ELEC564 – Spring 2023
Homework 2

Date: February 13th, 2023

Student's Name: Hsuan-You (Shaun) Lin / Net ID: hl116

Link to Colab notebook:

https://colab.research.google.com/drive/1poWypOU5YYFVBu4a3jSII7SqrHpl5iPb?usp=sharing

1.0 Hybrid Images

1.1 Gaussian kernel

Implement function gaussian2D(sigma, kernel_size) that returns a 2D gaussian blur kernel, with input argument sigma specifying a tuple of x,y scales (standard deviations), and kernel_size specifying a tuple of x,y dimensions of the kernel. The kernel should be large enough to include 3 standard deviations per dimension.

```
[ ] def gaussian2D(sigma, kernel_size):
        # Unpack the size of the kernel into x_size and y_size
        x_size, y_size = kernel_size

        # Calculate the limits of x and y for the kernel
        x_limits = (x_size - 1) / 2
        y_limits = (y_size - 1) / 2

        # Create a 2D grid of values for x and y
        x, y = np.mgrid[-x_limits:x_limits+1, -y_limits:y_limits+1]

        # Apply Gaussian filter (lowpass) from Lec 6.
        kernel = (1 / (2 * np.pi * sigma * sigma)) * np.exp(- (np.square(x) / (2 * np.square(sigma)) + np.square(y) / (2 * np.square(sigma))))

        # Normalize the values of the 2D Gaussian kernel so that their sum is 1
        kernel = kernel / np.sum(kernel)

        return kernel
```

Figure 1. gaussian2D function Python code

1.2 Create Hybrid Images

Choose two images (A and B) of your choice that you will blend with one another and convert them to grayscale (you can use opencv.cvtColor). These images can be of faces, or any other objects. Try to make the objects in the two images occupy roughly the same region in the image (if they don't you can use the imwarp function you wrote in Homework 1 to manually align them!).

Construct a hybrid image C from A (to be seen close-up) and B (to be seen far away) as follows: C = blur(B) + (A-blur(A)), where blur is a function that lowpass filters the image (use the Gaussian kernel you coded in 1.1 for this). Try

different values of sigma for the Gaussian kernel. How does the amount of blurring affect your perception of the results? In your report, please show your input images labeled clearly as A and B, and attach the result C for a value of sigma which you feel demonstrates the illusion the best, at both the original size and a downsampled size. As a sanity check, you should be able to see the identity change when looking at the original and downsampled versions.

Figure 2. Input image A

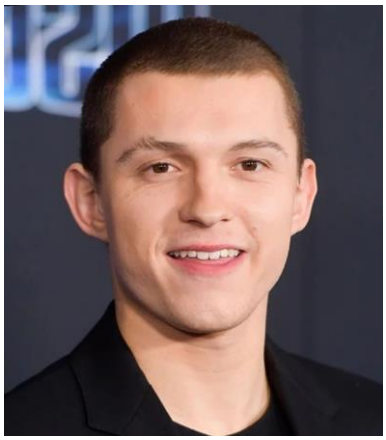

Figure 3. Input image B

C

Figure 4. Output hybrid image C

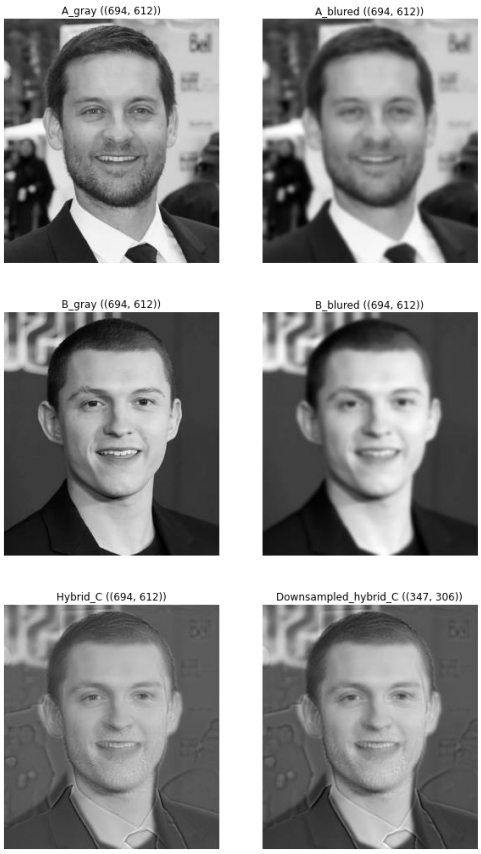

Figure 5. Image A_gray, A_blured, B_gray, B_blured, Hybrid C and Downsampled hybrid C

- **How does the amount of blurring affect your perception of the results?**
  From my observation, the amount of blurring will affect the result of the hybrid image, when the sigma value is higher, the Image A in hybrid image C will be clearer, on the contrary, the smaller the sigma value, the Image B in hybrid image C will be clearer, and I finally choose to set sigma=4 that I feel it can demonstrates the best illusion, as you can see image B (Tom Holland) appears to have grown some mustache.

