

영상신호처리 Homework #2

컴퓨터공학전공 2020451142 정수영

1. Get a grey level image which size is $N \times N$ and partition to 8×8 sub images.

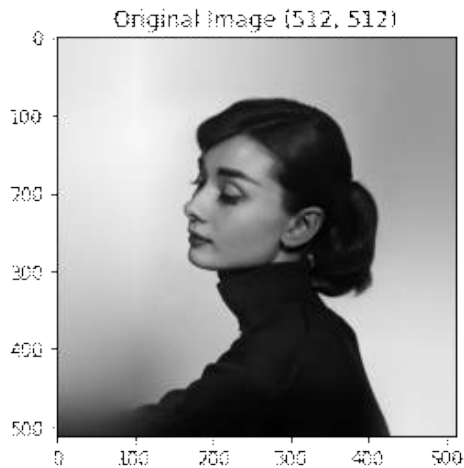
(1) 정사각형 이미지를 지정된 $N=512$ 값에 의해 흑백의 $N \times N$ 픽셀의 array로 변환하여 f에 저장한다.

Code

```
# import libraries
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from PIL import Image

# 1-(1). Get Gray level image; size N x N (512x512의 이미지 불러옴.)
N = 512
image = Image.open("hepburn.jpg").resize((N, N)).convert('L')
original_f = np.array(image)
plt.imshow(original_f, cmap="gray", vmin=0, vmax=255)
plt.title("Original Image " + str(original_f.shape))
plt.show() # 이미지 출력
```

Result



(2) $N \times N$ 이미지를 $K \times K$ 의 서브 이미지들로 바꾸어 주는 GetKxKs() 함수와 이 함수의 역인 GetNxN()을 정의한다. GetKxK() 함수를 이용하여 원본 f를 8×8 이미지들의 배열로 변환한다. 원본이 512×512 이므로 8×8 의 이미지로 분해하면 $64 \times 64 = 4096$ 개의 서브 이미지들의 배열로 변환된다. 이 이미지 배열을 f_s에 저장한다.

Code

```
# GetKxKs : NxN image to 8x8 (K=8) sub images
def GetKxKs(image, K):
    n = len(image)
    if n % K != 0: return
    ret = []
    for row in range(int(n/K)):
        for col in range(int(n/K)):
            ret.append(image[row*K:(row+1)*K, col*K:(col+1)*K])
    return np.array(ret)
```

```
# GetNxN : KxK(8x8) sub images to NxN image
def GetNxN(images):
    K = images.shape[1]
    N = int(np.sqrt(images.shape[0])) * K
    ret_image = np.empty((N, N), float)
    for i in range(N):
        for j in range(N):
            group = int(i/K) * int(N/K) + int(j/K)
            ret_image[i][j] = images[group][i%K][j%K]
    return np.array(ret_image)

# 1-(2). partition to 8*8 sub images
f_s = GetKxKs(original_f, 8)
print("8*8 서브 이미지들을 담은 f_s :", f_s.shape)
```

Result

8*8 서브 이미지들을 담은 f_s : (4096, 8, 8)

2. Apply DFT to these sub images, and get the fourier transformed image F .

- (1) 푸리에 공식(Fourier())을 이용하여 $N \times N$ DFT(푸리에변환)매트릭스를 생성하는 함수 $T()$ 를 정의하고, 역푸리에 공식(IFourier())을 이용하여 $N \times N$ IDFT(역푸리에변환)매트릭스를 생성하는 함수 $IT()$ 를 정의한다.
- (2) $T(8)$ 를 이용하여 f_s 이미지 f 들을 푸리에 변환하여 F 로 만든 후 이들을 F_s 배열에 저장한다.

$$F = T_8 f T_8^H \quad (\text{where } T_8 = T(8))$$

- (3) 푸리에 변환된 F_s 의 각 원소를 절댓값을 취한 후 $\text{GetNxN}()$ 함수를 이용해 512×512 배열로 다시 변환 후 푸리에 이미지를 출력한다.

$$F_s \text{의 } 8 \times 8 \text{ 이미지 } F \text{의 임의의 원소가 } a + bj \text{ 일 때, } |a + bj| = \sqrt{a^2 + b^2} \quad (\text{단, } j^2 = -1)$$

