SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Vision and Perception

2020 June

Nijat Mursali | 1919669

HOMEWORK 2

Rome, Italy

Exercises

1. Find out if circle and ellipse are separable as filters.

Firstly, we need to define what is separable. As we know from the lecture, a 2-D function $f(x, y)$ is said to be separable if it is formed by the product of two 1-D functions such as

$$f(x, y) = g(x) \times h(y) = F(\sqrt{x^2 + y^2})$$

So, we know that $rect(x, y)$, $sinc(x, y)$ and (x, y) are separable functions. From the lecture, we know that

$$circ(x, y) = rect(\sqrt{x^2 + y^2})$$

which gives us the result, if $rect(x, y)$ is separable function, then $circ(x, y)$ will also be a separable function as well. We can also show it in image where we can see the circle is also separable

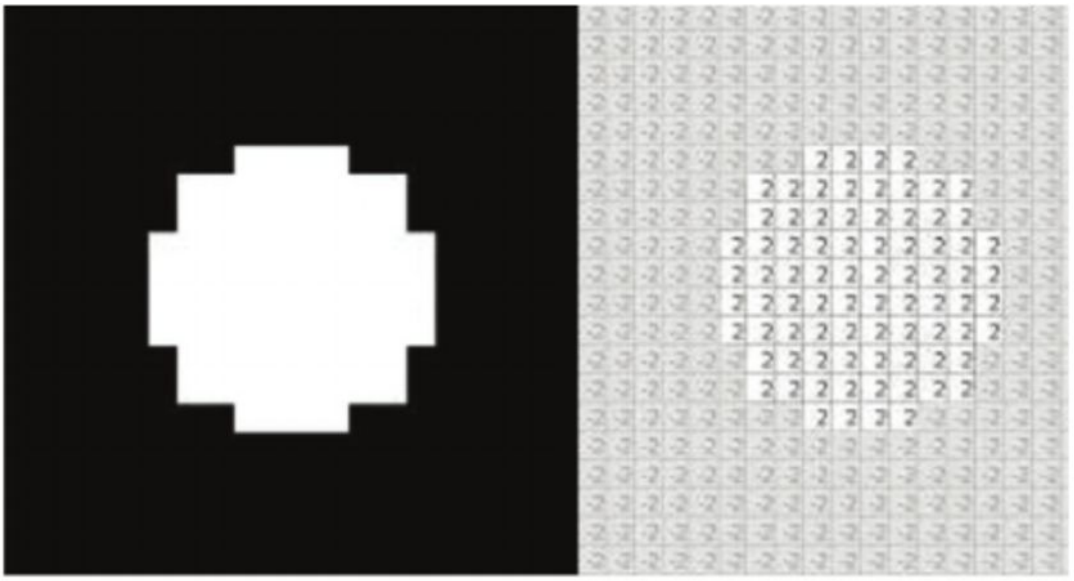


Fig. Circular Separable Filter

We can also do the same thing with an ellipse where we can solve it to show the ellipse with rectangles like we did in the circular image that will give us the result that the ellipse is also separable.

$$ellipse(x, y) = rect(\sqrt{x^2 + y^2}$$

Another way of solving this homework would be dividing the circle and ellipse into triangles where we can fill the circle with several numbers of triangles. As we already know, $rect(x, y)$ is a separable function and we also know rectangles are combinations of triangles

as well, thus we can say triangles are also separable functions. Thus, if we are able to fill the circle and ellipse with triangles we can also say circle and ellipse are also separable functions.

2. Given two images A and B do the Fourier transform of them, show the spectrum then try to reconstruct an image using the phase of A and the magnitude of B.

As we know, the Fourier Transform is one of the important image processing tools that can be used to decompose an image. In order to solve the problem, first we need to understand how exactly Fourier Transform works. For a square image of size $N \times N$, the two-dimensional Discrete Fourier Transform is given by:

$$F(k,\ l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\ e^{-i2\pi(\frac{k_i}{N}+\frac{t_j}{N})}$$

Where $f(a,\ b)$ is the image in the spatial domain and the exponential term is the basis functions corresponding to each point $F(k,\ l)$ in the Fourier space.

The Fourier Transform produces a very complex number valued output image which can be displayed with two images, either with the real and imaginary part or with magnitude and phase. In image processing, often only the magnitude of the Fourier Transform is displayed, as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image.

