Όραση Υπολογιστών

Εργασία 1

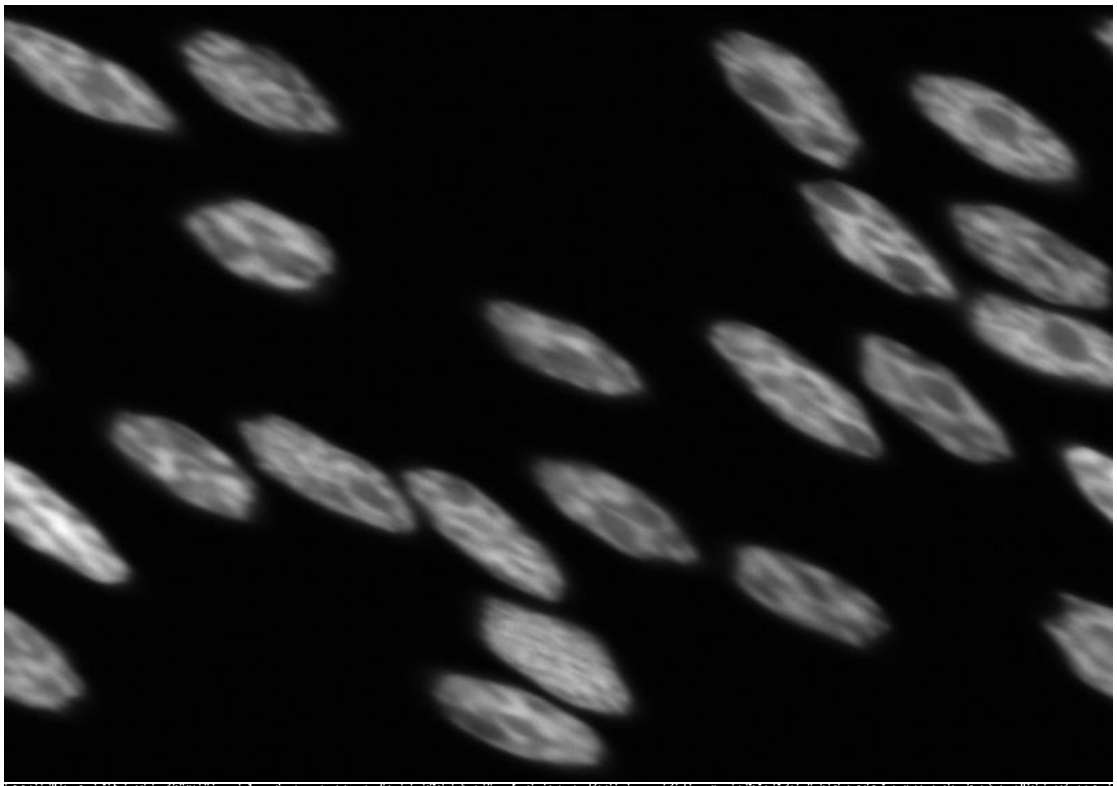
Ακαδημαϊκό Έτος: 2019-2020

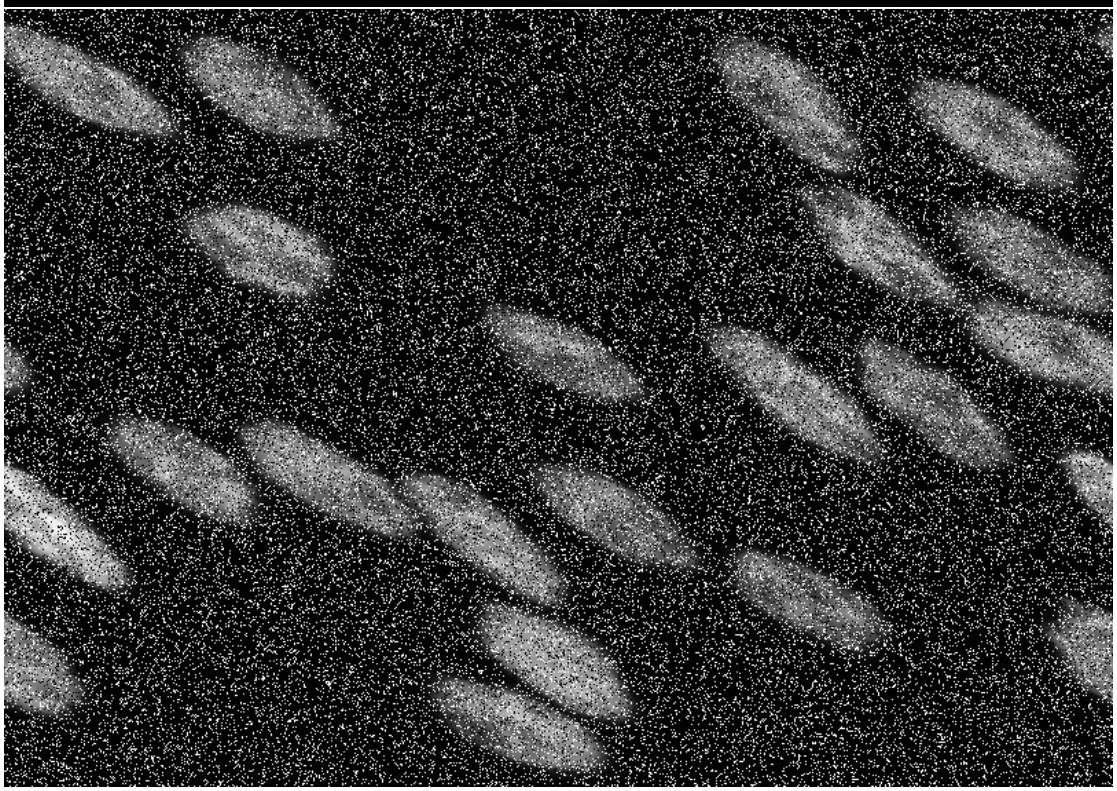
Καρασακαλίδης Αλέξανδρος Ηλίας

ΑΜ: 57448

- Ζητούμενα

Στην εκφώνηση της εργασίας μας ζητείται να μελετήσουμε τις ακόλουθες εικόνες:





Συγκεκριμένα, οι απαιτήσεις της εργασίας είναι οι εξής:

1. Μέτρηση αριθμού αντικειμένων (κυττάρων) στο χώρο εικόνας.
2. Μέτρηση επιφάνειας (ως αριθμός εικονοστοιχείων) κάθε αντικειμένου.
3. Μέτρηση της μέσης τιμής διαβάθμισης του γκρι των εικονοστοιχείων που περιέχονται στα περιβάλλοντα κουτιά (Bounding Boxes) των αντικειμένων με τέτοιο τρόπο ώστε η ταχύτητα εκτέλεσης υπολογισμού να είναι ανεξάρτητη του μεγέθους του αντικειμένου.

Επίσης, μας δόθηκαν οι εξής υποδείξεις:

1. Στην αρχική εικόνα να εφαρμόσετε γραμμικό ή μη γραμμικό φίλτρο απόρριψης θορύβου της επιλογής σας. Το βήμα αυτό θα πρέπει να υλοποιηθεί ΧΩΡΙΣ τη χρήση των αντίστοιχων συναρτήσεων της OpenCV.
2. Στην εικόνα του αποτελέσματος του παραπάνω βήματος να εφαρμόσετε κατάλληλο κατώφλι για τη μετατροπή της εικόνας διαβάθμισης του γκρι σε δυαδική εικόνα.
3. Για να γίνουν οι ζητούμενες μετρήσεις θα πρέπει να εφαρμόσετε κατάλληλη μεθοδολογία που να απομονώνει τα (υπό μελέτη) αντικείμενα.
4. Τα αντικείμενα στην εικόνα για τα οποία θα υπολογισθούν οι ζητούμενες μετρήσεις είναι μόνο αυτά που περιλαμβάνουν εικονοστοιχεία τα οποία δεν βρίσκονται στο περίγραμμα της εικόνας.

