ex2

May 8, 2019

0.1 Exercise 2

0.1.1 2.a) Refractor the PNSR definition such that the PSNR is expressed as a function of the noise variance $_z ^2 $. You may assume that $\frac{2}{z} = MSE(x, y)$

- PSNR original definition

  (1) $PSNR = 10 \log_{10} \frac{a^2}{MSE(x,y)}$

  (2) $MSE(x, y) = \sigma_z^2$

  (3) (2) $PSNR = 10 \log_{10} \frac{a^2}{\sigma_z^2}$

  $PSNR = 20 \log_{10} a - 10 \log_{10} \sigma_z^2$

  $PSNR - 20 \log_{10} a = 10 \log_{10} \sigma_z^2$

  $\frac{PSNR - 20 \log_{10} a}{10} = \log_{10} \sigma_z^2$

  (3) $10^{\frac{PSNR - 20 \log_{10} a}{10}} = \sigma_z^2$

  This relationship above is used in 2.b.

0.1.2 2.b) Add Gaussian noise to an image such that the PSNR ratio with the original image is 10dB, 20dB, 30dB and 40dB. Use randn, not imnoise.

- Noise function: $Z_i = N(\mu, \sigma^2)$

[23]:
```
#2b
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

def showTifGrayScale(img, title = ""):
    plt.imshow(img, cmap = 'gray')
    plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
    plt.title(title)
    plt.show()

PSNRs = [10, 20, 30, 40]
```

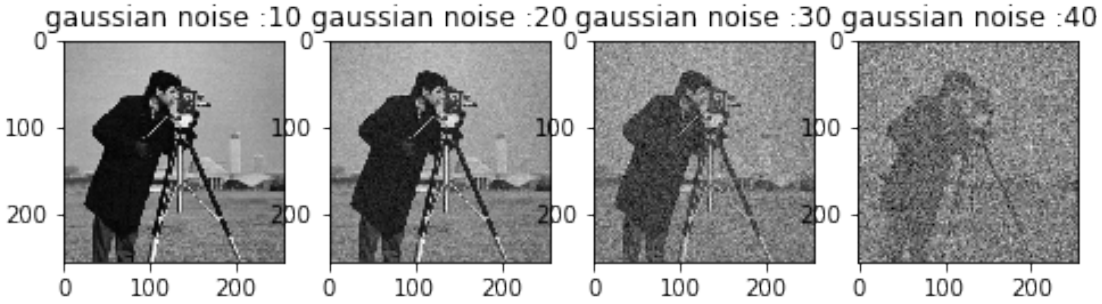
```
getVarianceForPSNR = lambda db: 10**((db-20 * math.log(255, 10))/10)
varianceList = [(db, getVarianceForPSNR(db)) for db in PSNRs ]

imgCam = cv2.imread('./data/cameraman.tif', -1)
imgCam_double = np.array(imgCam).astype("float32")
imgCam_double = np.interp(
    imgCam_double,
    (imgCam_double.min(), imgCam_double.max()),
    (0, 1)
)

showTifGrayScale(imgCam, "original")
```

0.1.3 2.c) Show the noisy images on the screen. How do they look?

They look bad the biggest the variance of the gaussian noise distribution.

[20]:
```
def experiment(db, variance, img):
    noisedImgArray, gaussNoiseMatrix = gaussianNoise(img, variance)
    return noisedImgArray

noised_images = [
    ("gaussian noise :" + str(db), experiment(db,var, imgCam_double ))
    for db,var in iter(varianceList)
]
```

```
def showImages(images, titles):
    fig=plt.figure(figsize=(8, 8))
    columns = 4
    rows = round(len(images)/columns)
    k=1
    for i in range(1, columns*rows +1):
        fig.add_subplot(rows, columns, i)
        img = images[k-1]
        plt.title(titles[k-1])
        plt.imshow(img, cmap = 'gray')
        k+= 1
    plt.show()

showImages(
    [im[1] for im in noised_images],
    [im[0] for im in noised_images],
)
```

