N20DCCN031

Nguyễn Nhật Kha

D20CQCNPM01-N

**Homework 5**

**Part 1a**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import cv2

import numpy as np

import os

import matplotlib.pyplot as plt

def linear\_average\_filter(image):

    padded\_image = cv2.copyMakeBorder(image, 3, 3, 3, 3, cv2.BORDER\_CONSTANT, value=0)

    result = np.zeros((262, 262))

    # Convolution

    for row in range(3, 259):

        for col in range(3, 259):

            result[row, col] = np.sum(padded\_image[row-3:row+4, col-3:col+4]) / 49

    # Drop padding

    result = result[3:259, 3:259]

    return result

def create\_full\_contrast(input):

    min\_val = np.min(input)

    max\_val = np.max(input)

    if min\_val == max\_val:

        return np.zeros\_like(input, dtype=np.uint8)

    scale = 255/(max\_val - min\_val)

    normalized = np.round(((input - min\_val) \* scale))

    return normalized

# Load image

parent\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

image = np.fromfile(f"{parent\_directory}/salesman.bin", dtype=np.uint8).reshape(256,256)

# Apply linear average filter

filtered\_image = linear\_average\_filter(image)

# Show images

plt.figure(figsize=(8,6))

plt.subplot(1,2,1)

plt.imshow(create\_full\_contrast(image),cmap="gray")

plt.axis("off")

plt.title("Original")

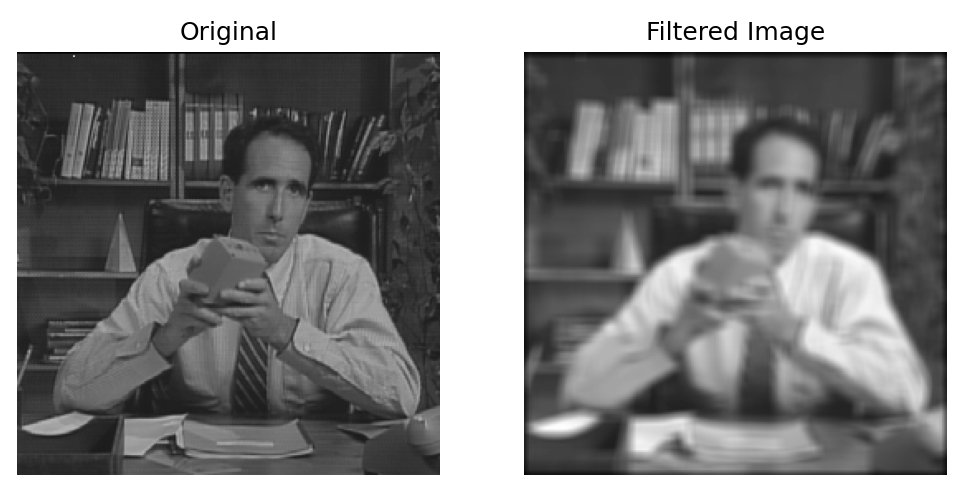
plt.subplot(1,2,2)

plt.imshow(create\_full\_contrast(filtered\_image),cmap="gray")

plt.axis("off")

plt.title("Filtered Image")

plt.show()

****

**Part 1b**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import cv2

import numpy as np

import os

from matplotlib import pyplot as plt

def create\_full\_contrast(input):

    min\_val = np.min(input)

    max\_val = np.max(input)

    if(min\_val == max\_val):

        return np.zeros(input.shape, dtype=np.uint8)

    scale = 255/(max\_val - min\_val)

    return np.round(((input - min\_val) \* scale))

# Load image

parent\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

original\_image = np.fromfile(f"{parent\_directory}/salesman.bin", dtype=np.uint8).reshape(256,256)

# Create impulse response H

H = np.zeros((128, 128), dtype=np.float32)

H[62:69, 62:69] = 1/49

# Zero-pad the original image

padded\_original = cv2.copyMakeBorder(original\_image, 0, 128, 0, 128, cv2.BORDER\_CONSTANT, value=0)

