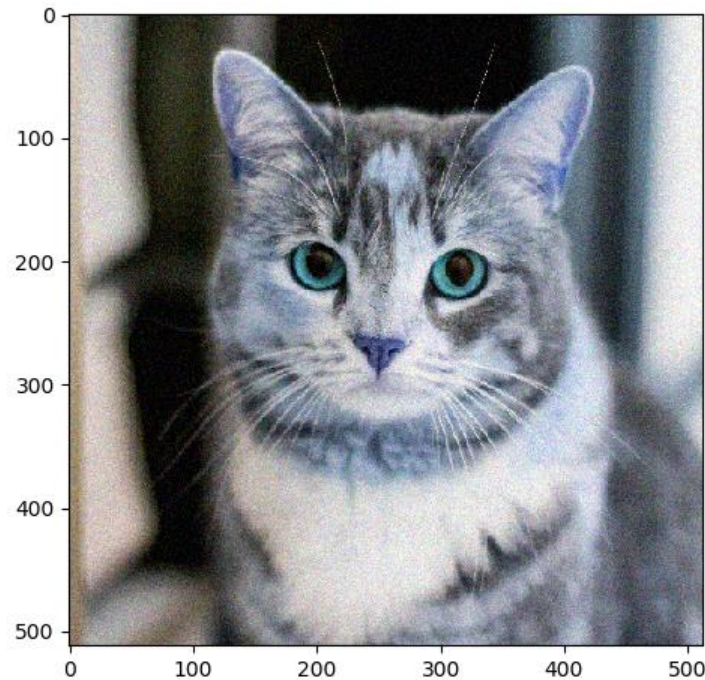


Fig 21: Adding Gaussian noise to color image

**4. In up to half a page, discuss the impact of smoothing on colour. What are the difficulties of smoothing in RGB colour space? Why is CIE-Lab space a good idea for this smoothing? (Compare images smoothed in CIE-Lab space vs RGB) Does the bilateral filter itself help? You may want to compare Gaussian smoothed colour images and bilateral filtered ones, and investigate filtering in different colour spaces. For this you can use your processed images as examples, possibly cropping and enhancing detail in your report to illustrate your discussion. (1 marks)**

Upon applying smoothing using smoothing filter like Gaussian on color images, the edges of the image become average of the 2 colors. For example, in an image with red and blue colors as neighbours forming an edge, after smoothing, the edge will become purple, their average. To address this problem, smoothing can be applied to the RGB channels of the image separately but intensity profiles across the edge in the three color channels are in general different, which will result in even more pronounced edge average color.

However, when bilateral filtering is performed in the CIE-Lab color space only visibly similar colors are averaged together, and only visibly important edges are preserved. Thus the image resulting from bilateral smoothing of the image has comparably very less averaged colors on the edges, and no different colors than the neighboring ones appear.

### **Task-6: Image Translation (3 marks)**

--	--	--

Choose an image among (image1.jpg, image2.jpg, image3.jpg) and resize it to 512 x 512.

**1. Implement your own function `my_translation()` for image translation by any given number of pixels between  $[-100, +100]$ , in both x and y. Note that this can be a real number (partial pixels). Display images translated by  $(2.0, 4.0)$ ,  $(-4.0, -6.0)$ ,  $(2.5, 4.5)$ ,  $(-0.9, 1.7)$ ,  $(92.0, -91.0)$  (0.20 for each image, 1.0 mark in total). Note: positive for away from upwards and right from the bottom left hand corner.**

The image “image1.jpg” is first converted to RGB color space and then resized to 512 x 512 using the inbuild resize function and clipped to  $[0, 255]$  range. The image is then displayed side by side with original image in Fig 22 below.

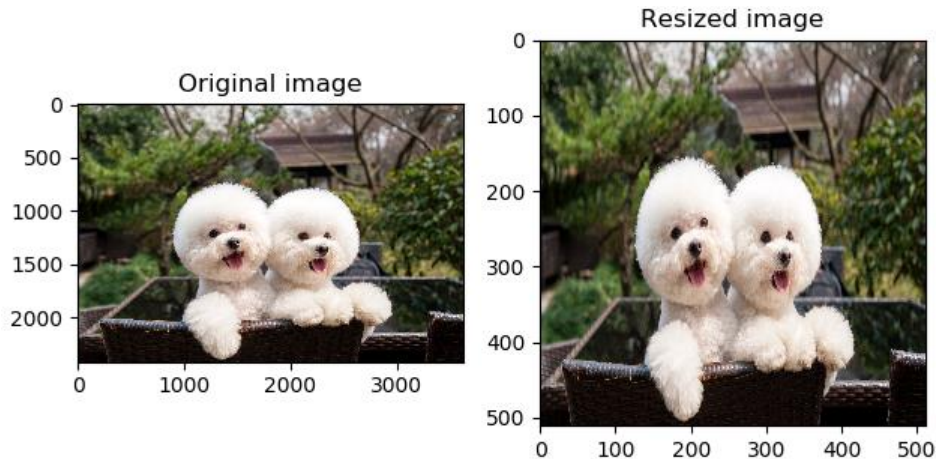


Fig 22: Comparing original image (left) and image with its red and green channels swapped (right)

