Computer vision :: Assignment #1

Image Filtering and Hybrid Images

Student Name: Behnam Moradi

Introduction:

This is the first assignment of a wonderful course on computer vision at the University of Regina. In this assignment we the students are supposed to dive into image filtering and one of it interesting application in creating hybrid images. At the very first step, an image filtering function has been developed from scratch accepting an image and a Gaussian filter kernel as its inputs. The second step was to write another function (from scratch again 🙂) to generate hybrid images. This function uses the developed image filtering function accepting two relevant images and the Gaussian filter kernel as its inputs. Also, given that a Gaussian filter is provided for us, but as an interesting part of this assignment, a faction has been developed to create a Gaussian filter kernel from scratch. This way one is able to simply understand the whole idea behind the provided filter.

Image Filtering

Why we filter images? In this part, I try to answer this question. An image is nothing but a 2D signal and can be modeled mathematically by the f (x, y) function. x and y are the exact locations on each pixel in the image. The output of this function is an uint8 integer value from 0 to 255. So basically, an image is simply a matrix and each element of this matrix has the pixel value stored in it. Having said that there exist 2 image types: one is grayscale and the other is colored. A grayscale image is a 2D matrix and a colored image is a 3D matrix and each dimension of this 3D matrix is one color channel of red, green, and blue. That is why we call it the RGB image. My algorithm will support both grayscale and RGB images as input.

As we discussed, the image is a signal and each signal carries some information in form of frequency. So basically, image processing is a specific sort of Signal Processing. Specifically, dealing with images, we are filtering for the following reasons:

1. Reducing the noise
2. feature detection (Edge detection specifically)
3. Blurring (Low-pass signal filtering)
4. Sharpening (High-pass signal filtering)

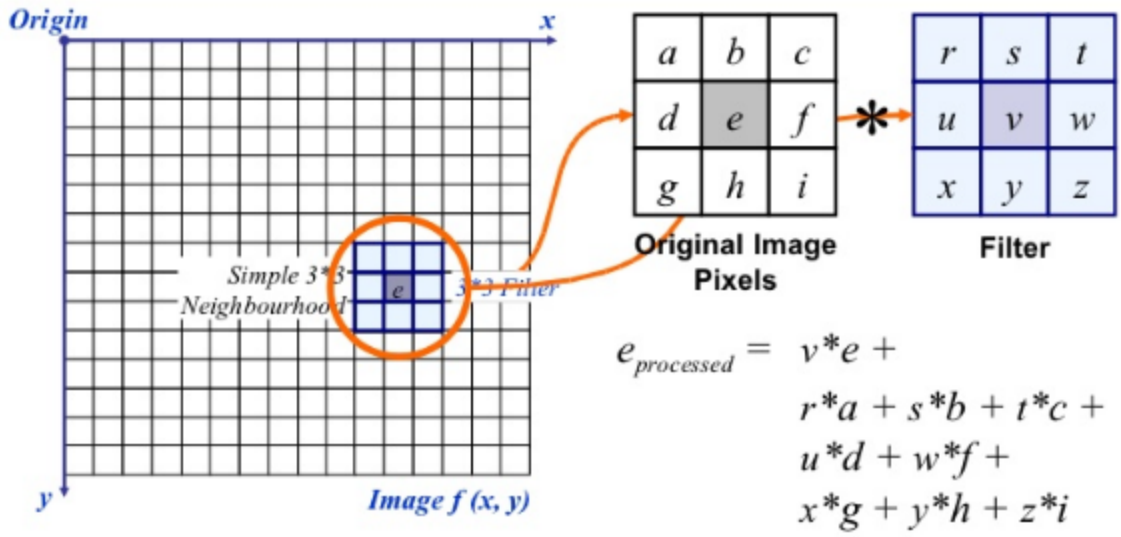
In this assignment, we the students are required to create a low-pass filtering algorithm to make an input image clear from its high-frequency signals. The generated low-frequency image will be used to generate the high-frequency image. So that is gonna be incredibly easy. Here are the steps:

1. Create a low-pass Gaussian filter
2. Use the created filter to generate a blurred(e.g., low-frequency) image
3. Use the blurred image to generate a sharpened (e.g., high-frequency) image by subtracting it from its original image

The Gaussian Filter

The NxN Gaussian filter can be generated by the Gaussian normal distribution function. this function will receive the element location (x, y) and a standard deviation factor. The output of this function will be the value of the respective (x, y) element inside the filter matrix. Having said that (x, y) can only accept ODD values. The center of the filtering matrix will be considered as (0, 0).