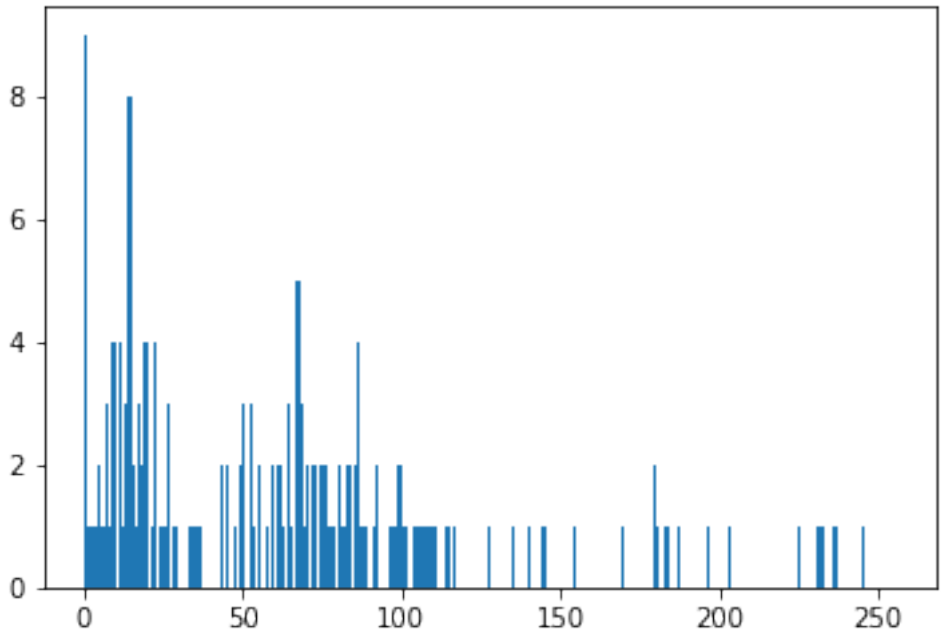


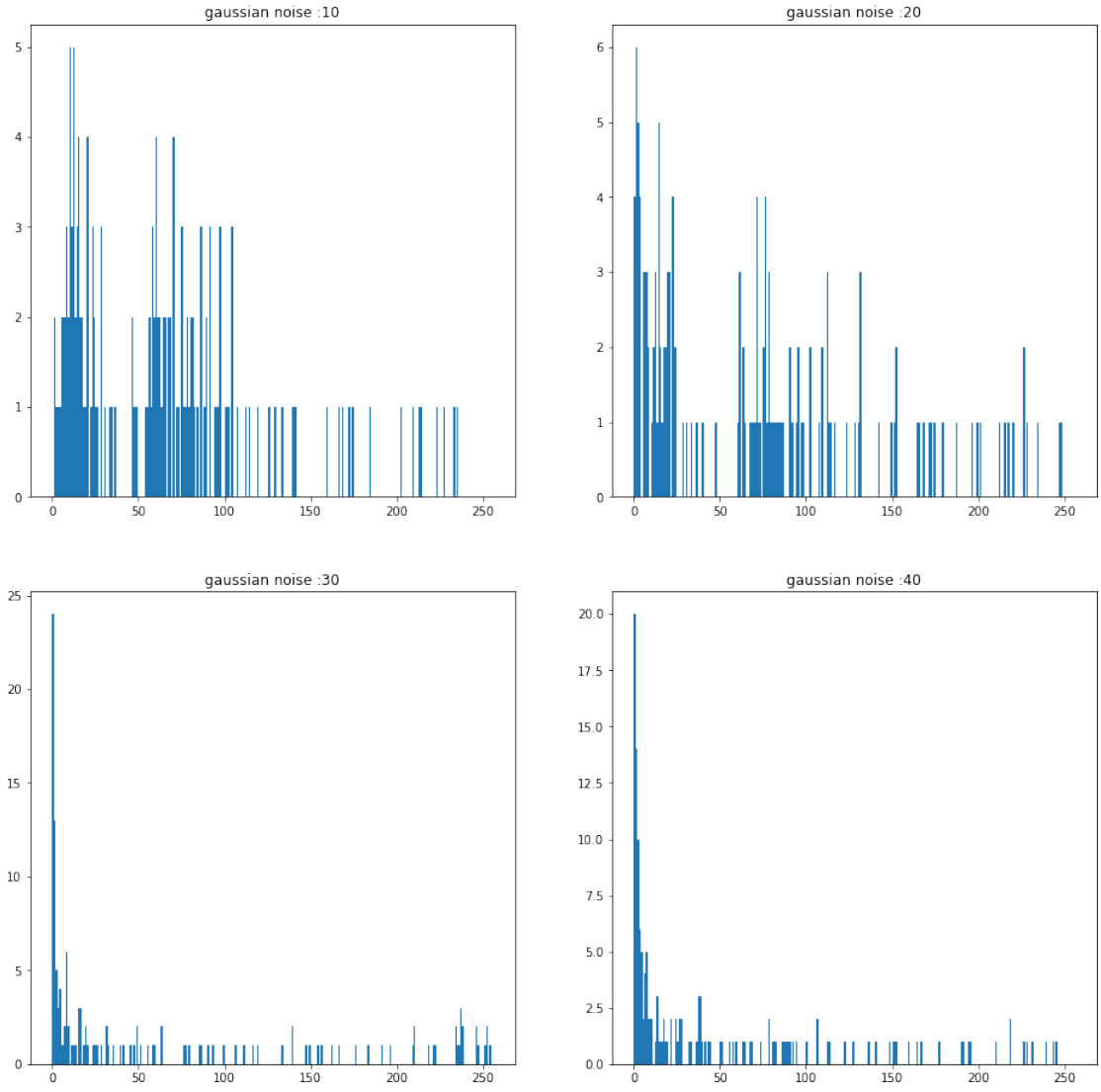
0.1.4 2.d) Show the histograms for these noisy images, can you explain what you see?

