

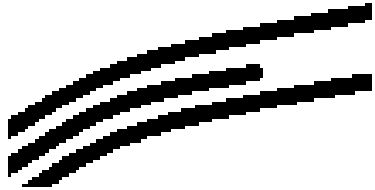
Comparaison d'empreintes digitales-Analyse

Sommaire :

- Objectif du programme informatique
- Principe général
- Présentation de quelques fonctions
- Limites et améliorations possibles

1. Objectif du programme informatique

- comparaison de deux empreintes digitales à partir des minuties
- types de minuties présentes dans une empreinte :



Terminaison



Bifurcation simple ou
point triple



Îlot

Nous n'allons considérer que l'étude des bifurcations simples

2. Principe général

3 étapes :

- binarisation
- 2 squelettisations en parallèle : Blanc + Noir
- sélections des minuties

binarisation = mise en noir et blanc



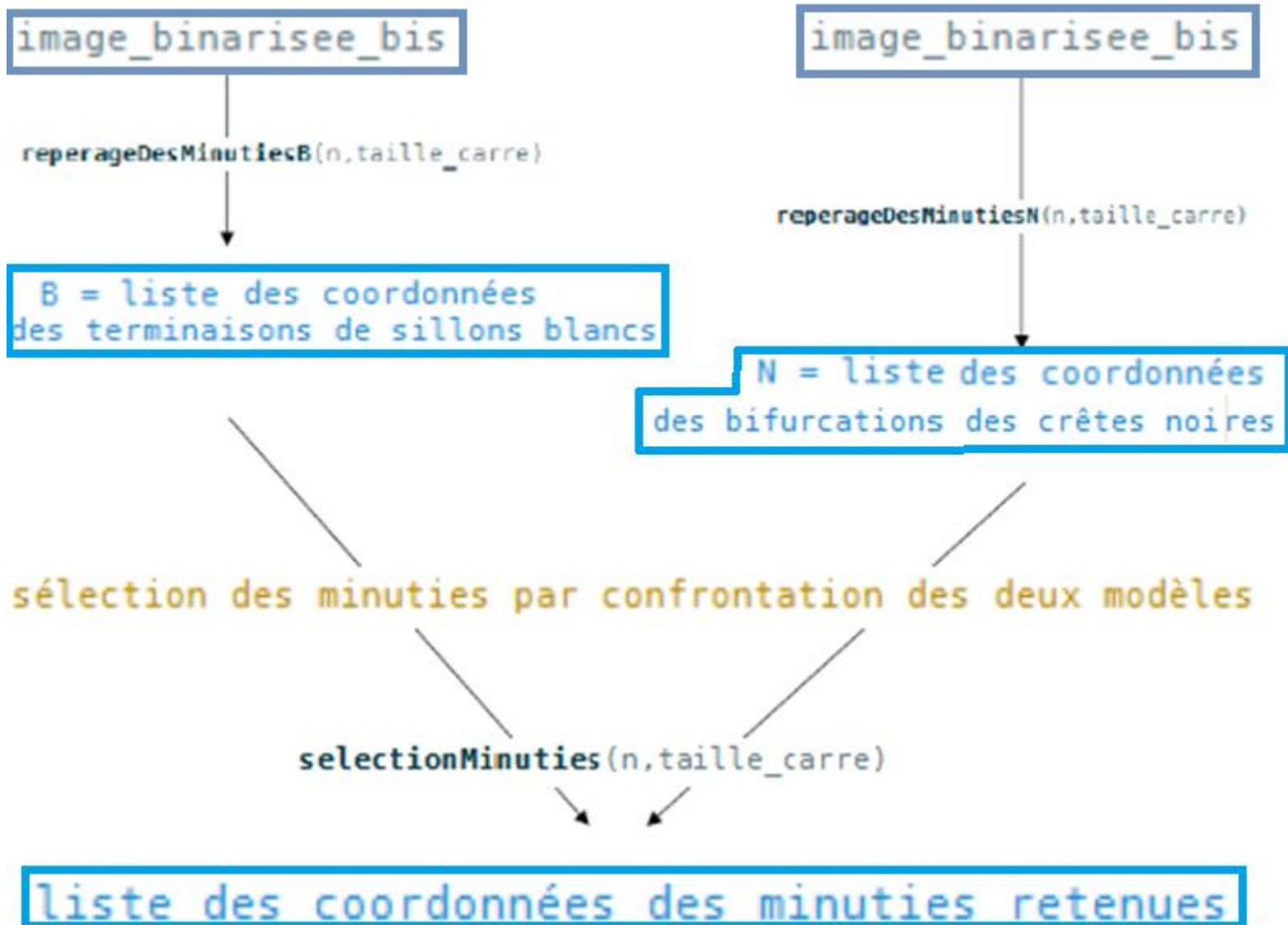
`image=Image.open('Empreinte.pgm')`

`moyenneCarre(taille_carre)`
`binarisation(taille_carre)`



`image_binarisee`

