Détermination de l'âge osseux à partir de l'analyse d'une radiographie de la main

Lepecuchelle Chad - n°10387

Introduction

2 ans

9 ans

Objectifs

But : Développer des outils pour analyser une radiographie de la main et être capable de déterminer l'âge osseux d'une personne

2 ans

9 ans

Plan

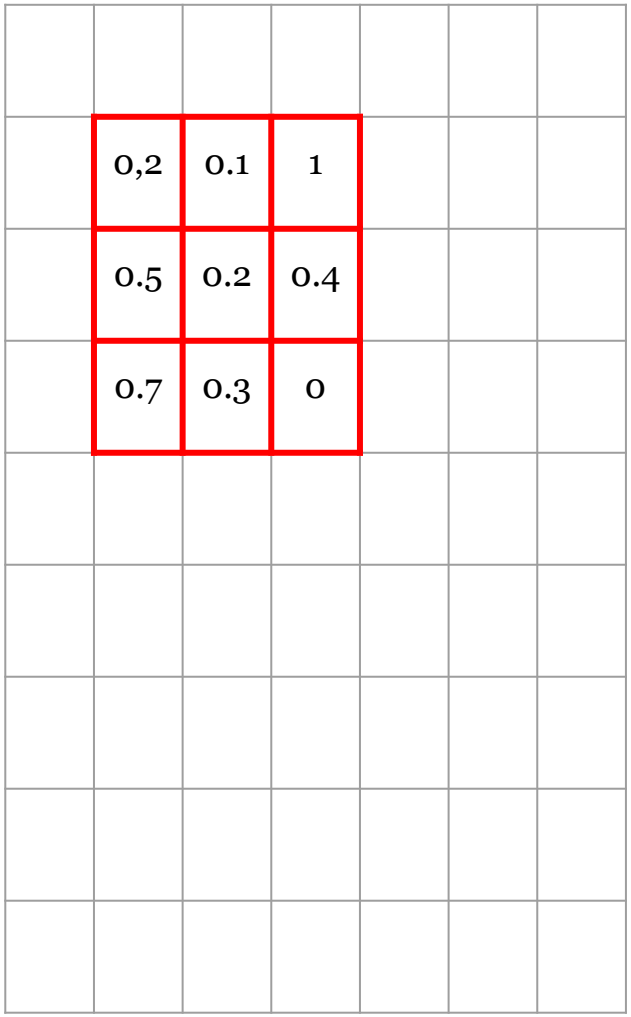
Image

Nuances de gris $\Rightarrow$ Pixel $\in [0,1]$

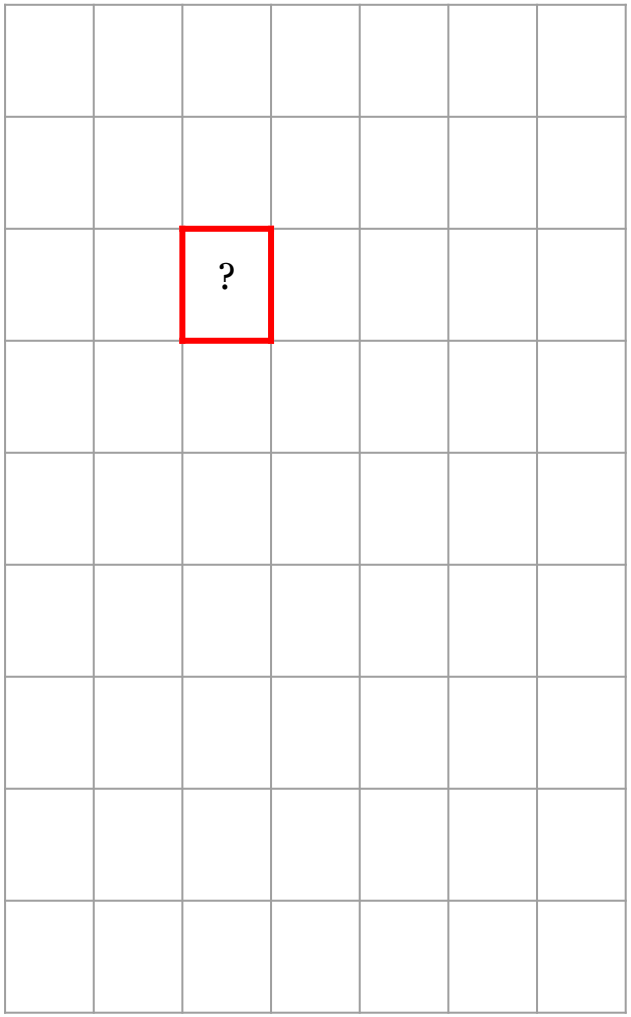
 $= 0$  $\in]0,1[$  $= 1$

```
array([[0, 0.13, ... , 0.2],
       [0.3, 0.21, ... , 1],
       ..................... ,
       ..................... ,
       [0.23, 0.5, ... , 0.2],
       [0.31, 0.2, ... , 1] ])
```

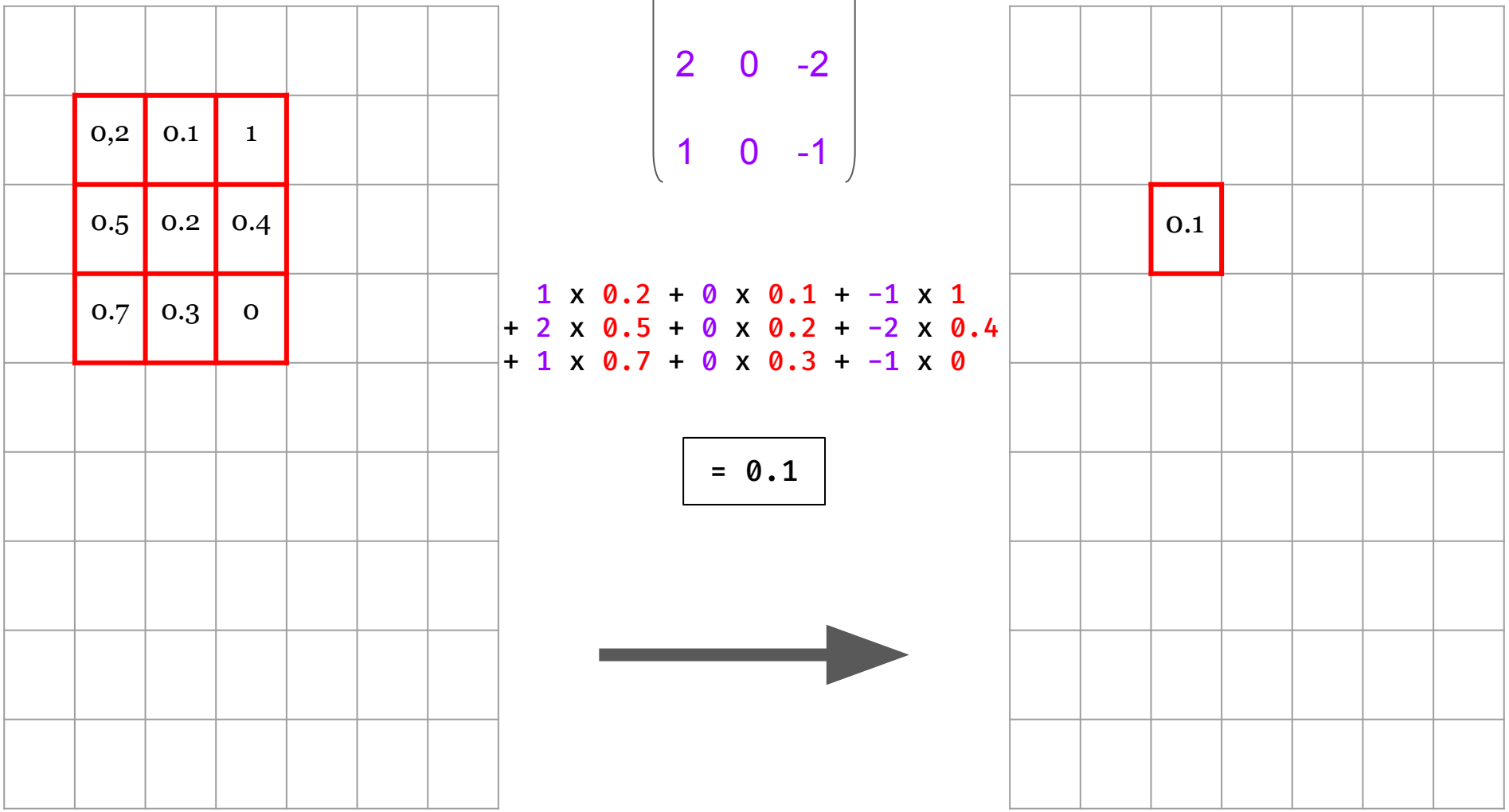
Convolution



$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$



Convolution



Convolution : Gradient

Gradient = Plus forte variation (contour)

Gradient selon x :

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Gradient selon y :