# Zero-pad the impulse response H

zero\_padded\_H = cv2.copyMakeBorder(H, 0, 256, 0, 256, cv2.BORDER\_CONSTANT, value=0)

# Compute the DFT

dft\_original = np.fft.fft2(padded\_original)

dft\_H = np.fft.fft2(zero\_padded\_H, s=dft\_original.shape)

# Compute the centered log-manitude

centered\_original = np.log(1 + np.abs(dft\_original))

centered\_H = np.log(1 + np.abs(dft\_H))

# Compute convolution by the pointwise multiplication

convo\_output =  dft\_original \* dft\_H

# Compute the IDFT of the result

padded\_output = np.abs(np.fft.ifft2(convo\_output))

# Crop padding

final\_output = padded\_output[65:256+65, 65:256+65]

plt.figure(figsize=(10,8))

plt.subplot(2,4,1)

plt.imshow(create\_full\_contrast(original\_image), cmap="gray")

plt.axis("off")

plt.title("Original")

plt.subplot(2,4,2)

plt.imshow(create\_full\_contrast(padded\_original), cmap="gray")

plt.axis("off")

plt.title("Zero Padded Image")

plt.subplot(2,4,3)

plt.imshow(create\_full\_contrast(zero\_padded\_H), cmap="gray")

plt.axis("off")

plt.title("Zero Padded Impulse Resp")

plt.subplot(2,4,4)

plt.imshow(np.fft.fftshift(centered\_original), cmap="gray")

plt.axis("off")

plt.title("Log spectrum of padded original")

plt.subplot(2,4,5)

plt.imshow(np.fft.fftshift(centered\_H), cmap="gray")

plt.axis("off")

plt.title("Log spectrum of padded H")

plt.subplot(2,4,6)

plt.imshow(np.fft.fftshift(np.log(1 + np.abs(convo\_output))), cmap="gray")

plt.axis("off")

plt.title("Log spectrum of padded result")

plt.subplot(2,4,7)

plt.imshow(create\_full\_contrast(padded\_output), cmap="gray")

plt.axis("off")

plt.title("Zero Padded Result")

plt.subplot(2,4,8)

plt.imshow(create\_full\_contrast(final\_output), cmap="gray")

plt.axis("off")

plt.title("Final Filtered Image")

plt.show()

def linear\_average\_filter(image):

    padded\_image = cv2.copyMakeBorder(image, 3, 3, 3, 3, cv2.BORDER\_CONSTANT, value=0)

    result = np.zeros((262, 262))

    # Convolution

    for row in range(3, 259):

        for col in range(3, 259):

            result[row, col] = np.sum(padded\_image[row-3:row+4, col-3:col+4]) / 49

    # Drop padding

    result = result[3:259, 3:259]

