Zewail City of Science and Technology

Computer vision
CIE 552

Generating Hybrid images and implementation of convolution(Computer vision project 1)

Ahmed Adel	201901464
Hossam Ashraf Fathy	201901898

March 10, 2023

Contents

1 Part one

Here we were asked to implement a convolution using the filter in 2 dimensions to imitate scipy.corelated2d(); it should have the following characteristics.

- Pad the input image with zeros.
- Support grayscale and color images.
- Support arbitrary shaped odd-dimension filters
- Return an error message for even filters, as their output is undefined

In the end, we were supposed to return the same image with an identity filter, with the same resolution.

1.1 Methods

Here we describe the whole pipeline in details.

1.1.1 create_gaussian_filter()

We used a mesh grid to generate X, and Y matrix, and then normalized the matrix by dividing by $\pi * sigma^2$ Inputs:

1. ksize: kernel size of the gaussian filter
2. sigma: Standard deviation of the gaussian filter

Outputs:

1. 2D matrix, of guassian filterr.

1.1.2 pad_image_reflection(): Bonus

We aimed here to pad the image with the proper number of pixels left, right, up, and down with reflection to avoid some problems related to convolution around borders. If the kernel is for example 9x9, we need to pad the image with 8 pixels 4 on the left, and 4 on the right.

To iron the details out for these function, consider a kernel of 9x9, and an image size of 200x200

1. We got the first 4(9 integer division by 2 = 4) column of each channel(Flipped by using numpy.fliplr()).
2. We secondly got the last four columns of the image of each channel flipped.
3. We hoizetally stacked all of them (using numpy.hstack).
4. We did the same thig for up and down part the image (using numpy.flipud() for flipping, and numpy.vstack() for stacking vertically).

Inputs:

1. img: Image to be padded
2. x2, y2: size of kernel

Outputs:

1. 2D/3d matrix, padded image

1.1.3 conv_2d()

Here we convolved the image in the spatial domain, using regular multiplications and additions, we first padded the image before convolution.

Inputs:

1. img: Image
2. kernel: kernel to convolve the image with.

Outputs:

1. 2D/3d matrix, convolved image

1.1.4 conv_fft(): Bonus

multiplication of two images in the frequency domain after FFT, can greatly improve the performance, we used the built-in function numpy.fft for converting the image to the frequency domain.

Details:

1. transformed both image and kernel to the frequency domain
2. multiplied both of them
3. returns the output to the spatial domain using ifft
4. took the absolute for the output, as the image can't have any negative values.

1.1.5 my_imfilter()

Convolving the image with a kernel, we supported colored images and gray images, and gave an option for it to be convolved in the spatial domain, or using fft as it's much faster.

Inputs:

1. img: Image
2. kernel: kernel to convolve the image with
3. type: if 0; then using normal convolution, if 1; then use fft.

Outputs:

1. 2D/3d matrix, padded image

1.2 Results

For the Identity kernel, the output image should be as same as the input image.

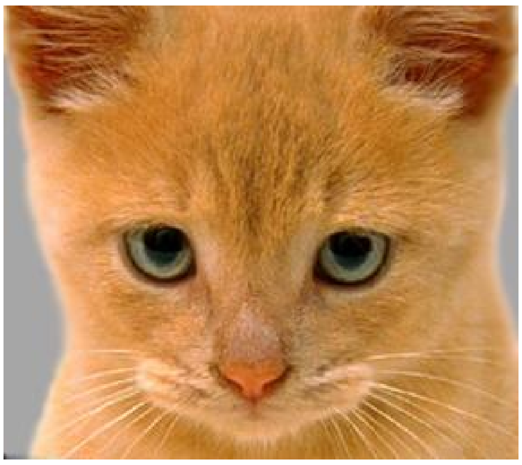
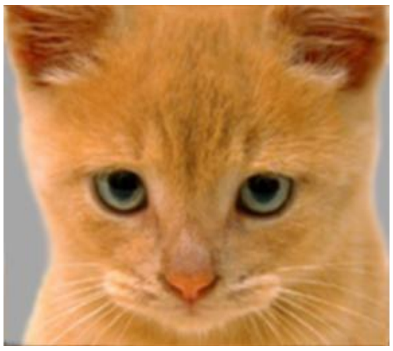


Figure 1: Identity kernel

For blurring using gaussian we tried 3x3, and 7x7 kernels.


(a) 3x3 kernel

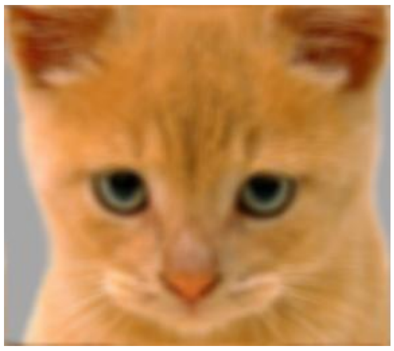

(b) 7x7 kernel

Figure 2: Constellation While not using Channel coding

We see that having a bigger kernel size causes the image to be more blurred

For getting high frequency only in the image, we subtracted the low-frequency content from the original image, and here is the result.