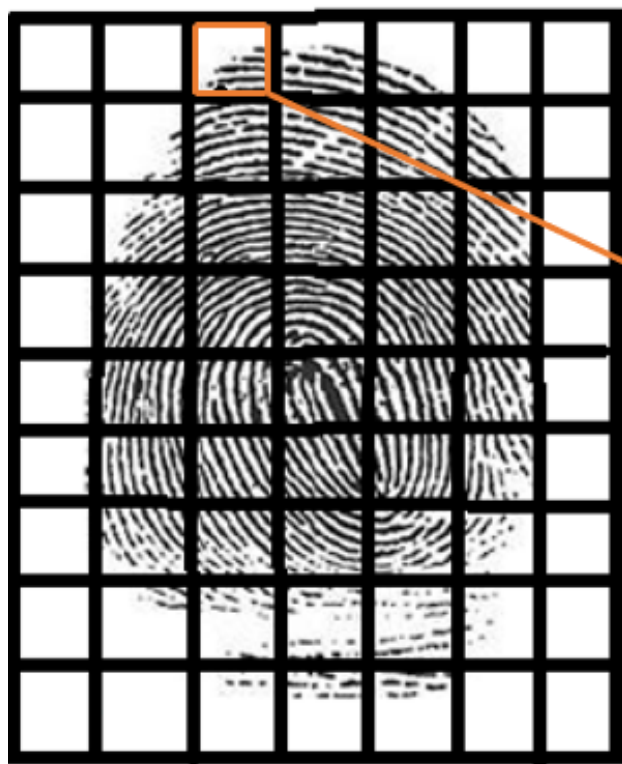




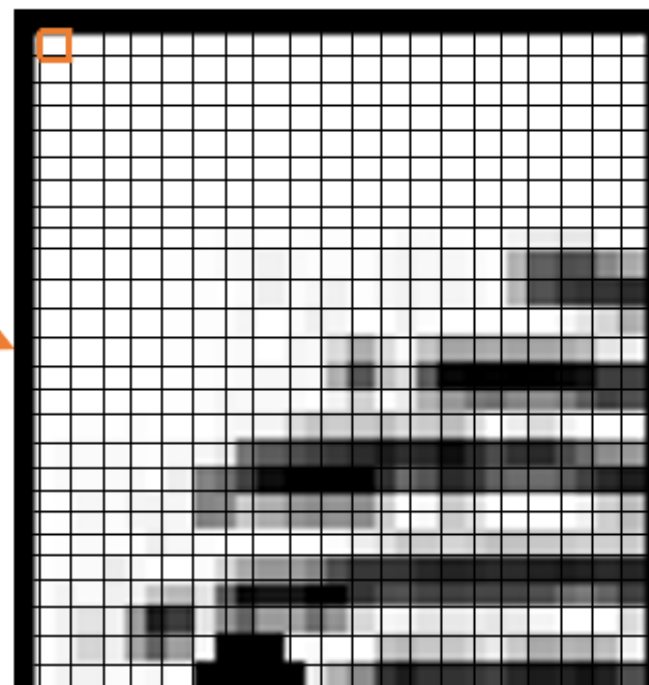
3. présentation de quelques fonctions :

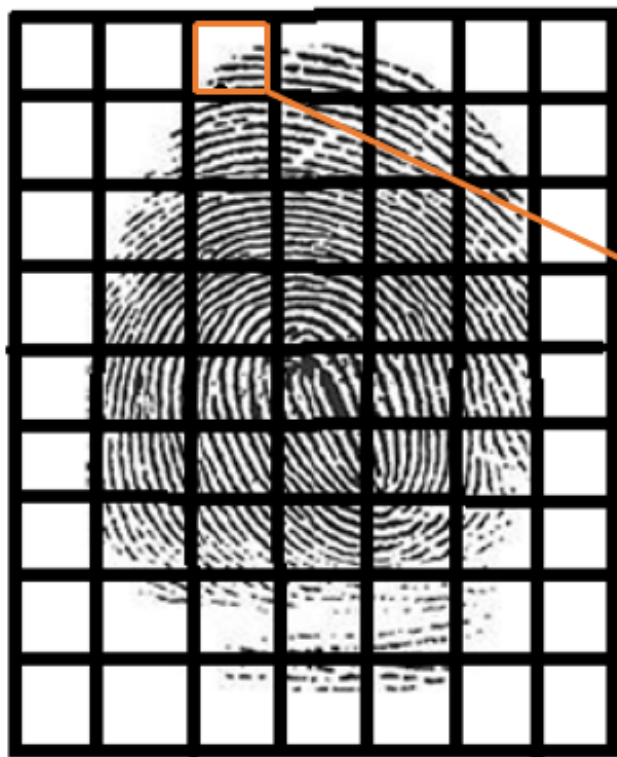
- moyenneCarre

```
def moyenneCarre(taille_carre):  
    (a,b)=image.size  
    nb_carres_colonne=int(a/taille_carre)  
    nb_carres_ligne=int(b/taille_carre)  
    liste_moyennes=[]  
    for p in range (0,nb_carres_colonne):  
        liste_moyennes.append([])  
        for l in range(0,nb_carres_ligne):  
            somme=0  
            for i in range (l*taille_carre,(l+1)*taille_carre):  
                for j in range (p*taille_carre,(p+1)*taille_carre):  
                    somme=image.getpixel((j,i))+somme  
            moyenne=int(somme/(taille_carre**2))  
            liste_moyennes[p].append(moyenne)  
    L=np.array(liste_moyennes)  
    return L
```