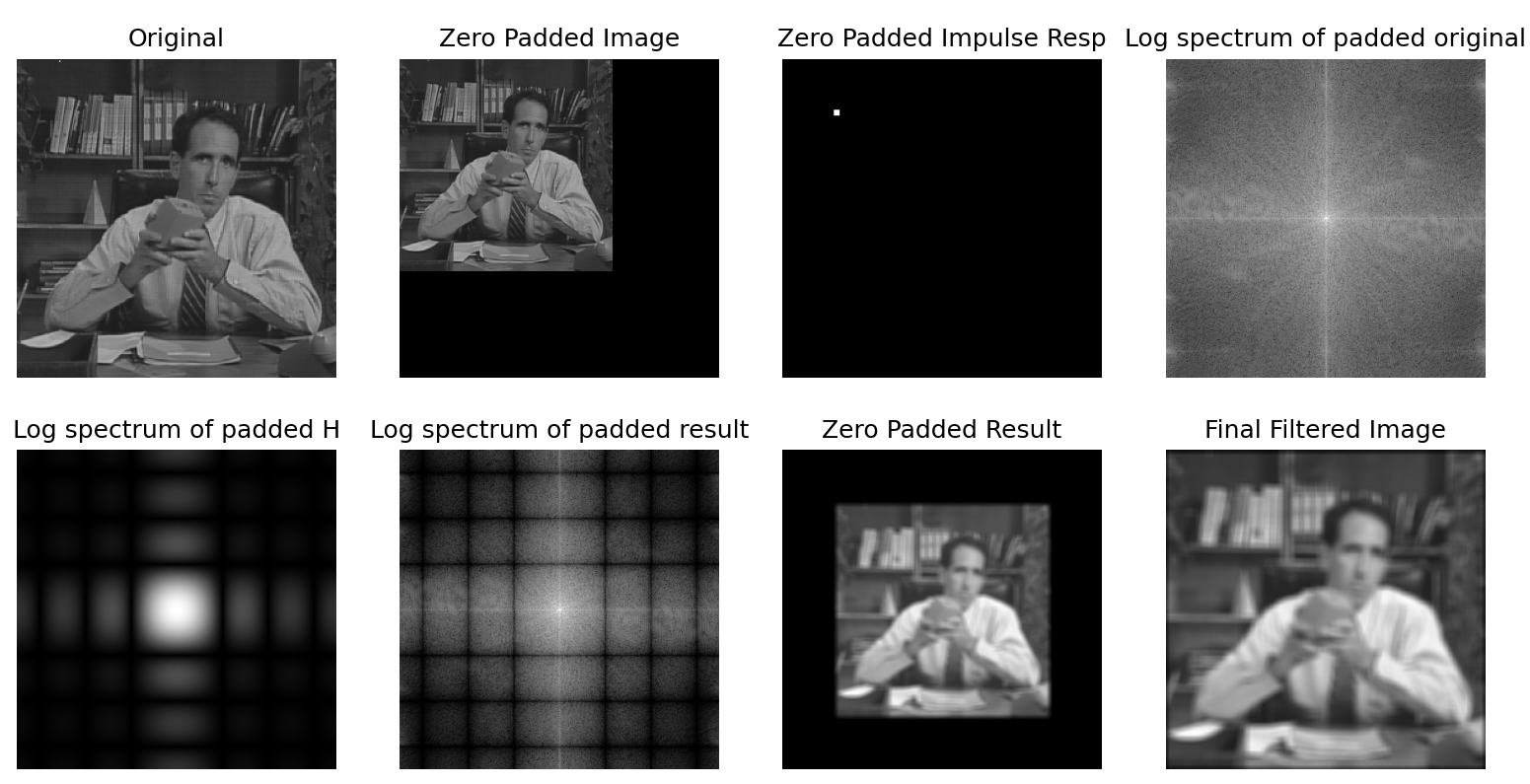
    return result

# Get result of part a

A = linear\_average\_filter(original\_image)

print("Max difference from part a:",end=' ')

print(np.max(np.abs(create\_full\_contrast(final\_output) - create\_full\_contrast(A))))

****

****

**Part 1c**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import numpy as np

import os

import matplotlib.pyplot as plt

import cv2

def create\_full\_contrast(input):

    min\_val = np.min(input)

    max\_val = np.max(input)

    if(min\_val == max\_val):

        return np.zeros(input.shape, dtype=np.uint8)

    scale = 255/(max\_val - min\_val)

    return np.round(((input - min\_val) \* scale))

# Load image

parent\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

original\_image = np.fromfile(f"{parent\_directory}/salesman.bin", dtype=np.uint8).reshape(256,256)

# Create the zero-padded original image

padded\_original = np.zeros((512, 512))

padded\_original[:256,:256] = original\_image

# Make the 256x256 impulse response image H

H = np.zeros((256, 256), dtype=np.float32)

H[125:132, 125:132] = 1/49

# Get the true zero-phase impulse response image

H2 = np.fft.fftshift(H)

# Create the zero-padded impulse response image

padded\_H = np.zeros((512, 512))

padded\_H[:128, :128] = H2[:128, :128]

padded\_H[:128, 384:512] = H2[:128, 128:256]

padded\_H[384:512, :128] = H2[128:256, :128]

padded\_H[384:512, 384:512] = H2[128:256, 128:256]

# Compute the filtered result by pointwise multiplication of DFT's

final\_output = np.fft.ifft2(np.fft.fft2(padded\_original) \* np.fft.fft2(padded\_H))

