Faculté des sciences et techniques, Beni Mellal

Computer Vision - TP2

Réalisé par :

- Bel-assal Mohamed

Encadré par :

- Pr. Elayachi Rachid

Année académique : 2023-2024

1 Preparation d'image

Tout d'abord, nous disposons d'une image couleur, et la première étape consistera à la transformer en une version en niveaux de gris. Cela signifie que nous allons retirer toutes les informations de couleur pour ne conserver que les différentes intensités de lumière, allant du noir au blanc.

```
import cv2
import numpy as np

image = cv2.imread('./fox.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

G_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Voici l'image originale à gauche et sa version convertie en niveaux de gris à droite :

FIGURE 1 – Transformation : Couleur vers Niveaux de Gris

Pour appliquer et comparer différents filtres, nous allons ajouter du bruit à l'image. Cela permettra de simuler des perturbations et d'évaluer l'efficacité des filtres pour les réduire tout en préservant les détails importants.

```
def add_noise(image, noise):
    i, j = image.shape
    noisy_image = image.copy()

    X = np.random.randint(0, i - 1, noise)
    Y = np.random.randint(0, j - 1, noise)

    for x, y in zip(X, Y):
        noisy_image[x:x+3, y:y+3] = np.random.choice([0, 255])

    return noisy_image

noisy_image = add_noise(G_image, 70)
```

FIGURE 2 – Image bruitée

Nous avons ajouté du bruit sous forme de petits clusters de 3x3 pixels afin de mieux visualiser les perturbations dans l'image. Cela permet de rendre les effets du bruit plus visibles et d'observer plus clairement comment les filtres peuvent le réduire.

2 Filtres spatials

2.1 Filtre Gaussien

Le filtre gaussien est un type de filtre utilisé en traitement d'image pour flouter une image, c'est-à-dire réduire les détails fins et les bruits. Il fonctionne en appliquant une fonction gaussienne (qui suit la courbe en cloche de la distribution normale) sur chaque pixel de l'image, en prenant en compte ses voisins proches.
L'idée principale du filtre gaussien est de donner plus de poids aux pixels proches du pixel central et moins de poids à ceux qui sont plus éloignés. Cela permet de lisser l'image de manière plus naturelle, en conservant les contours tout en réduisant le bruit.
La formule mathématique du filtre gaussien est :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Où σ est l'écart-type de la distribution gaussienne, qui détermine l'intensité du flou. Plus σ est grand, plus l'effet de flou est prononcé.

```
def gaussian_kernel(size, sigma):
    kernel = np.zeros((size, size))
    center = size // 2
    for x in range(size):
        for y in range(size):
            dx = x - center
            dy = y - center
            kernel[x, y] = np.exp(-(dx**2 + dy**2) / (2 * sigma**2))
    kernel /= np.sum(kernel)
    return kernel
```

La fonction `gaussian_kernel(size, sigma)` génère un noyau gaussien 2D utilisé pour flouter les images. Elle calcule les valeurs gaussiennes pour chaque élément de la matrice en fonction de la distance au centre, puis normalise le noyau pour que la somme des valeurs soit égale à 1.

```
def apply_filter(image, kernel):

    image_height, image_width = image.shape
    kernel_size = kernel.shape[0]
    pad = kernel_size // 2

    padded_image = np.pad(image, pad, mode='constant', constant_values=0)
    filtered_image = np.zeros_like(image)

    for i in range(image_height):
        for j in range(image_width):
            region = padded_image[i:i + kernel_size, j:j + kernel_size]
            filtered_image[i, j] = np.sum(region * kernel)

    return filtered_image
```

On utilise cette fonction pour appliquer le noyau gaussien, que l'on applique ensuite sur l'image pour réaliser un flou, comme suit :

