Computer Vision Coursework One

Zhikun Zhu; ID: 29356822

November 15, 2017

I Gaussian Kernels

In this section, algorithm which generate arbitrary shaped Gaussian kernels with both dimensions odd will be introduced.

Probability density function (pdf) of Bivariate Gaussian Distribution is given by:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^2|C|}} \exp\{-\frac{(\mathbf{x}-\mu)C^{-1}(\mathbf{x}-\mu)^T}{2}\} \quad (1)$$

Here we choose $C = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$ and $\mu = [0 \quad 0]$. Let $\mathbf{x} = [x_1 \quad x_2]$, we can simplify Eq.1 into:

$$p(x_1, x_2) = \frac{1}{2\pi\sigma^2} \exp\{-\frac{x_1^2 + x_2^2}{2\sigma^2}\} \quad (2)$$

Therefore, we can use such expression to generate random sized Gaussian kernels. Besides, it is needed to normalize the template that the sum of its elements is 1. The algorithm is shown in *gaussianKernel.m*. Following matrix is an example result that use this algorithm to generate a 5-by-3 Gaussian template.

0.0149	0.0246	0.0149
0.0669	0.1103	0.0669
0.1103	0.1819	0.1103
0.0669	0.1103	0.0669
0.0149	0.0246	0.0149

Table 1: 5-by-3 Gaussian Kernel ($\sigma^2 = 1$).

Above Gaussian kernel is normalized to zero, and $\sigma^2 = 1$. Increasing $sigma^2$ will flat the template, which means the template is getting close to averaging template.

II Template Convolution

In this section, template convolution as well as its MATLAB realization will be introduced. Template convolution is part of group operations, which use a group of pixels from original image to generate new pixel to form new image. For an M-by-N template ($M, N$are both odd), its convolution with old image can be calculated by:

$$N_{x,y} = \sum_{j=\frac{1-N}{2}}^{\frac{N-1}{2}} \sum_{i=\frac{1-M}{2}}^{\frac{M-1}{2}} O_{x+i,y+j} \times \omega_{x+i,y+j} \quad (3)$$

It is quite clear that the template need to be normalized to adjust the new pixel's value between 0 and 255. Besides, when we convolve the border of old image with template, there always have half of the template located outside of the image. Therefore, it is required to pad the original image with zeros, which length and width is decided by the size of template. Fig.1 shows the image after pad. Then, the original part of the paded image is used to convolve with the template.

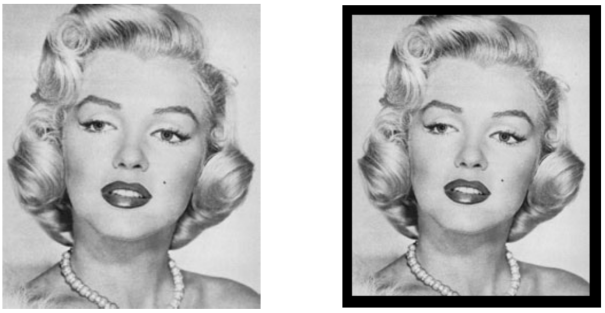

Figure 1: Compare paded image (Template size:21×21) with original image

```
function nImg = tempConv(image,te)
    %================= Templte Convolution Algorithm ==============%
    %======================== Auther: Zhikun Zhu ==================%
    %======================== Date: 10/11/2017 ====================%
    % Function: Calculate the template convolution of input image.
    % Support both gray and colour images.
    % Note: The number of the rows and columns of the template must
    % be odd.

    % Change data type for the convenience of following calculations.
    img = im2double(image);
    % State the size of new image(Same with old image).
    nImg = zeros(size(img));
    % Get template size.
    [teRow,teCol] = size(te);
    teRhalf = (teRow-1)/2;
    teChalf = (teCol-1)/2;
    % Get size after expand.
    [imRow,imCol,n] = size(img);

    %Calculate templete convolution for RGB, respectively. If the ...
       input image is mono, then n = 1.
    for i = 1:n
        % Pad image to suit convolution size.
        imPad = padarray(img(:,:,i),[teRhalf teChalf],'both');
        for x = (teChalf+1):(imCol+teChalf)
            for y = (teRhalf+1):(imRow+teRhalf)
                % Get the spectific area of image to convolve with ...
                   template
                partImg = ...
                    imPad((y-teRhalf):(y+teRhalf),(x-teChalf):(x+teChalf));
                % Matrixes dot product.
                temp = te.*partImg;
                % Sum all elements in temp to get new pixel's value.
                nPixel = sum(temp(:));
                % Save the new pixel in new image.
                nImg((y-teRhalf),(x-teChalf),i) = nPixel;
            end
        end
    end
end
```

The code above is the MATLAB code to complete template convolution, which can also found in the *tempConv.m*.
Firstly, for the convenience of futhre processing, the input image's date type is changed to double by *im2double(·)*, which function will also change the color space from [0 255] to [0 1]. Then, the *for* loop is used to processing RGB image, which use three matrixes to save red, green, blue colour respectively. Therefore, the algorithm can handle colour images as well as mono images by the different of their size.
Then, for each colour, the new pixel will be calculated by summing the dot product between template and part of old image with same size and centred at old pixel.
There is another problem need to solve. In the *for* loop, we only need to convolove the unpaded area, which means the counters for row and column need to avoid the half size of template. Besides, the indexes of new pixels should start at zero, so the couters need to be shifted to fit the indexes. The realization is shown in the code.