1.3 Fourier Spectra

For the sigma value you chose in 1.2, show images of the Fourier spectra magnitudes of images A, B, blur(B), A-blur(A), and C. You can get the magnitude of the Fourier spectrum coefficients of an image 'x' by running: X = numpy.abs(numpy.fftshift(numpy.fft2(x)))

By default, numpy.fft2 will place the zero frequency (DC component) of the spectrum at the top left of the image, and so numpy.fftshift is used here to place the zero frequency at the center of the image. When displaying the Fourier spectrum with matplotlib.pyplot.imshow, the image will likely look black. This is

because the DC component typically has a much higher magnitude than all other frequencies, such that after rescaling all values to lie in [0,1], most of the image is close to 0. To overcome this, display the logarithm of the values instead.

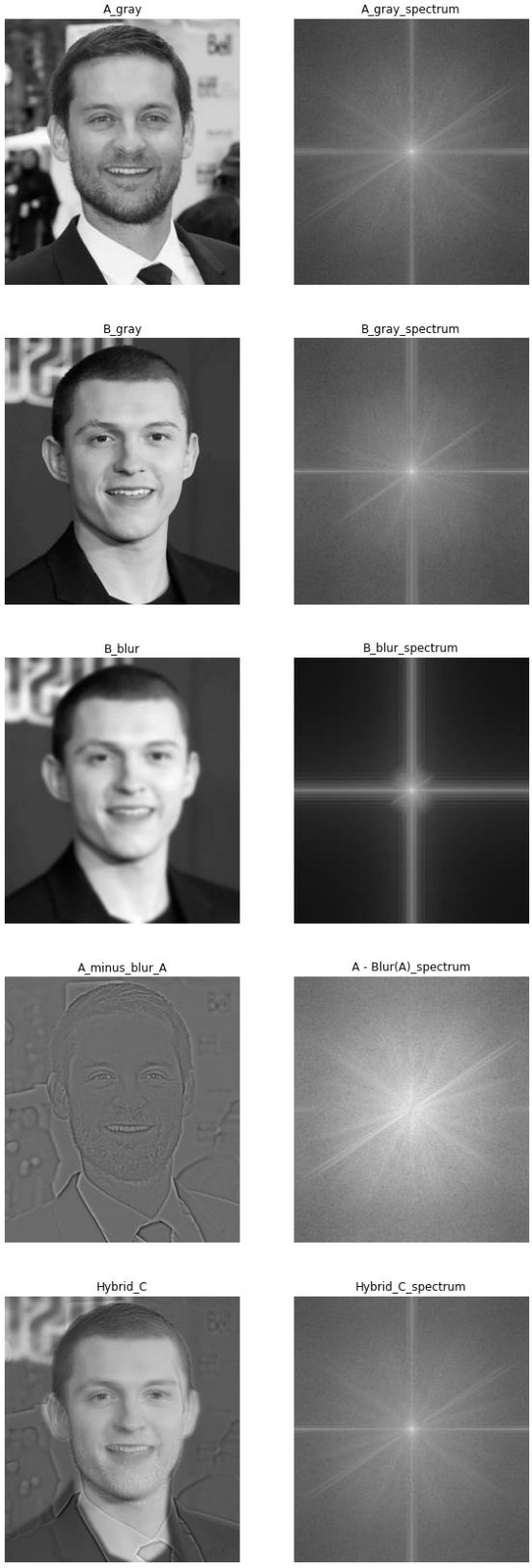

Figure 6. Fourier spectrum (right side) and its corresponding images (left side)

2.0 Laplacian Blending

2.1 Gaussian Pyramid

Write a function gausspyr(I, n_levels, sigma)that returns a Gaussian pyramid for image I with number of levels n_levels and Gaussian kernel scale sigma. The function should return a list of images, with element i corresponding to level i of the pyramid. Note that level 0 should correspond to the original image I, and level n_levels - 1 should correspond to the coarsest (lowest frequency) image.

```
[ ]  def downsample(img):
         # Downsample the image by taking every other row and column
         return img[::2, ::2, :]

     def gausspyr(I, n_levels, sigma):
         # Create a list to store the pyramid levels
         pyramid = []

         pyramid.append(I)

         for i in range(n_levels):
             # Blur the current level of the pyramid
             blurred = blur(I, sigma, kernel_size=(int(6*sigma), int(6*sigma)))

             # Downsample the blurred image
             downsampled = downsample(blurred)

             # Add the downsampled image to the pyramid
             pyramid.append(downsampled)
             I = downsampled

         return pyramid
```