In order to do the exercise, first we needed to start off applying the Fourier Transform for the image. We have taken the beautiful view of Colosseum and transformed that into 256×256 image as shown in the following picture:

Fig. Resized of Original Image

Then, we have calculated the magnitude from the complex result which is shown in following picture:

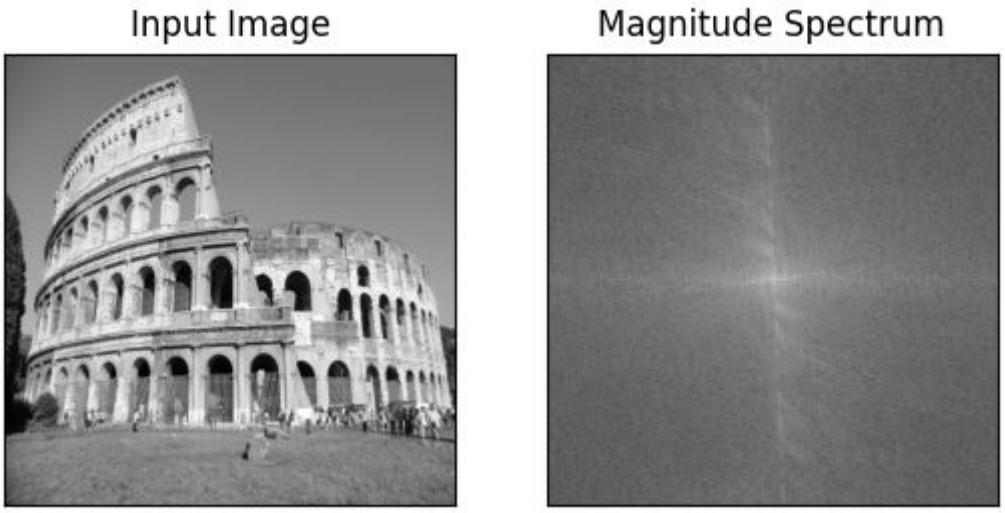


Fig. Magnitude of Image

and the phase spectrum is as following:

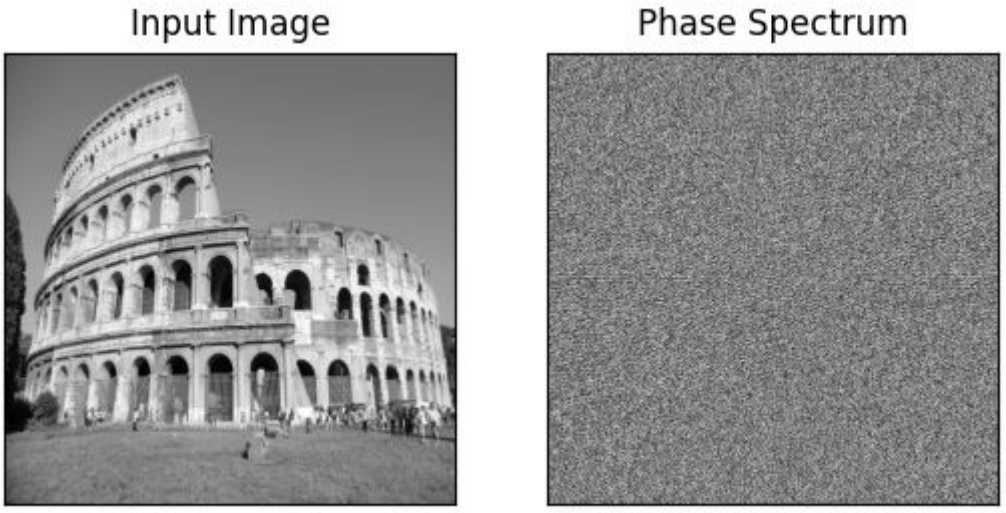


Fig. Phase Spectrum of Image

So, for the second image, we took the beautiful view of Pantheon and rescaled it again into 256×256 and made the Fourier transformation on them which we will show in the following paragraph.