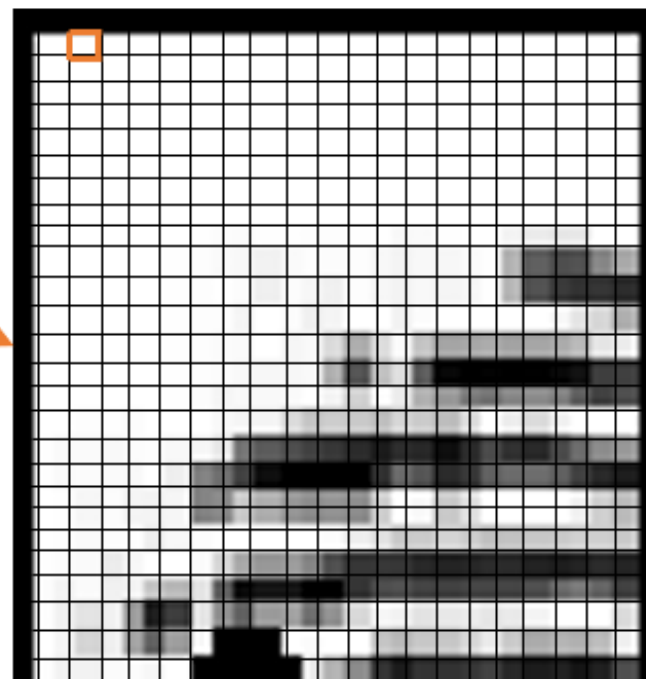


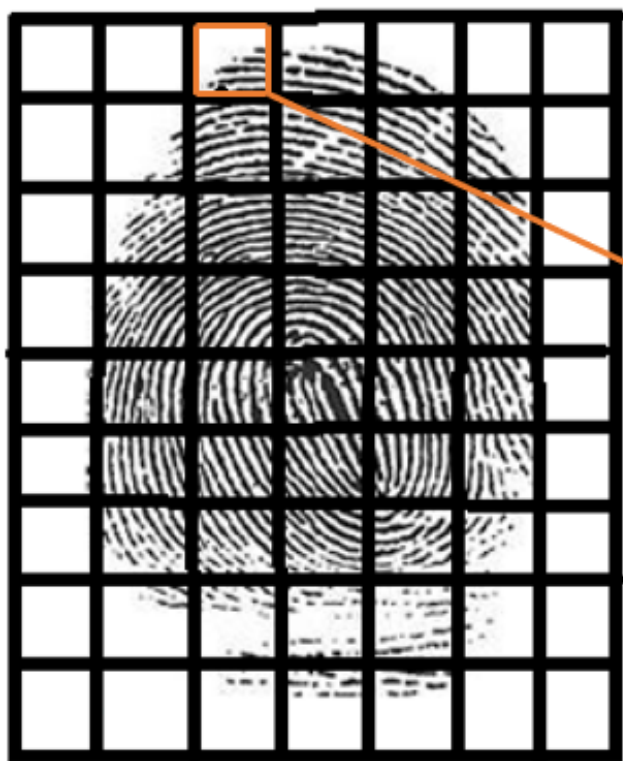
S=255



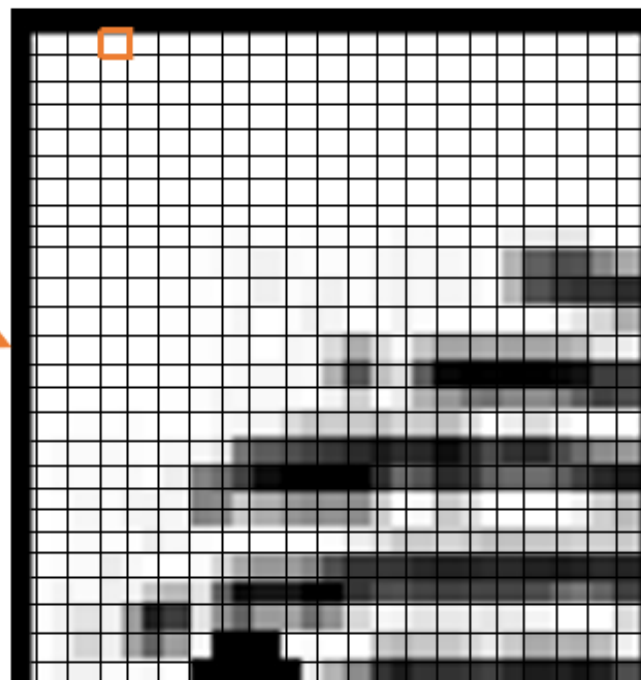


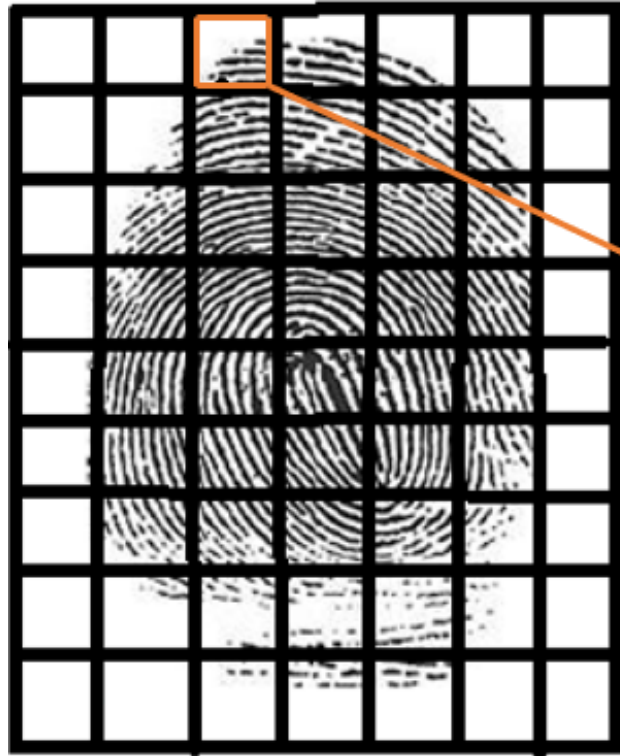
$$S=255+255$$



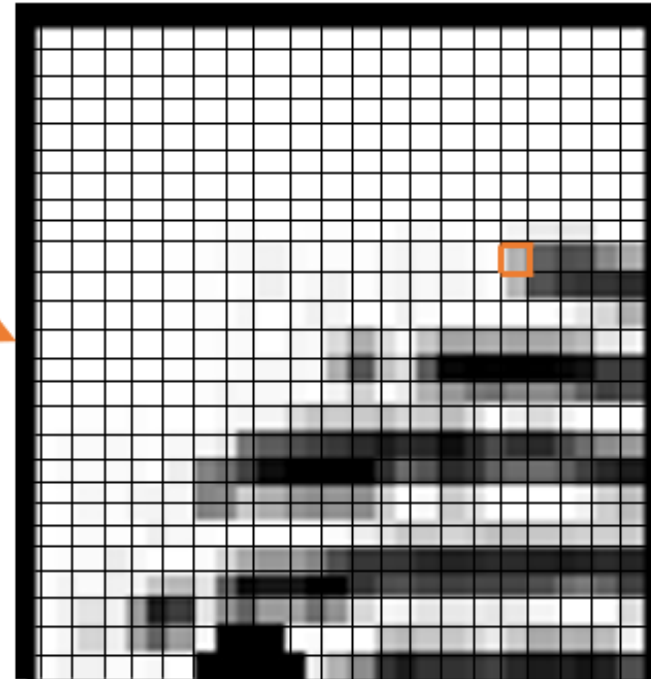


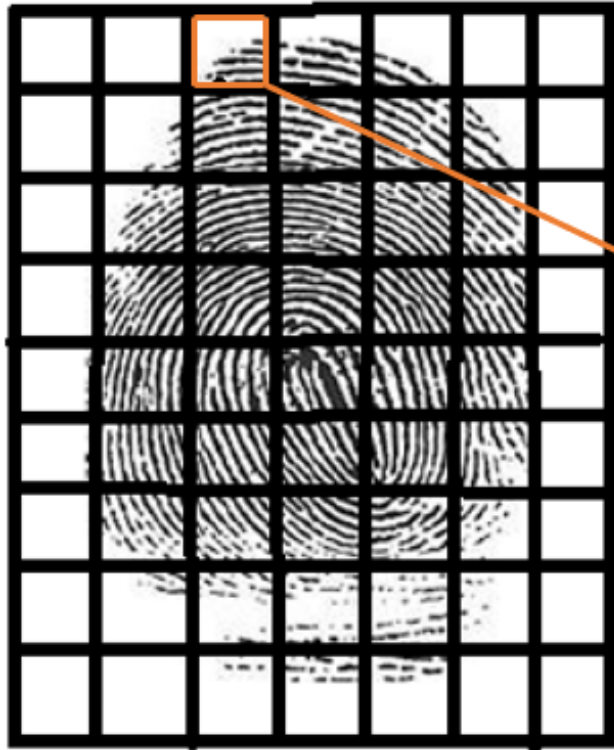
$$S=255+255+255$$



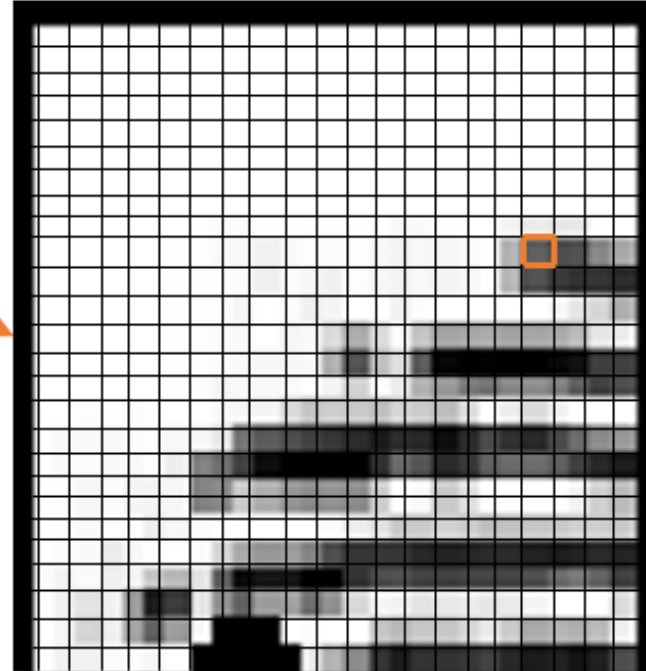


$$S=255+255+255+ \dots + 124$$



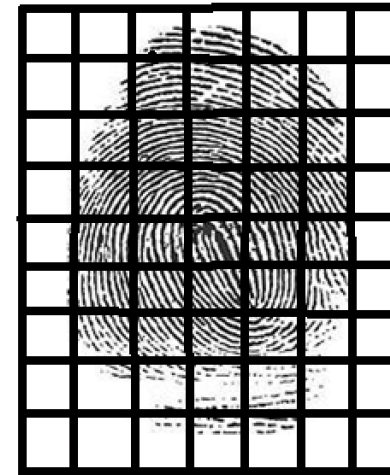


$$S=255+255+255+ \dots + 124+ 103$$



- Moyenne = $S / (\text{nombre de pixel du carré})$
- `Liste_moyenne[p].append(Moyenne)`

255	255	102	85	201	255	255
255	241	132	255	76	65	198
255	124	87	68	76	45	255
255	255	54	63	255	65	255
255	255	67	76	255	39	255
255	255	87	79	255	255	255
255	255	56	65	76	255	255
255	255	209	74	208	205	255
255	255	255	255	255	255	255



- binarisation

```
def binarisation(taille_carre):  
    L=moyenneCarre(taille_carre)  
    (a,b)=image.size  
    image_binarisee=Image.new(image.mode,(a,b))  
    nb_carres_colonne=int(a/taille_carre)  
    nb_carres_ligne=int(b/taille_carre)  
    for k in range(nb_carres_colonne):  
        for g in range(nb_carres_ligne):  
            seuil_binarisation=L[k,g]  
            for i in range (k*taille_carre,(k+1)*taille_carre):  
                for j in range (g*taille_carre,(g+1)*taille_carre):  
                    if image.getpixel((i,j))>=seuil_binarisation:  
                        image_binarisee.putpixel((i,j),255)  
    image_binarisee.show()  
    return image_binarisee
```

