TP 1 quantification et échantillonage

Dans ce TP, nous aimerions transformer une image pseudo analogique en image numérique. L'image à coordonnées (x,y) réelles (continues) et valeurs d'amplitude f(x,y) réelles (continues), devra être numérisée en:

- transformant d'abord les coordonnées en entiers
- transformant ensuite les amplitudes en entiers

L'image pseudo analogique est contenue dans le fichier de données à loader.

Importation des librairies necessaires au travail

Entrée [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

Lecture des fichiers d'amplitude et coordonnées

Pour ce TP, nous allons lire les données à partir de fichier numpy.

Entrée [2]:
```
outfile1 = 'lines1.npy'
outfile2 = 'Cols1.npy'
outfile3 = 'amplitude1.npy'
lines=np.load(outfile1)
cols=np.load(outfile2)
amplitude=np.load(outfile3)
```

Echantillonage

Entrée [3]:

```
def sampling(lines, cols, step):
    '''nous prennons des échantillons des linges, colonnes et de
    l'amplitude à chaque saut step '''
    if len(lines)%step != 0:
        a=int(len(lines)/step)+1
    else:
        a=int(len(lines)/step)
    if len(cols)%step != 0:
        b=int(len(cols)/step)+1
    else:
        b=int(len(cols)/step)

    img = np.zeros((a,b))
    k=0
    for line in range(0, len(lines), step):
        l=0
        for col in range(0, len(cols), step):
            img[k,l]=amplitude[line, col]
            l+=1
        k+=1
    return(img)
```

Quantification

Entrée [4]:
```
def quantification(nBits, amplitude):
    # calculer le nombre de niveaux de gris
    L= pow(2,nBits)
    # les niveaux de gris possibles
    colors = [i for i in range(L)]
    # int et np.rint sont équivalents pour real to integer
    # changer de réel vers entier
    old_min=np.amin(amplitude)
    old_max=np.amax(amplitude)
    new_min=min(colors)
    new_max=max(colors)
    # convertir les anciennes valeurs vers les nouvelles
    quantified = np.rint(((amplitude-old_min)*(new_max-new_min)/(old_max-old_min)))
    return quantified
```

Addition de deux images

Entrée [5]:
```
# vérifier que les deus images ont la même taille
# redimensionner les images au besoin
def checkSize(image1, image2):
    minL=min(image1.shape[0],image2.shape[0])
    minC=min(image1.shape[1],image2.shape[1])
    image1 = cv2.resize(image1, dsize=(minL, minC))
    image2 = cv2.resize(image2, dsize=(minL, minC))
    return image1,image2
```

Entrée [6]:
```python
# méthode manuelle
# le pixel superieur à 255 est remplacé par 255
def addImagesbasic(image1, image2):
    image1,image2=checkSize(image1, image2)
    # appliquer l'addition
    added=np.zeros((image1.shape[0], image1.shape[0]))
    for i in range(image1.shape[0]):
        for j in range(image1.shape[1]):
            # on considere 255 comme element neutre
            if(image1[i,j]== 255):
                value =image2[i,j]
            else:
                if (image2[i,j]== 255):
                    value =image2[i,j]
                else:
                    value = image1[i,j]+ image2[i,j]
            if(value> 255):
                value =255
            added[i,j] = value
    return added
```

Entrée [7]:
```python
# méthode manuelle
# le pixel superieur à 255 est remplacé par 255
def addImages(image1, image2):
    image1,image2=checkSize(image1, image2)
    # appliquer l'addition
    added=image1+image2
    return added
```

Entrée [8]:
```python
# méthode avec modulo
# le pixel superieur à 255 est remplacé par pixel modulo 255
def addImagesModulo(image1, image2):
    image1,image2=checkSize(image1, image2)
    added = image1+ image2
    added = added % 256
    return added
```