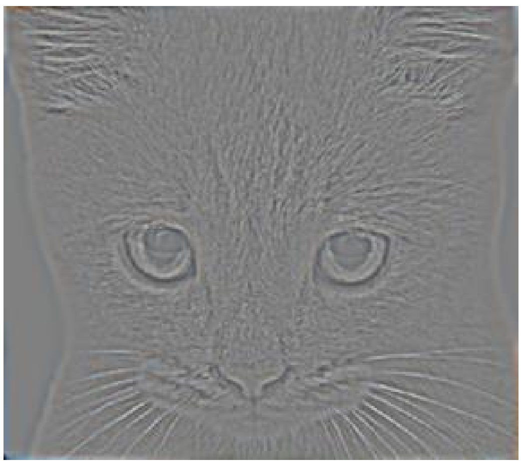


Figure 3: High pass image

We finally tried the laplacian filter, which also can be used to detect edges.

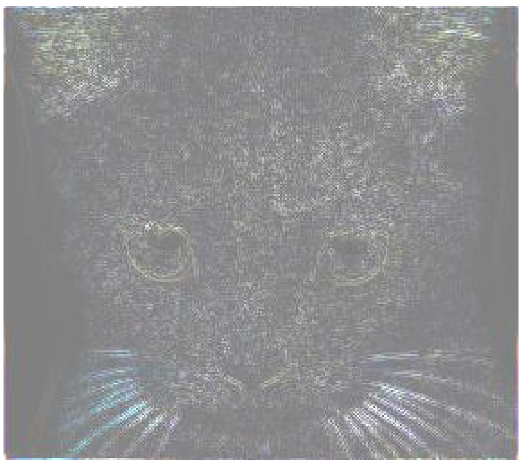


Figure 4: Laplacian kernel

2 Hybrid images

In simple words, a hybrid image is a combination of two images, we extract a low frequency from one image, extract the high frequency of another image, and add them together, supposedly; that creates an image that might have different perceptions from the human eye based on how close human eye to the image is. If the human eye is very close to the image, It is supposed that it will focus on every little detail, hence detecting the high-frequency image clearer, but if they are away from the image, or the image is very small, it will focus on a low-frequency image, as though they have an overview of the image.

2.1 Method

To clear the steps out, we did the following steps to generate hybrid images.

- We used a medium kernel (9) for blurring the first image to extract low frequency.
- We then convolved the kernel with the first image.
- To extract the high frequency from the second image, we first extract the low frequency from it and subtracted it from the original image.
- Note that we used different kernel sizes here(31), to be able to remove all low frequencies easily
- finally we added the two images, and clipped the results to be between 0, and 1.

