Traitement des Images

September 3, 2023

1 Traitement des images

Les images sur python sont représenté par des matrices. Chaque element de la matrice représente un pixel. - Pour les images noir et blanc les valeurs de la matrice seront soit 0 soit 1 ; - Pour les images nuances gris, les valeurs de la matrice seront compris entre 0 (pour le noir) et 255 (pour le blanc) ; - Pour les images couleurs on suit le code RGB. Dans ce cas chaque element de la matrice est une liste de 3 valeurs comprises entre 0 et 255, la première pour le rouge, la deuxième pour le vert et la troisieme pour le bleu. Dans ce cas [0,0,0] représente le noir et [255,255,255] représente le blanc.

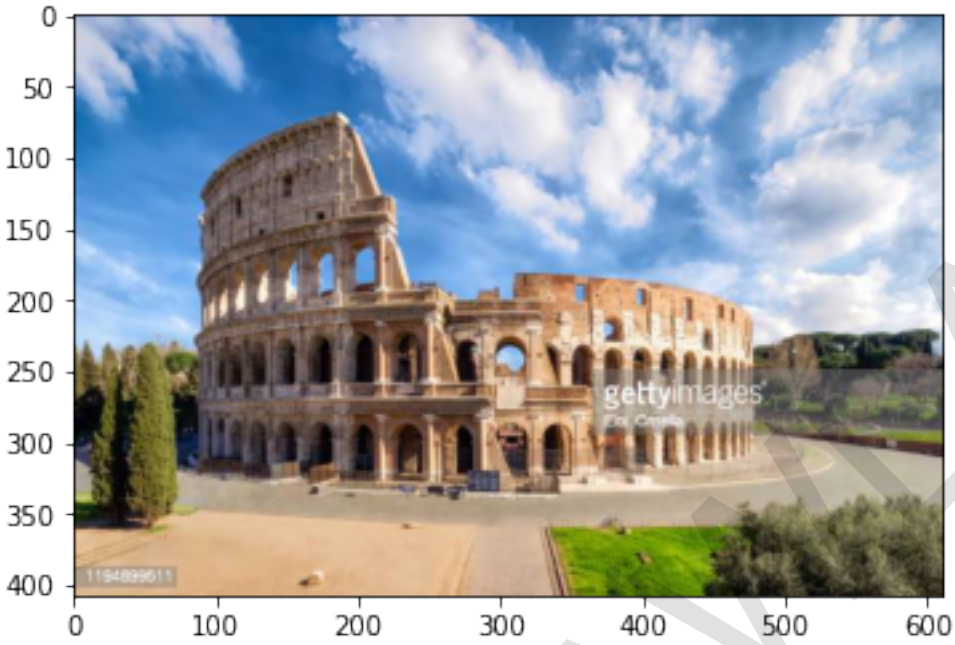
[ ]:
```
# Importer les bibliothèques
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
# La fonction imread importé de la bibliothèque matplotlib.pyplot permet de␣
↪transformer une image en un tableau array
image = plt.imread("Pic.jpg")
print(image)
```

```
[[[161 185 213]
  [160 184 212]
  [158 181 212]
  …
  [103 137 182]
  [106 140 185]
  [109 143 188]]

 [[164 188 216]
  [162 186 214]
  [162 183 212]
  …
  [107 141 186]
  [110 144 189]
  [113 147 192]]

 [[170 191 220]
  [169 190 219]
  [166 187 216]
```

```
 …
 [113 147 192]
 [116 150 195]
 [118 152 197]]

…

[[221 184 140]
 [222 185 141]
 [222 185 141]
 …
 [ 62  57  38]
 [ 72  67  48]
 [ 81  76  57]]

[[217 178 135]
 [217 178 135]
 [217 178 135]
 …
 [ 62  57  38]
 [ 77  72  53]
 [ 87  82  63]]

[[224 182 142]
 [224 182 142]
 [224 182 142]
 …
 [ 39  34  15]
 [ 55  50  31]
 [ 64  59  40]]]
```

[ ]:
```
# La fonction imshow() qui permet d'afficher une image stocké sur python sous␣
↪forme d'un tableau array
plt.imshow(image)
plt.show()
```



[ ]:
```
# La forme d'une image
image.shape
```

[ ]: (408, 612, 3)

[ ]:
```
# Le nombre de pixels d'une image couleur
s=image.shape
s[0]*s[1]
```

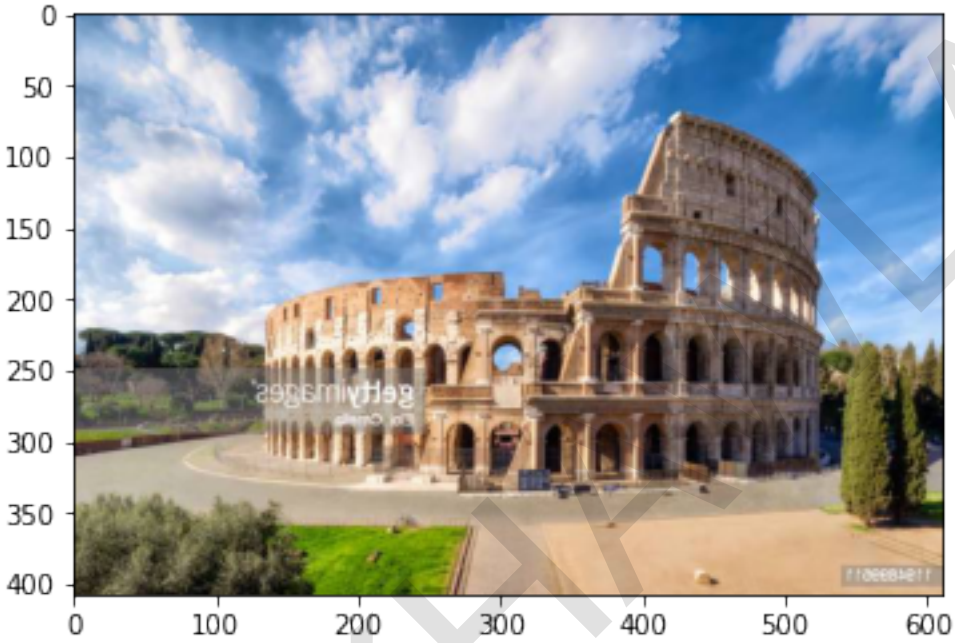
[ ]: 249696

1.1 Exercice 1 :

Ecrire une fonction symetriev(im) qui prend en paramètre une image im sous forme d'une matrice numpy et qui retourne une matrice numpy qui représente la symétrie de im autour de l'axe vertical passant par le milieu de l'image.