Entrée [9]:
```python
# addition avec opencv
def addImagesCV(image1, image2):
    image1,image2=checkSize(image1, image2)
    image1=image1.astype(np.int16)
    image2=image2.astype(np.int16)
    added = cv2.add(image1, image2)
    return added
```

Entrée [10]:
```python
# addition avec poids, opencv
def addImagesCVweight(image1, image2):
    image1,image2=checkSize(image1, image2)
    image1=image1.astype(np.int8)
    image2=image2.astype(np.int8)
    added = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)
    return added
```

Entrée [11]:
```python
# addition avec numpy
def addImagesNP(image1, image2):
    image1,image2=checkSize(image1, image2)
    added = np.add(image1,image2)
    return added
```

Négatif d'une image

Entrée [12]:
```python
def imageNegative(image):
    imageN=np.zeros((image.shape[0],image.shape[1]))
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            imageN[i,j] =max(0, 255 - image[i,j])
    return imageN
```

Afficher les résultats

Comparer plusieurs sauts pour la quantification

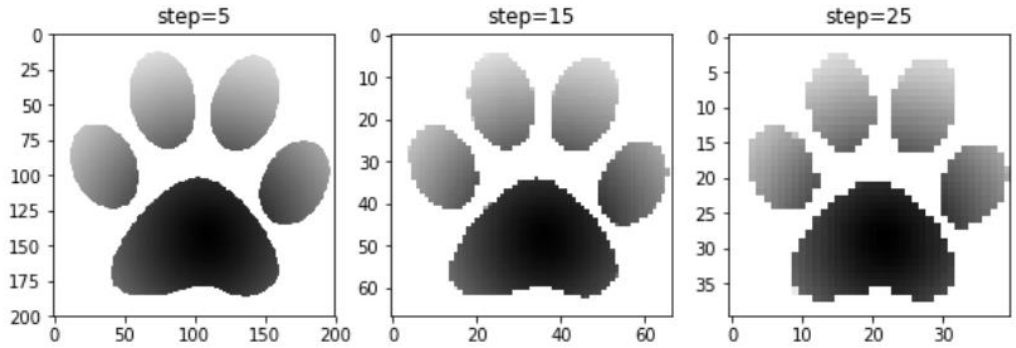
Entrée [13]:
```
step1=5
nBits=8
sampled1=sampling(lines, cols, step1)
quatified1=quantification(nBits, sampled1)
step2=15
sampled2=sampling(lines, cols, step2)
quatified2=quantification(nBits, sampled2)
step3=25
sampled3=sampling(lines, cols, step3)
quatified3=quantification(nBits, sampled3)
```

Affichage

Entrée [14]:
```
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.title('step='+str(step1))
plt.imshow(sampled1,cmap='gray')
plt.subplot(1,3,2)
plt.title('step='+str(step2))
plt.imshow(sampled2,cmap='gray')
plt.subplot(1,3,3)
plt.title('step='+str(step3))
plt.imshow(sampled3,cmap='gray')
```

Out[14]: <matplotlib.image.AxesImage at 0x24f9f64e320>

Out[14]: <matplotlib.image.AxesImage at 0x24f9f64e320>



En diminuant le pas de quantification, nous perdons du détail dans la forme de l'image

Comparer plusieurs nombre de bits pour la quantification

Entrée [15]:
```
nBits1=8
nBits2=4
nBits3=2
quantified11= quantification(nBits1, sampled1)
quantified21= quantification(nBits2, sampled1)
quantified31= quantification(nBits3, sampled1)

quantified12= quantification(nBits1, sampled2)
quantified22= quantification(nBits2, sampled2)
quantified32= quantification(nBits3, sampled2)

quantified13= quantification(nBits1, sampled3)
quantified23= quantification(nBits2, sampled3)
quantified33= quantification(nBits3, sampled3)
```