Figure 7. gausspyr(I, n_levels, sigma) function Python code

Figure 8. Test output image using gausspyr() function

2.2 Laplacian Pyramid

Write a function lappyr(I, n_levels, sigma)that returns a Laplacian pyramid for image I with number of levels n_levels and Gaussian kernel sigma. The function should return a list of images, with element i corresponding to level i of the pyramid. Note that level 0 corresponds to the details of the original image I, and level n_levels - 1 corresponds to the low-frequency residual image.

```python
def convolution(img):
    kernel = np.array([[0.25, 0.5, 0.25],
                       [0.5, 1, 0.5],
                       [0.25, 0.5, 0.25]])
    # Apply convolution to the input image
    convoluted = cv2.filter2D(img, -1, kernel)
    return convoluted

def upsample(img):
    # Create an empty image with double the size in both dimensions
    upsampled = np.zeros([img.shape[0] * 2, img.shape[1] * 2, img.shape[2]])

    # Copy values from the original image to every other row and column in the upsampled image
    upsampled[::2, ::2, :] = img[::1, ::1, :]

    # Return the upsampled image
    return upsampled

def lappyr(I, n_levels, sigma):
    # Create a list to store the pyramid levels
    pyramid = []

    # pyramid.append(I)

    for i in range(n_levels):
        # Apply Gaussian blur to the current level
        gaussianed = blur(I, sigma, kernel_size=(int(6*sigma), int(6*sigma)))
        # Downsample the blurred image
        downsampled = downsample(gaussianed)
        # Upsample the downsampled image
        upsampled = upsample(downsampled)
        # Perform convolution on the upsampled image
        convoluted = convolution(upsampled)
        # Compute the Laplacian level by subtracting the convoluted image from the original level
        L = I - convoluted
        # Append the Laplacian level to the pyramid
        pyramid.append(L)

        # Check if we have reached the last level
        if i == n_levels-1:
            # Append the downsampled image to the pyramid
            pyramid.append(downsampled)
        # Set the current level to the downsampled image for the next iteration
        I = downsampled
    return pyramid
```

Figure 9. lappyr(I, n_levels, sigma) function Python code

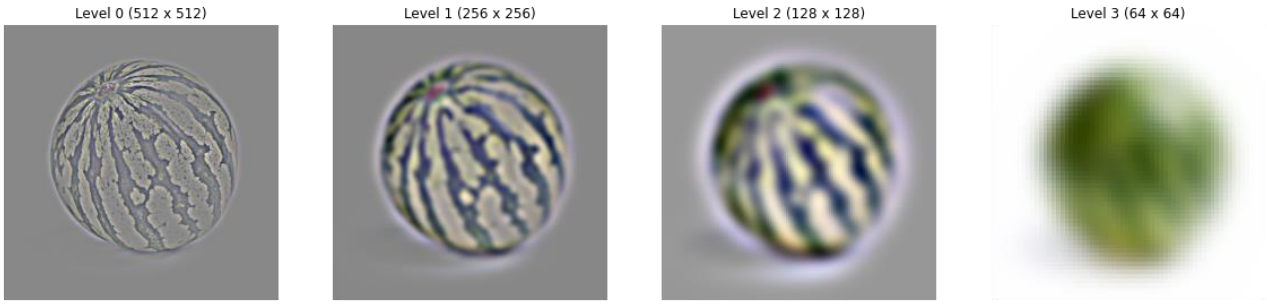

Figure 10. Test output image using lappyr() function

2.3 Image Blending

Choose two images A and B depicting different objects and resize them to the same shape. You may want to use your imwarp function from Homework 1 to align the scales/orientations of the objects appropriately (as was done in the example in Fig. 2) so that the resulting blend will be most convincing. Create a

binary mask image mask which will have 1s in its left half, and 0s in its right half (called a 'step' function). Perform blending with the following operations:

1. **Build Laplacian pyramids for A and B.**
2. **Build a Gaussian pyramid for mask.**
3. **Build a blended Laplacian pyramid for output image C using pyramids of A, B, and mask, where each level *lck* is defined by the equation $I_k^C = I_k^A * m_k + I_k^B * (1 - m_k)$.**
4. **Invert the combined Laplacian pyramid back into an output image C.**

Show the following in your report: (1) Images from all levels of the Laplacian pyramids for A and B, (2) images from all levels of the Gaussian pyramid for mask, and (3) your final blended image C.