Fig. Resized of Original Image

Then we applied both magnitude and phase spectrum for the resized image and the following two images are the results:

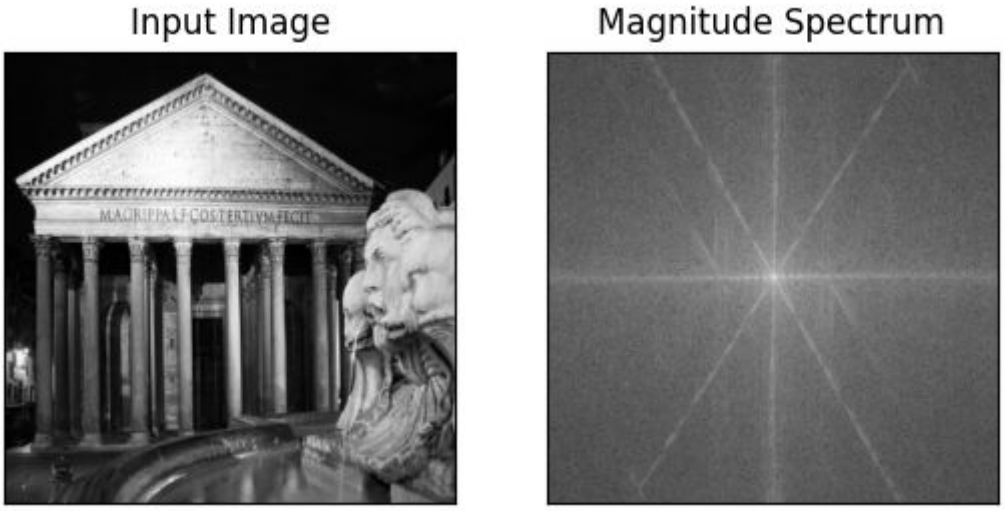


Fig. Magnitude of Image

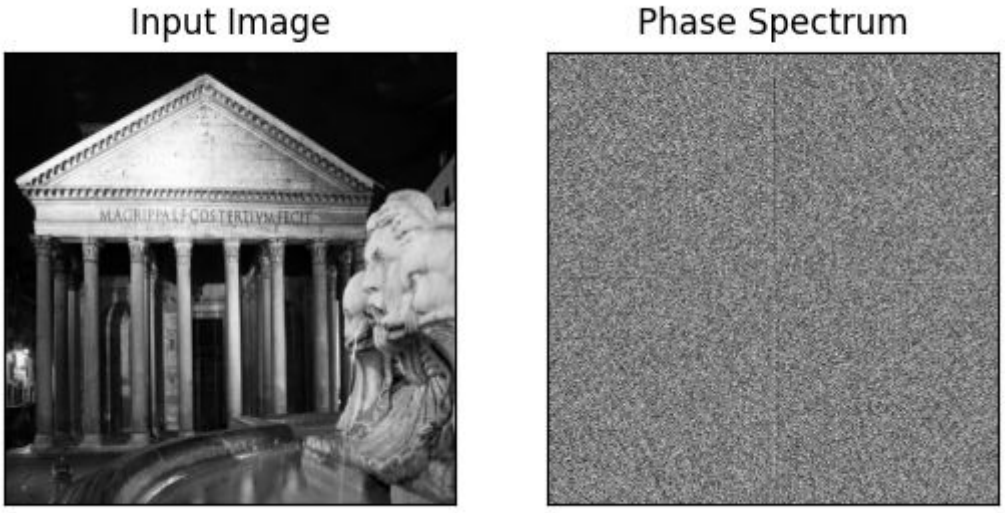


Fig. Phase of Image

Finally, it was asked to reconstruct an image with the phase of *A* and magnitude of *B* images. The problem I had in this part was that in the result it just showed the white and black parts of the image and those edges were not clear enough.

Fig. Result of Phase of A and Magnitude of B

3. Given an image A, use the convolution with the LoG filter to obtain the edges of the image, do the same using the Sobel filter and the Laplacian.

Edge detection is one of the main concepts of image processing where it is beneficial to reduce the number of pixels to process. For this exercise it was asked to apply the LoG, Sobel and Laplacian filter in order to get the edges of the image.

Before going deep into the convolution, we first need to know that an image is a multi-dimensional matrix where it holds the width and a height that matrix has. In convolution, there are three components that are important to consider for image processing:

1. An input image.

2. A kernel matrix that we need to apply for the image
3. An output image to store the output of the input image convolved with the kernel.

In order to solve the exercise, we first applied the LoG (Laplacian of Gaussian) filter in order to reduce the noise effect by smoothing the image. Mathematically, LoG can be achieved by:

$$LoG(x,\ y)\ =\ -\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We have achieved this using OpenCV by just calling *cv.GaussianBlur();*. Then, we were able to apply the Laplacian and Sobel filters to our image. Before going into the details, we chose the picture of a flower that would be beneficial for us to detect the edges.

