Project 1 Writeup

Instructions

- Provide an overview about how your project functions.
- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- List any extra credit implementation and result (optional).
- Use as many pages as you need, but err on the short side.
- **Please make this document anonymous.**

Project Overview

This project aims to implement two functions `myfilter()` and `gen_hybrid_image()`. Different types of filters are applied to images to test `myfilter()` function. As for `gen_hybrid_image()`, besides testing with extra pairs of images, the effect of image assignment sequence to image1 and image2 is also investigated.

Implementation Detail

Here is the implemetation for `myfilter()`. At first, shapes of the kernel and image are obtained and an error message is raised if the filter has an even-dimension. In order to support both grayscale and RGB pictures, the grayscale picture is reshaped to a 3d array, so that we don't need to worry about the dimensions difference between grayscale and RGB picture. Then, the image is padded with reflect mode on the hight and width dimension using `np.pad()`. The kernel is flipped using `np.flit()` as we want to realize convolution operation. After all above manipulations, the output is obtained by taken dot product between `flipped_kernel` and `padded_image`.

I have tried two ways to obtain the final output. (Both are included in code block below) The first implementation is an ordinary way: using two for loop over the height and width dimesion of image and use `np.tensordot()` to obtain the convolution output. The second implementation is using `np.lib.stride_tricks.as_strided()` and `np.einsum()` to avoid using for loop, which can shorten the excutation time from 2.6s to 0.08s, about 30 times faster.

At last, if the input image is grayscale, the output is reshaped back to 2D to keep the result the same dimension as the input.

```
(k, l) = kernel.shape
(m, n, c) = image.shape
if (k * l) % 2 == 0:
    raise Exception("Output with even filters are not defined!")

Grayscale = False
if len(image.shape) == 2:
    Grayscale = True
    image = np.reshape(image, (image.shape[0], image.shape[1], 1))

padded_image = np.pad(image, ((k // 2, k // 2), (l // 2, l // 2), (0,
                                    0)), "reflect")
# because we want to calculate convolution, we need to flip the kernel
flipped_kernel = np.flip(kernel)

# First Implementation
output = np.zeros(image.shape)
for i in range(m):
    for j in range(n):
        output[i, j] = np.tensordot(flipped_kernel, padded_image[i : i
                                                + k, j : j + l], axes=[(0,
                                                1), (0, 1)])

# Second Implementation
expanded_image = np.lib.stride_tricks.as_strided(
        padded_image,
        shape=(
            m,
            n,
            c,
            k,
            l,
        ),
        strides=(
            padded_image.strides[0],
            padded_image.strides[1],
            padded_image.strides[2],
            padded_image.strides[0],
            padded_image.strides[1],
        ),
        writeable=False,
    )

output = np.einsum("xyzij, ij->xyz", expanded_image, flipped_kernel)

if Grayscale:
    output = output.reshape(output, (m, n))
filtered_image = output
```

Result

1. Different filters are tested as implemented in file `proj1_part1.py`. The results for the `dog.bmp` are shown in Fig. 1 and all of them meet the expectation for the corresponding filters.

