Azimuthal Averaging 과 High-pass filtering 을 이용한 Fake 이미지 디텍션

김민석
광주과학기술원 지스트대학 전기전자컴퓨터전공

wsms8646@gist.ac.kr

Fake Image Detection via Azimuthal Averaging and High-pass filtering

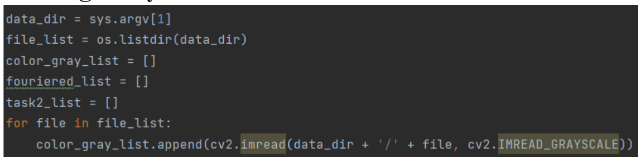
Minseok Kim
Gwangju Institute of Science and Technology (GIST)

요 약

With the recent development of image manipulation technologies such as DeepFake, demand for technologies that differentiate fake images is increasing to prepare for this. Therefore, in this report, we will find fake images through spatial frequency comparison using azimuthal averaging and finally distinguish fake images using a high-pass filtering method that can supplement the method using azimuthal averaging. For convenience, the above process was coded through Python.

Ⅰ. Source Code description

First, we read the data we will analyze on the gray_scale. Through this, the picture of the h*w pixel is represented as a two-dimensional matrix of the h*w size, not as a three-dimensional one of the h*w sizes, which greatly reduces future calculations.



```
data_dir = sys.argv[1]
file_list = os.listdir(data_dir)
color_gray_list = []
fouriered_list = []
task2_list = []
for file in file_list:
    color_gray_list.append(cv2.imread(data_dir + '/' + file, cv2.IMREAD_GRAYSCALE))
```

Picture 1. Load Image with grey_scale

$$F(u,v) = \frac{1}{MN} arr\_left \cdot Im \cdot arr\_right$$

$$arr_{left} = \sum_{v=0}^{N-1}\sum_{n=0}^{N-1} e^{-2\pi j \frac{nv}{N}}$$

$$arr_{right} = \sum_{u=0}^{M-1}\sum_{m=0}^{M-1} e^{-2\pi j \frac{mu}{M}}$$

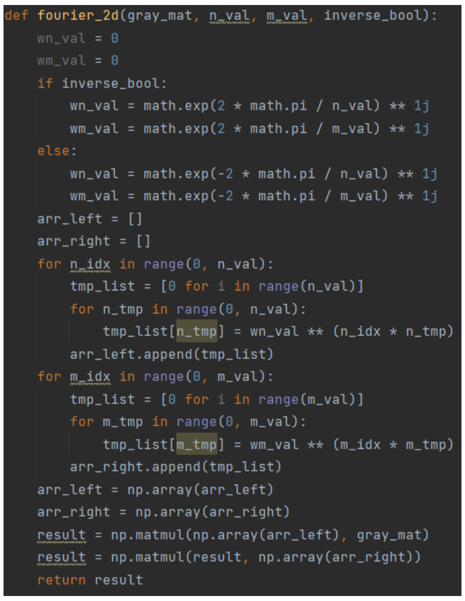
Equation 1. 2D Fourier Transform

$$Im(m,n) = arr\_left \cdot F \cdot arr\_right$$

$$arr_{left} = \sum_{v=0}^{N-1}\sum_{n=0}^{N-1} e^{2\pi j \frac{nv}{N}}$$

$$arr_{right} = \sum_{u=0}^{M-1}\sum_{m=0}^{M-1} e^{2\pi j \frac{mu}{M}}$$

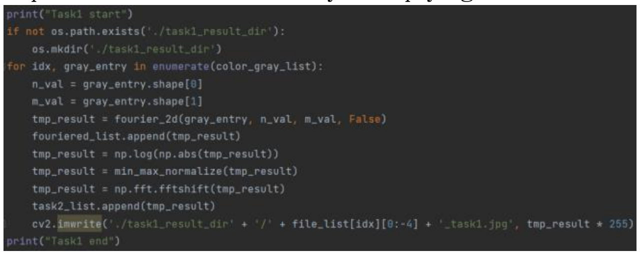
Equation 2. 2D Inverse Fourier Transform



```
def fourier_2d(gray_mat, n_val, m_val, inverse_bool):
    wn_val = 0
    wm_val = 0
    if inverse_bool:
        wn_val = math.exp(2 * math.pi / n_val) ** 1j
        wm_val = math.exp(2 * math.pi / m_val) ** 1j
    else:
        wn_val = math.exp(-2 * math.pi / n_val) ** 1j
        wm_val = math.exp(-2 * math.pi / m_val) ** 1j
    arr_left = []
    arr_right = []
    for n_idx in range(0, n_val):
        tmp_list = [0 for i in range(n_val)]
        for n_tmp in range(0, n_val):
            tmp_list[n_tmp] = wn_val ** (n_idx * n_tmp)
        arr_left.append(tmp_list)
    for m_idx in range(0, m_val):
        tmp_list = [0 for i in range(m_val)]
        for m_tmp in range(0, m_val):
            tmp_list[m_tmp] = wm_val ** (m_idx * m_tmp)
        arr_right.append(tmp_list)
    arr_left = np.array(arr_left)
    arr_right = np.array(arr_right)
    result = np.matmul(np.array(arr_left), gray_mat)
    result = np.matmul(result, np.array(arr_right))
    return result
```

