Ex. 1 - Box Blur (box linear filter)

In this exercise, you will implement the 2-D convolution operation without using the OpenCV library. You will consider a grayscale input image, stride equals to 1, and a kxk blur filter (k is an input parameter) to obtain a blurred image as output. Input and output images should have the same spatial size.

Tip: Use zero-padding to get the same spatial size for both input and output images. The padding value should be set to (k-1)/2, where k is the kernel size.

```python
def box_filter(image, kernel_size):
  '''
  Apply a kxk box filter to the input image.
  Inputs
  -----------
  image: np.array
    Input image.
  kernel_size: int
    Size of the squared kernel.
  Outputs
  -------
   output: np.array
   Filtered image.
  '''

  # Check input parameters
  assert len(image.shape) == 2, f"Input image has {image.shape[-1]} channels. \
                          Grayscale image is required."
  assert kernel_size % 2 != 0, "Kernel size must be an odd number."

  image = np.asarray(image, dtype=np.float32)

  # Initialize output image
  output = np.zeros_like(image)

  # Kernel definition
  alpha = 1 / (kernel_size ** 2)
  kernel = np.full((kernel_size, kernel_size), alpha)

  ## PADDING SECTION
  # Define padding size
  padding_size = (kernel_size - 1) // 2
  # Create new image
  image_padded = np.zeros((image.shape[0] + 2 * padding_size, image.shape[1] + 2
* padding_size))
  # Put input image into padded image
  image_padded[padding_size:-padding_size, padding_size:-padding_size] = image

  ## 2-D CONVOLUTION
  # Loop over all pixels of the input image
  for p in itertools.product(range(image.shape[0]), range(image.shape[1])):
      # Perform 2-D convolution
      # Extract a kxk patch and convolve it with the kernel
      patch = image_padded[p[0]:p[0]+kernel_size, p[1]:p[1]+kernel_size]
      output[p] = np.sum(patch * kernel)
  return output
```

Ex. 2 - Median filter

In this exercise, you will implement the median filter replacing each pixel of the input image with the median of its neighborhood. The median value is computed by sorting all the neighborhood values of the selected pixel in ascending order and then by replacing its value by the pixel value in the middle. Input and output images must have the same spatial size. The kernel size must be an odd number.

```
def median_filter(image, kernel_size, padding=True):
  '''
  This function applies the median filter to the input image.

  Inputs
  -----------
  image: np.array
    Input grayscale image
  kernel_size: int
    Dimension of a squared kernel.
  padding: bool
    If True, input image already padded.

  Output
  -----------
  output: np.array
    Filtered image
  '''

  # Check input parameters
  assert len(image.shape) == 2, f"Input image has {image.shape[-1]} channels. \
Grayscale image is required."
  assert kernel_size % 2 != 0, "Kernel size must be an odd number."

  # No need to define a kernel
  image = np.asarray(image, dtype=np.float32)

  ## PADDING SECTION
  # Define padding size
  padding_size = (kernel_size - 1) // 2
  if padding:
    # Create new image
    image_padded = np.zeros((image.shape[0] + 2 * padding_size, image.shape[1] +
2 * padding_size))
    # Put input image into padded image
    image_padded[padding_size:-padding_size, padding_size:-padding_size] = image
    # Define output image shape
    n_rows, n_cols = image.shape
  else:
    # Define output image shape
    n_rows, n_cols = image.shape[0] - 2 * padding_size, image.shape[1] - 2 *
padding_size
    # Image already padded
    image_padded = image

  # Output image
  output = np.zeros((n_rows, n_cols))

  ## Apply median filter
  # Loop over all pixels of the input image
  for p in itertools.product(range(output.shape[0]), range(output.shape[1])):
```

```
  image_kxk = image_padded[p[0]:p[0]+kernel_size, p[1]:p[1]+kernel_size]
  values = np.sort(image_kxk.flatten())
  median_value = values[(kernel_size ** 2) // 2]
  output[p] = median_value
return output
```