Assignment 2: Hybrid Images and Scale-space blob detection

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import os
from PIL import Image
from skimage.filters import gaussian_filter
from skimage.io import imshow, imsave
from skimage.color import rgb2gray, rgb2hsv
from google.colab.patch_import import *
from google.colab.patch_export import *
```

Part 1: Hybrid Images

```
# from the point the images must have the same size and same number of channels
# the function takes two images and a cutoff frequency and returns the hybrid image
# the function also saves the hybrid image in the current directory
# and displays the hybrid image
```

```
# gaussian_kernel(size, sigma) gaussian kernel of size x size
# sigma is the standard deviation
def gaussian_kernel(size, sigma):
    ax = np.linspace(-(size - 1) / 2, (size - 1) / 2, size)
    gauss = np.exp(-0.5 * np.square(ax) / np.square(sigma))
    return gauss / gauss.sum()

def apply_kernel(image, kernel):
    # print("image.shape", image.shape)
    # print("kernel.shape", kernel.shape)
    return cv.filter2D(image, -1, kernel)
```

```
# apply_2_kernels_gray(image, kernel1, kernel2) applies two kernels to a grayscale image
# kernel1 and kernel2 are the same size
# kernel1 is a low pass filter and kernel2 is a high pass filter
def apply_2_kernels_gray(image, kernel1, kernel2):
    # print("image.shape", image.shape)
    # print("kernel1.shape", kernel1.shape)
    # print("kernel2.shape", kernel2.shape)
    return cv.filter2D(image, -1, kernel1) + image - cv.filter2D(image, -1, kernel2)
```

```
# apply_2_kernels_color(image, kernel1, kernel2) applies two kernels to a color image
# kernel1 and kernel2 are the same size
# kernel1 is a low pass filter and kernel2 is a high pass filter
def apply_2_kernels_color(image, kernel1, kernel2):
    # print("image.shape", image.shape)
    # print("kernel1.shape", kernel1.shape)
    # print("kernel2.shape", kernel2.shape)
    return cv.filter2D(image, -1, kernel1) + image - cv.filter2D(image, -1, kernel2)
```

```
# low_pass(image, sigma) low pass filter
def low_pass(image, sigma):
    return apply_kernel(image, gaussian_kernel(image.shape[0], sigma))
```

```
# high_pass(image, sigma) high pass filter
def high_pass(image, sigma):
    return image - low_pass(image, sigma)
```

```
# hybrid_image(low_image, high_image, cutoff) hybrid image
# low_image and high_image are the same size and same number of channels
# cutoff is the cutoff frequency
# the function takes two images and a cutoff frequency and returns the hybrid image
def hybrid_image(low_image, high_image, cutoff):
    # print("low_image.shape", low_image.shape)
    # print("high_image.shape", high_image.shape)
    # print("cutoff", cutoff)
    low = low_pass(low_image, cutoff)
    high = high_pass(high_image, cutoff)
    # print("low.shape", low.shape)
    # print("high.shape", high.shape)
    return low + high
```

```
# save_hybrid_image(hybrid_image, name) saves the hybrid image
# name is the name of the file to be saved
# the function saves the hybrid image in the current directory
def save_hybrid_image(hybrid_image, name):
    cv.imwrite(name, hybrid_image)
```

```
# show_hybrid_image(hybrid_image) displays the hybrid image
def show_hybrid_image(hybrid_image):
    cv.imshow('hybrid_image', hybrid_image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# read_image_gray(name) reads the image in grayscale
def read_image_gray(name):
    return cv.imread(name, cv.IMREAD_GRAY)
```

```
# read_image_color(name) reads the image in color
def read_image_color(name):
    return cv.imread(name, cv.IMREAD_COLOR)
```

```
# read_image_rgb(name) reads the image in rgb
def read_image_rgb(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2RGB)
```

```
# show_image(image) shows the image
def show_image(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image(image, name) saves the image
def save_image(image, name):
    cv.imwrite(name, image)
```

```
# low_pass_color(image, sigma) low pass filter for color image
def low_pass_color(image, sigma):
    return apply_kernel(image, gaussian_kernel(image.shape[0], sigma))
```

```
# high_pass_color(image, sigma) high pass filter for color image
def high_pass_color(image, sigma):
    return image - low_pass_color(image, sigma)
```

```
# low_pass_gray(image, sigma) low pass filter for grayscale image
def low_pass_gray(image, sigma):
    return apply_kernel(image, gaussian_kernel(image.shape[0], sigma))
```

```
# high_pass_gray(image, sigma) high pass filter for grayscale image
def high_pass_gray(image, sigma):
    return image - low_pass_gray(image, sigma)
```

```
# hybrid_image_gray(low_image, high_image, cutoff) hybrid image
# low_image and high_image are the same size and same number of channels
# cutoff is the cutoff frequency
# the function takes two images and a cutoff frequency and returns the hybrid image
def hybrid_image_gray(low_image, high_image, cutoff):
    # print("low_image.shape", low_image.shape)
    # print("high_image.shape", high_image.shape)
    # print("cutoff", cutoff)
    low = low_pass_gray(low_image, cutoff)
    high = high_pass_gray(high_image, cutoff)
    # print("low.shape", low.shape)
    # print("high.shape", high.shape)
    return low + high
```

```
# save_hybrid_image_gray(hybrid_image, name) saves the hybrid image
# name is the name of the file to be saved
# the function saves the hybrid image in the current directory
def save_hybrid_image_gray(hybrid_image, name):
    cv.imwrite(name, hybrid_image)
```