Fig. Original Image with Gray Color

A Laplacian edge detector uses one kernel in order to calculate the second order derivatives in a single pass. A kernel used in this Laplacian detection looks like the following matrix:

$$A = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The fundamental problem with the Laplacian filter is that it is very sensitive noise as we work with second order derivatives, but as we have applied Gaussian blur in the beginning there will not be any problem. The following figure shows the result after applying the Laplacian filter.

;

Fig. Laplacian Edge Detector

The Sobel edge detector is a gradient based method which works with the first order derivatives. It calculates the first derivatives of the image separately for the X and Y axes. The derivatives are only approximations and in order to approximate them we need to use the following kernels for convolution:

$$H = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Fig. Horizontal and Vertical in the Sobel edge Detection

Fig. Vertical Sobel Edge Detector

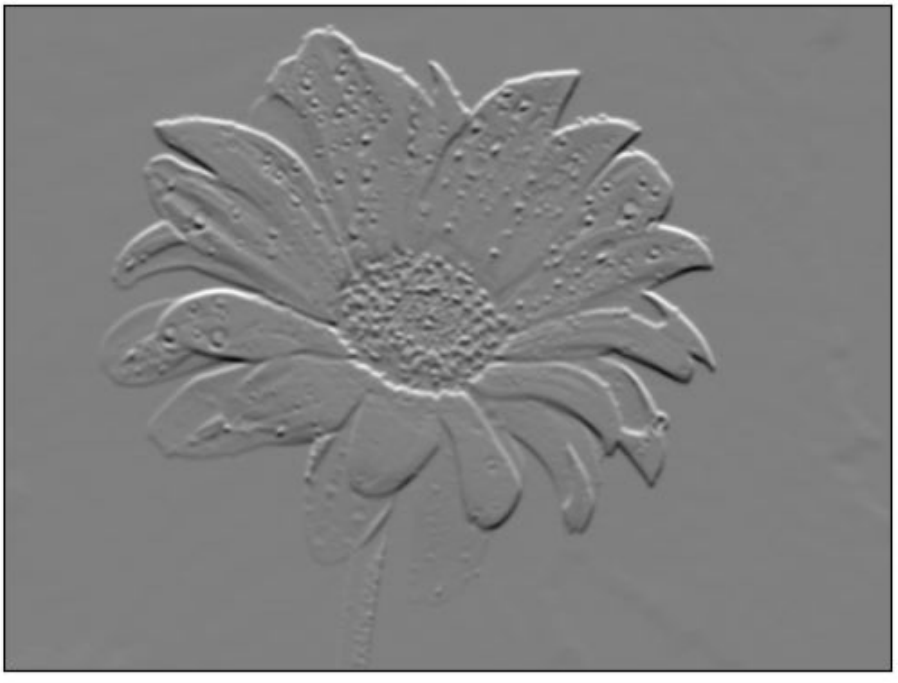


Fig. Horizontal Sobel Edge Detector

4. Take an image, transform it into a gray image. Lower its resolution, applying also a Gaussian filter so that the intensity value is reduced. Implement the histogram equalization, note that it amounts just to compute the formula (*) reported in slide 7 of Video Histogram and Entropy. Show the intensity values of the probabilities before and after in a table. Compute the entropy of the image, before and after the equalization.

The fundamental idea of this homework was to compute the entropy of the image, but before that we needed to do several steps. First, we have used OpenCV for this task, and in the beginning we have used $cv.COLOR\_BGR2GRAY$ tag in order to convert the normal colored image into grayscale colored image. After converting into gray image, we have applied simple dimensions to resize the image into 256×256 image. Next, we have applied Gaussian filter in order to lower resolution. *Fig n.* shows the result of the image after lowering resolution by using a Gaussian filter to reduce the intensity value.

