TP 2 - 2D Image filtering

HTI
Digital Content Protection

Marilou Grignoli
Andrea Miens

Mise en Place :

Ce TP a pour objectif de mettre en place différents filtres et de les appliquer sur les images afin de voir leur impact et leur utilité. Pour réaliser l'ensemble de ces méthodes d'encryptement, nous coderons en python en utilisant différentes bibliothèques :

- numpy
- opencv (appelé cv2 en python)
- matplotlib
- PIL
- random

Afin de réaliser nos expérience, nous avons besoin de charger les image de 'lena.jpp' et 'baboon.jpg' en niveau de gris. Le code ci-dessous nous permet de le faire :

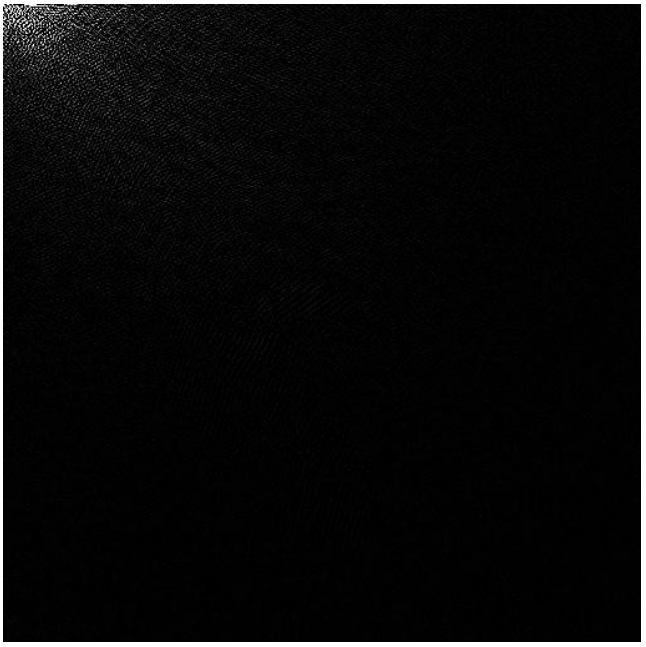
```
img = cv.imread('./lena.jpg', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('./baboon.jpg', cv.IMREAD_GRAYSCALE)

lena = np.array(img)
baboon = np.array(img2)
```

Example 1: Compute the 2D-DCT for the images

Cet exemple nous permet de mettre en place la DCT sur les images. Voici le code associé:

```
def dct2d(image):
    image1= np.float32(image)
    img = cv.dct(image1)
    return(img)
```

En appliquant notre fonction à *'lena.jgp'* on obtient l'image ci-dessous:



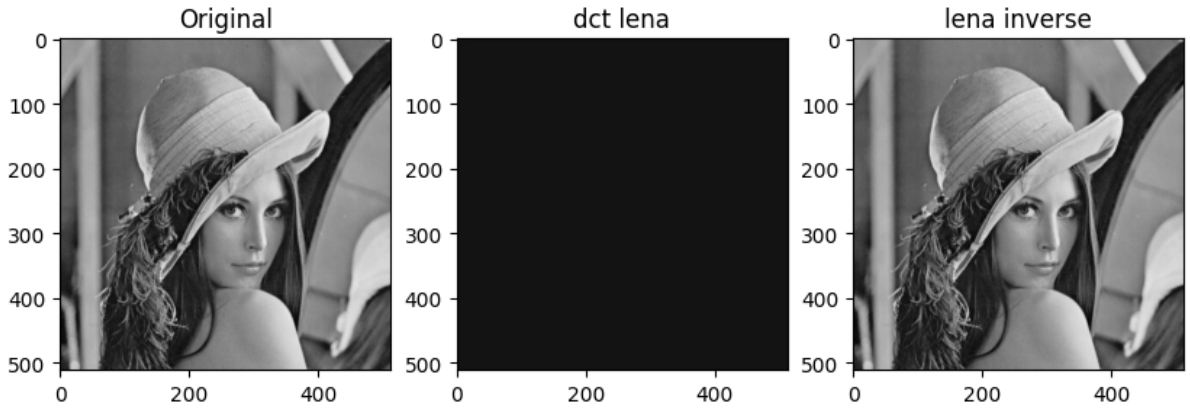
On peut remarquer qu'il y a de nombreuses tâches blanches dans le coin haut gauche. Chacune de ces tâches correspond à une fréquence spatiale obtenue grâce à la 2D-DCT. On remarque que notre image possède très peu de haute fréquence car l'angle en bas à droite est très sombre alors que le coin haut gauche est rempli de points blancs, ce qui montre la présence des basses fréquences. Ceci s'explique par notre image qui a peu de détails fins. On a un fond avec des décors uniformes, Lena a peu de texture, etc… Néanmoins on observe également des hautes fréquences à plus faibles valeurs qui doivent probablement correspondre à la plume de son chapeau qui possède de nombreux détails.