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



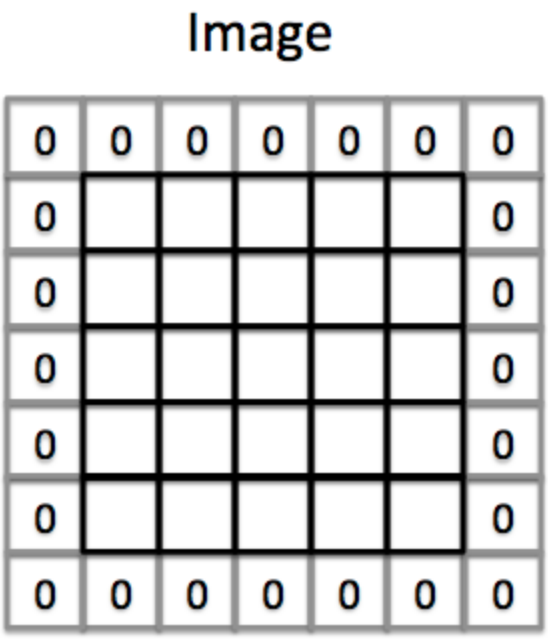
Now we know the whole idea behind Gaussian filtering and we are ready to implement it in python. The following function will receive kernel_dimenssion and standard deviation as its inputs and provides the filtering matrix as output.

Note: Gaussian filter is also provided by the existing python module 🙂 .

```
def create_gaussian_filter(kernel_size=3, sigma=1.0):
  kernel = np.zeros((kernel_size,kernel_size))
  i = -int(kernel_size/2)
  for r in range(0, kernel_size):
    j = -int(kernel_size/2)
    for c in range(0, kernel_size):
      kernel[r, c] = (1.0/(2.0*np.pi*sigma*sigma))*np.exp(-(i**2 + j**2)
/(2.0*sigma*sigma))
      j = j+1
    i = i+1
  # Kernel Normalization
  kernel = (kernel)/(kernel.max())
  return kernel
```

Image Padding

Image padding is necessary step before starting to filter an image. Image padding is simply adding zero layesr to the image margines. based on filter dimensions (NxN), we need to add int(N/2) zero layes to our input image.



have developed the following function to do this step. This function supports both color and grayscale images as inputs. In the results section, an output of this function is provided.

```
# Function: Will create image padding
def img_padding(img_data, kernel):
  padding_number = int(int(np.sqrt((kernel.size)))/2)
  print("Padding Number = " + str(padding_number))
  image_shape = img_data.shape
  if len(image_shape) == 3:
    i, j, k = img_data.shape
    zeros_column = np.zeros((1, i))
    zeros_row =  np.zeros((1, j + 2*padding_number))

    red_channel = img_data[:,:,0]
    green_channel = img_data[:,:,1]
    blue_channel = img_data[:,:,2]

    # for loop: to add zero padding to columns
    counter = 1
    for l in range(0, padding_number):

      red_channel = np.insert(red_channel, 0, zeros_column, axis=1)
      red_channel = np.insert(red_channel, j + counter , zeros_column,
axis=1)

      blue_channel = np.insert(blue_channel, 0, zeros_column, axis=1)
      blue_channel = np.insert(blue_channel, j + counter ,
zeros_column, axis=1)

      green_channel = np.insert(green_channel, 0, zeros_column, axis=1)
```

```
      green_channel = np.insert(green_channel, j + counter ,
zeros_column, axis=1)

      counter = counter + 1

    # for loop: to add zero padding to rows
    counter = 1
    for p in range(0, padding_number):
      red_channel = np.insert(red_channel, 0, zeros_row, axis=0)
      red_channel = np.insert(red_channel, i + counter , zeros_row,
axis=0)

      blue_channel = np.insert(blue_channel, 0, zeros_row, axis=0)
      blue_channel = np.insert(blue_channel, i + counter , zeros_row,
axis=0)

      green_channel = np.insert(green_channel, 0, zeros_row, axis=0)
      green_channel = np.insert(green_channel, i + counter , zeros_row,
axis=0)

      counter = counter + 1
    paded_image = np.stack((red_channel, green_channel, blue_channel),
axis=2)
    return paded_image

  # if condition: grayscale images
  if len(image_shape) != 3:
    i, j= img_data.shape
    zeros_column = np.zeros((1, i))
    zeros_row =  np.zeros((1, j + 2*padding_number))
    # for loop: to add zero padding to columns
    counter = 1
    for l in range(0, padding_number):

      img_data = np.insert(img_data, 0, zeros_column, axis=1)
      img_data = np.insert(img_data, j + counter , zeros_column, axis=1)
      counter = counter + 1

    # for loop: to add zero padding to rows
    counter = 1
    for p in range(0, padding_number):

      img_data = np.insert(img_data, 0, zeros_row, axis=0)
      img_data = np.insert(img_data, i + counter , zeros_row, axis=0)
      counter = counter + 1
    return img_data
```

