Image Filtering and Hybrid Images

Optical illusions with CV!!

Dipan banik
Debojyoti Mandal

Project 1
Semester 2

Big Data Analytics
RKMVERI
India
14 - 03 - 2021

1 Introduction

The objective of this project is to create a hybrid image by using the low frequency and high frequency component of two different images. **A hybrid image is the sum of a low-pass filtered version of the first image and a high-pass filtered version of a second image**. The cut-off frequency is a parameter of the filter which has been used to control the high-pass and low-pass frequency content of the images.

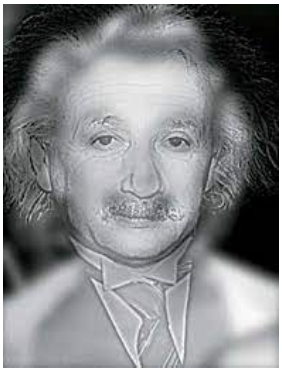


Figure 1: A Hybrid image of Einstein and Merlin Monroe

1.1 Filter working principle

This project focuses on the application of Gaussian filters on the given images. The low pass frequency content of an image can be developed by convolution of a Gaussian filter with the respective image. The high pass frequency image is fetched after subtracting the low pass frequency image from the main image.
In this project instead of using the standard '**cv2.getGaussianKernel()**' a '**my_getGaussianKernel()**' filter is constructed as a python function where the cutoff frequency and kernel size are taken as the parameters. Similarly instead of 'cv2.filter2d()' a python based 'my_imfilter()' has been constructed and used. The final hybrid image has been created by constructing and implementing 'create_hybrid_image()' function.

Apply the filtering formula to calculate the value of the dot product of the matrix of the image and the filter matrix:

$$H[m,n] = \Sigma_{k,l}\ G[k,l] * F[m+k, n+l]$$

point to be noted that here we have done correlation. Convolution does the exact same pointwise multiplication but it flips the kernel first. Anyway, as most of the kernels (also Gaussian Filter) are symmetric, it does not matter.

1.2 Create Hybrid Image

The hybrid image is constructed by combining two images filtered by high pass filter and low pass filter respectively. The key here is the cutoff frequency used to determine which part of the image can be preserved. For the first image, apply the filter to wipe off the components with higher frequency than the cutoff frequency and preserve the low frequency part. On the other hand, apply the same filter to the second image, and then subtract the filtered image from the original image to get rid of the low frequency part. Combine the images and get the hybrid image. Since the first image now only has low frequency part, it can be recognized when distance is large. The second image can be recognized better when the hybrid image is closer to eyes because high frequency part dominates in short distance.

2 Code Description

Here we have tried to implement the functions in *cv2* library without using the *cv2* library directly (at least to some extent!). *utils.py* module has some utility functions that has been used, but we shall discuss below only the functions in the module *student_code.py*.

2.1 Creating the filter - *my_getGaussianKernel*

cv2.getGaussianKernel gives a $n \times 1$ vector following Gaussian distribution. Then to make the $n \times n$ kernel we need to take the outerproduct of the vector with itself. But our function *my_getGaussianKernel* returns the whole kernel with a given sigma for the Gaussian distribution. It has also an option to give a custom mean (0 by default) but that would be redundant, as we can make the mean 0 by just a simple axis shift.

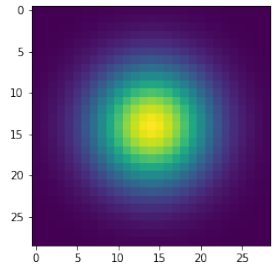


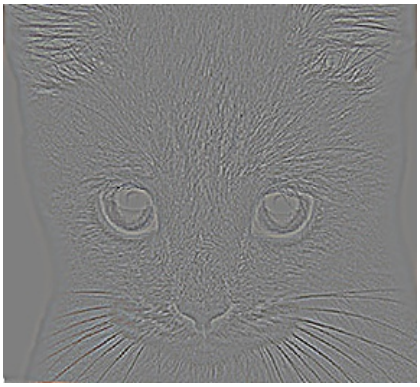
Figure 2: Gaussian kernel produced by our function

2.2 Applying the filter to the image - *my_imfilter*

this function tries to mimic the *cv2.filter2D* function. it convolves the kernel to the image (actually what it does is Correlation but as the kernel is symmetric ,

correlation and convolution becomes same here) and return the filtered image. Performance of our filter has been exhibited in Figure: 3.