Now, the resized image is to be translated by pixels using the `my_translation()` function defined in “Task 6.py” which takes image `im`, translation in x (`tx`), translation in y (`ty`) are the input parameters. These real valued `tx` and `ty` are round to integers using `np.round` function. Next the image is rolled by `np.roll` function along x and y (axis = 0 and 1 respectively). The rolled out part of image (`im_shifted`) is now filled by blackened according to the sign of `tx` and `ty` values in positive X and Y directions shown below (Fig 23).

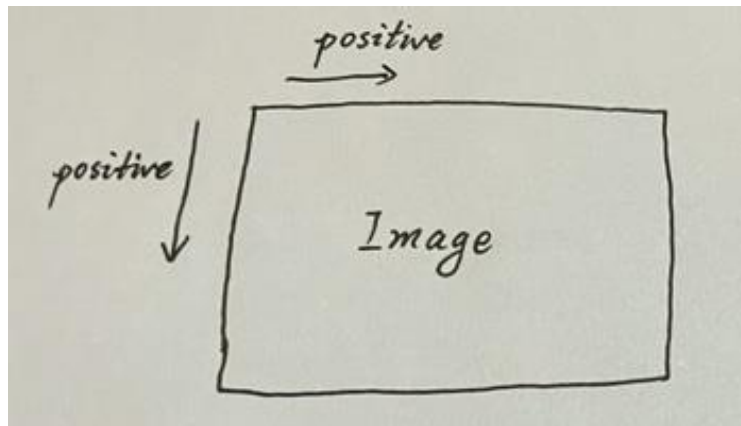


Fig 23: Positive X and Y direction for image translation

The shifted images are then displayed as below in Fig 24.

--	--	--

Translated image by  $tx = 2, ty = 4$



Translated image by  $tx = -4, ty = -6$



--	--	--

Translated image by  $tx = -1, ty = 2$



Translated image by  $tx = 92, ty = -91$

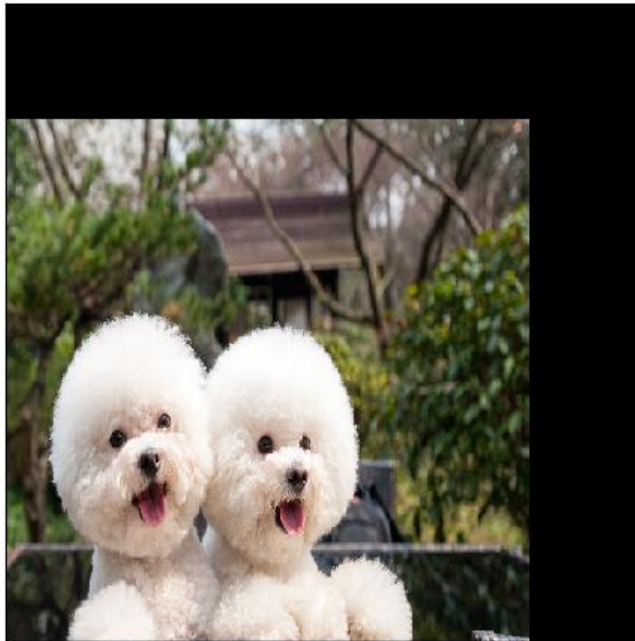


Fig 24: Translated by image by (a) (2.0,4.0), (b) (-4.0,-6.0), (c) (2.5, 4.5), (d) (-0.9,1.7), (e) (92.0,-91.0)

**2. Compare forward and backward mapping and analyze their difference (1.0 mark).**



--	--	--

**Hint:** When analyzing the difference, you can focus on: (1) visual results; (2) the principles in terms of formulation or others relevant; (3) advantages and drawbacks; (4) the computational complexity. You can also think about it from other aspects.

# Your description (and/or images, if needed) is here #

### **3. Compare different interpolation methods and analyze their difference (1.0 mark).**

**Hint:** When analyzing the difference, you can focus on: (1) visual results; (2) the principles in terms of formulation or others relevant; (3) advantages and drawbacks; (4) the computational complexity. You can also think about it from other aspects.

# Your description (and/or images, if needed) is here #