Code

```
# Fourier Formular
def Fourier(N, x, a):
    return complex(sp.cos((2. * sp.pi * x * a)/N), -1.*sp.sin((2. * sp.pi * x * a)/N))/np.sqrt(N)

# Inverse Fourier Formular
def IFourier(N, x, a):
    return complex(sp.cos((2. * sp.pi * x * a)/N), +1.*sp.sin((2. * sp.pi * x * a)/N))/np.sqrt(N)

# 2-(1). Define DFT function T() and IDFT function IT()
# T : DFT (Discrete Fourier Transform)
def T(N):
    U = []
    for i in range(N):
        row = []
        for j in range(N):
            row.append(Fourier(N, i, j))
        U.append(row)
    return np.array(U)

# IT : IDFT (Inverse Discrete Fourier Transform)
def IT(N):
    U = []
    for i in range(N):
        row = []
        for j in range(N):
```

```

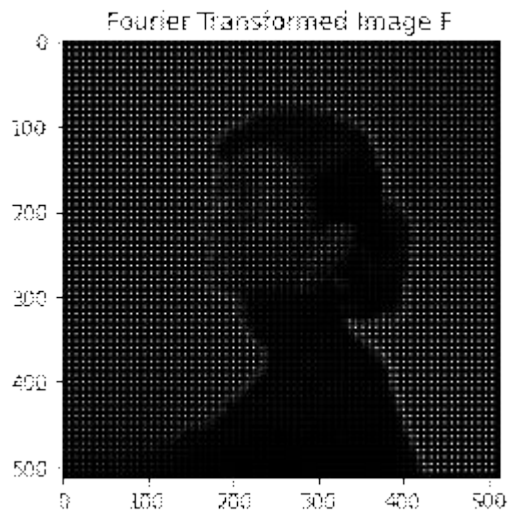
        row.append(IFourier(N, i, j))
    U.append(row)
    return np.array(U)

# 2-(2) Derived Fs images Applying DFT to these sub image f_s.
# Fs : The 8*8 image F list in Frequency Domain
Fs = []
T8 = T(8)
for f in f_s:
    Fs.append(np.dot(T8, np.dot(f, np.matrix.getH(T8))))
Fs = np.array(Fs)

# 2-(3) Get the fourier transformed image F
plt.imshow(abs(GetNxN(Fs)), cmap='gray', vmin=0, vmax=255 )
plt.title("Fourier Transformed Image F")
plt.show()

```

Result



3. Derive the proper subsampling function matrix S in spatial domain.

- (1) 주어진 공식에 따라 $n \times n$ 매트릭스를 $\frac{n}{2} \times \frac{n}{2}$ 매트릭스로 변환시켜줄 함수 $S()$ 를 정의한다.
- (2) 정의된 함수 $S()$ 를 이용해 8*8을 4*4로 변환시켜줄 매트릭스 $S = S(8)$ 을 구한다.

$$S = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Code

```

# 3-(1). S : Subsampling matrix n*n -> n/2*n/2 in Spatial Domain
def S(n):
    ret = []
    for i in range(int(n/2)):
        row = []
        for j in range(n):

```

```

        if i*2 == j or (i*2 +1) == j:
            row.append(0.5)
        else: row.append(0)
    ret.append(row)
    return np.array(ret)

```

```

# 3-(2). Derive the proper subsampling function matrix S in spatial domain
S = S(8)
print("S =", S)

```

Result

```

S = [[0.5 0.5 0. 0. 0. 0. 0. 0. ]
      [0. 0. 0.5 0.5 0. 0. 0. 0. ]
      [0. 0. 0. 0. 0.5 0.5 0. 0. ]
      [0. 0. 0. 0. 0. 0. 0.5 0.5]]

```

4. Multiplying proper 4*4 DFT matrix T to matrix S , derive the frequency version of subsampling function matrix, S_F .

(1) $S_F = T_4 S T_8^H = SF$ (where $T_4 = T(4)$, $T_8 = T(8)$)

Code

```

# 4-(1). Derive the frequency version of subsampling function matrix SF.
# SF = T(4) * S * T(8)*
SF = np.dot(T(4), np.dot(S, np.matrix.getH(T(8))))
print("SF =", SF)