```
kernel_size = 5
sigma = 1.6
kernel = gaussian_kernel(kernel_size, sigma)

filtered_image = apply_filter(noisy_image, kernel)
```

Voici l'image avant et aprés le filtrage gaussien :

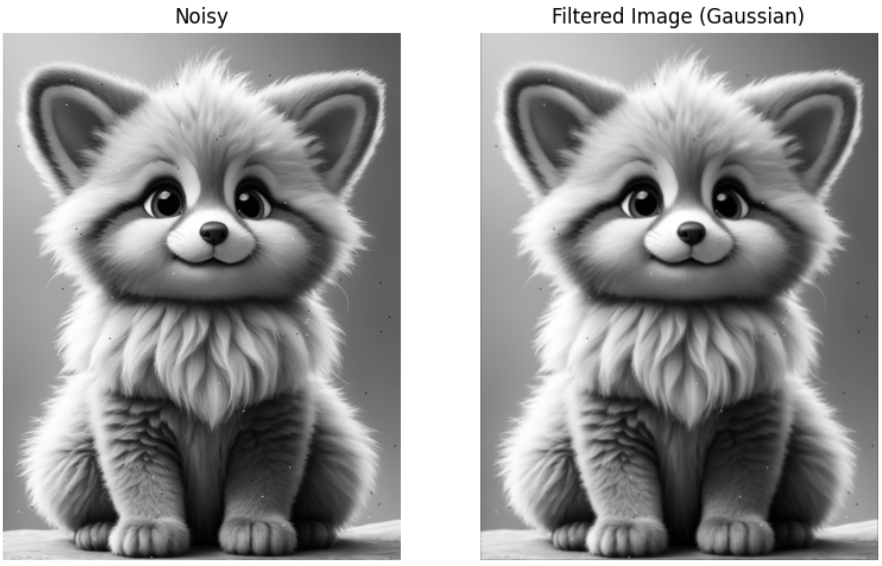

FIGURE 3 – Filtrage Gaussien

2.2 Filtre Binomial

Le filtre binomial est un filtre de lissage utilisé en traitement d'image, similaire au filtre gaussien, mais basé sur une distribution binomiale. Il est souvent préféré pour sa simplicité et ses calculs plus rapides. Le noyau du

filtre binomial est constitué de coefficients issus de la distribution binomiale, calculés à l'aide de la formule des coefficients binomiaux $C(n,k) = \frac{n!}{k!(n-k)!}$.

Par exemple, un noyau de taille 3x3 peut être :

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Ces coefficients sont ensuite normalisés pour que leur somme soit égale à 1, permettant de préserver la luminosité de l'image. Lors de l'application du filtre, chaque pixel de l'image est remplacé par une moyenne pondérée de ses voisins, selon les valeurs du noyau. Le filtre binomial réduit ainsi le bruit et les détails fins tout en conservant les bords plus nets.

```
def binomial_kernel(size):
    coeffs = [1]
    for i in range(size - 1):
        coeffs = [sum(x) for x in zip([0] + coeffs, coeffs + [0])]
    kernel_1d = np.array(coeffs) / sum(coeffs)
    kernel_2d = np.outer(kernel_1d, kernel_1d)
    return kernel_2d
```

La fonction `binomial_kernel(size)` génère un noyau binomial 2D de la taille spécifiée en calculant d'abord les coefficients binomiaux 1D, puis en normalisant ces coefficients et en créant un noyau 2D à l'aide du produit extérieur du noyau 1D. Nous utilisons la fonction déjà définie `apply_filter(image, kernel)` pour appliquer le noyau binomial à l'image

```
kernel_size = 3
kernel = binomial_kernel(kernel_size)