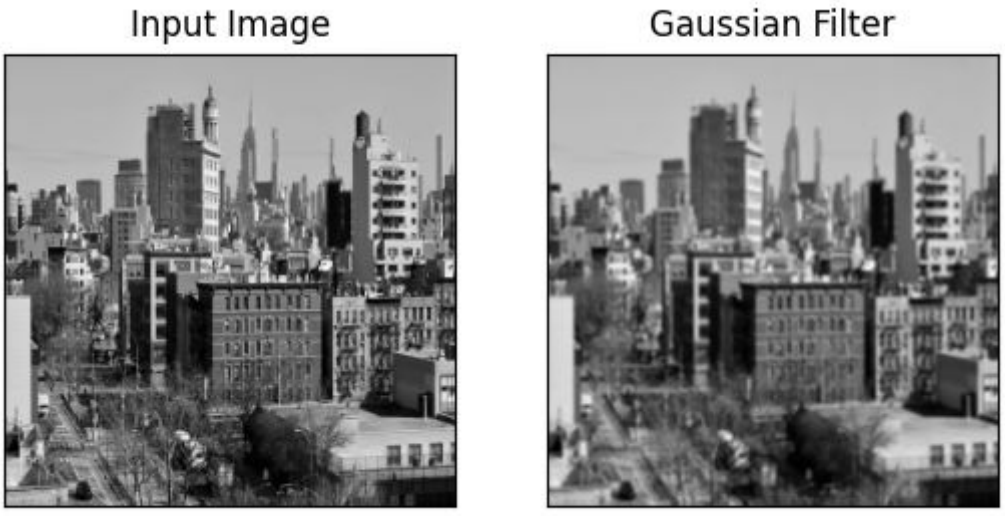
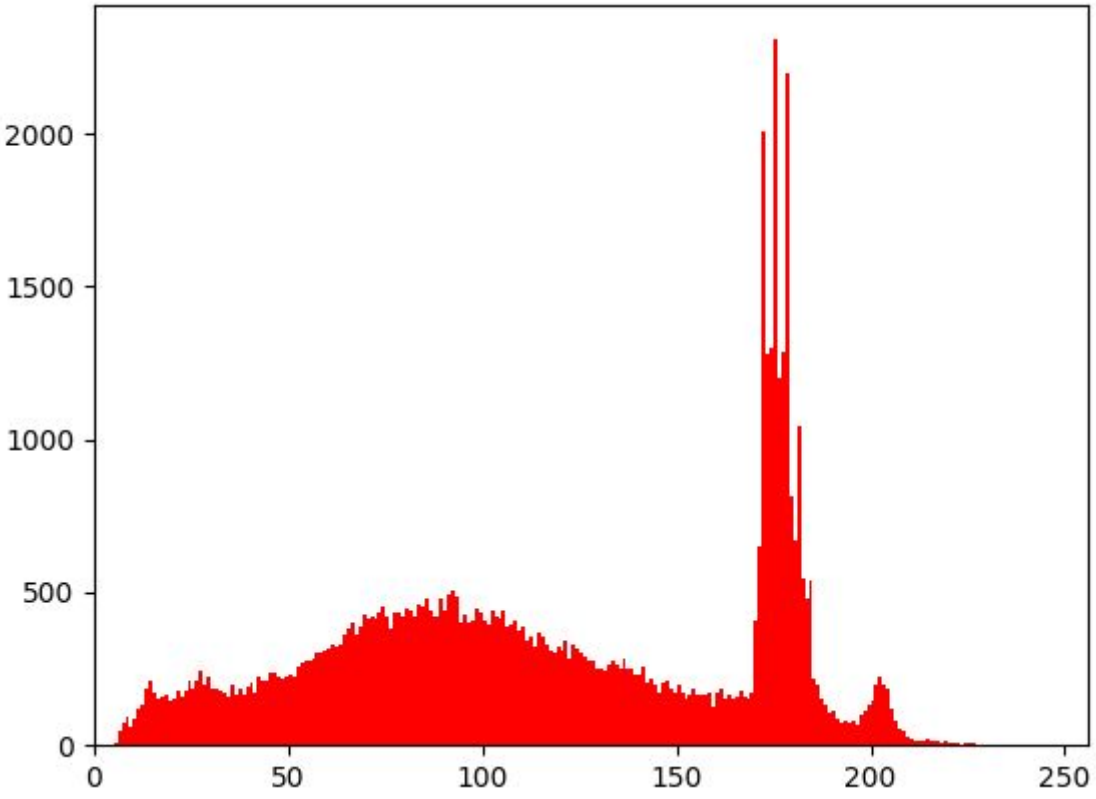


Fig n. Lowering resolution of Image

After that step, we applied histogram equalization that also comes with the *OpenCV* library which gave us the following results.