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

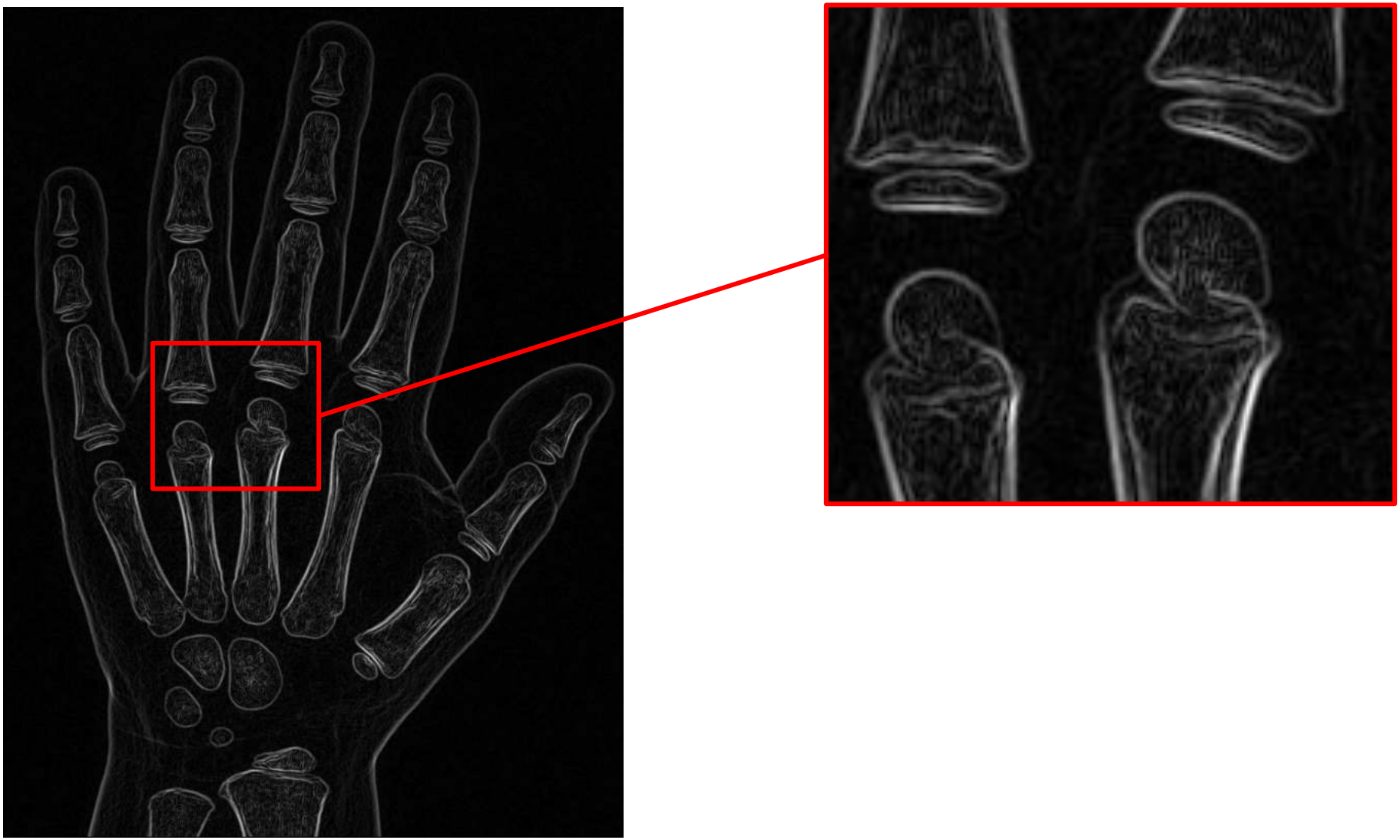
Norme : $\sqrt{GradX^2 + GradY^2}$

Direction : $\arctan \frac{GradY}{GradX}$

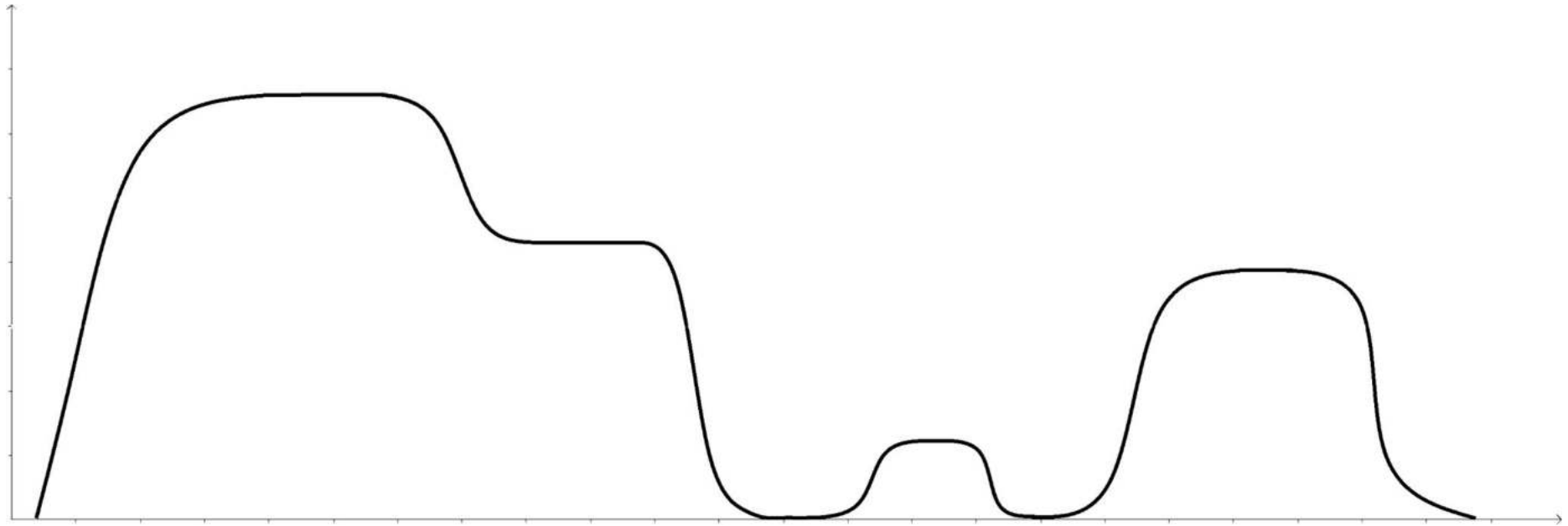
Convolution : Gradient

Seuil

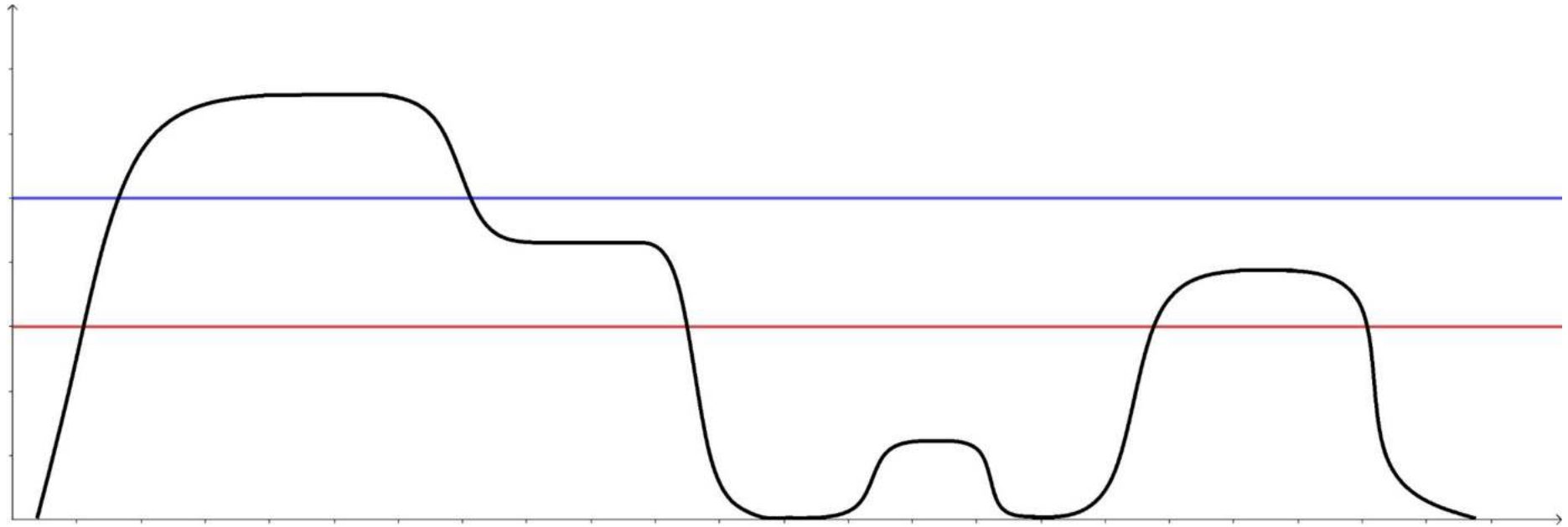
Seuil



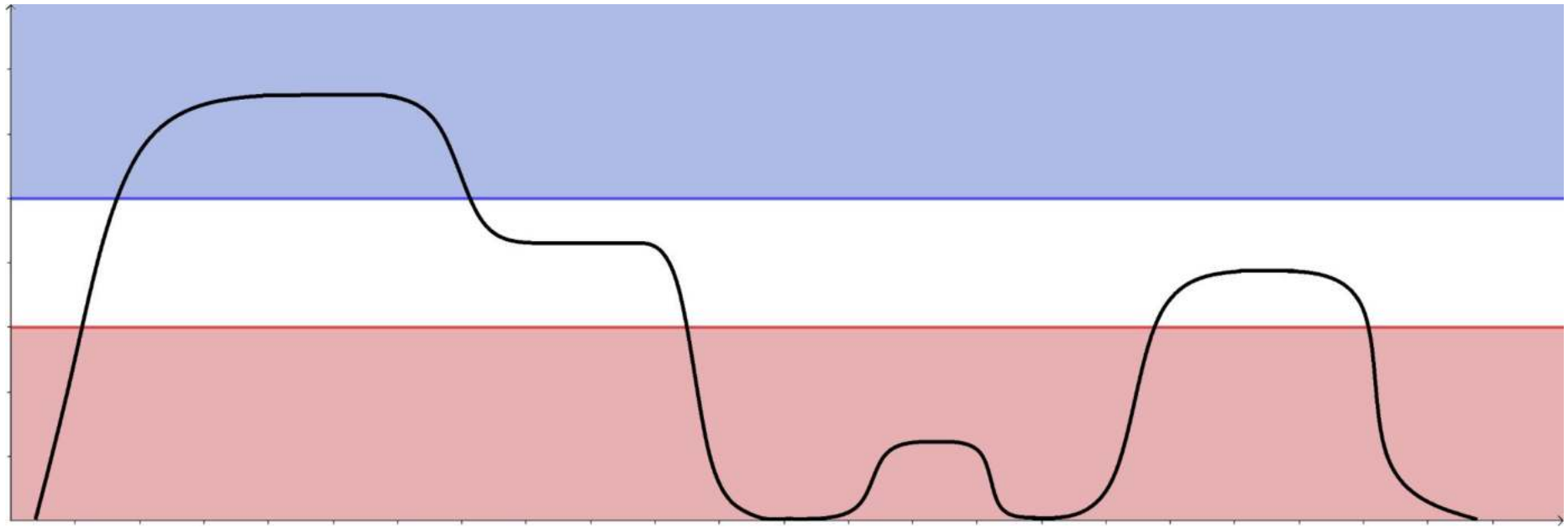
Seuil



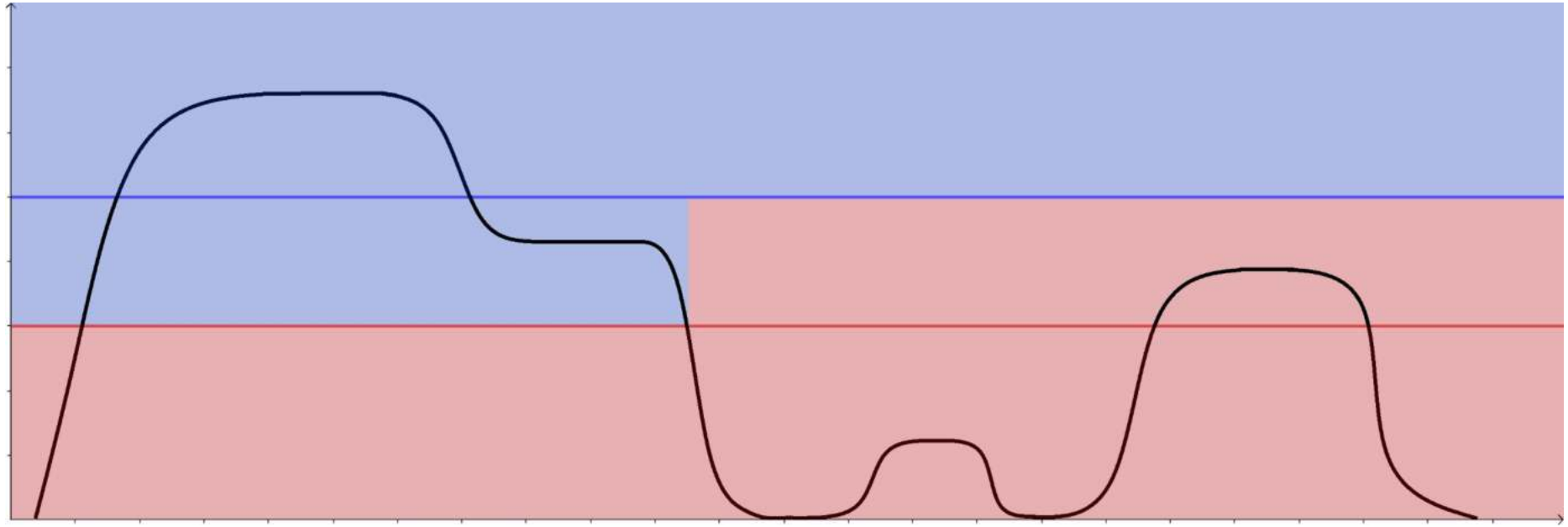
Seuil



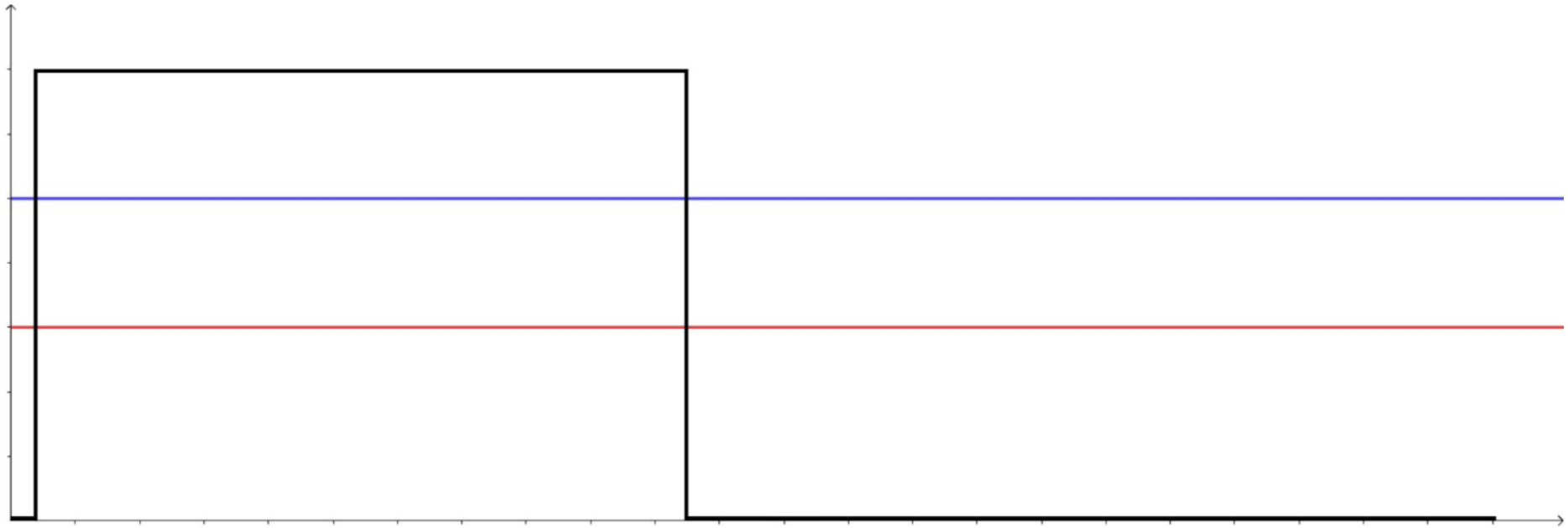
Seuil



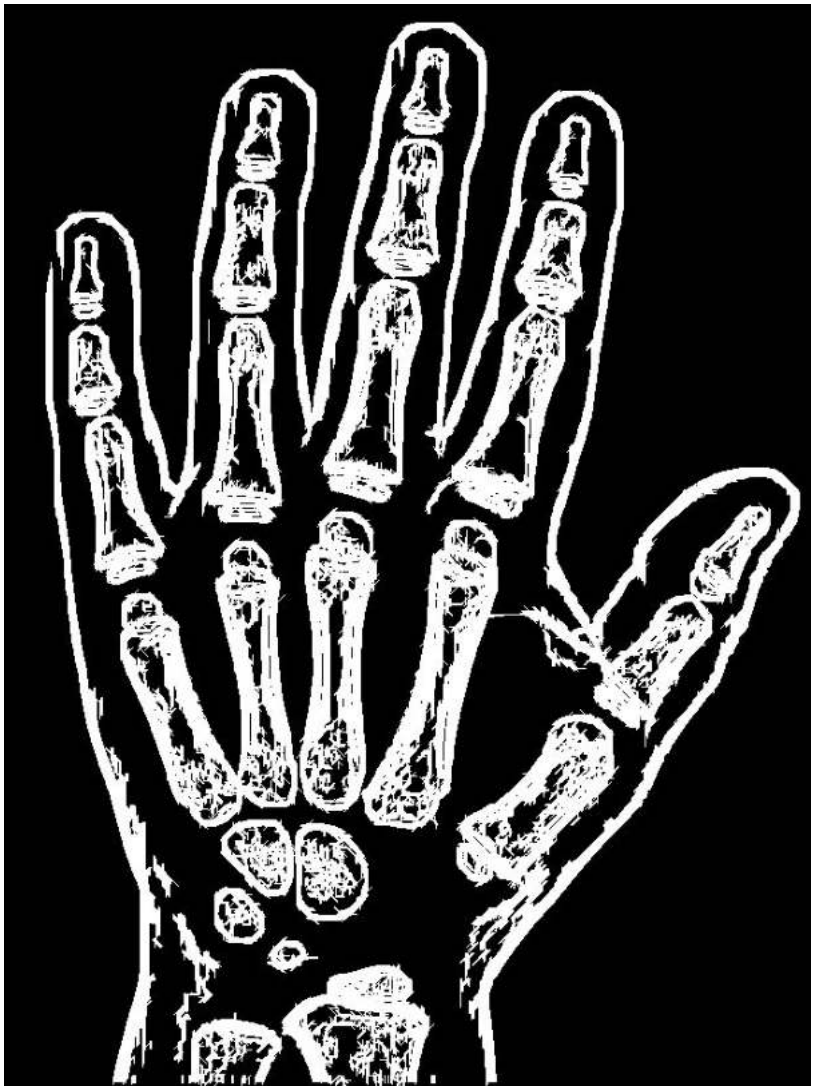