- binarisation

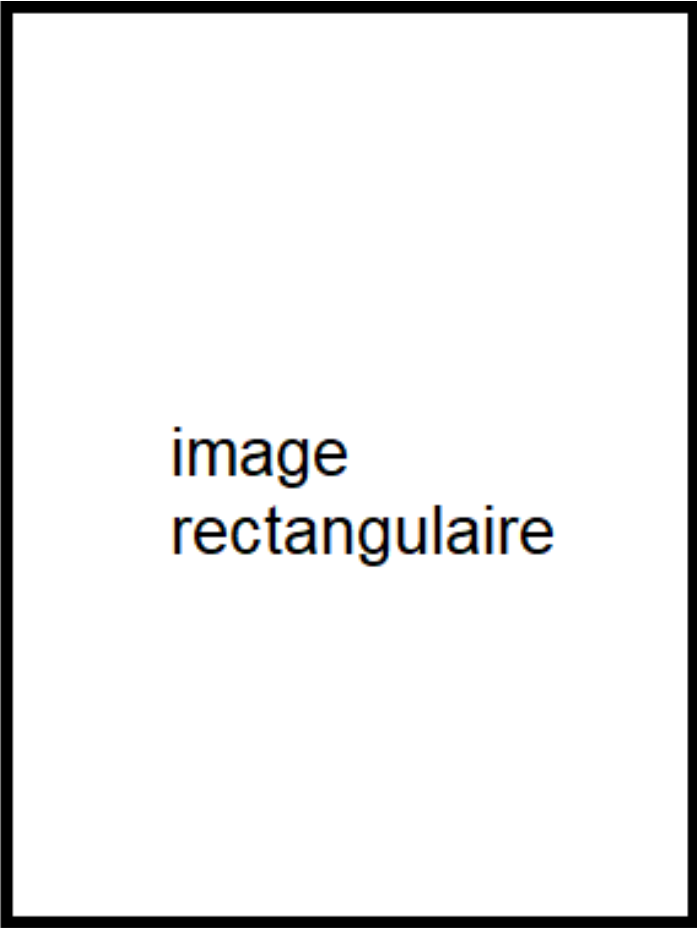
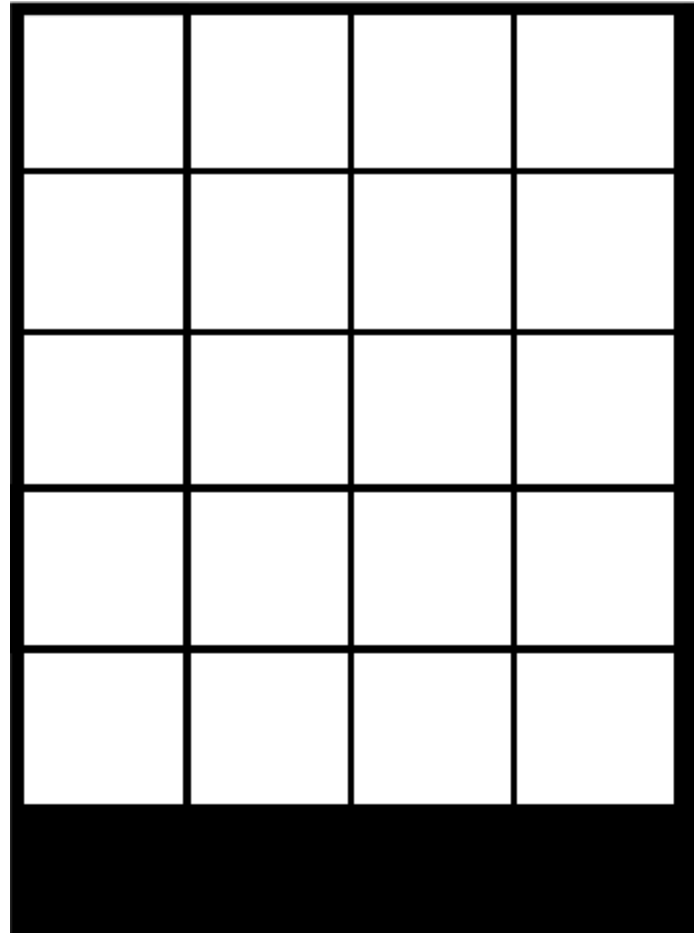


image
rectangulaire

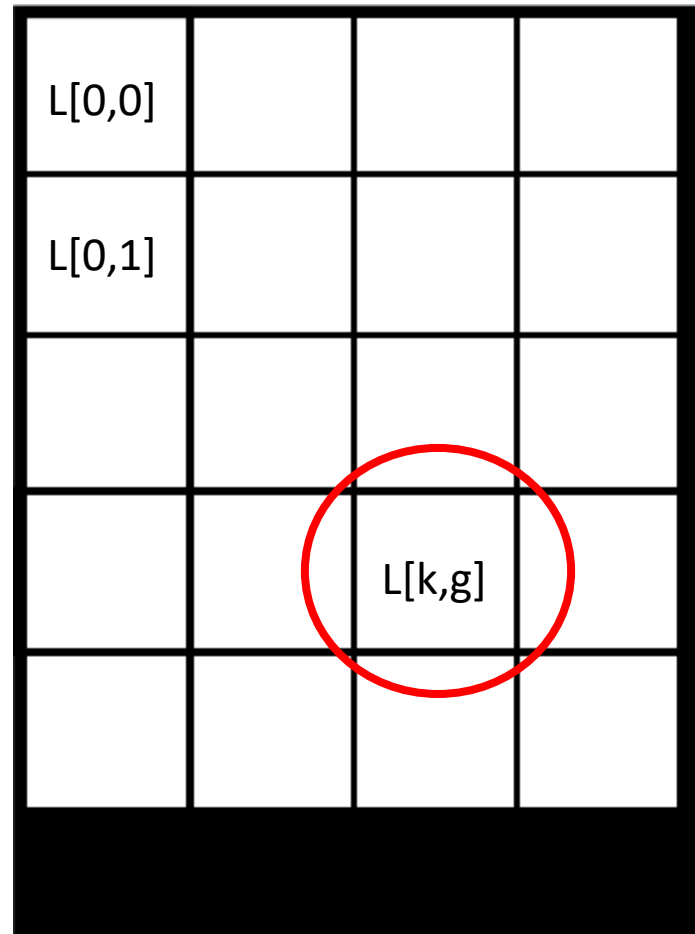
- binarisation

« Quadrillage » de l'image.
A chaque carré est affectée
la moyenne de la valeur des
pixels, contenue dans
Liste_moyenne.