Example 2: Compute the 2D-IDCT for the images

Voici le code qui nous permet de mettre en place la transformée inverse de la 2D-DCT:

```python
def idct2d(image):
    image1= np.float32(image)
    img = cv.idct(image1)
    return(img)
```

Voici l'affichage de l'originale et la transformée inverse de lena_dct:



Pour déterminer les écarts au sein des deux matrices de l'originale et de celle après la DCT, nous appliquons ce code :

```python
def substraction(img):
    lenght = np.size(img, 0)
    width = np.size(img, 1)
    img_dct = inversedct(dct(img))
    img_new = np.zeros((lenght, width))
    max = img[0][0]-img_dct[0][0]
    min = img[0][0]-img_dct[0][0]
    for x in range(0, lenght):
        for y in range(0, width):
            img_new[x][y] = np.abs(img[x][y]-img_dct[x][y])
            if img_new[x][y] > max:
                max = img_new[x][y]
            if img_new[x][y] < min:
                min = img_new[x][y]
    return max, min

print(substraction(chargement(path2)))
```

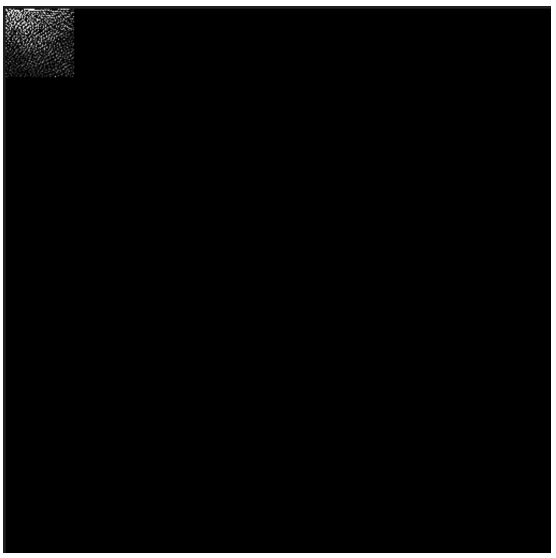
En appliquant, ce code on obtient un couple de valeur : (9.1552734375e-05, 0.0)
Ce couple correspond à l'écart maximum et minimum entre les deux matrices. Ce léger écart maximum est facilement compréhensible car nous avons des nombres flottants codés sur 32 bits or certaines valeurs ont besoin de plus de 32 bits pour être codées entièrement. Python réalise donc une troncature. Cette légère troncature provoque un décalage lorsque l'on applique la transformée inverse d'où le léger écart entre la matrice originale et la matrice après transformation.

Example 3: Study the visual impact of low-pass filtering, fc = 64

A partir de maintenant, nous allons réaliser une troncature sur le spectre obtenu grâce à la DCT. L'idée est de regarder les dégradations sur l'image lorsqu'on les soustrait du spectre.

```
def dctLp(image, fc):
    image_z = np.zeros(image.shape)
    T=image.shape[0]
    for x in range (fc):
        for y in range (fc):
                image_z[x][y] = image[x][y]
    img= np.float32(image_z)
    return(img)
```

Cet algorithme permet de mettre les coefficients à 0 des fréquences à fc. C'est ce qui explique que visuellement on observe un carré possédant des points blancs et que le restant du cube semble très noire en comparaison.