Seuil



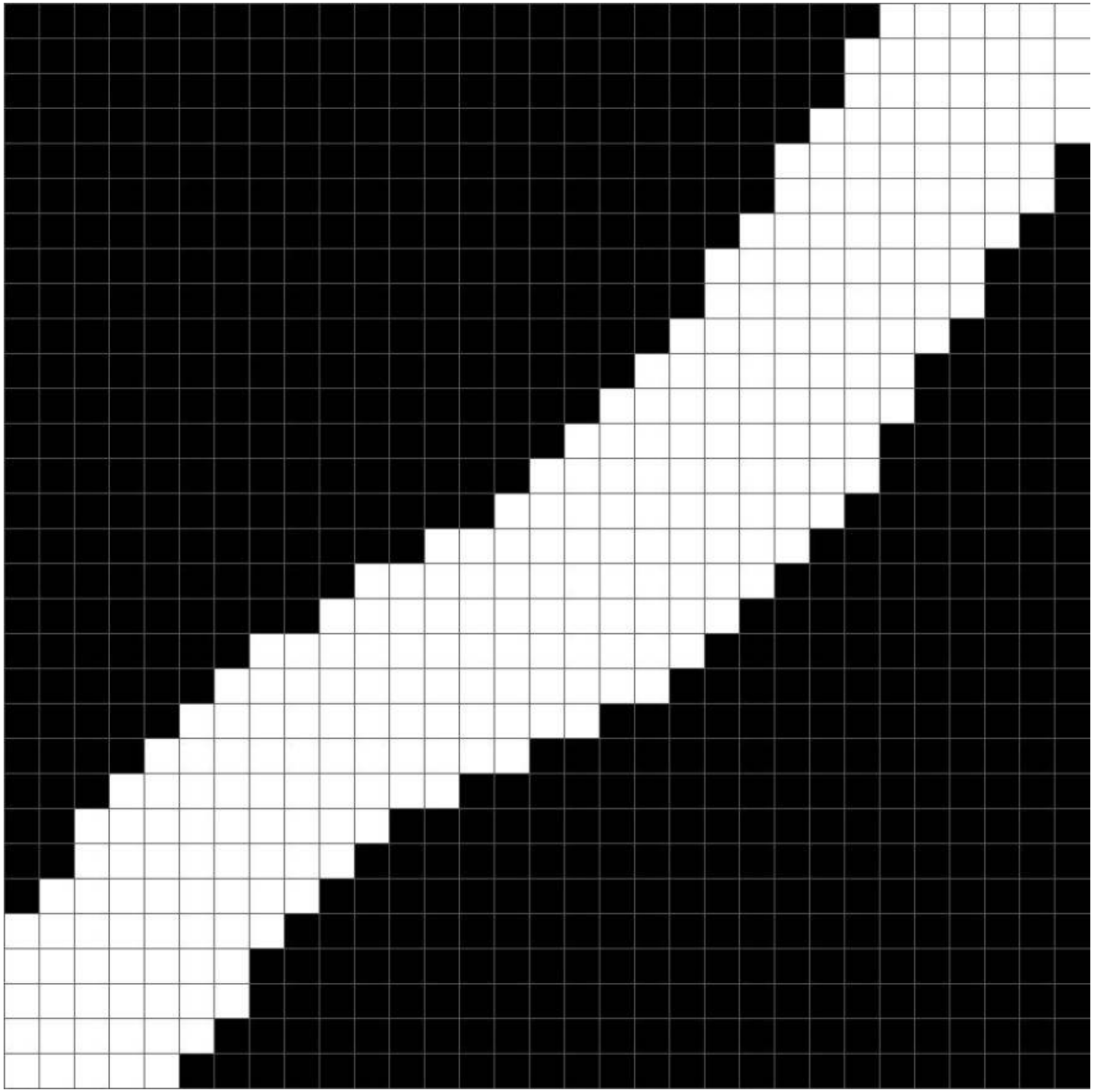
Seuil



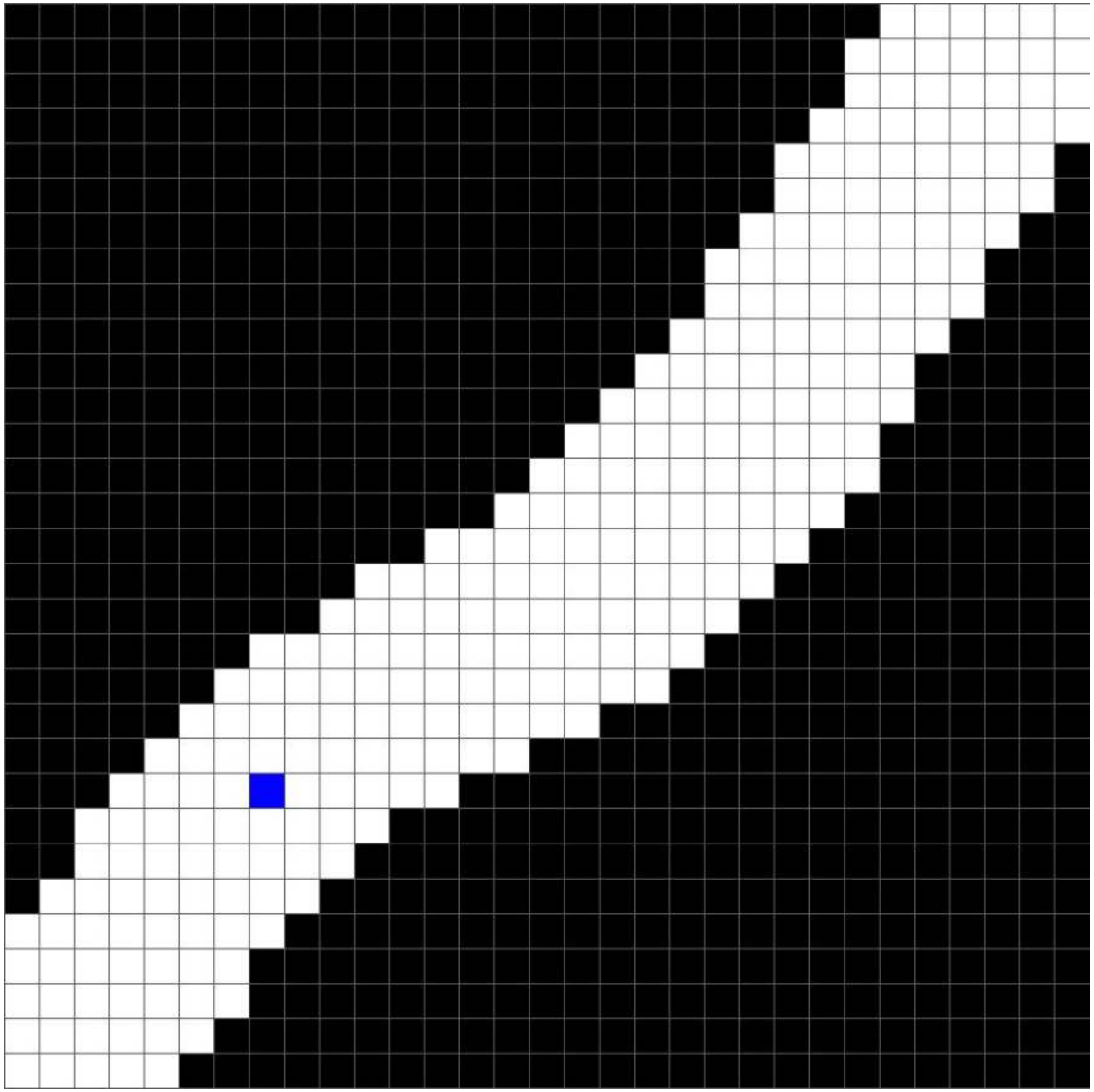
Seuil



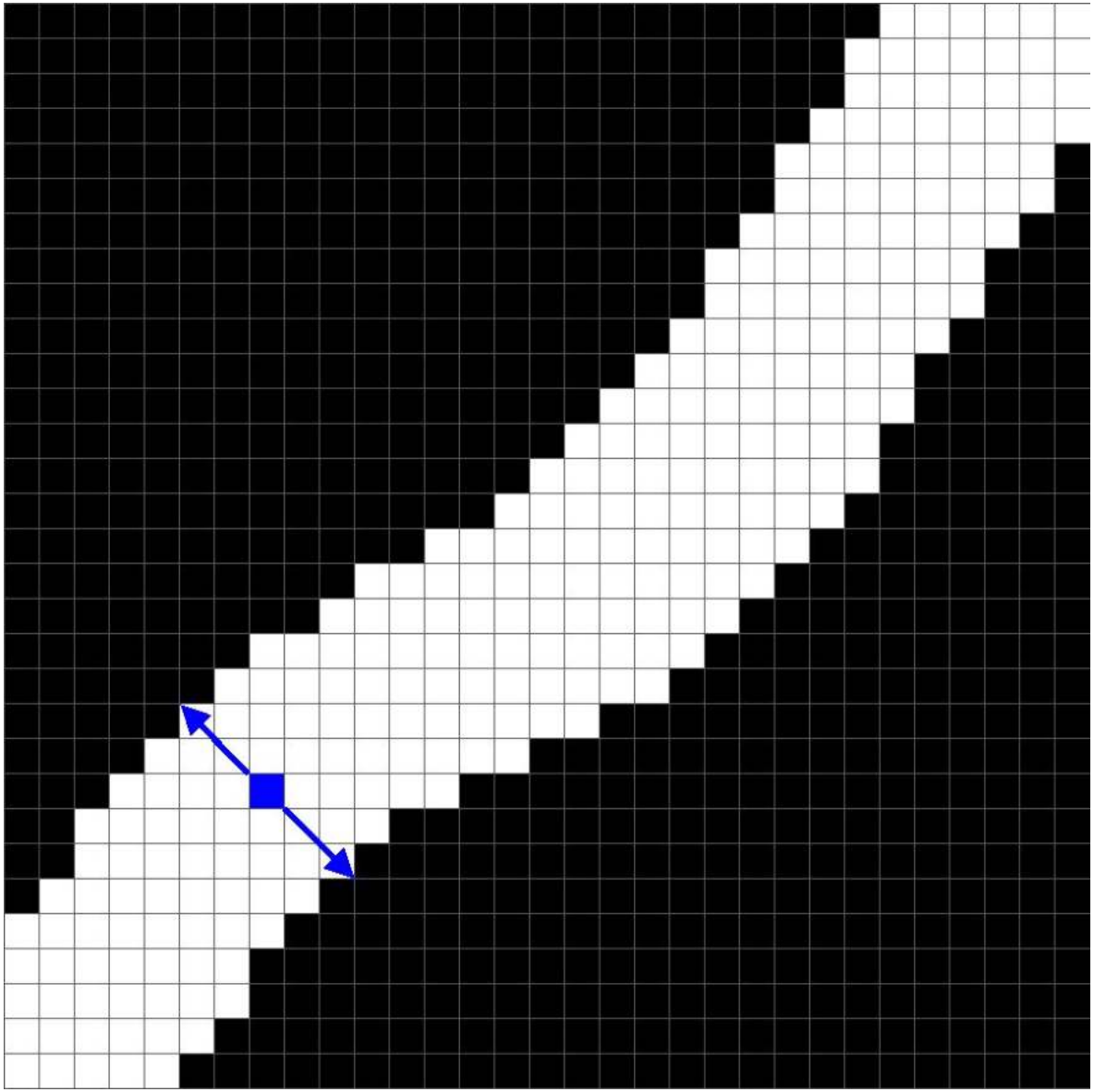
Contours minces



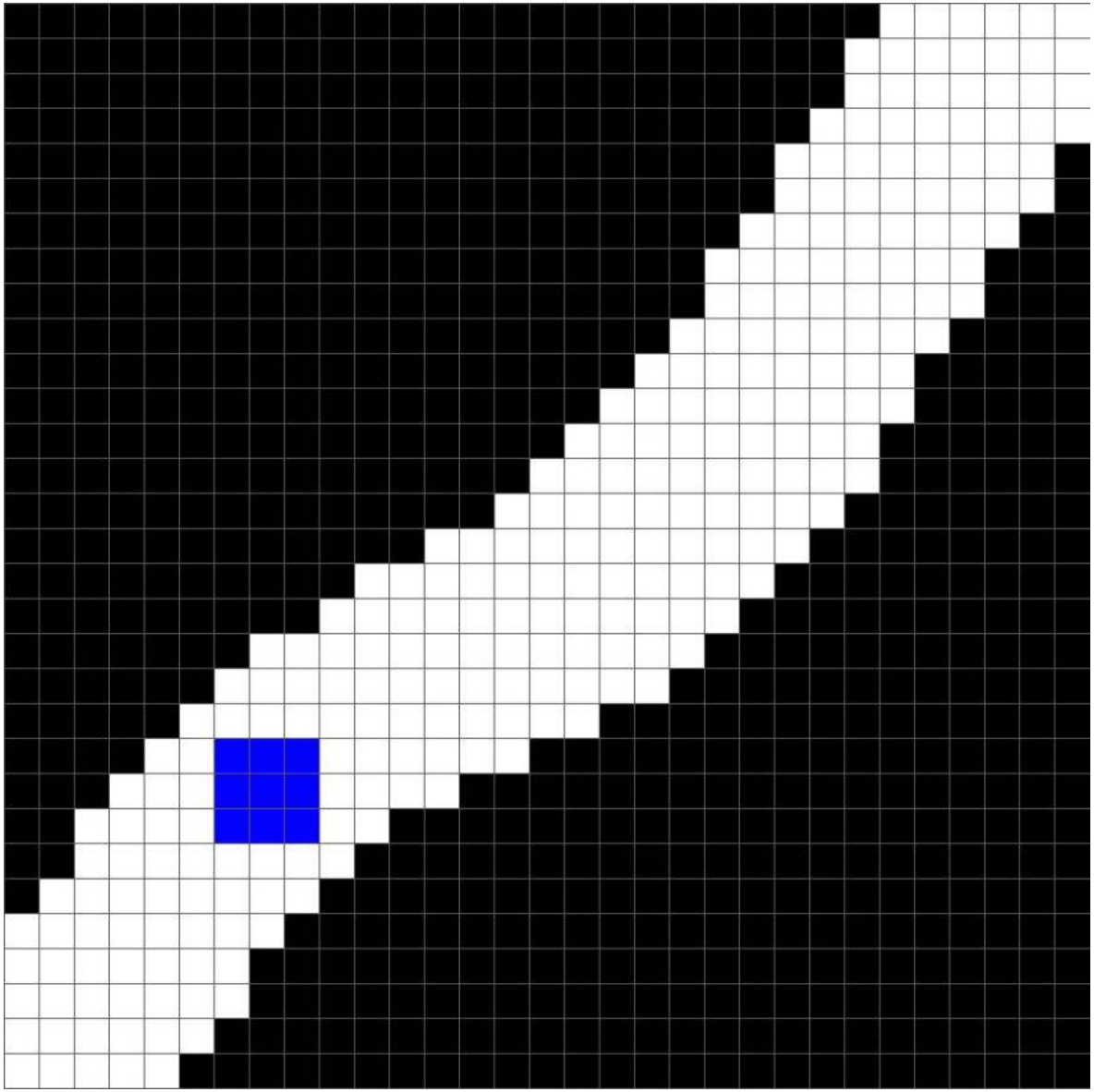
Contours minces



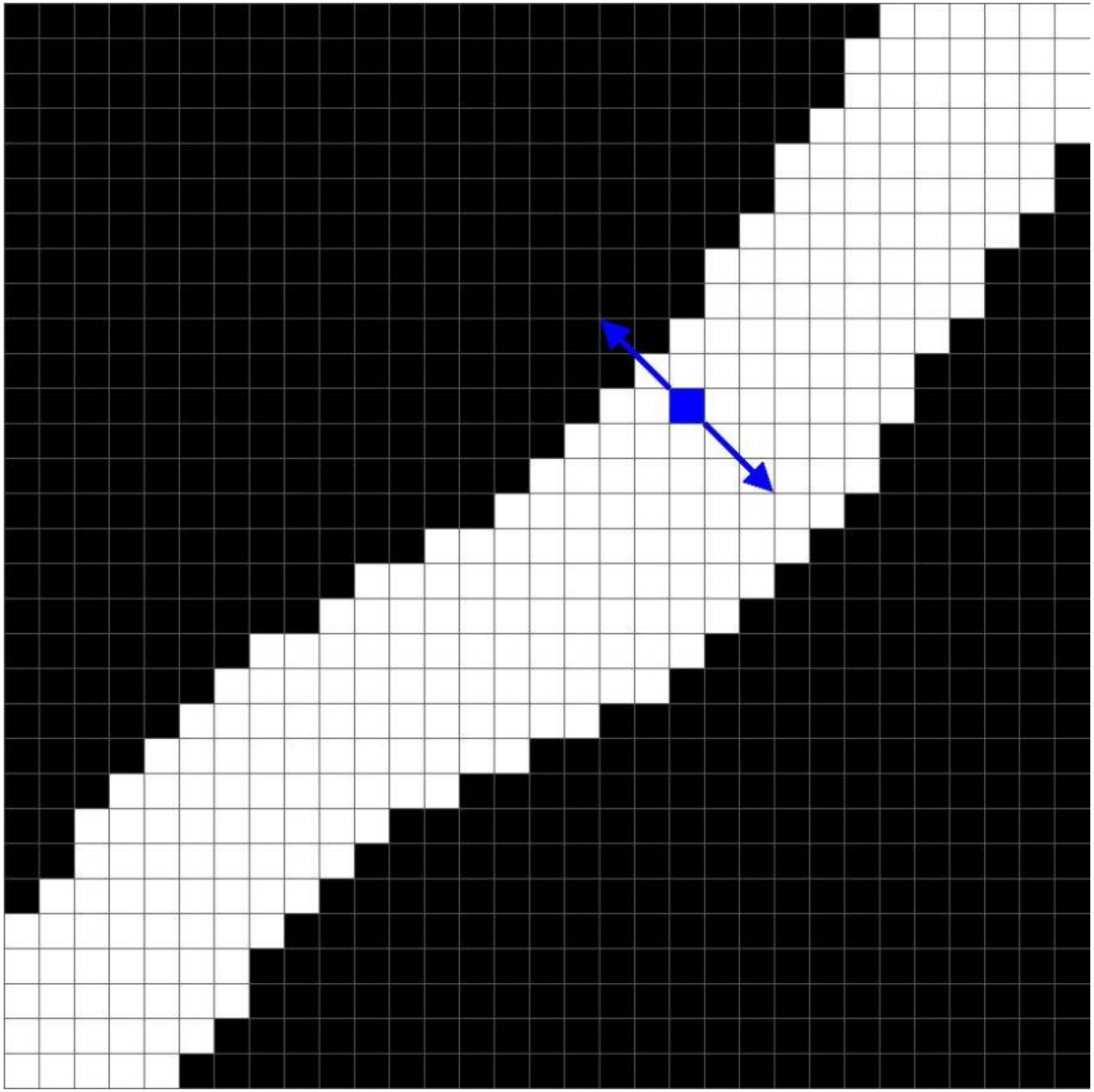
Contours minces



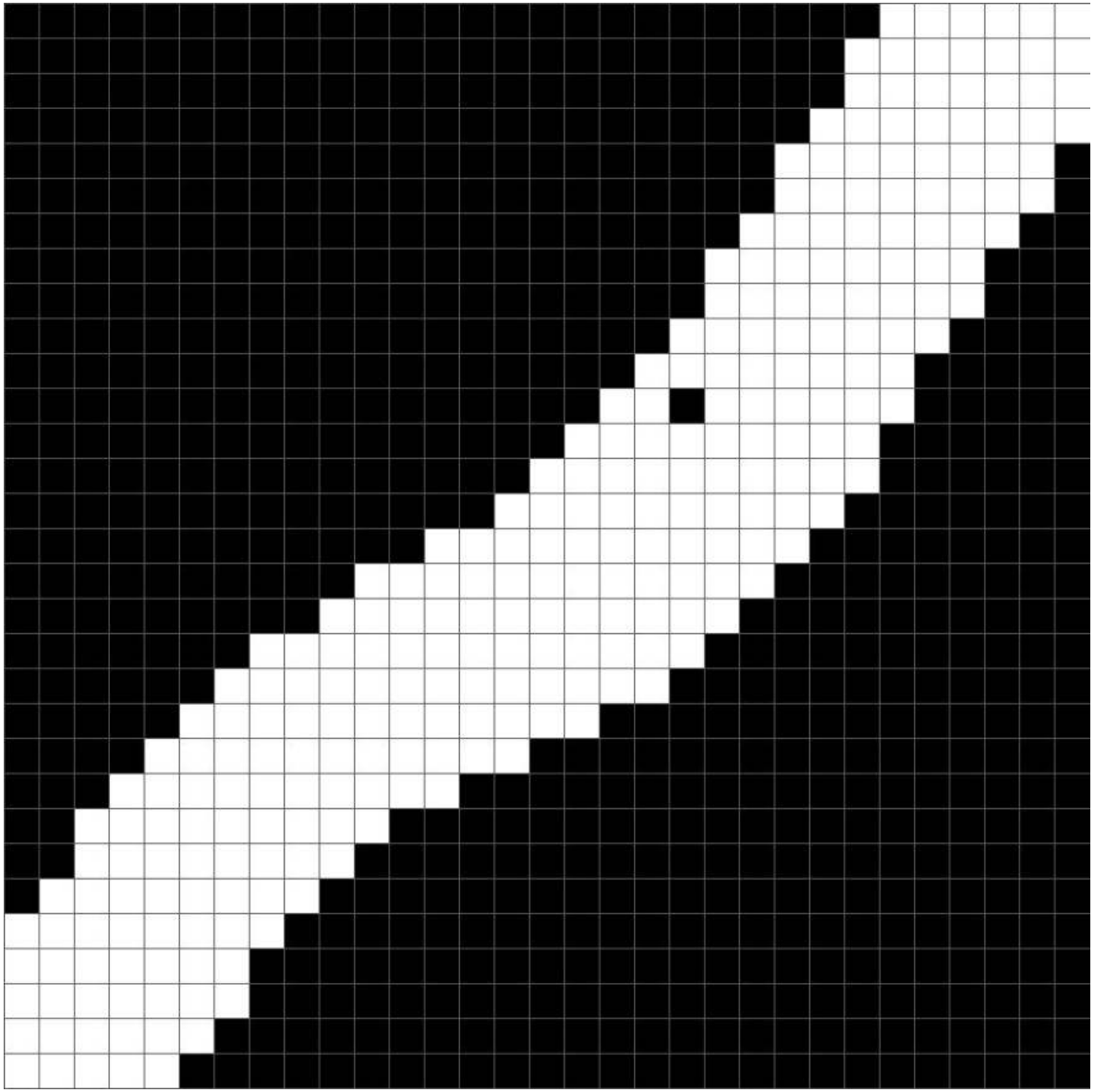
Contours minces



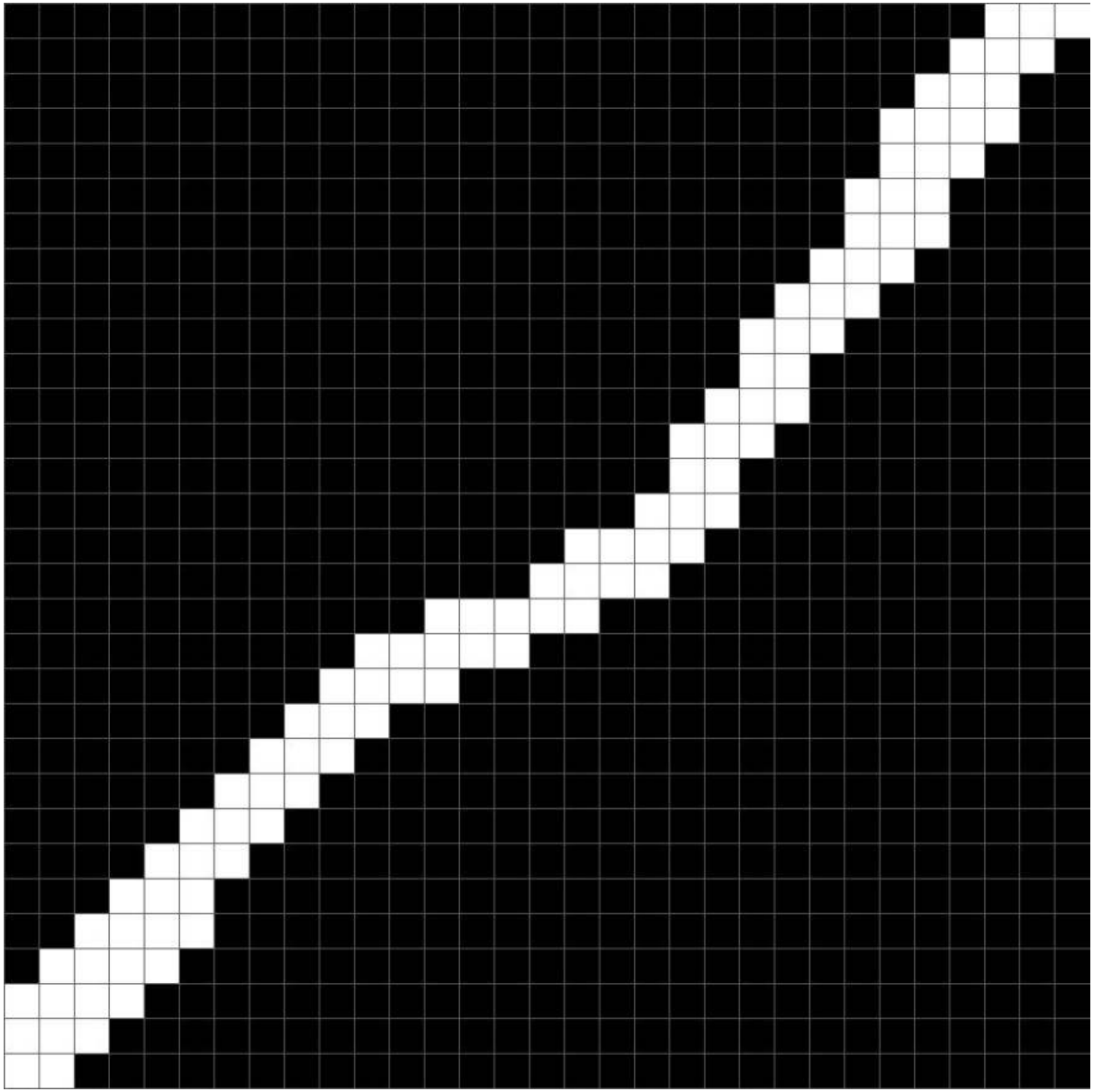
Contours minces



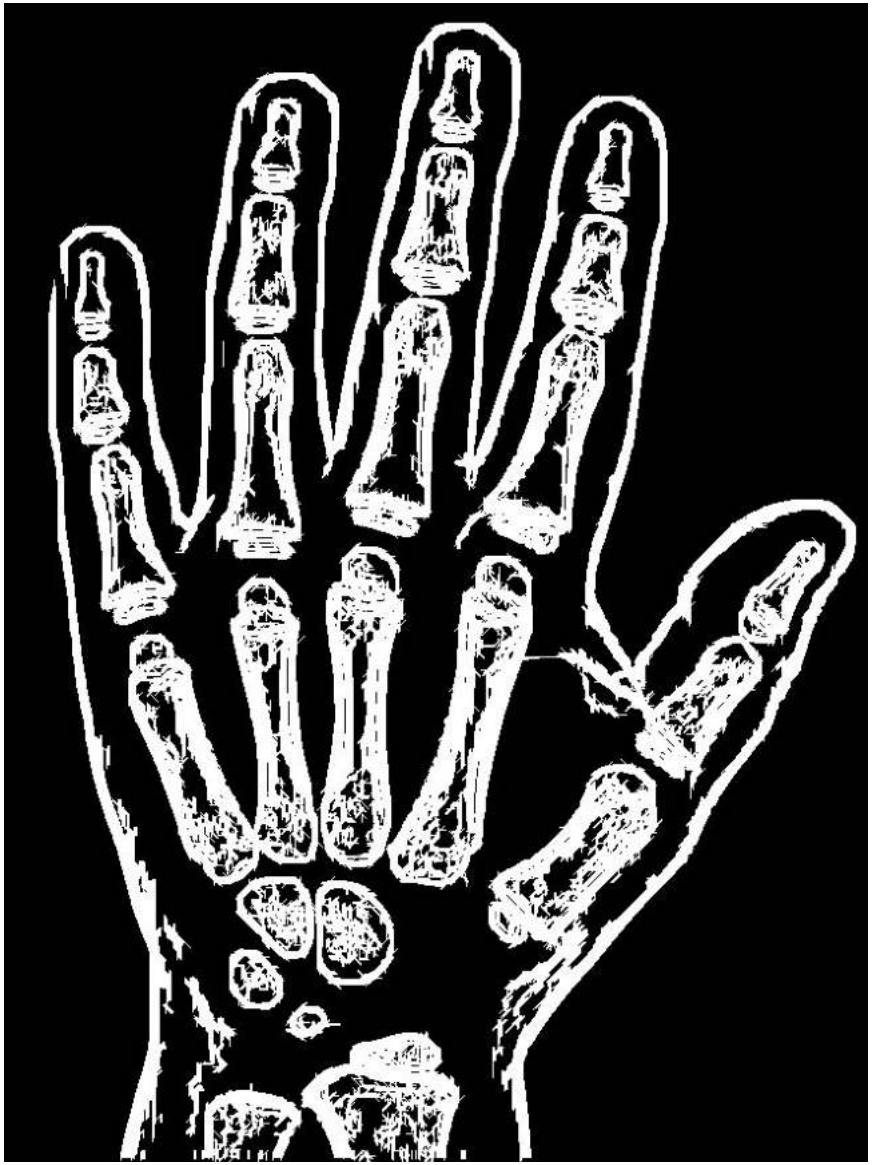
Contours minces