Figure 11. Input images A (watermelon), B (sweet melon) and binary mask

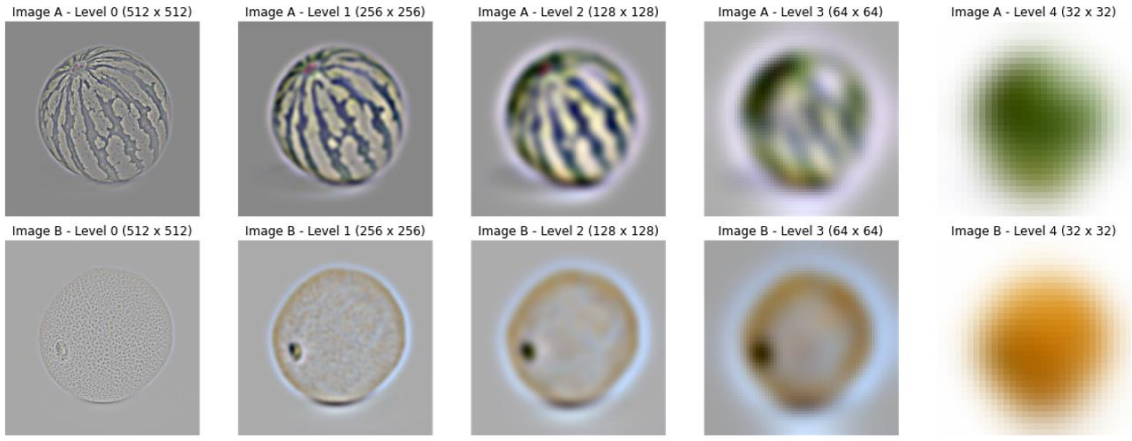

Figure 12. Images from all levels of the Laplacian pyramids A and B

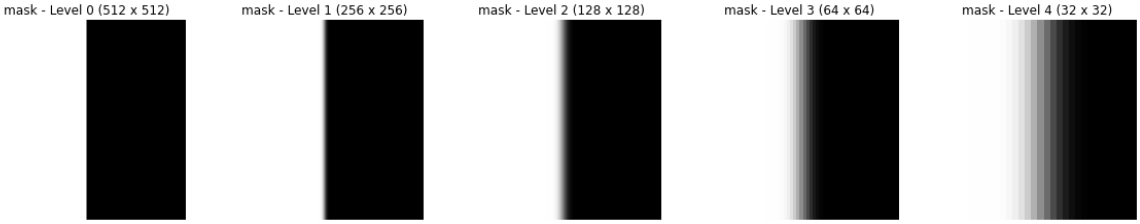

Figure 13. Images from all levels of the Gaussian pyramid for mask

Figure 14. Final blended image C and images from all levels of pyramid for image C

2.4 Blending two images with a mask other than a step

Laplacian blending is not restricted to only combining halves of two images using a step mask. You can set the mask to any arbitrary function and merge images, as shown in this example. Demonstrate a Laplacian blend of two new images using a mask other than step.

Figure 15. Input images A (the ancient one from Marvel movie), B (shifted the ancient image) and binary mask