En appliquant la transformée inverse à notre DCT ayant subi un filtre passe-bas (fc=64), on obtient notre image de gauche.
On remarque qu'on conserve plutôt bien le fond de notre image (même s'il est légèrement plus flou) car il correspond à des fréquences basses. En revanche, les détails du chapeau et de la plume sont beaucoup impactés par notre filtre. Le chapeau et la plume paraissent beaucoup plus flous et uniformes que sur l'image d'origine.

Example 4: Study the visual impact of low-pass filtering, as a function of fc

Voici l'évolution de l'image Lena lorsqu'on applique la transformée inverse au DCT ayant subi un filtre passe-bas pour différentes fréquence:

fc=4

fc=8

fc=16

fc=32

fc=64

fc=128

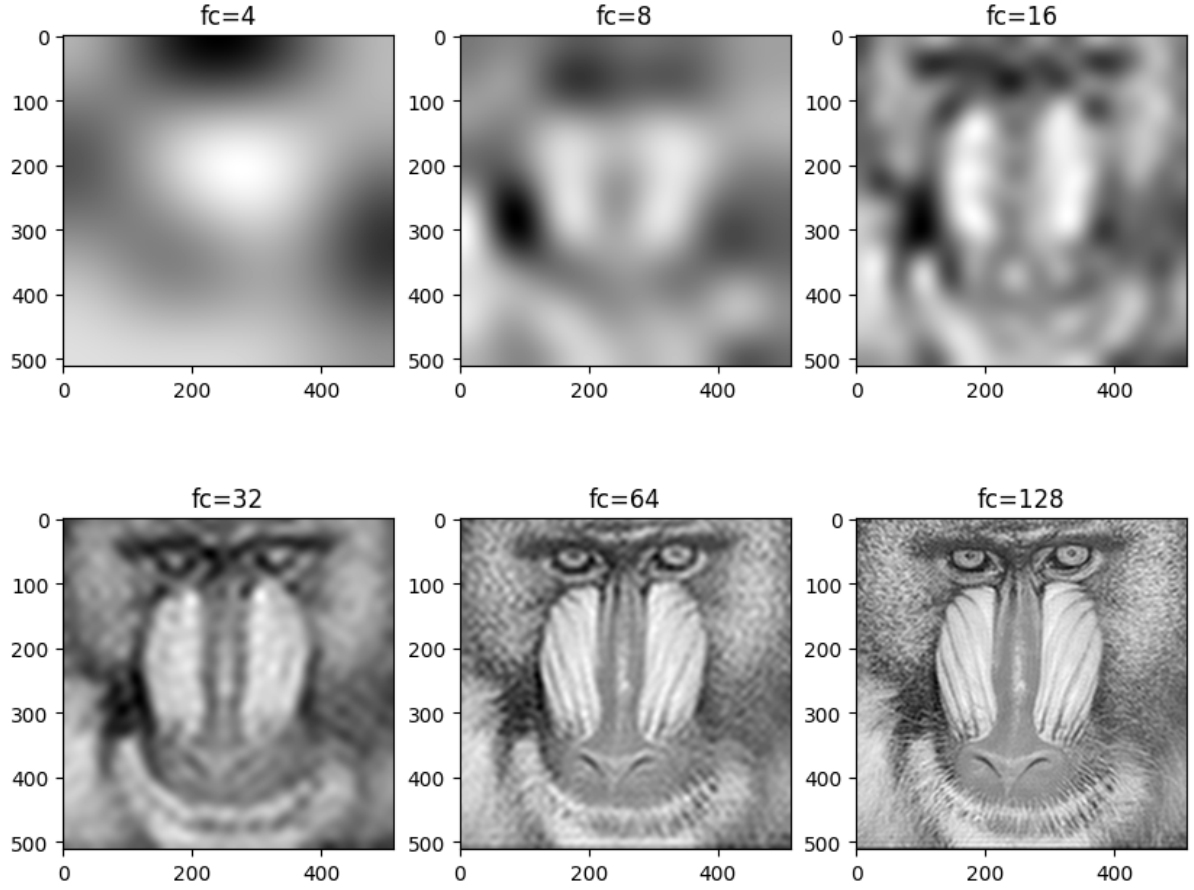
fc=256

A chaque fois que l'on augmente la frontière des fréquences que l'on garde pour la reconstruction, on voit les structures fines réapparaître petit à petit car elles correspondent aux hautes fréquences.