The more noisy the image the higher the difference with the original graph. We see the count for every pixel value of the image. :

[24]:
```
### 2.d) Show the histograms for these noisy images, can you explain what you␣
 ↪see?
def convertBack(im):
    imconv= np.interp(
        im,
        (im.min(), im.max()),
        (0, 255)
    )
    imconv = imconv.astype("uint8")
    return imconv

def hist(img, title=""):
```

```
    imconv = convertBack(img)
    hist , bins = np.histogram(imconv.ravel(),256,[0,256])
    plt.hist(hist, bins= bins)
    plt.title(title )
    plt.show()

hist(imgCam_double, "Original image")

titles = [im[0]  for im in noised_images]
images  = [im[1]  for im in noised_images]
fig=plt.figure(figsize=(16, 16))
columns = 2
rows = 2
k=1
for i in range(1, columns*rows +1):
        fig.add_subplot(rows, columns, i)
        img = images[k-1]
        plt.title(titles[k-1])
        imconv = convertBack(img)
        hist , bins = np.histogram(imconv.ravel(),256,[0,256])
        plt.hist(hist, bins= bins)# plt.imshow(img, cmap = 'gray')
        k+= 1
plt.show()
```





0.1.5 2.e1) Add salt & pepper Noise to an image until the PSNR ratio between the original and the noisy image is 40 dB.

The noise and pepper function is imlemented below

0.1.6 2.e.2) Visually compare it to the 40dB noisy image to which Gaussian noise was added. What can you conclude?

With salt and pepper noise function the picture is barely modified at 40db. While with gaussia noise it is geavily modified.

To validate that the salt and pper nois is coorectly implemented I trie to augmenta the noise heavily.

[26]:
```
### 2.e.1) Add salt & pepper Noise to an image until the PSNR ratio between the original and the noisy image is 40 dB.
import copy

def mse(imageA, imageB) -> float:
        err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
        err /= float(imageA.shape[0] * imageA.shape[1])
        return err


def gaussianNoise(image, var, mean = 0):
      row,col = image.shape
      sigma = var**0.5
      randomGaus = np.random.normal(mean,sigma,(row,col))
      gaussNoiseMatrix = randomGaus.reshape(row,col)
      noisy = image + gaussNoiseMatrix
      return noisy, gaussNoiseMatrix

def psnr(mseres):
    PIXEL_MAX_SQUARE = 1
    mse_square = math.sqrt(mseres )
    part1 =  math.log10(PIXEL_MAX_SQUARE / mse_square)
    return 20 * part1

imgCam = cv2.imread('./data/cameraman.tif', -1)
imgCam_double = np.array(imgCam).astype("float32")
imgCam_double = np.interp(
    imgCam_double,
    (imgCam_double.min(), imgCam_double.max()),
    (0, 1)
)

def get_img_with_pepper_salt_noise(img, p, q):
    im = copy.deepcopy(img)
    randnums = np.random.rand(256,256)
    im[np.logical_and(randnums > p, randnums < q)] = 1
    im[ randnums <= p] = 0
    return im

pepperedImg = get_img_with_pepper_salt_noise(imgCam_double, p=0.00015, q=0.0003)
mseNum  = mse(imgCam_double,pepperedImg)
print("mse: ", mseNum)
print("psnr: ", psnr(mseNum))
showTifGrayScale(noised_images[3][1], "gaussian image")
showTifGrayScale(pepperedImg, "Peppered image 40 DB psnr")
showTifGrayScale(imgCam_double, "Original image")
```

```
### 2.e.2) Visually compare it to the 40dB noisy image to which Gaussian noise
 ↪was added. What can you conclude?
pepperedImg = get_img_with_pepper_salt_noise(imgCam_double, p=0.15, q=0.3)
mseNum  = mse(imgCam_double,pepperedImg)
print("high noise image mse: ", mseNum)
print("high noise image psnr: ", psnr(mseNum))
showTifGrayScale(pepperedImg, "Peppered image")
```

```
mse:  0.0001810497329573739
psnr:  37.42202111417113
high noise image mse:  0.0946661067722857
high noise image psnr:  10.238054832616301
```

Peppered image 40 DB psnr

Original image


Peppered image