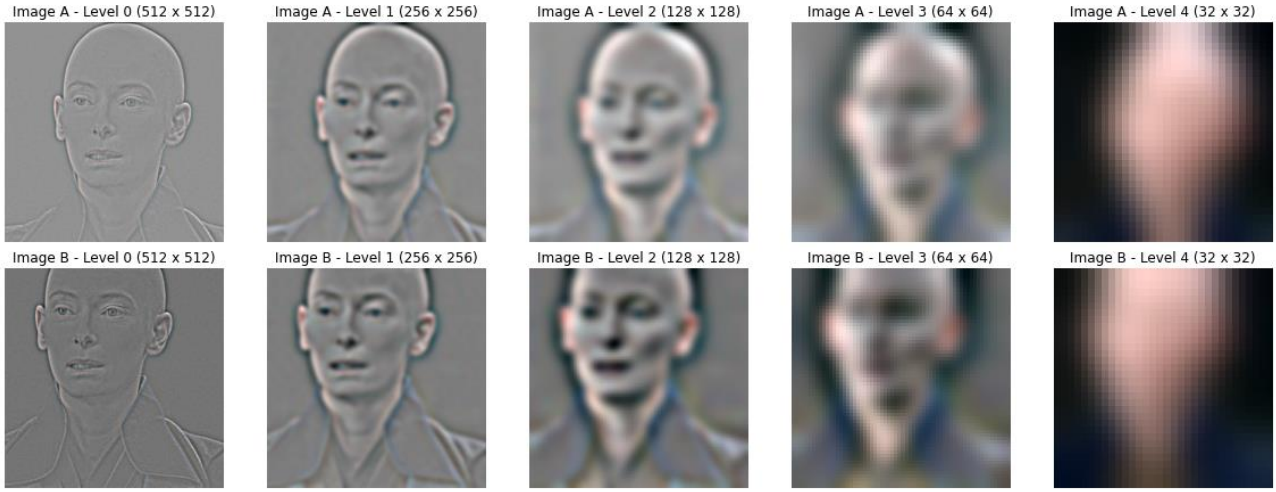

Figure 16. Images from all levels of the Laplacian pyramids A and B

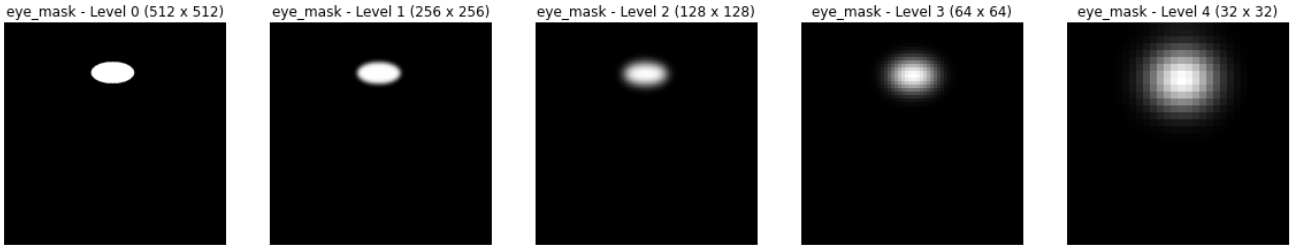

Figure 17. Images from all levels of the Gaussian pyramid for mask

Figure 18. Final blended image C and images from all levels of pyramid for image C

3.0 Pulse Estimation from Video

3.1 Read video into notebook and define regions of interes

Upload the video into your Colab environment. Note that it may take several minutes for the upload to complete due to the size of the file. You can then read the video frames into a numpy array using the read_video_into_numpy function provided here.

Using the first video frame, manually define rectangles (row and column boundaries) that capture 1) one of the cheeks and 2) the forehead.

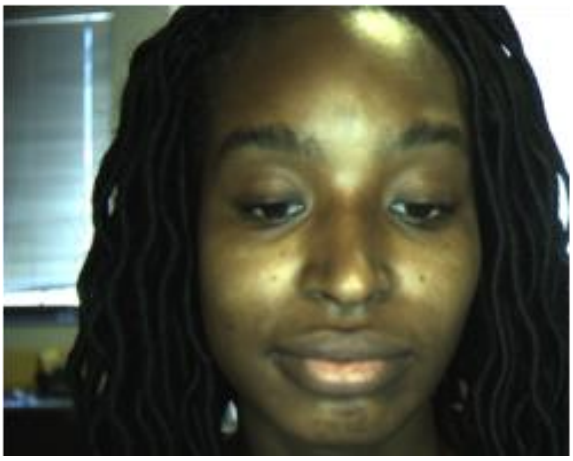

Figure 19. Alice’s first video frame image

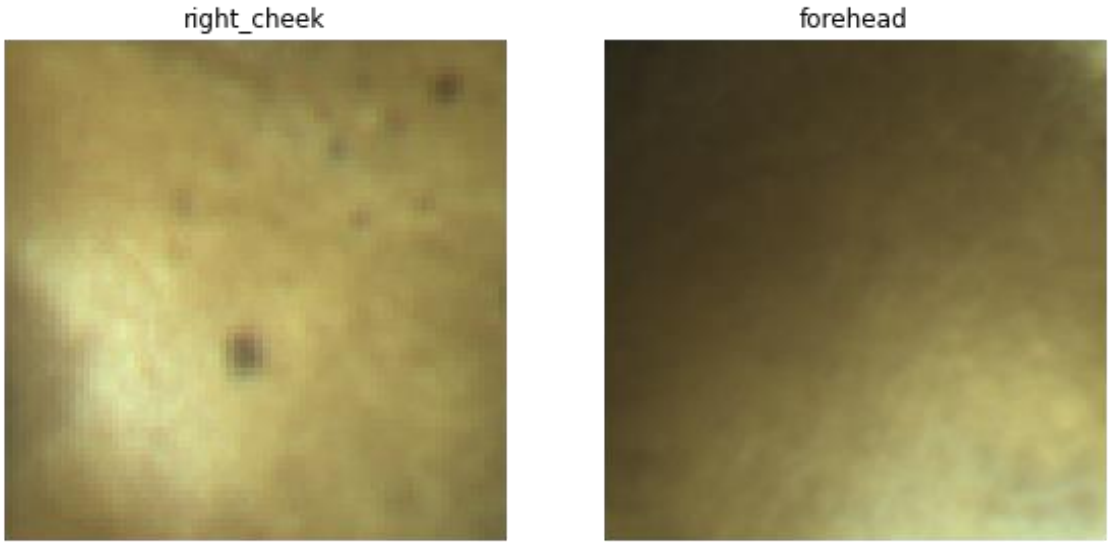

Figure 20. Alice's cheek and forehead images

3.2 Compute signals

Now compute the average Green value of pixels for all frames for each facial region (cheek, forehead). This gives a 1D signal in time called the Photoplethysmogram (PPG) for each region.