[ ]:
```
def symetriev(im):
  l,c,d=im.shape
  nim=np.zeros((l,c,d),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=im[i,-j-1]
  return nim
```

[ ]:
```
# Affichage de la symetrie de l'image selon l'axe verticale qui passe au milieu
↪de l'image
simv=symetriev(image)
plt.imshow(simv)
plt.show()
```

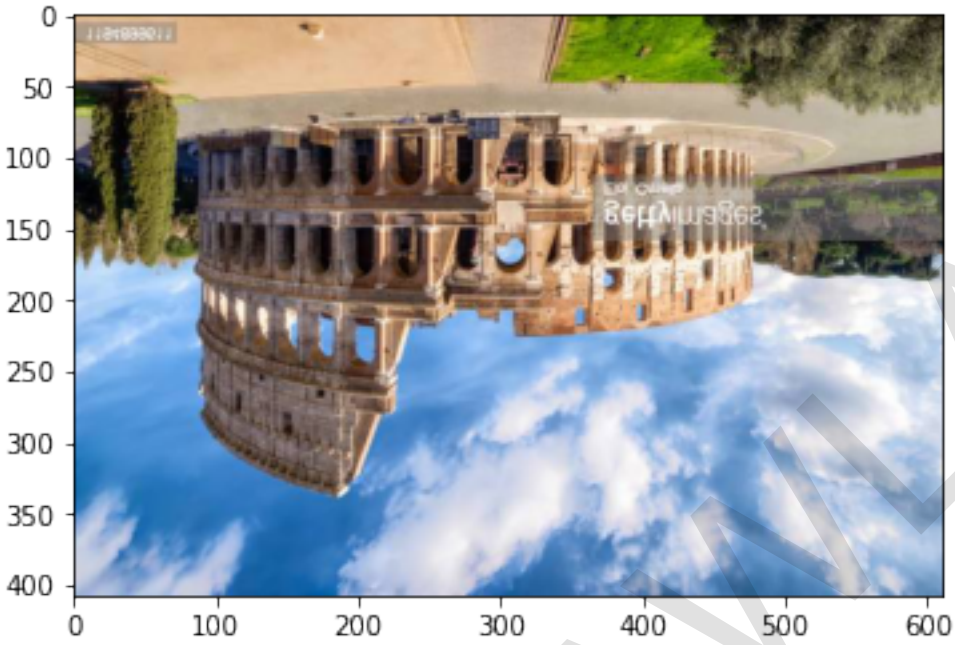


1.2 Exercice 2 :

Ecrire une fonction symetrieh(im) qui prend en paramètre une image im sous forme d'une matrice numpy et qui retourne une matrice numpy qui représente la symétrie de im autour de l'axe horizontal passant par le milieu de l'image.

[ ]:
```
def symetrieh(im):
  l,c,d=im.shape
  nim=np.zeros((l,c,d),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=im[-i-1,j]
  return nim
```

[ ]:
```
# Affichage de la symetrie de l'image selon l'axe verticale qui passe au milieu
↪de l'image
simh=symetrieh(image)
plt.imshow(simh)
plt.show()
```

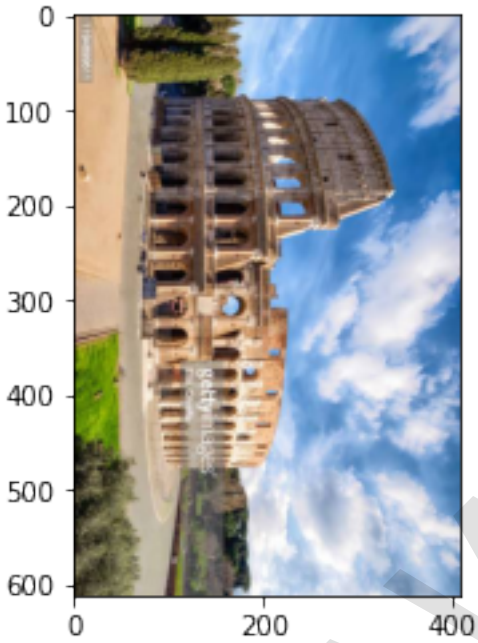


1.3 Exercice 3 :

Ecrire une fonction rotation(im) qui prend en paramètre une image im sous forme d'une matrice numpy et qui retourne une matrice numpy qui représente la rotation d'un angle pi/2 de im autour du centre de l'image dans le sens horaire.

[ ]:
```
def rotation(im):
  l,c,d=im.shape
  nim=np.zeros((c,l,d),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[j,-i-1]=im[i,j]
  return nim
```

[ ]:
```
# Affichage de la rotation de l'image d'un angle pi/2 autour du centre de␣
↪l'image dans le sens horaire.
rim=rotation(image)
plt.imshow(rim)
plt.show()
```

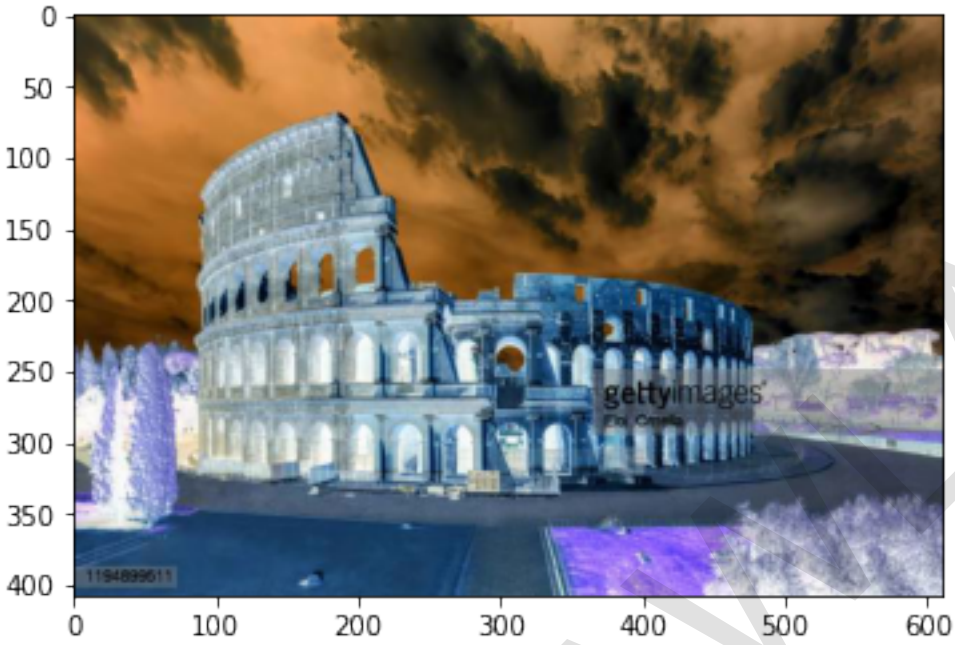


1.4 Exercice 4 :

Ecrire une fonction negatif(im) qui prend en paramètre une image im sous forme d'une matrice numpy et qui retourne une matrice numpy qui représente le négatif de l'image. Le négatif d'un pixel [R,G,B] est le pixel [255,255,255] - [R,G,B].