- # Ανάλυση του Κώδικα

Μας δίνονται δύο εικόνες. Η N7 και η NF7 απεικονίζουν το ίδιο σύνολο κυττάρων (τα οποία και είναι τα υπό-μελέτη αντικείμενα) με μοναδική διαφορά ότι η N7 περιέχει θόρυβο της μορφής “Salt and Pepper”.

Ξεκινάμε με την φόρτωση των βιβλιοθηκών που χρησιμοποιούμε και των εικόνων σε πίνακες (υπό μορφή Grayscale – δηλαδή ασπρόμαυρη) και την εμφάνισή τους στην οθόνη μας:

```
import cv2
import numpy as np

fN7 = "N7.png"
fNF7 = "NF7.png"

N7 = cv2.imread(fN7, cv2.IMREAD_GRAYSCALE)
NF7 = cv2.imread(fNF7, cv2.IMREAD_GRAYSCALE)

cv2.namedWindow('NF7')
cv2.imshow('NF7', NF7)

cv2.namedWindow('N7')
cv2.imshow('N7', N7)

cv2.waitKey(0)
```

Συγκεκριμένα:

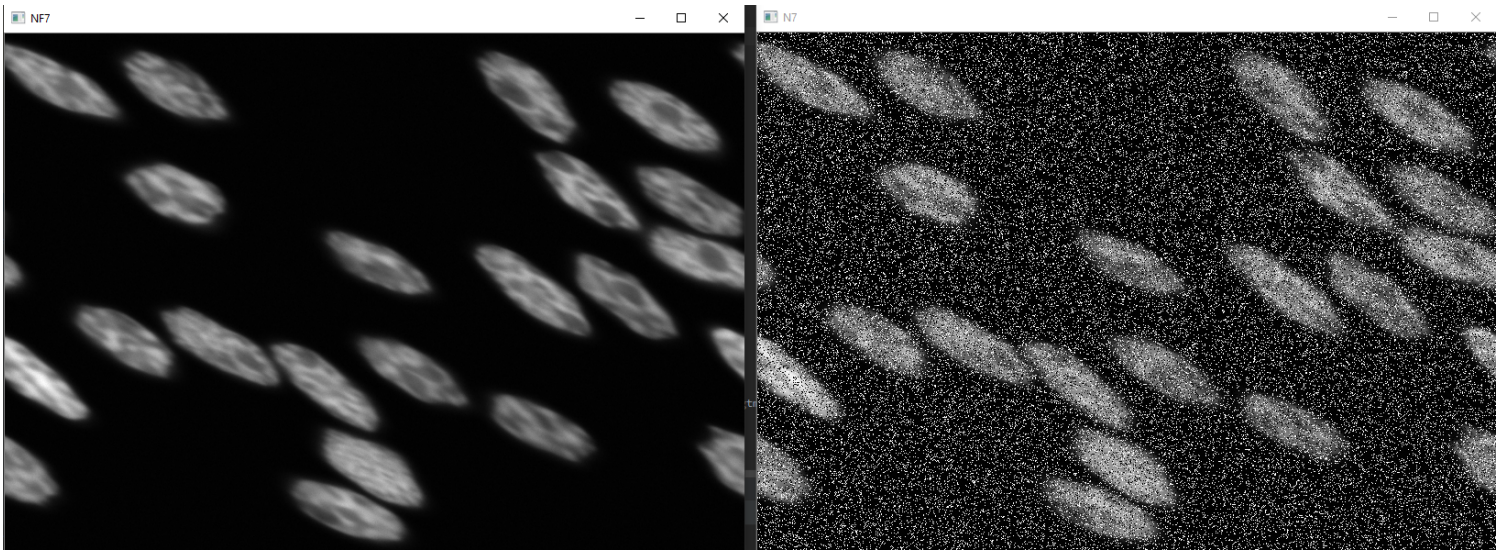
- Η βιβλιοθήκη numpy μας παρέχει ένα σύνολο εργαλείων για την μεταχείριση διανυσμάτων/πινάκων.
- Η βιβλιοθήκη cv2 (OpenCV) μας παρέχει ένα σύνολο εργαλείων για την μεταχείριση των εικόνων (υπό μορφή πινάκων).

- Η εντολή cv2.imread αποθηκεύει σε μια μεταβλητή τον πίνακα που περιγράφει την αντίστοιχη εικόνα υπό μορφή Grayscale (cv2.IMREAD_GRAYSCALE).

- Η εντολές cv2.namedWindow και cv2.imshow δημιουργούν ένα παράθυρο και μεταφράζουν/απεικονίζουν έναν πίνακα σε εικόνα αντίστοιχα.

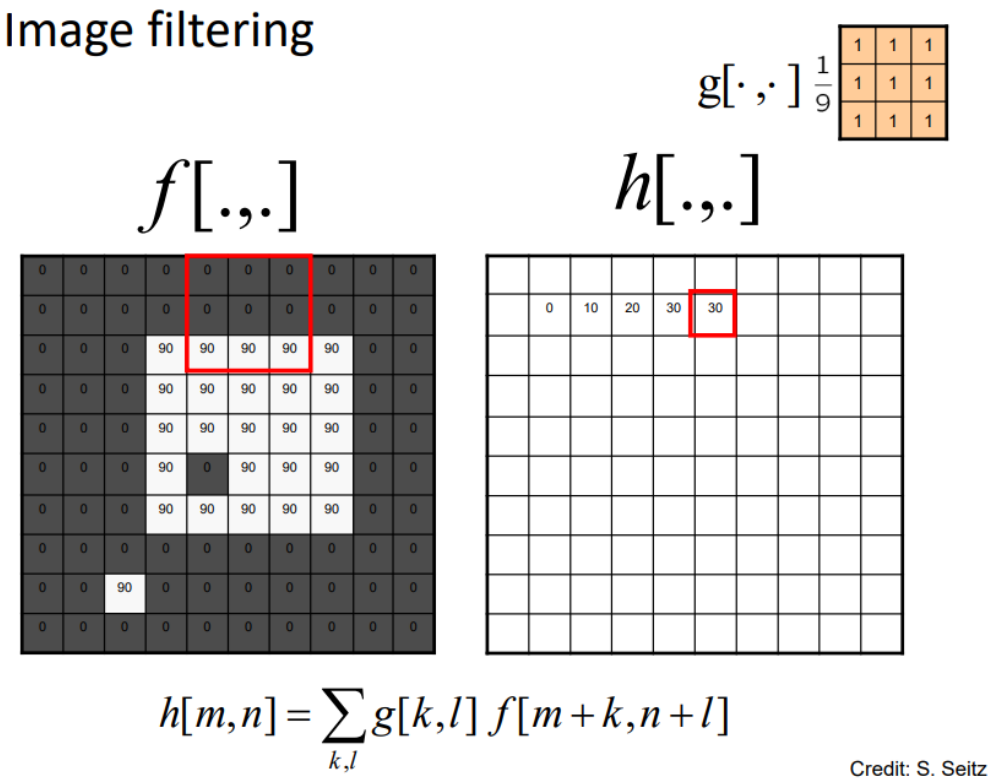
- Η εντολή cv2.waitKey μας επιτρέπει να ‘παγώσουμε’ την εκτέλεση του κώδικά μας, μέχρι να πατήσουμε κάποιο πλήκτρο

Σαν αποτέλεσμα έχουμε την των παραπάνω εικόνων:



Αφού λοιπόν έχουμε οικειοποιηθεί με τις εικόνες που μελετάμε, ήρθε η ώρα να προχωρήσουμε ‘αποθορυβοποιήσουμε’ την N7.