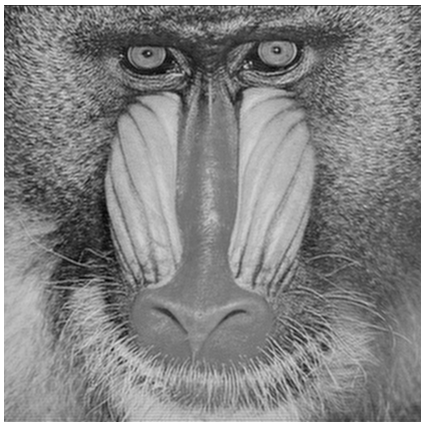
Néanmoins, la structure de l'image devient reconnaissable à partir fc=16 ou fc=32 car le fond est assez uniforme et donc représente des fréquences basses qui réapparaissent très rapidement lorsque l'on augmente la valeur de filtrage.

Example 5: Study the visual impact of low-pass filtering, as a function of fc and for different images

On applique la transformée inverse au DCT de baboon ayant subi un filtre passe-bas pour différentes fréquences.



fc=256

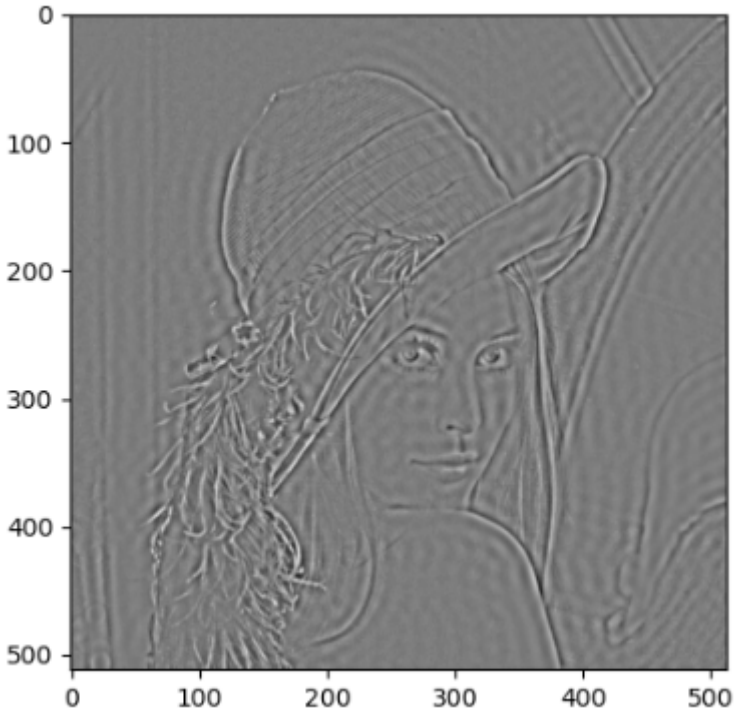


Étant donné que nous utilisons un filtre passe-bas, nous préservons principalement les basses fréquences. Cependant, le babouin est une texture très détaillée, ce qui signifie qu'il contient des fréquences plus élevées. Par conséquent, en appliquant ce filtre, nous éliminons rapidement les informations détaillées du babouin. Ainsi l'image baboon apparaît plus clairement pour une fréquence fc plus élevée que pour lena.