[ ]:
```
def negatif(im):
  l,c,d=im.shape
  nim=np.zeros((l,c,d),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=np.array([255,255,255])-im[i,j]
  return nim
```

[ ]:
```
# Affichage du négatif de l'image
negim=negatif(image)
plt.imshow(negim)
plt.show()
```



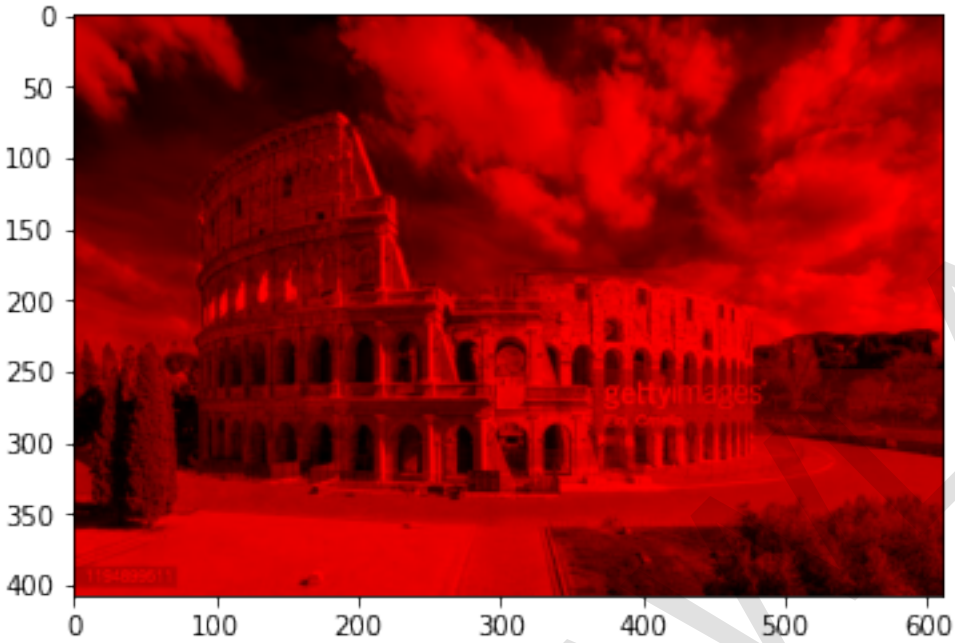
2 Afficher la couche Rouge de l'image

2.1 Exercice 5 :

Ecrire une fonction qui renvoie le canal rouge de l'image

[ ]:
```
def couche_rouge(im):
  l,c,d=im.shape
  nim=im.copy()
  for i in range(l):
    for j in range(c):
      nim[i,j,1]=0
      nim[i,j,2]=0
  return nim
```

[ ]:
```
# Afficher la couche rouge
red_layer = couche_rouge(image)
plt.imshow(red_layer)
plt.show()
```



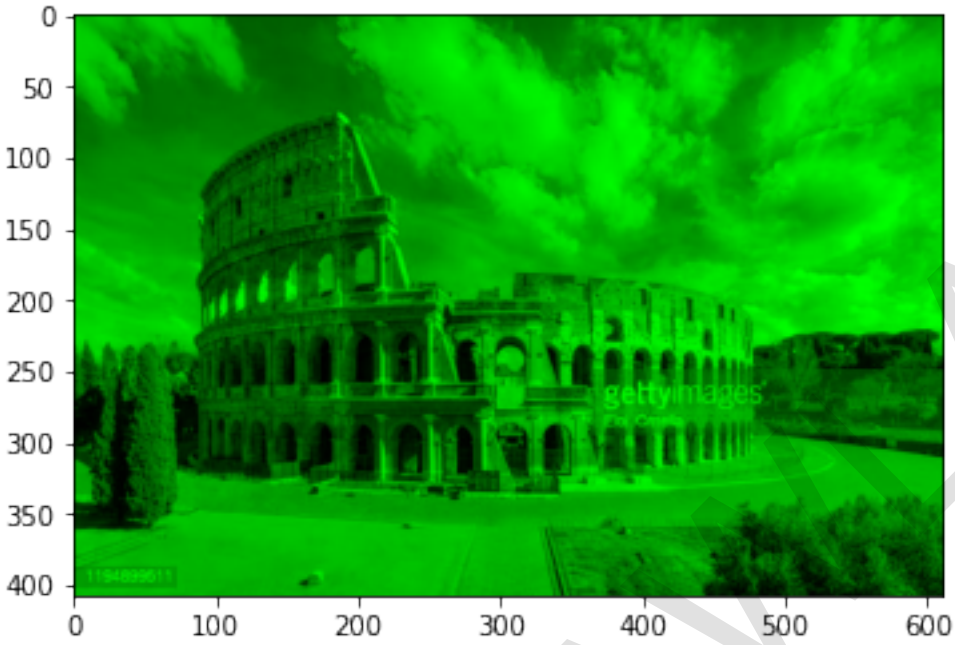
3 Afficher la couche Verte de l'image

3.1 Exercice 6 :

Ecrire une fonction qui renvoie le canal vert de l'image

[ ]:
```
def couche_verte(im):
  l,c,d=im.shape
  nim=im.copy()
  for i in range(l):
    for j in range(c):
      nim[i,j,0]=0
      nim[i,j,2]=0
  return nim
```

[ ]:
```
# Afficher la couche verte
green_layer = couche_verte(image)
plt.imshow(green_layer)
plt.show()
```



