Computer Vision Assignment 1 (CLab1) S1-2024

UID: u7351505, Name: Yifan Luo

Task 1: Basic Image I/O (2 marks)

1. Save them to JPG image files of size 1024 columns × 720 rows, named “image1.jpg”, “image2.jpg” and “image3.jpg”. (0 marks)

The original images are listed as below:

Figure 1 Original Images

Save images to JPG files of size 1024 columns and 720 rows:

Figure 2 Task 1 Images

2. Using image1.jpg, develop code and functions that do the following tasks:

(a) Read this image from its JPG file and resize it to 384 columns × 256 rows. (0.2 marks)

The following plot shows all resized pictures:

Figure 3 Resized Images

(b) Separate the colour image into red, green, and blue (RGB) channels, and display each separately as a grayscale image. (0.2 marks per channel, 0.6 marks in total)

The following plot shows the grayscale images after being transformed from their corresponding RGB format:

Figure 4 Grayscale Images

The following group of plots show the separated R, G and B channels of each image:

Figure 5 R, G, B Channels of image1.jpg

Figure 6 R, G, B Channels of image2.jpg

Figure 7 R, G, B Channels of image3.jpg

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

The following group of plots show pairs of the grayscale image and its corresponding histogram:

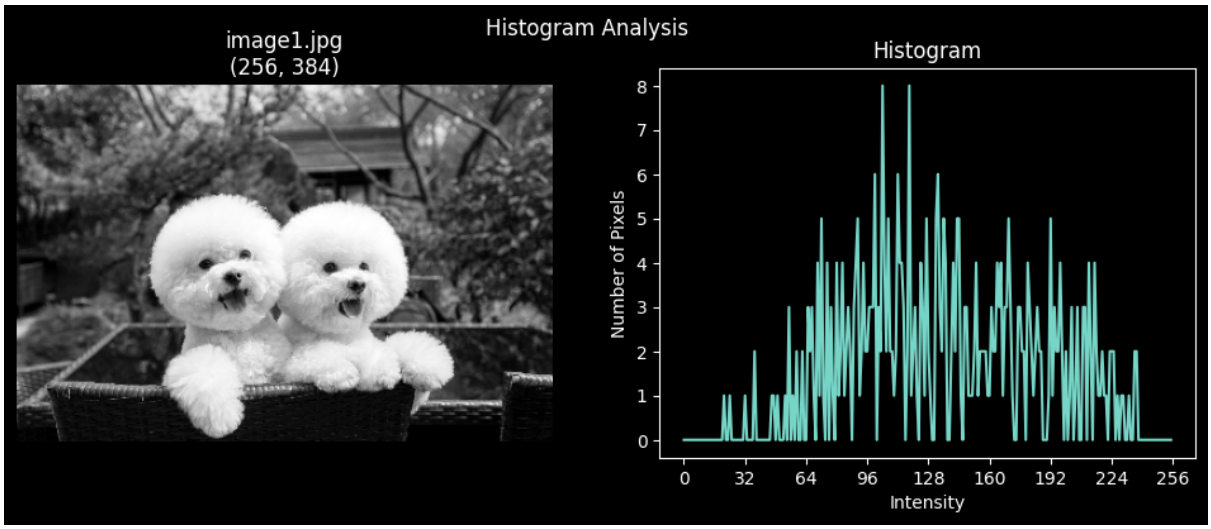

Figure 8 Hist. of image1.jpg

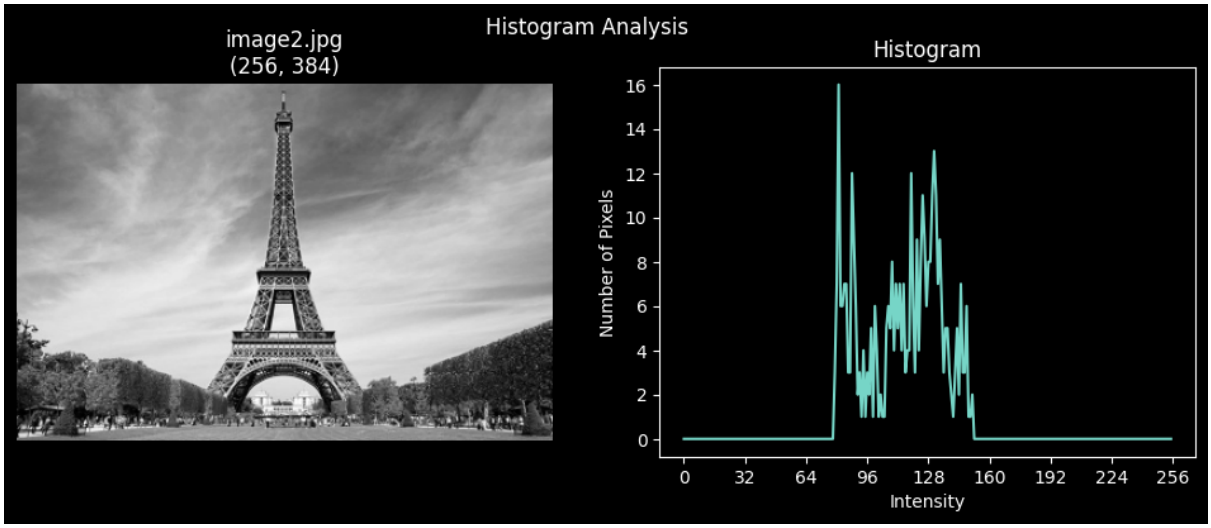

Figure 9 Hist. of image2.jpg

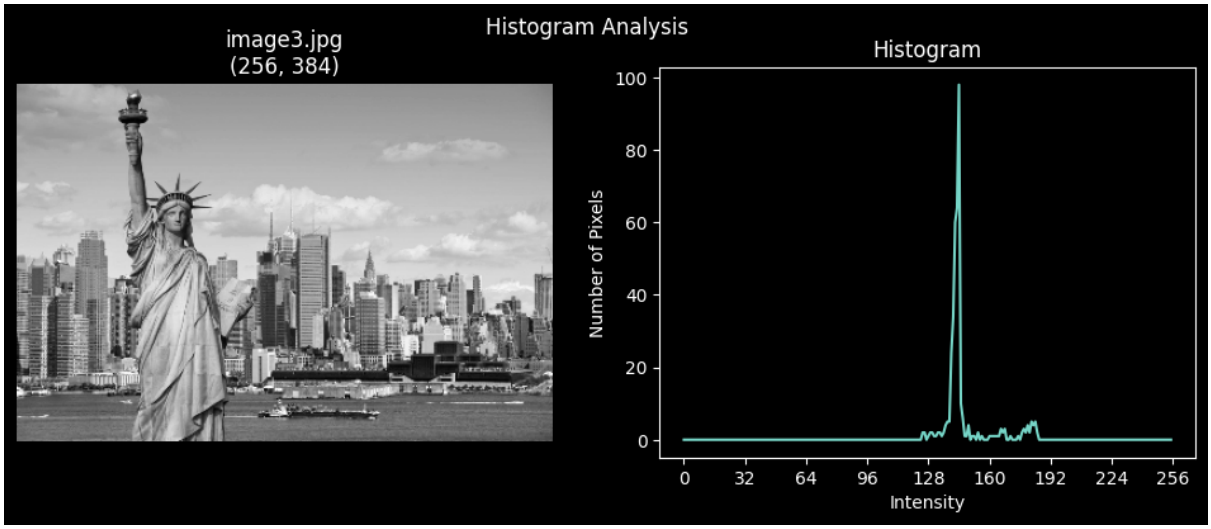

Figure 10 Hist. of image3.jpg

(d) Apply histogram equalisation to the individual red, green, and blue channels and then merge the three channels into a colour image. Display the three new histograms after equalisation and the new colour image. (0.15 marks per histogram and image, 0.6 marks in total)

The following images show the results of the RGB images after histogram equalisation of each channel. The first one is the result from a self-implement histogram equalisation function, and the second one is from built-in functions from OpenCV:

Figure 11 RGB Images after HE

Figure 12 RGB Images after HE by OpenCV

The following group of plots show the corresponding grayscale images and histograms of self-implement histogram equalisation function:

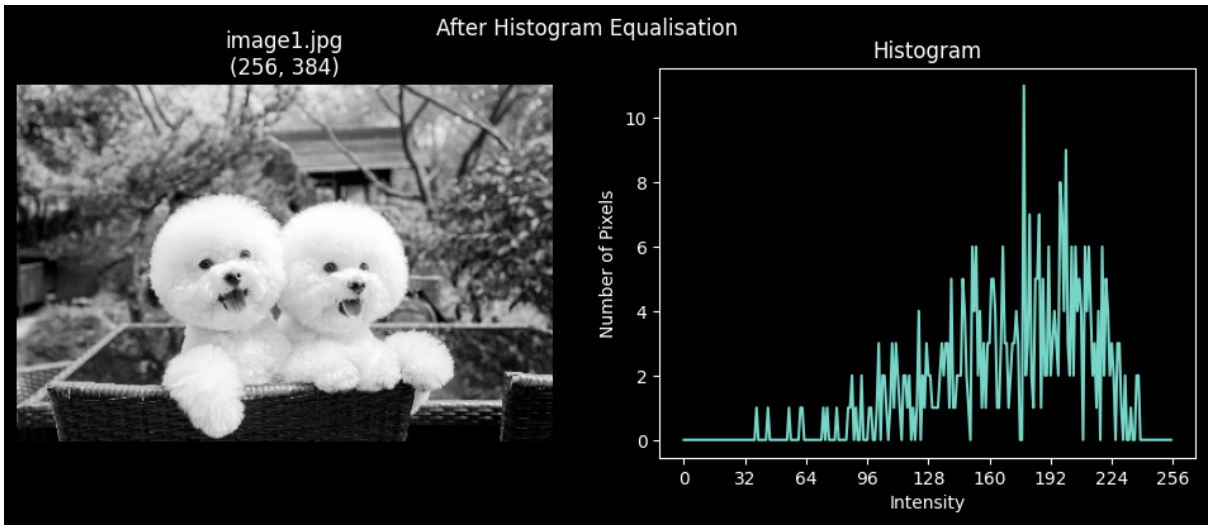

Figure 13 Hist. of image1.jpg after HE

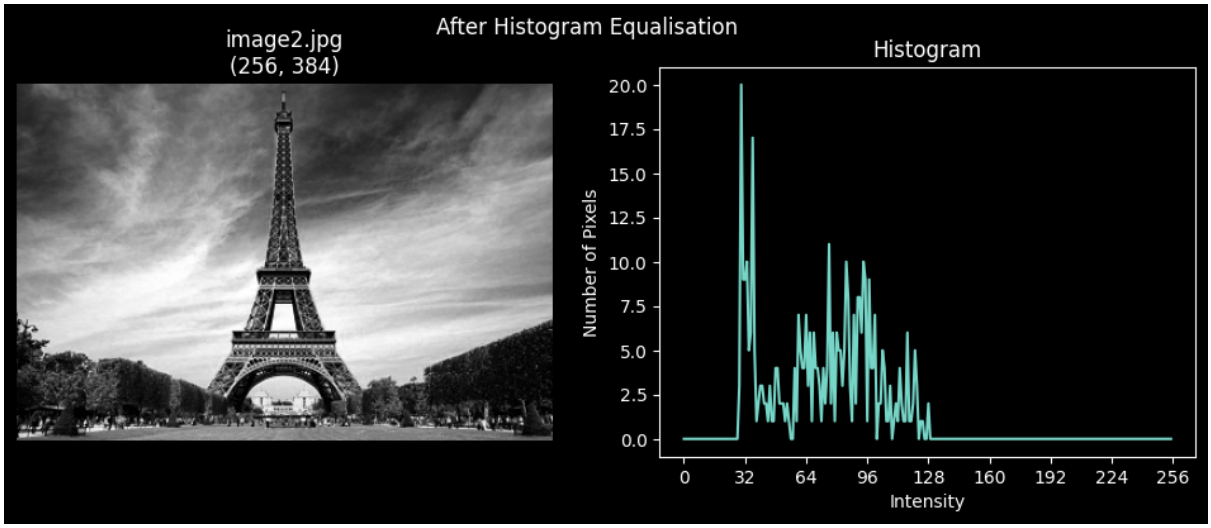

Figure 14 Hist. of image2.jpg after HE

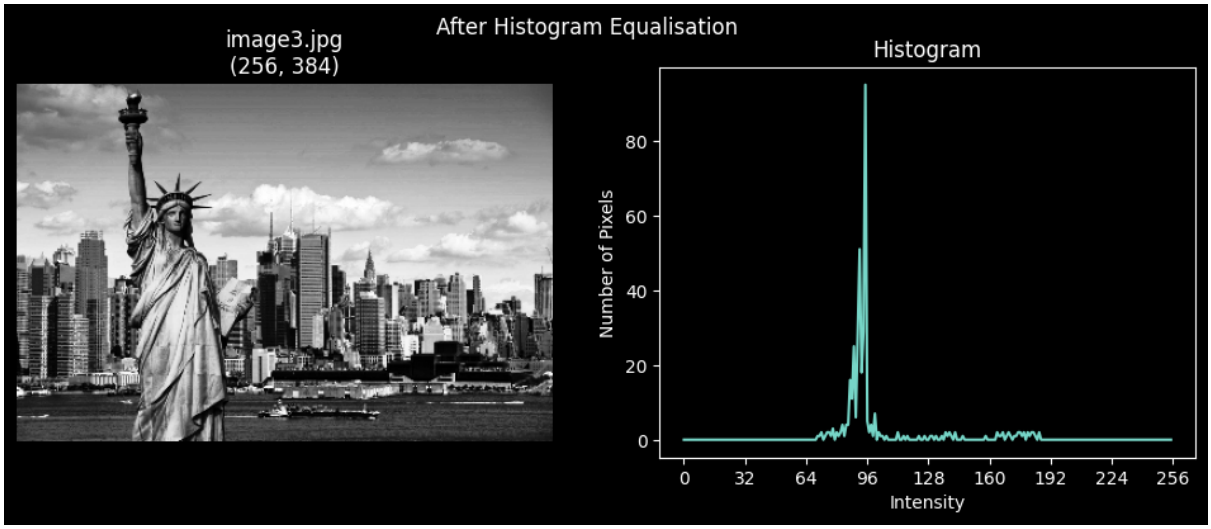

Figure 15 Hist. of image3.jpg after HE

For comparison, the following plots show the results from built-in functions in OpenCV:

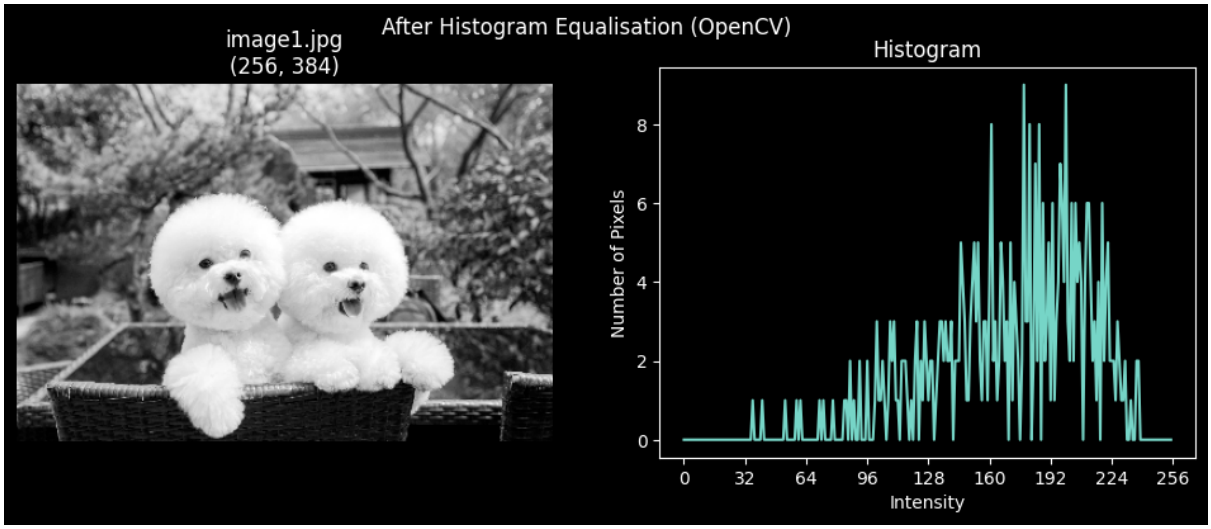

Figure 16 Hist. of image1.jpg after HE by OpenCV

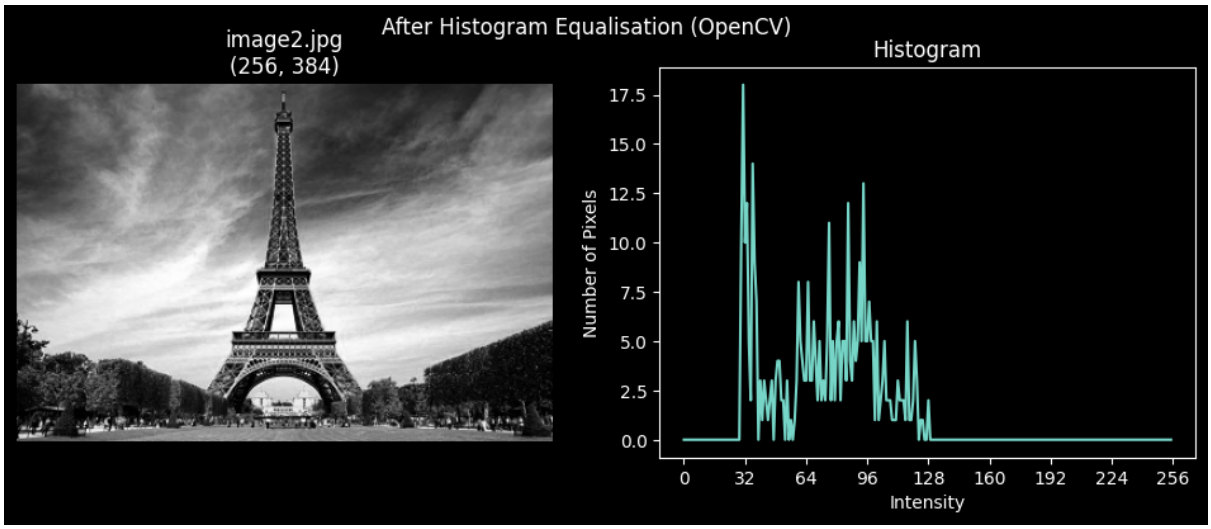

Figure 17 Hist. of image2.jpg after HE by OpenCV

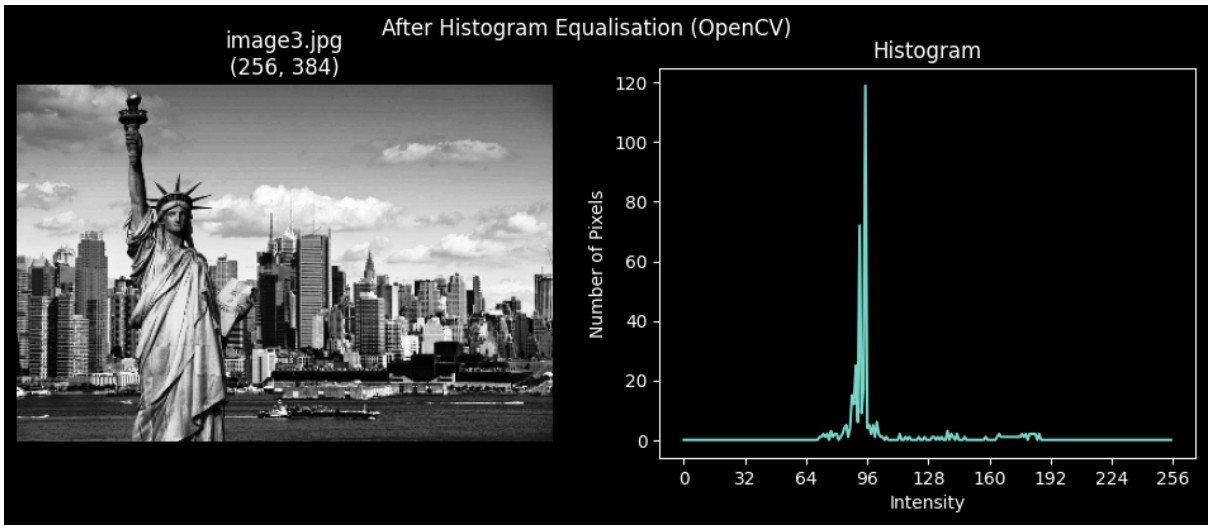

Figure 18 Hist. of image3.jpg after HE by OpenCV

Task 2: Colour Space Conversion (3 marks)