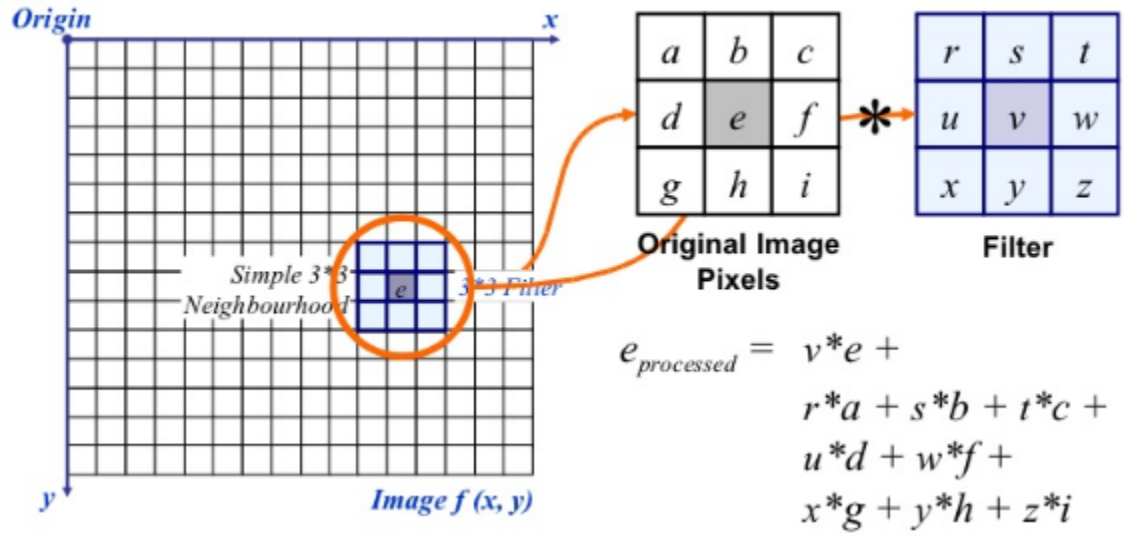
Image filtering algorithm

my_imfilter() function receives an image and a filter kernel as inputs and provides a filtered image as its output. At the very beginning of the algorithm, the image_padding() function will take care image padding step. Then it will check the image type (e.g., colored or grayscale). Based on the following figure, it starts looping over the image pixel by pixel.

As it can be seen, the value of pixel number “e“ is calculated by the following dot product of filter and its respective part on the original image.

my_imfilter() function is developed to generate a low-frequency image from its original one. To extract the high-frequency one, simply we can clear the original image from its low-frequency:

