Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Rapport d'analyse d'image

TP n°1 :

Q1. On change les pixels et donc lors du calcul des coordonnées de XR YR, les résultats sont des chiffres non entier, sauf que dans les pixels, eux, sont représentés par des entiers.

Il faut donc arrondir les résultats et certains endroits ne seront jamais renseignés du fait de l'arrondi.

Q2. Voici notre nouveau code de rotation avec disparition des points noirs :

```python
def RotationCartesienne(image, angle):
    # Initialisation
    angle = m.radians(angle)
    cos = m.cos(angle)
    sin = m.sin(angle)
    h, w = image.shape
    h2 = int(h/2)
    w2 = int(w/2)
    image2 = np.zeros((h, w), dtype=np.uint8)

    # Rotation de l'image avec disparition des points noirs
    for i in range(h):
        for j in range(w):
            x = int((j-w2)*cos - (i-h2)*sin + w2)
            y = int((j-w2)*sin + (i-h2)*cos + h2)
            if x >= 0 and x < w and y >= 0 and y < h:
                image2[i, j] = image[y, x]
    return image2
```

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Voici nos résultats sur coco.bmp :

Q3. Il n'y a pas de points noirs dans l'image modifiée car l'image suit le déplacement (rotation) des pixels de l'image originale. Elle tourne autour d'elle même ce qui ne cause pas de perte de pixels.

Voici notre algorithme :

```
def RotationPolaires(image, angle):
    # Initialisation
    angle = radians(angle)
    h, w = image.shape
    h2 = int(h/2)
    w2 = int(w/2)
    image2 = np.zeros((h, w), dtype=np.uint8)

    # Rotation de l'image avec disparition des points noirs
    for i in range(h):
        for j in range(w):
            x = int((j-w2)*cos(angle) - (i-h2)*sin(angle) + w2)
            y = int((j-w2)*sin(angle) + (i-h2)*cos(angle) + h2)
            if x >= 0 and x < w and y >= 0 and y < h:
                image2[i, j] = image[y, x]
    return image2
```

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Voici le résultat sur coco.bmp :

Q4. Voici le résultat obtenu :

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Q5.

```
#Lecture de l'image
image =
plt.imread(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))+"\\Imag
es\\coco.bmp")

#algorithme de zoom en coordonnées cartésiennes
def ZoomCart(image, zoom):
    #initialisation
    h, w = image.shape
    h2 = int(h/2)
    w2 = int(w/2)
    image2 = np.zeros((h, w), dtype=np.uint8)
    #zoom de l'image en coordonnées cartésiennes
    for i in range(h):
        for j in range(w):
            x = int((i-h2)/zoom+h2)
            y = int((j-w2)/zoom+w2)
            if x>=0 and x<h and y>=0 and y<w:
                image2[i, j] = image[x, y]
    return image2
```

Voici les résultats :

Q6.

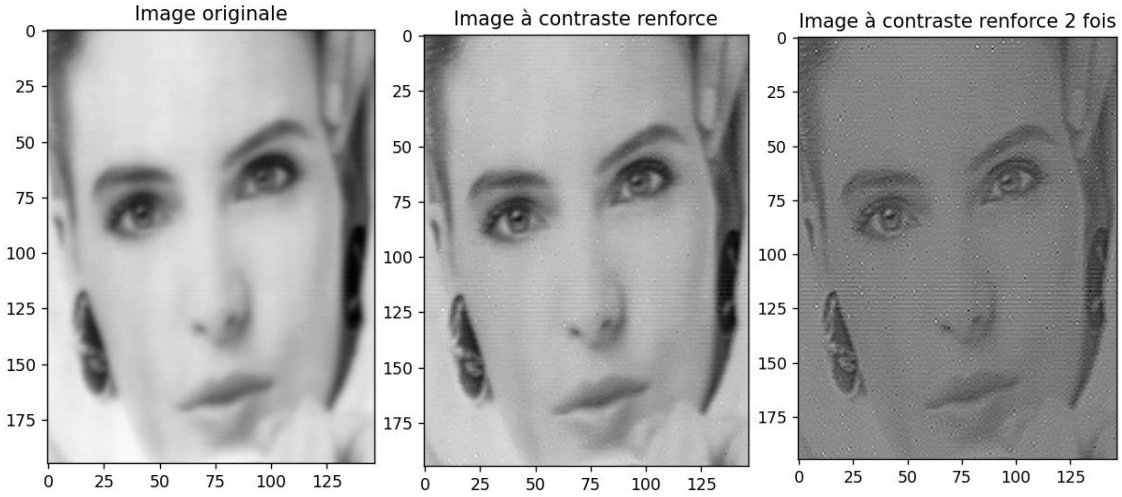
```
def RecadrageDynamique(image):
    # Initialisation
    h, w = image.shape
    image2 = np.zeros((h, w), dtype=np.uint8)
    min = 255
```

```
    max = 0

    # Recherche du minimum et du maximum
    for i in range(h):
        for j in range(w):
            if image[i, j] < min:
                min = image[i, j]
            if image[i, j] > max:
                max = image[i, j]

    # Recadrage de dynamique
    for i in range(h):
        for j in range(w):
            image2[i, j] = int((image[i, j] - min) * 255 / (max - min))
    return image2
```