1. Based on the formulation of RGB-to-HSV conversion:

(a) Write your own function cvRGB2HSV that converts the RGB image to HSV colour space. (1 mark)

The following screenshot shows the implementation of cvRGB2CSV:

Figure 19 Screenshot of cvRGB2CSV Function

(b) Read in Figure 2(a) and convert it with your function, and then display the H, S, and V channels in your report. (0.5 marks)

The original images in RGB format:

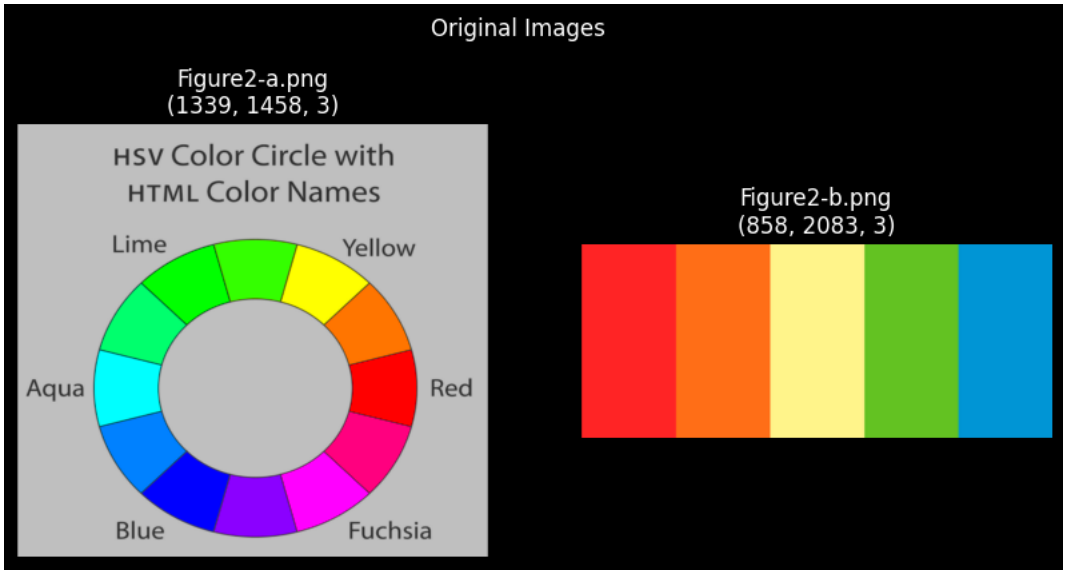

Figure 20 Original Images in Task 2

The corresponding images in HSV format by using cvRGB2HSV:

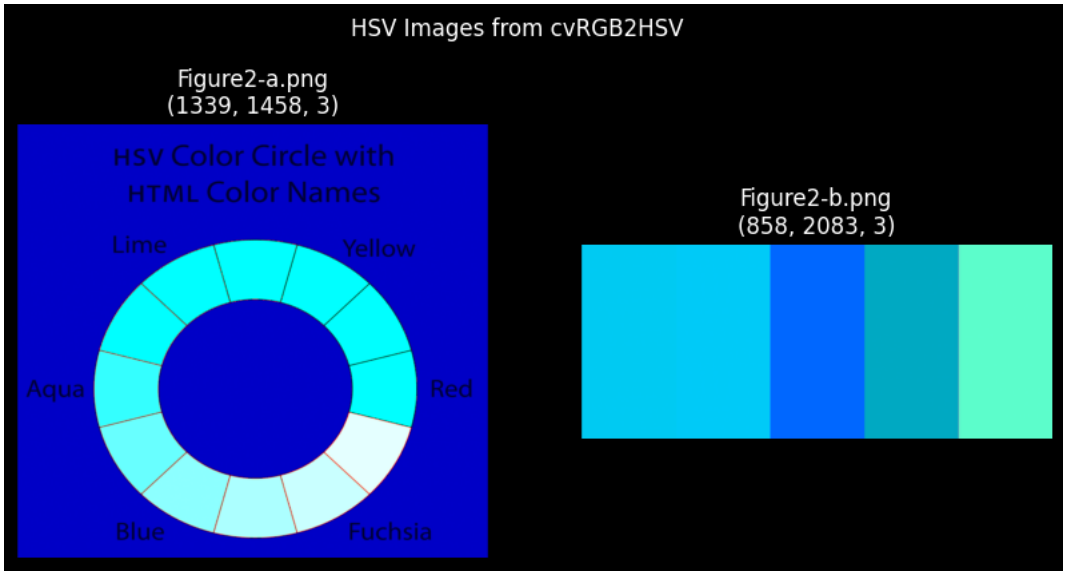

Figure 21 HSV Images in Task 2

The corresponding H, S and V channels by using cvRGB2HSV:

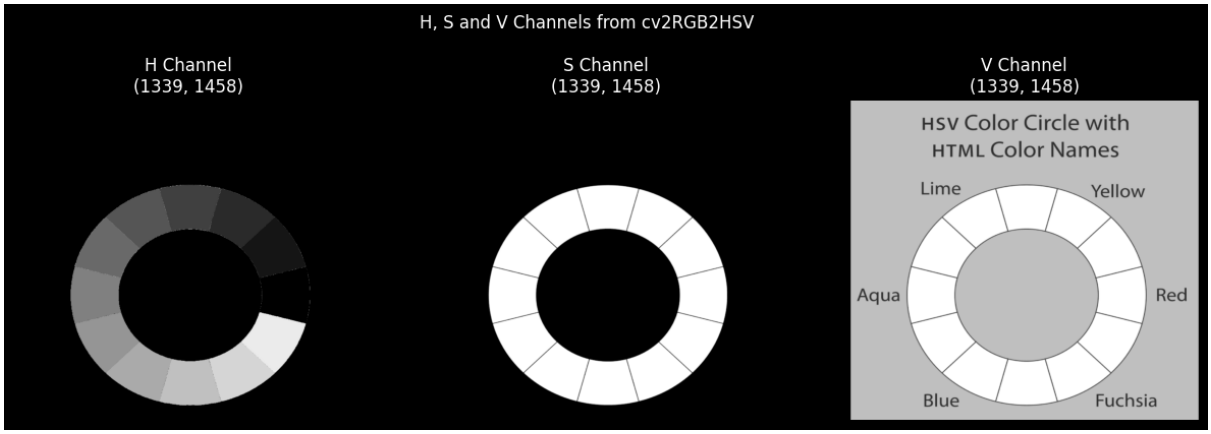

Figure 22 H, S and V Channels from cvRGb2CSV

2. Compute the average hue (H) values of the five colour regions in Figure 2(b) with your function and Python's built-in (scikit-image) function rgb2hsv.

(a) Print both results under the corresponding regions. (0.5 marks)

The average hue values of Figure2-b.png from cvRGB2CSV and scikit-image:

```
Average hue values of Figure2-b.png
cvRGB2HSV: 0.20505668323745524
scikit-image: 0.20505668323745524
```

The comparison of HSV images from cvRGB2CSV and scikit-image:

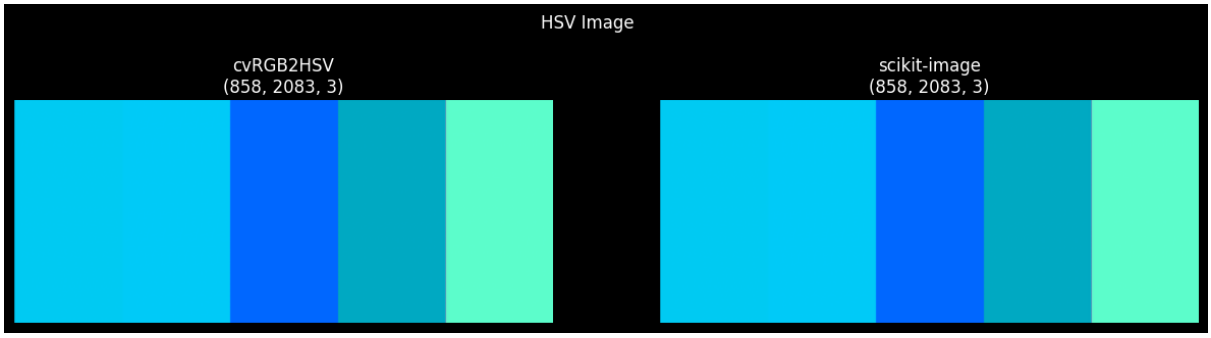

Figure 23 HSV Image of Figure2-b.jpg

The comparison of their corresponding grayscale images:

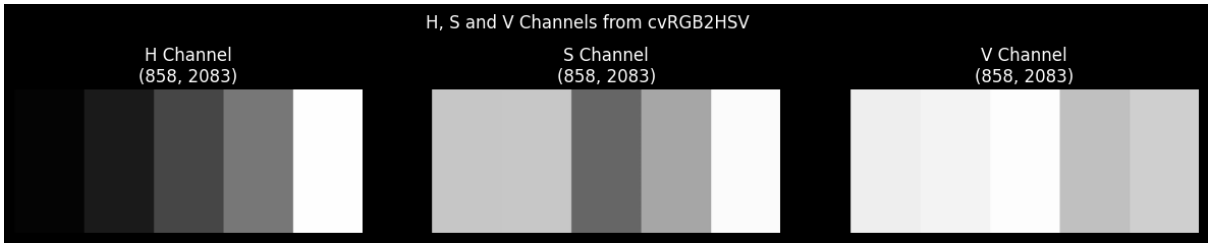

Figure 24 H, S and V Channels of Figure2-b.jpg by cvRGB2HSV

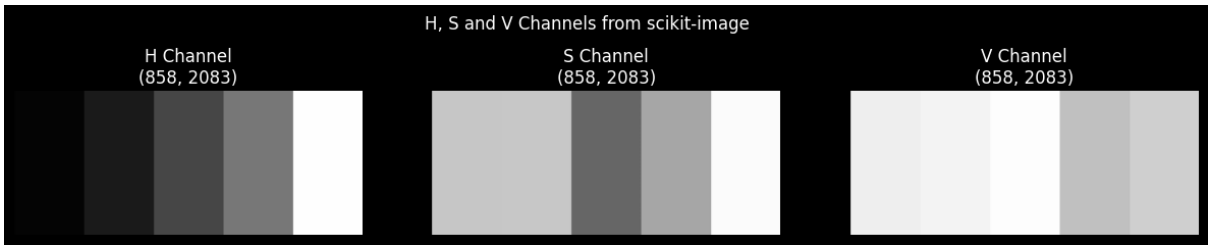

Figure 23 H, S and V Channels ofFigure2-b.jpg by scikit-image

(b) Explain how to distinguish and divide the five regions, and how to calculate the average hue value. More efficient solutions will obtain higher marks. (1 mark)

Assume pixels in each region share the same range of hue value, e.g.:

- The pixels in the red region should have their hue values in the range of [0.0, 0.05];
- The pixels in the orange region should have their hue values in the range of [0.05, 0.15];
- The pixels in the red region should have their hue values in the range of [0.15, 0.25];
- The pixels in the red region should have their hue values in the range of [0.25, 0.55];
- The pixels in the red region should have their hue values in the range of [0.55, 1.0];

The following plot shows the 5 regions, each stair represents the change of regions:

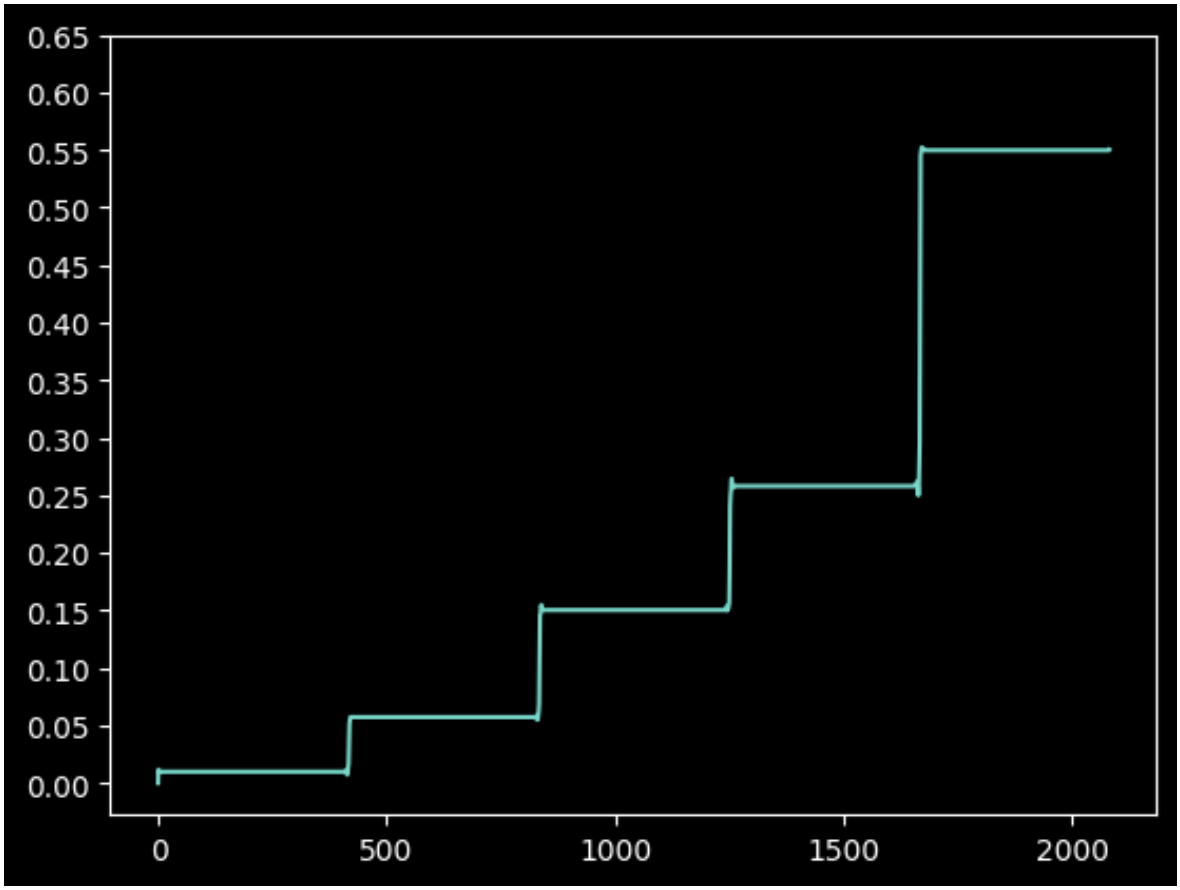

Figure 24 The Stair Plot of Hue Values

Hence, one method is to mask off the specific region, i.e.:

If the orange region is the target to mask off, set the hue values inside the range [0.05, 0.15] to 0; If the blue region is the target to mask off, set the hue values inside the range [0.55, 1.0] to 0.

The following plot shows the process of masking off each region by setting the hue values inside the range to 0. After being masking off, the corresponding region shows in black colour:

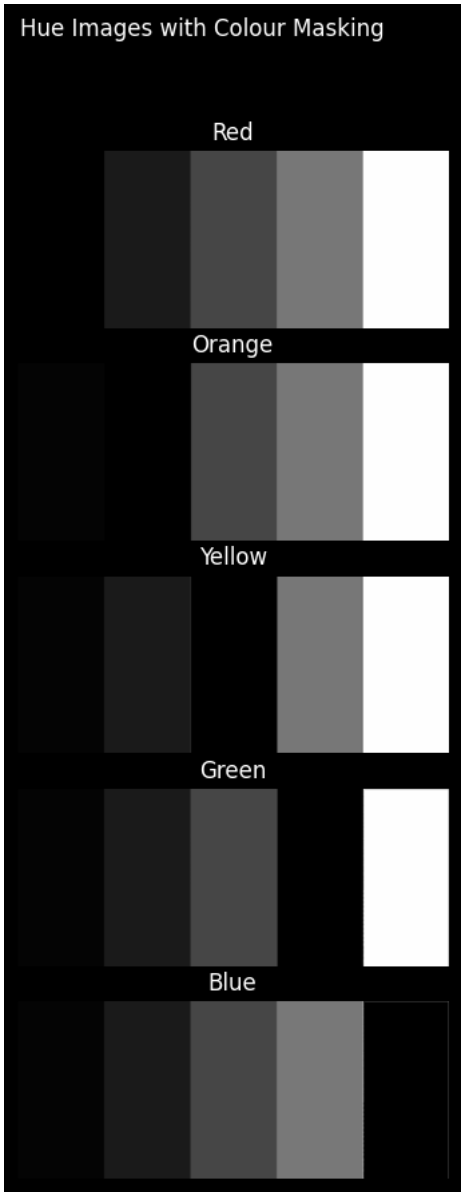

Figure 25 The Process of Masking-off Each Region

Following the same assumption, one way to compute the average hue value efficiently is:

- Separate each region based on hue values, e.g., [0.0, 0.05], [0.05, 0.15], etc.
- Sample one pixel from each region's central location, which represent the other pixels in the same region.
- Compute the average hue values from the samples from the last step. If there are only 5 regions, compute the average value from their 5 extracted samples.

The comparison of the average hue values:

```
The average hue value from all pixels: 0.2027485380966361

The average hue value from 5 samples: 0.20508075392642117
```

Task 3: Image Denoising via a Gaussian Filter

1. Read in image4.jpg. Crop a square image region corresponding to the central facial part of the image, resize it to 512 × 512, and save this square region to a new grayscale image. Display the two images. (0 marks)

The following plot shows the resized central part of the RGB image and its grayscale image:

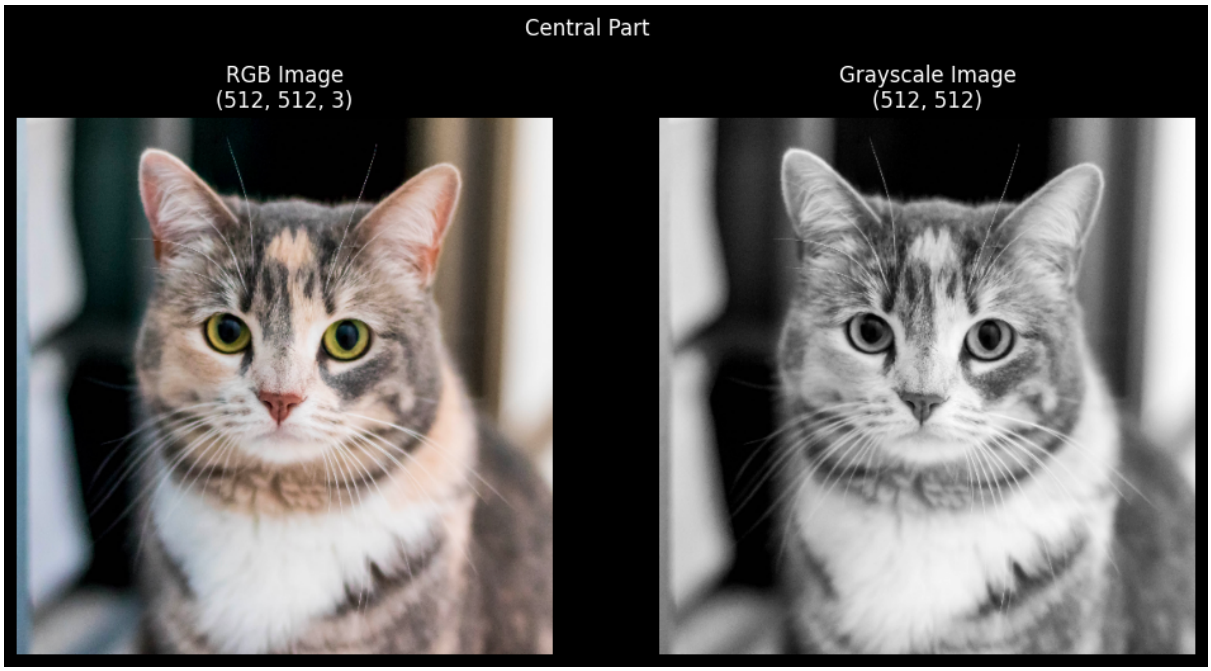

Figure 26 RGB and Grayscale Images of image4.jpg

2. Add Gaussian noise to the new 512 × 512 image, with zero mean and a standard deviation of 15. The intensity values of the noised image should be clipped to be within [0, 255]. (0.5 marks)

The following plot shows the RGB and grayscale pictures with Gaussian noise (0 mean and 15 standard deviation):

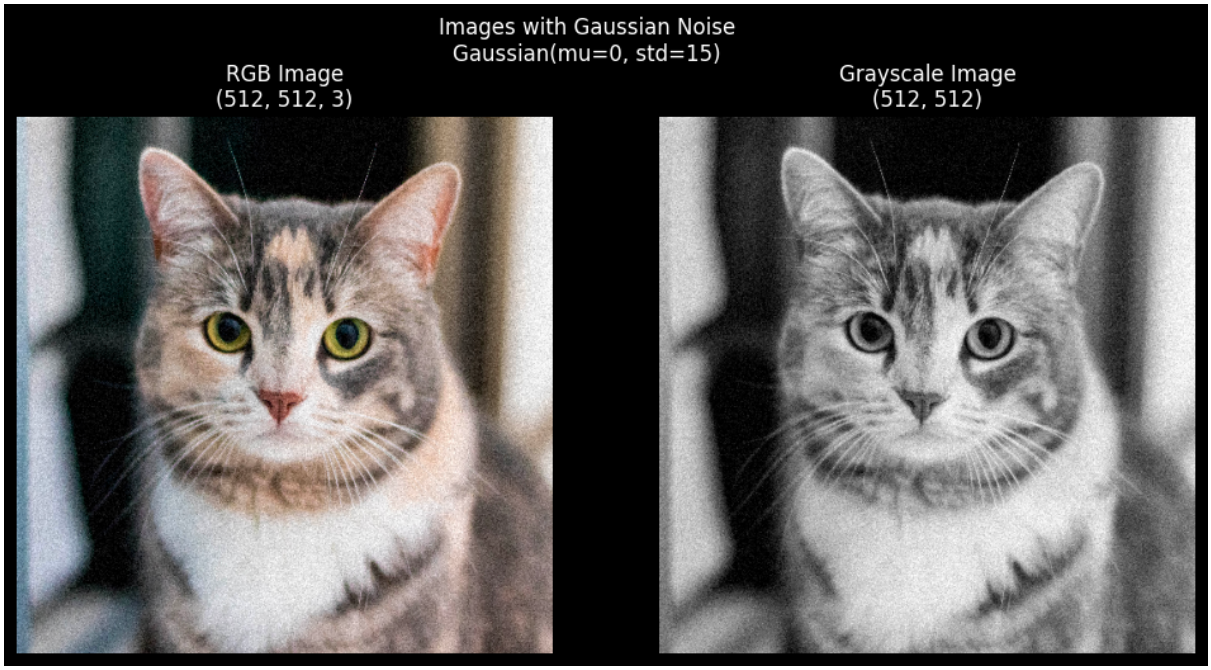

Figure 27 RGB and Grayscale Images after Adding Gaussian Noise

3. Compute and display two intensity histograms side by side, one before adding the noise and one after adding the noise. Describe what you can observe from the histograms and explain the reasons why they are similar/different. (1 mark)

The following pair of images shows the intensity histogram before adding Gaussian noise:

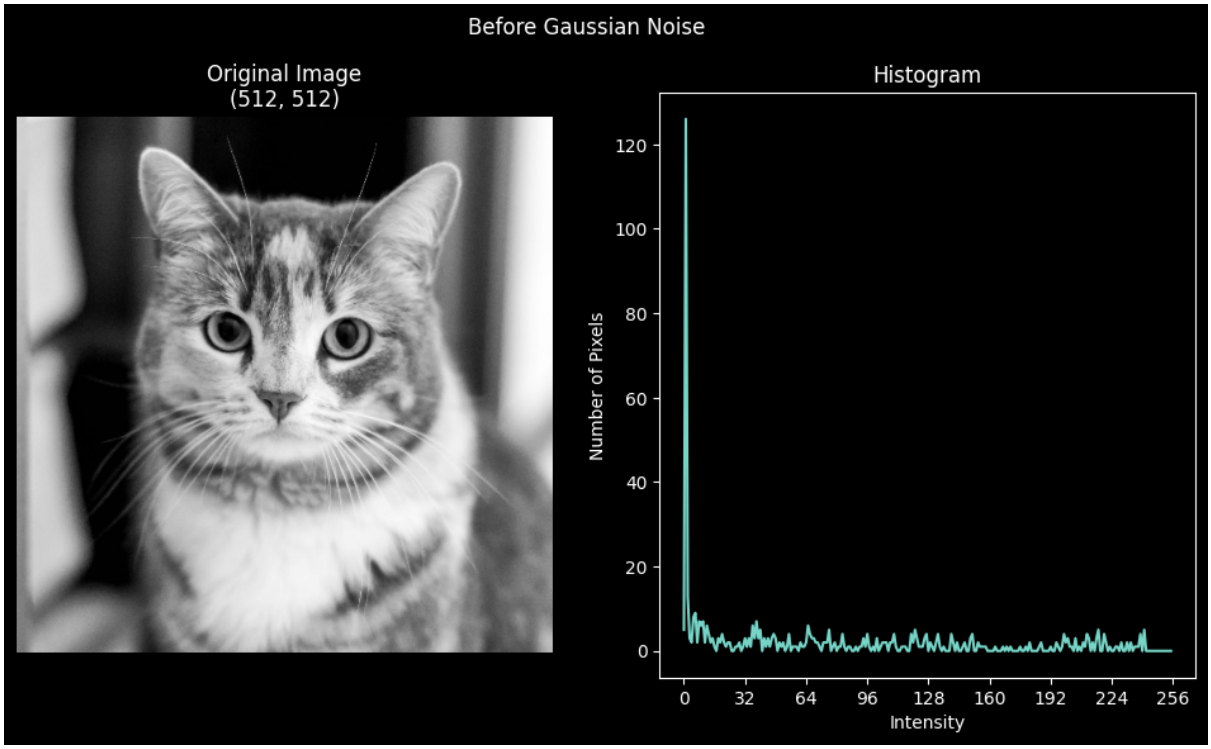

Figure 28 The Grayscale Image and its Histogram before Adding Noise

After adding Gaussian noise (0 mean, 15 standard deviation):

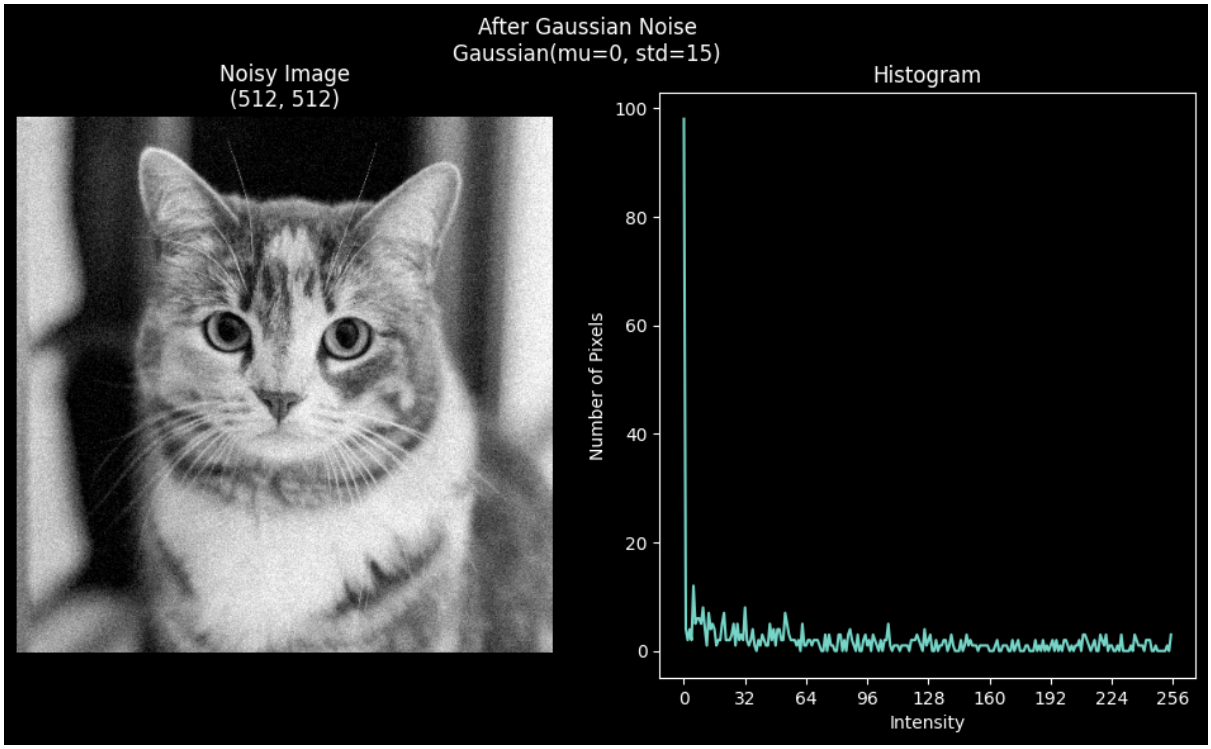

Figure 29 The Grayscale Image and its Histogram after Adding Noise

The 2 histograms don't have much difference in terms of their intensity distribution. Notice that all the intensity values are clipped into the range [0, 255].