High-freq_image = Original_image - Low_frq_image



```
# Function: will filter the inpute image. It supports colored and
grayscale images
def my_imfilter(image, filter):
  image = img_padding(image, filter)
  # show_img([])
  # filter = np.dot(filter, filter.T)
  pad_number = int(int(np.sqrt((filter.size)))/2)
  image_shape = image.shape
  if len(image_shape) == 3:
    i, j, k = image.shape
    low_pass_filtered_img = image.copy()
    for dim in range(0, k):
      print("dim = " + str(dim))
      for row in range(0 + pad_number, i - pad_number):
        for col in range(0 + pad_number, j - pad_number):
          sep_data = image[row - pad_number:row + pad_number+1, col -
pad_number:col + pad_number+1, dim]*filter
          low_pass_filtered_img[row, col, dim] = np.sum(sep_data)
/filter.size
    return low_pass_filtered_img[0 + pad_number:i - pad_number, 0 +
pad_number:j - pad_number]

  if len(image_shape) != 3:
    i, j = image.shape
    low_pass_filtered_img = image.copy()
    for row in range(0 + pad_number, i - pad_number):
      for col in range(0 + pad_number, j - pad_number):
        sep_data = image[row - pad_number:row + pad_number+1, col -
pad_number:col + pad_number+1]*filter
        low_pass_filtered_img[row, col] = np.sum(sep_data)/filter.size
    return low_pass_filtered_img[0 + pad_number:i - pad_number, 0 +
pad_number:j - pad_number]
```

Generating Hybrid Image

A hybrid image is simply an image with two different interpretations from near and far distances. You might ask what does it mean? 🙂. It simply means:

considering this scenario:

Image #1: we generate its low-frequency image and we call it the low-frequency image

Image #2: we generate its high-frequency image and we call it the high-frequency image

Hybrid Image = (low-frequency image) + (high-frequency image)

1. From a near distance: it looks like image #1
2. from the far distance, it looks like image #2

The following function generates a hybrid image from relevant images. It receives two images and a filter kernel and provides the following outputs:

1. Low-frequency Image

2. High-frequency Image
3. Hybrid Image

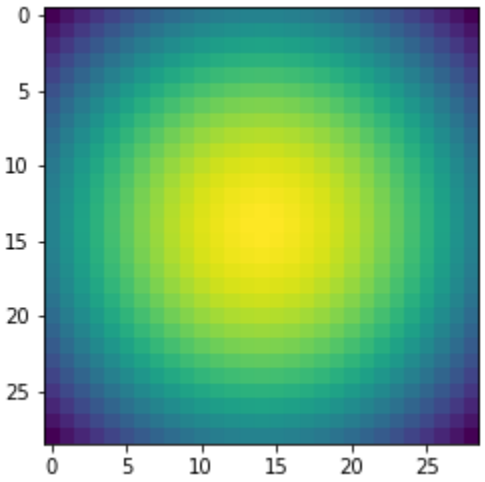
The create_hybrid_image() function is developed to do this step.

```
def create_hybrid_image(image1, image2, filter):
  low_freq = my_imfilter(image1, filter)
  high_freq = image2 - my_imfilter(image2, filter)
  hybrid_image = high_freq + low_freq
  print(high_freq)
  return[low_freq, high_freq, hybrid_image]
```