Picture 2. Function for 2D (inverse) fourier transform

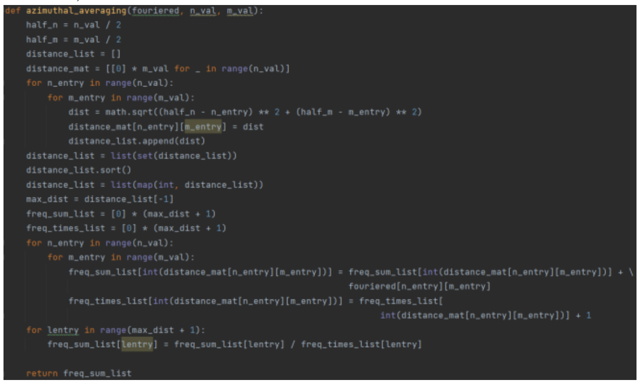
Let's summarize the two-dimensional Fourier transform formula. As shown in Equation 1 and 2, the Fourier transform formula is of the same form except for the 1/(M*N) multiplication and sign of exponent to power of natural log. Therefore, when creating fourier_2d, an inverse_bool is placed to determine the sign of exponent to power of natural log to make it Fourier Transform if the inverse_bool is false and Inverse Fourier Transform if true (1/(M*N) is applied outside the function for function code's simplicity). After making arr_left and arr_right, the two-dimensional Fourier (reverse) transformation was implemented as a function by multiplying the matrix.



```python
print("Task1 start")
if not os.path.exists('./task1_result_dir'):
    os.mkdir('./task1_result_dir')
for idx, gray_entry in enumerate(color_gray_list):
    n_val = gray_entry.shape[0]
    m_val = gray_entry.shape[1]
    tmp_result = fourier_2d(gray_entry, n_val, m_val, False)
    fouriered_list.append(tmp_result)
    tmp_result = np.log(np.abs(tmp_result))
    tmp_result = min_max_normalize(tmp_result)
    tmp_result = np.fft.fftshift(tmp_result)
    task2_list.append(tmp_result)
    cv2.imwrite('./task1_result_dir' + '/' + file_list[idx][0:-4] + '_task1.jpg', tmp_result * 255)
print("Task1 end")
```

Picture 3. Post Processing and save it for Task2, 3

After Fourier transform, complete Task1 by post-processing that it is high frequency centered shift of frequency domain (since these values are used in Task2, all of them are stored).

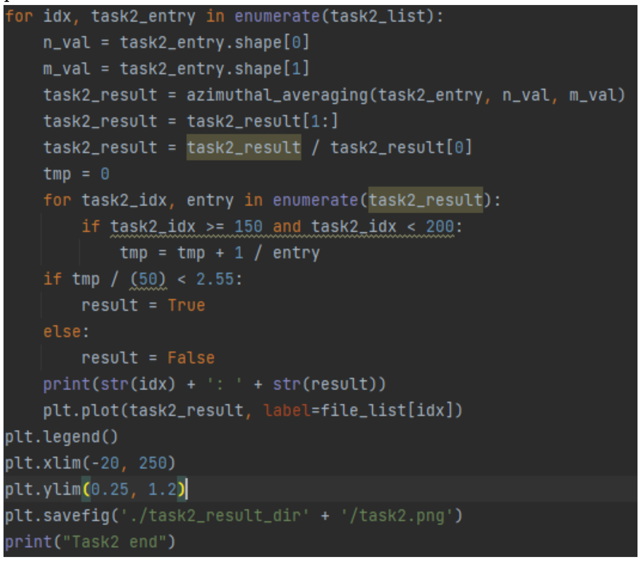


```python
def azimuthal_averaging(fouriered, n_val, m_val):
    half_n = n_val / 2
    half_m = m_val / 2
    distance_list = []
    distance_mat = [[0] * m_val for _ in range(n_val)]
    for n_entry in range(n_val):
        for m_entry in range(m_val):
            dist = math.sqrt((half_n - n_entry) ** 2 + (half_m - m_entry) ** 2)
            distance_mat[n_entry][m_entry] = dist
            distance_list.append(dist)
    distance_list = list(set(distance_list))
    distance_list.sort()
    distance_list = list(map(int, distance_list))
    max_dist = distance_list[-1]
    freq_sum_list = [0] * (max_dist + 1)
    freq_times_list = [0] * (max_dist + 1)
    for n_entry in range(n_val):
        for m_entry in range(m_val):
            freq_sum_list[int(distance_mat[n_entry][m_entry])] = freq_sum_list[int(distance_mat[n_entry][m_entry])] + \
                                                                 fouriered[n_entry][m_entry]
            freq_times_list[int(distance_mat[n_entry][m_entry])] = freq_times_list[
                                                                       int(distance_mat[n_entry][m_entry])] + 1
    for lentry in range(max_dist + 1):
        freq_sum_list[lentry] = freq_sum_list[lentry] / freq_times_list[lentry]

    return freq_sum_list
```