Fig n. Result of Histogram Equalization



Next, we were asked to compute the entropy of the image before and after the equalization. As we have already done the equalization part, we just needed to add the images to get the entropy that is shown in following figures where the first one is the entropy before equalization and second one is the entropy after the equalization.

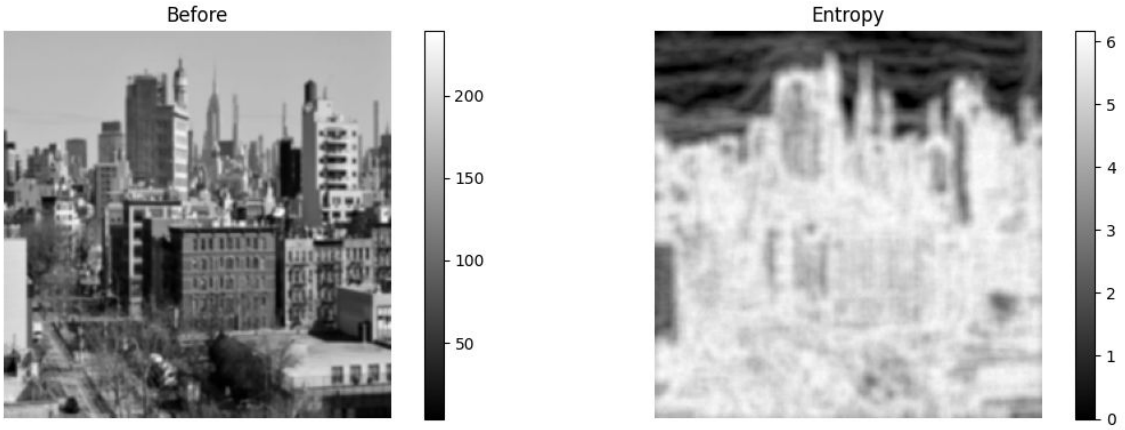


Fig. Entropy before Equalization

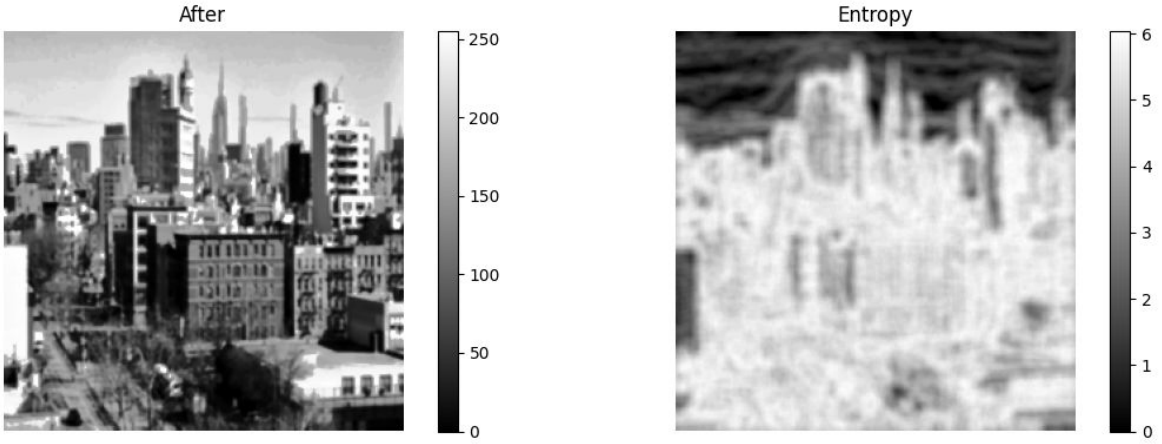


Fig. Entropy after Equalization

5. Is the Harris corner detection invariant with respect to affine transformations and intensity? Tell in three words when a region of an image should be considered salient.

Before explaining the fundamental problem about the case, let's first explain what Harris corner detector is. Harris corner detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. The Harris affine detector can identify similar regions between the images that are related through affine transformations and have different illuminations.

These affine-invariant detectors should be capable of identifying similar regions in images taken from different viewpoints that are related by a simple geometric transformation which are scaling, rotation and shearing. These detected regions have been called both invariant and covariant. However, the regions are detected invariant of the image transformation but the regions covariantly change with image transformation. Harris affine detector relies on the combination of corner points detected through Harris corner detection, multi-scale analysis through Gaussian scale space and affine normalization using an iterative affine shape adaptation approach.