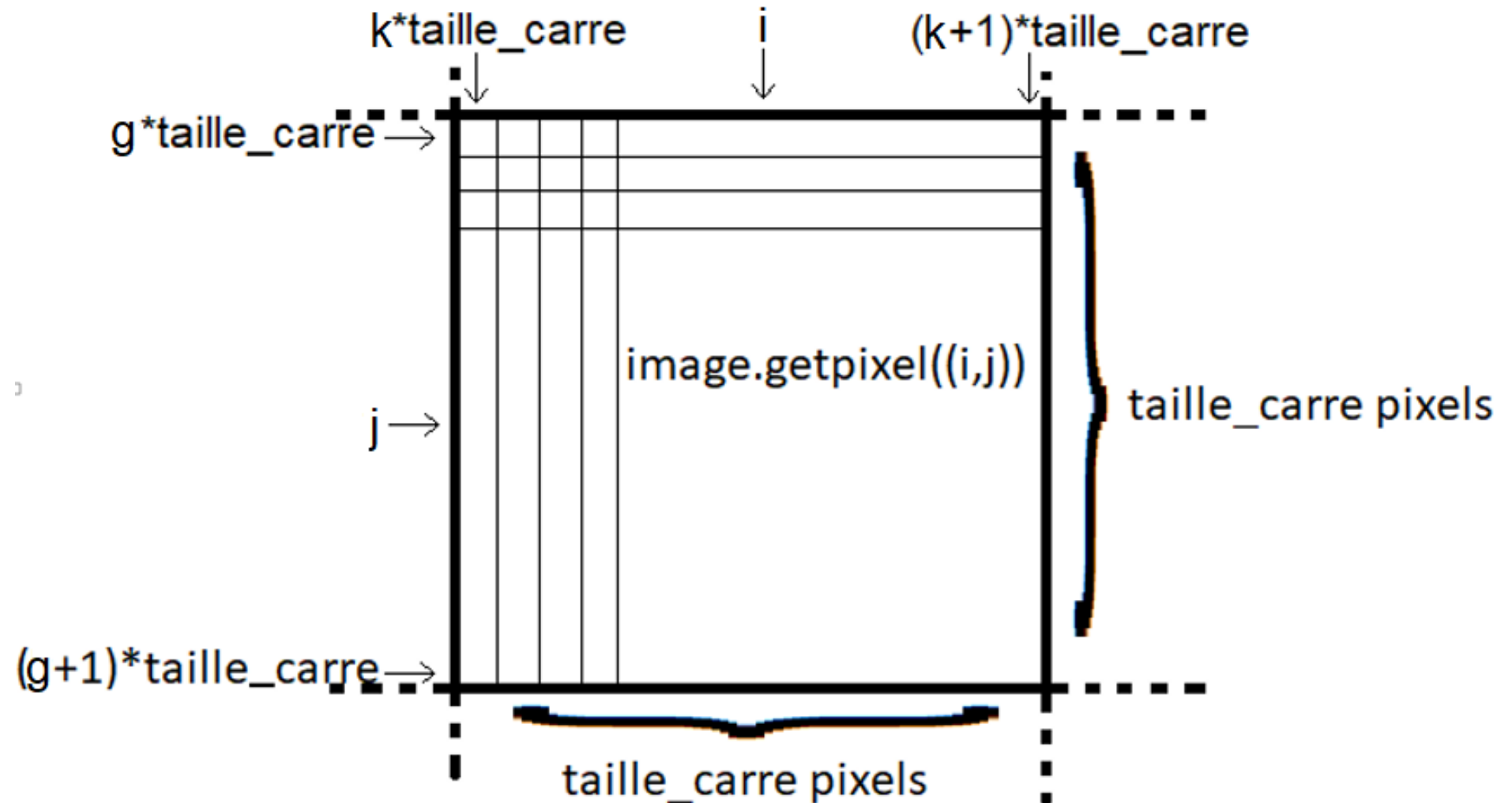
- binarisation

« Quadrillage » de l'image.
A chaque carré est affectée
la moyenne de la valeur des
pixels, contenue dans
Liste_moyenne.



- binarisation

Zoom sur le
carré (k,g):