# Drop padding and take the real part

final\_output = final\_output[:256, :256].real

plt.figure(figsize=(10,8))

plt.subplot(2,2,1)

plt.title("Original Image")

plt.imshow(original\_image, cmap="gray")

plt.axis("off")

plt.subplot(2,2,2)

plt.title("256x256 H")

plt.imshow(H2, cmap='gray')

plt.axis('off')

plt.subplot(2,2,3)

plt.title("512x512 zero-padded H")

plt.imshow(padded\_H, cmap='gray')

plt.axis('off')

plt.subplot(2,2,4)

plt.imshow(final\_output, cmap='gray')

plt.title('Final Filtered Image')

plt.axis('off')

plt.show()

def linear\_average\_filter(image):

    padded\_image = cv2.copyMakeBorder(image, 3, 3, 3, 3, cv2.BORDER\_CONSTANT, value=0)

    result = np.zeros((262, 262))

    # Convolution

    for row in range(3, 259):

        for col in range(3, 259):

            result[row, col] = np.sum(padded\_image[row-3:row+4, col-3:col+4]) / 49

    # Drop padding

    result = result[3:259, 3:259]

    return result

# Get result of part a

A = linear\_average\_filter(original\_image)

print("Max difference from part a:",end=' ')

print(np.max(np.abs(create\_full\_contrast(final\_output) - create\_full\_contrast(A))))

****

****

**Part 2a**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import numpy as np

import os

import matplotlib.pyplot as plt

# Load image

def get\_file(name):

    current\_file\_path = os.path.abspath(\_\_file\_\_)

    parent\_directory = os.path.dirname(current\_file\_path)

    return np.fromfile(f"{parent\_directory}/{name}", dtype=np.uint8)

girl = get\_file("girl2.bin").reshape(256,256)

girl\_noise = get\_file("girl2Noise32.bin").reshape(256,256)

girl\_hi\_noise = get\_file("girl2Noise32Hi.bin").reshape(256,256)

# Compute MSE

MSE\_hi\_noise = np.mean((girl\_hi\_noise - girl) \*\* 2)

print(f'MSE of girl2Noise32Hi: {MSE\_hi\_noise}')

MSE\_noise = np.mean((girl\_noise - girl) \*\* 2)

print(f'MSE of girl2Noise32: {MSE\_noise}')

plt.figure(figsize=(10,8))

plt.subplot(1,3,1)

plt.title("Original")

plt.imshow(girl, cmap="gray")

plt.axis("off")

plt.subplot(1,3,2)

plt.title("Girl2Noise32")

plt.imshow(girl\_noise, cmap="gray")

plt.axis("off")

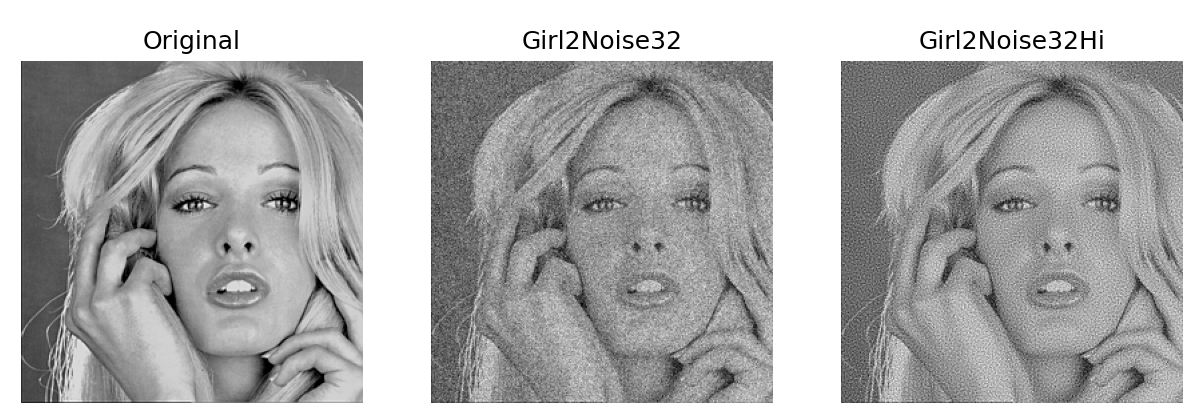
plt.subplot(1,3,3)

plt.title("Girl2Noise32Hi")

plt.imshow(girl\_hi\_noise, cmap="gray")

plt.axis("off")

plt.show()