Results

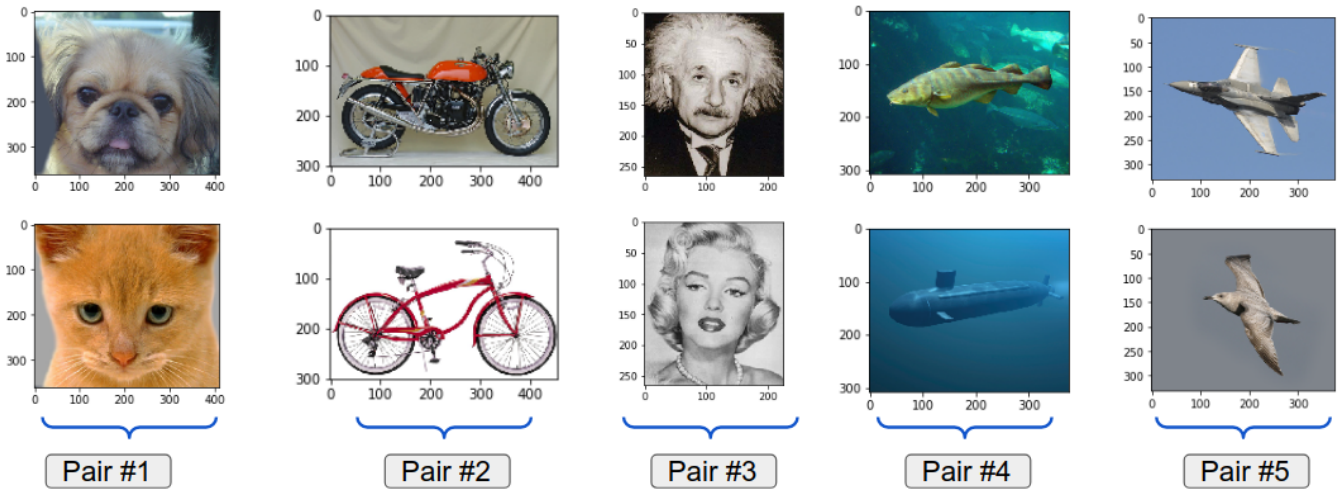
The Gaussian Filter

Dimensions: 29x29
Standard deviation: 21

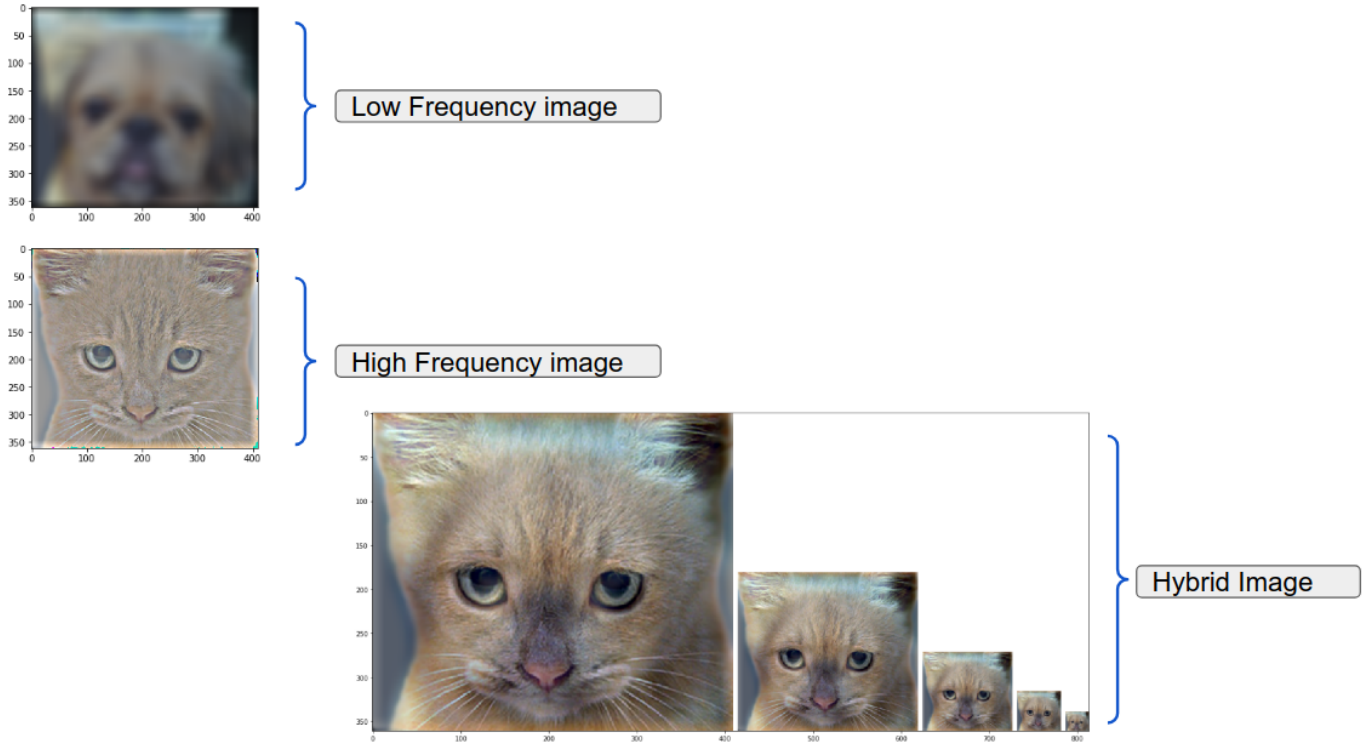


Input Image pairs

In this assignment the following image pairs have been used to generate 5 hybrid images:


Pair #1
Pair #2
Pair #3
Pair #4
Pair #5

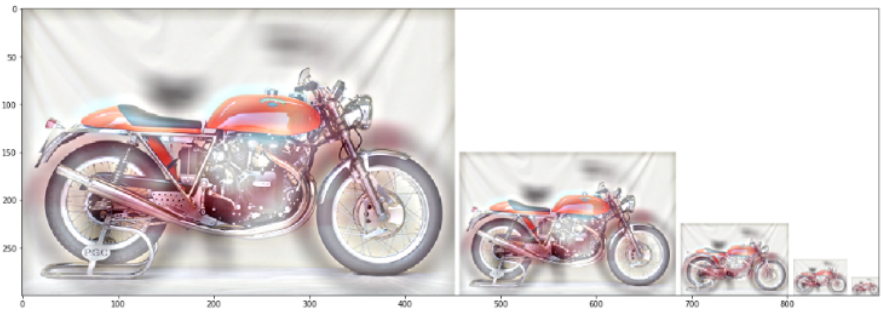
Pair #1


Low Frequency image
High Frequency image
Hybrid Image

Pair #2

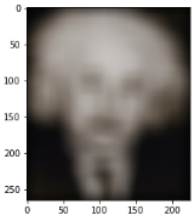
Low Frequency image

High Frequency image

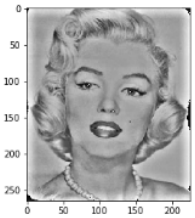


Hybrid Image

Pair #3



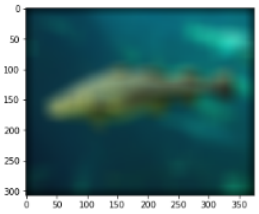
Low Frequency image



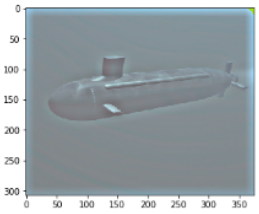
High Frequency image

Hybrid Image

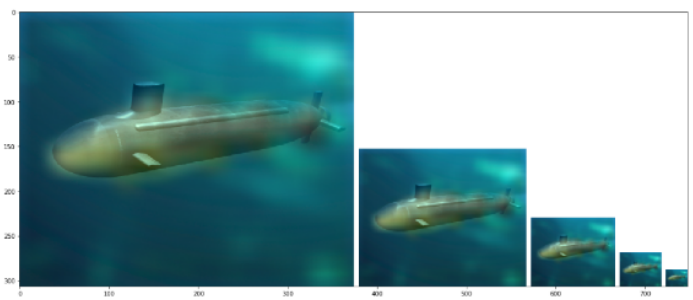
Pair#4



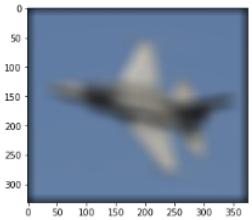
Low Frequency image



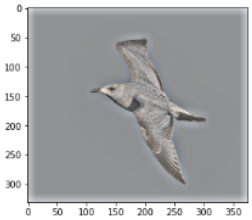
High Frequency image



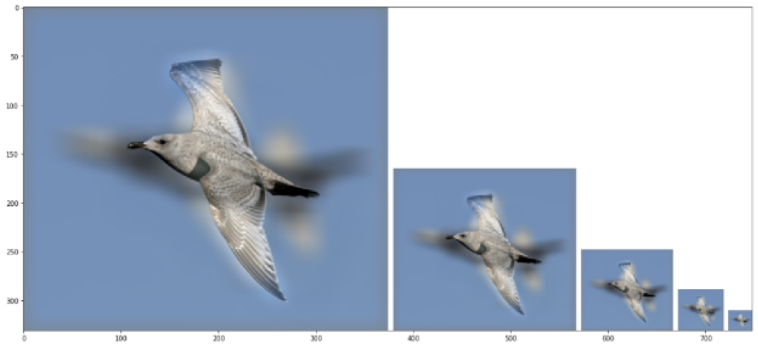
Hybrid Image



Low Frequency image



High Frequency image



Hybrid Image

Conclusion

In this assignment, we have developed an image filtering algorithm to generate a low-pass filtered image from its original ones. Then we used this function to generate hybrid images by making a summation of low-frequency and high-frequency images. We have tested the whole algorithm by importing 5 pairs of related images. Some important points to consider:

1. One needs to make sure that the image#1 & #2 can be used to create hybrid images. Non-relevant images might end up with false results.

2. There are two factors that one needs to tune to get reliable results:
   - Kernel Size
   - Standard Deviation

The End