TP n°2 :

Q1. Pour notre cas, nous avons choisi d'utiliser signal.convol2d(I, Mask).

Voici les résultats :

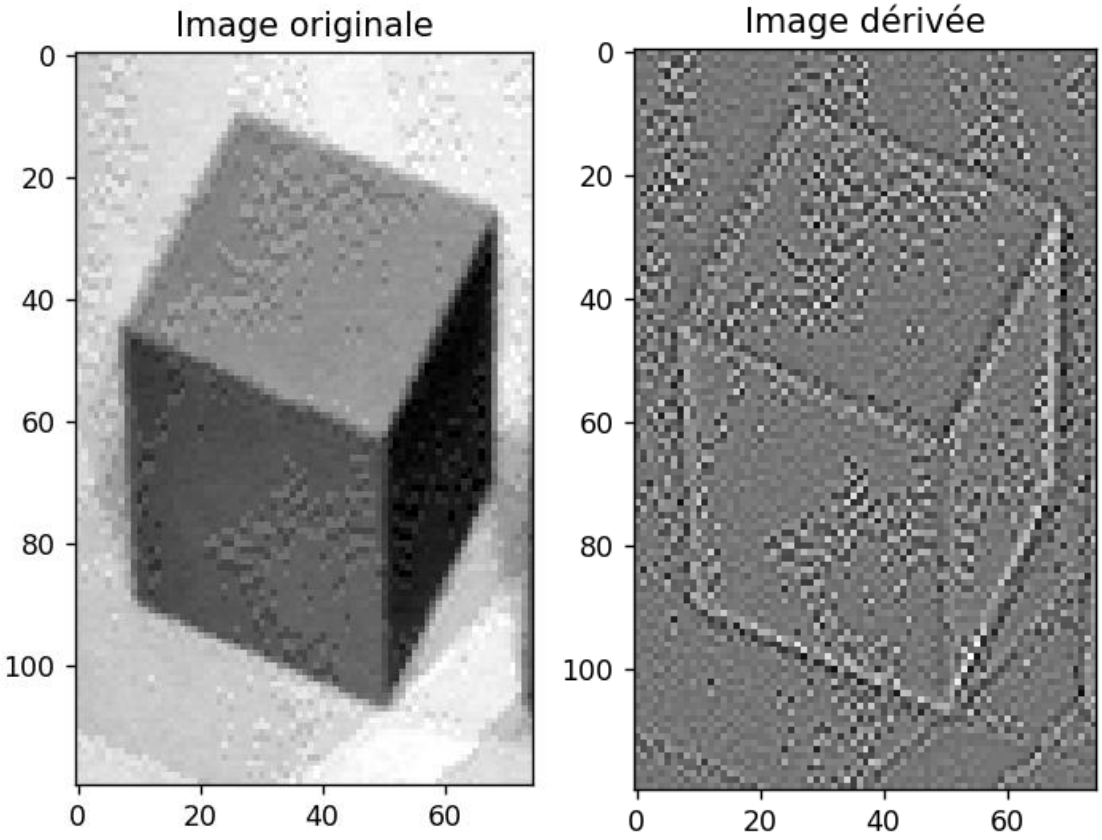


Cette fonction permet de renforcer le contraste d'une image, ici deux fois, en parcourant l'image de pixel en pixel, et en modifiant le pixel selon ceux présent autour de lui ainsi que le masque.