****

****

**Part 2b**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import numpy as np

import os

import matplotlib.pyplot as plt

# Load image

def get\_file(name):

    current\_file\_path = os.path.abspath(\_\_file\_\_)

    parent\_directory = os.path.dirname(current\_file\_path)

    return np.fromfile(f"{parent\_directory}/{name}", dtype=np.uint8)

original = get\_file("girl2.bin").reshape(256,256)

Noise = get\_file("girl2Noise32.bin").reshape(256,256)

HiNoise = get\_file("girl2Noise32Hi.bin").reshape(256,256)

U\_cutoff = 64

U, V = np.meshgrid(np.arange(-128, 128), np.arange(-128, 128))

HLtildeCenter = np.double(np.sqrt(U\*\*2 + V\*\*2) <= U\_cutoff)

HLtilde = np.fft.fftshift(HLtildeCenter)

# Apply ideal low-pass filter to the original image

LPF\_original = np.fft.ifft2(np.fft.fft2(original) \* HLtilde).real

MSE\_LPF\_original = np.mean((LPF\_original - original)\*\*2)

print(f'MSE: ideal LPF on Original: {MSE\_LPF\_original}')

# Hi Pass Noise Image: Apply filter, compute MSE, ISNR

LPF\_HiNoise = np.fft.ifft2(np.fft.fft2(HiNoise) \* HLtilde).real

MSE\_LPF\_HiNoise = np.mean((LPF\_HiNoise - original)\*\*2)

ISNR\_LPF\_HiNoise = 10 \* np.log10(np.mean((HiNoise - original)\*\*2) / MSE\_LPF\_HiNoise)

print(f'MSE: ideal LPF on Noise32Hi: {MSE\_LPF\_HiNoise}')

print(f'ISNR: ideal LPF on Noise32Hi: {ISNR\_LPF\_HiNoise} dB')

# Noise Image: Apply filter, compute MSE, ISNR

LPF\_Noise = np.fft.ifft2(np.fft.fft2(Noise) \* HLtilde).real

MSE\_LPF\_Noise = np.mean((LPF\_Noise - original)\*\*2)

ISNR\_LPF\_Noise = 10 \* np.log10(np.mean((Noise - original)\*\*2) / MSE\_LPF\_Noise)

print(f'MSE: ideal LPF on Noise32: {MSE\_LPF\_Noise}')

print(f'ISNR: ideal LPF on Noise32: {ISNR\_LPF\_Noise} dB')

def create\_full\_contrast(input):

    min\_val = np.min(input)

    max\_val = np.max(input)

    if(min\_val == max\_val):

        return np.zeros(input.shape, dtype=np.uint8)

    scale = 255/(max\_val - min\_val)

    return np.round(((input - min\_val) \* scale))

# Display images

plt.figure(figsize=(10,8))

plt.subplot(1,3,1)

plt.title('LPF on Original')

plt.imshow(create\_full\_contrast(LPF\_original), cmap='gray')

plt.axis('off')

plt.subplot(1,3,2)

plt.title('LPF on Girl2Noise32Hi')

plt.imshow(create\_full\_contrast(LPF\_HiNoise), cmap='gray')

plt.axis('off')