(a) identity kernel



(b) laplacian kernel

(c) sobel kernel

(d) Gaussian kernel with high sigma

Figure 3: *my_imfilter* on different kernels

2.3 creating the hybrid image - *create_hybrid_image*

as discussed before, we already know what is an hybrid image and how it can be formed. the concept has been implemented in this function i.e. one image has been passed through a low pass filter (gaussian filter) and the other one has been passed through a high pass filter (sharpening filter[1]). Then both of the filtered images have been added (the result can exceed the limit 0-255. So normalization has been done to make sure all the pixel values are in [0,1]. Which is know as **Clipping**.) the result is exactly our hybrid image. A 3×3 sharpening filter is shown below :

[1]Sharpening filters are basically filters with high value in the middle cell and equal negative values in the cells around. We have subtracted the low passed image from its original which results to the same effect.

-1	-1	-1
-1	9	-1
-1	-1	-1

(a) initially Blurred cat image

(b) sharp cat image

(c) the blurred dog image

(d) sum of those low and high frequency images

Figure 4: Hybrid image creation

3 Conclusion

From the above images, it can be concluded that the hybrid image processing is dependent on the choice of a proper Gaussian kernel. The reason for the success of this kind of image processing is the small size of the filter. The padding width determined by the filter size is also small, thus the influence of the zero values is not significant.
It is quite astonishing to observe that the running time of the direct method in which we convolve the image with a full 2D Gaussian filter is lesser in

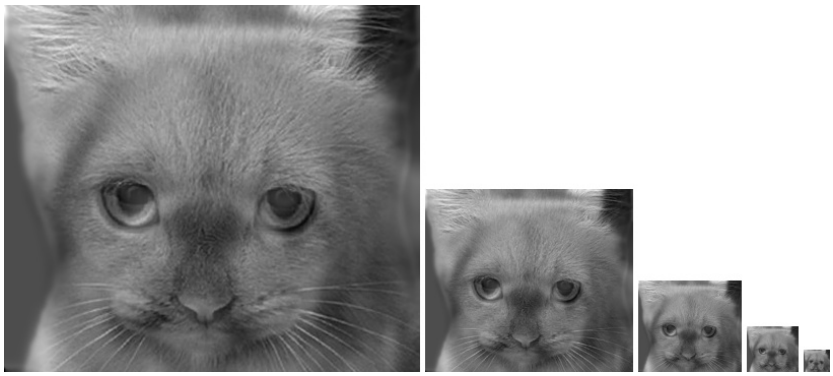
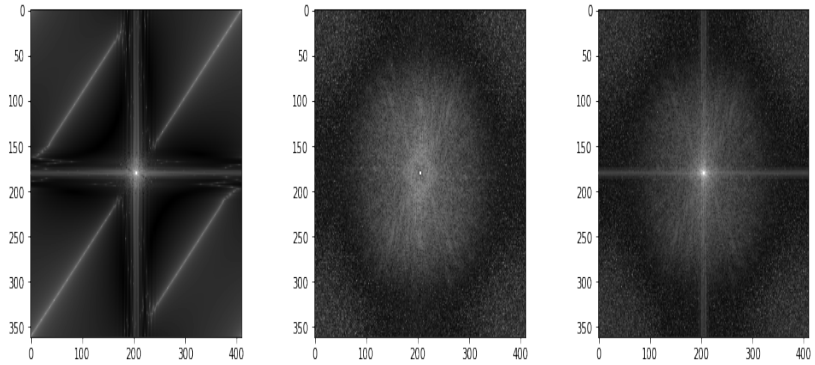