Q2. Le masque 2D de l'opération de dérivation est :

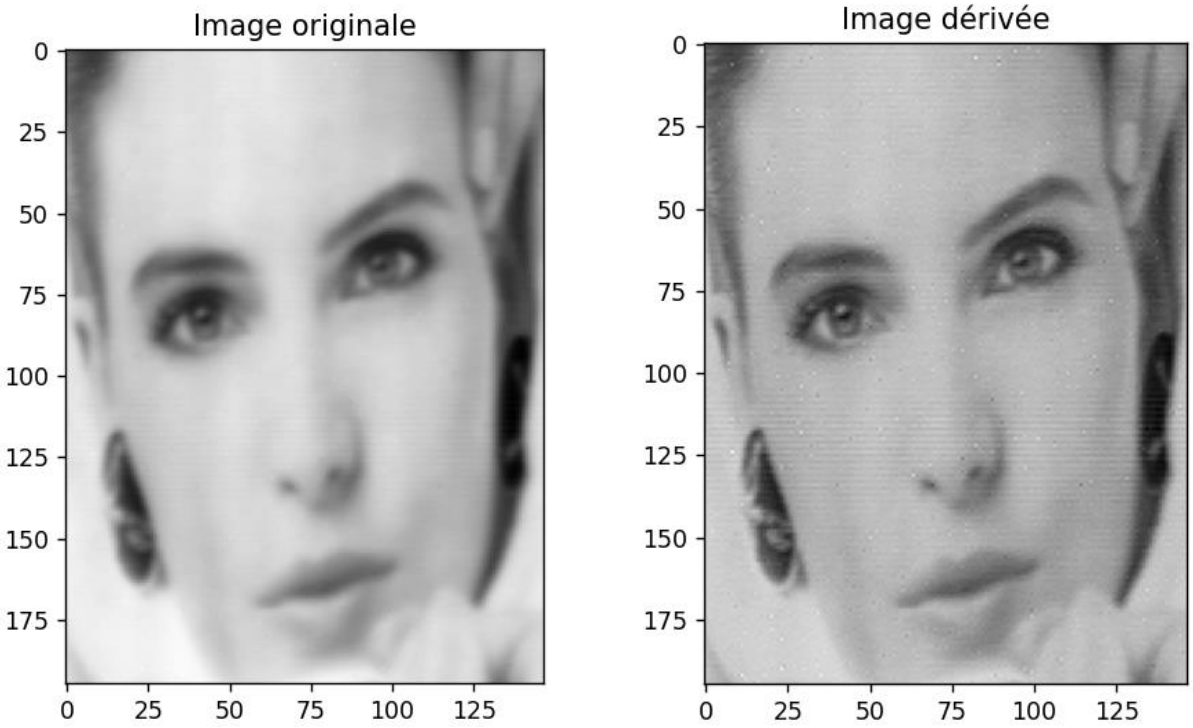
```
Mask = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
```

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022
Voici les résultats :



Ainsi, grâce à la dérivation, il est possible de mettre plus en avant les contours du cube dans l'image

Q3. Voici les résultats :