Example 6: Study the visual impact of high-pass filtering, fc = 64

On considère maintenant un filtre qui enlève les fréquences inférieures à la fréquence de référence. En utilisant le code des questions précédentes, on a:

```
def selectif(img, fc):
    lenght = np.size(img, 0)
    width = np.size(img, 1)
    img_new = np.zeros((lenght, width))
    for x in range(0, fc):
        for y in range(0, fc):
            img_new[x][y] = img[x][y]
    return img_new

def select_high(img_dct, frc):
    img_new = img_dct - selectif(img_dct, frc)
    return img_new
```



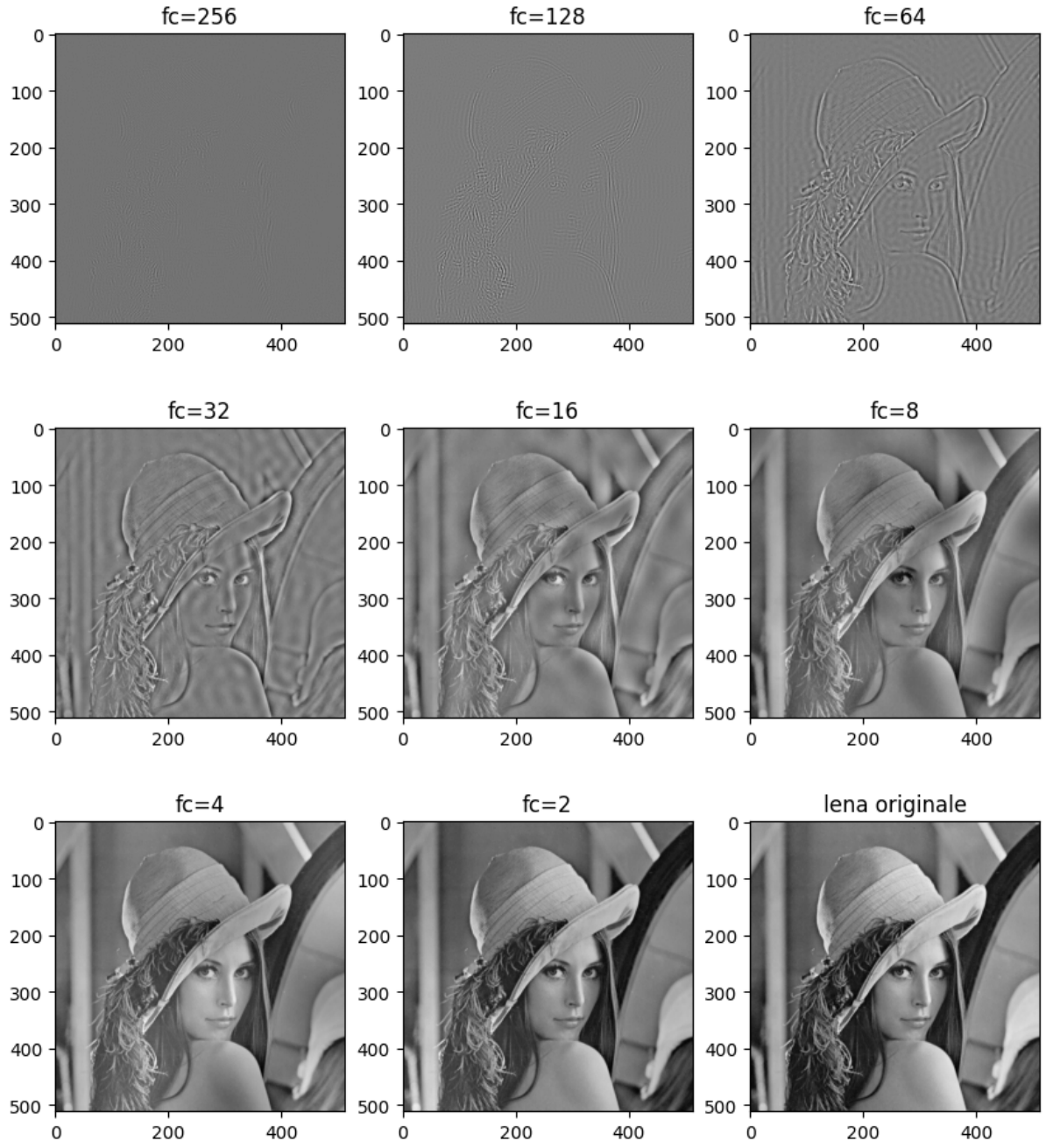
En appliquant la transformée inverse à notre DCT ayant subi un filtre passe-haut (fc=64), on obtient l'image ci-dessus.

