Report for Image Filtering and Hybrid Images

Feng Xie 30502322

1.Algorithm description

To create a hybrid image as described in the coursework, firstly we need to implement a function to generate a 2D gaussian convolution kernel and after that we need to implement a function for template convolution to apply the gaussian template to the image.

The programming language I used for this coursework is Python with the version of 3.6. Additionally, I used OpenCV to do some basic image operations.

1.1 Gaussian Template

Here is the code snippet which shows how I implemented a function to generate a gaussian template:

```
def gaussian_template(self):
        # compute the window size according to the method specified in the coursework detail
        winsize = int(8.0 * self.sigma + 1.0)
        if winsize % 2 == 0:
            winsize += 1   # the window size must be odd
        # initialize a 2-D array for the template and set the values to zeros
        template = np.zeros((winsize,winsize))
        # compute the centre of the template
        centre = math.floor(winsize/2) + 1
        # create a gaussian template
        sum = 0
        for i in range(1, winsize + 1):
            for j in range(1, winsize + 1):
                template[j - 1][i - 1] = math.exp(-((i-centre) * (i-centre) + (j-centre) * (j-centre))/(2 * self.sigma
* self.sigma))
                sum += template[j - 1][i - 1]
        return template/sum  # normalise by the total sum
```

Firstly, following the formula specified in the coursework details, I computed the window size based on the given sigma value. Here in my code the sigma will be specified at the initiation of the class. And the window size must be odd. Then I initialize a zero-value 2-D array using Numpy for setting the values in the template.

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

According to the Gaussian function g specified as above, I created a double loop to compute coefficients for a Gaussian template. Because I need to divide the template by the sum of its values at the end, the coefficient of 1/(2 * pi * sigma * sigma) in the equation above can be ignored. I started the index of 1 instead of 0 when doing the loop to help comprehension. The function above will output a gaussian template which will be convolved with an image later.

1.2 Template Convolution

Next, we need to implement a function for template convolution. The code snippet is shown as below:

```
def template_convolution(self, image, template):
        '''
        the function for template convolution

        :param array image: a 2-D matrix of an image
        :param array template: a 2-D matrix of a template to be convolved with the image

        '''
        # get the shapes of the image and the template
```

```
        rows, cols = image.shape
        trows, tcols = template.shape
        # filp the template
        flipped_template = np.zeros((trows, tcols))
        for i in range(trows):
            for j in range(tcols):
                flipped_template[i][j] = template[trows - i - 1][tcols - j - 1]
        # get the half of the rows and columns of the template
        tr_half = floor(trows / 2)
        tc_half = floor(tcols / 2)
        # initialize the resulting image to black(zero brightness levels)
        output_image = np.zeros(image.shape)
        # convolve the template
        for x in range(tc_half, cols - tc_half):  # address all columns except border
            for y in range(tr_half, rows - tr_half): # address all rows except border
                # get the area of the image which corresponds to the location of the template
                pixel_set = image[y - tr_half:y + tr_half + 1, x - tc_half: x + tc_half + 1]
                # get the resulting value of the point by multiplying two matrices element-wise and getting the
sum.
                output_image[y][x] = floor(np.multiply(flipped_template, pixel_set).sum())
        return output_image
```

Firstly, I got the dimensions of the image and the template separately. Because the template should be flipped either before or during convolution, here I choose to flip the template beforehand. Next I initialized a 2-D array filled with zero as the output image. By creating a zero-value array, I set the border pixels to black as well. Secondly, by using a double loop specified above, the function will compute the new values of the image point by point except for the border. During each pass of the loop, the new value of the point will be computed by multiplying two matrices element-wise instead of using one more double loop.

In practice, initially I implemented the convolution with four loops, and it turned out that the computation time increased fast as the window size became larger. My program even got stuck when the window is too large. It may be caused by the slow numeric calculation in Python. Later I used the matrix operation implemented in Numpy and it led to a boost in performance because many Numpy operations are implemented in C language which is good at calculations.

1.3 Hybrid Images

In order to apply the convolution to color images, I used OpenCV to split the color image into three single color channels and apply the convolution operator to them separately. Here I create three processes to speed up the program so that it can handle three times of convolution in the same time. After getting a low-pass version of the image, I achieved the high-pass filtering by subtracting the low-pass version of the image from itself. And by combining a low-pass version of an image and a high-pass version of another one, I got a hybrid image.

The code snippet demonstrating how I created a low-pass version of an image and got a hybrid image is shown below:

```
    def low_pass(self, image):
        '''
        create the low-pass version of an image

        :param array image: a 2-D matrix of an image

        '''
        # create a gaussian template
        template = self.gaussian_template()
        # split the image into 3 separate color channels
        b, g, r = cv2.split(image)
```

```
        # create a pool with 3 processes
        pool = Pool(processes=3)
        # assign a task of template convolution to an independent process
        # and run the processes in an asynchronous way
        multi_res = [pool.apply_async(self.template_convolution, (i, template)) for i in (b, g, r)]
        # get the result from the processes
        res = [i.get() for i in multi_res]
        # merge three color channels into a new image
        new_image = np.uint8(cv2.merge(res))
        return new_image

    def high_pass(self, low_pass, original):
        '''
        create the high-pass version of an image

        :param array low_pass: a low-pass version of an image
        :param array original: the original image

        '''
        # get the high-pass version of an image
        # by subtracting the low-pass version from the image itself
        return original - low_pass

    def main(self):
        '''
        create a hybrid image
        '''
        # mark the start time of the operation
        start = time.time()
        # get the low-pass versions of the two input images
        low_pass1 = self.low_pass(self.image1)
        low_pass2 = self.low_pass(self.image2)
        # get the high-pass version of the first input image
        high_pass1 = self.high_pass(low_pass1, self.image1)
        # create a hybrid image
        # using the high-pass version of the first input image
        # and the low-pass version of the second input image
        hybrid_img = high_pass1 + low_pass2
        # output the execution time of the operations above
        print('Execution time: %.2f s' % (time.time() - start))
```

