COMP6223 Computer Vision
Coursework 1: Image Filtering and Hybrid Images

Yuyang Xue

[1]Student ID: 29084016

yx2n17@soton.ac.uk

Abstract. *This coursework includes two parts of exercises: one is template convolution, which is using a small template kernel to move all over the original picture and get the new picture by calculating the convolution of its original pixels and kernel weightings. The other is Hybrid image. By using Gaussian filter to get one image's low frequent output and the other image's high frequent output, it can be added as a hybrid image. Use downsampling to show 5 scales of the original hybrid image to see the difference between distances human eyes can see.*

Keywords*: template convolution, hybrid image, Gaussian filter*

1. Template Convolution

Template convolution is a group operation to calculate new image's pixel values from the corresponding neighbour pixels of the original image according to the weighting coefficient in template kernel. In this coursework, a two dimensional matrix with odd rows and columns, such as 7×9 and 3×5. Pixel values are multiplied by the corresponding weighting coefficient and added to an overall sum.

This exercise should consider the colour channels of grey images and RBG images. First, load the image into the program. The image is supposed to convert into single number, rescaling the data if necessary, in case some operation makes pixel value larger than 255. Then we can get the height and width from both the original image and template. The example from Mark's book [Nixon and Aguado 2008] is using template to move throughout the whole original image. According to commutative property of convolution, it means $Img_O * T = T * Img_O$. It turns out that the same result can be derived by moving the original image through out the template itself. By using *Matlab*, a programming language can handle matrix conveniently, each block matrix can be calculated by only $height_T \times width_T \times Channels$ times. The result can be stored in a slightly larger all zero matrix, which is a black picture. After convolution process, just clip the true convoluted area of the image and put it in a black edge image to get what we want.

The convolution example and code is shown below:

The original image

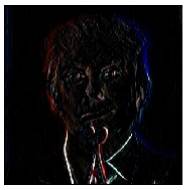

Convoluted by Sobel

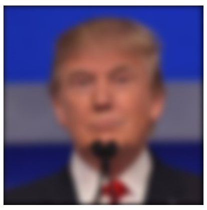
Figure 1. Convolute the original image by Sobel

Listing 1. Convolution Code Snippet

```
%function testimage = template_convolution(img, template, flag)
for x = 1 : tcols
    for y = 1 : trows
        %for each colour channel do a convolution operation
        for c = 1 : channels
            tcimage(y : end - (trows - y),x : end - (tcols - x),
                c) = tcimage(y : end - (trows - y), x : end - (
                tcols - x), c) + image(:, :, c) * template(y, x);
        end
    end
end
```

1.1. Gaussian Filter

Now a Gaussian filter should be implemented. A 2D Gaussian convolution kernel of $Size_T \times Size_T$ pixels. The $Size_T$ is width and height of the template in pixels. The Gaussian filter removes all the high frequencies and produce a low pass image. In the mean time, using the original image to subtract the low pass image can get the high pass image. By using Gaussian averaging operator $\frac{1}{2\pi\sigma^2}\exp^{-(\frac{x^2+y^2}{2\sigma^2})}$ [Wikipedia 2017], the desired convolution template can be derived. The size of the Gaussian filter is defined according to this coursework's website as $size = 8 \times \sigma + 1$ [Prof. Mark Nixon 2017]. Two σ value can input arbitrarily. The high frequency image should add 0.5 to every pixel in each colour channel in order to see it clearly. Example of low pass image ,high pass image and code snippet are shown below.


Low pass image

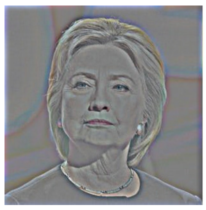

high pass image

Figure 2. Convolute the original image by Gaussian Filter

Listing 2. Gaussian Filter Code Snippet

```
%function I = Gaussianblur(img, sigma, flag)
temp = zeros(siz, siz);
sum = 0;
for i = f1
    for j = f1
        temp((siz-1)/2+i+1,(siz-1)/2+j+1)= 1/(2*pi*sigma^2)* exp
            (-1*(i^2+j^2)/(2*sigma^2));
        sum = sum + temp((siz-1)/2+i+1,(siz-1)/2+j+1);
    end
end
temp = temp ./ sum;
```

Also, using **Fast Fourier Transform** to generate a Gaussian Filter is feasible. But the template should be exactly the same size of the image, the only controllable parameter is σ. Example of implemented *FFT* to get low pass image and *FFT* code snippet are shown below.