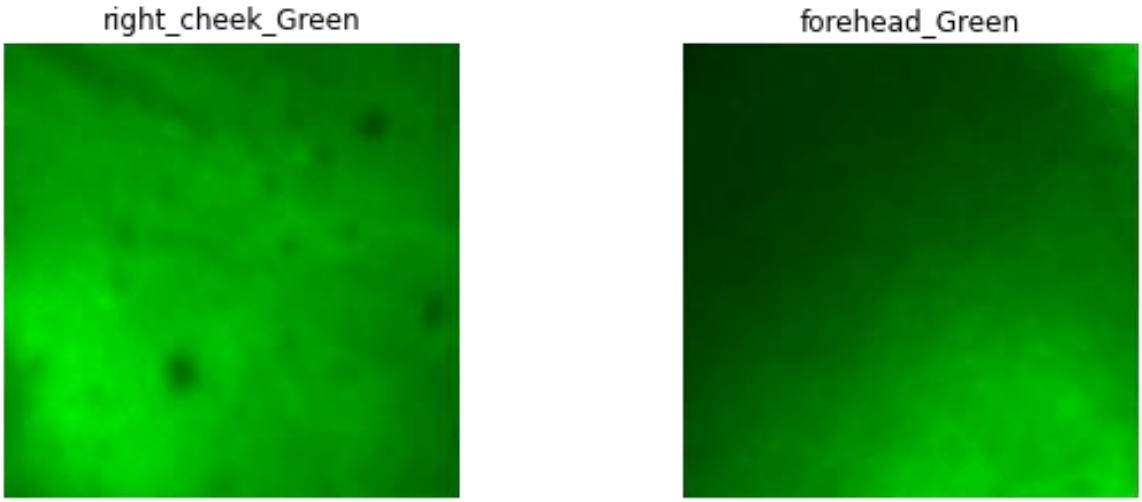

Figure 21. Alice's average green cheek and forehead images

3.3 Bandpass filter

It is often useful to filter a signal to a particular band of frequencies of interest ('pass band') if we know that other frequencies don't matter. In this application, we know that a normal resting heart rate for an adult ranges between 60-100 beats per minute (1-1.7 Hz). Apply the bandpass_filter function to your signals

provided here. You can set low_cutoff = 0.8, high_cutoff = 3, fs = 30, order = 1. Plot the filtered signals.

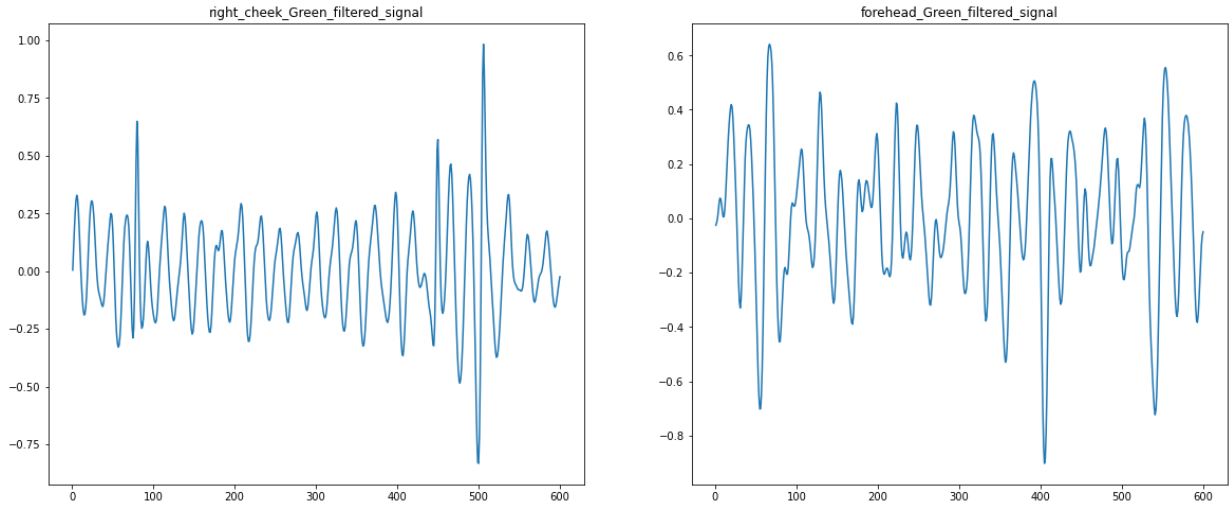

Figure 22. Alice's cheek and forehead bandpass filtered signals

3.4 Plot Fourier spectra

Plot the Fourier magnitudes of these two signals using the DFT, where the x-axis is frequency (in Hertz) and y-axis is amplitude. DFT coefficients are ordered in terms of integer indices, so you will have to convert the indices into Hertz. For each index n = [-N/2, N/2], the corresponding frequency is Fs * n / N, where N is the length of your signal and Fs is the sampling rate of the signal (30 Hz in this case). You can use numpy.fftfreq to do this conversion for you.