III Hybrid Images

In this section, two images will hybrid into a new image utilizing the method illustrated in [1], which hybrid two images' low frequency component and high frequency conponent together. Choosing different

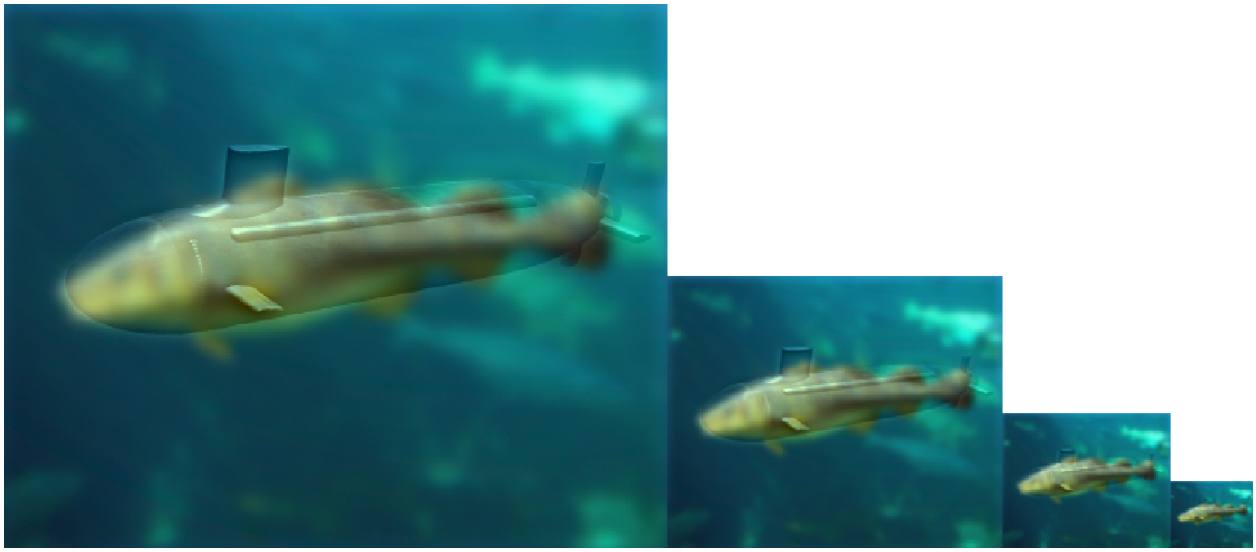

Figure 2: Hybrid plane (high freq.) and brid (low freq.)

value of σ and template size will shift the cut off frequency of Gaussian low pass filter. High frequency conponent can be acquired by subtract the low pass frequency component from original image. Fig.2 shows an example of such algorithm.

Fig.2 is an imitation of the scenario that viewing the image from proximal to distal, which is quite clear that the high frequency attenuation is getting severe with the increasing distance. The path loss of electromagnetic wave in free space is proportional to $\frac{\lambda^2}{d^2}$, where λ is wavelength, d is propagation distance.

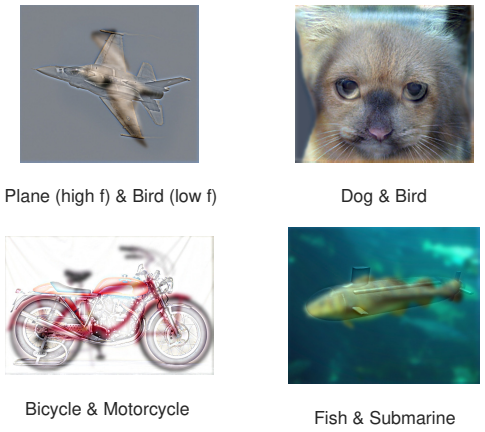

Figure 3: Hybrid images

There would be a jet, cat, motorcycle and submarine in Fig.3 if you take the image up close to your eyes, whereas, if you put it far away, you will only see bird, dog, bicycle and fish, respectively. The only reason of