Additionally, as we have discussed in first homework, given the two triangles formed by an arbitrary collection of lines can have several points that intersect and make the edge of the triangles. Then, we can calculate the area of the two triangles using the *Heron's relation:*

$$S = \sqrt{s(s-a)(s-b)(s-c)}$$

where we can show the relationship for both triangles as in shown formulas

$$S_A = \sqrt{s_A(s_A - \overline{p_1p_3})(s_A - \overline{p_1p_2})(s_A - \overline{p_2p_3})} \text{ and } S_B = \sqrt{s_B(s_B - \overline{q_1q_3})(s_B - \overline{q_1q_2})(s_B - \overline{q_2q_3})}$$

We can show that, for the both triangles, their respective areas can be simply described by the distance of the intersecting points. In this case, the intersecting points will be the edges of the triangles and in the image we can also say that affine transformations preserve the intersections and parallel lines, thus the corner detector should be always able to find corners in the image.

For the image to be considered salient there should be *greater amount of noise* to get better salient region detection.

6. Blend two images. Use skimage.transform and only pyramid expand and pyramid reduce. You can generate the mask interactively. For those who are interested in blending try to do it pseudo automatically, and give criteria.

In this exercise, it was asked to blend two images by the help of *skimage.transform* and we could only use *pyramid_expand* and *pyramid_reduce*. Before going deep into the implementation, let's first explain what blending really is. Blending procedure takes the two images and the mask, and splits them into RGB (which stands for red, green and blue)

channels. After splitting into channels, it then blends every channel separately into parts. Then, we need to care about two important things: laplacian pyramid and gaussian pyramid. The Laplacian pyramid will be constructed for the two images and by using the mask it will construct a gaussian pyramid for it. At last, it blends two pyramids and collapses them down to the output image. There are two cases we need to care about:

1. Reduce

   In reduce function, it takes the image and subsamples it down to a quarter of the size by simply dividing the height and width of the image into two.

2. Expand

   Instead of reducing function, this function takes an image and sumsamples it four times the size, but with multiplying the height and width of two. After increasing the size, we have to interpolate the missing values by running over it with a *smoothing filter.*

In the Gaussian Pyramid we take the image and build a pyramid out of that image. This pyramid basically contains several layers inside that each of them does some kind of job inside. The first layer is just an original image that we imported and other layers are the reduced form of the previous layer. In the Laplacian Pyramid we simply take the gaussian pyramid and turn it into the laplacian pyramid.

The next step in this pipeline is to implement a blend function in which we simply take the both gaussian and laplacian pyramids and then implement a collapse function. Our blend function takes three arguments or pyramids simply:

1. white - a *laplacian pyramid* of an image A
2. black - a *laplacian pyramid* of image B
3. mask - a *gaussian pyramid* of a mask image

For the demonstration, we have used two images: orange and apple images and resized them into 512×512. We took the rectangle mask for both black and white sides. The result will be as following:

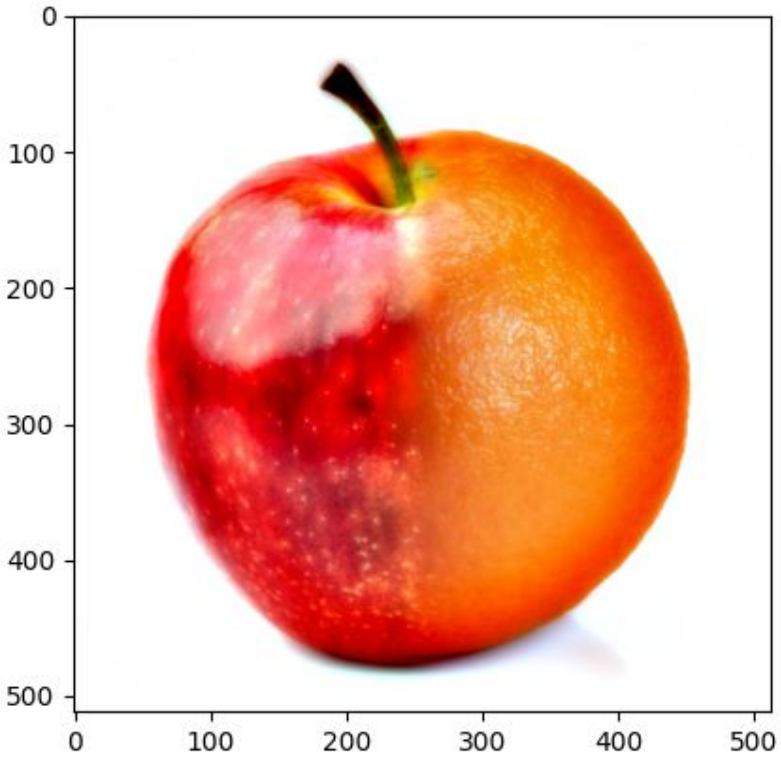


Fig. Output Image

7. Compute $H(Y|X)$ in bits recalling $p(x \mid y)\, p(y) = p(x,\ y)$

As we know from the lectures, when we use log_2 the entropy is measured in bits, otherwise it is measured in nats. The entropy in *bits* can be calculated like following:

$$H(X) \ = \ \sum_k p(x = k)\ log\ p(x = k)$$

If we consider $p_x(k)$ as a distribution of intensity values then entropy is a measure of histogram dispersion. If the entropy is a function of the distribution of X, then it does not depend on the realization of the random variable. Then, we will have the formula like following

$$H(X) = -\ p\ log\ p - (1 - p)\ log(1 - p) = H(p)$$

If we consider the joint distribution of two discrete random variables X and Y, then:

$$H(X,\ Y) = -\sum_{x \in X}\sum_{y \in Y} p(x,\ y)\ log_2\ p(x,\ y) = -\ E\ log_2\ p(x \mid y)\ p(y)$$

where *E is the expectation.* The conditional entropy is

$$H(Y \mid X) = -\sum_{x \in X} p(x)\ H(Y \mid X = x) = \sum_{x \in X} p(x) \sum_{y \in Y} p(y \mid x)\ log_2\ p(y \mid x)$$

which it can also derive that

$$H(Y \mid X) = -\sum_{x \in X}\sum_{y \in Y} p(x,\ y)\ log\ \frac{p(x,y)}{p(x)} = -\ (\frac{1}{8} log \frac{\frac{1}{8}}{\frac{1}{2}} + \frac{1}{16} log(\frac{\frac{1}{16}}{\frac{1}{4}}) + \frac{1}{32} log(\frac{\frac{1}{32}}{\frac{1}{8}}) +$$

$$+ \frac{1}{32} log(\frac{\frac{1}{32}}{\frac{1}{8}}) + \frac{1}{16} log \frac{\frac{1}{16}}{\frac{1}{2}} + \frac{1}{16} log(\frac{\frac{1}{16}}{\frac{1}{4}}) + \frac{1}{8} log \frac{\frac{1}{8}}{\frac{1}{4}} + \frac{1}{32} log \frac{\frac{1}{32}}{\frac{1}{8}} + \frac{1}{32} log \frac{\frac{1}{32}}{\frac{1}{8}} +$$

$$+ \frac{1}{16} log \frac{\frac{1}{16}}{\frac{1}{2}} + \frac{1}{16} log \frac{\frac{1}{16}}{\frac{1}{4}} + \frac{1}{16} log \frac{\frac{1}{16}}{\frac{1}{8}} + \frac{1}{16} log \frac{\frac{1}{16}}{\frac{1}{8}} + \frac{1}{4} log \frac{\frac{1}{4}}{\frac{1}{2}}) =$$

$$= -\ (\frac{1}{8} * 2 + \frac{1}{16} * 2 + \frac{1}{32} * 2 + \frac{1}{32} * 2 + \frac{1}{16} * 3 + \frac{1}{16} * 2 + \frac{1}{8} * 1 + \frac{1}{32} * 2 +$$

$$+ \frac{1}{32} * 2 + \frac{1}{16} * 3 + \frac{1}{16} * 2 + \frac{1}{16} * 1 + \frac{1}{16} * 1 + \frac{1}{4} * 1) =$$

$$= -\ (\frac{2}{8} + \frac{2}{16} + \frac{2}{32} + \frac{2}{32} + \frac{2}{16} + \frac{2}{16} + \frac{1}{8} + \frac{2}{32} + \frac{2}{32} + \frac{3}{16} + \frac{2}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{4} =$$

$$= -\ \frac{27}{16}$$

where $P(X) = [1/2\ 1/4\ 1/8\ 1/8]^T$

Thus, the final result for $H(Y \mid X) = -\ \frac{27}{16}$.