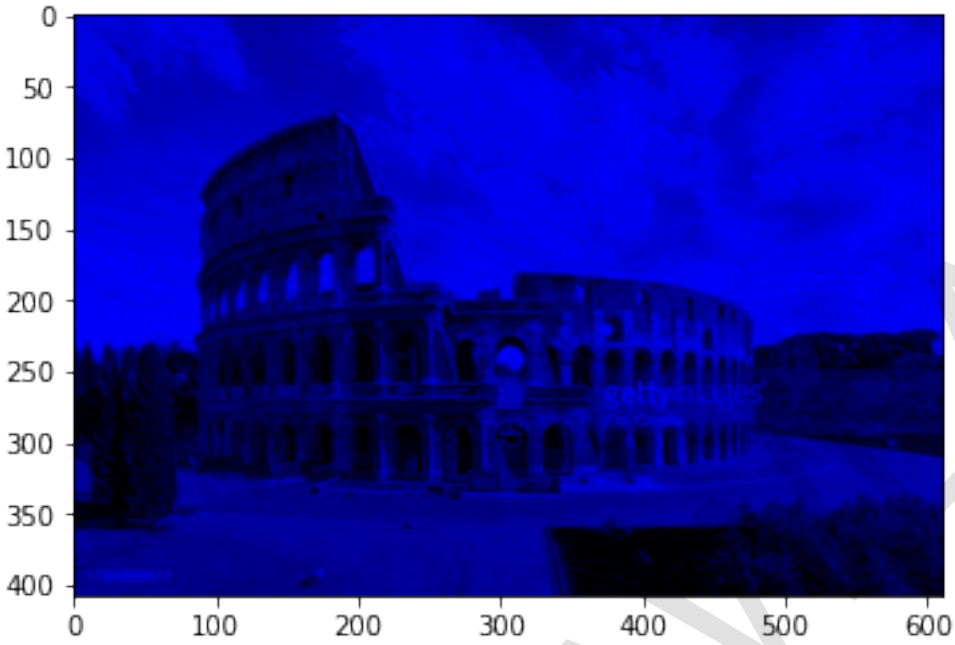
4 Afficher la couche Bleue de l'image

4.1 Exercice 7 :

Ecrire une fonction qui renvoie le canal bleu de l'image

[ ]:
```
def couche_bleue(im):
  l,c,d=im.shape
  nim=im.copy()
  for i in range(l):
    for j in range(c):
      nim[i,j,0]=0
      nim[i,j,1]=0
  return nim
```

[ ]:
```
# Afficher la couche bleue
blue_layer = couche_bleue(image)
plt.imshow(blue_layer)
plt.show()
```

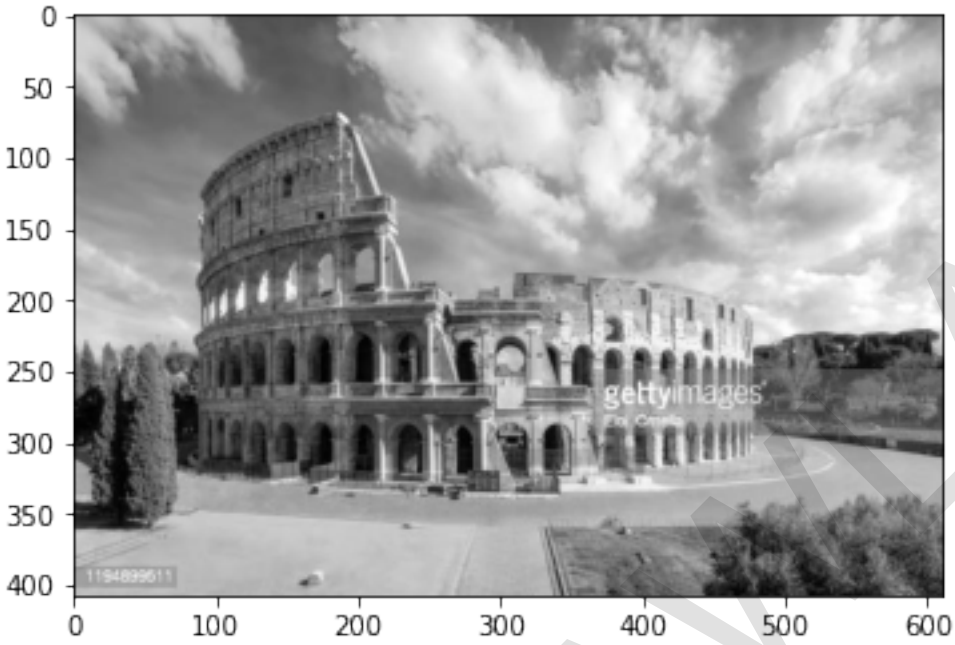


5 Conversion en niveau de gris

Conversion en niveau de gris : Pour convertir une image couleur en niveau de gris, un pixel représenté par ses composantes (r,g,b) doit être remplacé par un pixel à une seule composante y [0,255] appelée luminance. Cette quantité, qui traduit la sensation visuelle de luminosité, dépend de manière inégale des trois composantes RGB. La formule communément recommandée pour cette conversion est la suivante : y = 0,2126 r + 0,7152 g + 0,0722 b (y étant arrondi à l'entier le plus proche) (pour un œil humain le vert parait plus lumineux que le rouge, lui-même plus lumineux que le bleu).

[ ]:
```
def conversion_gris(im):
  l,c,d=im.shape
  nim=np.zeros((l,c),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=round(0.2126*im[i,j,0] + 0.7152*im[i,j,1] + 0.0722*im[i,j,2])
  return nim
```

[ ]:
```
# Convertir l'image en gris
gim=conversion_gris(image)
plt.imshow(gim,cmap='gray')
plt.show()
```



6 Histogramme d'une image

L'histogramme d'une image permet de compter le nombre de pixels d'un niveau de gris donné.