```

Result

```

SF = [[0.70710678+0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j ]
      [0. +0.j 0.60355339+0.25j 0. +0.j 0. +0.j 0. +0.j 0.10355339-0.25j 0. +0.j 0. +0.j ]
      [0. +0.j 0. +0.j 0.35355339+0.35355339j 0. +0.j 0. +0.j 0. +0.j 0.35355339-0.35355339j 0. +0.j ]
      [0. +0.j 0. +0.j 0. +0.j 0.10355339+0.25j 0. +0.j 0. +0.j 0. +0.j 0.60355339-0.25j ]]

```

5. Similarly, derive the frequency version of interpolation matrix : I

(1) 먼저 Spatial Domain에서의 Interpolation matrix I 를 구한다.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(2) $I_F = T_8 I T_4^H = IF$ (where $T_4 = T(4)$, $T_8 = T(8)$)

Code

```

# 5-(1). I : Interpolation matrix in Spatial Domain, shape : 8 x 4
I = np.array([
    [1, 0, 0, 0 ],
    [0.5, 0.5, 0, 0 ],
    [0, 1, 0, 0 ],

```

```

        [0, 0.5, 0.5, 0 ],
        [0, 0, 1, 0 ],
        [0, 0, 0.5, 0.5],
        [0, 0, 0, 1 ],
        [0, 0, 0, 1 ]
    ])

```

```

# 5-(2). Derive the interpolation matrix in frequency domain IF.
# IF : Interpolation matrix in Frequency Domain
# IF = T(8) x I x T(4)*
IF = np.dot(T(8), np.dot(I, np.matrix.getH(T(4))))
print("IF =", IF)

```

Result

```

IF = [[ 1.41421356+0.j -0.08838835-0.08838835j -0.1767767 +0.j -0.08838835+0.08838835j]
 [ 0. +0.j 1.20710678-0.125j -0.125 -0.125j -0.125 +0.j ]
 [ 0. +0.j 0.08838835-0.08838835j 0.70710678-0.1767767j -0.08838835-0.08838835j]
 [ 0. +0.j 0.125 +0.j 0.125 -0.125j 0.20710678-0.125j ]
 [ 0. +0.j 0.08838835+0.08838835j 0.1767767 +0.j 0.08838835-0.08838835j]
 [ 0. +0.j 0.20710678+0.125j 0.125 +0.125j 0.125 +0.j ]
 [ 0. +0.j -0.08838835+0.08838835j 0.70710678+0.1767767j 0.08838835+0.08838835j]
 [ 0. +0.j -0.125 +0.j -0.125 +0.125j 1.20710678+0.125j ]]

```

6. Apply these matrices to each sub image of F , and get the final image, taking inverse DFT.

- (1) $F_S = S_F F S_F^H$ (F_S : Subsampling된 4*4 푸리에 이미지) F_S 를 모은 배열 SubFs를 구한다.
- (2) F_S 를 이미지로 출력한다.

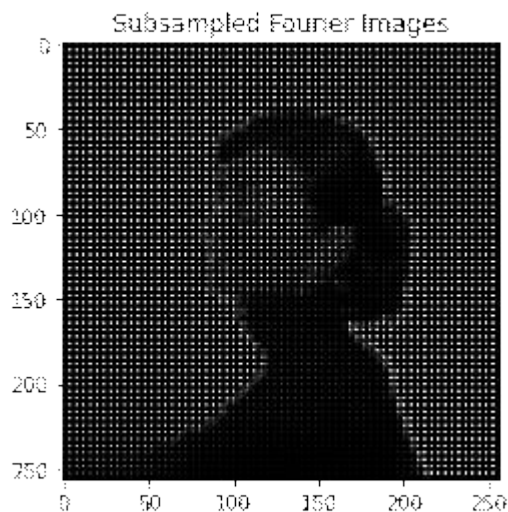
Code

```

# 6-(1). Subsampling F's
SubFs = []
for F in Fs:
    SubFs.append(np.dot(SF, np.dot(F, np.matrix.getH(SF))))
SubFs = np.array(SubFs)
# 6-(2) Subsampling 된 푸리에 이미지를 출력한다.
plt.imshow(abs(GetNxN(SubFs)), cmap='gray', vmin=0, vmax=255 )
plt.title("Subsampled Fourier Images")
plt.show()