On remarque qu'on conserve très mal le fond de notre image car il correspond à des fréquences basses. En revanche, les détails du chapeau et de la plume et les contours de Léna sont beaucoup moins impactés par notre filtre. Toutes ces zones apparaissent bien plus car ce sont des zones très texturées ou très marquées et donc des fréquences plus hautes élevées.

Nous pouvons discerner des fréquences élevées dans les textures et conserver le premier coefficient à zéro, ce qui correspond à la composante continue.

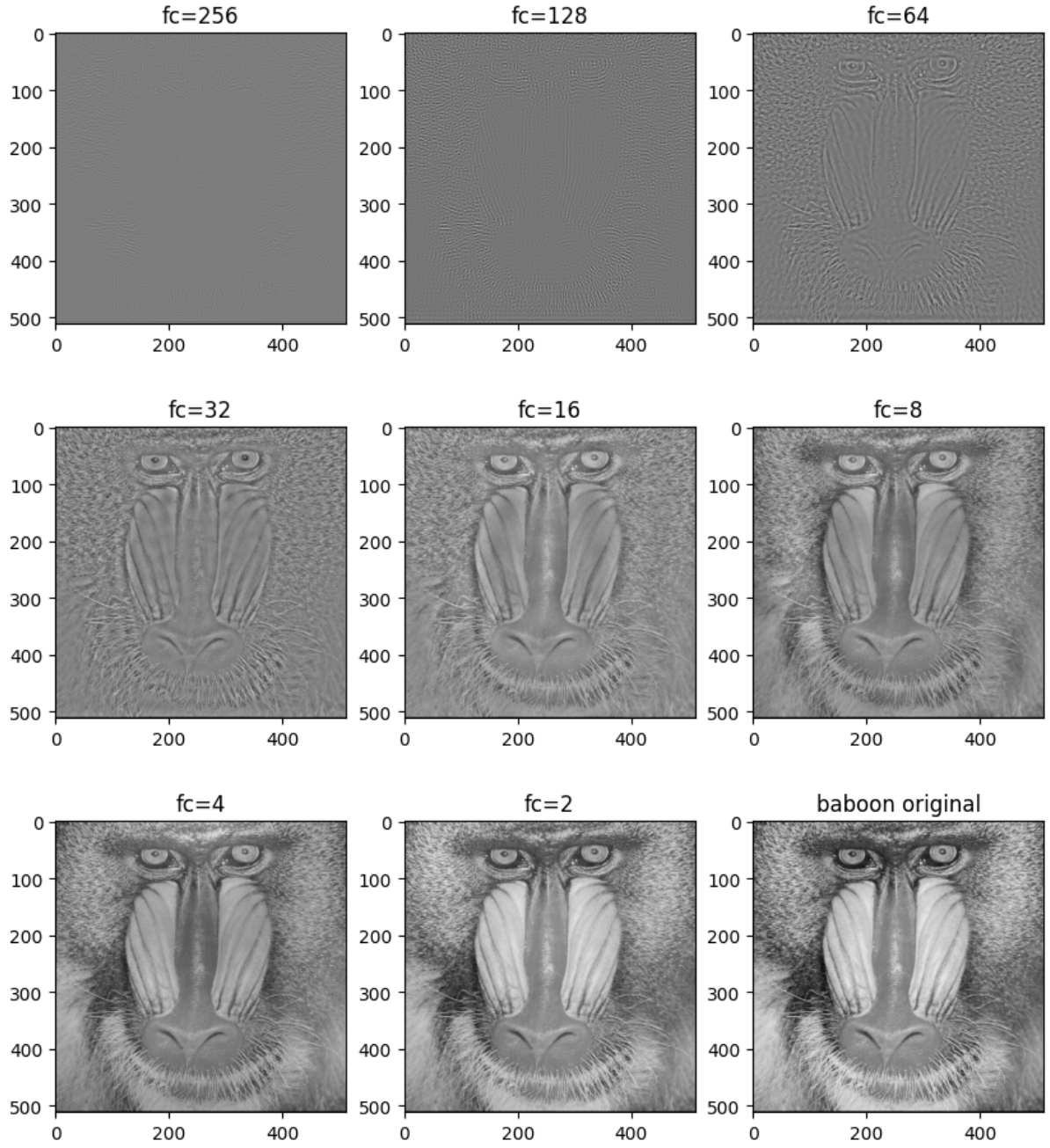
Les zones qui ont disparues sont les zones plutôt uniforme et faible en fréquence