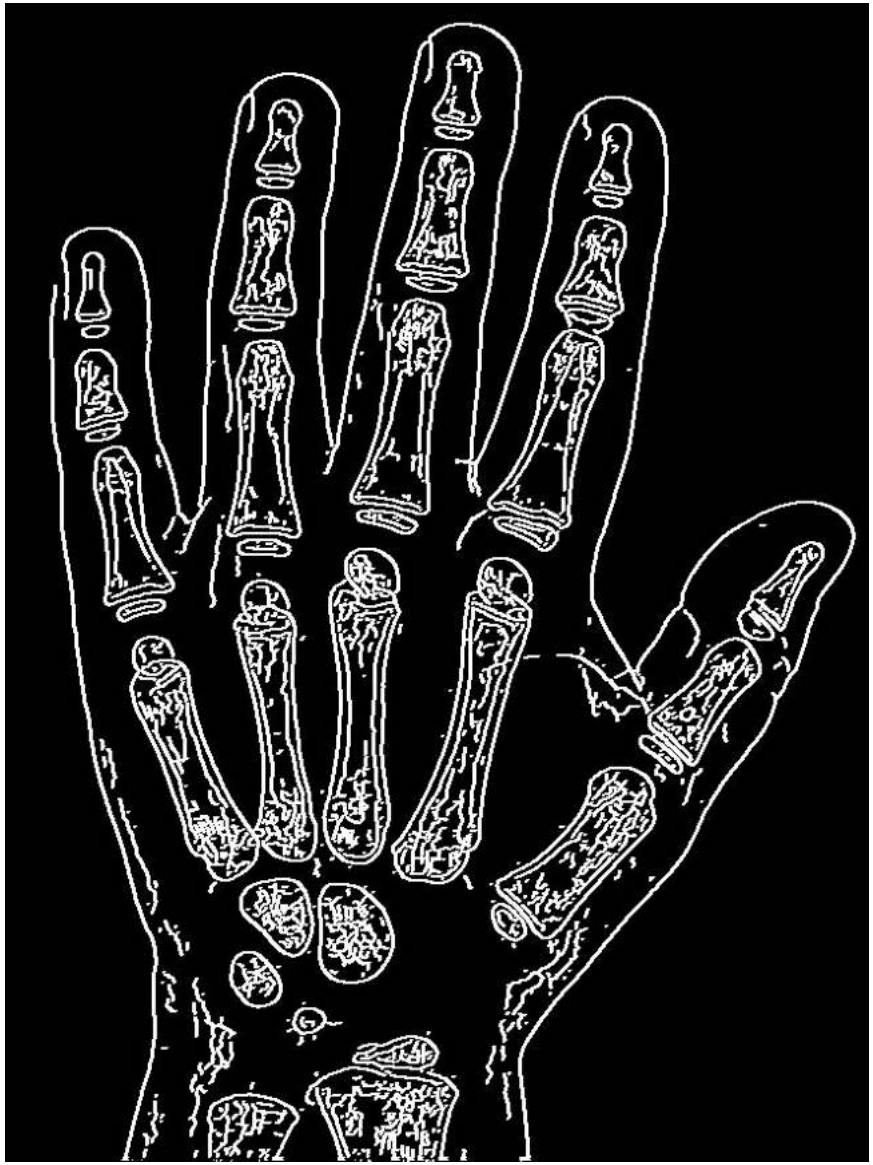


Contours minces

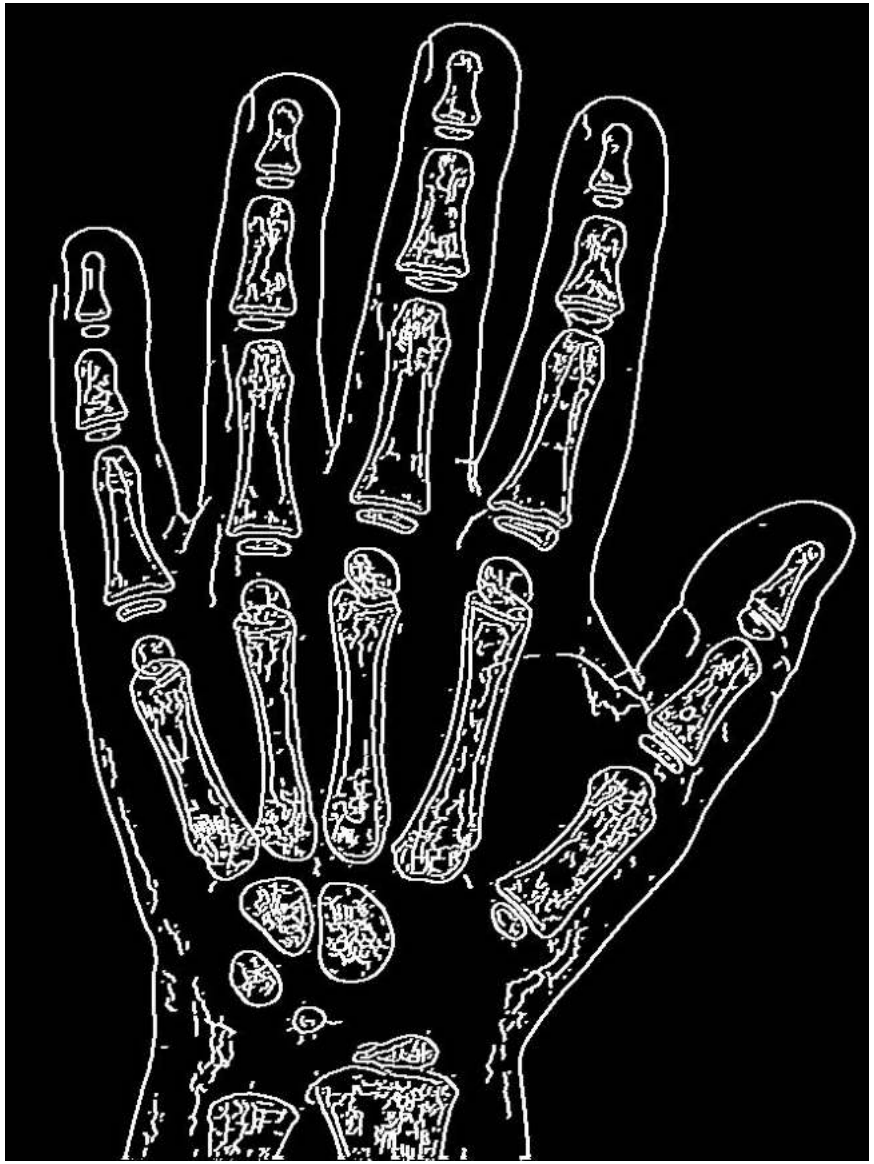


Contours minces

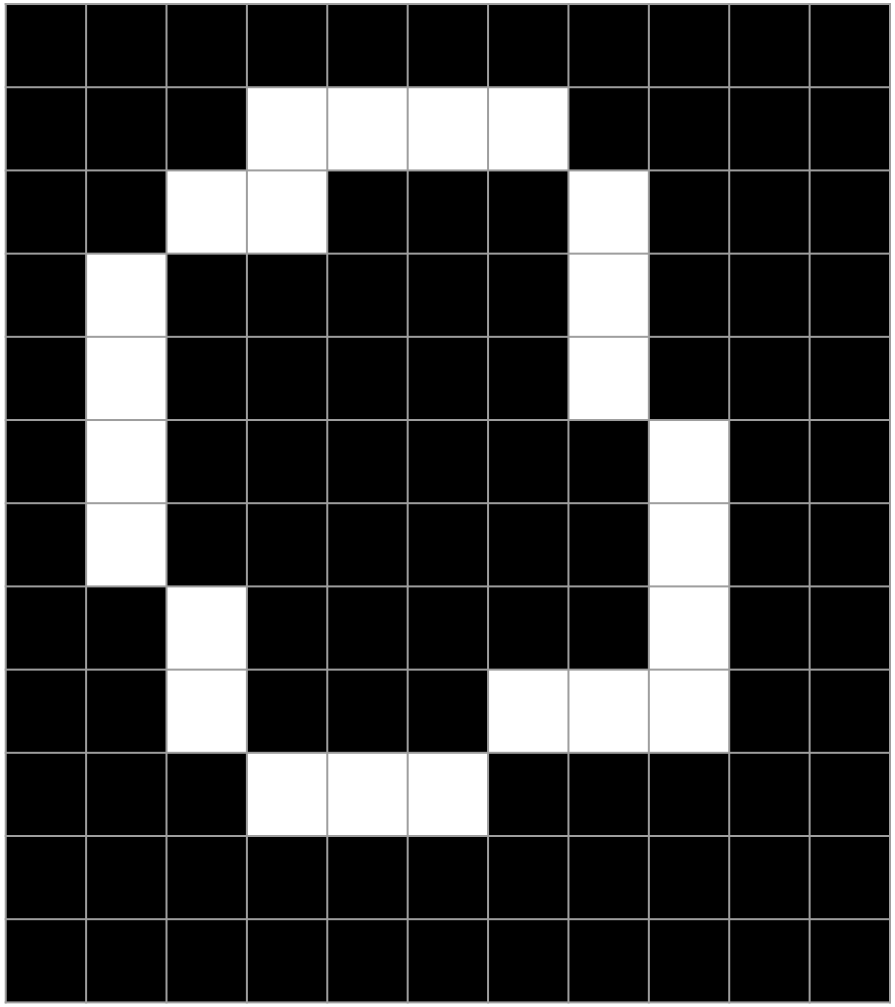




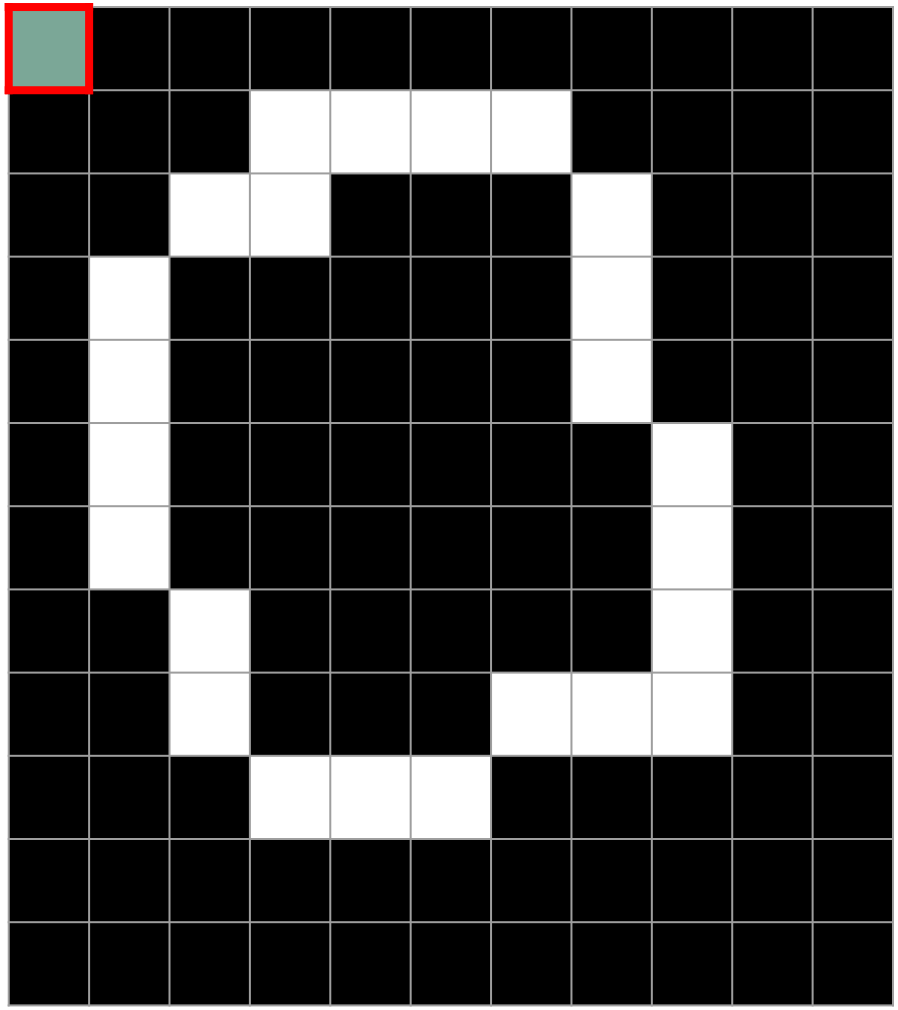
Contours



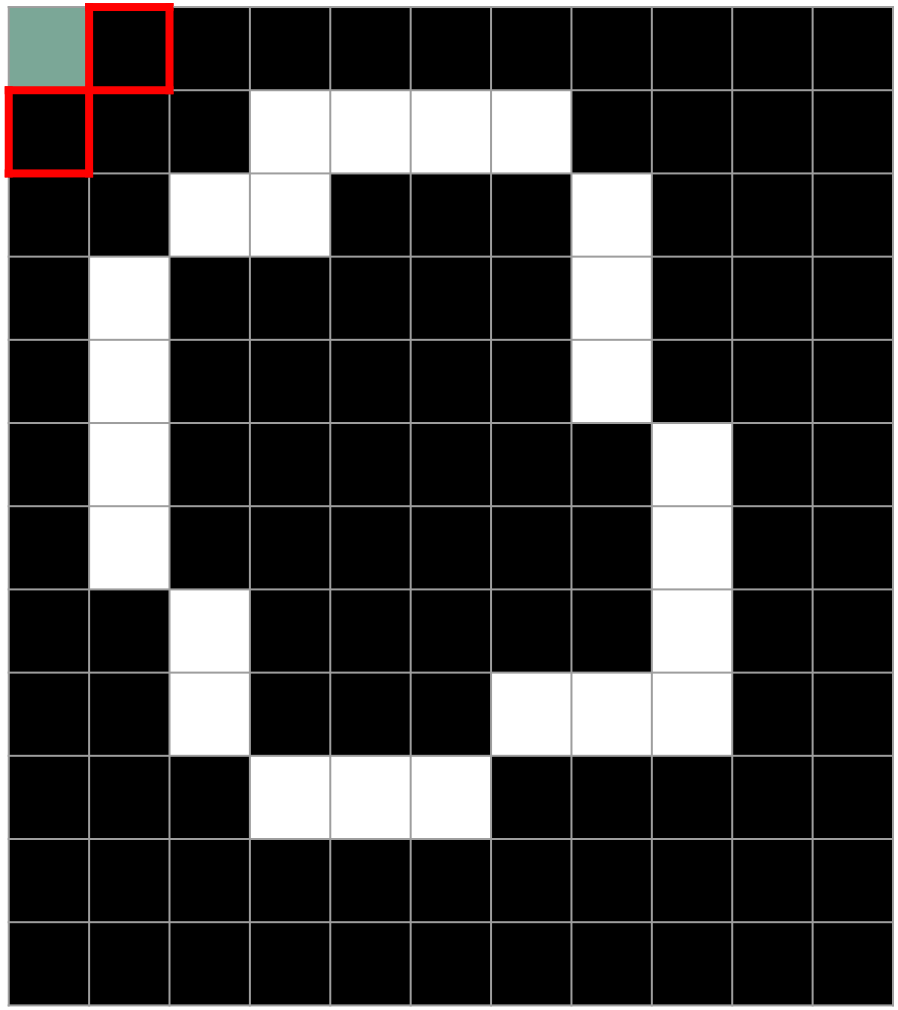
Composantes connexes



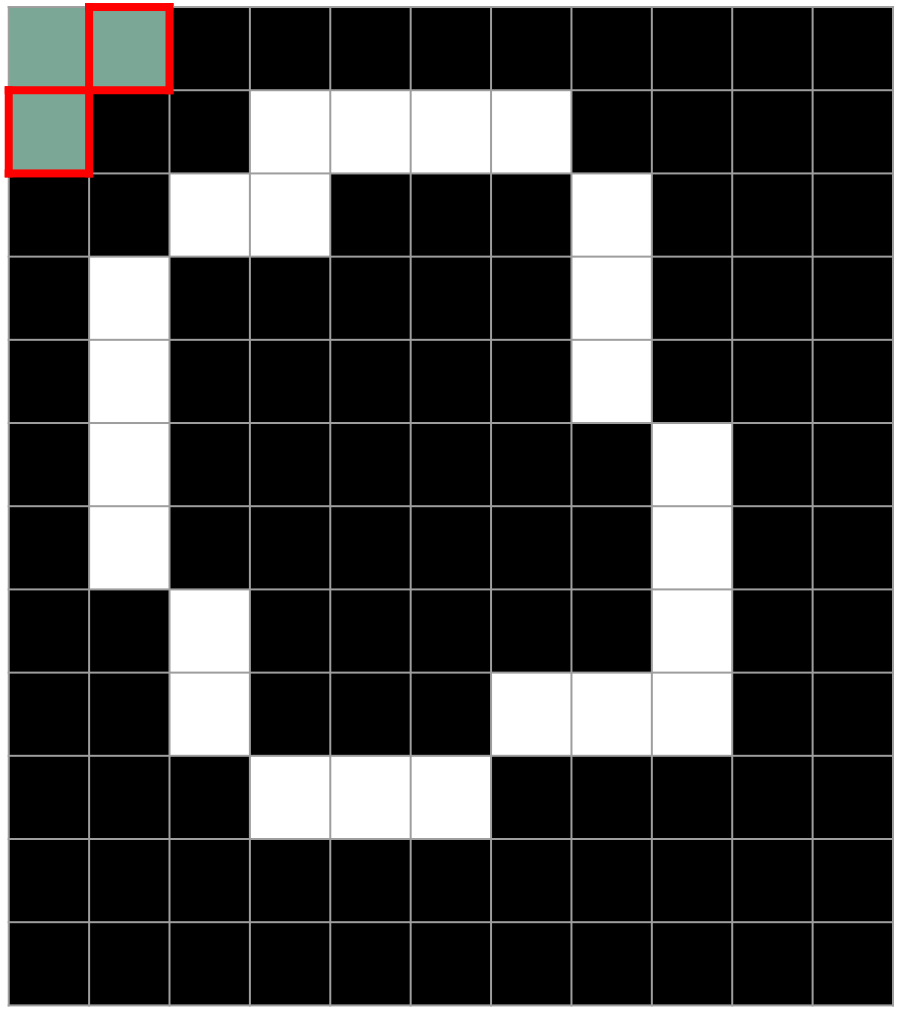
Composantes connexes



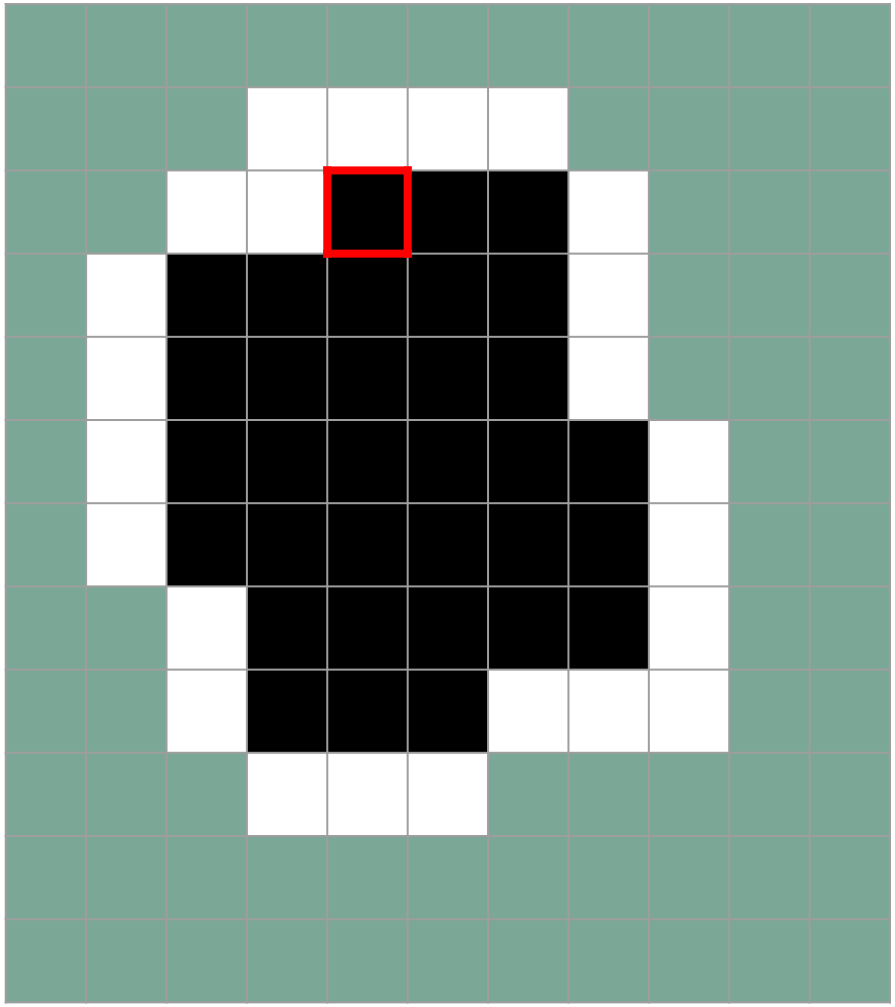
Composantes connexes



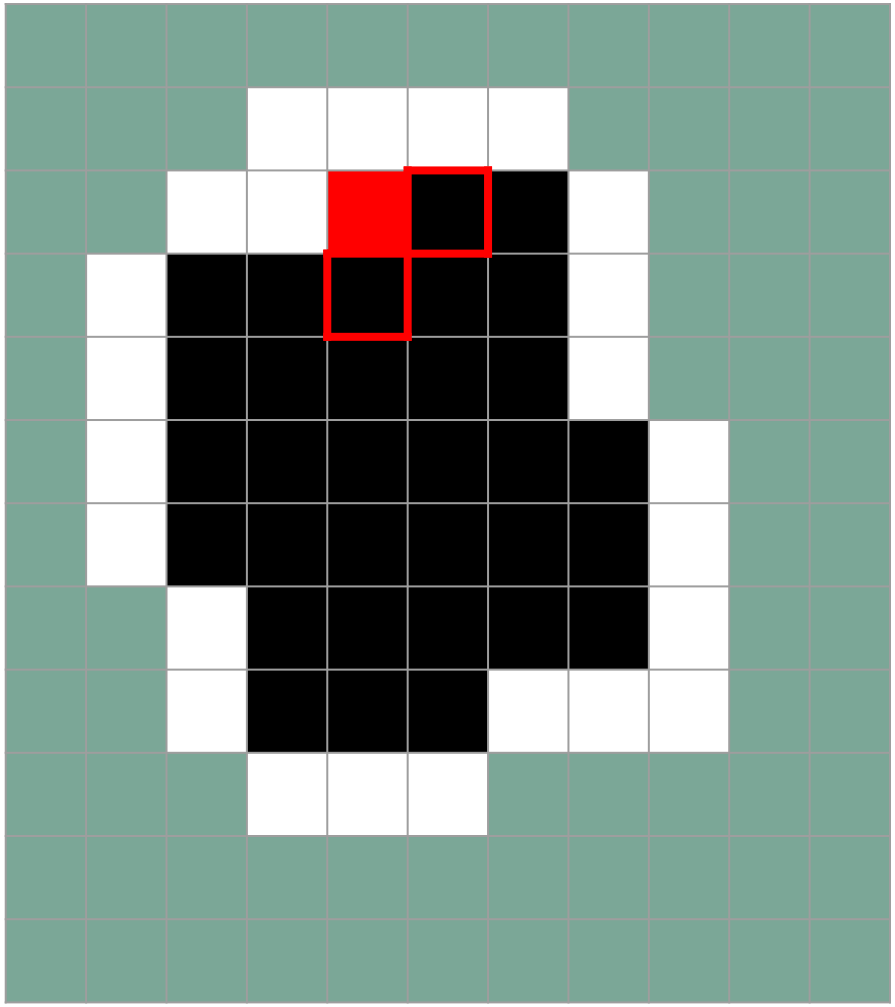
Composantes connexes