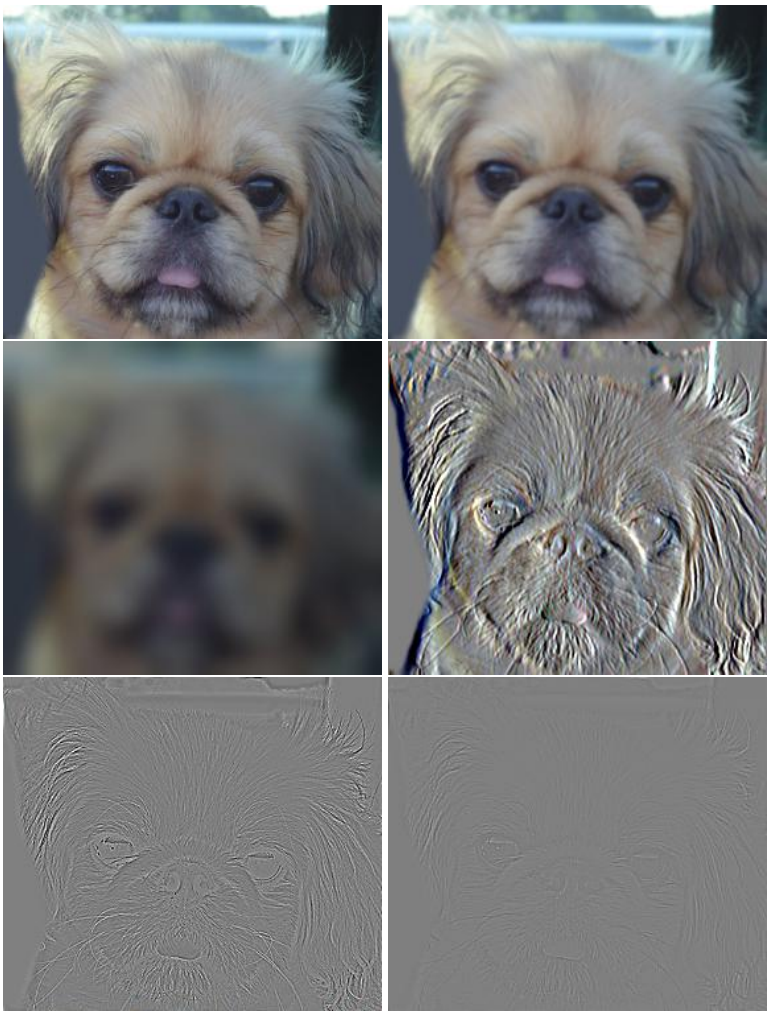

Figure 1: *Left:* (from top to bottom)identity filter, large blur filter, laplacian fliter. *Right:* (from top to bottom) blur fliter, sobel fliter, high pass fliter.

2. For the hybrid image generation, it will differ depending on which image is assign as image1 (which will provide the low frequencies) and which image is asign as image2 (which will provide the high frequencies). One simple example is presented below.

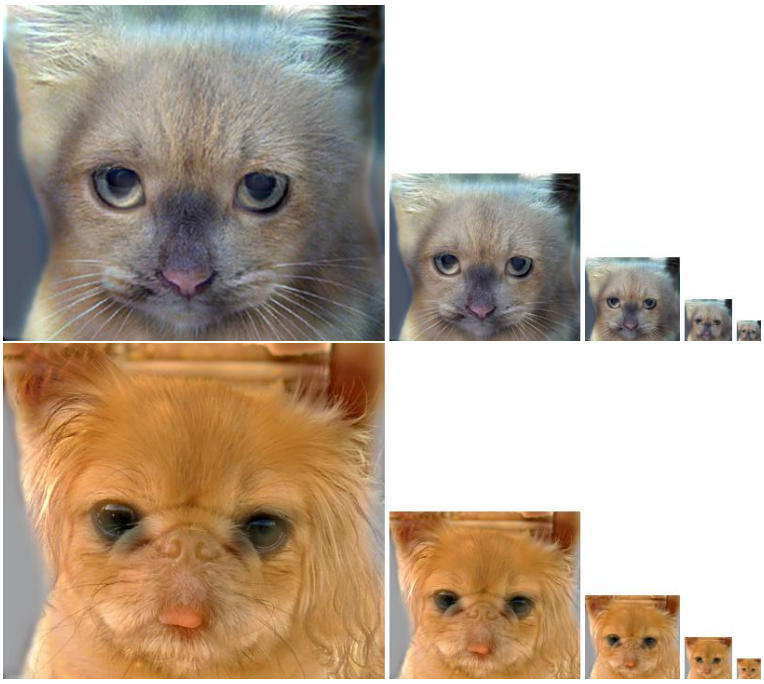

Figure 2: *Top:* image1 is dog.bmp and image2 is cat.bmp *Bottom:* image1 is cat.bmp and image2 is dog.bmp.

Extra Credit (Optional)

1. Pad with reflected image content.

```
padded_image = np.pad(image, ((k // 2, k // 2), (l // 2, l // 2),
                              (0, 0)), "reflect")
```

(Please check next page for 2 and 3)

2. Create own hybrid image. Here is my own hybrid image result shown below.

Figure 3: Own hybrid image: created from Putin.jpg and Trump.jpg

3. Here is the code for my FFT-based convolution. I use `scipy.fftpack()` for the FFT transformation. It can produce similar images as presented in Fig. 1. However, I don't know why it hasn't passed the autograder.

```
(k, l) = kernel.shape
if (k * l) % 2 == 0:
    raise Exception("Output with even filters are not defined!")

Grayscale = False
if len(image.shape) == 2:
    Grayscale = True
    image = np.reshape(image, (image.shape[0], image.shape[1], 1)
                                                        )
(m, n, c) = image.shape
# because we want to calculate convolution, we need to flip the
                                            kernel
flipped_kernel = np.flip(kernel)
(k, l) = flipped_kernel.shape
padded_image = np.pad(
    image, ((k // 2, k // 2), (l // 2, l // 2), (0, 0)), "reflect
                                                        ",
```

```
)

fft_kernel = fftpack.fft2(flipped_kernel, shape=padded_image.
                                    shape[:2], axes=(0, 1))

fft_image = fftpack.fft2(padded_image, axes=(0, 1))
fft_output = fft_kernel[:, :, np.newaxis] * fft_image
output = fftpack.ifft2(fft_output, axes=(0, 1)).real
output = np.clip(output[k // 2 : -(k // 2), l // 2 : -(l // 2)],
                                    0, 1)
if Grayscale:
    output = np.reshape(output, (m, n))
filtered_image = output
```