- SquelettisationN (noir)

```
def squelettisationN(taille_carre):
    '''Entrée: taille_carre = entier strictement positif = dimension d'une sous-unité du cadrillage appliqué à l' image
    Sortie: imagebinarisee_bis = image = image squelettisée une première fois et qui subira de nouvelles squelettisations'''

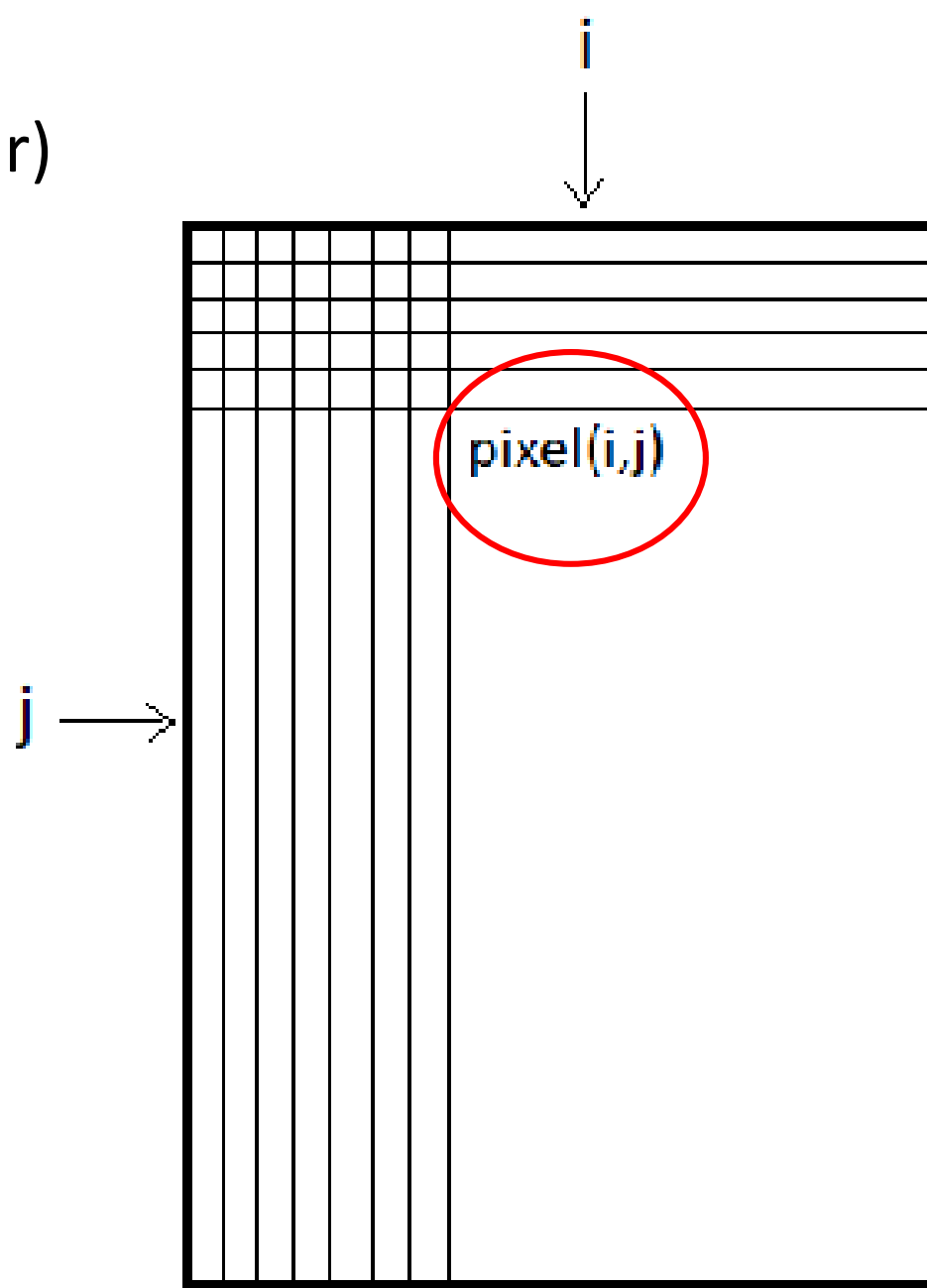
    image_binarisee=binarisation(taille_carre)
    masques_squelettisation=[[[255,255,0],[255,0,0],[255,0,0]],[[0,0,255],[0,0,255],[0,255,255]],[[0,0,0],[255,0,0],[255,255,255]],[[255,255,255],[0,0,255],
    [0,0,0]],[[0,255,255],[0,0,255],[0,0,255]],[[255,0,0],[255,0,0],[255,255,0]],[[0,0,0],[0,0,255],[255,255,255]],[[255,255,255],[255,0,0],[0,0,0]],[[0,0,0],
    [255,0,0],[255,255,0]],[[0,0,0],[0,0,255],[0,255,255]],[[0,255,255],[0,0,255],[0,0,0]],[[255,255,0],[255,0,0],[0,0,0]],[[0,0,0],[0,0,0],[255,255,255]],[
    [255,255,255],[0,0,0],[0,0,0]],[[0,0,255],[0,0,255],[0,0,255]],[[255,0,0],[255,0,0],[255,0,0]]]

    a=binarisation(taille_carre)
    for l in range(0,16):
        for j in range(2,np.shape(image_binarisee)[0]-3):
            for i in range(2,np.shape(image_binarisee)[1]-3):
                if [[a.getpixel((i-1,j-1)),a.getpixel((i-1,j)),a.getpixel((i-1,j+1))],[a.getpixel((i,j-1)),a.getpixel((i,j)),a.getpixel((i,j+1))],
                [a.getpixel((i+1,j-1)),a.getpixel((i+1,j)),a.getpixel((i+1,j+1))]]==masques_squelettisation[l]: # si un carré (3*3) de l'image correspond à un masque
                    a.putpixel((i,j),255)

    a.show()
    image_binarisee_bis=a
    return image_binarisee_bis
```

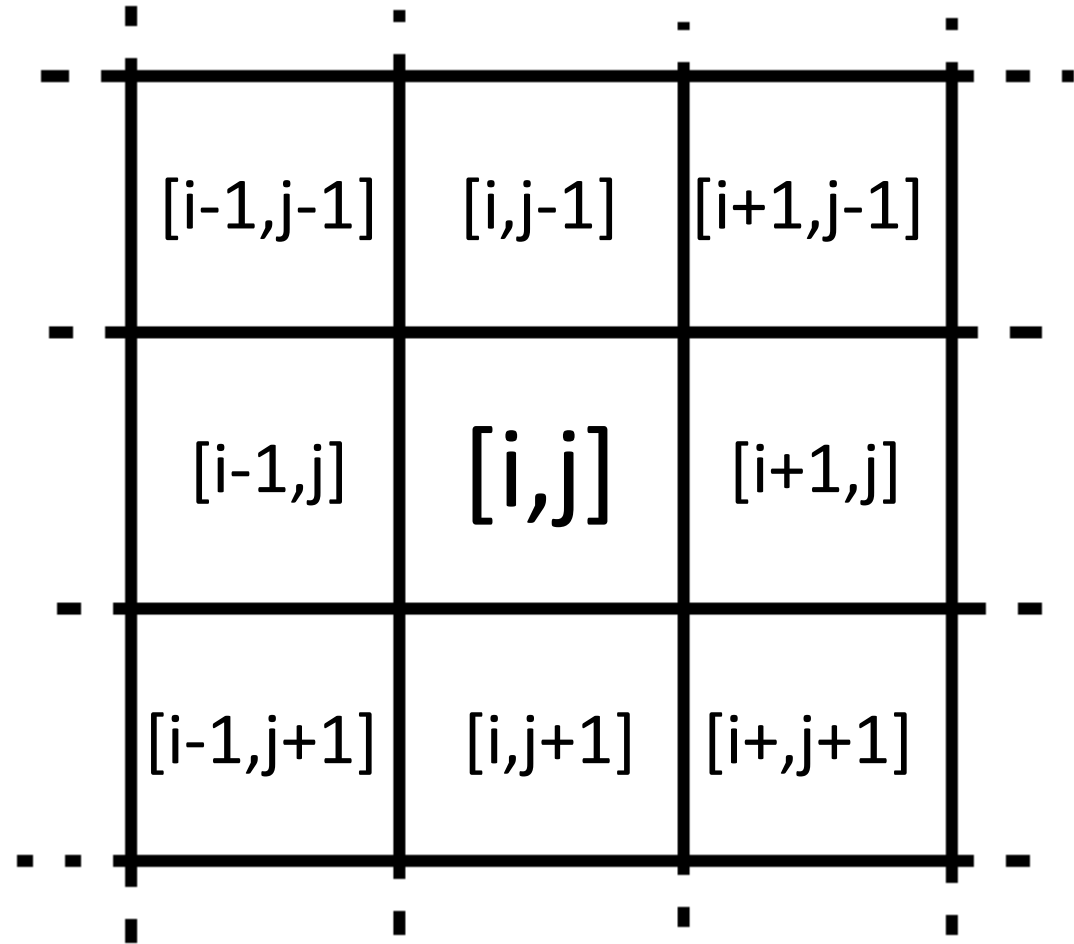
- SquelettisationN (noir)