Είναι γνωστό ότι ο θόρυβος Salt and Pepper αντιμετωπίζεται πολύ εύκολα με εφαρμογή median filter. Πως εφαρμόζεται όμως αυτό στην εικόνα μας (συγκεκριμένα στον πίνακά μας);



Όπως έχουμε δει στην θεωρία, ένα φίλτρο εφαρμόζεται μελετώντας τα γειτονικά εικονοστοιχεία του κάθε εικονοστοιχείου. Συγκεκριμένα, το πλήθος τους εξαρτάται από το μέγεθος του kernel.

Ο kernel είναι ένας τετραγωνικός πίνακας ο οποίος – με κέντρο το εικονοστοιχείο στο οποίο εφαρμόζεται – λαμβάνει τις τιμές των εικονοστοιχείων πολλαπλασιασμένες 1-1 με τις τιμές των αντίστοιχων εικονοστοιχείων του πίνακα της εικόνας. Στην περίπτωση της παραπάνω εικόνας από την παρουσίαση που έγινε στην παράδοση της θεωρίας του αντίστοιχου κεφαλαίου, βλέπουμε πάνω δεξιά ένα kernel με μέγεθος 3.

Έπειτα, εφαρμόζεται κάποια μαθηματική συνάρτηση σε αυτά και επιστρέφεται η τιμή του νέου, ‘φιλτραρισμένου’ εικονοστοιχείου σε έναν νέο πίνακα. Αφού επαναληφθεί η παραπάνω διαδικασία για όλα τα εικονοστοιχεία, έχουμε έναν νέο πίνακα ο οποίος είναι ουσιαστικά ο φιλτραρισμένος μας πίνακας.

Στην περίπτωση του median filter, η μαθηματική συνάρτηση που ακολουθείται στα στοιχεία του kernel παρά η διάμεσός (median) τους.

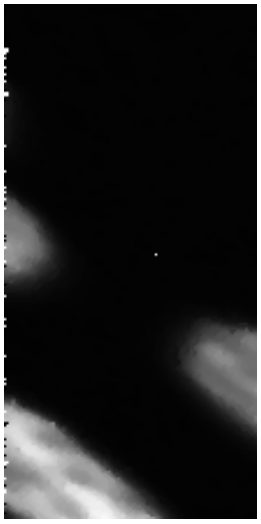
```
img = cv2.copyMakeBorder(N7,1,1,1,1,cv2.BORDER_REFLECT)

for k in range(0,3):
    imgtmp = img
    print("Please wait. Filter being applied: ", k + 1, "/", 4)
    for i in range(1,imgtmp.shape[1]-1):
        for j in range(1,imgtmp.shape[0]-1):
            kernel = np.array([
            [imgtmp[j-1][i-1],imgtmp[j-1][i],imgtmp[j-1][i+1]],
            [imgtmp[j][i-1],imgtmp[j][i],imgtmp[j][i+1]],
            [imgtmp[j+1][i-1],imgtmp[j+1][i],imgtmp[j+1][i+1]]])
            img[j][i] = np.median(kernel).astype(np.int8)

print("Done!")

img = np.delete(img,img.shape[1]-1,1)
img = np.delete(img,img.shape[0]-1,0)
img = np.delete(img,0,0)
img = np.delete(img,0,1)

cv2.namedWindow('NF7 Median Blur')
cv2.imshow('NF7 Median Blur', img)

cv2.waitKey(0)
```

Στην εφαρμογή όμως συναντάμε πρόβλημα στην εφαρμογή του φίλτρου στα ακραία εικονοστοιχεία της εικόνας μας αφού, σύμφωνα με το kernel, θα πρέπει να αξιοποιήσουμε για αυτά εικονοστοιχεία που είναι εκτός του πίνακα!

Με την cv2.copyMakeBorder λοιπόν δημιουργούμε στην N7 μια επιπλέον ‘σειρά εικονοστοιχείων’ σε κάθε πλευρά του πίνακα (1η και τελευταία στήλη – 1η και τελευταία σειρά) – όπως ορίζουμε με τους τέσσερεις άσσους. Το τελευταίο όρισμα της συνάρτησης είναι η μέθοδος με την οποία θα συμπληρωθούν οι τιμές των εικονοστοιχείων. Μετά από δοκιμές των διάφορων μεθόδων, δεδομένου το φίλτρο που θα χρησιμοποιήσουμε, επιλέχθηκε η cv2.BORDER_REFLECT, με την οποία ‘αντικατροπτίζουμε’ (αντιγράφουμε στην περίπτωσή μας) τα αντίστοιχα εικονοστοχεία της σειράς/στήλης του πίνακα.

Για παράδειγμα, η cv2.BORDER_CONSTANT που συμπληρώνει με μια σταθερή τιμή τις σειρές/στήλες, δημιουργεί πολλές αλλοιώσεις στα άκρα της εικόνας μας μετά την εφαρμογή του φίλτρου, όπως βλέπουμε στην διπλανή εικόνα.

Η επαναληπτική διαδικασία που ακολουθεί δεν είναι τίποτε άλλο, παρά η διεργασία της εφαρμογής του median filter που περιγράψαμε παραπάνω. Συγκεκριμένα, χρησιμοποιήσαμε kernel μεγέθους 3 και επαναλάβαμε το φίλτρο και 2η φορά.

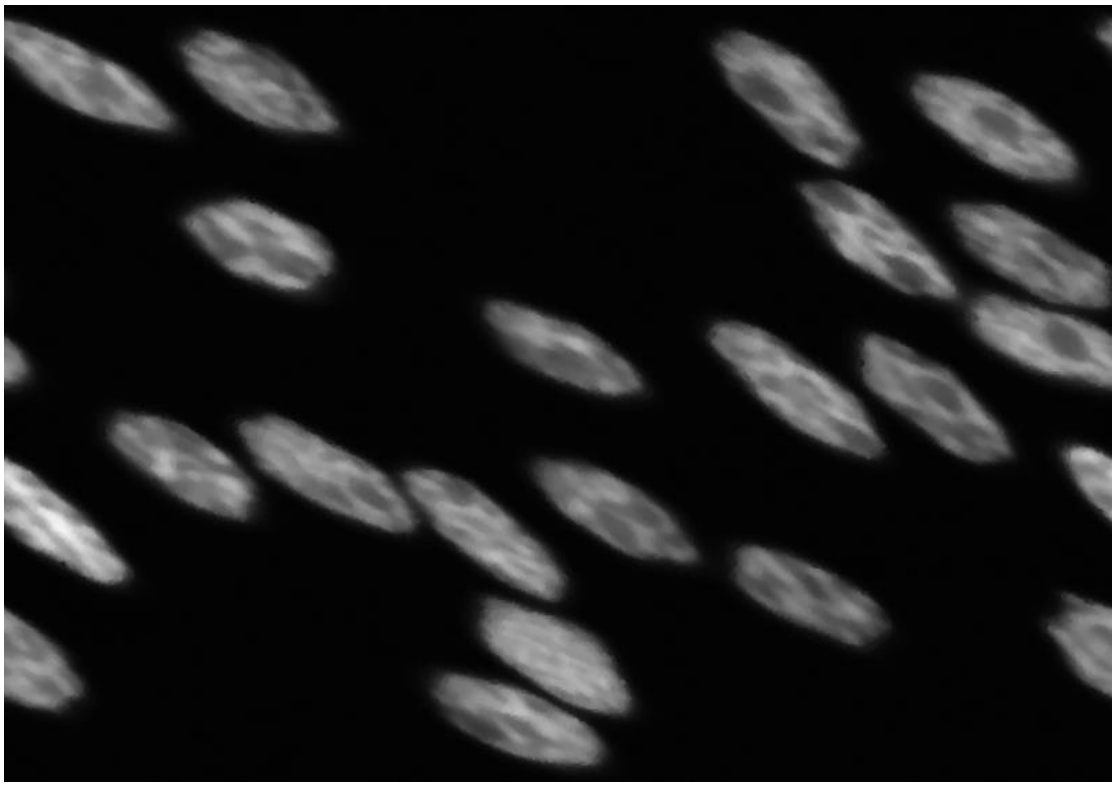
1η εφαρμογή: Ελάχιστος Θόρυβος* 	**2η εφαρμογή:** Καθόλου Θόρυβος* 

*Εξαιρώντας τα εικονοστοιχεία της πρώτης στήλης.