```

Result



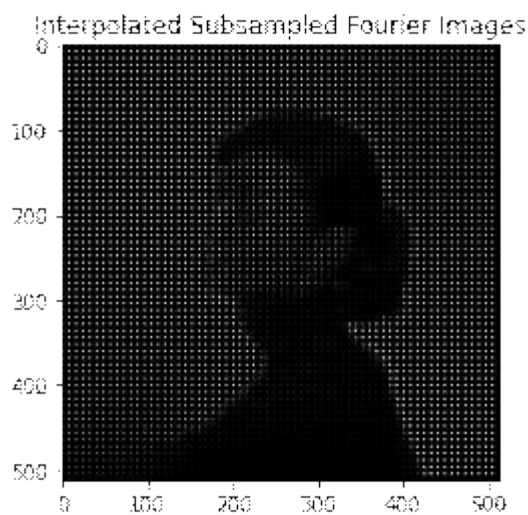
- (3) $F_{SI} = I_F F_S I_F^H$, (F_{SI} : F_S 를 Interpolating 시킨 푸리에 이미지) F_{SI} 들을 모은 배열이 InterSubFs
 (4) F_{SI} 이미지를 출력한다.

Code

```
# 6-(3). Interpolating Subsampled F's
InterSubFs = []
for SubF in SubFs:
    InterSubFs.append(np.dot(IF, np.dot(SubF, np.matrix.getH(IF))))
InterSubFs = np.array(InterSubFs)

# 6-(4) Subsampling -> Interpolating된 푸리에 이미지 출력
plt.imshow(abs(GetNxN(InterSubFs)), cmap="gray", vmin=0, vmax=255 )
plt.title("Interpolated Subsampled Fourier Images")
plt.show()
```

Result



- (5) F_{SI} 의 역푸리에 구해서 $IDFT(F_{SI})$ 를 InvInterSubFs 배열에 저장한다.

$$f' = IDFT(F_{SI}) = IF \times F_{SI} \times IF^H$$

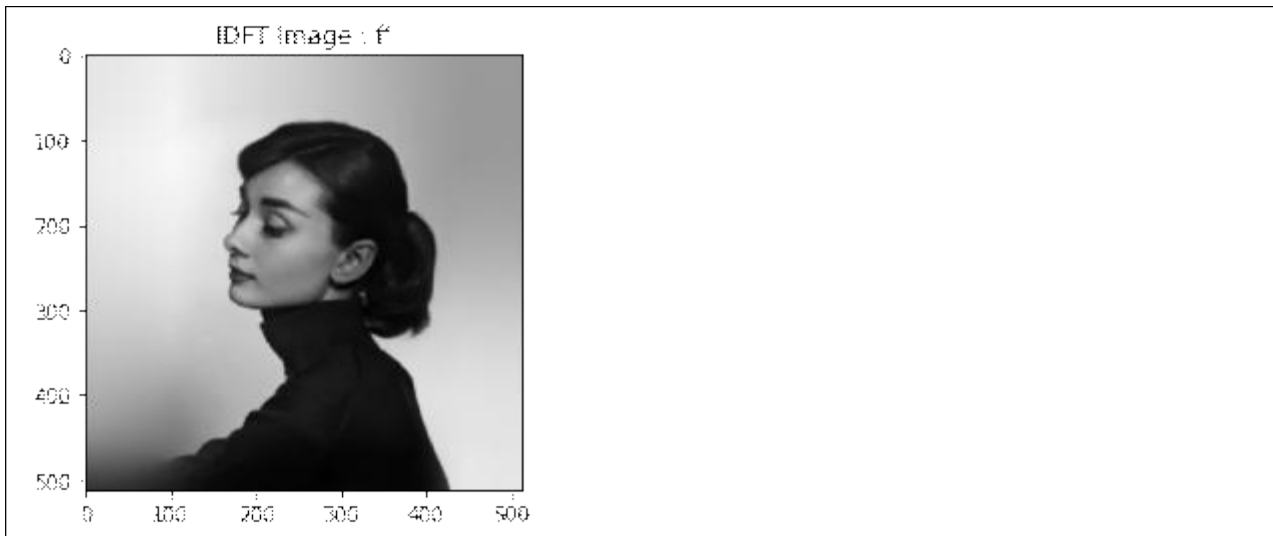
- (6) f' 이미지를 출력한다.