filtered_image = apply_filter(noisy_image, kernel)
```

Voici l'image avant et aprés le filtrage binomial :

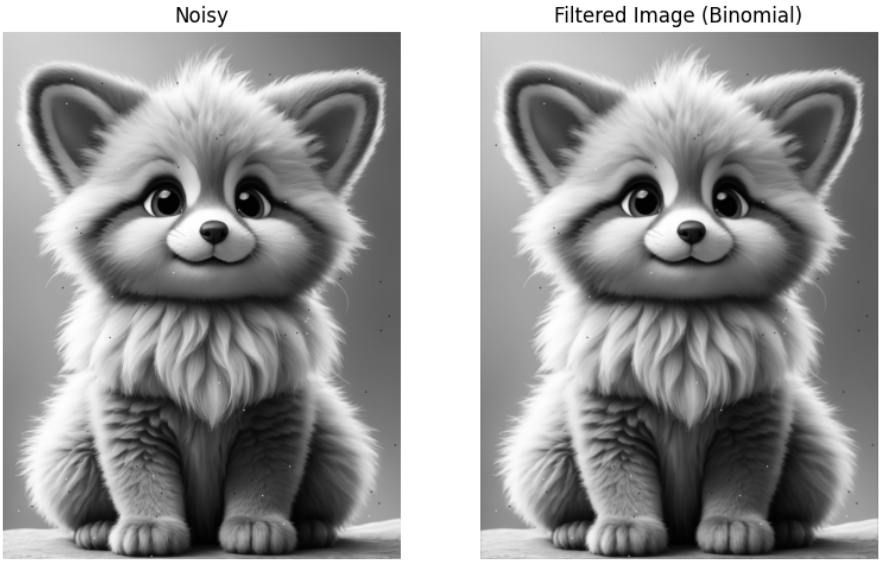

FIGURE 4 – Filtrage Binomial

2.3 Filtre Moyenne

Le filtre moyenne est un filtre spatial utilisé pour réduire le bruit en remplaçant chaque pixel par la moyenne des valeurs de ses voisins dans une fenêtre carrée ou rectangulaire. Par exemple, pour une fenêtre 3x3, la nouvelle valeur d'un pixel est la moyenne de ses 9 voisins. Ce filtre est simple et efficace pour lisser une image, mais il peut entraîner une perte de détails fins et une atténuation des bords. La formule pour un pixel $I'(x, y)$ est la suivante :

$$I'(x, y) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x + i, y + j)$$

où $I(x, y)$ représente la valeur du pixel original et $I'(x, y)$ la nouvelle valeur après l'application du filtre.

```
def filtre_moyenne(image, taille_kernel):
    kernel = np.ones((taille_kernel, taille_kernel)) / (taille_kernel *
        taille_kernel)
    image_height, image_width = image.shape

    pad = taille_kernel // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)

    filtered_image = np.zeros_like(image)
    for i in range(image_height):
        for j in range(image_width):
            region = padded_image[i:i+taille_kernel, j:j+taille_kernel]
            filtered_image[i, j] = np.sum(region * kernel)

    return filtered_image
```

On applique le filtre

```
filtered_image = filtre_moyenne(noisy_image, 3)
```

Voici l'image avant et aprés le filtrage moyenne :

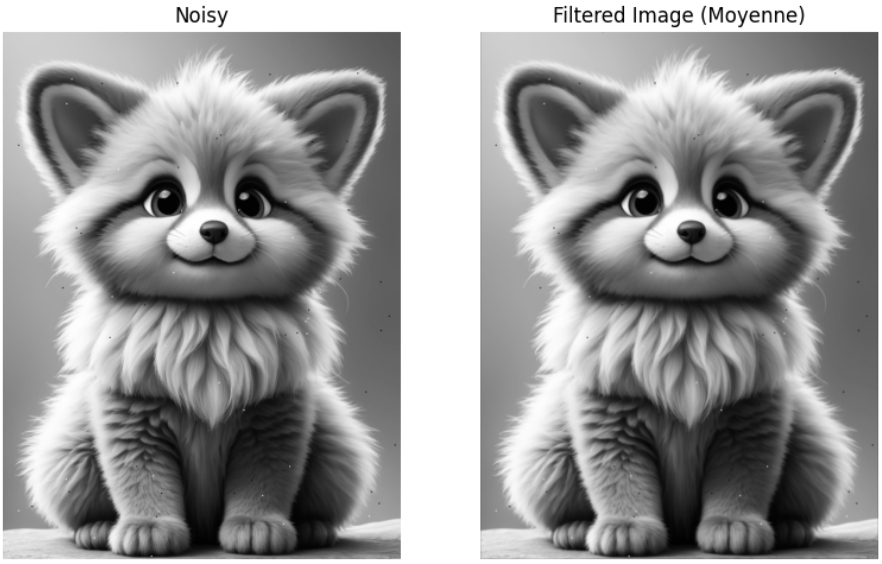

FIGURE 5 – Filtrage Moyenne

3 Filtres non lineaires

3.1 Filtre Mediane

Le filtre médian est un filtre non linéaire utilisé pour réduire le bruit impulsionnel (sel et poivre) tout en préservant les bords. Il remplace chaque pixel par la valeur médiane de ses voisins dans une fenêtre (par exemple 3x3). Contrairement aux filtres linéaires, qui calculent une moyenne, le filtre médian trie les valeurs des pixels voisins et remplace le pixel central par la valeur médiane de cette liste triée. Ce processus est répété pour chaque pixel de l'image, ce qui permet de conserver les contours tout en éliminant les valeurs extrêmes causées par le bruit.