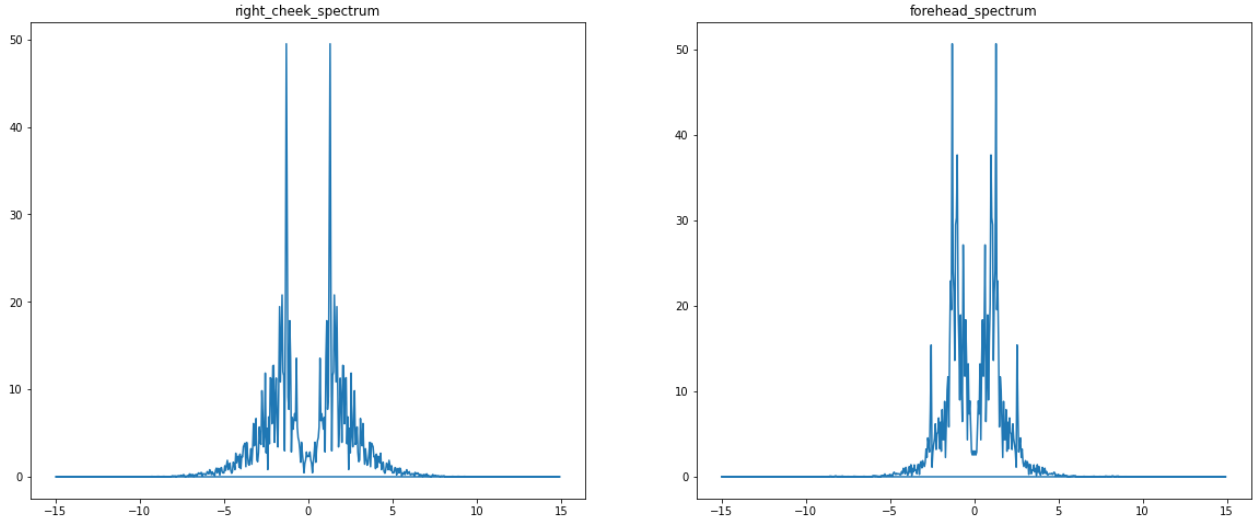

Figure 23. Alice's cheek and forehead Fourier spectrum signals

3.5 Estimate Alice's average pulse rate

A normal resting heart rate for adults ranges between 60-100 beats per minute.

```
if __name__ == '__main__':
    # Calculate the heart rate from the fourier_signals and frequency
    right_cheek_heart_rate = get_heart_rate(right_cheek_fourier, right_cheek_frequency)
    forehead_heart_rate = get_heart_rate(forehead_fourier, forehead_frequency)

    # Print the results of the heart rate calculation for the right cheek, and forehead
    print("right cheek max peak frequency:", right_cheek_heart_rate[0], "Hz = ", right_cheek_heart_rate[1], "BPM")
    print("forehead max peak frequency:", forehead_heart_rate[0], "Hz = ", forehead_heart_rate[1], "BPM")
```

```
right cheek max peak frequency: 1.3 Hz = 78.0 BPM
forehead max peak frequency: 1.3 Hz = 78.0 BPM
```

Figure 24. Alice's cheek and forehead frequency and heart rate

1. **What rate does the highest peak in Alice's Fourier spectrum correspond to?**

   As you can see from *Figure 24*, Alice's cheek has 1.3 Hz and equal to 78 BPM, then the forehead also have 1.3 Hz and equal to 78 BPM.

2. **Which facial region provides the cleanest spectrum (the one which has the clearest single peak and low energy elsewhere)?**

   According to *Figure 22* and *Figure 23*, the spectrum of the cheek is cleaner than the forehead.

3. **Is Alice likely a human or not?**

   Based on the results, I think Alice is a human, a heart rate of 78 BPM is within the typical range of a human heart rate (60 to 100 BPM).

3.6 Find your own pulse

Take a 15-20 second video of yourself using a smartphone, webcam, or personal camera. Your face should be as still as possible, and don't change facial expressions. Do a similar analysis above as you did with Alice's video. Show some frames from your video.