Picture 4. Azimuthal averaging with one circuit

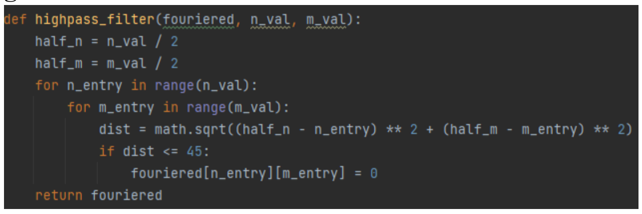
Azimuthal averaging aims to calculate the average frequency power value, which is the average of values for each specific section based on the distance from the center. However, if this is calculated purely, we must do a circuit of all point for every section. So, the function called azimuthal_averaging was implemented by obtaining the distance from the center for all points and storing it in distance_mat and obtaining all average frequency power values in one circuit of points.



```python
for idx, task2_entry in enumerate(task2_list):
    n_val = task2_entry.shape[0]
    m_val = task2_entry.shape[1]
    task2_result = azimuthal_averaging(task2_entry, n_val, m_val)
    task2_result = task2_result[1:]
    task2_result = task2_result / task2_result[0]
    tmp = 0
    for task2_idx, entry in enumerate(task2_result):
        if task2_idx >= 150 and task2_idx < 200:
            tmp = tmp + 1 / entry
    if tmp / (50) < 2.55:
        result = True
    else:
        result = False
    print(str(idx) + ': ' + str(result))
    plt.plot(task2_result, label=file_list[idx])
plt.legend()
plt.xlim(-20, 250)
plt.ylim(0.25, 1.2)
plt.savefig('./task2_result_dir' + '/task2.png')
print("Task2 end")
```

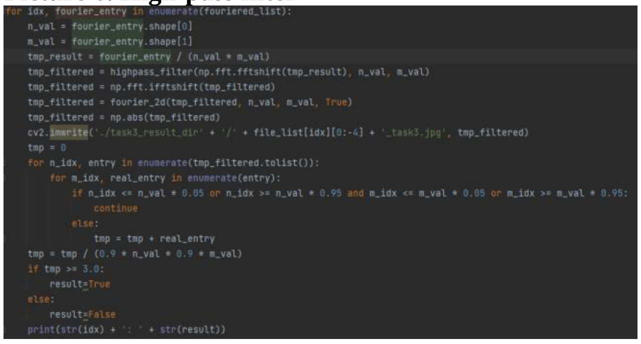
Picture 5. Task2 source code

Since the average frequency power value far from the origin is often low when we apply Azimuthal averaging to the Fake image with pre-processing of Task1, Fake image detection was performed through the average of the reciprocal of the average frequency power value more than a certain distance from the origin.



```python
def highpass_filter(fouriered, n_val, m_val):
    half_n = n_val / 2
    half_m = m_val / 2
    for n_entry in range(n_val):
        for m_entry in range(m_val):
            dist = math.sqrt((half_n - n_entry) ** 2 + (half_m - m_entry) ** 2)
            if dist <= 45:
                fouriered[n_entry][m_entry] = 0
    return fouriered
```

Picture 6. High-pass filter



```python
for idx, fourier_entry in enumerate(fouriered_list):
    n_val = fourier_entry.shape[0]
    m_val = fourier_entry.shape[1]
    tmp_result = fourier_entry / (n_val * m_val)
    tmp_filtered = highpass_filter(np.fft.fftshift(tmp_result), n_val, m_val)
    tmp_filtered = np.fft.ifftshift(tmp_filtered)
    tmp_filtered = fourier_2d(tmp_filtered, n_val, m_val, True)
    tmp_filtered = np.abs(tmp_filtered)
    cv2.imwrite('./task3_result_dir' + '/' + file_list[idx][0:-4] + '_task3.jpg', tmp_filtered)
    tmp = 0
    for n_idx, entry in enumerate(tmp_filtered.tolist()):
        for m_idx, real_entry in enumerate(entry):
            if n_idx <= n_val * 0.05 or n_idx >= n_val * 0.95 and m_idx <= m_val * 0.05 or m_idx >= m_val * 0.95:
                continue
            else:
                tmp = tmp + real_entry
    tmp = tmp / (0.9 * n_val * 0.9 * m_val)
    if tmp >= 3.0:
        result=True
    else:
        result=False
    print(str(idx) + ': ' + str(result))
```