Example 7: Study the visual impact of high-pass filtering, as a function of fc



A chaque fois que l'on augmente la frontière des fréquences pour la reconstruction, on voit les structures fines disparaître petit à petit car elles correspondent aux basses fréquences.

Example 8: Study the visual impact of high-pass filtering, as a function of fc and for different images



Étant donné que nous utilisons un filtre passe-haut, nous préservons principalement les hautes fréquences. Le babouin est une texture très détaillée, ce qui signifie qu'il contient des fréquences plus élevées. Par conséquent, en appliquant ce filtre, nous gardons plus les informations détaillées du babouin.

Example 10: Illustrating how the spatial DCT frequencies are related to the visual content

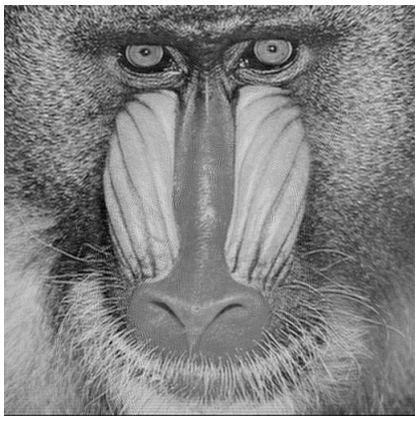
Dans cet exemple, nous souhaitons remplacer une partie des fréquences de l'une des images par celles de l'autre. On va donc utiliser les fonctions réalisées ci-dessus pour construire notre nouvelle image en concaténant le résultat du filtre du passe-bas et du filtre passe-haut. Cette fonction qui mixe les images est décrites ci-dessous :

```
def selectif(img, fc):
    lenght = np.size(img, 0)
    width = np.size(img, 1)
    img_new = np.zeros((lenght, width))
    for x in range(0, fc):
        for y in range(0, fc):
            img_new[x][y] = img[x][y]
    return img_new

def select_high(img_dct, frc):
    img_new = img_dct - selectif(img_dct, frc)
    return img_new

def mixeur(dct1, dct2, fc):
    imgl = selectif(dct1, fc)
    imgh = select_high(dct2, fc)
    img_mixed = imgl + imgh
    return img_mixed
```

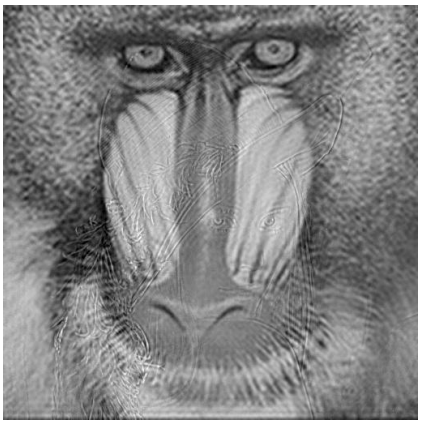
Cette fonction prend en entrée les DCT des images à mixer. Nous commencerons pour le mixage de Lena vers Baboon, c'est-à-dire que l'image initiale sera Léna.