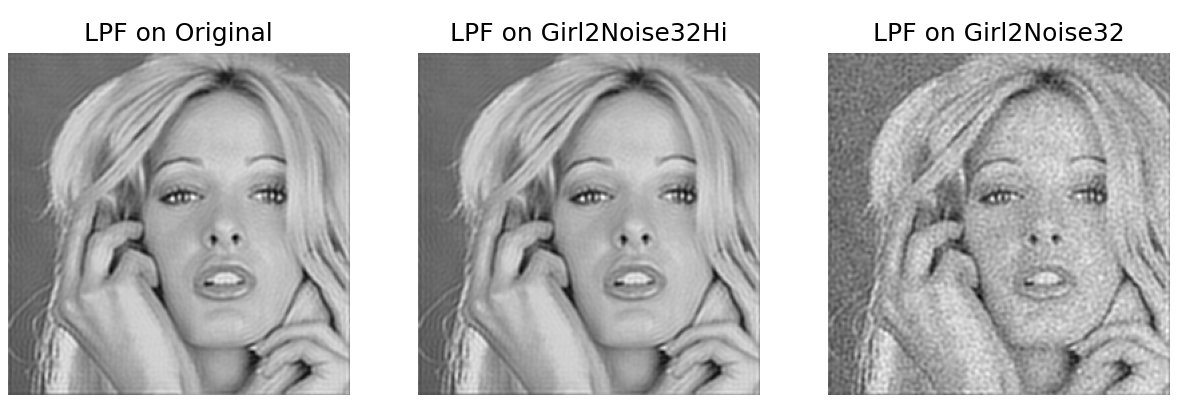
plt.subplot(1,3,3)

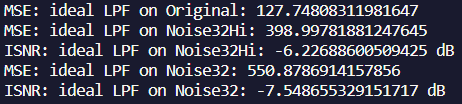
plt.title('LPF on Girl2Noise32')

plt.imshow(create\_full\_contrast(LPF\_Noise), cmap='gray')

plt.axis('off')

plt.show()

****

****

**Part 2c**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import numpy as np

import os

import matplotlib.pyplot as plt

# Load image

def get\_file(name):

    current\_file\_path = os.path.abspath(\_\_file\_\_)

    parent\_directory = os.path.dirname(current\_file\_path)

    return np.fromfile(f"{parent\_directory}/{name}", dtype=np.uint8)

original = get\_file("girl2.bin").reshape(256,256)

Noise = get\_file("girl2Noise32.bin").reshape(256,256)

HiNoise = get\_file("girl2Noise32Hi.bin").reshape(256,256)

U\_cutoff\_H = 64

SigmaH = 0.19 \* 256 / U\_cutoff\_H

U, V = np.meshgrid(np.arange(-128, 128), np.arange(-128, 128))

HtildeCenter = np.exp((-2 \* np.pi\*\*2 \* SigmaH\*\*2) / (256\*\*2) \* (U\*\*2 + V\*\*2))

Htilde = np.fft.fftshift(HtildeCenter)

H = np.fft.ifft2(Htilde).real

H2 = np.fft.fftshift(H)

ZPH2 = np.zeros((512, 512))

ZPH2[:256, :256] = H2

# Original: Apply Gaussian low-pass filter and compute MSE

ZP\_original = np.zeros((512, 512))

ZP\_original[:256, :256] = original

LPF\_original = np.fft.ifft2(np.fft.fft2(ZP\_original) \* np.fft.fft2(ZPH2)).real

LPF\_original = LPF\_original[128:384, 128:384]

MSE\_LPF\_original = np.mean((LPF\_original - original)\*\*2)

print(f'MSE: Gaussian LPF on Original: {MSE\_LPF\_original}')

# Hi Pass Noise: Apply Gaussian low-pass filter, compute MSE and ISNR

ZP\_HiNoise = np.zeros((512, 512))

ZP\_HiNoise[:256, :256] = HiNoise

LPF\_HiNoise = np.fft.ifft2(np.fft.fft2(ZP\_HiNoise) \* np.fft.fft2(ZPH2)).real

LPF\_HiNoise = LPF\_HiNoise[128:384, 128:384]

MSE\_LPF\_HiNoise = np.mean((LPF\_HiNoise - original)\*\*2)

print(f'MSE: Gaussian LPF on Noise32Hi: {MSE\_LPF\_HiNoise}')

ISNR\_LPF\_HiNoise = 10 \* np.log10(np.mean((HiNoise - original)\*\*2) / MSE\_LPF\_HiNoise)

print(f'ISNR: Gaussian LPF on Noise32Hi: {ISNR\_LPF\_HiNoise} dB')