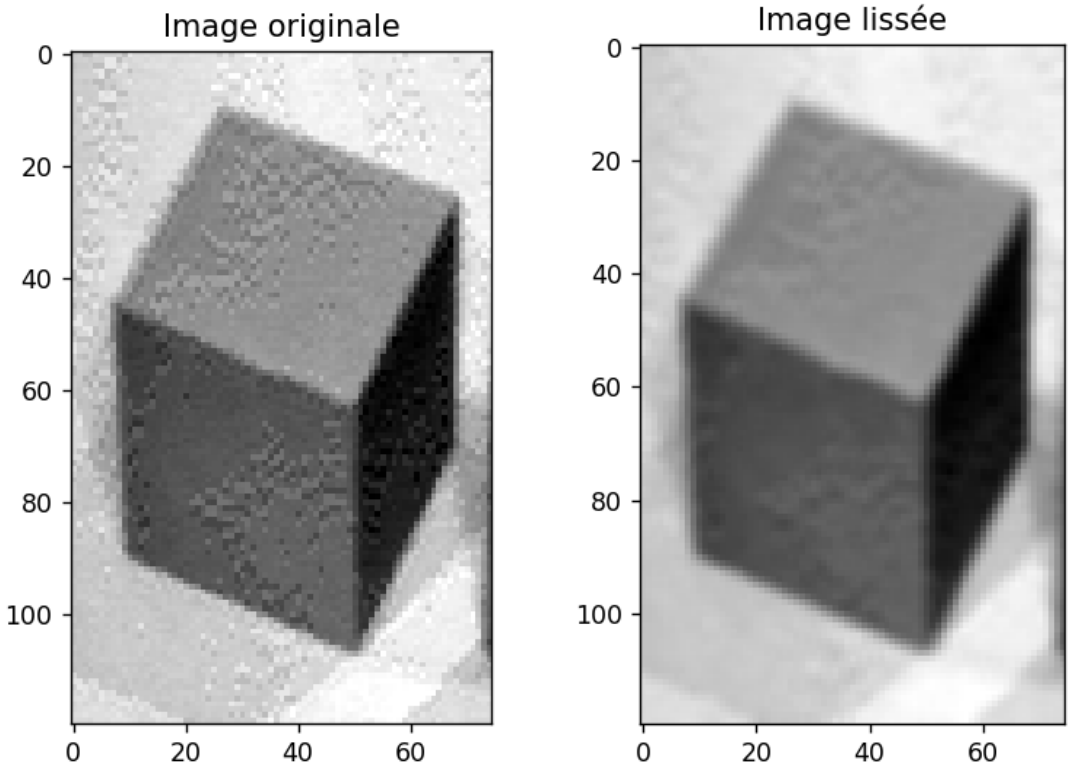
Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

La précision semble plus grande qu'avec la convolution, mais nous perdons plus de pixels dans la partie gauche et supérieure de l'image

Q4. Le masque de l'opérateur de lissage 3x3 par la moyenne est le suivant :

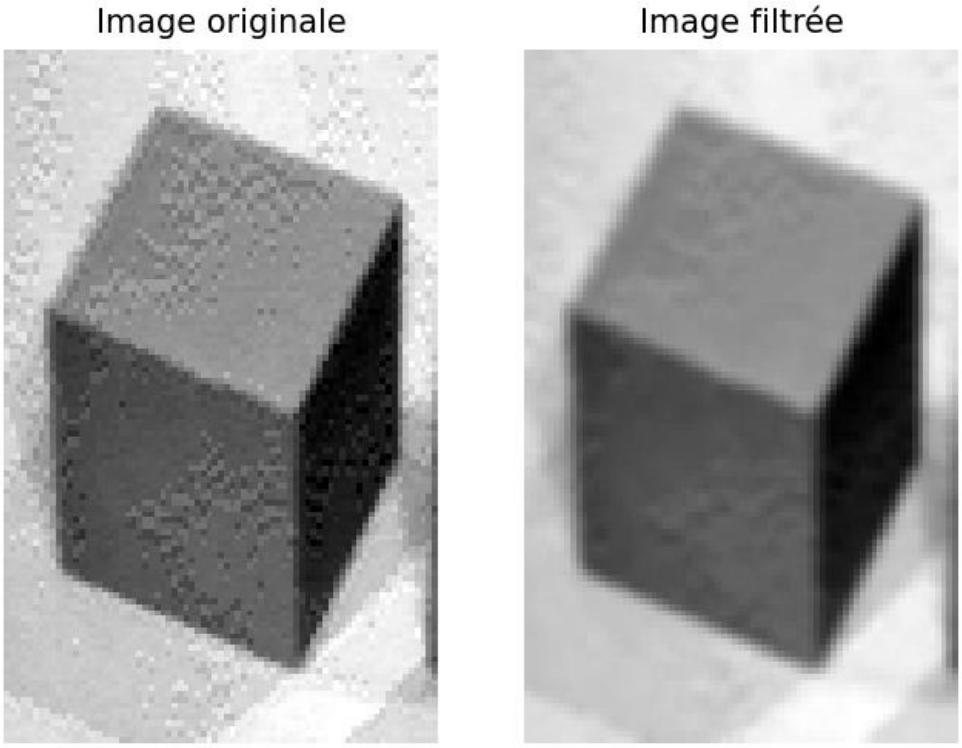
```
Mask = np.array([[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]])
```