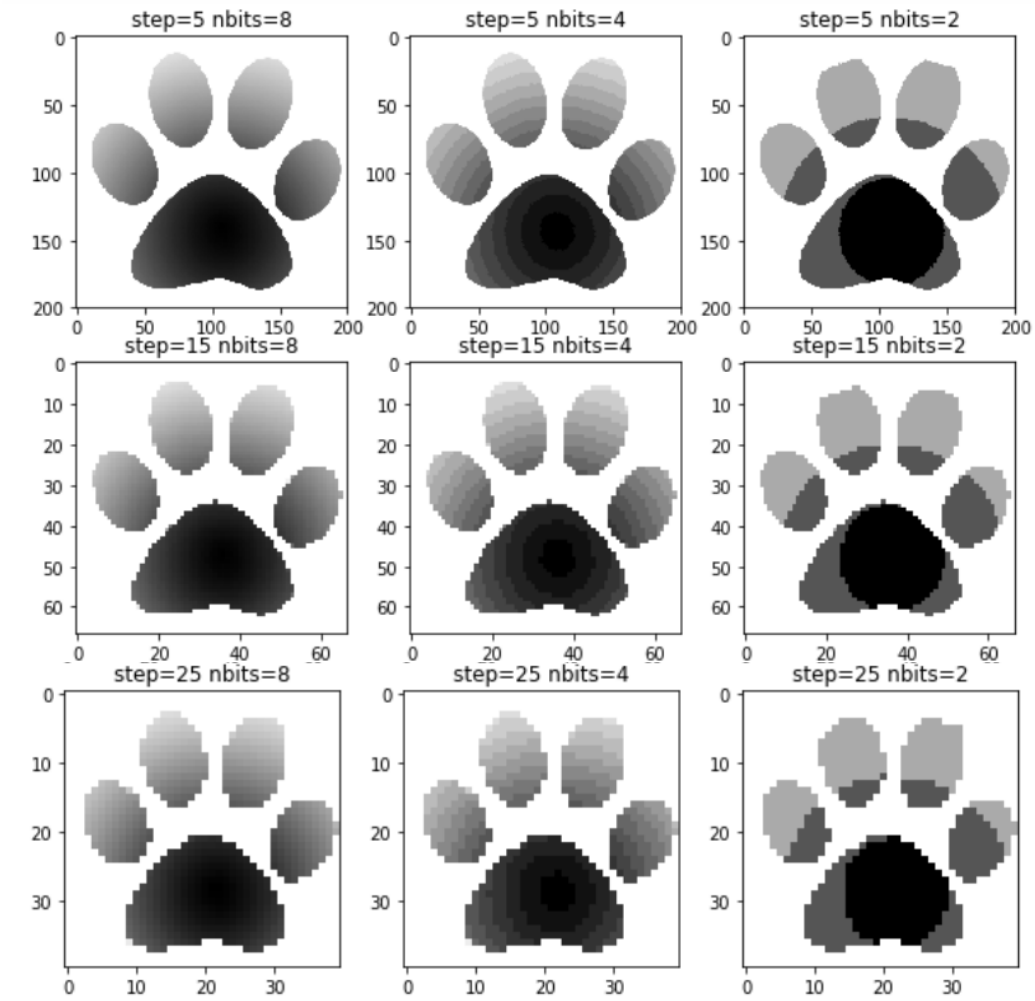
Affichage

Entrée [16]:
```
plt.figure(figsize=(10,10))
plt.subplot(3,3,1)
plt.title('step='+str(step1)+' nbits='+str(nBits1))
plt.imshow(quantified11,cmap='gray')
plt.subplot(3,3,2)
plt.title('step='+str(step1)+' nbits='+str(nBits2))
plt.imshow(quantified21,cmap='gray')
plt.subplot(3,3,3)
plt.title('step='+str(step1)+' nbits='+str(nBits3))
plt.imshow(quantified31,cmap='gray')

plt.subplot(3,3,4)
plt.title('step='+str(step2)+' nbits='+str(nBits1))
plt.imshow(quantified12,cmap='gray')
plt.subplot(3,3,5)
plt.title('step='+str(step2)+' nbits='+str(nBits2))
plt.imshow(quantified22,cmap='gray')
plt.subplot(3,3,6)
plt.title('step='+str(step2)+' nbits='+str(nBits3))
plt.imshow(quantified32,cmap='gray')

plt.subplot(3,3,7)
plt.title('step='+str(step3)+' nbits='+str(nBits1))
plt.imshow(quantified13,cmap='gray')
plt.subplot(3,3,8)
plt.title('step='+str(step3)+' nbits='+str(nBits2))
plt.imshow(quantified23,cmap='gray')
plt.subplot(3,3,9)
plt.title('step='+str(step3)+' nbits='+str(nBits3))
plt.imshow(quantified33,cmap='gray')
```