```
def median_kernel(image, size):
    pad = size // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)

    filtered_image = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i+size, j:j+size]
            filtered_image[i, j] = np.median(region)

    return filtered_image
```

On applique le filtre

```
filtered_image = median_kernel(noisy_image, 3)
```

Voici l'image avant et aprés le filtrage mediane :

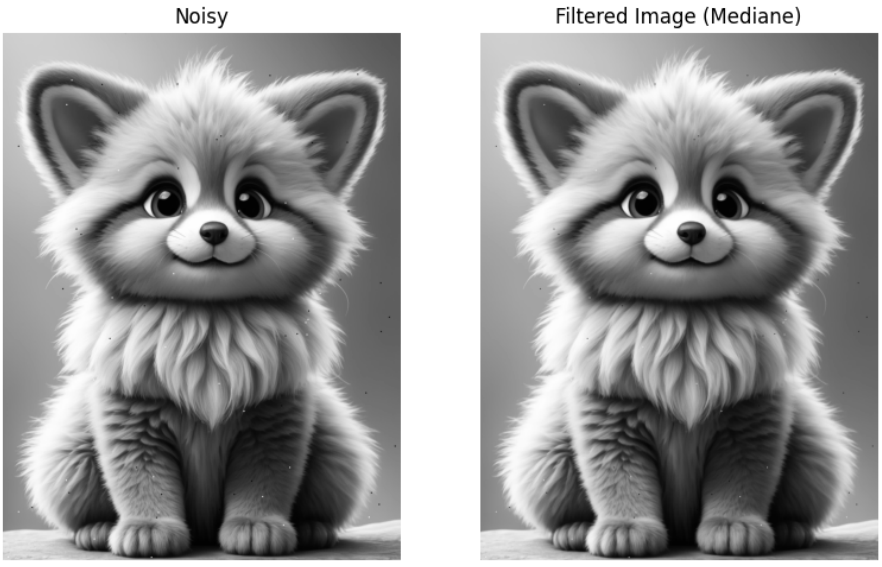

FIGURE 6 – Filtrage Mediane

3.2 Filtre Maximum

Le filtre maximum est un filtre non linéaire utilisé pour accentuer les zones les plus lumineuses d'une image. Il fonctionne en remplaçant chaque pixel par la valeur maximale de ses voisins dans une fenêtre (par exemple, 3x3). Contrairement au filtre médian qui utilise la valeur médiane, le filtre maximum sélectionne la valeur la

plus élevée parmi les pixels voisins. Ce filtre est particulièrement utile pour éliminer les petites imperfections sombres ou le bruit dans l'image, tout en préservant les zones lumineuses. Ce processus est appliqué à chaque pixel de l'image, améliorant ainsi les régions lumineuses tout en atténuant les zones sombres.