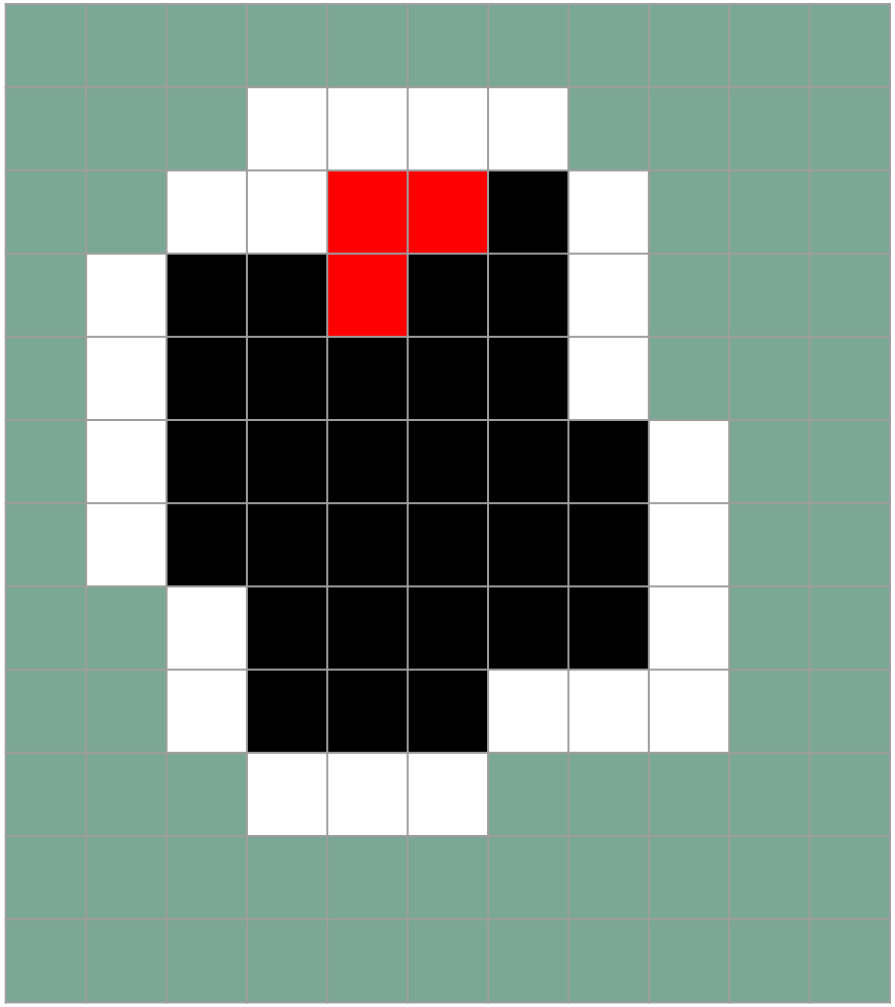
Composantes connexes



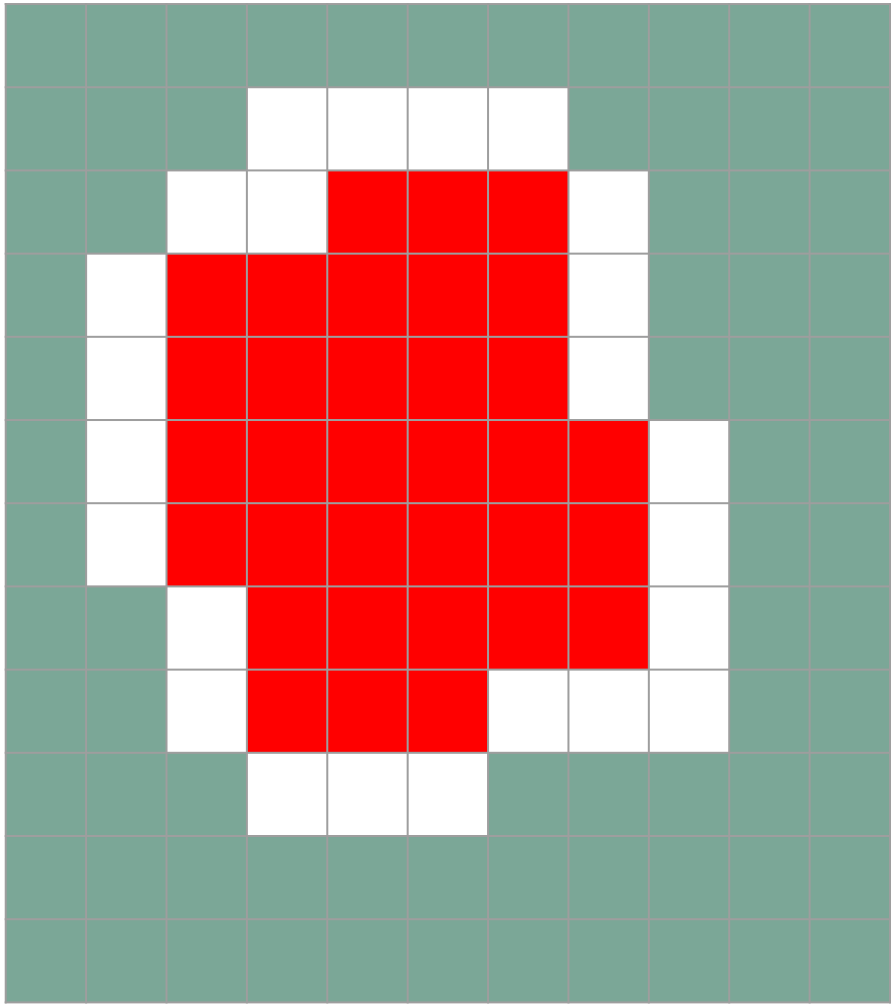
Composantes connexes



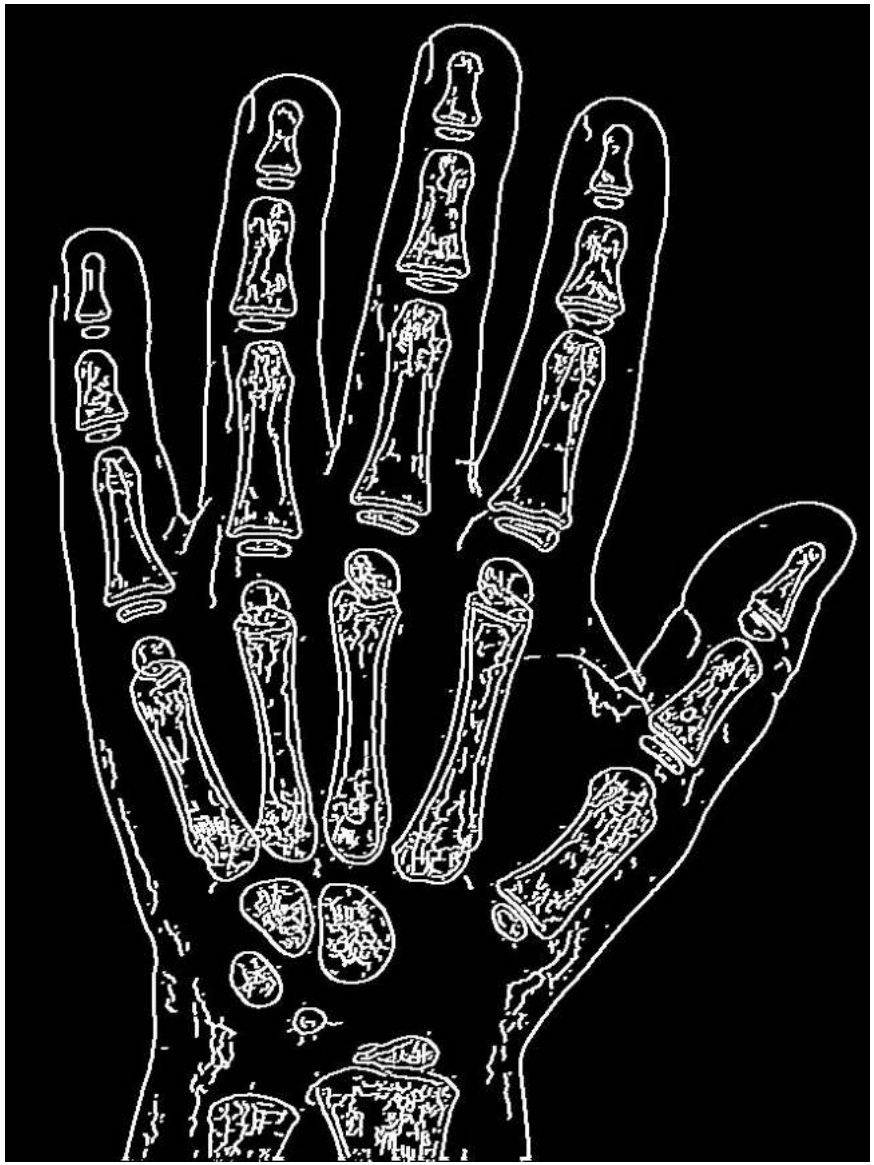
Composantes connexes



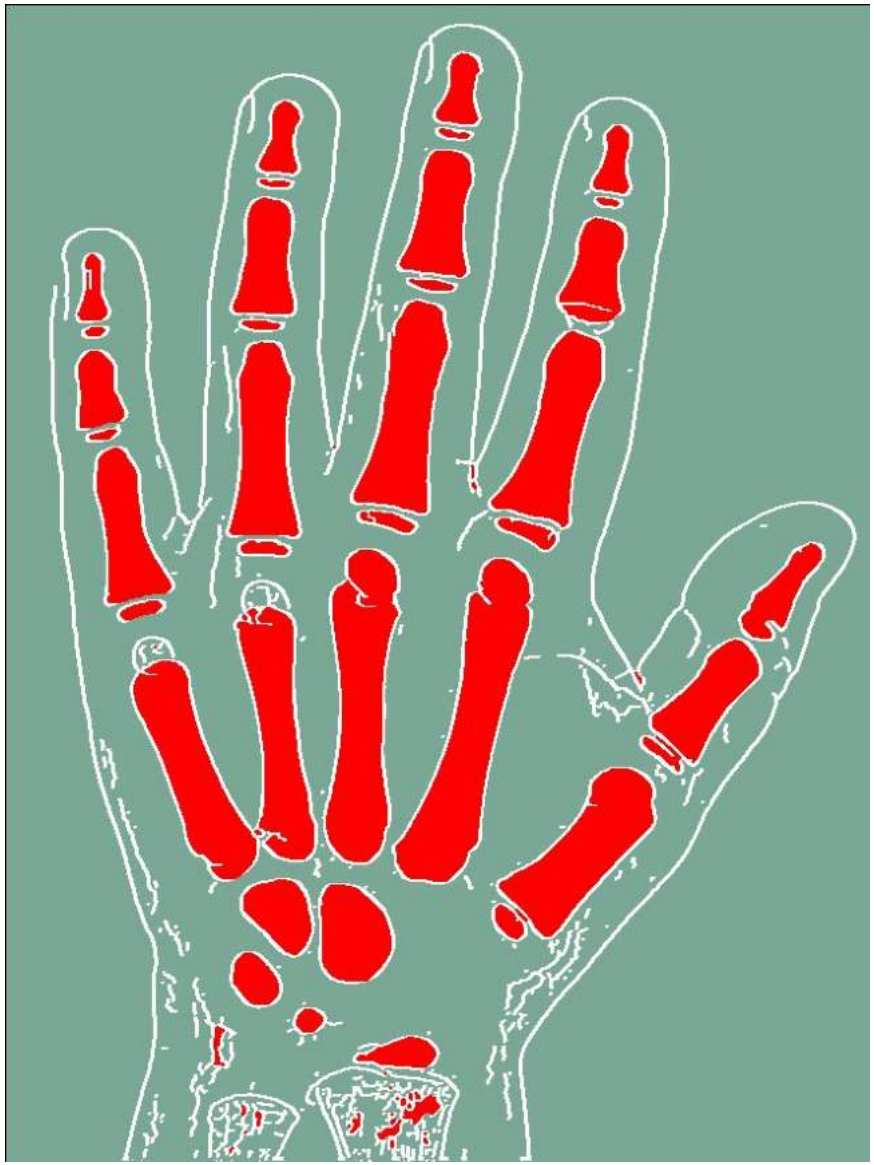
Composantes connexes



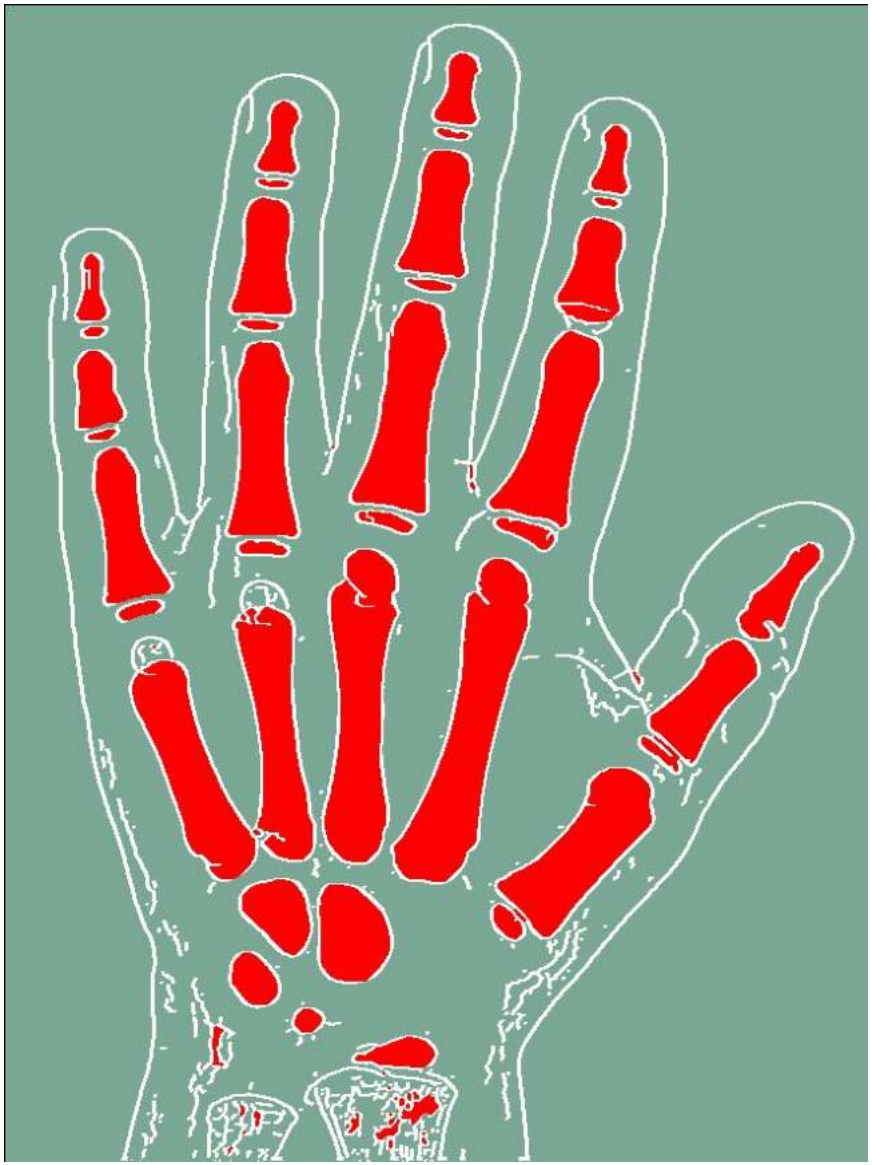
Composantes connexes



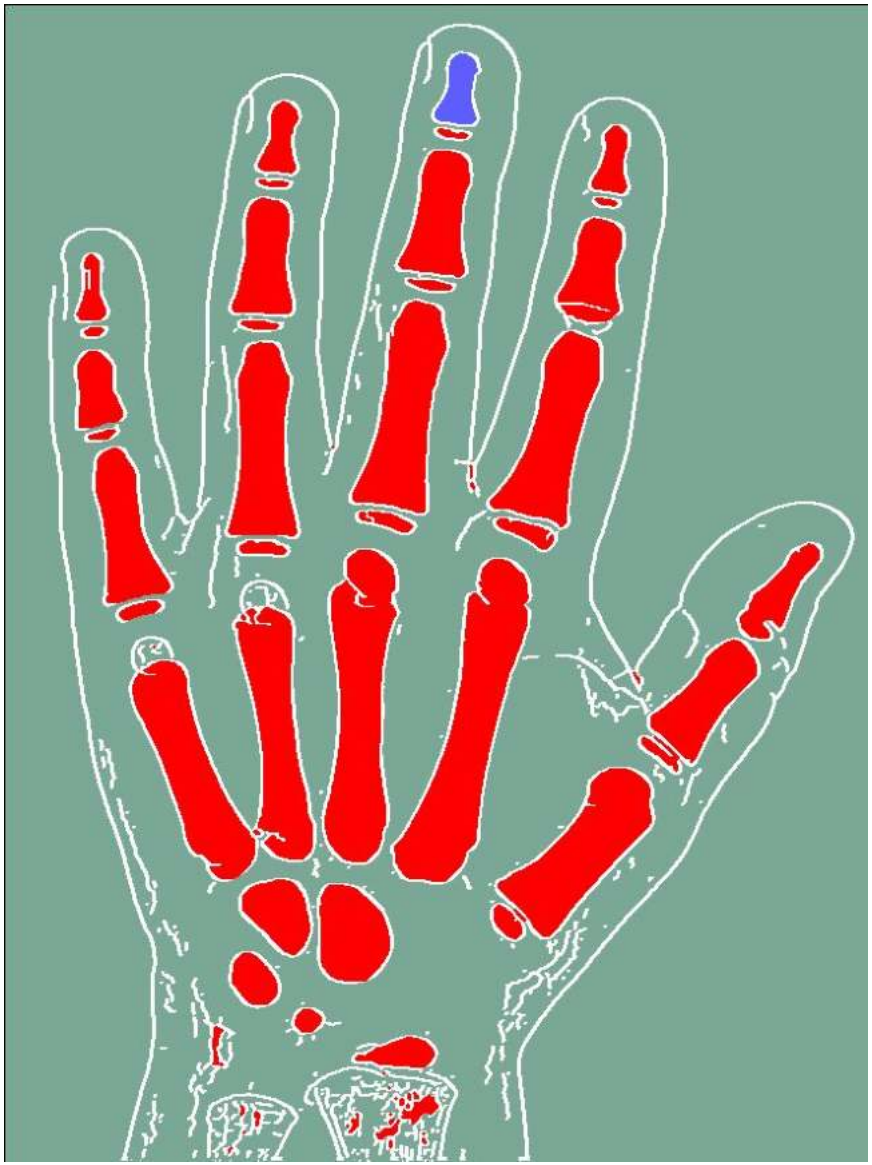
Composantes connexes



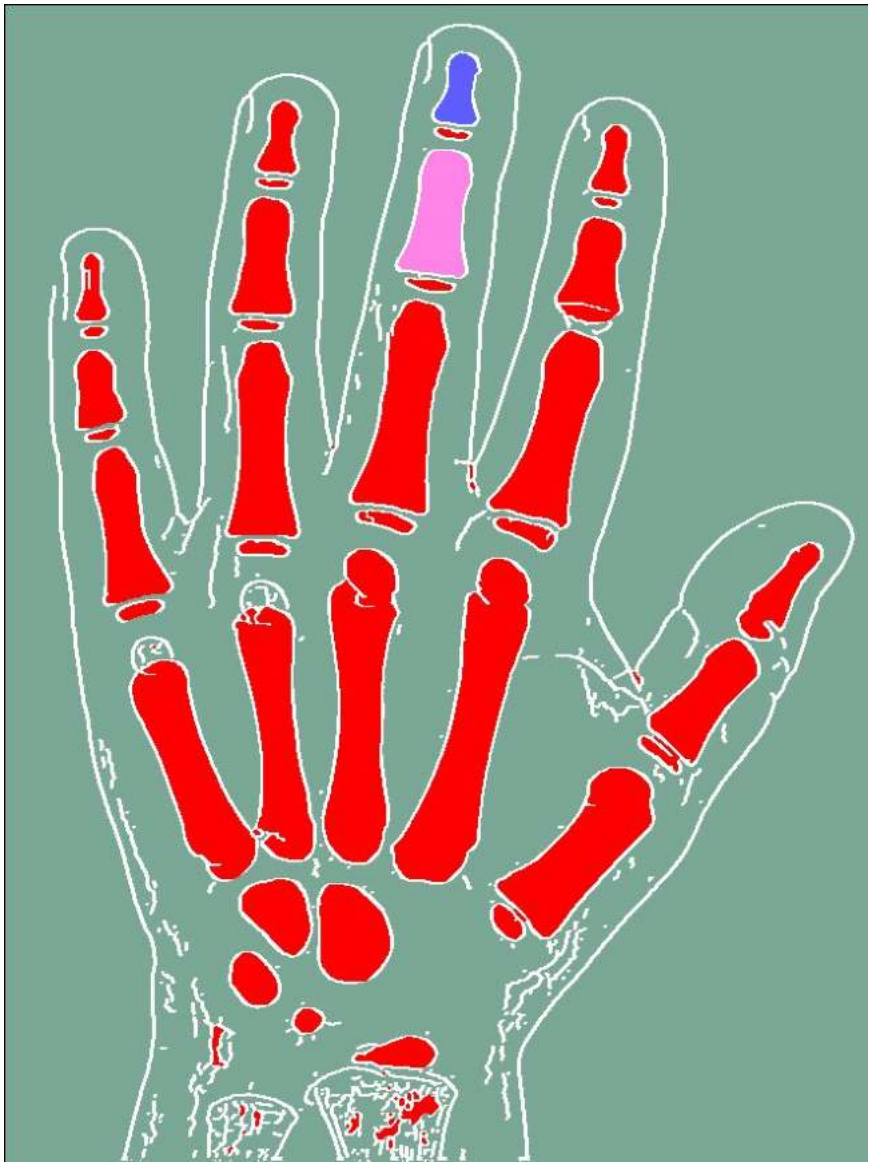
Différencier chaque os



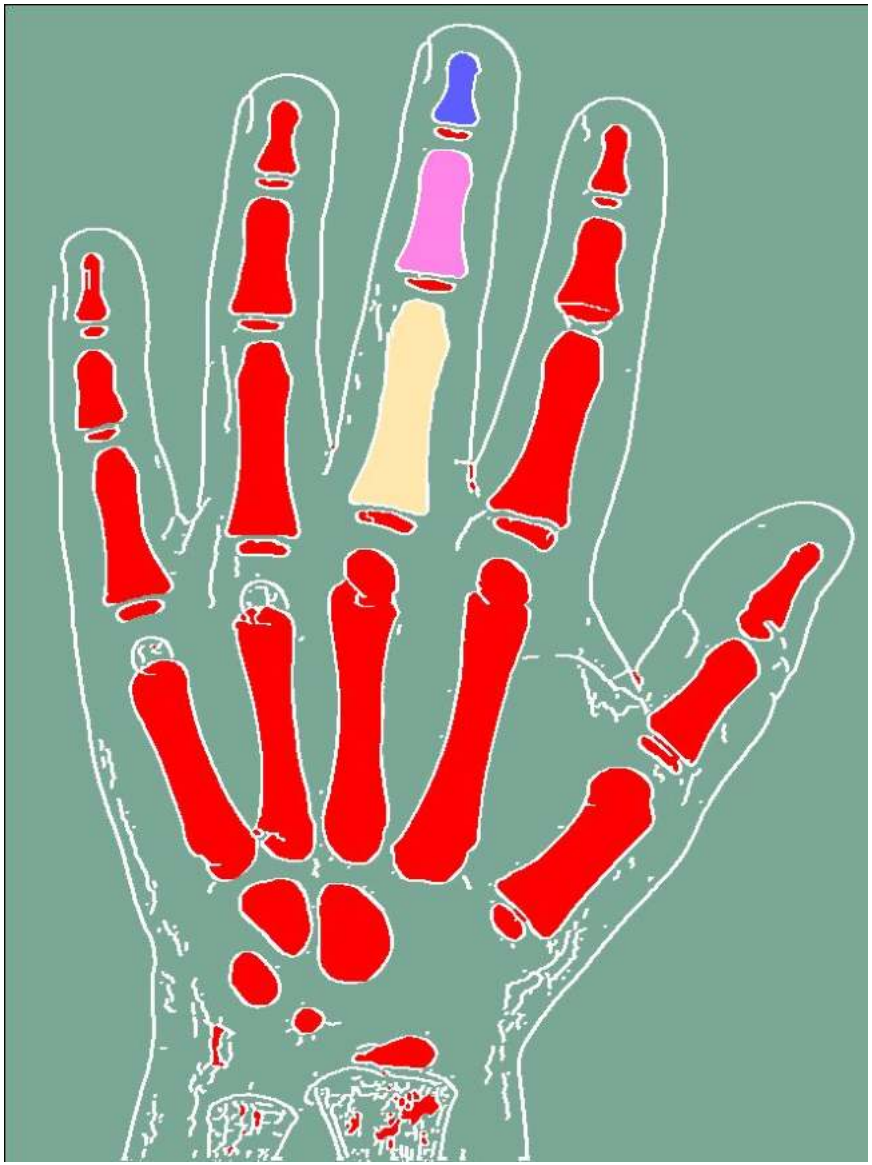