Cas 1 : Remplacement des fréquences de Baboon par celle de léna

fc= 256



fc = 128



fc= 64

fc = 32

fc = 16

fc= 8

fc = 4

fc = 2

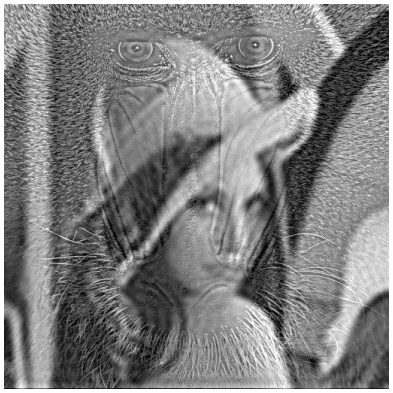
Dans ce premier cas, on remplace petit à petit les hautes fréquences de baboon par celle de Léna. Léna possède moins de hautes fréquences de baboon, on voit donc apparaître clairement Léna à partir de fc = 64. A partir de fc = 16, on voit très distinctement léna bien que l'on conserve les basses fréquences de baboon (visible par les couleurs de baboon superposées aux contours de léna. Enfin dans le cas fc = 2, on voit encore légèrement des altérations des couleurs de léna par rapport à celle baboon.

<u>Cas 2 :</u> Remplacement des fréquences de Léna par celle de Baboon

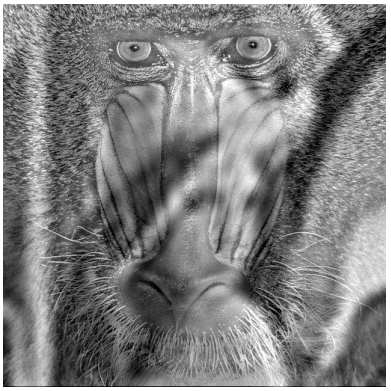
fc = 256

fc = 128

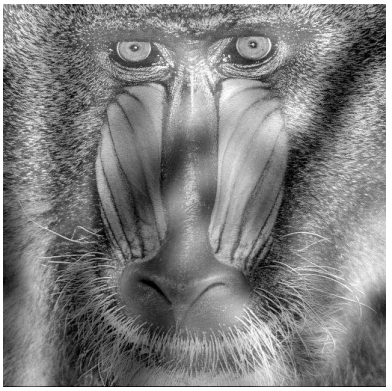
fc=64



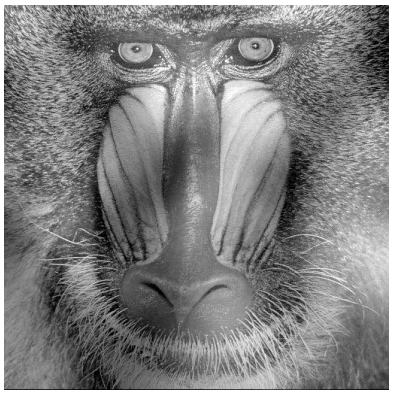
fc=32



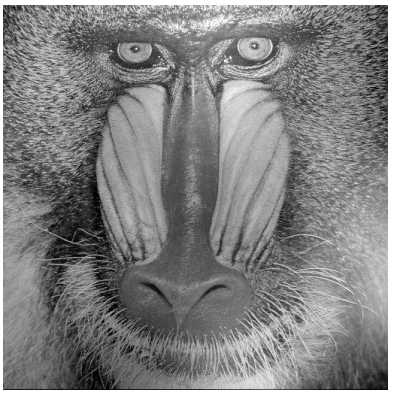
fc=16



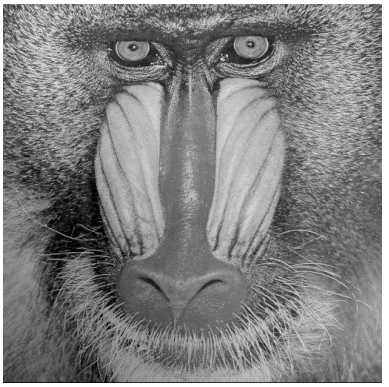
fc= 8



fc = 4



fc = 2



Dans ce second cas, on remplace petit à petit les hautes fréquences de Léna par celle de Baboon. Baboon possède beaucoup de hautes fréquences, on voit donc clairement apparaître les traits de Baboon à partir de fc = 32. A partir de fc = 16, on voit très distinctement baboon bien que l'on conserve les basses fréquences de léna (visible par les couleurs de léna superposées aux contours de babon). Enfin dans le cas fc = 2, on voit encore légèrement des altérations des couleurs de léna par rapport à celle baboon.