Code

```
# 6-(5). IDFT image F's which is interpolated and subsampled
InvInterSubFs = []
IT8 = IT(8)
for isF in InterSubFs:
    InvInterSubFs.append(np.dot(IT8, np.dot(isF, np.matrix.getH(IT8))))
InvInterSubFs = np.array(InvInterSubFs)

# 6-(6). Show IDFT Image
f_processed_frequency = abs(GetNxN(InvInterSubFs))
plt.imshow(f_processed_frequency, cmap='gray', vmin=0, vmax=255 )
plt.title("IDFT Image : f'")
plt.show()
```

Result



7. Apply subsampling/interpolation function matrix S/I to original image in spatial domain and get the final image.

(1) 임의의 8×8 원본 이미지 f 에 대하여, Subsampling \rightarrow Interpolating한 이미지를 f'' 이라 하면,

$$f'' = I(S_8 f S_8^H) I^H$$

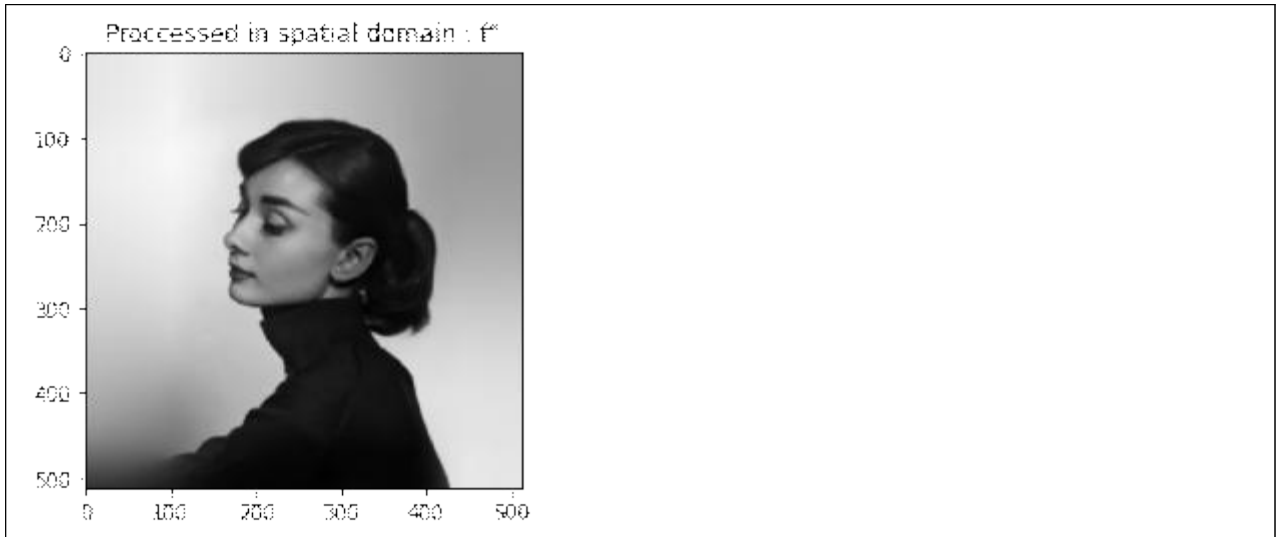
(2) f'' 이미지를 출력

Code

```
# 7-(1). Apply S and I to original image in Spatial domain
subfs = []
for f in f_s:
    subfs.append(np.dot(S, np.dot(f, np.transpose(S))))
subfs = np.array(subfs)
intersubfs = []
for sf in subfs:
    intersubfs.append(np.dot(I, np.dot(sf, np.transpose(I))))
intersubfs = np.array(intersubfs)