```
# show_hybrid_image_gray(hybrid_image) displays the hybrid image
def show_hybrid_image_gray(hybrid_image):
    cv.imshow('hybrid_image', hybrid_image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# show_image_gray(image) shows the image
def show_image_gray(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_gray(image, name) saves the image
def save_image_gray(image, name):
    cv.imwrite(name, image)
```

```
# low_pass_color_gray(image, sigma) low pass filter for color image
def low_pass_color_gray(image, sigma):
    return apply_kernel(image, gaussian_kernel(image.shape[0], sigma))
```

```
# high_pass_color_gray(image, sigma) high pass filter for color image
def high_pass_color_gray(image, sigma):
    return image - low_pass_color_gray(image, sigma)
```

```
# low_pass_gray_color(image, sigma) low pass filter for grayscale image
def low_pass_gray_color(image, sigma):
    return apply_kernel(image, gaussian_kernel(image.shape[0], sigma))
```

```
# high_pass_gray_color(image, sigma) high pass filter for grayscale image
def high_pass_gray_color(image, sigma):
    return image - low_pass_gray_color(image, sigma)
```

```
# hybrid_image_color(low_image, high_image, cutoff) hybrid image
# low_image and high_image are the same size and same number of channels
# cutoff is the cutoff frequency
# the function takes two images and a cutoff frequency and returns the hybrid image
def hybrid_image_color(low_image, high_image, cutoff):
    # print("low_image.shape", low_image.shape)
    # print("high_image.shape", high_image.shape)
    # print("cutoff", cutoff)
    low = low_pass_color(low_image, cutoff)
    high = high_pass_color(high_image, cutoff)
    # print("low.shape", low.shape)
    # print("high.shape", high.shape)
    return low + high
```

```
# save_hybrid_image_color(hybrid_image, name) saves the hybrid image
# name is the name of the file to be saved
# the function saves the hybrid image in the current directory
def save_hybrid_image_color(hybrid_image, name):
    cv.imwrite(name, hybrid_image)
```

```
# show_hybrid_image_color(hybrid_image) displays the hybrid image
def show_hybrid_image_color(hybrid_image):
    cv.imshow('hybrid_image', hybrid_image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# show_image_color(image) shows the image
def show_image_color(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_color(image, name) saves the image
def save_image_color(image, name):
    cv.imwrite(name, image)
```

```
# read_image_color_rgb(name) reads the image in color rgb
def read_image_color_rgb(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2RGB)
```

```
# show_image_color_rgb(image) shows the image in color rgb
def show_image_color_rgb(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_color_rgb(image, name) saves the image in color rgb
def save_image_color_rgb(image, name):
    cv.imwrite(name, image)
```

```
# read_image_gray_rgb(name) reads the image in grayscale rgb
def read_image_gray_rgb(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_GRAY), cv.COLOR_GRAY2RGB)
```

```
# show_image_gray_rgb(image) shows the image in grayscale rgb
def show_image_gray_rgb(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_gray_rgb(image, name) saves the image in grayscale rgb
def save_image_gray_rgb(image, name):
    cv.imwrite(name, image)
```

```
# read_image_rgb_gray(name) reads the image in rgb grayscale
def read_image_rgb_gray(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2GRAY)
```

```
# show_image_rgb_gray(image) shows the image in rgb grayscale
def show_image_rgb_gray(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_rgb_gray(image, name) saves the image in rgb grayscale
def save_image_rgb_gray(image, name):
    cv.imwrite(name, image)
```

```
# read_image_hsv(name) reads the image in hsv
def read_image_hsv(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2HSV)
```

```
# show_image_hsv(image) shows the image in hsv
def show_image_hsv(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_hsv(image, name) saves the image in hsv
def save_image_hsv(image, name):
    cv.imwrite(name, image)
```

```
# read_image_hsv_gray(name) reads the image in hsv grayscale
def read_image_hsv_gray(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2HSV)[:,:,2]
```

```
# show_image_hsv_gray(image) shows the image in hsv grayscale
def show_image_hsv_gray(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_hsv_gray(image, name) saves the image in hsv grayscale
def save_image_hsv_gray(image, name):
    cv.imwrite(name, image)
```

```
# read_image_hsv_color(name) reads the image in hsv color
def read_image_hsv_color(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2HSV)
```

```
# show_image_hsv_color(image) shows the image in hsv color
def show_image_hsv_color(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_hsv_color(image, name) saves the image in hsv color
def save_image_hsv_color(image, name):
    cv.imwrite(name, image)
```

```
# read_image_hsv_color_rgb(name) reads the image in hsv color rgb
def read_image_hsv_color_rgb(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2HSV)
```

```
# show_image_hsv_color_rgb(image) shows the image in hsv color rgb
def show_image_hsv_color_rgb(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_hsv_color_rgb(image, name) saves the image in hsv color rgb
def save_image_hsv_color_rgb(image, name):
    cv.imwrite(name, image)
```

```
# read_image_hsv_gray_rgb(name) reads the image in hsv grayscale rgb
def read_image_hsv_gray_rgb(name):
    return cv.cvtColor(cv.imread(name, cv.IMREAD_COLOR), cv.COLOR_BGR2HSV)[:,:,2]
```

```
# show_image_hsv_gray_rgb(image) shows the image in hsv grayscale rgb
def show_image_hsv_gray_rgb(image):
    cv.imshow('image', image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

```
# save_image_hsv_gray_rgb(image, name) saves the image in hsv grayscale rgb
def save_image_hsv_gray_rgb
```

0
50
100
150
200
250
0
50
100
150
200
250

0
50
100
150
200
250
0
50
100
150
200
250