6.1 Exercice 8 :

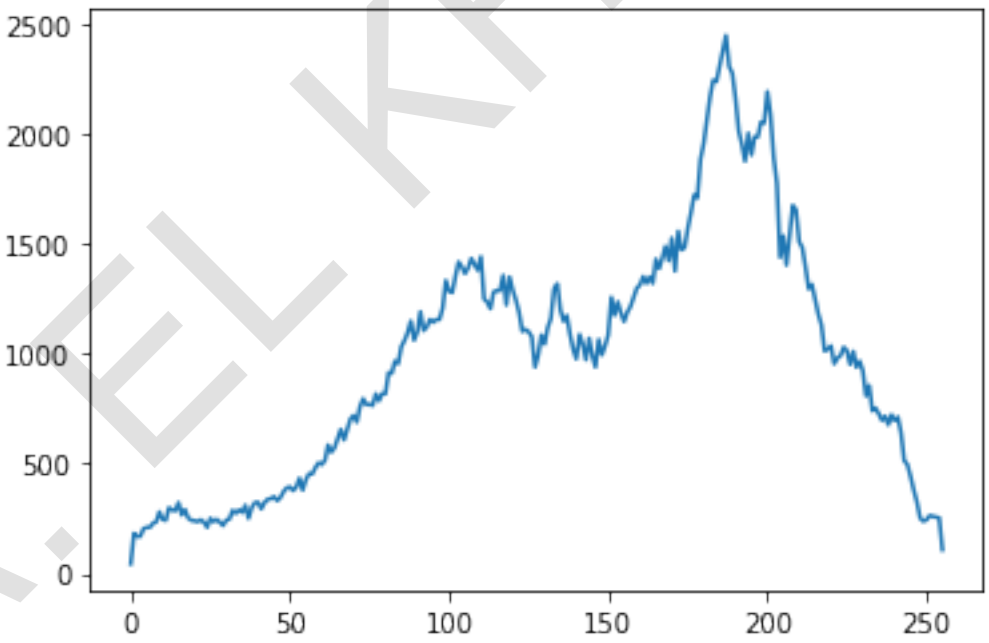
Ecrire une fonction histo(im), qui prend en argument une image en niveau de gris (décrite par une matrice dont chaque pixel est représenté seulement par le niveau de gris). Cette fonction doit renvoyer une liste de taille 256 : en première position (indice 0), le nombre de pixels noirs (gris 0), en deuxième position (indice 1), le nombre de pixels gris 1, . . . , en dernière position (255), le nombre de pixels blancs (gris 255).

[ ]:
```
def histo(im):
  H=np.zeros(256,dtype=int)
  l,c=im.shape
  for i in range(l):
    for j in range(c):
      gris = im[i,j]
      H[gris] = H[gris] + 1
  return H
```

[ ]:
```
# Calculer l'histogramme de l'image
image_gris = conversion_gris(image)
histogramme = histo(image_gris)
print(histogramme)
```

```
[  45  183  170  170  205  210  211  229  236  281  247  243  300  290
  287  323  267  292  254  243  241  237  245  234  211  252  237  247
  234  219  244  248  288  275  290  279  311  252  300  323  324  295
  323  338  341  350  332  346  373  390  394  378  394  434  380  427
  454  453  482  504  496  515  585  552  572  616  656  608  655  702
  719  689  758  795  768  770  764  817  786  818  818  914  910  966
  954 1032 1059 1096 1148 1061 1097 1190 1105 1121 1155 1142 1155 1155
 1207 1332 1283 1277 1351 1416 1394 1364 1389 1432 1405 1379 1440 1248
 1234 1204 1280 1286 1286 1355 1221 1348 1288 1243 1187 1101 1110 1099
 1076  939  993 1084 1043 1118 1160 1294 1317 1197 1147 1173 1087 1020
  976 1087 1044  972 1070  991  939 1064  994 1035 1085 1256 1174 1235
 1184 1146 1188 1214 1258 1298 1312 1350 1319 1349 1319 1429 1386 1433
 1487 1422 1525 1375 1556 1471 1481 1568 1640 1720 1707 1883 1954 2062
 2166 2242 2235 2306 2383 2445 2304 2273 2159 2011 1951 1874 2001 1902
 1984 1984 2051 2047 2190 2086 1890 1774 1437 1535 1402 1541 1673 1653
 1508 1481 1392 1294 1314 1245 1180 1130 1012 1020 1035  954  979  991
 1029 1013  951 1010  938  966  922  808  855  740  752  727  697  716
  677  722  695  712  641  514  496  442  380  326  253  239  245  264
  261  259  254  109]
```

[ ]:
```
# Dessiner l'histogramme
plt.plot(np.arange(0,256,1),histogramme)
plt.show()
```



7 Amélioration du contrast d'une image grise

Pour améliorer le contrast d'une image grise on applique une transformation T sur tous les pixels de l'image. Soit x le niveau de gris d'une pixel quelconque (0 <= x <= 255), on a :

T(x) = 255 * CDF(x)

Avec CDF, la fonction de distribution cumulative ; CDF(x) = (n0 + n1 + n2 + … + nx) / N

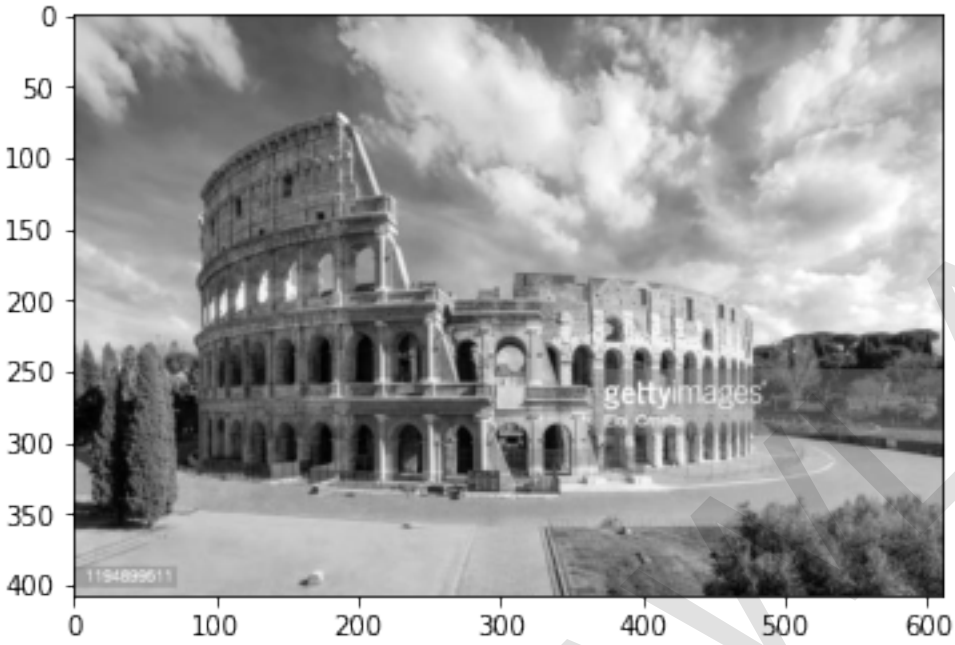
Avec nj le nombre d'occurence de gris j dans l'image (nj sera extrait de l'histogramme de l'image) et N le nombre total des pixels de l'image.