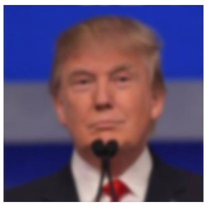


Figure 3. Using FFT with $\sigma = 5$

Listing 3. Gaussian Filter Code Snippet

```
%function I = fftblur(img, sigma, flag)
f1=-fix(h/2):ceil(h/2)-1;
f2=-fix(w/2):ceil(w/2)-1;
[fx,fy]=meshgrid(f1,f2);
%Fourier Transform of Binary Gaussian Distribution
X=exp(-(1/2*pi*pi*sigma*sigma)*((fx/h).^2+(fy/w).^2));
image = real(ifft2(ifftshift(fftshift(fft2(img)) .* X')));
```

2. Hybrid Image

According to the paper of *Hybrid Image* [Oliva et al. 2006], adding a low pass image and a high pass image can combine a whole new hybrid image. In different distances or different scales of image, people can see two different image. The closer you look or the bigger scale the hybrid image is, the more it looks like the low pass image, and vice versa. Two parameters, σ_1 and σ_2 can be modified easily to see in which value can get the best hybrid image. Hybrid example and code snippet are shown below.

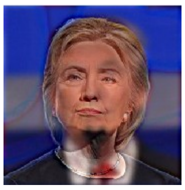


Figure 4. Hybrid image when both σ are 5

Listing 4. Hybrid Image Code Snippet

```
%function hyimage = hybrid_image(image1, image2, hc_frequency,
    lc_frequency)
low_frequencies = Gaussianblur(image1, lc_frequency, 0);
high_frequencies = Gaussianblur(image2, hc_frequency, 1);
hyimage = low_frequencies + high_frequencies;
```

2.1. Downsampling

Using downsampling can see more clearly in different scale of the hybrid image. Setting a divider and place the scaled image together to get a bigger output.
Example of downsampling image and code snippet are shown below.