Voici les résultats :



Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Q5. Voici les résultats :



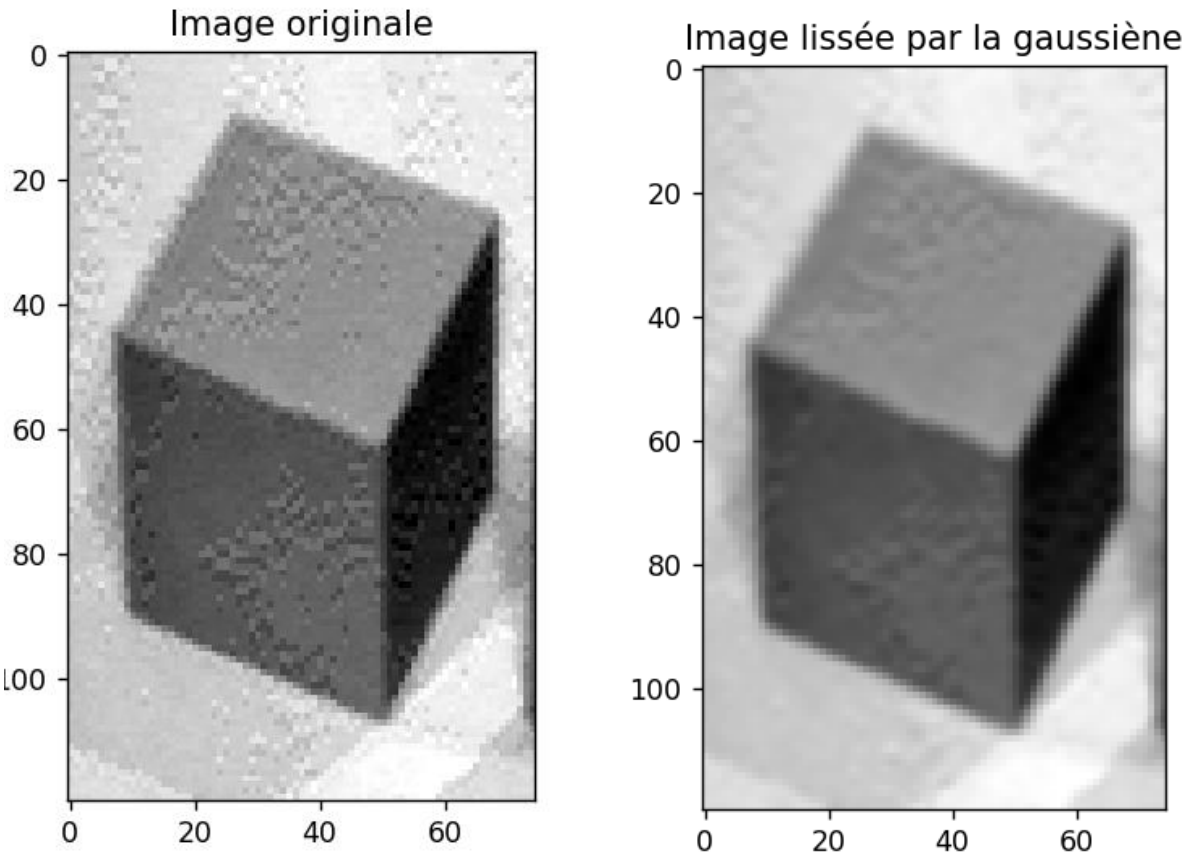
L'opérateur de lissage médian 3x3 est un filtre non linéaire. Il est donc plus adapté pour supprimer le bruit impulsionnel que le bruit gaussien.