In the histogram before adding noise, most of the pixels have intensity values under 16, which correspond to the dark background and dark features on the kitty. After adding Gaussian noise, the low-intensity-value pixels have negative intensity values, which are clipped to 0. Hence, most of the intensity values are still below 16.

4.Implement your own function that performs a 7×7 Gaussian filtering operation. (1.5marks)

The following screenshot contains 2 main functions in the Gaussian filtering operation. The first function gauss_kernel generates a Gaussian kernel with the input mean, standard deviation and size parameters. The second function my_Gauss_filter conducts the Gaussing filtering operation. It takes one picture and one Gaussian kernel (7 * 7 in this task) as input parameters.

undefined - u7351505.ipynb

```python
def gauss_kernel(mu=0, std=15, size=7):
    """Generate a Guassian kernel.

    Args:
        mu (int, optional): The mean of Gaussian distribution. Defaults to 0.
        std (int, optional): The sigma of Gaussian distribution. Defaults to 15.
        size (int, optional): The size of the kernel. Defaults to 7.

    Returns:
        kernel (numpy.ndarray): Generated Gaussian kernel.
    """
    kernel = np.fromfunction(lambda x, y: (1/(2*np.pi*std**2)) * np.exp(-((x-(size-1)/2)**2 + (y-(size-1)/2)**2)/(2*std**2)), (size, size))
    kernel /= np.sum(kernel)
    return kernel

# Generate a Gaussian kernel
kernel = gauss_kernel(0, 3, 7)

def my_Gauss_filter(noisy_img, gauss_kernel):
    """Gaussian filtering operation.

    Args:
        noisy_img (numpy.ndarray): The input noisy grayscale image.
        gauss_kernel (numpy.ndarray): The input Gaussian kernel.

    Returns:
        clean_img (numpy.ndarray): The image after Gaussian filtering.
    """
    gauss_kernel /= np.sum(gauss_kernel)  # normalisation

    clean_img = np.empty_like(noisy_img)

    padding_size = gauss_kernel.shape[0] // 2
    noisy_img_padded = np.pad(noisy_img, ((padding_size, padding_size), (padding_size, padding_size)), mode='edge')

    for i in range(padding_size, noisy_img_padded.shape[0] - padding_size):
        for j in range(padding_size, noisy_img_padded.shape[1] - padding_size):
            neighbour = noisy_img_padded[i - padding_size:i + padding_size + 1, j - padding_size:j + padding_size + 1]
            clean_img[i - padding_size, j - padding_size] = np.sum(neighbour * gauss_kernel)

    return clean_img
```

Figure 30 The Screenshot of gauss_kernel function and my_Gauss_filter function

5. Apply your Gaussian filter to the noisy image from step 2 and display the smoothed images and visually verify their noise-removal effects. (0.5 marks)

The following plot compares the grayscale image before and after Gaussian filtering operation. The left image is a noisy image from step 2, and the right image is a filtered image with a 0 mean and 3 standard deviation Gaussian kernel.

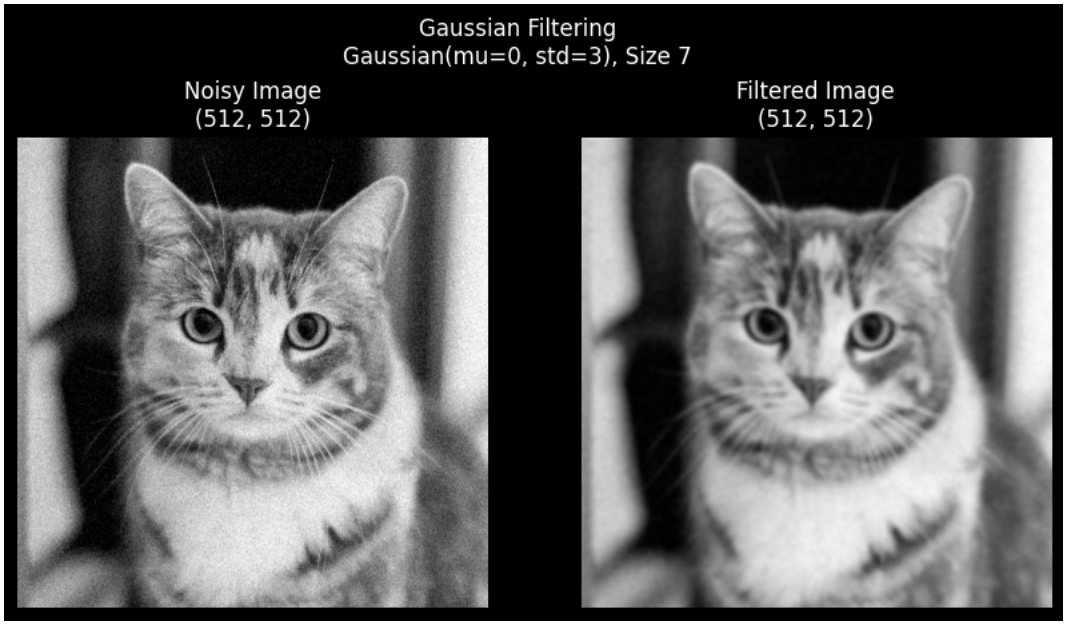

Figure 31 The Before/After Comparison

The filtered image has significantly been smoothed by using the Gaussian kernel. However, the filtered picture is less clear compared to the original image shown in step 2. It's difficult to distinguish the edge of the fluffy furry kitty from the background. The facial expression is also less recognisable.

6. One of the key parameters to choose for the task of image filtering is the standard deviation of your Gaussian filter. Test and compare several different Gaussian kernels with different standard deviations. (1 mark)

7. Compare your result with a built-in 7×7 Gaussian filter function (e.g., cv2.GaussianBlur) and show that the two results are nearly identical. (0.5 marks)

The following group of pictures show the results before and after implying Gaussian filtering operation. The second and third columns shows the results from the self-implement function and OpenCV. The mean values are 0s in all experiments, and the standard deviation values are in the following order: 0.5, 1, 5, 10, 50.

