Zewail City of Science and Technology
University of Science and Technology
Communications & Information Engineering Program
CIE 552 - Spring 2020

project 1

Image Filtering and Hybrid Images

Report

Table of contents

Project objective

This project aimed to generate hybrid images from two images with its high frequency content comes from one image and its low frequency content comes from the other. We implemented all the used functions from scratch except the visualization functions.

Algorithm in brief

The algorithm we follow in this project is as following:

- Get two suitable images (same size and have similar background).
- Implement a low pass filter and high pass filter functions.
- Implement a function that does convolution.
- Using the previous function, we convolve the two images with the filters. One image with the low pass filter and the other one with the high pass filter
- Finally, add the two resulting images together to get the hybrid image.
- It is obvious that you can see the image with low frequency far a distance but you can see the image with the high frequency from a close distance.

Implementation

The first function implemented **my_imfilter** is used to convolve the image and the filter kernel in the spatial domain.
We do all the requirements in this function and we will illustrate more in the following discussion.
Inputs: the image and the filter
Output: the image after filtering
Steps:
1.Getting the shape of the input image and checking if it is 2D or 3D , then get the filter shape to determine the required padding size
And also make an assertion to prevent the user from entering even size filter to be easier to determine the central pixel.

```
def my_imfilter(image: np.ndarray, filter: np.ndarray):

  # filter dimensions:
```

```
  filter_rows = filter.shape[0]
  filter_columns = filter.shape[1]

  assert (((filter_rows % 2) or (filter_columns % 2)) != 0), " Size of the
kernel should be odd "



  img_shape = image.shape

  if len(img_shape) == 3 :

    r = image[:, :, 0]
    g = image[:, :, 1]
    b = image[:, :, 2]

  # image demnsions:
    image_rows = r.shape[0]
    image_columns = r.shape[1]
```

2.Padding the image according to the filter size to be able to do the convolution between them.

```
# Padding the image:
    pad_rows = (filter_rows-1) // 2
    pad_columns = (filter_columns-1) // 2
    padded_img = []
    padded_img.append(np.pad(r, ((pad_rows, pad_rows), (pad_columns,
pad_columns)), 'constant', constant_values=0))
    padded_img.append(np.pad(g, ((pad_rows, pad_rows), (pad_columns,
pad_columns)), 'constant', constant_values=0))
    padded_img.append(np.pad(b, ((pad_rows, pad_rows), (pad_columns,
pad_columns)), 'constant', constant_values=0))
```

3. Doing the convolution by looping over each group of pixels in the image and multiply them with the corresponding filter pixels then sum all and put the value in a new image.

```
 filtered_image = np.zeros_like(r)
```

```
    for channel in padded_img:
      new_img = []
      for m in range(image_rows):
        for n in range(image_columns):
          summ = 0
          summ = np.sum(np.multiply(channel[m:m+filter_rows,
n:n+filter_columns], filter))
          new_img.append(summ)
      new_img = np.asarray(new_img)
      new_img = new_img.reshape(image_rows, image_columns)
      filtered_image = np.dstack((filtered_image, new_img))
```

4. Removing the first channel which is zeros only in the case of 3D image and get the final image.

```
filtered_image = filtered_image[:, :, 1:]
```

The following part is in the case of 2D image and it has the same steps of the 3D part considering the different dimensions.

```
  elif len(img_shape) == 2:
    # image demnsions:
    image_rows = image.shape[0]
    image_columns = image.shape[1]

    # Padding the image:
    pad_rows = (filter_rows - 1) // 2
    pad_columns = (filter_columns - 1) // 2

    padded_img = []
    padded_img.append(np.pad(image , ((pad_rows, pad_rows), (pad_columns,
pad_columns)), 'constant', constant_values=0))

    padded_img = np.asarray(padded_img)
```

```
    padded_img = padded_img.reshape(image_rows+ (2*pad_rows),
image_columns+(2*pad_columns))

    filtered_image = np.zeros_like(image)



    new_img = []
    for m in range(image_rows):
      for n in range(image_columns):
        summ = 0
        summ = np.sum(np.multiply(padded_img[m:m + filter_rows, n:n +
filter_columns], filter))
        new_img.append(summ)
    new_img = np.asarray(new_img)
    new_img = new_img.reshape(image_rows, image_columns)



  return filtered_image
```

Second function implemented is **`create_gaussian_filter.`** The function creates a symmetric gaussian filter with mean zero and a given standard deviation. The output filter is normalized so that the summation of its elements equals one.

