SYDE 671 Assignment 1

Pascale Walters

Image Filtering and Hybrid Images

September 23, 2019

Q1: Explicitly describe image convolution: the input, the transformation, and the output. Why is it useful for computer vision?

A1: In image processing, convolution refers to a filter being applied to an image. The input is an image (colour or grayscale) and a two-dimensional filter, or kernel, that usually has an odd dimension.

The transformation occurs as the flipped kernel is passed over the image from left to right, then top to bottom. The kernel is multiplied with the image's pixel values and summed to give the output at the centre of the kernel. For an image f and kernel g, the formula for discrete convolution is as follows:

$$(f * g)[m, n] = \sum_{i,j} f[m, n]g[i - m, j - n] \tag{1}$$

The output is a filtered image of the same size as the input image. Convolution is useful for computer vision because it allows for the filtering of images. Depending on the kernel selected, the image can be blurred, sharpened or enhanced.

Q2: What is the difference between convolution and correlation? Construct a scenario which produces a different output between both operations.

A2: The difference between convolution and correlation is that the filter matrix is flipped left-right in convolution. There is no flipping of the filter in correlation. A scenario in which there would be a different output between the two operations would be in the case of an asymmetrical kernel about a vertical axis.

Using an asymmetrical kernel results in different outputs in the following example:

```
import numpy as np
import scipy.ndimage as scimage
from skimage import io, img_as_ubyte, img_as_float32
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

test_image = rgb2gray(img_as_float32(io.imread('data/cat.bmp')))
kernel = np.asarray([[2, 10, 0, 0, 0], [2, 10, 0, 0, 0],
    [3, 10, 0, 0, 0], [4, 10, 0, 0, 0], [6, 10, 0, 0, 0]],
    dtype=np.float32)
kernel /= np.sum(kernel, dtype=np.float32)

convolve = scimage.convolve(test_image, kernel,
    mode='constant', cval=0.0)
correlate = scimage.correlate(test_image, kernel,
    mode='constant', cval=0.0)
```

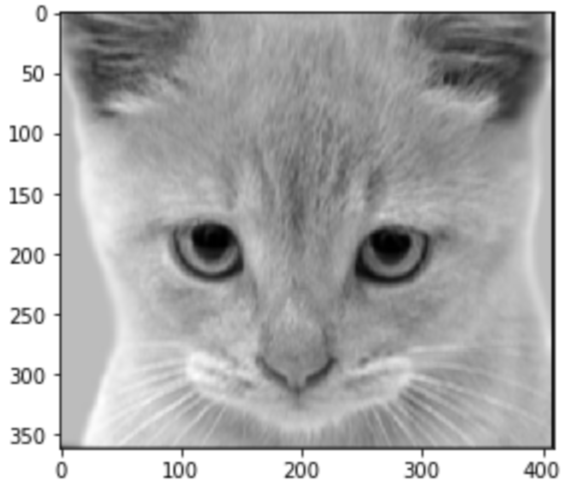

Figure 1: Image with convolution

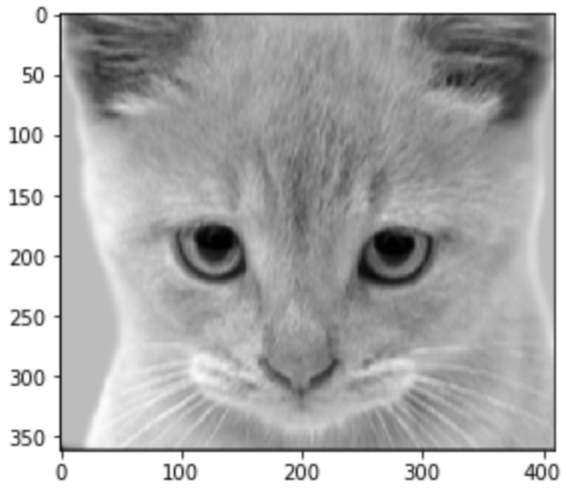

Figure 2: Image with correlation

It can be seen in Figure 1 that black pixels are added on the bottom and right of the image, whereas in Figure 2, they are added along the bottom and left.

Q3: What is the difference between a high pass filter and a low pass filter in how they are constructed, and what they do to the image? Please provide example kernels and output images.

A3: A low pass filter removes high frequency components from the image, whereas a high pass filter removes low frequency components from the image. In a low pass filter, all elements of the kernel sum to 1, which results in local averaging. High pass filters are a low pass filter subtracted from an identity kernel. All elements in the kernel sum to 0.

