Image and Vision Computing Lab-1

This lab includes three parts. We will use the Python environment throughout. Part one uses different kinds of filters to manipulate the image. In part two, we will write a convolution function to understand the process of convolution. Part three introduces an example of fitting a line to detected edges. The attached files are two example images and the skeleton lab1part3.m. You may wish to read the documentation on Scikit functions mentioned in the appendix of this document. The instructions also include supplementary exercises for keen students. These can be skipped first and tried after the core exercises are complete.

Some basics in case you aren't familiar

https://numpy.org/doc/stable/user/quickstart.html This is the package we will use to read and manipulate images.

Part 1. Warmup with convolution and correlation

1.1 Denoising

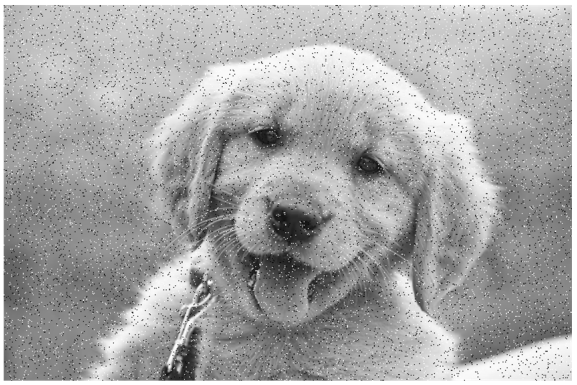
In this section, we will warm up by trying different filters to manipulate the image. You can read an image into Python, and view it using commands like:

```
from PIL import Image
from numpy import *

img = array(Image.open('puppy.png').convert(mode="L")) #to obtain grayscale image from RGB, need this since all operations need grayscale input
Image.fromarray(img).show()
```

We can create a noisy version of this image to work with denoising:
To do this, we use *skimage.util.random_noise(image, mode='gaussian', seed=None, clip=True, **kwargs)* function.

```
img_sp_noise = random_noise(img, mode='s&p', seed=None, clip=True, amount=0.18)
```



Now we can make a 5x5 box filter and convolve it with the image for denoising.

```
def box_filter(dim):
    return np.ones((dim, dim)) * 1 / dim ** 2
img_denoised_box_conv = convolve2d(
    img_sp_noise, box_filter(5), mode="same").astype(np.uint8)
```

You should see a denoised but slightly blurred version of the input image.

Points to Explore:
1. Vary the size of the box filter (3x3, 7x7, etc) and observe the tradeoff between increased denoising effect and increased blurring effect. Remember to normalise the filter so that the values sum to 1.0.
2. Verify that you can achieve the same effects with the correlation rather than convolution.
Why do convolution and correlation give the same result here?
3. Compare the output of convolution with a Gaussian filter and median fliter to the box filter kernel.
Hint: from scipy.ndimage.filters import median_filter, gaussian_filter

1.2 Edge Detection

Define a simple edge filter as simple_edge_filter = [[-1, 0, 1]]. Convolve over the image using this. The output should be as seen below.

Using correlation instead of convolution will result with something like below.

Points to explore:

1.Detect horizontal edges instead.

2.Use Prewitt's kernels to detect edges in both directions. Add them up using this formula and look at the output.

$$\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

The final output should look something like this:

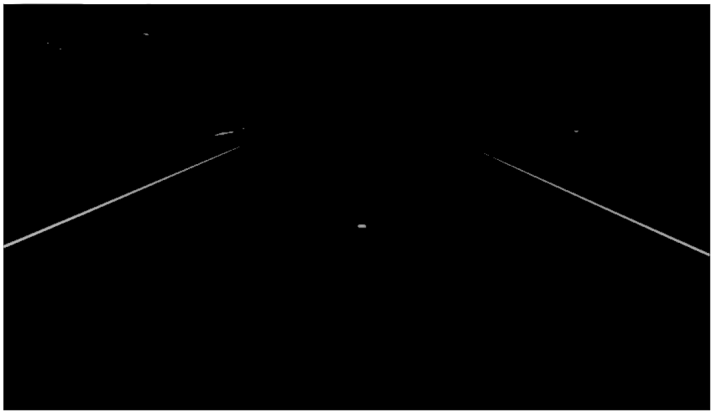
3. Explore the impact of filtering the image first with a blur filter (box or Gaussian), before performing edge detection, as discussed in the lecture notes. You should be able to use this to reduce the "noise" edges, and detect thicker/thinner edges.

4. Bonus [Moderate]: Verify the associativity of convolution numerically. If you convolve the edge filter and blur filter first, then convolve the result with the image; it should achieve the same result as sequentially blurring then edge detecting. Which way is faster?

Part 2: Line Detection

In this section we will detect edges in a road image and use these to fit lines to the image. A

self-driving car may need to do this in order to localise the road for steering.
Open the provided skeleton code line_fitting_template.py. This code performs edge detection, and extracts a list of coordinates of every edge point in the left (edge1) and right (edge2) parts of the image. These will correspond to the two sides of the road.
Implement the simple least squares line-fitting from the lecture notes to fit a line to the left and right sides of the road. You can use the *numpy.polyfit()* function for the same. It should look like this:



Bonus [Advanced]: Implement a RANSAC line detector for robust line fitting.