2.2 Results

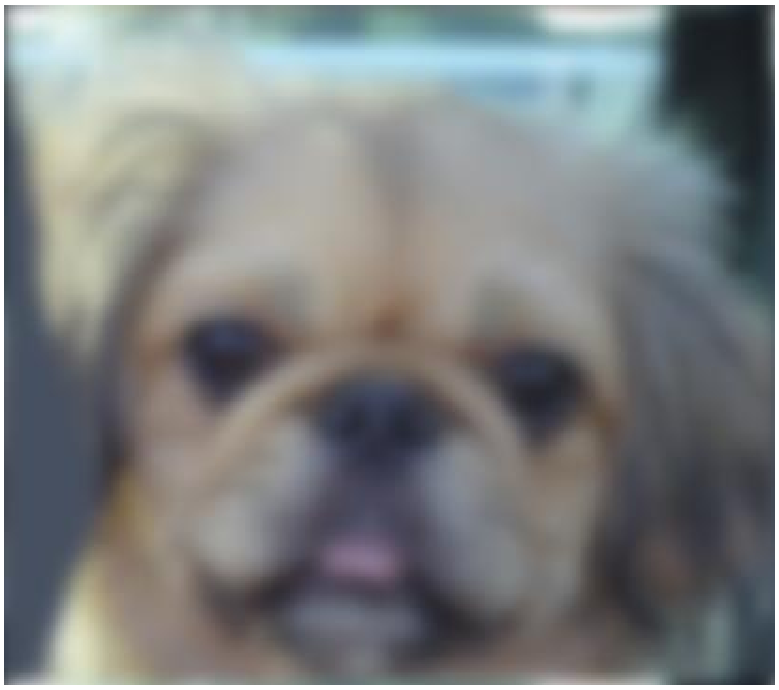


Figure 5: Low frequency image

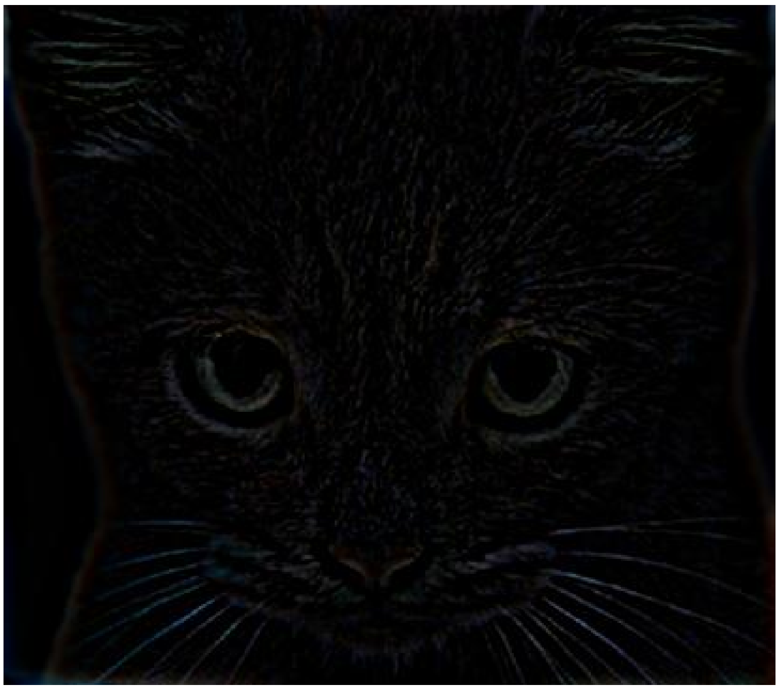


Figure 6: High frequency image

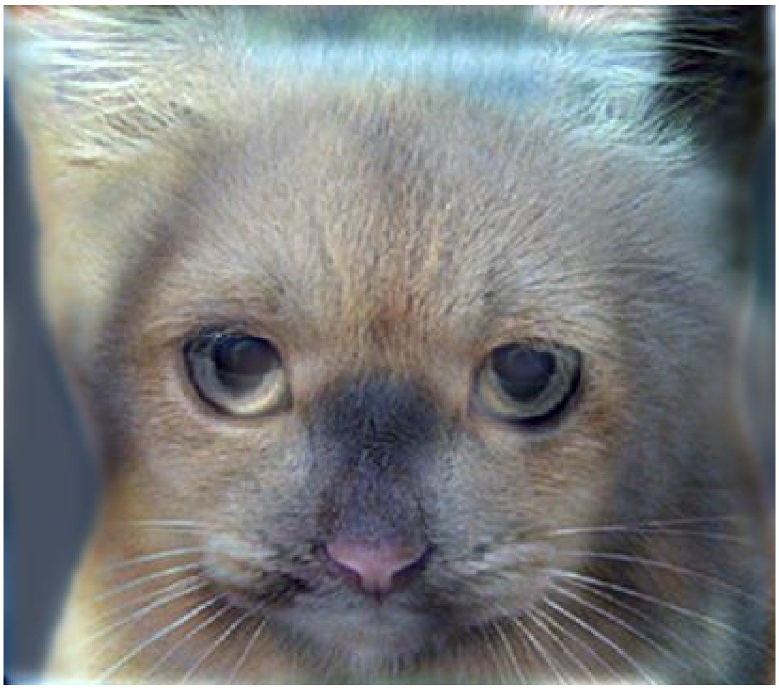


Figure 7: Hybrid image

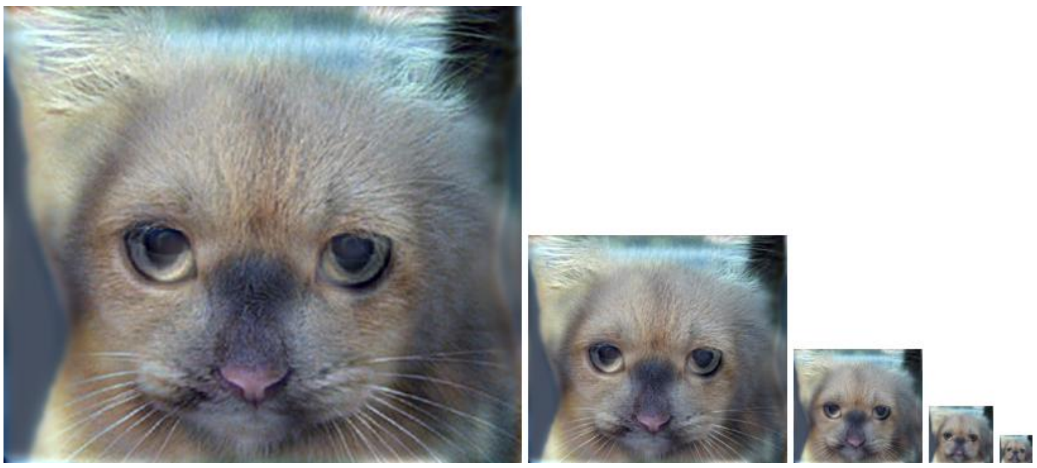


Figure 8: Hybrid image with scales

We see here, for the hybrid images, for a big image we can see the cat, but for the small image, we can see the dog.