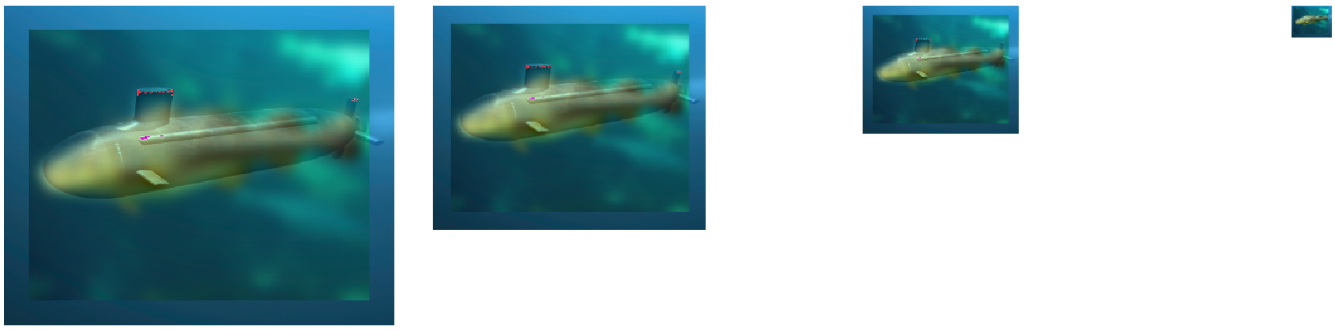
2. The result of the algorithm

2.1 Hybrid images

The hybrid of cat.bmp and dog.bmp(sigma: 4; window size: 33 * 33):

The hybrid of bird.bmp and plane.bmp(sigma: 4; window size: 33 * 33):

The hybrid of fish.bmp and submarine.bmp(sigma: 6; window size: 49 * 49):



The hybrid of bicycle.bmp and motorcycle.bmp(sigma: 4; window size: 33 * 33):

The hybrid of Einstein.bmp and Marilyn.bmp(sigma: 2; window size: 17 * 17):

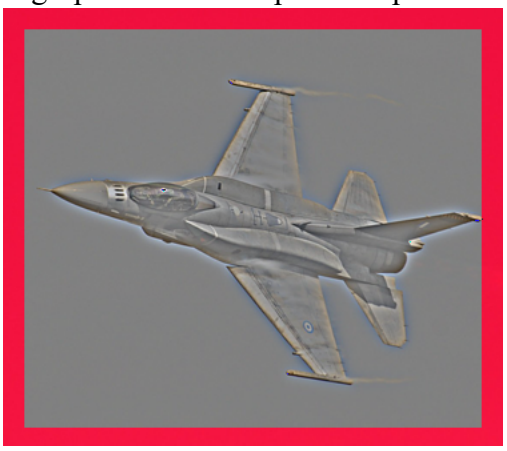



I tried applying different sigma values to the pairs of images and changing the input order of high-pass images and low-pass images and I found that the window size and the input order really matter to the hybrid result. It turned out that the combination of cat.bmp with dog.bmp and Einstein.bmp with Marilyn.bmp got the better result than the others. I think it probably depends on the input images themselves. In terms of the performance of the program, it usually takes less than 4 seconds to generate a hybrid image when the sigma value is less than ten.

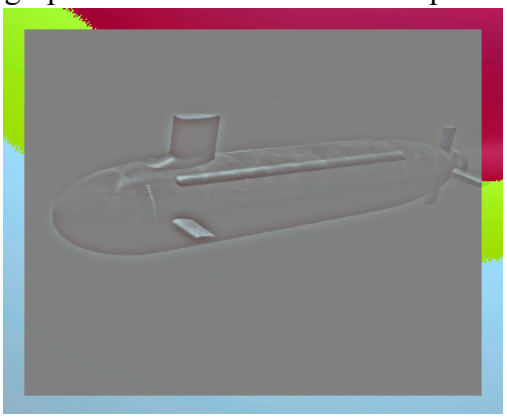
2.2 Intermediate images

Some of intermediate images are shown below.

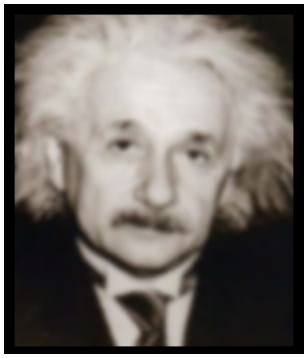
The low-pass version of bird.bmp and the high-pass version of plane.bmp:

The low-pass version of fish.bmp and high-pass version of submarine.bmp:

The low-pass version of Einstein.bmp and high-pass version of Marilyn.bmp:




3. References

[1] Nixon M, Aguado A S. Feature extraction and image processing for computer vision[M]. Academic Press, 2012.

[2] Oliva, Aude, Antonio Torralba, and Philippe G. Schyns. "Hybrid images." *ACM Transactions on Graphics (TOG)*. Vol. 25. No. 3. ACM, 2006.

[3] http://comp3204.ecs.soton.ac.uk/cw/c6223_coursework1.html