# Noise: Apply Gaussian low-pass filter, compute MSE and ISNR

ZP\_Noise = np.zeros((512, 512))

ZP\_Noise[:256, :256] = Noise

LPF\_Noise = np.fft.ifft2(np.fft.fft2(ZP\_Noise) \* np.fft.fft2(ZPH2)).real

LPF\_Noise = LPF\_Noise[128:384, 128:384]

MSE\_LPF\_Noise = np.mean((LPF\_Noise - original)\*\*2)

print(f'MSE: Gaussian LPF on Noise32: {MSE\_LPF\_Noise}')

ISNR\_LPF\_Noise = 10 \* np.log10(np.mean((Noise - original)\*\*2) / MSE\_LPF\_Noise)

print(f'ISNR: Gaussian LPF on Noise32: {ISNR\_LPF\_Noise} dB')

# Display images

plt.figure(figsize=(8,10))

plt.subplot(1,3,1)

plt.title('Gauss on Original')

plt.imshow(LPF\_original, cmap='gray')

plt.axis('off')

plt.subplot(1,3,2)

plt.title('Gauss on Noise32Hi')

plt.imshow(LPF\_HiNoise, cmap='gray')

plt.axis('off')

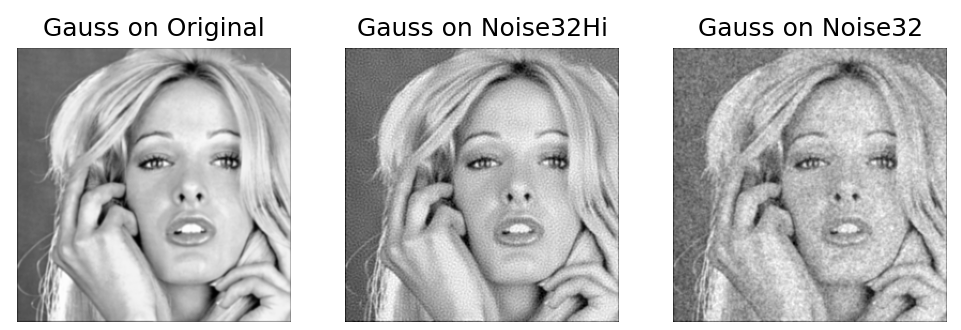
plt.subplot(1,3,3)

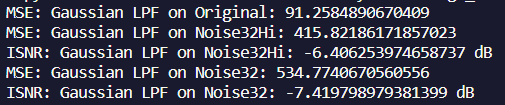
plt.title('Gauss on Noise32')

plt.imshow(LPF\_Noise, cmap='gray')

plt.axis('off')

plt.show()

****

****

**Part 2d**

# MSSV: N20DCCN031

# Nguyễn Nhật Kha

# Lớp: D20DCCNPM01-N

import numpy as np

import os

import matplotlib.pyplot as plt

# Load image

def get\_file(name):

    current\_file\_path = os.path.abspath(\_\_file\_\_)

    parent\_directory = os.path.dirname(current\_file\_path)

    return np.fromfile(f"{parent\_directory}/{name}", dtype=np.uint8)

original = get\_file("girl2.bin").reshape(256,256)

Noise = get\_file("girl2Noise32.bin").reshape(256,256)

HiNoise = get\_file("girl2Noise32Hi.bin").reshape(256,256)

U\_cutoff\_H = 77.5

SigmaH = 0.19 \* 256 / U\_cutoff\_H

U, V = np.meshgrid(np.arange(-128, 128), np.arange(-128, 128))

HtildeCenter = np.exp((-2 \* np.pi\*\*2 \* SigmaH\*\*2) / (256\*\*2) \* (U\*\*2 + V\*\*2))

Htilde = np.fft.fftshift(HtildeCenter)

H = np.fft.ifft2(Htilde).real

H2 = np.fft.fftshift(H)

ZPH2 = np.zeros((512, 512))

