LAB2 SOLUTIONS
Computer Vision Foundations: Images and Filtering

EX. 1

```
# Perform binary thresholding using four different values: 30, 60, 90, 120.
# As input image, use a grayscale image.

# Using Numpy

thrs = [30,60,90,120]
fig, ax = plt.subplots(nrows = 1, ncols = len(thrs), figsize = (3*len(thrs),
3*len(thrs)))
for i, thr in enumerate(thrs):
      img = np.where(lena_gs <= int(thr), 0, 255 )
      ax[i].imshow(img, cmap=plt.cm.gray)
      ax[i].axis('off')
      ax[i].set_title(f'thrs = {thr}')
plt.show()


# OpenCV2 example
# cv2.threshold(src, thresh, maxval, type[, dst]) → retval, dst
# It returns the used threshold and the thresholded image.

thrs = [30,60,90,120]
fig, ax = plt.subplots(nrows = 1, ncols = len(thrs), figsize = (3*len(thrs),
3*len(thrs)))
for i, thr in enumerate(thrs):
      _, img = cv2.threshold(lena_gs, int(thr), 255, cv2.THRESH_BINARY)
      ax[i].imshow(img, cmap=plt.cm.gray)
      ax[i].axis('off')
      ax[i].set_title(f'thrs = {thr}')
plt.show()
```

EX. 2

```
def box_filter(kernel_size):
'''
Create a k x k box filter.
Input parameters
----------------
kernel_size: int
Size of the squared kernel.

Output parameters
```

```
----------------
Filter: np.array [k x k]
Box filter

To check
------------
kernel size must be an odd number
'''

      assert kernel_size % 2 != 0

      kernel = np.ones([kernel_size, kernel_size]) / kernel_size**2

      return kernel



def conv_2d(image, kernel):
'''
Apply a k x k filter to the input image.
Input parameters
----------------
image: np.array
Input image.
kernel: np.array [k x k]
Filter to apply

Output parameters
-----------------
Image: np.array

To check
------------
Grayscale image is required
'''

 assert len(image.shape) == 2

 kernel_size = kernel.shape[0]

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

 output = np.zeros_like(image)

 kernel = np.flipud(np.fliplr(kernel))

 padding_size = (kernel_size - 1)//2
 img_padded = np.pad(image, (padding_size, padding_size))

 h,w = image.shape
 for row in range(h):
```

```
    for col in range(w):
       output[row,col] = (img_padded[row:row+kernel_size,
col:col+kernel_size]*kernel).sum()

 return output
```

EX. 3

```
def gaussian_2d(size=11, sigma=1):
"""
2-D Gaussian Filter

Input Parameters
----------------
size: int
Size of the squared kernel
sigma: float
Standard deviation

Output Parameter
----------------
filter: np.array
2-D Gaussian Filter
"""
      # Check input parameters
      assert size % 2 != 0, "Kernel size must be an odd number."


      ax = np.linspace(-(size - 1) / 2., (size - 1) / 2., size)
      xx, yy = np.meshgrid(ax, ax)

      kernel = 1/(np.sqrt(2 * np.pi * np.square(sigma))) * np.exp(-0.5 *
(np.square(xx) + np.square(yy)) / np.square(sigma))

      return kernel / np.sum(kernel)
```

EX. 4

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

Input Parameters
-----------------
image: np.array
```

```
Input grayscale image
kernel_size: int
Dimension of a squared kernel.
Padding: bool
If True, the input image is padded with zeros in order to have as output
an image with the same spatial size of the input image.
If False, no padded is applied and the output image will be smaller than
the input one.

Output Parameters
-----------------
image: np.array
Filtered image
'''
 # Check inputs
 assert len(image.shape) == 2
 assert kernel_size % 2 != 0

 image = np.asarray(image,dtype=np.float32)
 h,w = image.shape

 if padding:
      output = np.zeros_like(image)

      padding_size = (kernel_size - 1)//2
      img_padded = np.pad(image, (padding_size, padding_size))

      for row in range(h):
            for col in range(w):
                  output[row,col] = np.median(img_padded[row:row+kernel_size,
col:col+kernel_size])
 else:
      out_h = h - kernel_size + 1
      out_w = w - kernel_size + 1
      output = np.zeros([out_h, out_w])

      for row in range(out_h):
            for col in range(out_w):
                  output[row,col] = np.median(image[row:row+kernel_size,
col:col+kernel_size])

 return output
```

EX. 5

```
lena_blur = cv2.GaussianBlur(lena_gs, (21,21), 10)
lena_details = cv2.addWeighted(lena_gs,1,lena_blur,-1,0)

fig, ax = plt.subplots(nrows = 2, ncols = 3, figsize = (15,12))
ax[0,0].imshow(lena_gs, cmap=plt.cm.gray)
ax[0,0].axis('off')
ax[0,0].set_title('Original')

ax[0,1].imshow(lena_blur, cmap=plt.cm.gray)
ax[0,1].axis('off')
ax[0,1].set_title('Blur')

ax[0,2].imshow(lena_details, cmap=plt.cm.gray)
ax[0,2].axis('off')
ax[0,2].set_title('Details')

for i,k in enumerate([1,2,3]):
      lena_sharp = cv2.addWeighted(lena_gs,1,lena_details,k,0)
      ax[1,i].imshow(lena_sharp, cmap=plt.cm.gray)
      ax[1,i].axis('off')
      ax[1,i].set_title(f'k={k}')
```