Image rectangulaire
dont on récupère la
valeur de chacun des
pixels.



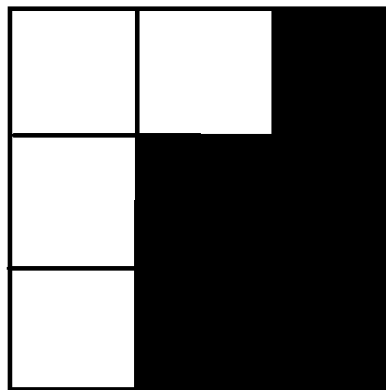
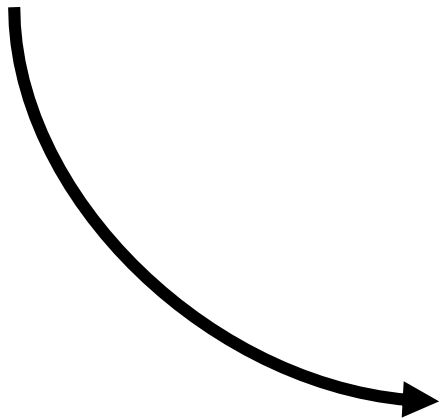
- SquelettisationN (noir)

Zoom sur le carré de taille
3*3 pixels, de centre (i,j):

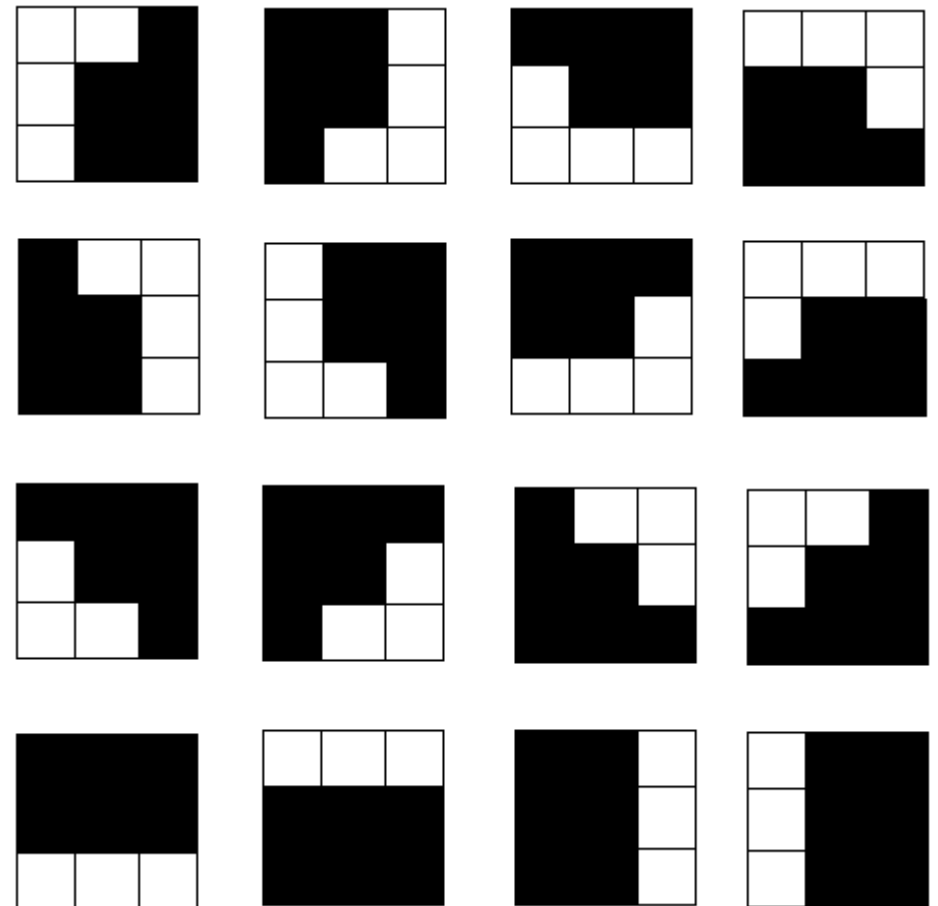


SquelettisationN (noir)

255	255	0
255	0	0
255	0	0



Masques de squelettisation:



4 . objectifs non atteints : ce que nous aurions fait :

- Transformation de la 2ème image au même format : orientation et taille (pour que les coordonnées des minuties soient comparables)
- application du même programme à la deuxième image
- comparaison des positions des minuties des deux images

Améliorations de notre programme :

- prolongement par continuité des fragments de crêtes effacés