7.1 Exercice 9 :

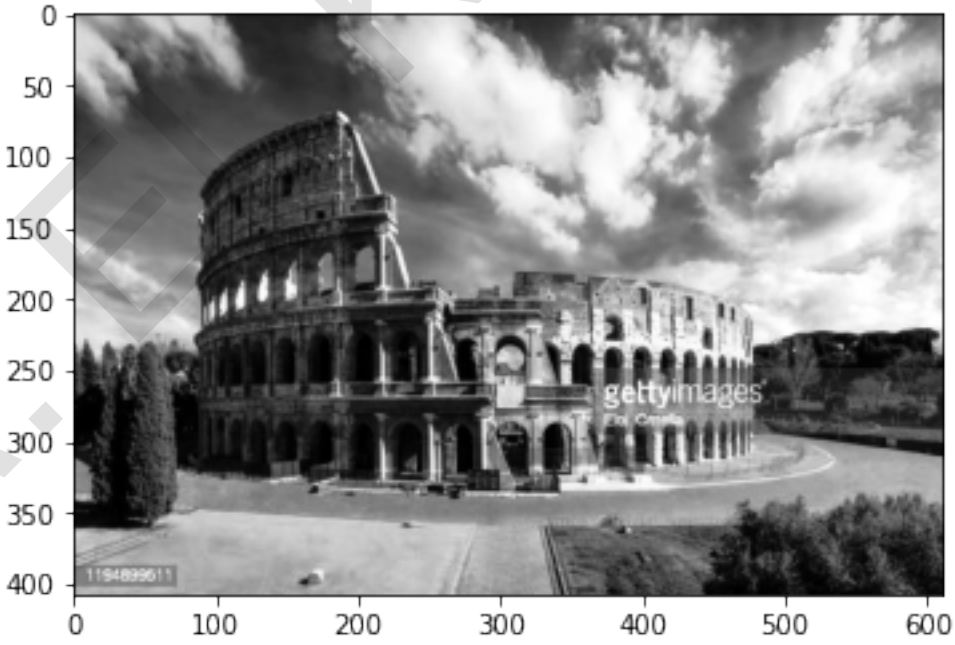
Ecrire une fonction ameliorer_contrast_gris(im) qui prend en paramètre une image grise im et qui retourne sa version amélioré

[ ]:
```
# Premièrement on définit la fonction CDF
def CDF(i,hist,im):
  s=0
  N=im.size
  for j in range(i+1):
    s=s+hist[j]
  return s/N

# Deuxièmement on définit la transformation T
def T(i,hist,im):
  return 255*CDF(i,hist,im)

# Finalement on ecrit la fonction ameliorer_contrast_gris
def ameliorer_contrast_gris(im):
  hist=histo(im)
  l,c=im.shape
  nim=np.zeros((l,c),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=T(im[i,j],hist,im)
  return nim
```

[ ]:
```
# Afficher l'image non améliorée
image_gris = conversion_gris(image)
plt.imshow(image_gris,cmap='gray')
plt.show()
```



[ ]:
```
# Afficher la nouvelle image améliorée
nouvelle_image = ameliorer_contrast_gris(image_gris)
plt.imshow(nouvelle_image,cmap='gray')
plt.show()
```



8 Application d'une matrice de convolution (Noyau) sur une image

En traitement d'images, un noyau, une matrice de convolution ou un masque est une petite matrice utilisée pour l'application des filtres sur des images tels que le floutage, l'amélioration de la netteté de l'image, le gaufrage, la détection de contours, et d'autres. Tout cela est accompli en faisant une convolution entre le noyau et l'image.

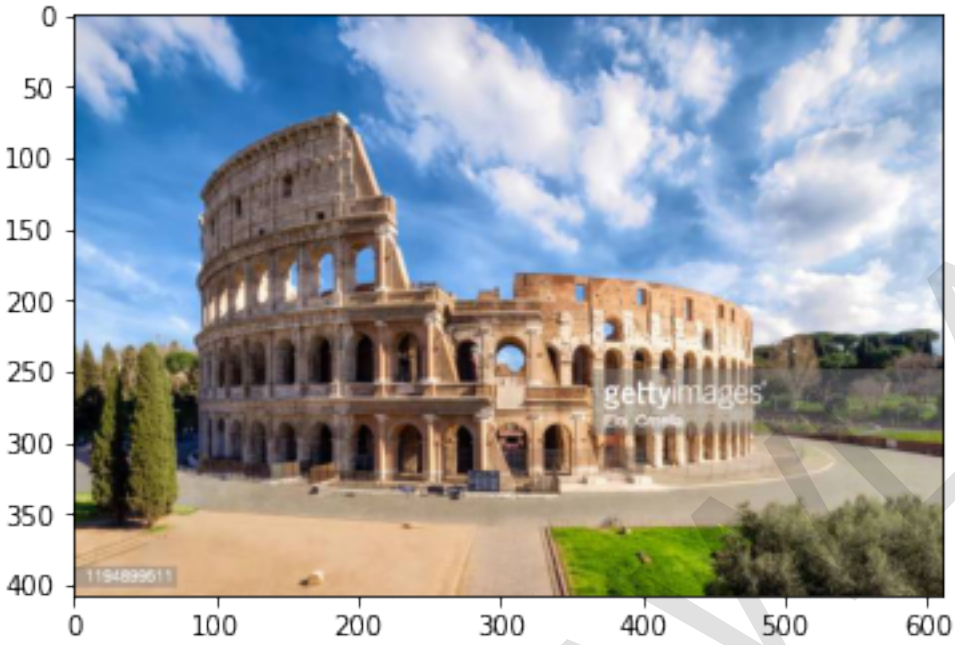
Généralement la matrice de convolution est une matrice carée de dimensions impairs (3x3, 7x7, 5x5, ….).

On s'interesse aux matrice de convolution de dimensions 3x3.