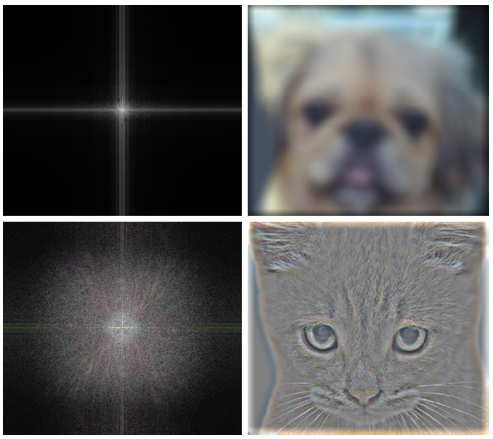

Figure 4: Upper: Low path filtered dog (Right) and its amplitude spectrum (Left); Lower: High path filtered cat (Right) and its amplitude spectrum.

such phenomenon is that the high frequency conponents decayed rapidly as distance increase. Fig.4 shows the amplitude spectrums of the dog and cat image after filtered by using *tempConv.m* and the original spectrums. The figure's colour is normalized to fit all gray level and it is defined by [2]:

$$N_{x,y} = \frac{N_{max} - N_{min}}{O_{max} - O_{min}} \times (O_{x,y} - O_{min}) + N_{min}$$

Since the image's pixel is normalized to $[0, 1]$ colour space, so $N_{max} = 1$ and $N_{min} = 0$.

Then above equation can be simplied into Eq.4:

$$N_{x,y} = \frac{O_{x,y} - O_{min}}{O_{max} - O_{min}} \quad (4)$$

The degree of alignment will also affect the perception of human vision [1]. In Fig.4, two images' frequency conponents are jointed together by three different methods. The original hybrid image is shown in left lower. In the upper left area, the edges are flipped, which is clear that the blobs (low frequency conponents) aren't shaded by the edges. Futhermore, at the right side, the blobs is stretched to fit the size of the edges. Then the edges and blobs can mask each other in the up close and in distance, respectively.

Figure 5: Effects of alignment

According to previous sections, the high frequency conponents of an image are visiable only in a short distance. Then, hybrid images is useful in protecting privacy[1]. Fig.6 is a application of such hybrid methods. The characters in the upper of the figure is filtered with a high path filter and the texture is filtered by a low path filter, then, the characters can only been seen in a very close distance, which can used to protect important message like PIN, birthday, etc.

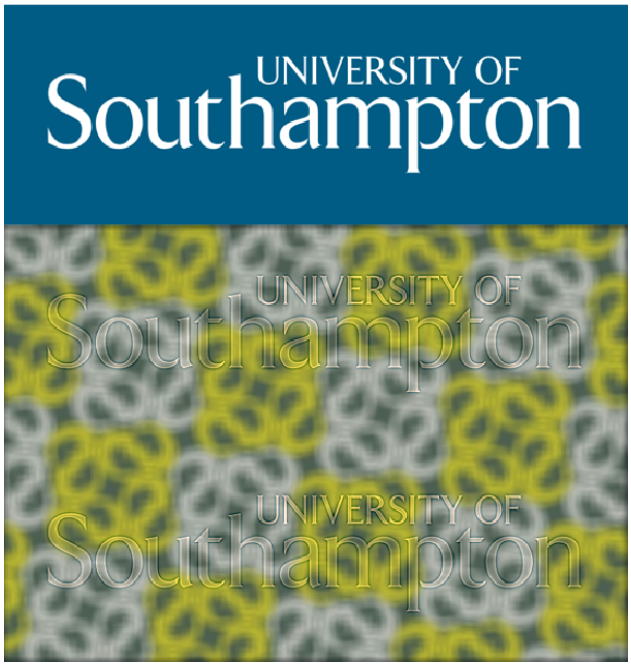

Figure 6: Shade high frequency characters by low frequency layer.

IV Conclusion

The Gaussian template realization (*gaussianKernel.m*) can support random input number, that means, you can only use σ to generate a square template which size is depend on σ or you can use $[\sigma \quad row]$ to generate a template which size is $row \cdot row$, or use $[\sigma \quad row \quad col]$ to generate a template with size $row \cdot col$.

The algorithm used to bybrid images (*hybridImg.m*) can support random sized input images, which will pad zeros to the small size image if two images' size are not match. It will also transform the colour iamge into mono if one of two images is mono.

The *tempConv.m*, which used to perform template convolution, will pad zeros around the input images before the convolution and remove the paded zeros after convolution. So the size of output iamge should equals to input image.

During the period of writing algorithms, I learned skills to process image with MATLAB as well as get familier with various kinds of functions like *padarray(·)*, *imresize(·)*, *cat(·)*, etc. The concept of template convolution and group operations is also acquired.

Besides, I obtained experiences of writting reports with LaTeX, which is efficient and artistic. For the convenience of readable and understandable, code is organised following specific criterion of variables naming, code layouts and comments.

References

[1] A. Oliva, A. Torralba, and P. G. Schyns, "Hybrid images," in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 527–532, ACM, 2006.

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