Figure 32 Noise Image and Filtered Images with std=0.5

Figure 33 Noise Image and Filtered Images with std=1.0

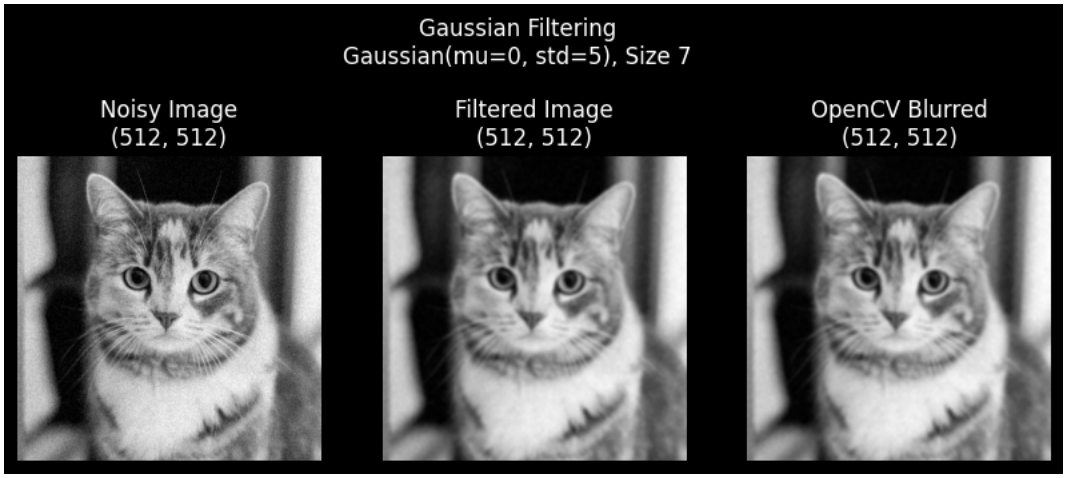

Figure 34 Noise Image and Filtered Images with std=5

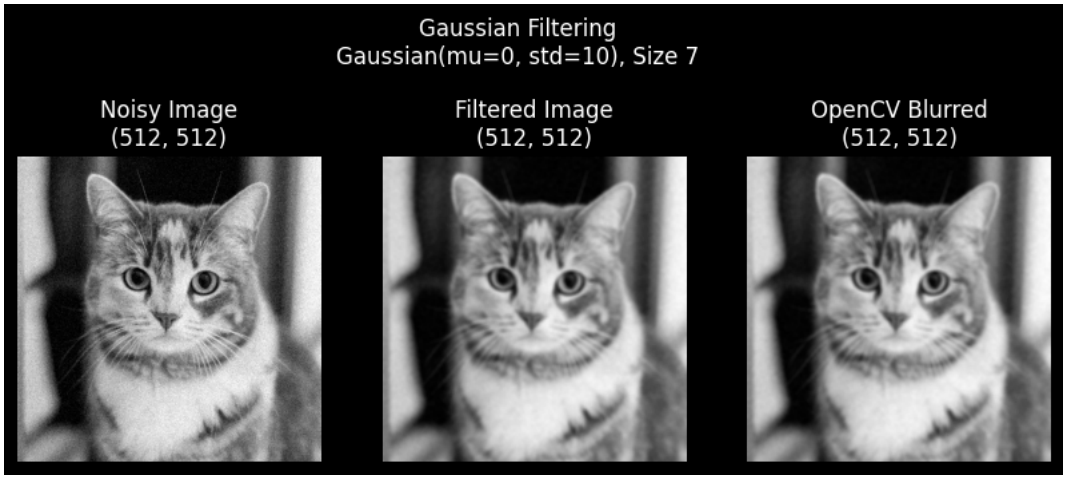

Figure 35 Noise Image and Filtered Images with std=10

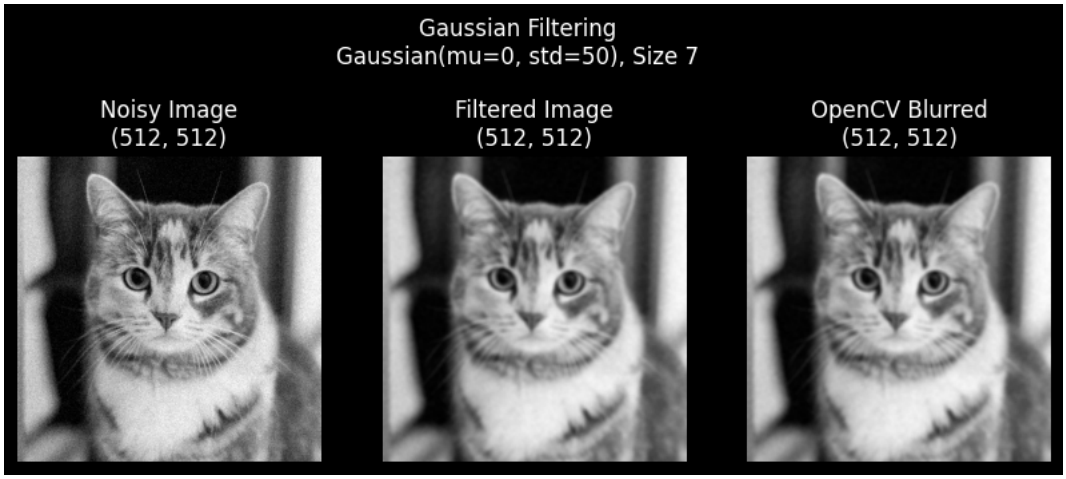

Figure 36 Noise Image and Filtered Images with std=50

The results from self-implement function and OpenCV are visually similar. To examine the difference, the following screenshot show the absolute difference of intensity values in all 5 comparisons:

undefined - u7351505.ipynb

```
# Compare pixel values in element-wise with a absolute tolerance
print("Std. 0.5:", np.allclose(gs_std_0_5, gs_gb_0_5, atol=7))
print("Std. 1:", np.allclose(gs_std_1, gs_gb_1, atol=12))
print("Std. 5:", np.allclose(gs_std_5, gs_gb_5, atol=13))
print("Std. 10:", np.allclose(gs_std_10, gs_gb_10, atol=14))
print("Std. 50:", np.allclose(gs_std_50, gs_gb_50, atol=13))
```

Figure 37 The Screenshot Showing the Absolute Difference

```
Std. 0.5: True
Std. 1: True
Std. 5: True
Std. 10: True
Std. 50: True
```

Notice than when the standard deviation increases, the absolute difference also increases. The overall difference is less significant.

Task 4: Sobel Edge Filter (5 marks)

1. Implement your own 3 × 3 Sobel filter. Do not use any built-in edge detection filter. The implementation of the filter should be clearly presented with your code. Test your filter on sample images (image2.jpg, image4.jpg), first converting them to grayscale, and visualise the results. (1.5 marks)

The original grayscale images:

Figure 38 Original Grayscale Images

The following screenshot shows the implementation of the Sobel edge filtering operation. The function takes a grayscale image as input and generate its corresponding gradient magnitude approximation as output.

undefined - u7351505.ipynb

```python
def sobel_edge_filter(gs_imgs):
    """Sobel edge filtering operation on a grayscale image.

    Args:
        gs_imgs (List[numpy.ndarray]): The input grayscale images.

    Returns:
        grad_magnitude (List[numpy.ndarray]): The gradient magnitude approximation results.
    """
    filter_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    filter_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    grad_magnitude = []
    for img in gs_imgs:
        grad = np.zeros_like(img)  # grad. magnitude
        img_padded = np.pad(img, ((1, 1), (1, 1)), mode='constant')
        for i in range(1, img_padded.shape[0] - 1):
            for j in range(1, img_padded.shape[1] - 1):
                neighbour = img_padded[i - 1:i + 1 + 1, j - 1:j + 1 + 1]
                G_x = np.sum((filter_x * neighbour))**2  # Horizontal filtering
                G_y = np.sum((filter_y * neighbour))**2  # Vertical filtering
                grad[i - 1, j - 1] = np.sqrt(G_x + G_y)  # Compute magnitude
        grad_magnitude.append(grad)
    return grad_magnitude
```

Figure 39 The Screenshot of sobel_edge_filter Function

The results after Sobel edge filtering operation:

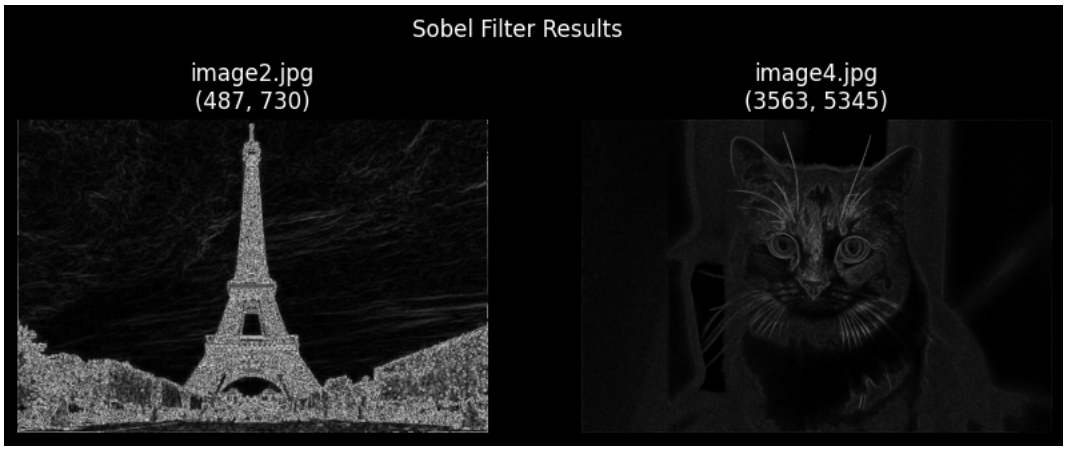

Figure 40 Grad. Magnitude of After Sobel Edge Filtering

(a) Compare your result with the built-in Sobel edge detection function. (0.5 marks)

The following plot shows the Sobel edge filtering by using built-in functions from OpenCV:

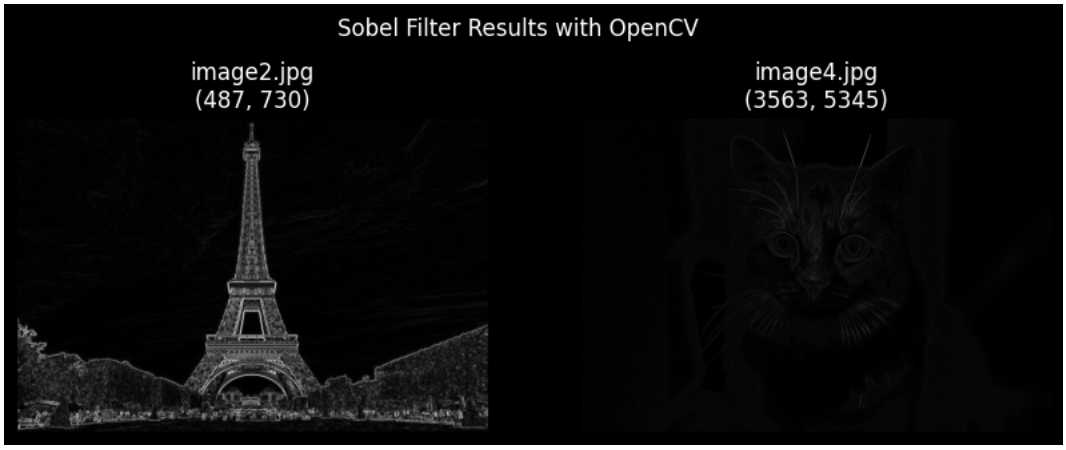

Figure 41 Grad. Magnitude of After Sobel Edge Filtering by OpenCV

Visually, the 2 Sobel edge filtering results show clearly the edges in each picture. In the results from the self-implement function, more edges are detected in the building and the plants areas, and some of the intensity values are much higher compared to the OpenCV results. Same for the second image, but less difference.

(b) Briefly explain the purpose of the Sobel filter and how it achieves this. (1 mark)

The Sobel filter is a tool for edge detection and is generally used in object detection, feature extraction and image segmentation tasks. The purpose of the filter is to detect edges by computing the gradient of the intensity values at each pixel. It has 2 kernels, one for the horizontal direction and one for the vertical direction. When it encounters edges, the gradient values are normally will be high, and it normally shows up brighter on its grayscale image visualisation.

To detect edges in the image, the Sobel filter performs the following steps:

1. Convolve the image with the Sobel_x kernel to compute the horizontal gradient.
2. Convolve the image with the Sobel_y kernel to compute the vertical gradient.
3. Compute the magnitude of the gradient vector at each pixel using the formula: Magnitude = sqrt(Gx^2 + Gy^2).

4. Compute the orientation of the gradient vector at each pixel using the formula: Orientation = arctan(Gy / Gx).

The magnitude represents the strength of the edge at each pixel, while the orientation indicates the orientation of the edge. By thresholding the magnitude, we can detect the locations of edges in the image.