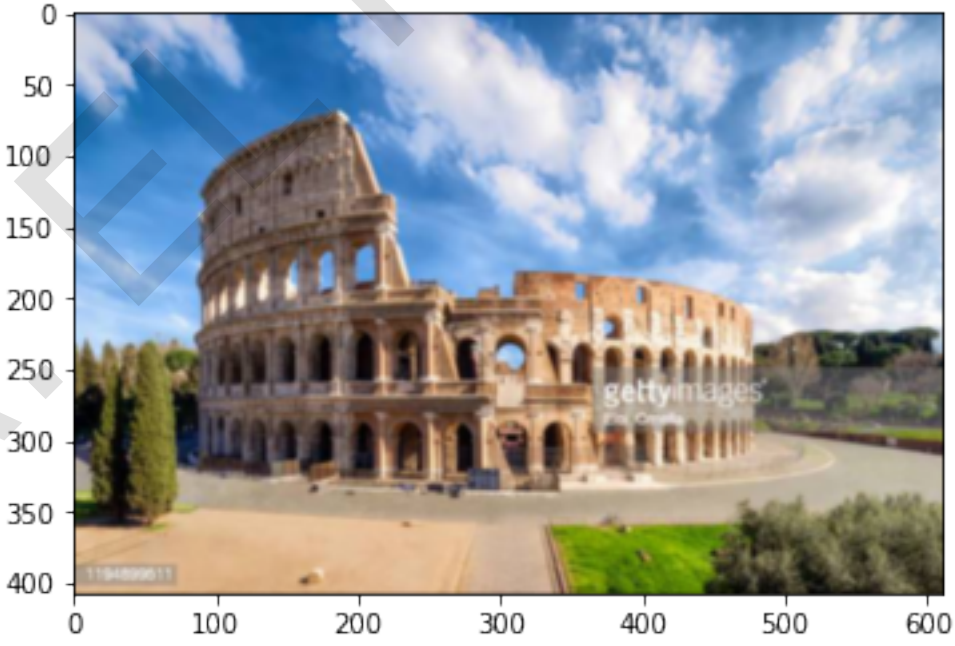
[ ]:
```
# Premièrement on rédige une fonction qui nous aide à appliquer un noyau sur un pixel (i,j) d'une image
def appliquer_noyau_pixel(N,im,i,j):
  somme_pixels = np.zeros(3,dtype=int)
  for p in range(3):
    for q in range (3):
      somme_pixels  = somme_pixels + N[p,q]*im[p+i-1,q+j-1]
  for p in range(len(somme_pixels)):
    if somme_pixels[p]<0:
      somme_pixels[p]=0
    if somme_pixels[p]>255:
      somme_pixels[p]=255
  return somme_pixels

# Finalement on crée la fonction convolution
def convolution(im,N):
  l,c,d = im.shape
  im_couverte=np.zeros((l+2,c+2,d),dtype=int)
  im_couverte[1:-1,1:-1]=im
  nim = np.zeros((l,c,d),dtype=int)
  for i in range(l):
    for j in range(c):
      nim[i,j]=appliquer_noyau_pixel(N,im_couverte,i+1,j+1)
  return nim
```

8.1 Le filtre lissage

[ ]:
```
# Afficher l'image avant le lissage
plt.imshow(image)
plt.show()
```

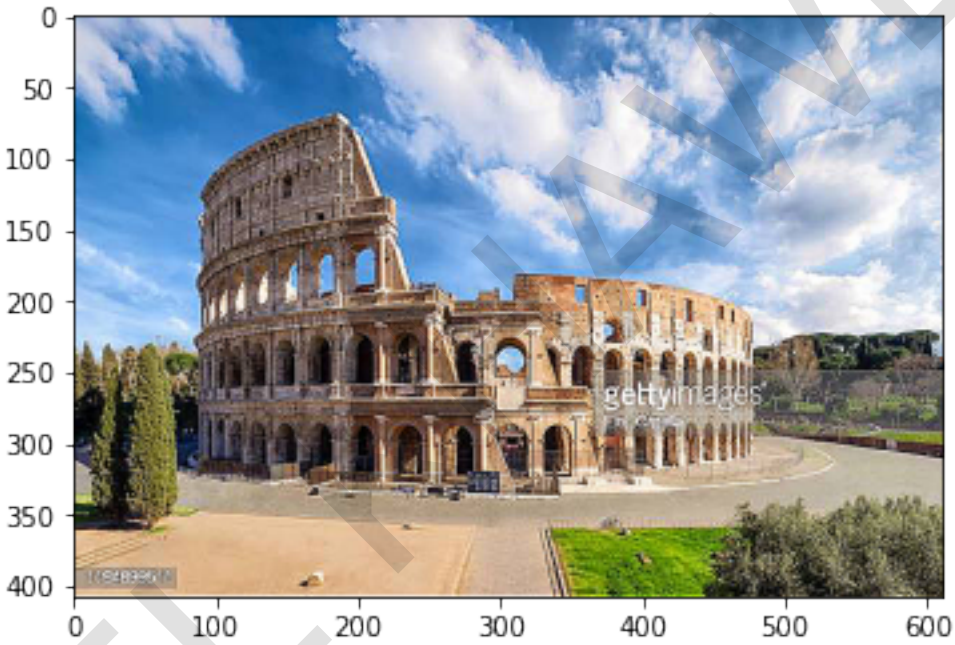


[ ]:
```
# Affiche l'image aprés le lissage
L=np.ones((3,3))*(1/9)
image_lissé = convolution(image,L)
plt.imshow(image_lissé)
plt.show()
```



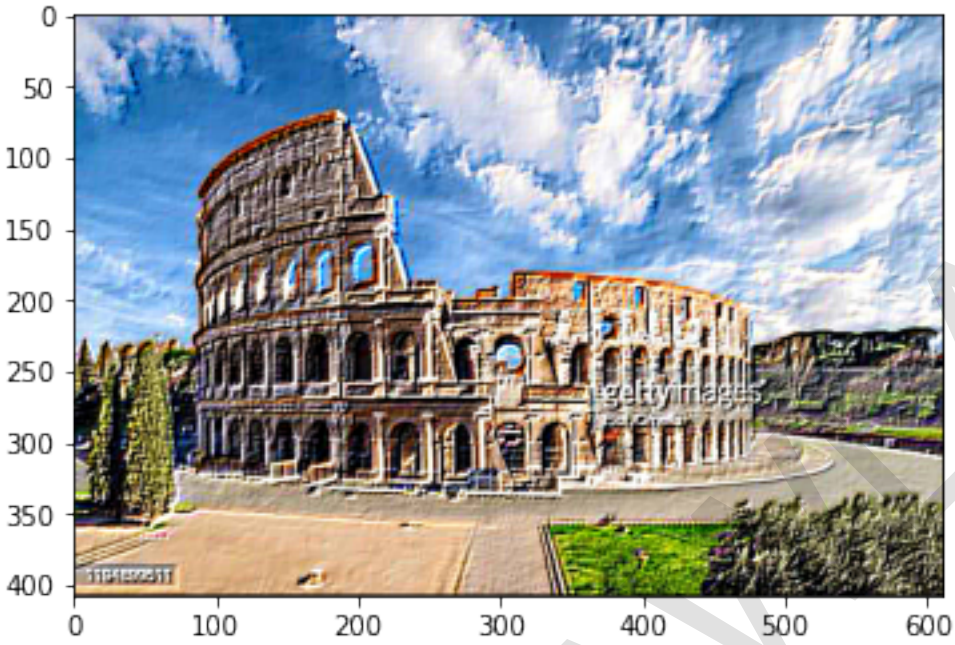
8.2 Filtre d'augmantation du contrast d'une image couleur

```
[ ]: # Application du filtre d'augmentation du contrast
AC=np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
image_amélioré = convolution(image,AC)
plt.imshow(image_amélioré)
plt.show()
```