ZPH2[:256, :256] = H2

# Original: Apply Gaussian low-pass filter and compute MSE

ZP\_original = np.zeros((512, 512))

ZP\_original[:256, :256] = original

LPF\_original = np.fft.ifft2(np.fft.fft2(ZP\_original) \* np.fft.fft2(ZPH2)).real

LPF\_original = LPF\_original[128:384, 128:384]

MSE\_LPF\_original = np.mean((LPF\_original - original)\*\*2)

print(f'MSE: Gaussian LPF on Original: {MSE\_LPF\_original}')

# Hi Pass Noise: Apply Gaussian low-pass filter, compute MSE and ISNR

ZP\_HiNoise = np.zeros((512, 512))

ZP\_HiNoise[:256, :256] = HiNoise

LPF\_HiNoise = np.fft.ifft2(np.fft.fft2(ZP\_HiNoise) \* np.fft.fft2(ZPH2)).real

LPF\_HiNoise = LPF\_HiNoise[128:384, 128:384]

MSE\_LPF\_HiNoise = np.mean((LPF\_HiNoise - original)\*\*2)

print(f'MSE: Gaussian LPF on Noise32Hi: {MSE\_LPF\_HiNoise}')

ISNR\_LPF\_HiNoise = 10 \* np.log10(np.mean((HiNoise - original)\*\*2) / MSE\_LPF\_HiNoise)

print(f'ISNR: Gaussian LPF on Noise32Hi: {ISNR\_LPF\_HiNoise} dB')

# Noise: Apply Gaussian low-pass filter, compute MSE and ISNR

ZP\_Noise = np.zeros((512, 512))

ZP\_Noise[:256, :256] = Noise

LPF\_Noise = np.fft.ifft2(np.fft.fft2(ZP\_Noise) \* np.fft.fft2(ZPH2)).real

LPF\_Noise = LPF\_Noise[128:384, 128:384]

MSE\_LPF\_Noise = np.mean((LPF\_Noise - original)\*\*2)

print(f'MSE: Gaussian LPF on Noise32: {MSE\_LPF\_Noise}')

ISNR\_LPF\_Noise = 10 \* np.log10(np.mean((Noise - original)\*\*2) / MSE\_LPF\_Noise)

print(f'ISNR: Gaussian LPF on Noise32: {ISNR\_LPF\_Noise} dB')

# Display images

plt.figure(figsize=(8,10))

plt.subplot(1,3,1)

plt.title('Gauss on Original')

plt.imshow(LPF\_original, cmap='gray')

plt.axis('off')

plt.subplot(1,3,2)

plt.title('Gauss on Noise32Hi')

plt.imshow(LPF\_HiNoise, cmap='gray')

plt.axis('off')

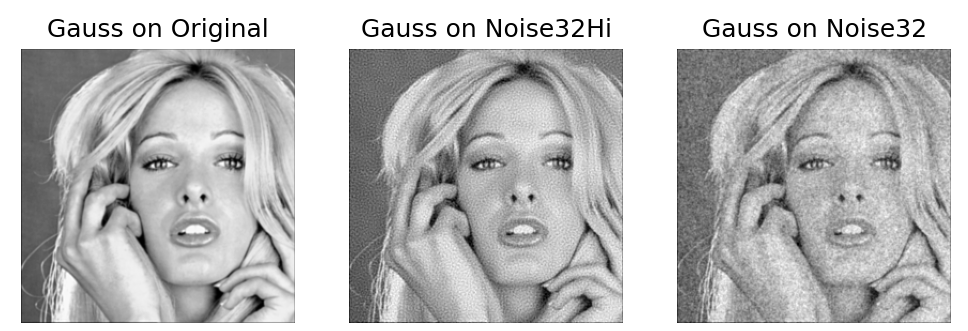
plt.subplot(1,3,3)

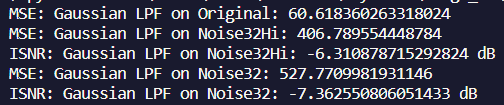
plt.title('Gauss on Noise32')

plt.imshow(LPF\_Noise, cmap='gray')

plt.axis('off')

plt.show()

****

****