Ο θόρυβος που υπάρχει στις άκρες της εικόνας μας είναι αδιάφορος γιατί θα εξαλειφθούν. Μετά την εφαρμογή του φίλτρου δηλαδή, με την np.delete θα διαγράψουμε την πρώτη και την τελευταία στήλη/γραμμή που είχαμε προσθέσει πριν για την εφαρμογή του φίλτρου.

Αξίζει να σημειωθεί ότι ένα μεγαλύτερο kernel που δοκιμάστηκε (μεγέθους 5) προκαλούσε μεγαλύτερη αλλοίωση στα κύτταρα.

Εμφανίζουμε λοιπόν το αποτέλεσμα του φίλτρου στο παράθυρο με τίτλο ‘NF7 Median Blur’:



Ακολουθεί ο παρακάτω κώδικας, ο οποίος θα εξηγηθεί αργότερα διότι αξιοποιείται για την επίλυση του 3ου ζητήματος.

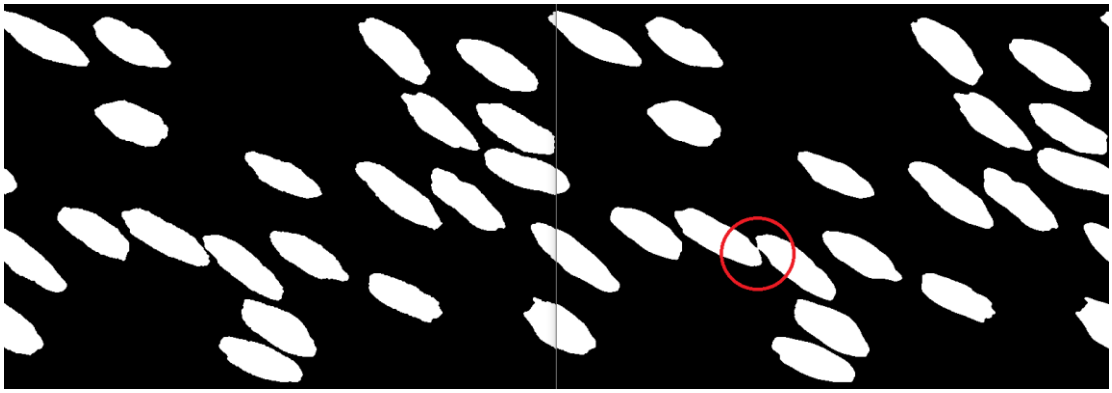
```
sN7 = img.astype('int64')

for i in range(1,sN7.shape[1]):
    sN7[0][i] = sN7[0][i] + sN7[0][i-1]
for i in range(1,sN7.shape[0]):
    sN7[i][0] = sN7[i][0] + sN7[i-1][0]
for j in range(1,sN7.shape[0]):
    for i in range(1,sN7.shape[1]):
        sN7[j][i] = sN7[j][i] + sN7[j][i - 1] + sN7[j-1][i] - sN7[j-1][i-1]
```

Ακολουθεί η επεξεργασία της ‘καθαρής’ εικόνας και η μετατροπή των εικόνων σε δυαδική μορφή.

```
img2 = cv2.medianBlur(NF7,3)

strel = np.zeros((7,7), np.uint8)

for i in range(0,6):
    strel[i,i]=1

img2 = cv2.morphologyEx(img2, cv2.MORPH_OPEN, strel)

ret,bnr = cv2.threshold(img,53,255,0)
CLRret,CLRbnr = cv2.threshold(img2,52,255,0)

cv2.namedWindow('Binary (N7)')
cv2.imshow('Binary (N7)', bnr)
cv2.namedWindow('Binary (NF7)')
cv2.imshow('Binary (NF7)', CLRbnr)
```

Ξεκινάμε εφαρμόζοντας επίσης median blur στην 2η εικόνα (NF7) για την εξάλειψη οποιουδήποτε μικρού θορύβου που μπορεί να είχε (με την χρήση της έτοιμης συνάρτησης: cv2.medianBlur και kernel μεγέθους 3).

Παρακάτω, με την χρήση της cv2.threshold που είδαμε στο εργαστήριο, μετατρέπουμε τις φιλτραρισμένες εικόνες μας σε δυαδικής μορφής εικόνες επιλέγοντας threshold 53 και 52 για την N7 και την NF7 αντίστοιχα (bnr, CLRbnr). Μικρότερο threshold μεταφέρει ανεπιθύμητο θόρυβο στις εικόνες μας. Μεγαλύτερο threshold εξαλείφει κομμάτια των κυττάρων. Το αποτέλεσμα φαίνεται στις παρακάτω εικόνες.

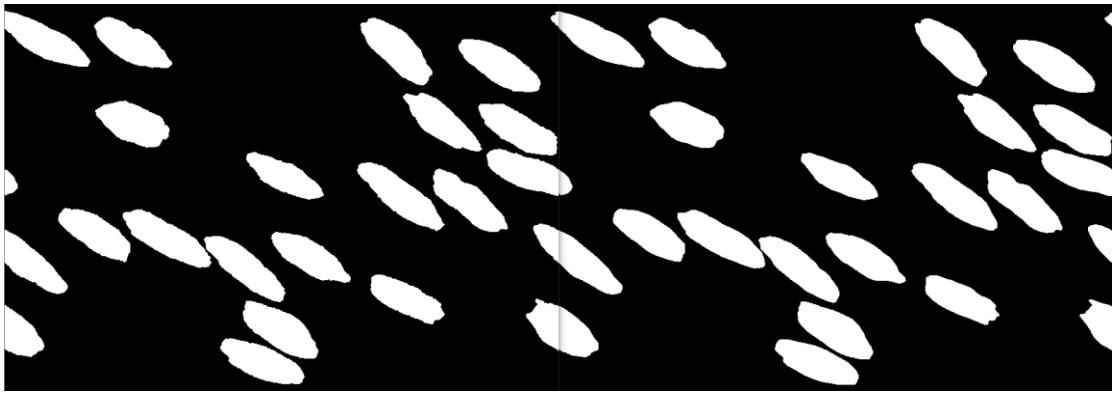


bnr CLRbnr

Παρατηρούμε ότι στην δυαδική εικόνα της NF7 δύο κύτταρα ενώνονται. Για την λύση αυτού του προβλήματος έχουμε εφαρμόσει ένα opening φίλτρο με ένα συγκεκριμένο kernel που «στοχεύει» στην γεωμετρία της ένωσης.

1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

Έτσι, «αποκόβουμε» τα δύο κύτταρα, όπως φαίνεται και στην παρακάτω εικόνα.