```
ef maximum_kernel(image, size):
    pad = size // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)

    filtered_image = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i+size, j:j+size]
            filtered_image[i, j] = np.max(region)

    return filtered_image
```

On applique le filtre

```
filtered_image = maximum_kernel(noisy_image, 3)
```

Voici l'image avant et aprés le filtrage maximum :

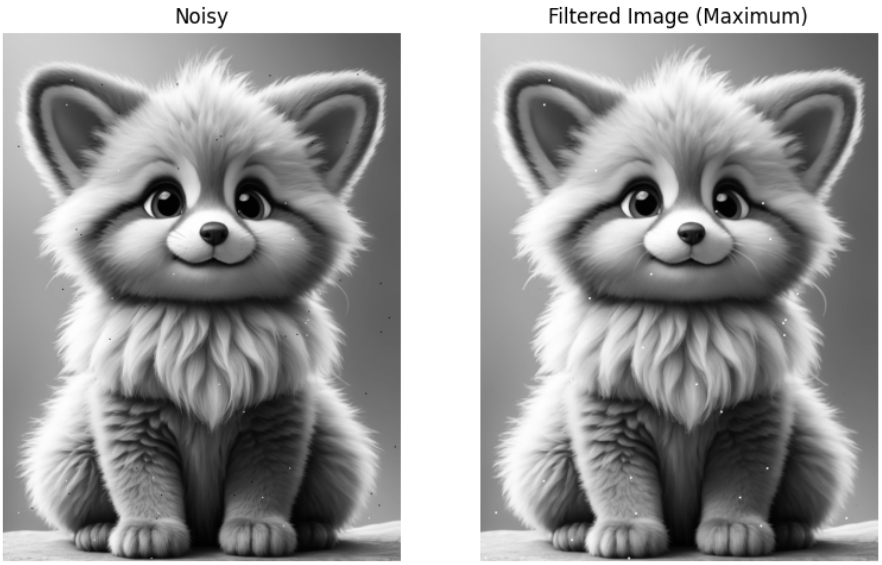

FIGURE 7 – Filtrage Maximum

4 Mesure de performance

L'Erreur Quadratique Moyenne (EQM), ou Mean Squared Error (MSE) en anglais, est une mesure de la qualité d'une image après un traitement (compression, filtrage, etc.). Elle quantifie la différence entre une image originale $I(x, y)$ et une image approximée $\hat{I}(x, y)$. Plus l'EQM est faible, plus l'image modifiée est proche de l'originale.
La formule de l'EQM est la suivante :

$$EQM = \frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} \left[I(x, y) - \hat{I}(x, y) \right]^2$$

où :

— M et N sont les dimensions de l'image (nombre de lignes et de colonnes),

— $I(x, y)$ est la valeur du pixel à la position (x, y) dans l'image originale,

— $\hat{I}(x, y)$ est la valeur du pixel à la position (x, y) dans l'image approximée.

Une faible valeur de l'EQM indique que l'image approximée est proche de l'image originale, tandis qu'une valeur élevée indique une grande différence, donc une perte de qualité importante.

```
def EQM(image_originale, image_reconstruite):

    if image_originale.shape != image_reconstruite.shape:
        raise ValueError("Taille diff")

    diff = image_originale - image_reconstruite
    eqm = np.mean(np.square(diff))

    return eqm
```

Nous appliquons cette fonction sur tous les filtres que nous avons utilisés précédemment afin de les comparer. Cela nous permet de quantifier la différence entre l'image originale et les images filtrées obtenues après l'application des filtres (moyenne, médian, binomial, gaussien, etc.).

```
eqm_valeur = EQM(G_image, filtered_image)

print("L'EQM entre les deux images est :", eqm_valeur)
```

Les résultats sont regroupés dans le tableau 1

Filtre utilise	Erreur Quadratique Moyenne
Gaussien	9.6128
Binomial	3.5026
Moyenne	5.1181
Mediane	2.6059
Maximum	29.4832

TABLE 1 – Erreur Quadratique Moyenne (EQM) pour chaque filtre.

Les résultats de l'Erreur Quadratique Moyenne (EQM) montrent que le filtre médian (2.6059) préserve le mieux l'image originale, suivi du filtre binomial (3.5026). Le filtre moyenne (5.1181) et le filtre gaussien (9.6128) modifient davantage l'image, mais restent raisonnables. Enfin, le filtre maximum (29.4832) introduit les plus grandes différences, indiquant qu'il est moins adapté pour préserver la qualité de l'image.