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Q6. Le masque 2D de l'opérateur de lissage Gaussien 3x3 est le suivant :

```
Mask = np.array([[1/16, 2/16, 1/16], [2/16, 4/16, 2/16], [1/16, 2/16, 1/16]])
```

Voici les résultats :



Q7. L'opérateur de lissage de Nagao est un filtre non linéaire. Il est donc plus adapté pour supprimer le bruit impulsionnel que le bruit gaussien.

Q8. Voir notre code dans le GitHub, nous avons ce problème :

```
32    plt.imshow(D, cmap='gray')
```

```
Une exception s'est produite : TypeError ×
Image data of dtype complex128 cannot be converted to float

  File "C:\Users\alexa\Downloads\Git\AnalyseImage\TP2\Q8.py", line 32, in <module>
    plt.imshow(D, cmap='gray')
TypeError: Image data of dtype complex128 cannot be converted to float
```

TP n°3 :

Q1. (le code étant assez long, je vous recommande d'aller voir sur le GitHub le code de cette question en cliquant ici)

Q2.

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

TP n°4 :

Q1. Voici les deux algorithmes :

```python
def dilatation(image, se):
    # Initialisation de la sortie
    output = np.zeros(image.shape, dtype=np.uint8)
    # On explore chaque pixel de l'image
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # On parcourt chaque élément du masque
            for k in range(se.shape[0]):
                for l in range(se.shape[1]):
                    # Si le pixel est à 1 dans le masque
                    if se[k, l] == 1:
                        # On applique la dilatation
                        if image[i-k, j-l] == 1:
                            output[i, j] = 1
    return output


def erosion(image, kernel):
 output = np.zeros_like(image)
 image_padded = np.zeros((image.shape[0] + 2, image.shape[1] + 2))
 image_padded[1:-1, 1:-1] = image
 for x in range(image.shape[1]):
    for y in range(image.shape[0]):
        output[y,x]=(kernel*image_padded[y:y+3,x:x+3]).min()
    return output
```

Il y a une tendance vers une stabilité du motif après un grand nombre de dilatations (respectivement d'erosions). Si on applique une erosion suivie d'une dilatation, cela aura pour effet de réduire le nombre d'objets (ou contours) dans le motif et de les lisser. Si on applique une dilatation suivie d'une erosion, cela aura pour effet d'augmenter le nombre d'objets (ou contours) dans le motif et de les rendre plus prononcés, de cette manière on peut extraire les contours d'image binaire.

Alexandre Francony – Pierre Valles
Victor Bonnin – Joris Casadavant
ESILV BIN A3 Dev - 2022

Q2.

```python
#Calcul de l'axe median
def median_axis(img):
    #Calcul de la distance de chaque pixel à l'origine
    dist = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            dist[i,j] = np.sqrt(i**2+j**2)
    #Calcul de l'axe median
    axis = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i,j] == 1:
                axis[i,j] = dist[i,j]
    return axis
```

Q3. (le code étant assez long, je vous recommande d'aller voir sur le GitHub le code de cette question en cliquant ici)

Q4.

```python
def dilatation(image, kernel):
    """
    Cette fonction applique une dilatation sur l'image de luminance donnée
    en utilisant un noyau spécifié.

    Arguments:
    image -- L'image de luminance à traiter
    kernel -- Le noyau à utiliser pour appliquer la dilatation

    Retourne:
    dilated_image -- L'image dilatée
    """
    # Applique la dilatation
    dilated_image = cv2.dilate(image, kernel)

    return dilated_image

# Fonction pour éroder l'image
def erosion(image, kernel):
    """
    Cette fonction applique une érosion sur l'image de luminance donnée
    en utilisant un noyau spécifié.

    Arguments:
    image -- L'image de luminance à traiter
    kernel -- Le noyau à utiliser pour appliquer l'érosion

    Retourne:
    eroded_image -- L'image érodée
    """
    # Applique l'érosion
    eroded_image = cv2.erode(image, kernel)

    return eroded_image
```

TP n°5 :

Q1.

Q2.

Q3.