8.3 Filtre repoussage

```
[ ]: # Application du filtre repoussage
R=np.array([[-2,-1,0],[-1,1,1],[0,1,2]])
image_repoussé = convolution(image,R)
plt.imshow(image_repoussé)
plt.show()
```

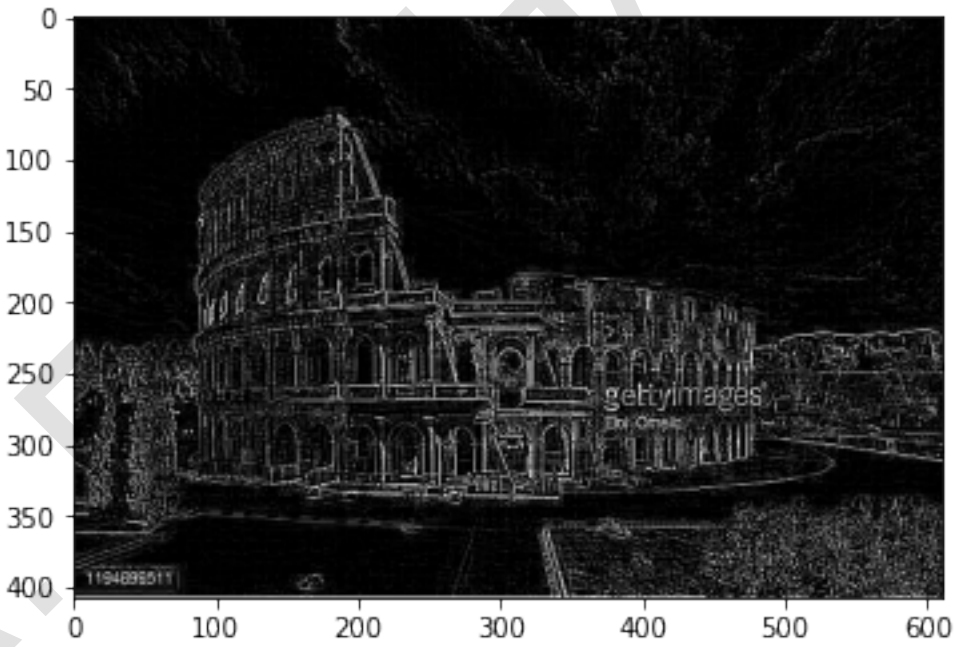


8.4 Detection des contours

[ ]:
```
# Premièrement on rédige une fonction qui nous aide à appliquer un noyau sur un pixel (i,j) d'une image grise
def appliquer_noyau_pixel_grise(N,im,i,j):
  somme_pixels = 0
  for p in range(3):
    for q in range (3):
      somme_pixels  = somme_pixels + N[p,q]*im[p+i-1,q+j-1]
  if somme_pixels<0:
    somme_pixels=0
  if somme_pixels>255:
    somme_pixels=255
  return somme_pixels

# Deuxièmement on crée la fonction convolution
def convolution_grise(im,N):
  l,c = im.shape
  im_couverte=np.zeros((l+2,c+2),dtype=int)
  im_couverte[1:-1,1:-1]=im
  nim = np.zeros((l,c),dtype=int)
  for i in range(l):
    for j in range(c):
```

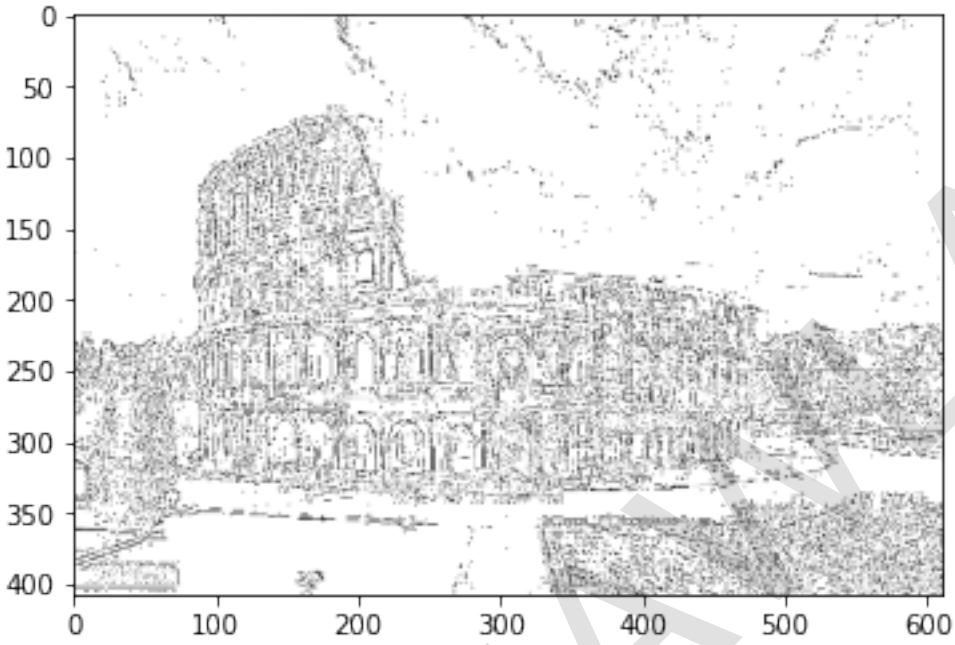
```
      nim[i,j]=appliquer_noyau_pixel_grise(N,im_couverte,i+1,j+1)
  return nim

# Finalement on crée la fonction du seuillage
def seuillage(im,seuil):
  l,c = im.shape
  nim=np.zeros((l,c),dtype=int)
  for i in range(l):
    for j in range(c):
      if im[i,j]<seuil:
        nim[i,j]=255
      else:
        nim[i,j]=im[i,j]
  return nim
```

[ ]:
```
# Application du détection des contours sans seuillage
DC=np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
image_grise=conversion_gris(image)
contours = convolution_grise(image_grise,DC)
plt.imshow(contours,cmap='gray')
plt.show()
```



[ ]:
```
# Application du détection des contours avec seuillage : seuil = 50
countours_seuillage=seuillage(contours,50)
plt.imshow(countours_seuillage,cmap='gray')
```

```
plt.show()
```



[ ]:
```
# Application du détection des contours avec seuillage : seuil = 10
countours_seuillage=seuillage(contours,10)
plt.imshow(countours_seuillage,cmap='gray')
plt.show()
```