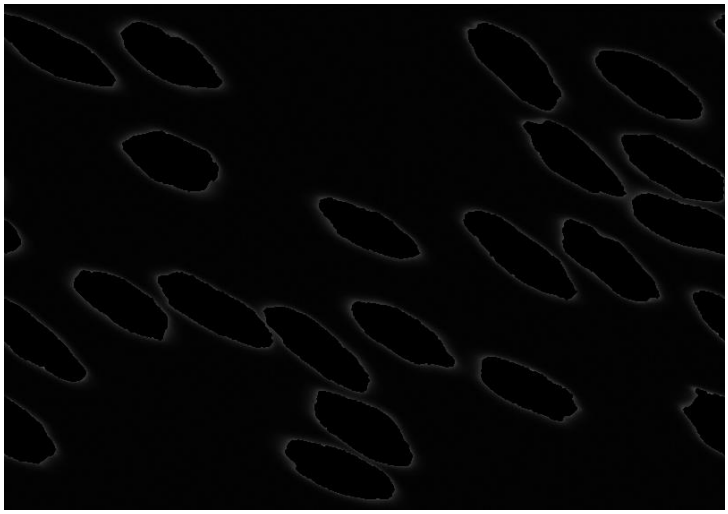


Αξίζει επίσης να σημειωθεί, ότι στην πρώτη εικόνα τα δύο κύτταρα είχαν «αποκοπεί» ως αποτέλεσμα του threshold και δεν χρειάστηκε να εφαρμόσουμε κάποιο φίλτρο.

Το παρακάτω κομμάτι του κώδικα βρίσκεται σε μορφή σχολίων.

```
bnrc = np.where(bnr>0,0,255)
cmp = NF7*bnrc

cv2.namedWindow('NF7 BINARY COMPARE')
cv2.imshow('NF7 BINARY COMPARE', cmp)
```

Με αυτόν τον κώδικα αντιστρέφεται ο δυαδικός πίνακας. Αν πολλαπλασιαστεί 1 προς 1 με την NF7 θα μας δώσει τα κομμάτια της εικόνας που δεν είναι άσπρα. Έτσι, μπορούμε οπτικά να αξιολογήσουμε το κατά πόσο αποτελεσματικά έχουμε απομονώσει τα κύτταρα!



Έχοντας τα απομονωμένα τα κύτταρα στην δυαδική εικόνα, βρίσκουμε τα περιγράμματά τους.

```
_,cntrs,_ = cv2.findContours(bnr,cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE,)
_,CLRcntrs,_ = cv2.findContours(CLRbnr,cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
```

Η cv2.findContours μας επιστρέφει ένα διάνυσμα το οποίο περιέχει σε διανύσματα τις συντεταγμένες των εικονοστοιχείων που αποτελούν το διάγραμμα των κυττάρων! Το όρισμα cv2.RETR_LIST ορίζει ότι οι συντεταγμένες θα μας δίνονται χωρίς να έχουν ταξινομηθεί με κάποιον τρόπο και το cv2.CHAIN_APPROX_NONE ορίζει την μορφή με την οποία θα μας δώσει τις συντεταγμένες, οι οποίες στην προκειμένη περίπτωση είναι οι συντεταγμένες κάθε εικονοστοιχείου του περιγράμματος όπως έχουν.

Πέρα από τις συντεταγμένες, η συνάρτηση επιστρέφει και την ιεραρχία που έχουν αποθηκευτεί οι συντεταγμένες στο διάνυσμα – κάτι το οποίο παραλείπουμε τελείως και δεν αποθηκεύουμε.

Αφού έχουμε βρει τα περιγράμματα των κυττάρων και στις δύο εικόνες, θα απορρίψουμε σύμφωνα με την υπόδειξη, κάθε κύτταρο του οποίου εικονοστοιχεία βρίσκονται στο περίγραμμα της εικόνας.

```
ttlclls = 0
CLRttlclls = 0

for i in range(len(cntrs)):
    m=False
    for j in range(len(cntrs[i])):
            if (cntrs[i][j][0][0]==0 or cntrs[i][j][0][1]==0 or
cntrs[i][j][0][0]==bnr.shape[1]-1 or cntrs[i][j][0][1]==bnr.shape[0]-1):
                m=True
                break
    if m:
        akr[i]=0
    else:
        if len(cntrs[i])>20:
            ttlclls=ttlclls+1

for i in range(0,akr.size-1):
    k=akr.size-1-i
    if akr[k]==0:
        del cntrs[k]
```

Αυτό θα κάνουμε μέσα στην διπλά επαναληπτική διαδικασία. Αρχίζοντας για κάθε περίγραμμα (for I in range (len(cntrs)) – όπου το len(cntrs) μας δίνει το μέγεθος του διανύσματος που περιέχει τα διανύσματα με τις συντεταγμένες του κάθε περιγράμματος, άρα και τον αριθμό τον περιγραμμάτων) και για κάθε συντεταγμένη του περιγράμματος, ελέγχουμε αν ανήκει στην πρώτη στήλη/γραμμή

ή στην τελευταία. Αν μια συντεταγμένη για την οποία ισχύει το παραπάνω, σημειώνουμε τον δείκτη του διανύσματος του περιγράμματος στο οποίο ανήκει.

Αφού λοιπόν βρούμε τα περιγράμματα που ‘ακουμπάνε’ στα άκρα της εικόνας, με την δεύτερη επαναληπτική διαδικασία, στην οποία με την “del” τα διαγράφουμε από το διάνυσμά μας.

Να αναφέρουμε επίσης ότι ένας ακόμη παράγοντας που θα αποτελούσε κριτήριο για να απορρίψουμε ένα περίγραμμα (όπως φαίνεται στο 2ο if της διπλής for) θα ήταν αν το πλήθος συντεταγμένων ήταν μικρότερο ή ίσο των 20 εικονοστοιχείων. Δηλαδή, εάν υπήρχε κάποιο ελάχιστα μικρό αντικείμενο που δεν απαλείφθηκε από το φιλτράρισμα των εικόνων, θα το απορρίπταμε. Σε κάποιες από τις δοκιμές στο φιλτράρισμα που εφαρμόστηκε δοκιμαστικά στις εικόνες, υπήρξαν τέτοια «σκουπιδάκια» και γι’αυτό το κομμάτι αυτό παρέμεινε στον κώδικα, παρ’όλο που στον παραδοτέο κώδικα δεν υπάρχουν τέτοια σημεία στις εικόνες.

Το ίδιο επαναλαμβάνουμε και για την δεύτερη εικόνα:

```
for i in range(len(CLRcntrs)):
    m=False
    for j in range(len(CLRcntrs[i])):
            if (CLRcntrs[i][j][0][0]==1 or CLRcntrs[i][j][0][1]==1 or
CLRcntrs[i][j][0][0]==bnr.shape[1]-2 or CLRcntrs[i][j][0][1]==bnr.shape[0]-
2):
                m=True
                break
    if m:
        CLRakr[i]=0
    else:
        if len(CLRcntrs[i])>20:
            CLRttlclls=CLRttlclls+1

for i in range(0,CLRakr.size-1):
    k=CLRakr.size-1-i
    if CLRakr[k]==0:
        del CLRcntrs[k]
```

Η μεταβλητές ttlclls, CLRttlclls αποθηκεύουν το πλήθος των διαγραμμάτων που έχουμε κρατήσει (άρα και τον αριθμό των κυττάρων που έχουμε τελικά αναγνωρίσει!).

Στην συνέχεια θα μετρήσουμε τον αριθμό των εικονοστοιχείων που περιβάλλεται από το κάθε διάγραμμα.

```
cntrsl = np.zeros((len(cntrs),),dtype=int)
CLRcntrsl = np.zeros((len(CLRcntrs),),dtype=int)

for k in range(0,len(cntrs)):
    cntrsl[k] = cv2.contourArea(cntrs[k])

for k in range(0,len(CLRcntrs)):
    CLRcntrsl[k] = cv2.contourArea(CLRcntrs[k])
```

Η cv2.contourArea μας δίνει το εμβαδόν του διαγράμματος του κάθε κυττάρου. Το επαναλαμβάνουμε για την $2^η$ εικόνα.