Différencier chaque os



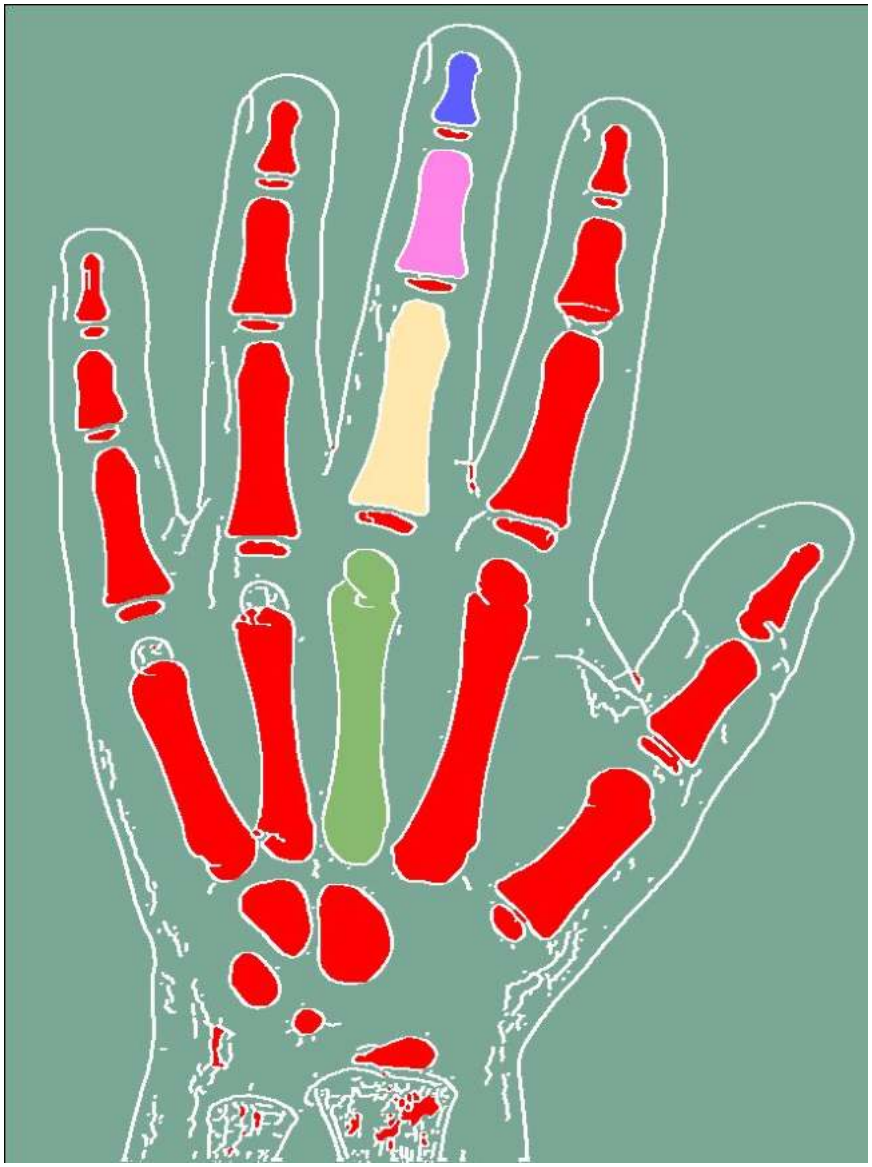
Différencier chaque os



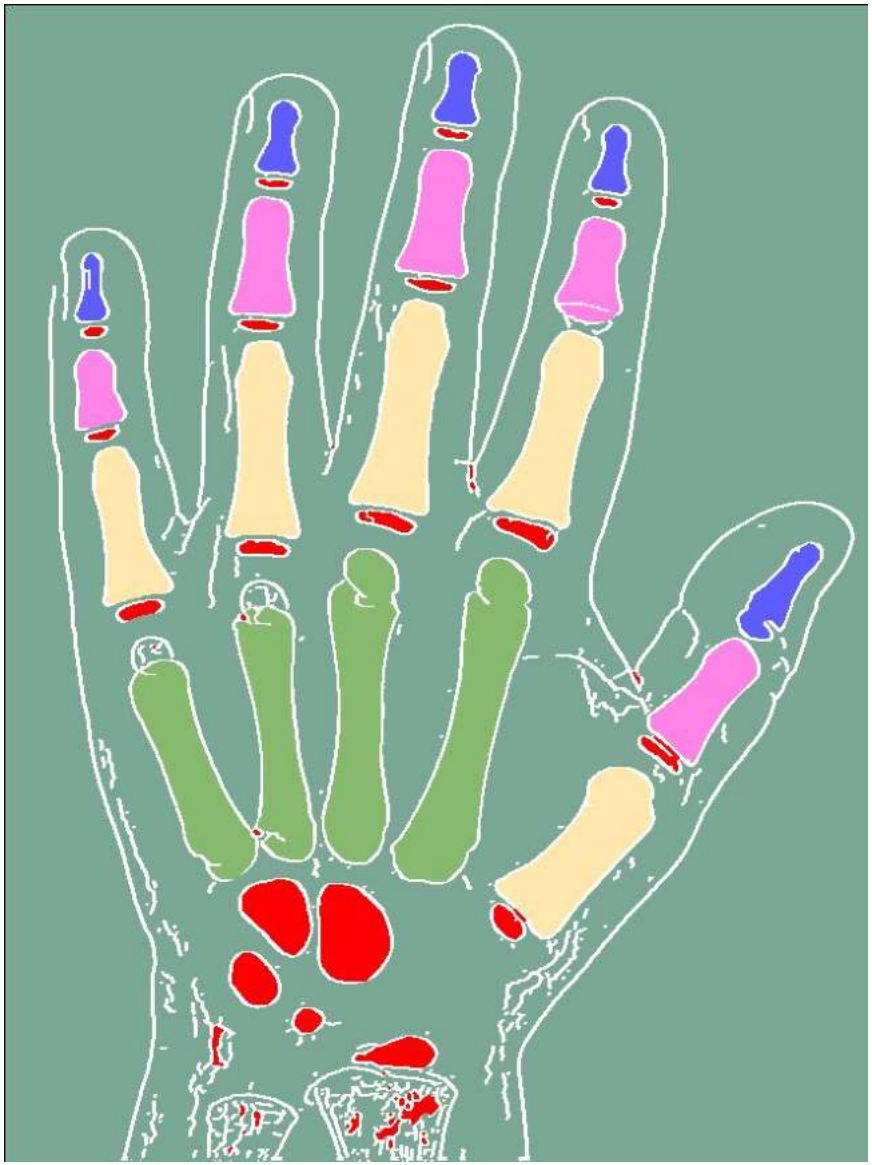
Différencier chaque os



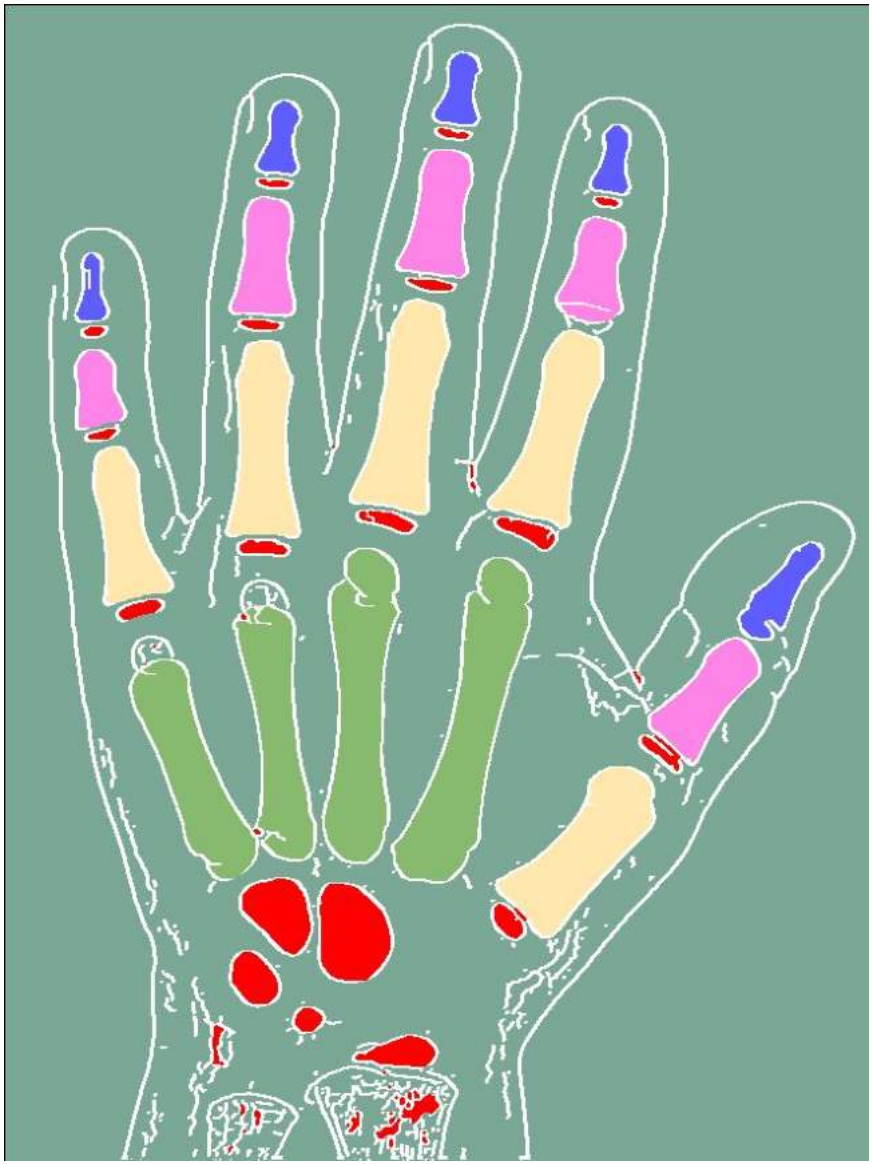
Différencier chaque os

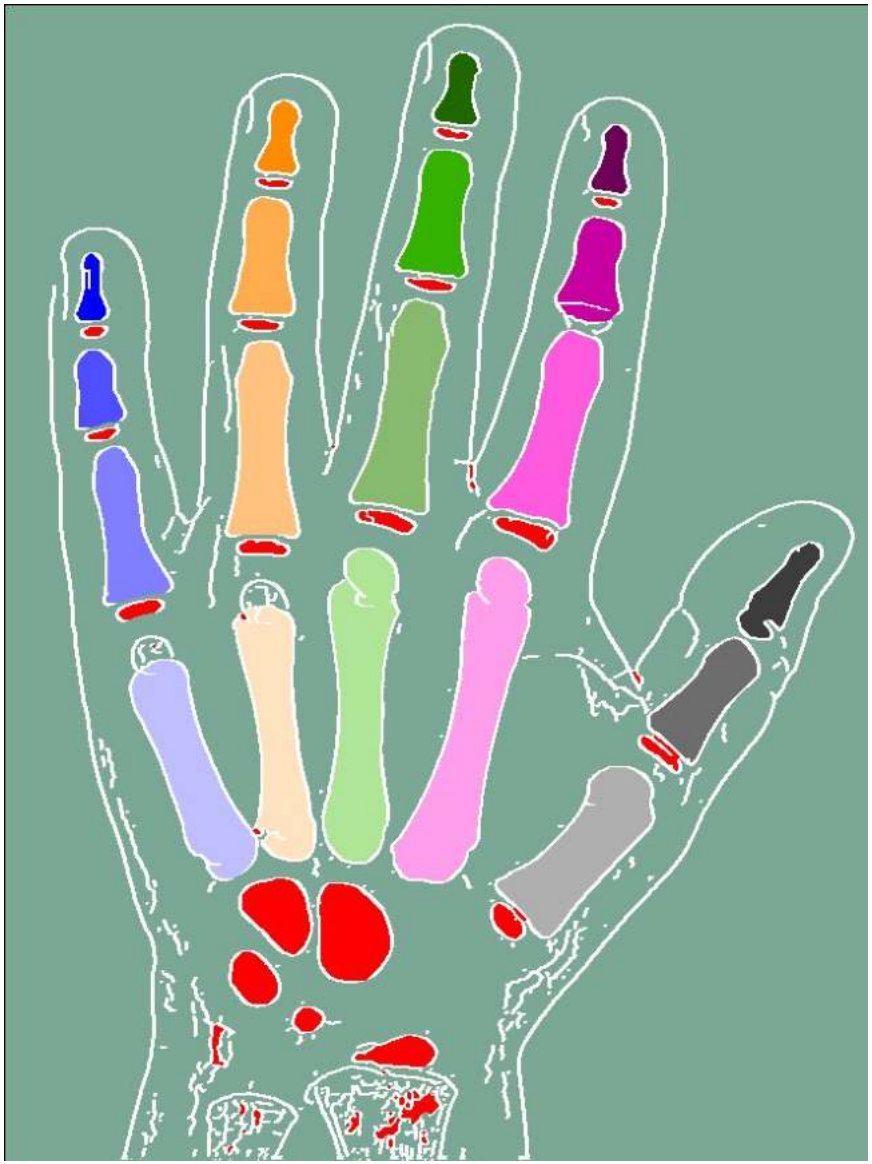


Différencier chaque os

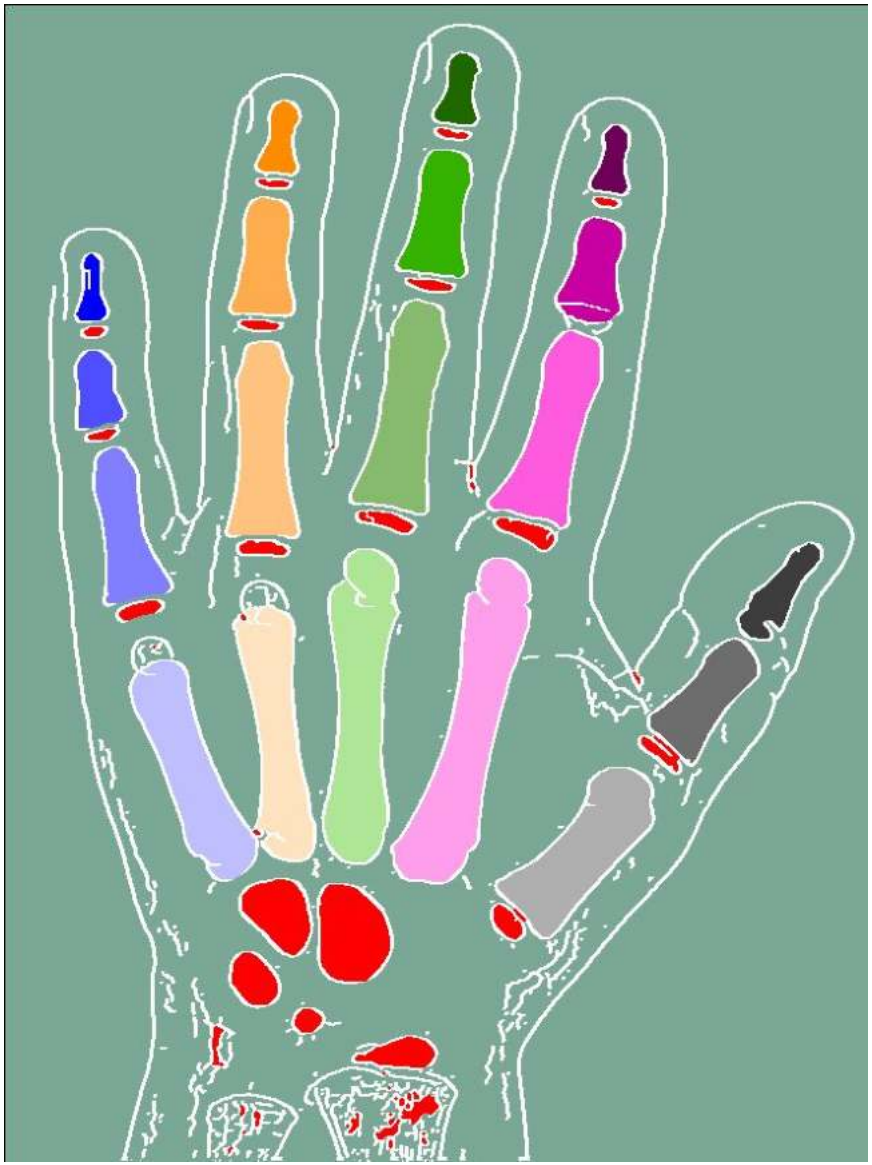


Différencier chaque os





Différencier chaque os



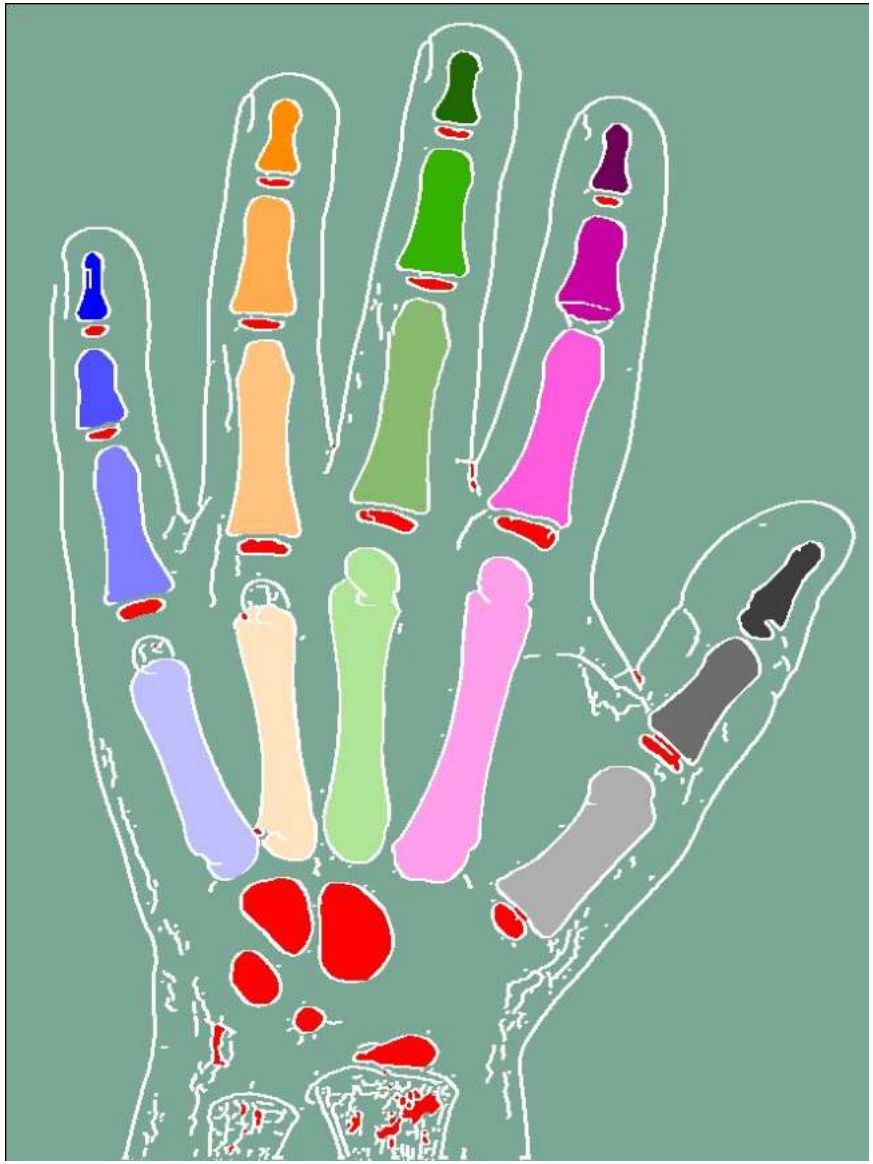
Utilisation des outils : paramètres pertinents