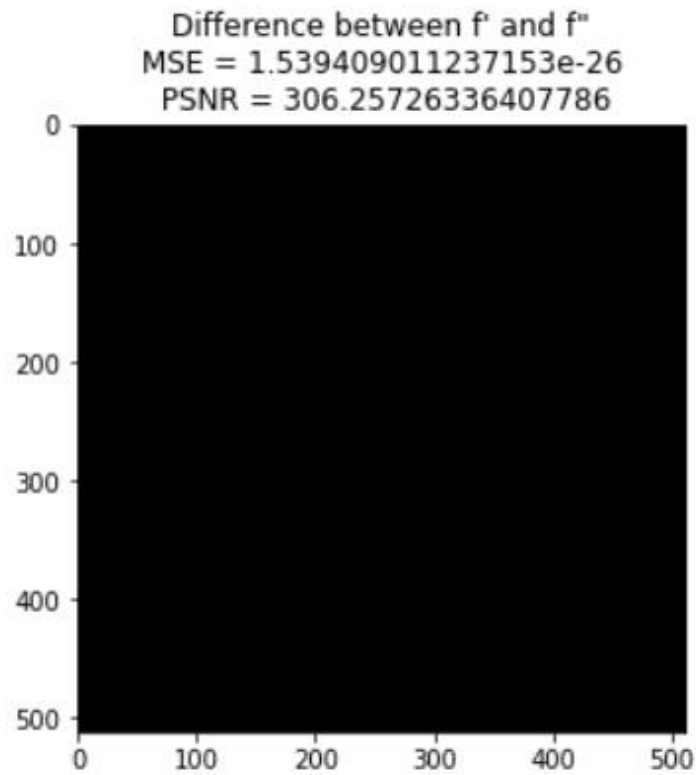
# 7-(2). Show the image proccessed in spatial domain
f_proccessed_spatial = GetNxN(intersubfs)
plt.imshow(f_proccessed_spatial, cmap='gray', vmin=0, vmax=255 )
plt.title("Proccessed in spatial domain : f'")
plt.show()
```

Result



[비교]

- (1) 6번의 f' (Frequency Domain에서 변환)과 7번 f'' (Spatial Domain에서 변환) 이미지를 비교해보면 $MSE = 1.54 \times 10^{-26}$ 으로 거의 오차가 0에 가까움.



실제, 이론 상으로는 f' 과 f'' 의 오차는 없어야 함.

$$\begin{aligned}
 DEF(f') &= I_F(S_F F S_F^H) I_F^H \\
 &= (T_8 I T_4^H)(T_4 S T_8^H)(T_8 f T_8^H)(T_4 S T_8^H)^H (T_8 I T_4^H)^H \\
 &= (T_8 I T_4^H)(T_4 S T_8^H)(T_8 f T_8^H)(T_8 S^H T_4^H)(T_4 I^H T_8^H) \\
 &= T_8 I (T_4^H T_4) S (T_8^H T_8) f (T_8^H T_8) S^H (T_4^H T_4) I^H T_8^H \\
 &= T_8 \times I \times S \times f \times S^H \times I^H \times T_8^H
 \end{aligned}$$

$$T_8 f' T_8^H = T_8 \times I \times S \times f \times S^H \times I^H \times T_8^H$$

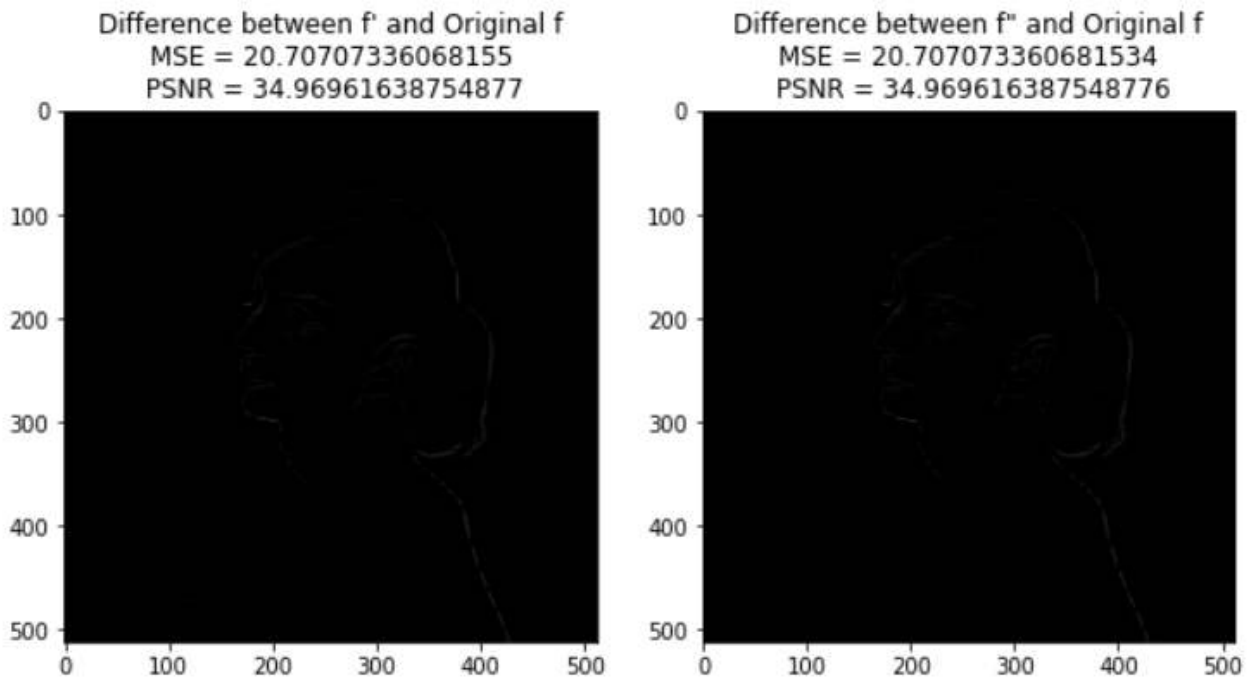
$$f' = T_8^H (T_8 \times I \times S \times f \times S^H \times I^H \times T_8^H) T_8$$