Στην συνέχεια, θα βρούμε τα άκρα των κυττάρων, βρίσκοντας την μέγιστη και την ελάχιστη οριζόντια και κάθετη συνιστώσα στις συντεταγμένες τους.

```
maxmins = np.zeros((len(cntrs),4),dtype=int)
CLRmaxmins = np.zeros((len(CLRcntrs),4),dtype=int)

for k in range(0,len(cntrs)):
    max=[0,0]
    min=[img.shape[1],img.shape[1]]
    print("[NF7]Calculating area for cell: ", k + 1, "/", len(cntrs))
    for p in range(0,len(cntrs[k]-1)):
        if max[0]<cntrs[k][p][0][0]:
            max[0]=cntrs[k][p][0][0]
            maxmins[k,2]=cntrs[k][p][0][0]
        if max[1]<cntrs[k][p][0][1]:
            max[1]=cntrs[k][p][0][1]
            maxmins[k,3]=cntrs[k][p][0][1]
        if min[0]>cntrs[k][p][0][0]:
            min[0]=cntrs[k][p][0][0]
            maxmins[k,0]=cntrs[k][p][0][0]
        if min[1]>cntrs[k][p][0][1]:
            min[1]=cntrs[k][p][0][1]
            maxmins[k,1]=cntrs[k][p][0][1]
```

Δηλαδή η μέγιστη και η ελάχιστη συνιστώσα των X,Y του κάθε διαγράμματος αποθηκεύεται στον πίνακα maxmins (CLRmaxmins για την δεύτερη εικόνα αντίστοιχα).

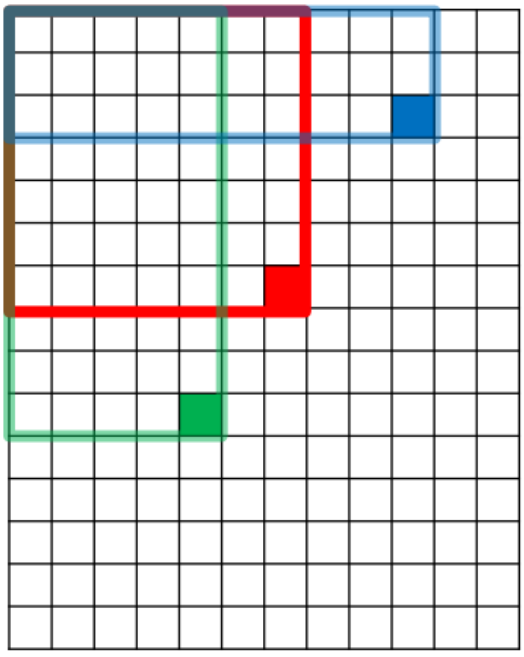
Το ίδιο επαναλαμβάνουμε για την άλλη εικόνα.

```
for k in range(0,len(CLRcntrs)):
    max=[0,0]
    min=[img2.shape[1],img2.shape[1]]
    print("[N7]Calculating area for cell: ", k + 1, "/", len(CLRcntrs))
    for p in range(0,len(CLRcntrs[k]-1)):
        if max[0]<CLRcntrs[k][p][0][0]:
            max[0]=CLRcntrs[k][p][0][0]
            CLRmaxmins[k,2]=CLRcntrs[k][p][0][0]
        if max[1]<CLRcntrs[k][p][0][1]:
            max[1]=CLRcntrs[k][p][0][1]
            CLRmaxmins[k,3]=CLRcntrs[k][p][0][1]
        if min[0]>CLRcntrs[k][p][0][0]:
            min[0]=CLRcntrs[k][p][0][0]
            CLRmaxmins[k,0]=CLRcntrs[k][p][0][0]
        if min[1]>CLRcntrs[k][p][0][1]:
            min[1]=CLRcntrs[k][p][0][1]
            CLRmaxmins[k,1]=CLRcntrs[k][p][0][1]
```

Αφού έχουμε μετρήσει τα κύτταρα και το εμβαδόν τους σε κάθε εικόνα, επιστρέφουμε στο κομμάτι που είχαμε παραλείψει προηγουμένως στον κώδικά μας:

```
sN7 = img.astype('int64')

for i in range(1,sN7.shape[1]):
    sN7[0][i] = sN7[0][i] + sN7[0][i-1]
for i in range(1,sN7.shape[0]):
    sN7[i][0] = sN7[i][0] + sN7[i-1][0]
for j in range(1,sN7.shape[0]):
    for i in range(1,sN7.shape[1]):
        sN7[j][i] = sN7[j][i] + sN7[j][i - 1] + sN7[j-1][i] - sN7[j-1][i-1]
```

Στην παράδοση της θεωρίας μας παρουσιάστηκε μια μέθοδος υπολογισμού του αθροίσματος του κάθε ορθογωνικού σχήματος με την εξής μέθοδο:



Υπολογίζουμε σε έναν πίνακα το εξής άθροισμα:

$$I(x,y) = I(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

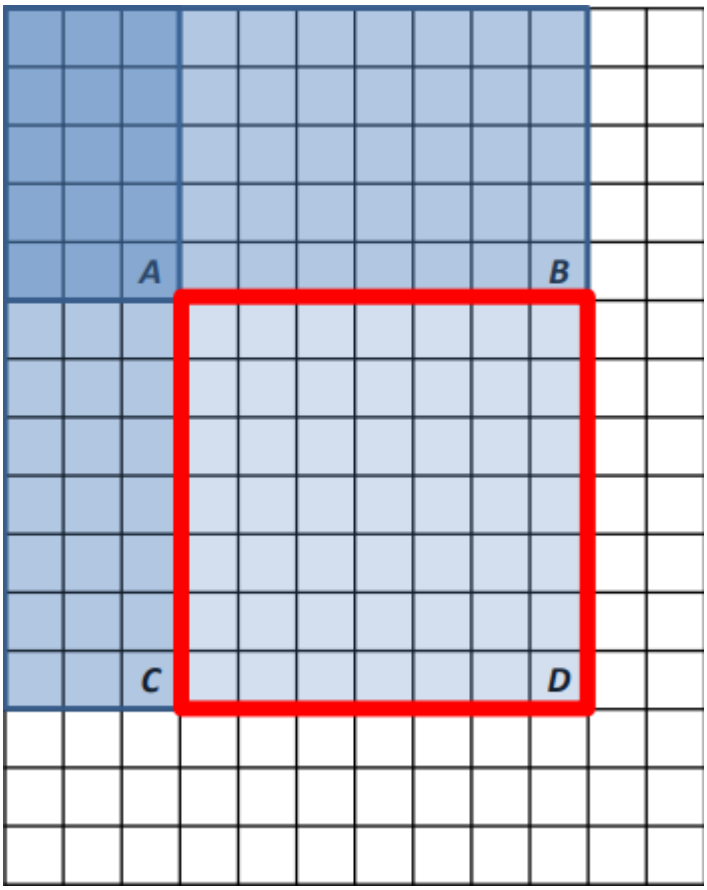
2	4	8	3	2	6	8	3	2	6	14	3	2	6	14	17	2	6	14	17
5	7	9	1	5	7	9	1	5	7	9	1	5	7	9	1	7	7	9	1
4	6	8	3	4	6	8	3	4	6	8	3	4	6	8	3	4	6	8	3
4	3	4	1	4	3	4	1	4	3	4	1	4	3	4	1	4	3	4	1
2	6	14	17	2	6	14	17	2	6	14	17	2	6	14	17	2	6	14	17
7	7	9	1	7	7	9	1	7	17	9	1	7	17	34	1	7	17	34	38
11	6	8	3	11	6	8	3	11	6	8	3	11	6	8	3	11	6	8	3
4	3	4	1	15	3	4	1	15	3	4	1	15	3	4	1	15	3	4	1

Και συνεχίζει…

Αν το έχουμε κάνει αυτό για όλο τον πίνακα, μπορούμε οποιαδήποτε στιγμή να υπολογίσουμε το άθροισμα των στοιχείων οποιουδήποτε ορθογωνίου υποπίνακα!