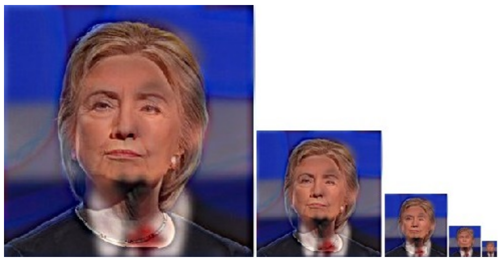


Figure 5. Hybrid image downsampling

Listing 5. Hybrid Image Code Snippet

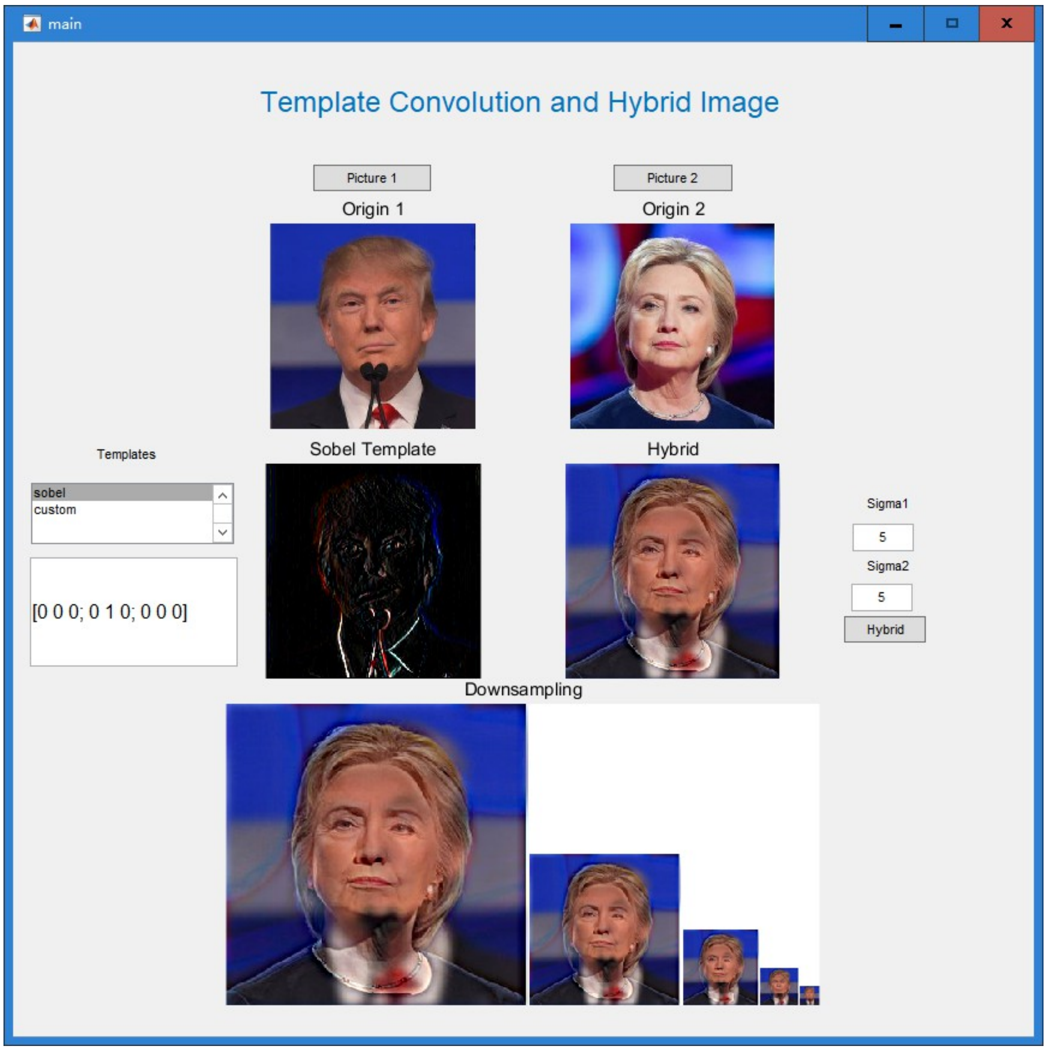
```
%function output = downsampling_hybrid_image(hybrid_image)
for i = 1 : numbers - 1
    output = cat(2, output, ones(height, gap, channels));
    cur_image = imresize(cur_image, devided_by);
    tmp = cat(1, ones(height - size(cur_image, 1), size(cur_image
        , 2), channels), cur_image);
    output = cat(2, output, tmp);
end
```

3. GUI

In order to show the whole coursework in a more intuitive way, a graphic user interface is made to illustrate my coursework more easily. It can choose images by the users themselves, but both images should be equal size. Running *main.m*, users can get a gui window of this coursework. The section of template convolution can use default Sobel operator, or user custom matrix by editing the text box below options. Also in hybrid image section, users can modify both σ conveniently and press *hybrid* bottom to get the hybrid image and its downsampling. But always remember to press **Enter** when modify any parameters to let the program know what you are entered.

4. Acknowledgement

Discussing Gaussian filter and FFT method with my classmates Nan Zhao, and Yipeng Shen. Thanks for their signal processing knowledge about helping me work out the Gaussian filter.



[0 0 0; 0 1 0; 0 0 0]

Sigma1: 5

Sigma2: 5

Figure 6. GUI for coursework 1

References

Nixon, M. and Aguado, A. S. (2008). *Feature Extraction & Image Processing, Second Edition*. Academic Press, 2nd edition.

Oliva, A., Torralba, A., and Schyns, P. G. (2006). Hybrid images. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 527–532, New York, NY, USA. ACM.

Prof. Mark Nixon, D. J. H. (2017). Comp6223 computer vision (msc) coursework 1. [Online; accessed 7-November-2017].

Wikipedia (2017). Gaussian filter - wikipedia. [Online; accessed 4-November-2017].