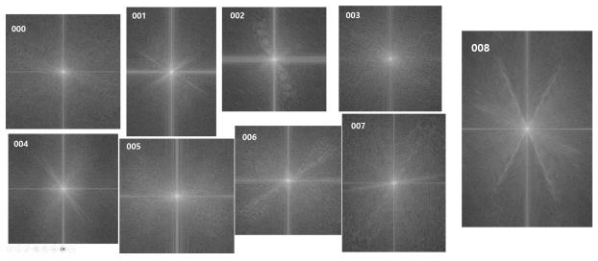
Picture 7. Task3 source code

The high-pass filter can be made to zero the value of the part where the distance is 45 or less at the center. When checking the photos obtained by inverse Fourier transform after applying the high-pass filter, there was a characteristic that appeared only in the photos suspected of fake. The photos suspected of fake were difficult to see some part which is suspicious of manipulation that because the value of the suspicious part was close to zero. Therefore, we averaged the values of the inverse Fourier transformed matrix and hypothesized that if it was below a certain value, the value of the suspicious part would be low to the average. And it means it is Fake image. Since then, the edges of all photos have bright values, so the average excluding the edges of the photos showed a clearer difference between the photos inferred by real image and the photos inferred by fake image.

Ⅱ. Result & Discussion

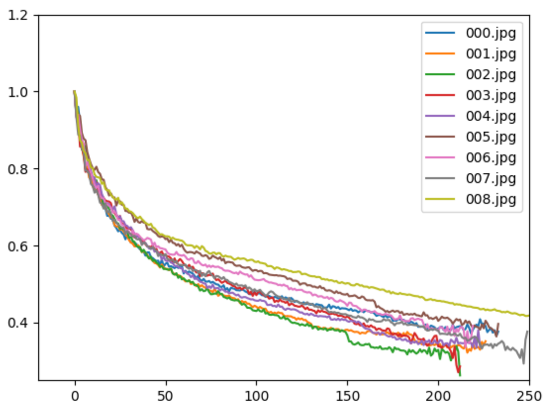
Picture 8. Additional dataset

In this report, to see if the experimental method is limited to example data, we added an image of me, and an image generated using stable diffusion as a dataset.



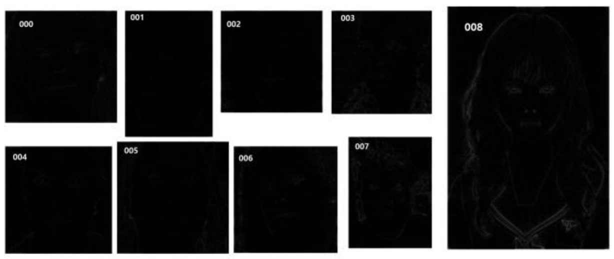
Picture 9. Result of Task1

Picture 9 is a picture of the Fourier transformed image through the source code of Task1.



Picture 10. Result of Task2

Picture 10 shows the results of azimuthal averaging for all photos with Task2. In the results of Task2, since Fake image detection is done through the tendency of parts somewhat away from the center, values between 150 and 200 were taken as reciprocal and averaged, and if the threshold was 2.55 or less, it was determined as a real picture.



Picture 11. Result of Task3

Picture 11 is the result of Inverse Fourier transform after passing high-pass filter for all photos with Task3. For the suspected fake image results, result matrix had low value for the face like eyes and mouth. So, fake image detection was performed by comparing average value to the threshold. However, since the values of the edges were high in common in all images, the average was obtained excluding the values of the edges, so that the comparison between the real image and the fake image became clearer.

Index	1D Power Spectrum	Filtering and 2D IFT	Conclusion
0	T	T	T
1	F	F	F
2	F	F	F
3	F	T	T
4	F	F	F
5	T	F	F
6	T	T	T
7(additional)	T	T	T
8(additional)	T	T	T

Picture 12. Final Fake Image Detection Result

The final decision was to follow the result of Task3. This is because though the result of Task2 shows some tendency but also has a limitation in that there is a fake type that is difficult to detect. As a result, it was determined that 001, 002, 004, and 005 were fake images. In the case of 007 and 008, which are additional data sets, 007 and 008 gave an additional conclusion. 007's results showed that the actual image could be determined as the actual image, but 008's results showed that method like Task2 and Task3 can't determine fake image generated by recent method like stable diffusion.