Όπου:

AREA=D+B+C-A



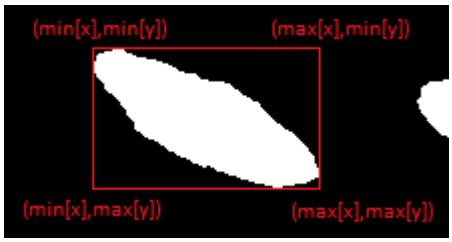
Αυτόν τον πίνακα ακριβώς (και με την σειρά που υπολογίζονται στους προηγούμενους πίνακες του παραδείγματος) υπολογίζουμε στο κομμάτι αυτό και το αποθηκεύουμε στον sN7.

Τέλος, εκτελούμε τους τελευταίους υπολογισμούς, πριν εμφανίζουμε τα στοιχεία των κυττάρων.

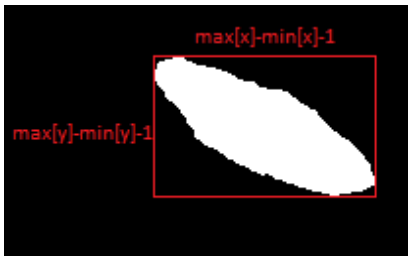
```
for i in range(0,len(CLRcntrs)):
    x=CLRmaxmins[i,2]+int((CLRmaxmins[i,0]-CLRmaxmins[i,2])/2)
    y=CLRmaxmins[i,3]+int((CLRmaxmins[i,1]-CLRmaxmins[i,3])/2)

    avgGrey=round((sN7[CLRmaxmins[i,3]-1,CLRmaxmins[i,2]-1]-
sN7[CLRmaxmins[i,1]+1,CLRmaxmins[i,2]-1]
-sN7[CLRmaxmins[i,3]-1,CLRmaxmins[i,0]+1]
+sN7[CLRmaxmins[i,1]+1,CLRmaxmins[i,0]+1])
/((CLRmaxmins[i,3]-CLRmaxmins[i,1]-1)*(CLRmaxmins[i,2]-CLRmaxmins[i,0]-1))
,2)

    cv2.putText(C2NF7, str(CLRcntrsl[i]), (x - 10, y - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0), 1, cv2.LINE_AA)

    cv2.putText(C2NF7, str(avgGrey), (x - 10, y + 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0),1, cv2.LINE_AA)

    cv2.putText(C2NF7, str(i+1), (CLRmaxmins[i, 0]+15, CLRmaxmins[i, 3]-
15), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0), 1, cv2.LINE_AA)
    cv2.rectangle(C2NF7, (CLRmaxmins[i, 2], CLRmaxmins[i, 3]),
(CLRmaxmins[i, 0], CLRmaxmins[i, 1]),  (255, 2, 2), 1)
```

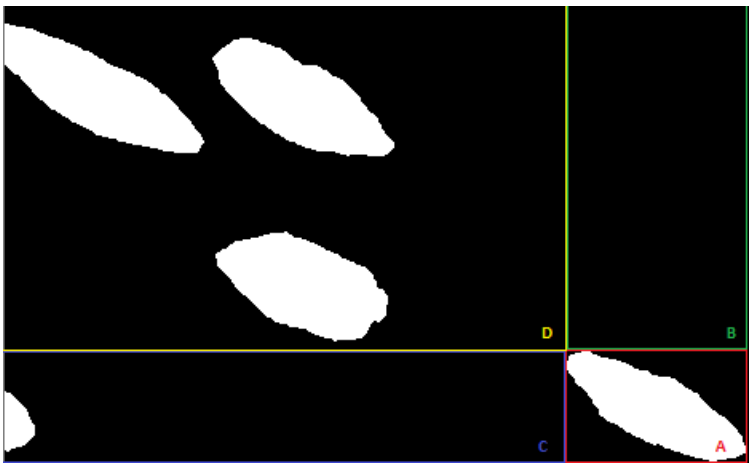
Αρχικά υπολογίζουμε το μέσο του κάθε κυττάρου με τις μέγιστες/ελάχιστες συνιστώσες του περιγράμματός του πού είχαμε υπολογίσει στο minmax στις μεταβλητές x,y (x = min[x] + (max[x]-min[x]), y=min[y]+(max[y]-min[y]).

Οι συνιστώσες αυτές μάλιστα αποτελούν τις συντεταγμένες του bounding box του αντίστοιχου κυττάρου:



Και έτσι μπορούμε να υπολογίσουμε το εμβαδόν του bounding box βρίσκοντας τις πλευρές του:





Ξέροντας λοιπόν τις συντεταγμένες υπολογίζουμε με την μέθοδο που περιγράψαμε προηγουμένως το άθροισμα των διαβαθμίσεων του γκρι των εικονοστοιχείων μέσα το κάθε bounding box.

Διαιρώντας το με το εμβαδόν του, βρίσκουμε την μέση διαβάθμιση του γκρι μέσα σε αυτό. Αυτό ακριβώς αποθηκεύουμε στην avgGrey.

Η cv2.putText τυπώνει στην εικόνα που θέλουμε, το string που της δίνουμε, αρχίζοντας από τις συντεταγμένες που επιθυμούμε, με το font, το scale, το χρώμα, το πάχος που θέλουμε και το αν θέλουμε να υποστεί anti-aliasing για να φαίνεται καθαρό.

Με τα πρώτα δύο putText, τυπώνουμε λίγο πάνω και λίγο κάτω από το κέντρο του bounding box (άρα και του κυττάρου) το πλήθος των εικονοστοιχείων που αποτελούν το κύτταρο και υπολογίσαμε με την cv2.contourArea και το πλήθος την μέση διαβάθμιση του γκρι των εικονοστοιχείων του bounding box που μόλις υπολογίσαμε αντίστοιχα.

Το τρίτο, τυπώνει τον αριθμό του κάθε κυττάρου στο κάτω αριστερό μέρος του bounding box (Η απαρίθμηση των κυττάρων έχει γίνει από το κάτω μέρος της εικόνας, προς το πάνω).

Τέλος, η cv2.rectangle τυπώνει στην εικόνα ένα ορθογώνιο, με άκρα τα σημεία που επισημαίνουμε (τα 2 απέναντι άκρα του bounding box, δηλαδή τα (max[x],max[y]), (min[x],min[y]), με το χρώμα και το πάχος (σε εικονοστοιχεία) που επιθυμούμε.