2. Investigate the accuracy of the Sobel filter for predicting the orientation of edges. If it is used for edge detection, what might cause inaccuracies in the estimation of edge orientations? Look at the histogram of the gradient orientation and find out the major edge orientations in the image. Discuss whether the gradient orientation histogram aligns with the real expected edges for the image or not. (2 marks)

The reasons may cause inaccuracy for predicting the orientation of edges:

- Image Noise: The Sobel filter is sensitive to noise and high levels of noise can result in incorrect edge orientation estimates.
- Smoothed Image: Smoothed or blurred images may change the local structure of edges and affect the accuracy.
- Edge Thickness: The Sobel filter only detects the “edges”. If an edge has variable thickness or if it is not well-defined, the results may not be accurate to represent the true orientation.
- Discontinuities: Some non-linear or curve-shape edges can be complicate for Sobel filters.
- Kernel Size: Large kernel size may generate smoother gradient estimates but may also lose edge details.

The histogram of orientation by using OpenCV built-in functions:

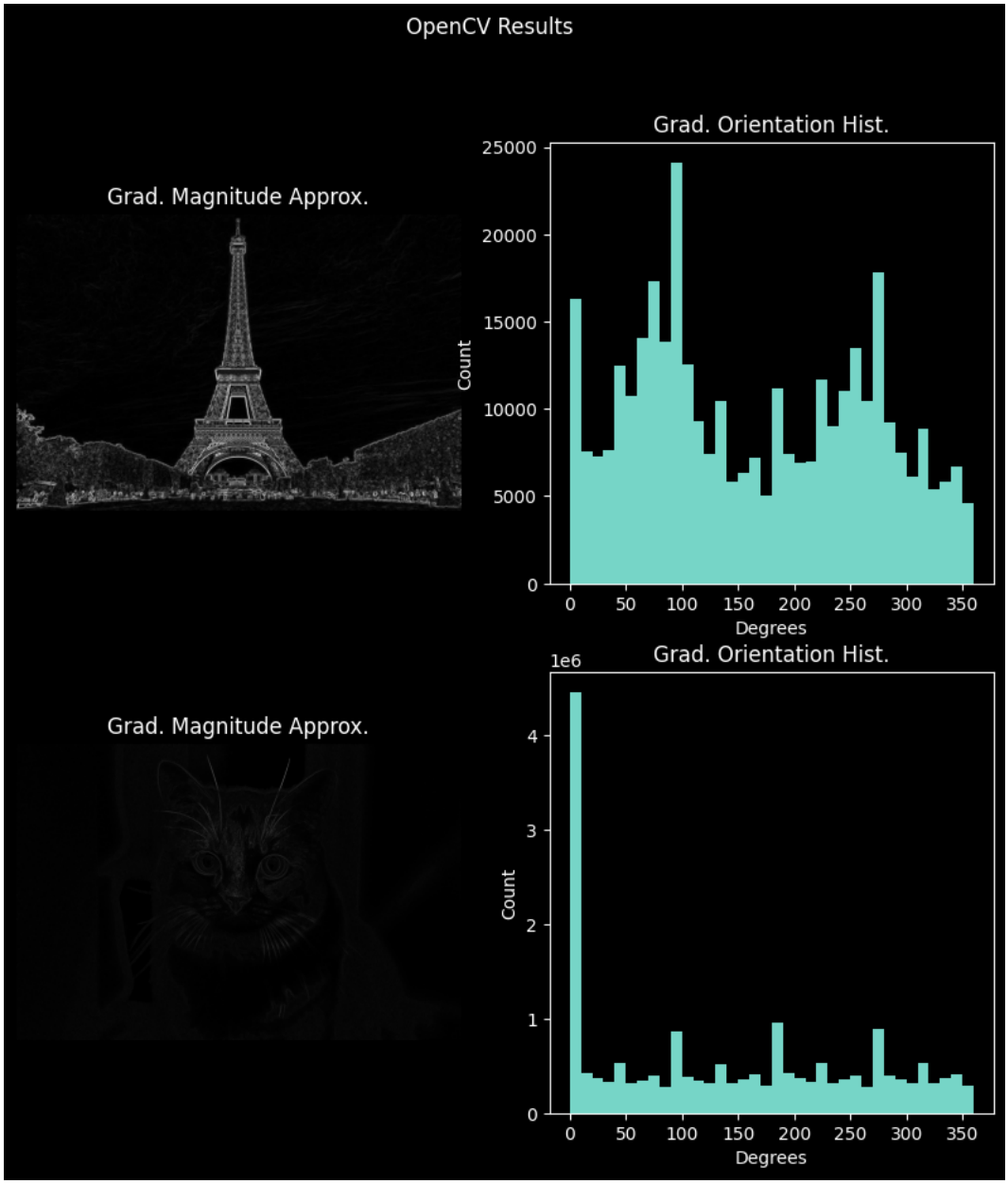

Figure 42 Grad. Magnitude and Grad. Orientation Histogram by OpenCV

From the above plot, the edges around the building and plants are well detected, but loses the edges of the clouds. The orientation histogram shows that most of the edge are around 0, 90 and 270 degrees, since the building and plants have obvious vertical and horizontal edges, the 270 degrees may sociated with the perspective in the picture.

In the second histogram, most of the edge orientations are around 0 degree, and some are around 90, 180 and 270 degrees. It shows that most of the edges are horizontal edges, others are from vertical edges or related to the perspective of the image. It's not obvious in the grayscale image since the background and the kitty's fur are mixed and blurred together. The magnitude image barely shows any edges in the background, and only limited amount of

kitty's facial structure is illustrated. It's difficult to visually distinguish the edges from the picture.

Task 5: Short Response Questions (5 marks)

1. Table 1 is a separable filter.

(a) What does it mean to be a separable filter? (0.5 marks)

It means a type of filter that can be decomposed into the product of two 1D filters. Mathematically, if U represent a 2D filter, then U_x and U_y represent the 1D filters along its x and y dimensions. Then U is separable if it can be expressed as U = U_x ・ U_y.T, where U_y.T means the transpose of U_y.

(b) Write down the separate components of the filter in Table 1. (1 mark)

Suppose U = [[4, 4, 6], [4, 4, 6], [6, 6, 9]]. Then U_x = [2, 2, 3].T, and U_y = [2, 2, 3].T.

2. In this assignment, we have so far looked at applying convolution kernels to single-channel (grayscale) images.

(a) Propose an approach for extending convolutions to multi-channel inputs with multi-channel outputs, where the number of input and output channels may differ. (1.5 marks)

Building up a neural network where the input layer depends on the number of input channels, and the output layer depends on the number of output channels. Each input neuron in the input layer handles one channel from the multi-channel inputs, and each output neuron represent the corresponding output channels based on the requirement.

Different convolution operation can be conducted on each input neuron. After the convolution for all inputs, a linear combination of feature maps, i.e., the results from the previous convolution, can be used to generate corresponding outputs based on the requirements. The number of different linear combinations depends on the number of output channels.

The illustration of this idea is shown in 5.2 (c).

(b) Discuss why these more generalised convolution operators may be useful in computer vision. (1 mark)

- Flexibility: Extended convolutions for multi-channel inputs with multi-channel outputs increases the flexibility of the neural network models.
- Feature Learning: Extended convolution allows the neural network to learn complex visual features during its training process, which leads to more discriminative feature representations.
- Capacity: Extended convolution allows models to learn richer feature representations and hence to have improved performance on modern computer vision tasks.
- Generalisation: The complex feature representation can generalise better to potential future data. This helps to improve performance and robustness of computer vision models.

(c) Design and write down a 3 × 3 kernel that discards the green channel and switches the red and blue channels before averaging over the kernel window. (1 mark)

The designed neural structure and its kernels are listed as bellow:

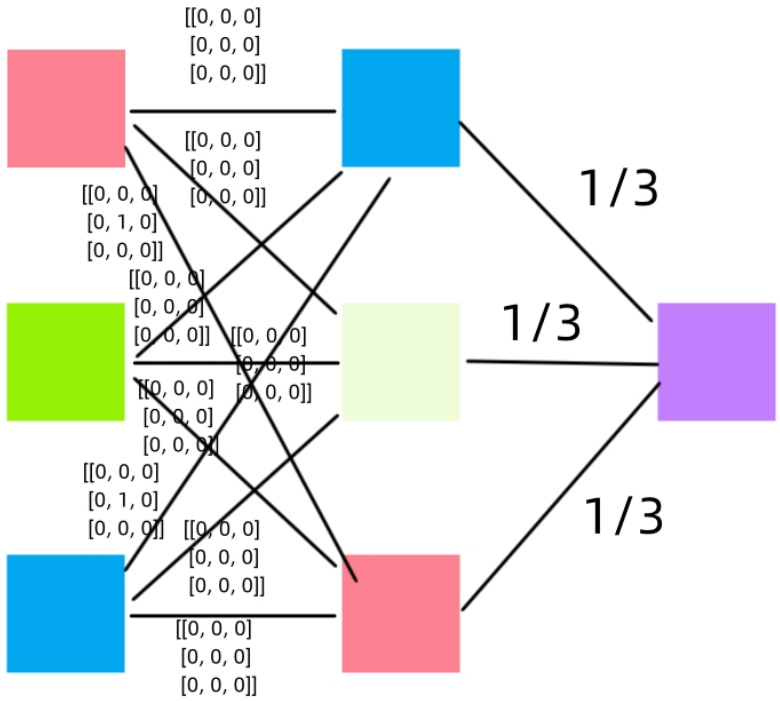

Figure 43 The Designed Network Structure and Proposed Filter

The 0-kernel is used to discard the input, and the S = [[0, 0, 0], [0, 1, 0], [0, 0, 0]] kernel is used to keep the original pixel values. After the convolution operation on the input layer, the green kernel is discarded by the 0-kernel, and the blue and red kernels switch positions by using the kernel S and 0-kernel before being averaged.