$$f' = (T_8^H \times T_8) \times (I \times S \times f \times S^H \times I^H) \times (T_8^H \times T_8)$$

$$= I \times S \times f \times S^H \times I^H = f''$$

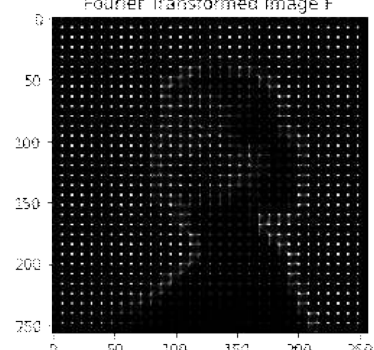
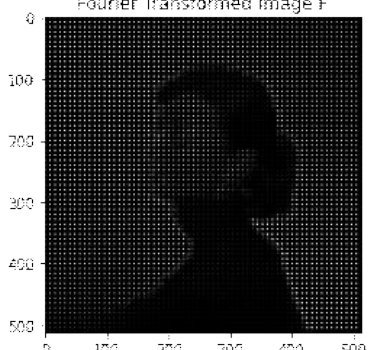
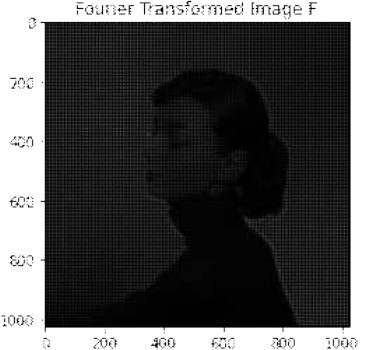
$$\therefore f' = f''$$