Out[16]: <matplotlib.image.AxesImage at 0x24f9f7b69e8>


step=5 nbits=8
step=5 nbits=4
step=5 nbits=2
step=15 nbits=8
step=15 nbits=4
step=15 nbits=2
step=25 nbits=8
step=25 nbits=4
step=25 nbits=2

En diminuant le nombre de bits, nous diminuons le détail dans les couleurs (niveaux de gris) de l'image

Lecture et affichage d'une autre image

Entrée [17]:
```
# lecture de l'image en niveau de gris
image = cv2.imread('toAdd.bmp',0)
```

Entrée [18]:
```
# afficher avec opencv
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Réaliser les operations sur l'image

Addition des deux images

Entrée [19]:
```
image1=quantified11.copy()
image2=image.copy()
```

Entrée [20]:
```
im1,im2=checkSize(image1, image2)
```

Entrée [21]:
```
fig=plt.figure(figsize=(12,12))
plt.subplots_adjust(wspace=1.2)
plt.subplot(4,3,1)
plt.imshow(im1,cmap='gray')
plt.title('image 1')
plt.subplot(4,3,2)
plt.imshow(im2,cmap='gray')
plt.title('image 2')
plt.subplot(4,3,3)
plt.title('add images manually')
```

```
plt.imshow(addImages(image1, image2),cmap='gray')
plt.subplot(4,3,4)
plt.title('image1+image2')
plt.imshow(addImagesbasic(image1, image2),cmap='gray')