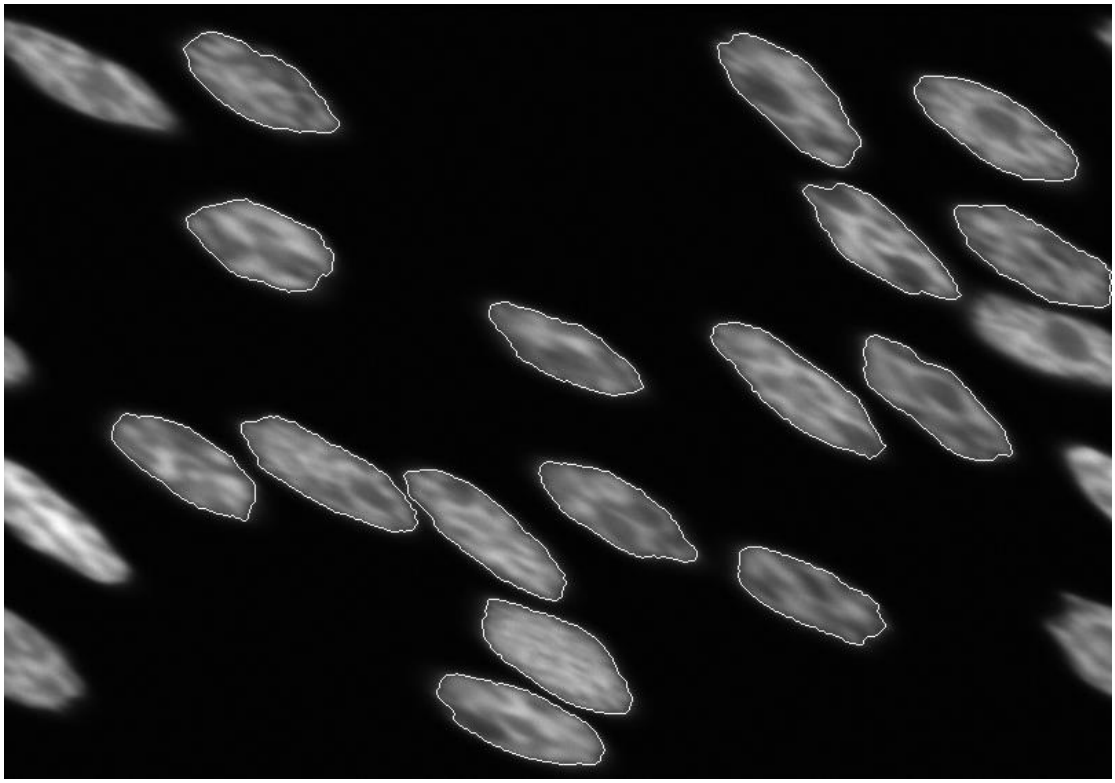
Τα παραπάνω τα εφαρμόζουμε πάνω στην αρχική, καθαρή εικόνα NF7. Επαναλαμβάνουμε για την $2^{η}$ εικόνα.

```
for i in range(0,len(CLRcntrs)):
    x=CLRmaxmins[i,2]+int((CLRmaxmins[i,0]-CLRmaxmins[i,2])/2)
    y=CLRmaxmins[i,3]+int((CLRmaxmins[i,1]-CLRmaxmins[i,3])/2)
    avgGrey=round((sN7[CLRmaxmins[i,3]-1,CLRmaxmins[i,2]-1]-
sN7[CLRmaxmins[i,1]+1,CLRmaxmins[i,2]-1]-sN7[CLRmaxmins[i,3]-
1,CLRmaxmins[i,0]+1]+sN7[CLRmaxmins[i,1]+1,CLRmaxmins[i,0]+1])/((CLRmaxmins
[i,3]-CLRmaxmins[i,1]-1)*(CLRmaxmins[i,2]-CLRmaxmins[i,0]-1)),2)
    cv2.putText(C2NF7, str(CLRcntrsl[i]), (x - 10, y - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0), 1, cv2.LINE_AA)
    cv2.putText(C2NF7, str(avgGrey), (x - 10, y + 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0),1, cv2.LINE_AA)
    cv2.putText(C2NF7, str(i+1), (CLRmaxmins[i, 0]+15, CLRmaxmins[i, 3]-
15), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (245, 0, 0), 1, cv2.LINE_AA)
    cv2.rectangle(C2NF7, (CLRmaxmins[i, 2], CLRmaxmins[i, 3]),
(CLRmaxmins[i, 0], CLRmaxmins[i, 1]),  (255, 2, 2), 1)
```

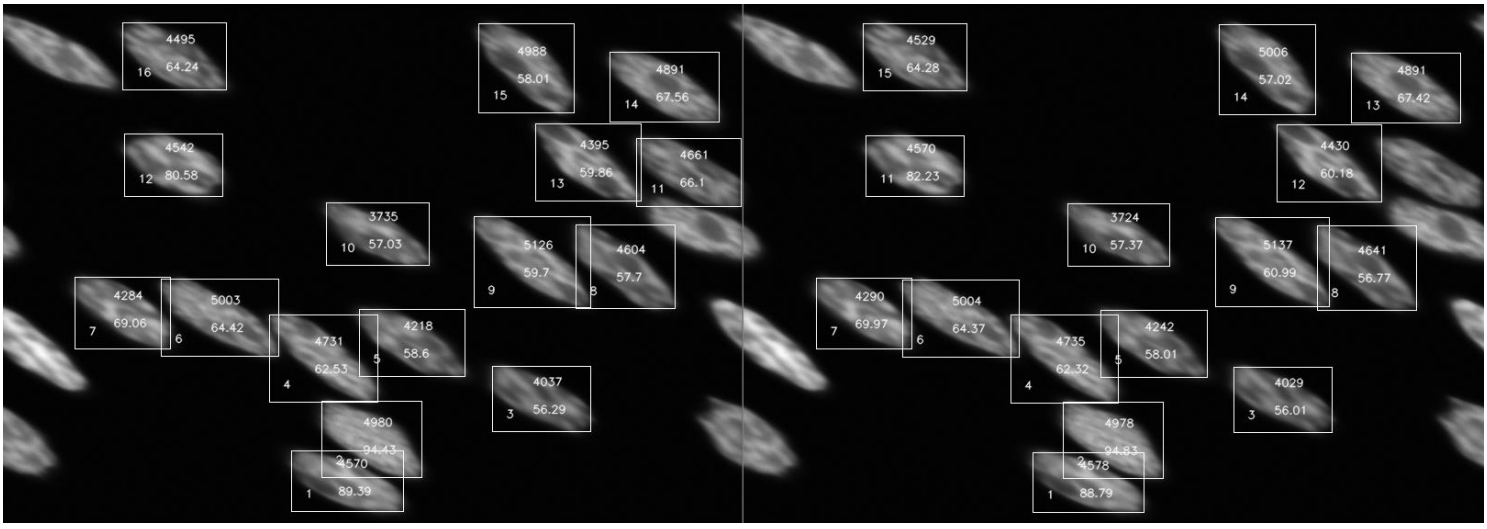
Ο παρακάτω κώδικας που βρίσκεται και πάλι στα σχόλια, χρησιμοποιούταν για να σχεδιάσει και να εμφανίσει στην αρχική εικόνα τα περιγράμματα που είχαν βρεθεί για το κάθε κύτταρο.

```
cv2.drawContours(NF7,cntrs,-1,(245,5,5),1)
cv2.namedWindow('NF7 CNTRS')
cv2.imshow('NF7 CNRS', NF7)
v2.waitKey(0)
```

Αυτό εφαρμοζόταν στα πρώτα στάδια της ολοκλήρωσης της εργασίας για την επαλήθευση των αποτελεσμάτων, όπως φαίνεται στην παρακάτω εικόνα.



Εμφανίζουμε τις τελικές εικόνες (και τυπώνουμε τον αριθμό των κυττάρων της κάθε μιας).



Αριστερά βλέπουμε την NF7 και δεξιά την N7. Παρατηρούμε ότι στην NF7 (η εικόνα που είχε αρχικά θόρυβο) εντοπίζεται ένα από τα κύτταρα που φαινομενικά ακουμπάει στην άκρη της εικόνας και όντως στην δεξιά εικόνα δεν περιλαμβάνεται. Φαίνεται λοιπόν πως στο φιλτράρισμά της, όλα τα εικονοστοιχεία δεξιά του, έχουν «μαυρίσει» όλα!

Επειδή για το συγκεκριμένο κύτταρο είναι αμφιλεγόμενο το αν «ακουμπάει» ή όχι την άκρη της εικόνας ακόμη και οπτικά, δεν προσπάθησα να το διαγράψω. Αν θέλαμε να μην περιληφθεί στο σύνολο των κυττάρων, θα μπορούσαμε για παράδειγμα να μειώσουμε το threshold της μετατροπής της εικόνας σε binary, ώστε κατά πάσα πιθανότητα να «ακουμπήσει» τελικά το άκρο. Όμως - προσωπικά - δεν το θεώρησα λάθος.

Ευχαριστώ για την προσοχή σας!

Καρασακαλίδης Αλέξανδρος Ηλίας

15/11/2019