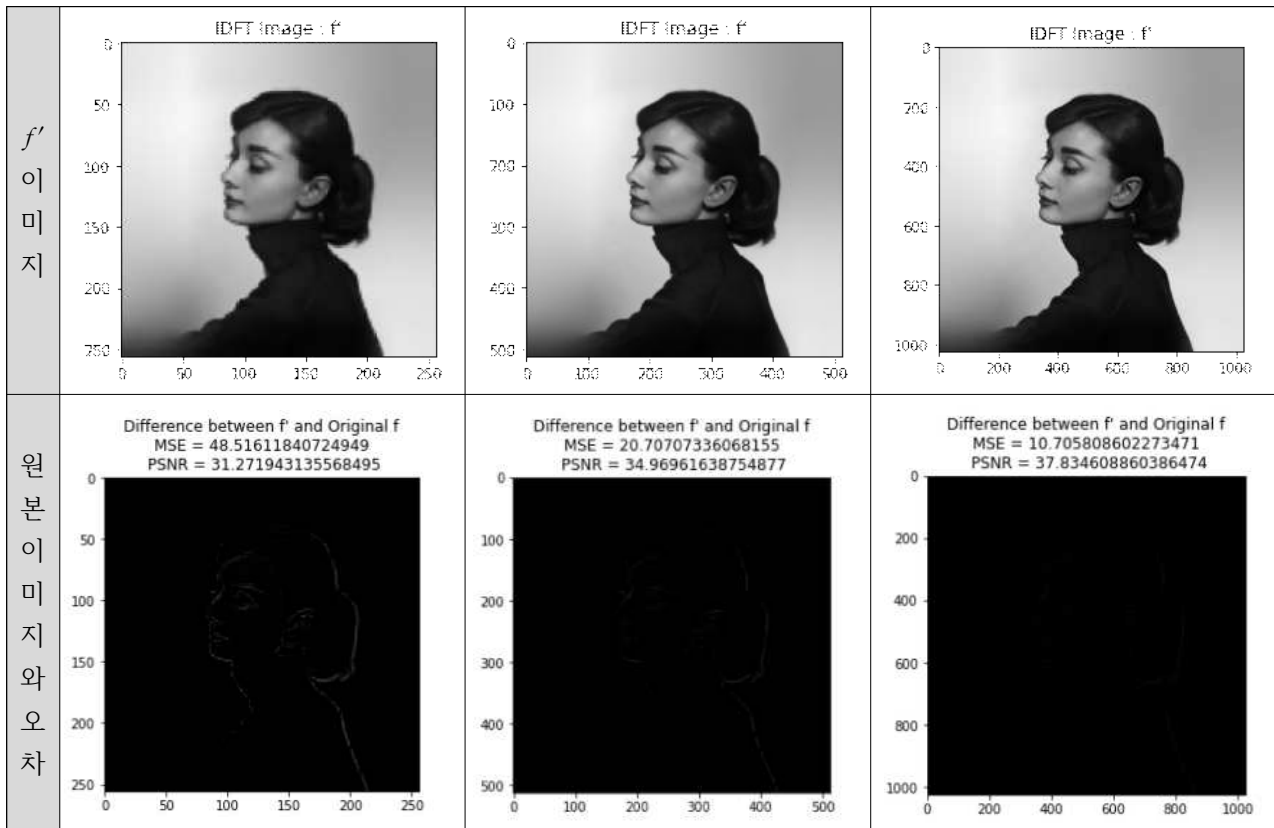
(2) $f' - f$, $f'' - f$ 를 구해보면 서로 비슷한 수준의 에러를 보임.



(5) 원본 이미지의 크기 $N \times N$ 에서 N 의 값에 따른 f' 비교

N 이 클수록 변환에 따른 원본과의 오차가 작아지는 경향이 있음. 또한 원본과의 오차는 대부분 윤곽선에서 생기는 경향이 있음.

	$N = 256$	$N = 512$	$N = 1024$
D F T 이 미 지	<p>Fourier Transformed Image F</p> 	<p>Fourier Transformed Image F</p> 	<p>Fourier Transformed Image F</p> 



Code

```
# Compare with  $f'$ (processed in frequency domain) and  $f''$ (processed in spatial domain)
f_diff1 = f_processed_frequency - f_processed_spatial
mse1 = np.square(f_diff1).mean(axis=None)
psnr1 = 20 * np.log10(255/np.sqrt(mse1))
error_msg1 = "MSE = {0}\n PSNR = {1}".format(mse1, psnr1)
plt.figure(figsize=(15,15))
plt.subplot(1, 3, 1)
plt.imshow(f_diff1, cmap= "gray", vmin=0, vmax=255 )
plt.title("Difference between  $f'$  and  $f''$ \n"+ error_msg1)

# Compare with  $f'$  and Original Image  $f$ 
f_diff2 = f_processed_frequency - original_f
mse2 = np.square(f_diff2).mean(axis=None)
psnr2 = 20 * np.log10(255/np.sqrt(mse2))
error_msg2 = "MSE = {0}\n PSNR = {1}".format(mse2, psnr2)
plt.subplot(1, 3, 2)
plt.imshow(f_diff2, cmap= "gray", vmin=0, vmax=255 )
plt.title("Difference between  $f'$  and Original  $f$ \n"+error_msg2)

# Compare with  $f''$  and Original Image  $f$ 
f_diff3 = f_processed_spatial - original_f
mse3 = np.square(f_diff3).mean(axis=None)
psnr3 = 20 * np.log10(255/np.sqrt(mse3))
error_msg3 = "MSE = {0}\n PSNR = {1}".format(mse3, psnr3)
plt.subplot(1, 3, 3)
```

```
plt.imshow(f_diff3, cmap= "gray", vmin=0, vmax=255 )
plt.title("Difference between f\" and Original f\\n" + error_msg3)
plt.show()
```

Result

