***Compte Rendu Du Travaux Pratique 2***

**Activité n°1 manipuler une image**

Lors de l’acquisition, de la transmission ou de la compression d’une image, il peut apparaître de nombreuses dégradations. Un des domaines principaux en traitement d’image consiste à traiter et corriger ces dégradations pour obtenir une image de meilleure qualité.

**Le bruit additif** : qui affecte tous les pixels de l’image avec un moyenne nulle et de variance σ2, qui affecte à la fois les basses et les hautes fréquences. Il est caractérisé par sa variance σ2 : plus σ2 est ´élevé, plus l’image est dégradée.

**Le bruit impulsionnel :** n’affecte que certains pixels de l’image qui est une dégradation de l’image sous la forme de pixels noirs et blancs répartis au hasard, le caractérise par le pourcentage p de pixels modifies : plus p est élevé, plus l’image est dégradée.

Premièrement, on va ouvrir image le lenna et Convertir en niveaux de gris par module Image de bibliotiques PIL.

from PIL import Image,ImageFilter

img = Image.open("../i

mages/Lenna.png").convert('L')



On va Utiliser les deux fonctions suivantes pour générer des images bruitées en changeant les paramètres.

def add\_gauss\_noise(img, variance):

img = np.array(img)

noise = np.random.normal(0, variance, img.shape)

noisy\_img = Image.fromarray(img + noise).convert('L')

return noisy\_img

img\_bruit\_gauss = add\_gauss\_noise(img, 20)

img\_bruit\_gauss.show()



def add\_imp\_noise(img,density):

row,col=img.size

number\_of\_pixels = int(density\*100)

for i in range(number\_of\_pixels):

y\_coord=random.randint(0, row - 1)

x\_coord=random.randint(0, col - 1)

img.putpixel((y\_coord,x\_coord), 255)

for i in range(number\_of\_pixels):

y\_coord=random.randint(0, row - 1)

x\_coord=random.randint(0, col - 1)

img.putpixel((y\_coord,x\_coord), 0)

return img

img\_bruit\_impl = add\_imp\_noise(img, 30)

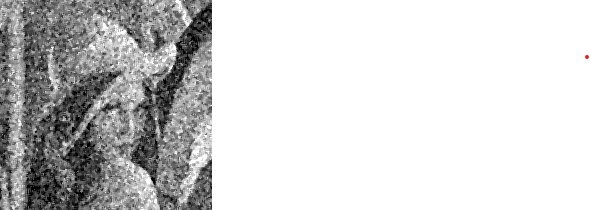
img\_bruit\_impl.show()



En utilisons la fonction ImageFilter. MedianFilter(), filtrer les images bruitées par le bruit « poivre et sel».

img\_medianFiler= img\_bruit\_gauss.filter(ImageFilter.MedianFilter(size=3))

img\_medianFiler.show()



img\_gaussFilter = img\_bruit\_gauss.filter(ImageFilter.BoxBlur(radius=0.08))

img\_gaussFilter.show()



**Code Complete d’activité :**

from PIL import Image,ImageFilter

import numpy as np

import random

img = Image.open("../images/Lenna.png").convert('L')

# Ajouter le bruits gaussienne a une image.

def add\_gauss\_noise(img, variance):

img = np.array(img)

noise = np.random.normal(0, variance, img.shape)

noisy\_img = Image.fromarray(img + noise).convert('L')

return noisy\_img

# Ajouter le Filter implusionnel

def add\_imp\_noise(img,density):

row,col=img.size

number\_of\_pixels = int(density\*100)

for i in range(number\_of\_pixels):

y\_coord=random.randint(0, row - 1)

x\_coord=random.randint(0, col - 1)

img.putpixel((y\_coord,x\_coord), 255)

for i in range(number\_of\_pixels):

y\_coord=random.randint(0, row - 1)

x\_coord=random.randint(0, col - 1)

img.putpixel((y\_coord,x\_coord), 0)

return img

# utiliser les deux images pour generer des images bruitees

img\_bruit\_gauss = add\_gauss\_noise(img, 80)

img\_bruit\_gauss.show()

img\_bruit\_impl = add\_imp\_noise(img, 15)

img\_bruit\_impl.show()

# en utilisant les methode MedianFilter et GaussienFilter de classe ImageFilter.

img\_medianFiler = img\_bruit\_gauss.filter(ImageFilter.MedianFilter(size=3))

img\_medianFiler.show()

img\_gaussFilter = img\_bruit\_gauss.filter(ImageFilter.GaussianBlur(radius=0.9))

img\_gaussFilter.show()

**Activité n°2 Filtrage d’une image**

Premièrement pour Filtrer une image on Explorer module ImageFilter

**from PIL import Image, ImageFilter**

**Pour réaliser un filtrage linéaire, il faut d’abord définir le masque h à utiliser. Pour cela, soit on le définit de façon analytique, soit on utilise la fonction GaussianBlur de la méthode ImageFilter.**