2 ans

9 ans

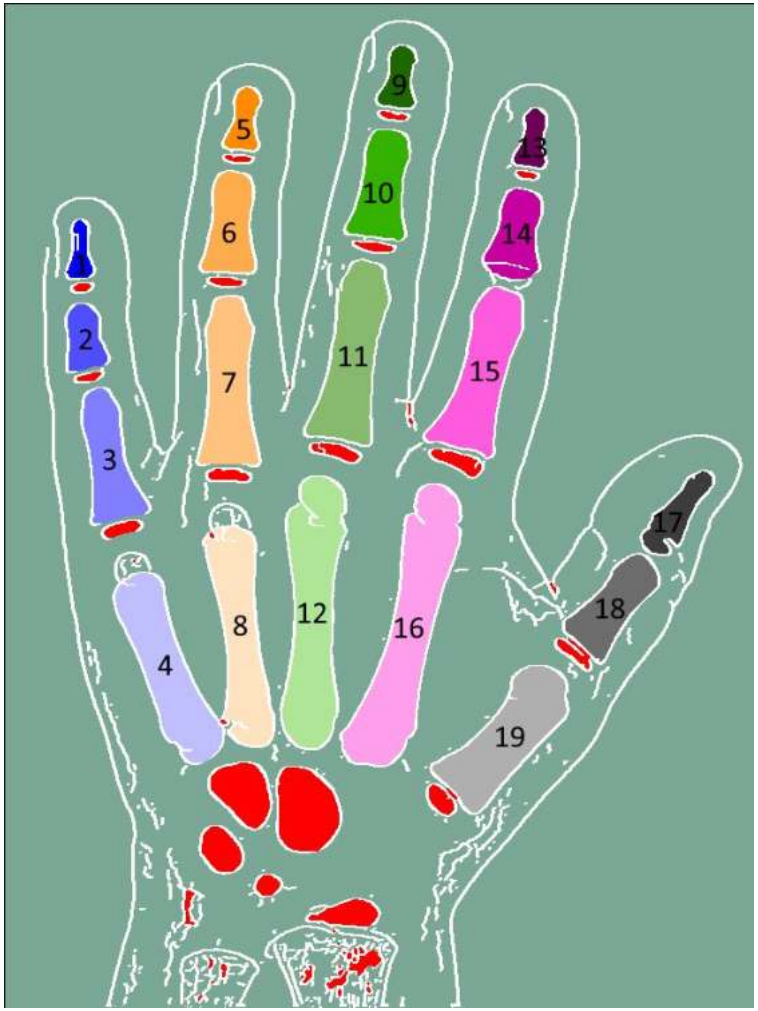
Utilisation des outils : paramètres pertinents

- Taille des os
- Distance entre les os



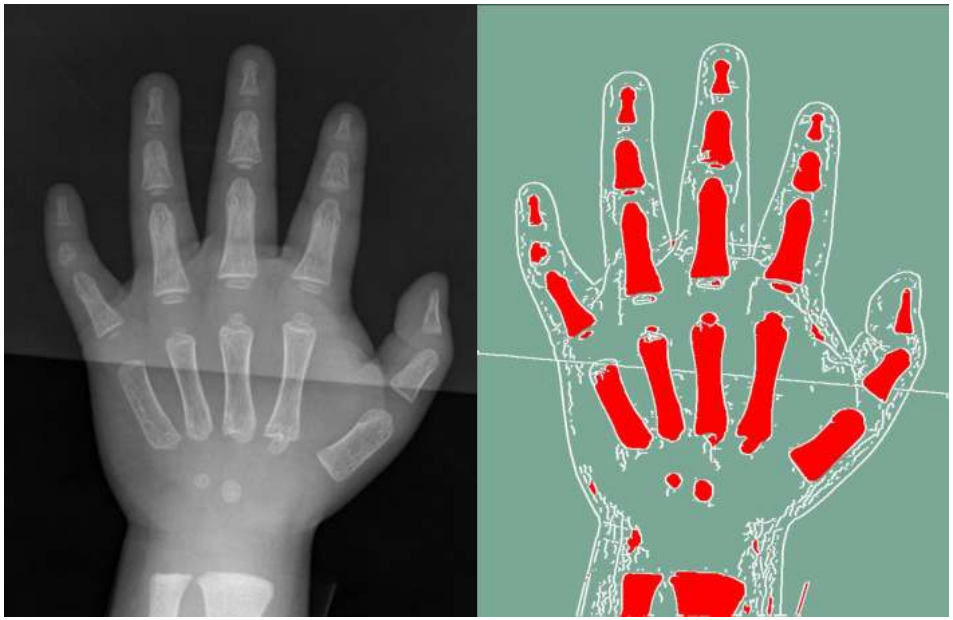
Utilisation des outils : paramètres pertinents

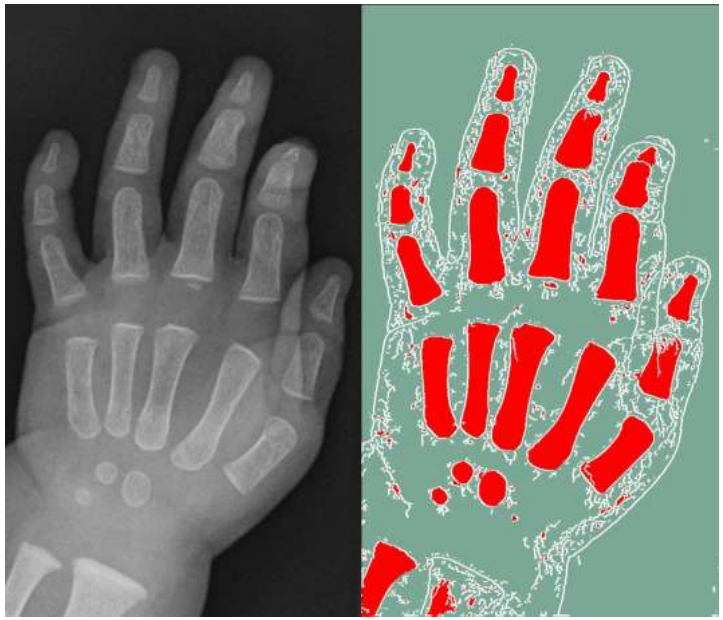
Os	Taille
1	5,83
2	8,96
3	13,46
4	19,93
5	6,37
6	10.33
7	17,09
8	22,05
9	6,29
10	11,19
11	18,43
12	27,01
13	5,92
14	9,09
15	16,95
16	28,2
17	8,96
18	27,01
19	28,20

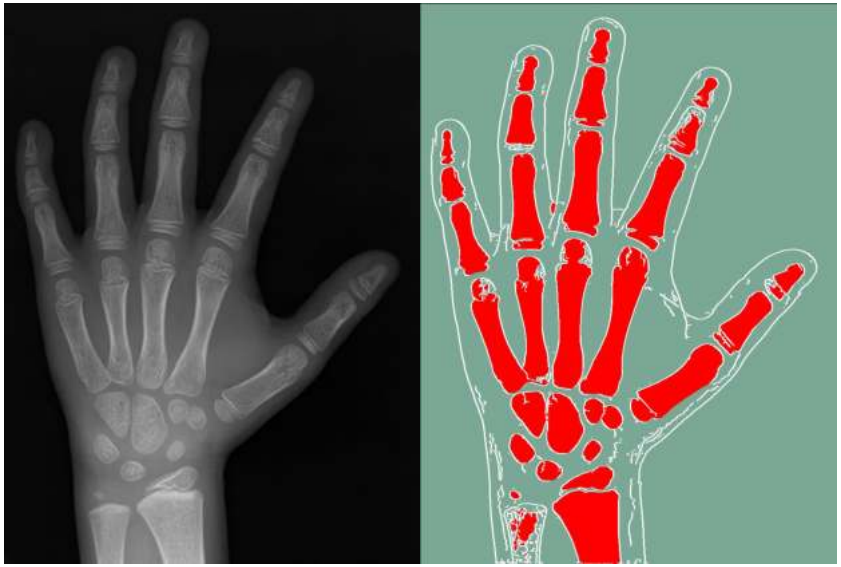


Os	Espacement
1-2	23
2-3	18,68
3-4	50,28
5-6	20,09
6-7	21,37
7-8	55
9-10	21,58
10-11	21,21
11-12	36
13-14	21,02
14-15	10,44
15-16	37,64
17-18	8,48
18-19	33,94

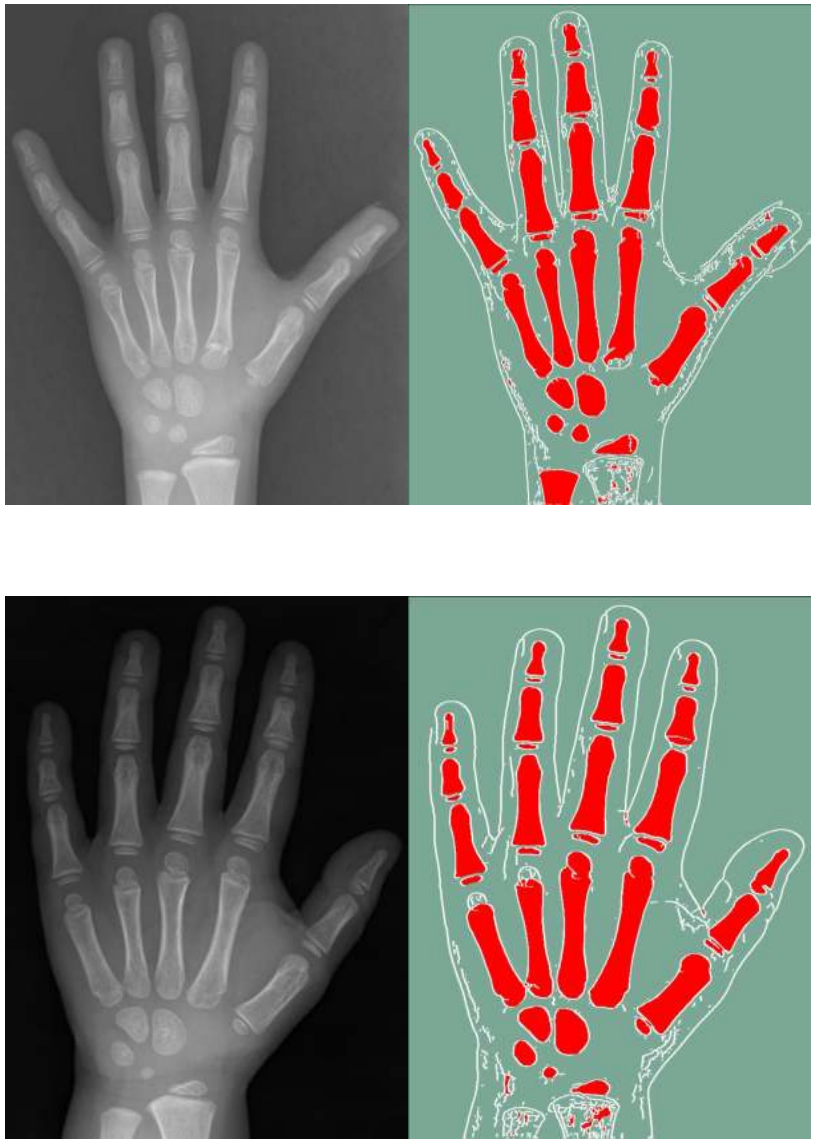
Utilisation des outils







Résultats



4 ans

Os	Taille
1	5,73
2	8,66
3	13,02
4	20,01
5	6,47
6	10.34
7	16,98
8	22,10
9	6,34
10	11,08
11	18,48
12	26,97
13	5,99
14	9,02
15	16,92
16	28,24
17	8,91
18	26,97
19	28,24

Os	Espacement
1-2	23,03
2-3	18,59
3-4	50,31
5-6	20,12
6-7	21,43
7-8	55,04
9-10	21,61
10-11	21,25
11-12	35,97
13-14	21,04
14-15	10,39
15-16	37,63
17-18	8,51
18-19	33,96

Résultats

8 ans
(âge réel 9 ans)

12 ans
(âge réel 11 ans)

4 ans
(âge réel 4 ans)

Résultats

8 ans
(âge réel 9 ans)

12 ans
(âge réel 11 ans)

4 ans
(âge réel 4 ans)

35 Images -> 23 images (65 %) ont donné un résultat positif à ± 2 ans

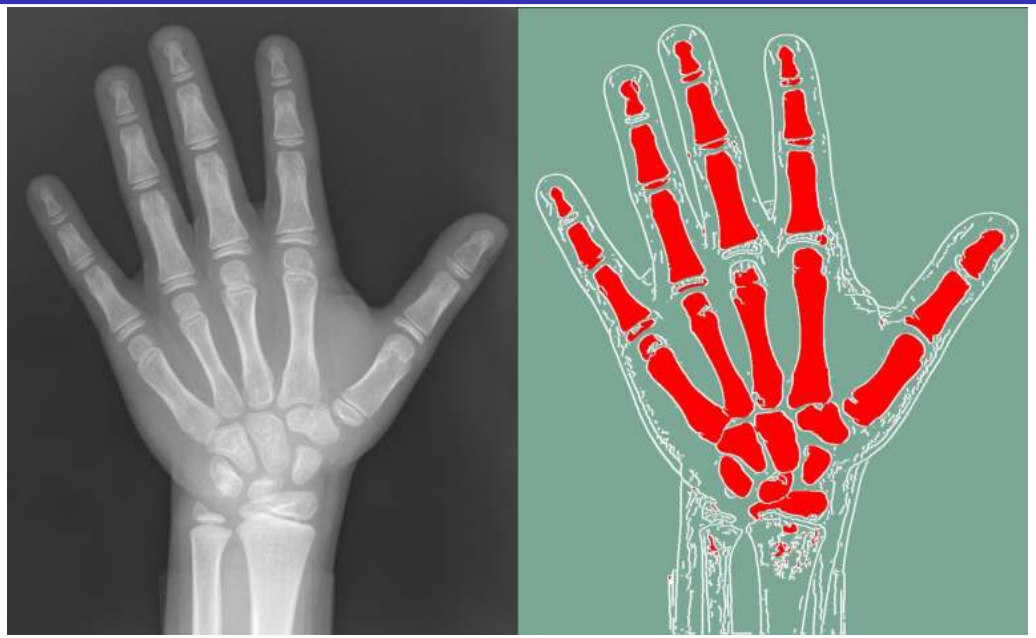
Conclusion

Améliorations possibles :

- Prise en compte des os du poignet
- Orientation de la main

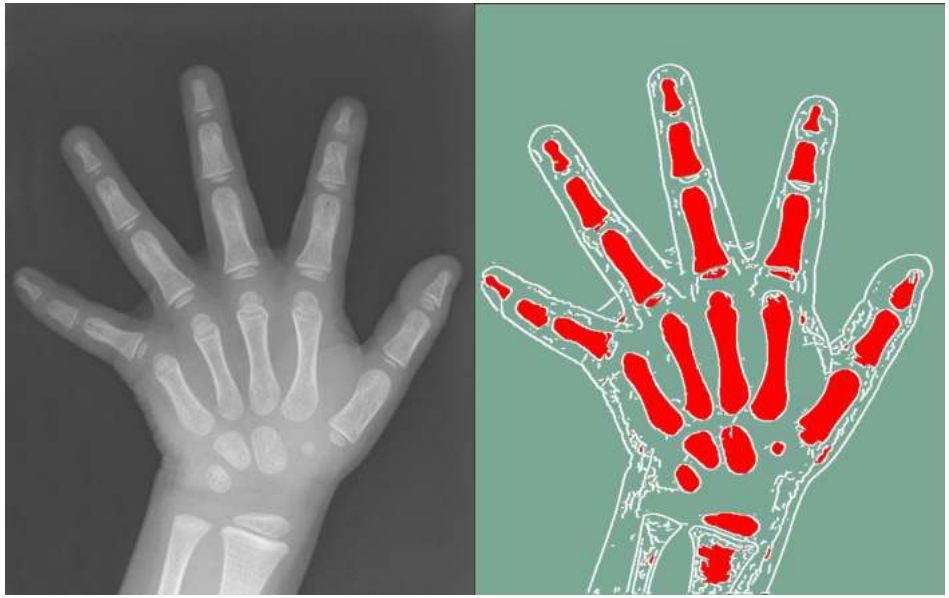
Annexe

Annexe



12 ans
(âge réel 11 ans)

4 ans
(âge réel 4 ans)



Annexe

```python
from tkinter import Image, LabelFrame
from typing import List
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from math import *
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import copy
import random as rd
import pickle

def rvba_to_rgb(image):
    m,n,_ = np.shape(image)
    image2 = np.zeros((m,n,3))
    for x in range(m):
        for y in range(n):
            image2[x,y] = [image[x][y][0],image[x][y][1],image[x][y][2]]
    return image2

# Convolution
def convoler(image, noyau):
    """Effectue la convolution de image avec le noyau de convolution
    Args:
        image : Image en noir et blanc sous forme d'un tableau Numpy
        noyau : Matrice de taille n*n

    Returns:
        L'image aprés avoir effectuer la convolution
    """
    m, n = np.shape(image)
    p, q = np.shape(noyau)
    img = np.zeros((m, n))
    p1 = (p - 1) // 2
    for i in range(p):
        for j in range(p):
            img[p1:-p1, p1:-p1] += (
                noyau[i, j] * image[i : m - p + 1 + i, j : n - q + 1 + j]
            )
    return img
```

Annexe

```python
# Gradient

def grad_image(image):
    """Cette fonction permet d'appliquer la méthode du gradient à une image

    Args:
        image : Image en noir et blanc sous forme d'un tableau Numpy

    Returns:
        grad, theta : grad la norme du gradient et theta la direction du gradient sous forme de tableau numpy de la même taille que image
    """
    m, n = np.shape(image)
    noyauGX = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    noyauGY = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    gradX = convoler(image, noyauGX)
    gradY = convoler(image, noyauGY)
    grad = np.sqrt(gradX**2 + gradY**2)
    theta = np.zeros((m, n))
    for x in range(m):
        for y in range(n):
            theta[x, y] = atan(gradY[x, y] / gradX[x, y])

    return grad, theta
```

Annexe

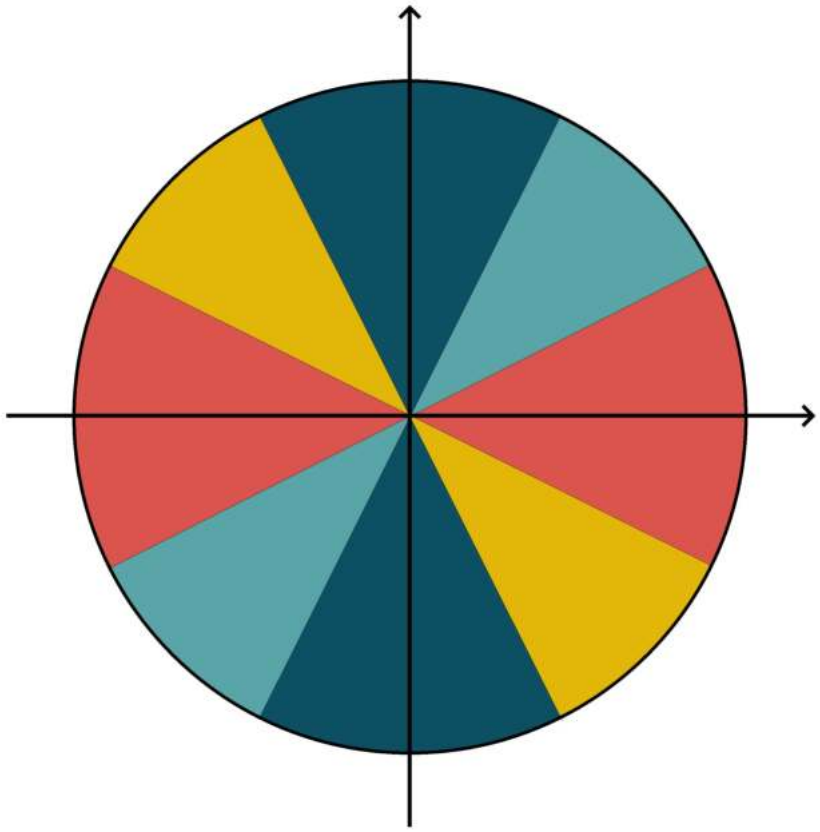
```python
# Contour

def seuil(grad, low, high):
    """Cette fonction seuil permet de supprimer les "faux" contour et de boucher les potentiel trou dans un contour

    Args:
        image : Image en noir et blanc sous forme d'un tableau Numpy
        low [float] : Compris entre 0 et 1
        high [float] : Compris entre 0 et 1
        grad : gradient de l'image sous forme d'un couple (grad, theta)

    Returns:
        Une nouvelle image de la même taille que image
    """
    image, theta = grad
    m, n = np.shape(image)
    img2 = np.zeros((m, n))
    for x in range(3, m - 3):
        for y in range(3, n - 3):
            angle = theta[x, y]
            voisins = []
            keep = False
            a = 0
            if img2[x, y] != 1:
                if image[x, y] < low:
                    img2[x, y] = 0
                elif image[x, y] > high:
                    img2[x, y] = 1

                elif (angle >= 2 * pi / 6 and angle <= pi / 2) or (
                    angle <= -2 * pi / 6 and angle >= -pi / 2
                ):
                    voisins.append((x, y + 1))
                    voisins.append((x, y - 1))
                    voisins.append((x, y + 2))
                    voisins.append((x, y - 2))
                    voisins.append((x, y + 3))
                    voisins.append((x, y - 3))
```