Figure 25. My first video frame

Figure 26. My left cheek, right cheek and forehead images

Figure 27. My average green left cheek, right cheek and forehead images

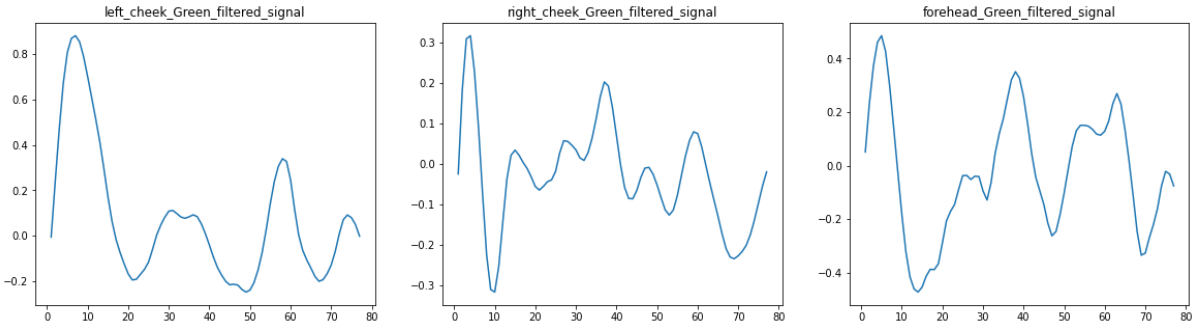

Figure 28. My left cheek, right cheek and forehead bandpass filtered signals

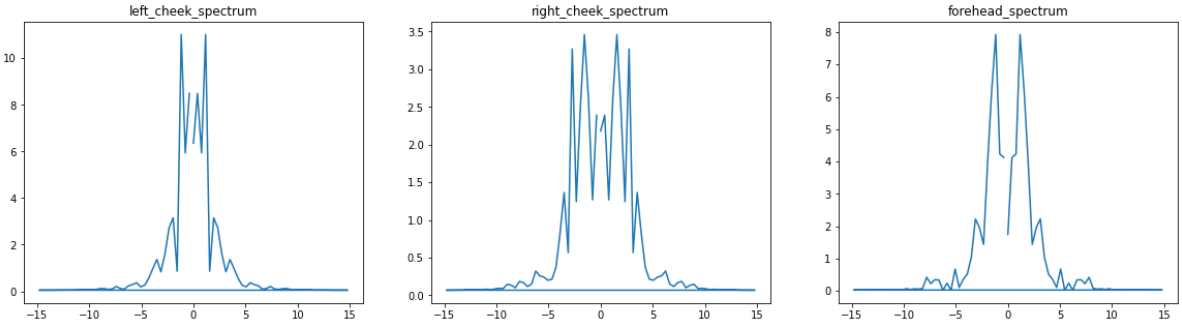

Figure 29. My left cheek, right cheek and forehead Fourier spectrum signals

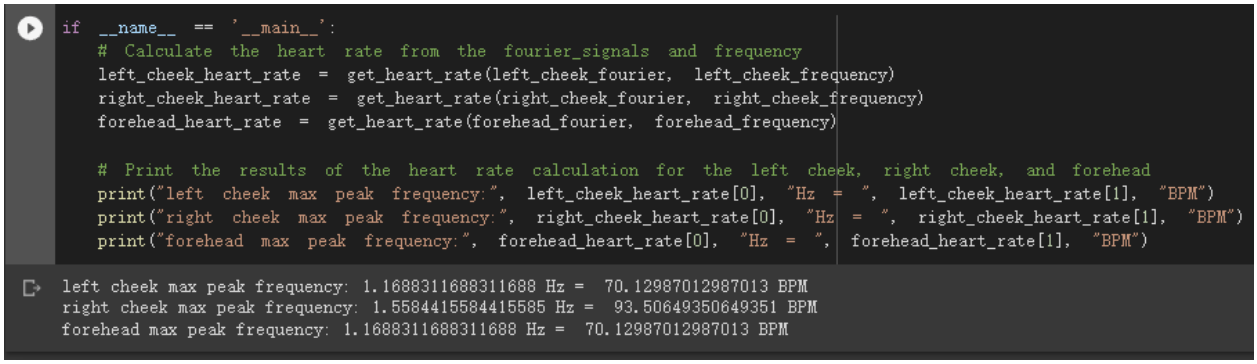

Figure 30. My left cheek, right cheek and forehead frequency and heart rate

1. **Was it easier/harder to estimate heart rate compared to the sample video we provided?**
   I think it's harder to predict my heart rate using the video compared to Alice's video. This might be because the lighting in my video was brighter, and this could have affected the results. As a result, you might notice that my heart rate as measured from my right cheek is different from the other measurements.
2. **What was challenging about it?**
   Overall, I think it's important to accurately identify the regions of the face where the pulse is most prominent, as this will affect the quality of the signal used to estimate the pulse rate. Additionally, even if the correct regions are selected, external factors such as lighting, movement, and noise can interfere with the signal and make it difficult to accurately extract the pulse rate.

References

[1] Oliva, Aude, Antonio Torralba, and Philippe G. Schyns. "Hybrid images." ACM Transactions on Graphics (TOG) 25.3 (2006): 527-532.
[2] Poh, Ming-Zher, Daniel J. McDuff, and Rosalind W. Picard. "Non-contact, automated cardiac pulse measurements using video imaging and blind source separation." Optics express 18.10 (2010): 10762-10774.