Inputs:

- Ksize : filter size (must be odd)
- sigma : standard deviation

Output: symmetric normalized gaussian filter with size (ksize×ksize), mean zero and standard deviation sigma.

Steps:

- Creating a vertical gaussian filter and a horizontal gaussian filter then multiply them with each other.
- Divide the matrix by its sum to make its sum equals one to avoid any saturation in the filtered image.

```
def create_gaussian_filter(ksize: int, sigma: float):
    x = gaussian(M=ksize, std=sigma)
```

```
    y = gaussian(M=ksize, std=sigma)
    x_1, y_1 = np.meshgrid(x, y)
    gaussian_kernel = x_1 * y_1
    gaussian_kernel = gaussian_kernel / gaussian_kernel.sum()
    return gaussian_kernel
```

Third function implemented is **`gen_hybrid_image`** to generate the hybrid image from two images. One image contributes with the low frequencies content, the other image contributes with the high frequency content and the hybrid image is the sum of those low and high frequencies. Function uses a gaussian filter with standard deviation equals to the cutoff frequency for removing high frequencies

Inputs:

- image1 : The image from which to take the low frequencies.
- image2 : The image from which to take the high frequencies.
- cutoff_frequency :The standard deviation, in pixels, of the Gaussian blur that will remove high frequencies.

Output: low frequencies content of the first image, high frequencies content of the second image, hybrid image

Steps:

- Creation of a gaussian function using **`create_gaussian_filter`** with the input cutoff_frequaency as the sigma and size equals to 3×sigma (as recommended).

```
def gen_hybrid_image(image1: np.ndarray, image2: np.ndarray,
cutoff_frequency: float):

   assert image1.shape == image2.shape
   kernel_size = 3*cutoff_frequency
   kernel = create_gaussian_filter(kernel_size, cutoff_frequency)
```

- Using **my_imfilter** to convolve the filter with the image1 to blur it and extract its low frequencies.

```
low_frequencies = my_imfilter(image1,kernel)
```

- Using m_imfilter to convolve the filter with image2 and extract its low frequencies content then subtract it from the original image to have the high frequencies content.

```
   high_frequencies = image2 - my_imfilter(image2,kernel)
```

- Generation of the hybrid image by the summation of the low and high frequencies

```
hybrid_image = low_frequencies+ high_frequencies
```

- Clipping of the values larger than one and lower than zero in order to avoid any problems with further used functions.

```
hybrid_image[hybrid_image < 0.0] = 0.0
hybrid_image[hybrid_image > 1.0] = 1.0
```

- Then return the results

```
return low_frequencies, high_frequencies, hybrid_image
```