Convolution with the low pass filter kernel in Equation 2 results in the blurred image in Figure 3.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

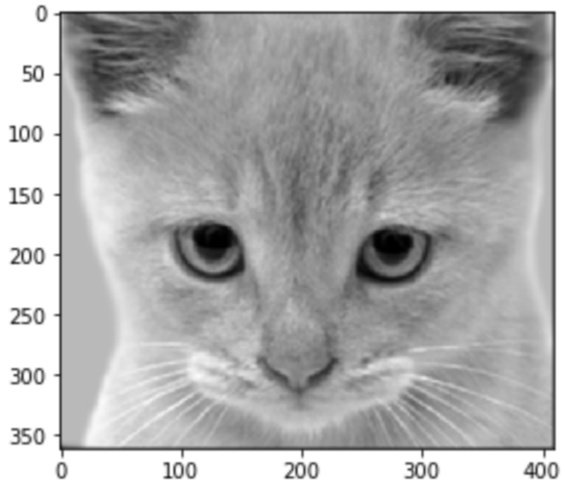

Figure 3: Image with low pass filter

Convolution with the high pass filter kernel in Equation 3 results in the blurred image in Figure 4.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3}$$

Q4: How does computation time vary with filter sizes from 3×3 to 15×15 (for all odd and square sizes), and with image sizes from 0.25 MPix to 8 MPix (choose your own intervals)? Do the results match your expectation given the number of multiply and add operations in convolution?

A4: There seems to be a linear relationship between the size of the filter, the size of the image and the time to perform convolution (Figure 5). This is unexpected, as one would think that there would be a quadratic relationship due to the number of multiply and add operations increasing with increasing with filter and image size. However, there likely is some optimization that allows it to perform faster. Further speedup would be possible by using *scipy.signal.fftconvolve*.

The code used to generate the plot in Figure 5 is as follows:

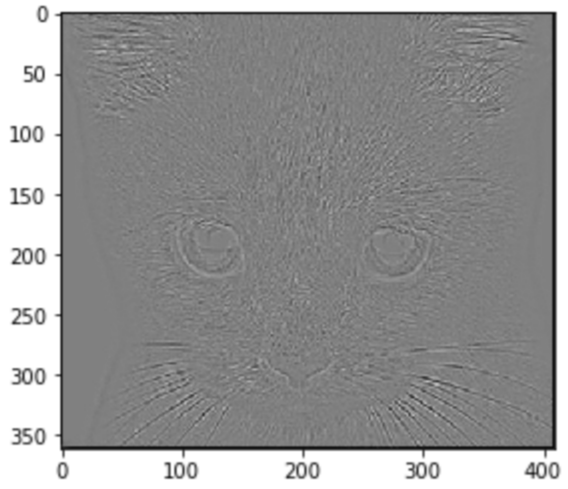

Figure 4: Image with high pass filter

```
im_h, im_w = test_image.shape
scale = im_h / float(im_w)

im_size = []
filter_size = []
conv_time = []
corr_time = []

for i in range(3, 16):
    if i % 2 == 1:
        lpf = np.ones((i, i))
        lpf /= np.sum(lpf, dtype=np.float32)
        for j in [2100, 1850, 1600, 1350, 1100, 850, 600, 350]:
            im = resize(test_image, (int(j), int(j / scale)))
            pixels = int(j * j / scale) / 1000000.0

            start_time = time.time()
            scimage.convolve(test_image, lpf, mode='constant', cval=0.0)
            time1 = time.time() - start_time

            start_time = time.time()
            scimage.correlate(test_image, lpf, mode='constant', cval=0.0)
            time2 = time.time() - start_time

            conv_time.append(time1)
            corr_time.append(time2)
            im_size.append(pixels)
            filter_size.append(i * i)
```

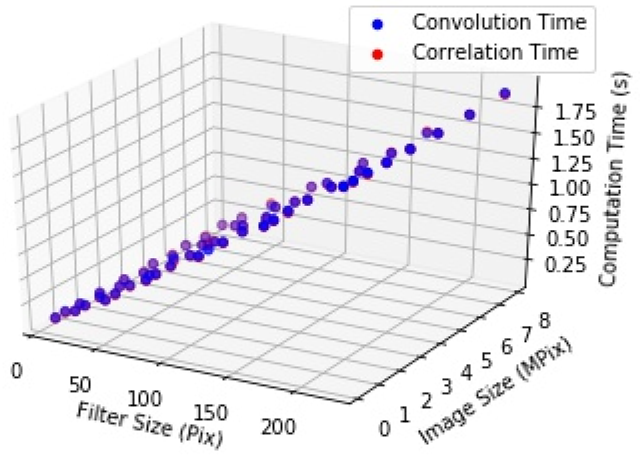

Figure 5: Time to perform convolution or correlation