Figure 5: A Hybrid image Gaussian Pyramid on Grayscale image



(a) lpf image frequency response (b) hpf image frequency response (c) hybrid image frequency response

this project than the convolution with two separable 1d filter while using the 'cv2.getGaussianKernel' and 'my_imfilter'. The respective running time are 1.729284 seconds and 2.534489 seconds. Whereas it is generally known that the approach of two separable 1d filter is faster than the direct method. But both the methods give the same output images irrespective of the running time.

In a Gaussian pyramid, subsequent images are weighted down using a Gaussian average (Gaussian blur) and scaled down. Each pixel containing a local average corresponds to a neighborhood pixel on a lower level of the pyramid. This technique is used especially in texture synthesis.

The output hybrid images clearly depicts that the high frequency components of a image are the edges where the values of the pixels varies sharply, and the low frequency components are blurred color blocks without clear edges among

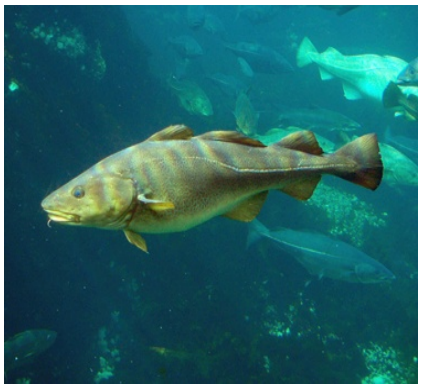
Figure 7: Gaussian pyramid of the hybrid image

them. When the image is close to the observer, the edge details are well caught by the eyes while the color blocks of image dominating when the image is far away. Thus we should choose the image with more sharp edges as the high frequency part and the image with less color blocks as the low frequency part to make a appropriate hybrid image.

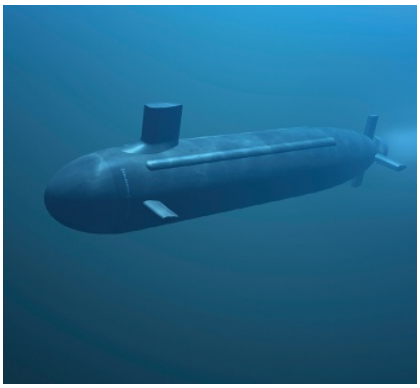
4 Participation :

1.my_getGaussianKernel()- This function has been done by Dipan Banik.
2.my_imfilter()- This function has been done by Debojyoti Mondal.
3.create_hybrid_image()-This function has been developed by both of us.
4.Code implementation and latex editing has been done collaboratively by both of us.

5 Some more examples



(a) original fish image



(b) original submarine image

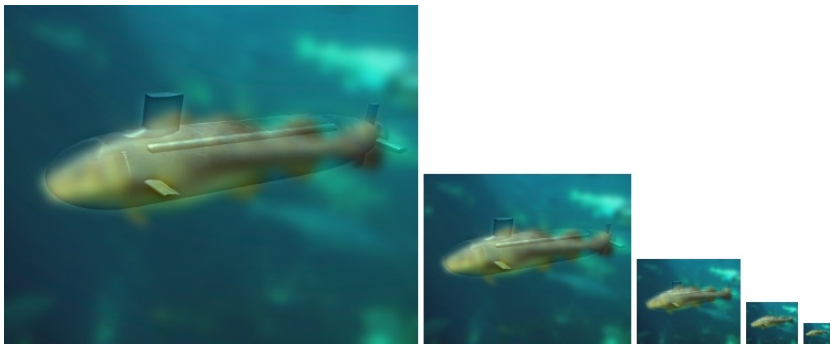


Figure 9: Gaussian pyramid of the hybrid image

(a) original bird image

(b) original plane image

Figure 11: Hybrid image of bird and plane