Results

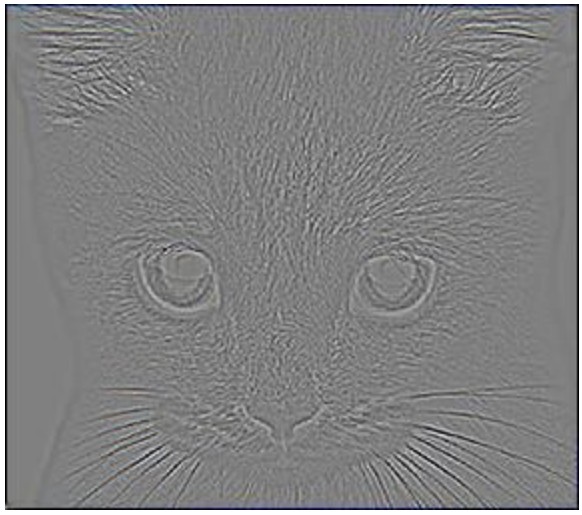
Part 1: Image filtering

Identity filter

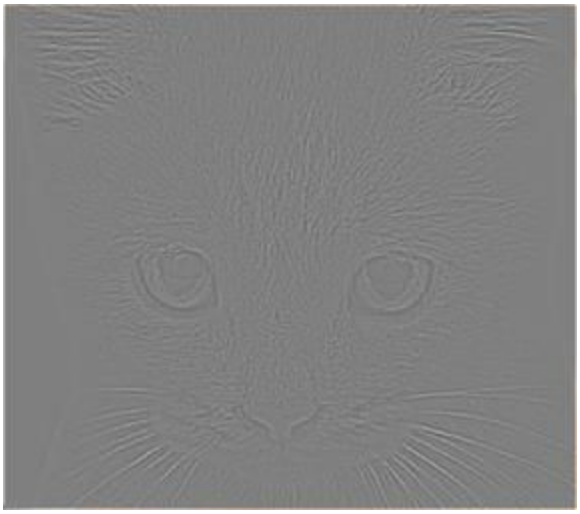
Blur Filter

Large Blur Filter

Sobel Filter



La Placian filter



High Pass filter

To show the difference between zero padding and content reflect padding, we increase the padding rows and columns to be 20 rows and 20 columns on the image, 10 on every side to produce the following result:

Zero Padding

Content Reflection

Part 2: Hybrid Image Generation

- **Cat and dog**

Low frequencies image

High frequencies content

Hybrid image

Hybrid image with different scales

Note: results from other image are in the data folder associated with the report

Bonus requirements

1. Pad with reflected image content

We make a separate function ” my_imfilter_reflect ” that do the same as “ my_imfilter “ but with content reflection padding.
The key difference in this function is the following line:

```
padded_img.append(np.pad(r, ((pad_rows, pad_rows), (pad_columns, pad_columns)), 'reflect'))
```

We used “reflect” as the mode for the function “pad” from “numpy” library.

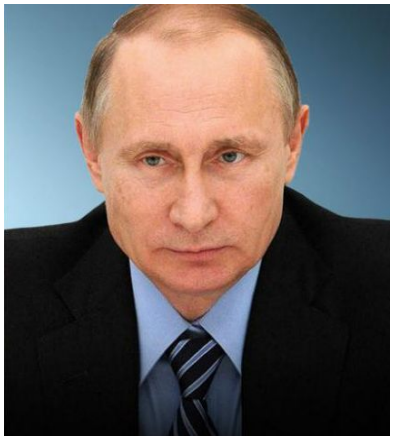
2. FFT-based convolution

```
def apply_filter_in_freq(img, f):
    img_in_freq = np.zeros(img.shape)
    img_in_freq[:,:,0] = fftpack.fft2(img[:,:,0])
    img_in_freq[:,:,1] = fftpack.fft2(img[:,:,1])
    img_in_freq[:,:,2] = fftpack.fft2(img[:,:,2])
    filter_in_freq = fftpack.fft2(f, img.shape[0:2])
    filtered_img_in_freq = np.zeros(img_in_freq.shape)
    filtered_img_in_freq[:,:,0] = np.multiply(img_in_freq[:,:,0], filter_in_freq)
    filtered_img_in_freq[:,:,1] = np.multiply(img_in_freq[:,:,1], filter_in_freq)
    filtered_img_in_freq[:,:,2] = np.multiply(img_in_freq[:,:,2], filter_in_freq)
    filtered_img = np.zeros(filtered_img_in_freq.shape)
    filtered_img[:,:,0] = fftpack.ifft2(filtered_img_in_freq[:,:,0])
    filtered_img[:,:,1] = fftpack.ifft2(filtered_img_in_freq[:,:,1])
    filtered_img[:,:,2] = fftpack.ifft2(filtered_img_in_freq[:,:,2])
    filtered_img = np.absolute(filtered_img)
    filtered_img = (filtered_img - np.min(filtered_img)) / (np.max(filtered_img) -np.min(filtered_img))
    return filtered_img
```

The code is not working properly as it should but this is trail version.

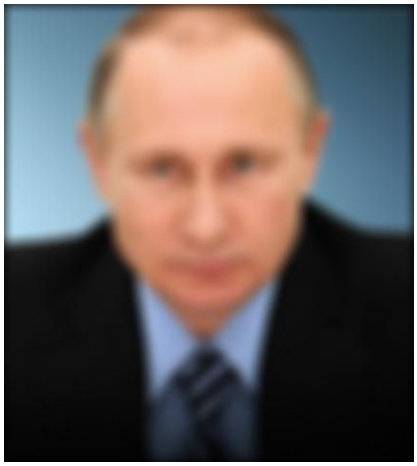
3. Our own hybrid image, formed from images not in the code distribution

We propose the following images:



Putin

Trump



Low Frequency Image

High Frequency Image

Hybrid Image