plt.subplot(4,3,5)
plt.title('add images with modulo')
plt.imshow(addImagesModulo(image1, image2),cmap='gray')
plt.subplot(4,3,6)
plt.title('add images with opencv ')
plt.imshow(addImagesCV(image1, image2),cmap='gray')
plt.subplot(4,3,7)
plt.title('add images with opencv weighted')
plt.imshow(addImagesCVweight(image1, image2),cmap='gray')
plt.subplot(4,3,8)
plt.title('add images with numpy')
plt.imshow(addImagesNP(image1, image2),cmap='gray')
```

Out[21]: <matplotlib.image.AxesImage at 0x24f9ff5d7f0>

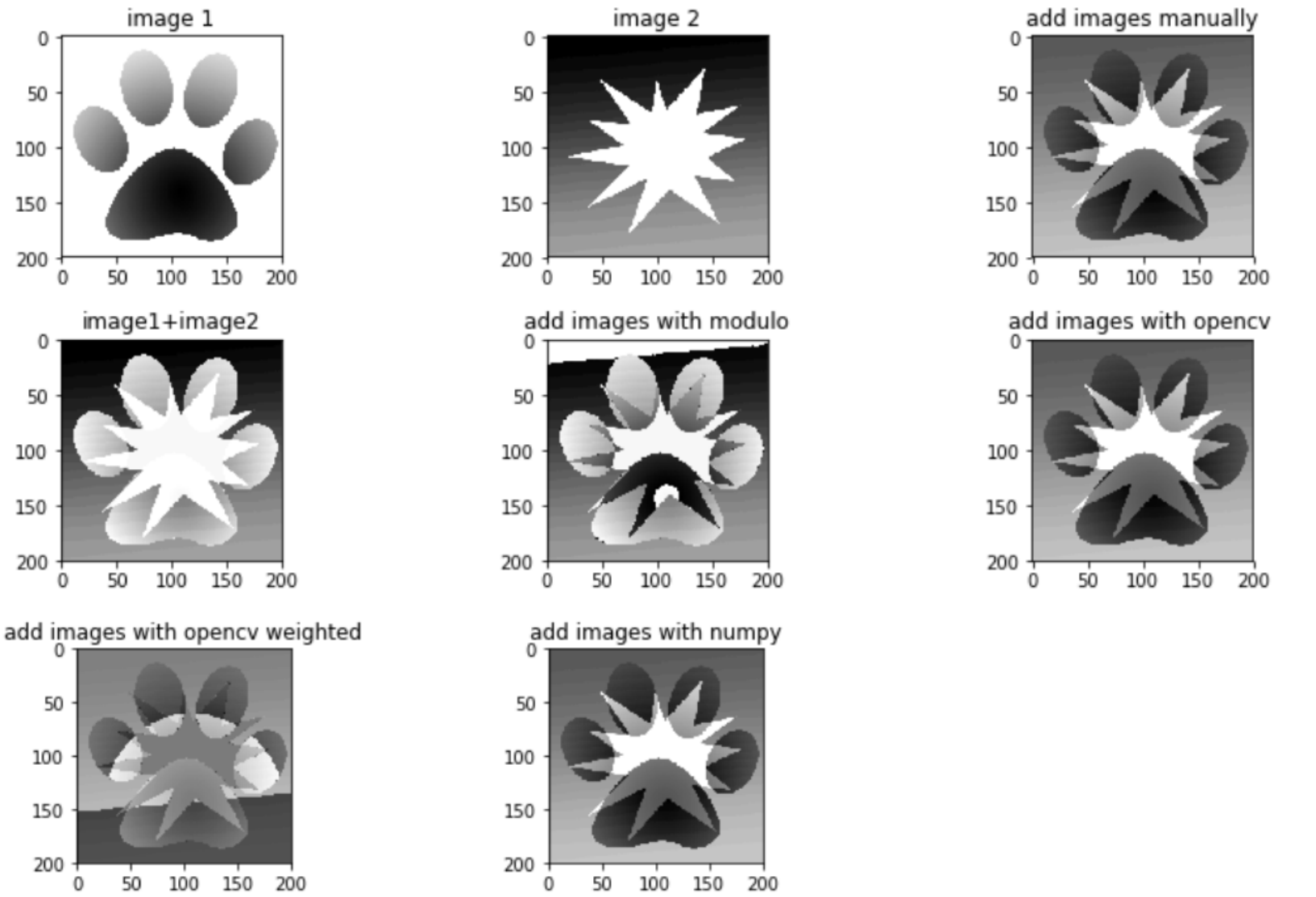


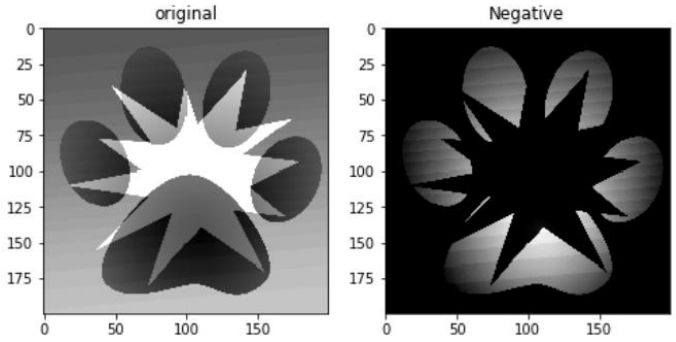
image 1
image 2
add images manually
image1+image2
add images with modulo
add images with opencv
add images with opencv weighted
add images with numpy

Negatif de l'image

Entrée [22]:
```
image=addImages(image1, image2)
imageN=imageNegative(image)

fig=plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.imshow(image,cmap='gray')
plt.title('original')
plt.subplot(1,2,2)
plt.imshow(imageN,cmap='gray')
plt.title('Negative')
```

Out[22]: Text(0.5, 1.0, 'Negative')



Extra

observer la ligne avant et après échnatillonage et quantification

Entrée [23]:

```
fig=plt.figure(figsize=(10,10))
plt.subplots_adjust(wspace=1)
plt.subplots_adjust(hspace=0.5,wspace=0.5)
plt.subplot(3,3,1)
plt.title('Amplitude')
plt.imshow(amplitude,cmap='gray')

plt.subplot(3,3,4)
plt.title('Sampled and qunatified')
plt.imshow(quantified21,cmap='gray')

plt.subplot(3,3,2)
plt.title('line:150')
plt.plot(amplitude[750])
```

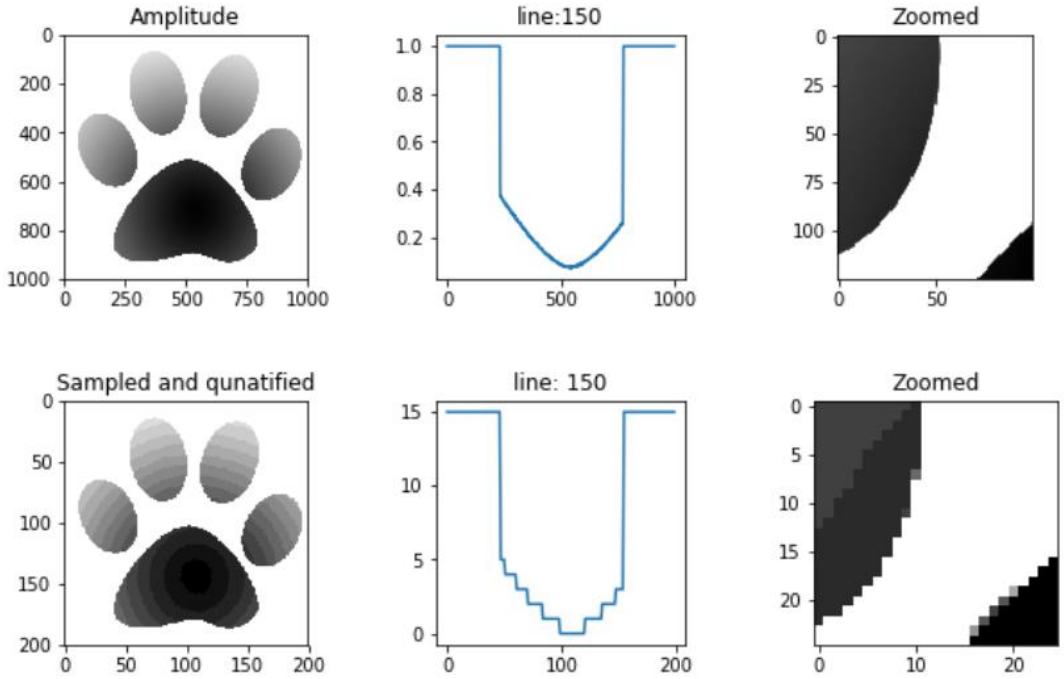
```
plt.subplot(3,3,5)
plt.title('line: 150')
plt.plot(quantified21[150])

subA = amplitude[500:625:1,250:350:1].copy()
subQ = quantified21[100:125:1,50:75:1].copy()

plt.subplot(3,3,3)
plt.title('Zoomed')
plt.imshow(subA,cmap='gray')

plt.subplot(3,3,6)
plt.title('Zoomed')
plt.imshow(subQ,cmap='gray')
```

Out[23]: <matplotlib.image.AxesImage at 0x24fa1a22780>



A faire :

Ecrire et comprendre le code précédant

Ecrire un code permettant :

1- De lire les images : lucy_paw_4bits.bmp et lucy_paw_8bits.bmp
2- Pour chacune des images, donner sa définition et sa résolution en niveau de gris