**img\_bruit\_gauss = add\_gauss\_noise(img, 80)**

**img\_bruit\_gauss.show()**

****

**Im\_flitree = img\_bruit\_gauss.filter(ImageFilter.GaussianBlur(radius = 0.9)).show()**

****

**image\_filtree = img\_bruit\_gauss.filter(ImageFilter.Kernel((3, 3),**

**(1,1,1,1,1,1,1,1,1)))**

**image\_filtree.show()**

****

Lorsqu'on applique du bruit gaussien, une variance plus élevée entraîne une image plus dégradée (ou bruitée). Pour améliorer la qualité de cette image, on applique un ensemble de filtres. En explorant le filtre gaussien (Gaussian Blur), on constate qu'un rayon plus petit permet de conserver une image plus lisible. Les résultats obtenus dans le noyau (kernel) sont presque identiques à ceux obtenus par l'application du filtre gaussien, mais la modification des paramètres influence significativement le résultat.

**Activité n°3 : détection de contours**

Ecrire une fonction Seuillage(seuil) qui permet de faire un seuillage d’une image d’entrée par un seuil donné.

**def seuillage(img,seuil ) :**

**pixel = img.load() # acces direct aux intensite d'image.**

**col , row = img.size**

**for i in range(col):**

**for j in range(row):**

**if pixel[i,j] >= seuil:**

**pixel[i,j]=255**

**else:**

**pixel[i,j]=0**

**return img**

**img\_seuil = seuillage(img, 130).show()**

****

**Utiliser la fonction ImageFilter.Kernel, générer le filtre de Sobel.**

* **Gradient Horizontal par sobel.**

**sobel\_gv = img.filter(ImageFilter.Kernel((3,3), (-1,0,1,-2,0,2,-1,0,1),1,0))**

**sobel\_gv.show()**

****

* **Gradient Vertical par sobel.**

**sobel\_gh = img.filter(ImageFilter.Kernel((3,3), (-1,-1,-1,0,0,0,1,1,1),1,0))**

**sobel\_gh.show()**

****

* **Addition G = Gx + Gh par sobel (Afficher après binarization)**

**addition\_s = np.asarray(sobel\_gv) + np.asarray(sobel\_gh)**

**img\_sobel = Image.fromarray(addition\_s)**

**img\_sobel.show()**



**Utiliser la fonction ImageFilter.Kernel, générer le filtre de Prewitt.**

* **Gradient Horizontal par Prewitt.**

**prewitt\_gh = img.filter(ImageFilter.Kernel((3,3), (-1,0,1,-1,0,1,-1,0,1),1,0))**

**prewitt\_gh.show()**

****

* **Gradient Vertical par Prewitt.**

**filter** **prewitt\_gv = img.filter(ImageFilter.Kernel((3,3), (-1,-1,-1,0,0,0,1,1,1),1,0))**

**prewitt\_gv.show()**

****

* **Addition G = Gx + Gh par Prewitt (Afficher après binarization)**

**addition\_p = np.asarray(prewitt\_gv) + np.asarray(prewitt\_gh)**

**img\_prewitt = Image.fromarray(addition\_p)**

**img\_prewitt.show()**

****

**Code Complete de cette partie :**

from PIL import Image,ImageFilter

import numpy as np

import random

"""

les filters pass haut Sobel et Previtt sont utiliser pour detection des contours

"""

img = Image.open("../images/Lenna.png").convert('L')

def seuillage(img,seuil ) :

pixel = img.load() # acces direct aux intensite d'image.

col , row = img.size

for i in range(col):

for j in range(row):

if pixel[i,j] >= seuil:

pixel[i,j]=255

else:

pixel[i,j]=0

return img

img\_seuil = seuillage(img, 130).show()

# utiliser les fonctions ImageFilter.Kernel detecter les conteurs par sobel

sobel\_gv = img.filter(ImageFilter.Kernel((3,3), (-1,0,1,-2,0,2,-1,0,1),1,0))

sobel\_gv.show()

sobel\_gh = img.filter(ImageFilter.Kernel((3,3), (-1,-1,-1,0,0,0,1,1,1),1,0))

sobel\_gh.show()

addition\_s = np.asarray(sobel\_gv) + np.asarray(sobel\_gh)

img\_sobel = Image.fromarray(addition\_s)

img\_sobel.show()

# utiliser les fonctions ImageFilter.Kernel detecter conteur par prewitt

prewitt\_gh = img.filter(ImageFilter.Kernel((3,3), (-1,0,1,-1,0,1,-1,0,1),1,0))

prewitt\_gh.show()

prewitt\_gv = img.filter(ImageFilter.Kernel((3,3), (-1,-1,-1,0,0,0,1,1,1),1,0))

prewitt\_gv.show()

addition\_p = np.asarray(prewitt\_gv) + np.asarray(prewitt\_gh)

img\_prewitt = Image.fromarray(addition\_p)

img\_prewitt.show()