Annexe

```
104                 elif angle <= pi / 6 and angle >= -pi / 6:
105                     voisins.append((x + 1, y))
106                     voisins.append((x - 1, y))
107                     voisins.append((x + 2, y))
108                     voisins.append((x - 2, y))
109                     voisins.append((x + 3, y))
110                     voisins.append((x - 3, y))
111
112                 elif angle >= pi / 6 and angle <= 2 * pi / 6:
113                     voisins.append((x + 1, y - 1))
114                     voisins.append((x - 1, y + 1))
115                     voisins.append((x + 2, y - 2))
116                     voisins.append((x - 2, y + 2))
117                     voisins.append((x + 3, y - 3))
118                     voisins.append((x - 3, y + 3))
119
120                 elif angle >= -2 * pi / 6 and angle <= -pi / 6:
121                     voisins.append((x + 1, y + 1))
122                     voisins.append((x - 1, y - 1))
123                     voisins.append((x + 2, y + 2))
124                     voisins.append((x - 2, y - 2))
125                     voisins.append((x + 3, y + 3))
126                     voisins.append((x - 3, y - 3))
127
128                 for v in voisins:
129                     if image[v[0], v[1]] > high or img2[v[0], v[1]] != 0:
130                         keep = True
131
132                 if keep:
133                     for v in voisins:
134                         img2[v[0], v[1]] = 1
135                         img2[x, y] = 1
136     return img2
```

Annexe

```python
def contour_mince(image, grad):
    """Reduit la taille des contour à 2 pixels

    Args:
        image : Image en noir et blanc sous forme de tableau numpy
        grad : gradient de l'image sous forme d'un couple (grad, theta)

    Returns:
        Une nouvelle image de la même taille que image
    """
    img, theta = grad
    m, n = np.shape(image)
    for x in range(2, m - 2):
        for y in range(2, n - 2):
            if image[x, y] != 0:
                keep = True
                voisins = []
                angle = theta[x, y]

                if (angle >= 2 * pi / 6 and angle <= pi / 2) or (
                    angle <= -2 * pi / 6 and angle >= -pi / 2
                ):
                    voisins.append((x + 1, y))
                    voisins.append((x - 1, y))
                    voisins.append((x + 2, y))
                    voisins.append((x - 2, y))

                elif angle <= pi / 6 and angle >= -pi / 6:
                    voisins.append((x, y + 1))
                    voisins.append((x, y - 1))
                    voisins.append((x, y + 2))
                    voisins.append((x, y - 2))

                elif angle >= pi / 6 and angle <= 2 * pi / 6:
                    voisins.append((x + 1, y + 1))
                    voisins.append((x - 1, y - 1))
                    voisins.append((x + 2, y + 2))
                    voisins.append((x - 2, y - 2))

                elif angle >= -2 * pi / 6 and angle <= -pi / 6:
                    voisins.append((x + 1, y - 1))
                    voisins.append((x - 1, y + 1))
                    voisins.append((x + 2, y - 2))
                    voisins.append((x - 2, y + 2))

                for v in voisins:
                    if img[x, y] < img[v[0], v[1]]:
                        keep = False
                if not keep:
                    image[x, y] = 0
                else:
                    image[x + 1, y] = 1
                    image[x - 1, y] = 1
                    image[x, y + 1] = 1
                    image[x, y - 1] = 1
                    image[x + 1, y + 1] = 1
                    image[x - 1, y - 1] = 1
                    image[x + 1, y - 1] = 1
                    image[x - 1, y + 1] = 1
                    image[x, y] = 1
    return image
```



Annexe

```
201
202 def contour(image):
203     """Permet de trouver les contour d'une image
204
205     Args:
206         image : image en noir et blanc sous la forme d'un tableau numpy
207
208     Returns:
209         Une nouvelle image avec les contour en blanc (1) et le fond en noir (0)
210     """
211     m, n = np.shape(image)
212
213     MatriceFlou = (1 / 159) * np.array(
214         [
215             [2, 4, 5, 4, 2],
216             [4, 9, 12, 9, 4],
217             [5, 12, 15, 12, 5],
218             [2, 4, 5, 4, 2],
219             [4, 9, 12, 9, 4],
220         ]
221     )
222     img = convoler(image, MatriceFlou)
223
224     grad = grad_image(img)
225     img = seuil(grad, 0.05, 0.1)
226     img = contour_mince(img, grad)
227     return img
228
229
230 # Composante connexe
231
232 class Pile:
233     def __init__(self):
234         self.valeurs = []
235
236     def ajouter(self, valeur):
237         self.valeurs.append(valeur)
238
239     def enlever(self):
240         #if self.valeurs:
241         return self.valeurs.pop()
242
243     def estVide(self):
244         return self.valeurs == []
245
```

Annexe

```python
def grey_to_color(image):
    """Transforme une image noir et blanc en une image RGB

    Args:
        image : Image noir et blanc sous forme d'un tableau numpy

    Returns:
        Image couleur avec 3 composantes [R,G,B] sous forme d'un tableau numpy
    """
    m, n = np.shape(image)
    res = np.zeros((m, n, 3))
    for i in range(m):
        for j in range(n):
            a = image[i, j]
            res[i, j] = [a, a, a]
    return res
```

Annexe

```
264
265 def color_to_grey(image):
266     """Transforme une image noir et blanc en une image RGB
267
268     Args:
269         image : Image noir et blanc sous forme d'un tableau numpy
270
271     Returns:
272         Image couleur avec 3 composantes [R,G,B] sous forme d'un tableau numpy
273     """
274     m, n, _ = np.shape(image)
275     res = np.zeros((m, n))
276     for i in range(m):
277         for j in range(n):
278             a = image[i, j]
279             res[i, j] = a[0]
280     return res
281
282
283 def EstEgale(L1: list, L2: list) -> bool:
284     """Test l'égalité entre deux liste python
285
286     Args:
287         L1 (list): Liste 1
288         L2 (list): Liste 2
289
290     Returns:
291         bool: True si les deux liste sont égales, False sinon
292     """
293     n = len(L2)
294     for i in range(n):
295         if L1[i] != L2[i]:
296             return False
297     return True
298
```

Annexe

```
299
300 def colorier(image, point: tuple, couleur: list):
301     """Colorie la zone autour d'un point donné
302
303     Args:
304         image : Image RGB sous forme d'un tableau numpy
305         point (tuple): (x,y) a partir du
306         couleur (list): [R,G,B]
307
308     Returns:
309         image avec la zone autour du point colorier
310     """
311     m, n, _ = np.shape(image)
312     res = copy.deepcopy(image)
313     p = Pile()
314     x, y = point
315     couleurPoint = image[x, y]
316     p.ajouter(point)
317     p.ajouter((x + 1, y))
318     p.ajouter((x - 1, y))
319     p.ajouter((x, y - 1))
320     p.ajouter((x, y + 1))
321     while not p.estVide():
322         xs, ys = p.enlever()
323         voisins = [(xs + 1, ys), (xs - 1, ys), (xs, ys - 1), (xs, ys + 1)]
324         for v in voisins:
325             if (
326                 v[0] > 0
327                 and v[0] < m
328                 and v[1] > 0
329                 and v[1] < n
330                 and EstEgale(image[v], couleurPoint)
331                 and not EstEgale(res[v], couleur)
332             ):
333                 p.ajouter(v)
334         res[xs, ys] = couleur
335     return res
336
337
338 def cc(image):
339     """Trouver les composantes connexes d'une image, et les affiche en une couleur aléatoire
340
341     Args:
342         image : Image RGB sous forme d'un tableau numpy
343
344     Returns:
345         Image avec toutes les composantes connexe mises en couleur
346     """
347     m, n, _ = np.shape(image)
348     res = copy.deepcopy(image)
349     for i in range(2, m - 1):
350         for j in range(2, n - 1):
351             if EstEgale(res[i, j], [0, 0, 0]):
352                 couleur = [rd.random(), rd.random(), rd.random()]
353                 res = colorier(res, (i, j), couleur)
354     return res
355
```

Annexe

```python
def cc2(image):
    """Trouver les comosantes connexes d'une image, et affiche la premiere en une couleur aléatoire et les autres en rouge

    Args:
        image : Image RGB sous forme d'un tableau numpy

    Returns:
        Image avec toutes les composantes connexe mises en couleur
    """
    m, n, _ = np.shape(image)
    m, n, _ = np.shape(image)
    res = copy.deepcopy(image)
    res = colorier(res, (10, 10), [123 / 255, 167 / 255, 151 / 255])
    for i in range(2, m - 1):
        for j in range(2, n - 1):
            if EstEgale(res[i][j], [0, 0, 0]):
                couleur = [1, 0, 0]
                res = colorier(res, (i, j), couleur)
    return res


def clear_cc(image):
    """Permet d'enlever les traits blancs à l'interieur des os

    Args:
        image : Image en noir et blanc sous la forme d'un tableau numpy

    Returns:
        Renvoie une nouvelle image de la même taille que image
    """
    m, n, _ = np.shape(image)
    for x in range(2, m - 2):
        for y in range(2, n - 2):
            if EstEgale(image[x, y], [1, 1, 1]):
                compteur = 0
                voisins = [
                    [x + 1, y],
                    [x - 1, y],
                    [x, y + 1],
                    [x, y - 1],
                    [x + 1, y + 1],
                    [x - 1, y - 1],
                    [x + 1, y - 1],
                    [x - 1, y + 1],
                ]
                for v in voisins:
                    if EstEgale(image[v[0], v[1]], [1, 0, 0]):
                        compteur += 1
                    if compteur >= 4:
                        image[x, y] = [1, 0, 0]
    return image
```

Annexe

```python
def recup_cord_cc(image, point):
    """Renvoie la liste des point d'une même composante connexe

    Args:
        image : image RVB sous forme d'un tableau numpy
        point (tuple): un point appartenant à la composante connexe

    Returns:
        list: La liste des points de la composante connexe
    """
    Liste = []
    res = copy.deepcopy(image)
    m, n, _ = np.shape(image)
    p = Pile()
    x, y = point
    couleurPoint = image[x, y]
    p.ajouter(point)
    p.ajouter((x + 1, y))
    p.ajouter((x - 1, y))
    p.ajouter((x, y - 1))
    p.ajouter((x, y + 1))
    while not p.estVide():
        xs, ys = p.enlever()
        voisins = [(xs + 1, ys), (xs - 1, ys), (xs, ys - 1), (xs, ys + 1)]
        for v in voisins:
            if (
                v[0] > 0
                and v[0] < m
                and v[1] > 0
                and v[1] < n
                and EstEgale(image[v], couleurPoint)
                and not EstEgale(res[v], [0, 0, 0])
            ):
                p.ajouter(v)
        res[xs, ys] = [0, 0, 0]
        Liste.append([xs, ys])
    return Liste, res

```

Annexe

```
448
449 def liste_cc(image):
450     """Renvoie la liste de toutes les composantes connexes
451
452     Args:
453         image : Image RVB sous la forme d'une tableau numpy
454
455     Returns:
456         list: une liste de liste de point correspondant à chaque composante connexe
457     """
458     m, n, _ = np.shape(image)
459     image2 = copy.deepcopy(image)
460     L = []
461     for i in range(2, m - 1):
462         for j in range(2, n - 1):
463             if EstEgale(image2[i, j], [1, 0, 0]):
464                 l, image2 = recup_cord_cc(image2, (i, j))
465                 L.append(l)
466     return L
467
468
469
470 # Outils
471
472 def maximum(a,b):
473     """Retourne le plus grand nombre entre a et b
474
475     Args:
476         a (int): premier nombre
477         b (int): deuxieme nombre
478
479     Returns:
480         int: le max entre a et b
481     """
482     if a > b:
483         return a
484     return b
485
486
487 def max_liste(liste: list) -> int:
488     """retourne le max d'une liste
489
490     Args:
491         liste (list): une liste
492
493     Returns:
494         int: le maximim de la liste liste
495     """
496     temp = liste[1]
497     for x in liste:
498         if x > temp:
499             temp = x
500     return temp
501
```

Annexe

```python
def min_liste(liste: list) -> int:
    """retourne le min d'une liste

    Args:
        liste (list): une liste

    Returns:
        int: le maximim de la liste liste
    """
    temp = liste[1]
    for x in liste:
        if x < temp:
            temp = x
    return temp

def reduire_list(liste: list, nbr: int) -> list:
    """Enléve les sous listes ayant une longeur < nbr

    Args:
        liste (list(list)): une liste contenant des sous listes
        nbr (int): longeur minimum des sous liste

    Returns:
        list: Liste de liste, où toutes les sous liste ont une longeur >= nbr
    """
    L = []
    for x in liste:
        if len(x) >= nbr:
            L.append(x)
    return L
```

Annexe

```python
def barycentre(listePoint: list[list]) -> list:
    """Trouve le barycentre d'une liste de points

    Args:
        listePoint (list): liste des points

    Returns:
        list: coordinée [x,y,z] du barycentre
    """
    n = len(listePoint)
    sx = 0
    sy = 0
    for p in listePoint:
        sx += p[0]
        sy += p[1]
    return [int(sx / n), int(sy / n)]


def save_list(Liste: list, fichier: str) -> None:
    """Enregistrer une liste python dans un fichier

    Args:
        Liste (list): La liste à enregistrer
        fichier (str): nom du fichier
    """
    with open(fichier, "wb") as temp:
        pickle.dump(Liste, temp)
```

Annexe

```python
def recup_list(fichier: str) -> list:
    """Recupére une liste enregistrer dans un fichier

    Args:
        fichier : Nom du fichier

    Returns:
        list: La liste enregistrer dans le fichier
    """
    with open(fichier, "rb") as temp:
        liste = pickle.load(temp)
    return liste


def distance(a, b):
    """Donne la distance entre deux points

    Args:
        a (list): [x1,y1]
        b (list): [x2,y2]

    Returns:
        float: distance entre a et b
    """
    return sqrt(abs(a[0] - b[0]) ** 2 + abs(a[1] - b[1]) ** 2)


def plus_proche_point(liste: list, indice: int) -> int:
    """Trouve le point le plus proche d'un autre

    Args:
        liste (list): liste de point
        indice (int): indice du point de réference

    Returns:
        int: l'indice du point le plus proche de liste[indice]
    """
    mini = distance(liste[indice], liste[indice - 1])
    indiceMini = indice - 1
    for i in range(len(liste)):
        d = distance(liste[indice], liste[i])
        if d < mini and d != 0:
            mini = d
            indiceMini = i
    return indiceMini

```

Annexe

```python
def longeur_os(os: list[list[int]]) -> int:
    """Renvoie la longeur d'un os

    Args:
        os (list): liste des points composant l'os

    Returns:
        int: distance entre le point le plus haut et le point le plus bas (selon y)
    """
    liste = [x[0] for x in os]
    return max_liste(liste) - min_liste(liste)

def volume_os(os):
    """Renvoie le volume d'un os

    Args:
        os (list): Liste des points composant l'os

    Returns:
        int: volume de l'os
    """
    return len(os)

def extremite_os(liste:list):
    liste.sort()
    return liste[0], liste[-1]


#Différencier les os

def os_en_dessous(Liste, indice):
    b = Liste[indice][1]
    dmin = 8000
    indiceMin = 0
    for i in range(len(Liste)):
        if i != indice:
            h = Liste[i][0]
            d = distance(h,b)
            if d < dmin:
                dmin = d
                indiceMin = i
    return Liste[indiceMin], indiceMin
```

Annexe

```
653 def find_phalanges(os:list):
654     """Differencie les os de la première, deuxième, troisième et quatrième phalanges
655 
656     Args:
657         os (list): liste de liste des point des os
658 
659     Returns:
660         list: [a,b,c,d] ou a,b,c,d sont les listes des os des phalanges 1, 2, 3 et 4
661     """
662 
663     res = [[], [], [], []] #Liste des os dans l'ordre de la hauteur
664     L = []
665     for x in os:
666         h,b = extremite_os(x)
667         L.append( [h,b, x, False] ) #Liste de Liste (point le plus haut, point le plus bas, la liste des points,
    boolean utile par la suite)
668 
669     L.sort()
670 
671     NbrColorier = 0
672     indice = 0
673     while NbrColorier < 19:
674         if indice == 50:
675             break
676         point = L[indice]
677         if not L[indice][3]:
678             res[0].append(point[2])
679             NbrColorier += 1
680             L[indice][-1] = True
681             indiceTemp = indice
682 
683             over = 0
684             j=0
685             while j < 3:
686                 if over == 20:
687                     break
688                 nouveau, indiceTemp = os_en_dessous(L, indiceTemp)
689                 if len(nouveau[2]) > len(point[2]) and not nouveau[-1]:
690                     res[j+1].append(nouveau[2])
691                     NbrColorier += 1
692                     j+=1
693                 L[indiceTemp][-1] = True
694                 over += 1
695         indice += 1
696     return res
```

Annexe

```
698 def tri_barycentre(Liste):
699     """Fonction utilisée pour faire un tri selon la deuxieme coordonée d'un barycentre d'une liste de point
700 
701     Args:
702         Liste (list[tuple]): Liste de point
703 
704     Returns:
705         float: Deuxiéme coordonée du barycentre
706     """
707     return barycentre(Liste)[1]
708 
709 
710 def find_bones(listeP):
711     """Renvoie une liste d'os triée gauche à droite et de haut en bas
712 
713     Args:
714         listeP (list[tuple]): Liste des phalanges
715 
716     Returns:
717         list: liste d'os triée
718     """
719     sortie = []
720 
721     for p in listeP:
722         a = sorted(p, key = tri_barycentre)
723         for x in a:
724             sortie.append(x)
725 
726     return sortie
727 
728 
729 def second_mem(l):
730     """ Utilisé pour un tri, permet de trier par rapport au deuxieme element d'une liste"""
731     return l[1]
```

Annexe

```python
def distance_entre_os(liste1:list, liste2:list) -> list:
    """Calcul la distance entre deux os donnés sous forme de liste

    Args:
        liste1 : Une liste de point d'un os
        liste2 : Une liste de point d'un os different de la premiere

    Returns:
        list: Une liste de 3 elements, les deux points de distance minimal (le 1er de liste1 et le 2ème de liste2) et la distance
    """
    mini = distance(liste1[0], liste2[0])
    p1 = liste1[0]
    p2 = liste2[0]
    for i in liste1:
        for j in liste2:
            if distance(i,j) < mini:
                mini = distance(i,j)
                p1 = i
                p2 = j
    return [p1, p2, mini]

def espace_entre_os(os):
    """Renvoie la liste des distances entre les os

    Args:
        os (lise): liste des os

    Returns:
        list: La liste des distances entre les os : [1er os, 2ème os, distance entre les deux]
    """
    res = []
    for i in range(14):
        haut = sorted(os[i])[-len(os[i])//5:]
        bas = sorted(os[i+5])[:len(os[i+5])//5]
        res.append(distance_entre_os(haut, bas))
    return res
```

Annexe

```python
def plus_proche_element(x, L):
    """Renvoie l'indice de l'element le "plus proche" de x

    Args:
        x (float): valeur à verifier
        L (list): liste de flotant

    Returns:
        int: indice de l'element le "plus proche" de x
    """
    d = abs(x-L[0])
    indice = 0
    for i in range(len(L)):
         if abs(L[i] - x) < d:
             d = abs(L[i] - x)
             indice = i
    return indice

def age(base, test):
    """Permet de trouver les similitude entre les données d'une main et les statistiques

    Args:
        base (liste): Liste de tailles/Espacement moyen de chaque âge
        test (liste): Liste de tailles/espacement d'une main dont on veut determiner l'âge

    Returns:
        liste: liste des âges auquels correspond chaque os
    """
    L = []
    for i in range(len(test)):
        age = plus_proche_element(test[i], L[i])
        L.append(age+1)

    return L

def max_nombre_occurence(liste):
    """Cherche la valeur qui apparait le plus de fois dans une liste

    Args:
        Liste (list): _description_

    Returns:
        int: valeur dont le nombre d'occurence est la plus grande
    """
    maxOcc = liste.count(liste[0])
    maxVal = Liste[0]
    for x in liste:
        if liste.count(x) > maxOcc:
            maxOcc = liste.count(x)
            maxVal = x
    return maxVal
```