1η Εργασία
Όραση Υπολογιστών
Μπούρχα Ιωάννα 58019

Επιβλέπων καθηγητής: Ιωάννης Πρατικάκης
Επιβλέπων εργαστηρίου: Λάζαρος Τσοχατζίδης

Ακαδημαϊκό έτος: 2022 - 2023

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΜ&ΜΥ

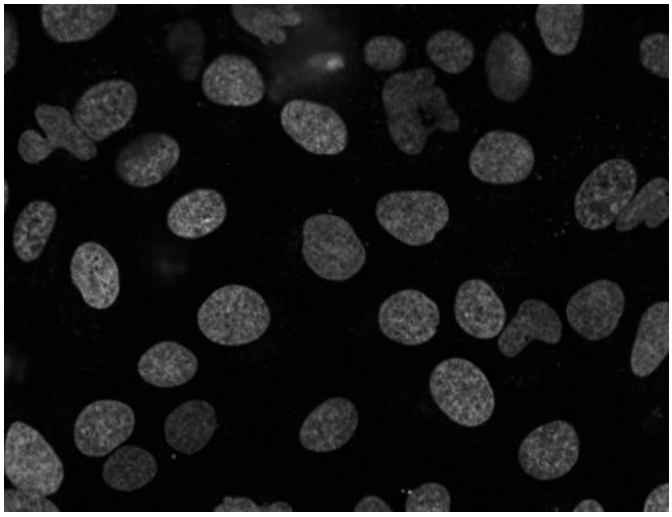
ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

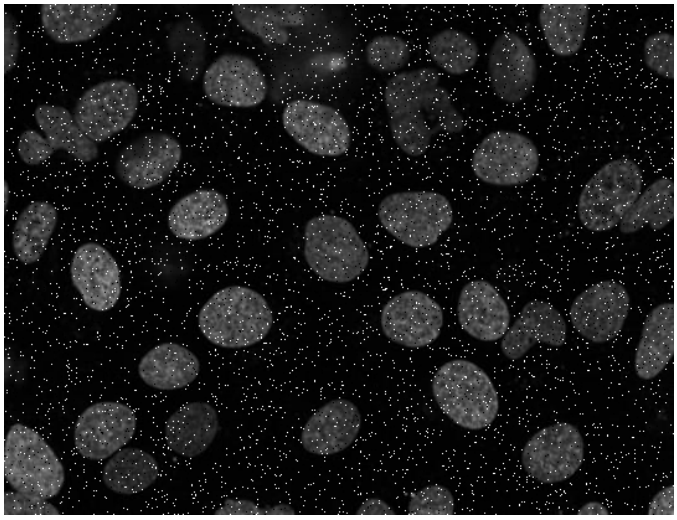
Στην παρούσα αναφορά περιγράφεται η λύση της πρώτης εργασίας στο μάθημα «Όραση Υπολογιστών». Πρόκειται να αναλυθεί το θεωρητικό υπόβαθρο, ο κώδικας και τα ενδιάμεσα αποτελέσματα. Τέλος, θα παρουσιαστούν οι προβληματισμοί της συγγραφέως.

Σύμφωνα με την εκφώνηση, δίνονται δύο εικόνες, μία χωρίς θόρυβο (9_original) και μία με θόρυβο τύπου αλατιού-πιπεριού (9_noise), και ζητείται η ανάπτυξη αλγορίθμου ο οποίος θα υπολογίζει:

- Τον αριθμό των αντικειμένων στο χώρο της εικόνας
- Την επιφάνεια (ως αριθμός εικονοστοιχείων) κάθε αντικειμένου και του κουτιού που τα περιβάλει
- Την μέση τιμή της διαβάθμισης του γκρι με τρόπο ώστε η ταχύτητα εκτέλεσης του του υπολογισμού να είναι ανεξάρτητη του μεγέθους του αντικειμένου.



Εικόνα 1: 9_original



Εικόνα 2: 9_noise

Επίσης, δόθηκαν οι ακόλουθες 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, χρησιμοποιώντας τη συνάρτηση ‘cv2.threshold’.

3. Για να γίνουν οι ζητούμενες μετρήσεις θα πρέπει να εφαρμόσετε κατάλληλη μεθοδολογία που να βρίσκει συνδεδεμένα αντικείμενα, που υλοποιείται στη συνάρτηση ‘cv2.connectedComponents’.

4. Για την εύρεση του περιβάλλοντος κουτιού ενός συνόλου (λευκών) εικονοστοιχείων, μπορείτε να χρησιμοποιήσετε τη συνάρτηση ‘cv2.boundingRect’.

5. Για την σχεδίαση παραλληλόγραμμων και κειμένου σε μια εικόνα, μπορείτε να χρησιμοποιήσετε τις συναρτήσεις ‘cv2.rectangle’ και ‘cv2.putText’, αντίστοιχα.

Εισαγωγή δεδομένων στον αλγόριθμο

Ξεκινάμε φορτώνοντας τις βιβλιοθήκες που θα χρησιμοποιήσουμε και τις εικόνες σε πίνακες και μορφή grayscale (ασπρόμαυρες).

```
import cv2
import numpy as np


filename = '9_original.png'
img1 = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

filename = '9_noise.png'
img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
```

Η βιβλιοθήκη cv2 της OpenCV παρέχει ένα σύνολο εργαλείων για την επεξεργασία εικόνων υπό μορφή πινάκων.

Η βιβλιοθήκη numpy παρέχει σύνολο εργαλείων για επεξεργασία διανυσμάτων και πινάκων.

Η εντολή cv2.imread αποθηκεύει στην μεταβλητή τον πίνακα που περιγράφει την αντίστοιχη εικόνα υπό μορφή grayscale λόγω του flag cv2. IMREAD_GRAYSCALE.

Δημιουργία φίλτρου για την αφαίρεση θορύβου

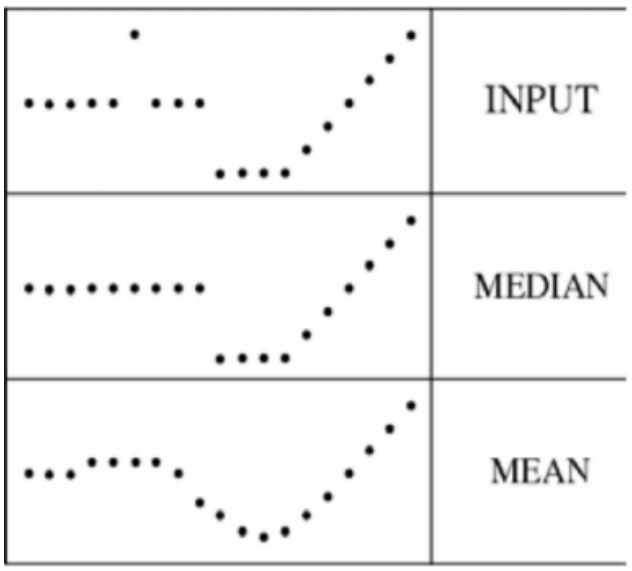
Προκειμένου τα αποτελέσματα μας να είναι σωστά και για τις δύο εικόνες θα πρέπει να αφαιρέσουμε από την δεύτερη τον θόρυβο τύπου αλατιού-πιπεριού. Αυτό γίνεται χρησιμοποιώντας ένα φίλτρο το οποίο παίρνει σαν είσοδο την εικόνα με θόρυβο και μου επιστρέφει εικόνα χωρίς θόρυβο. Η επιλογή του φίλτρου γίνεται με βάση το είδος και τα χαρακτηριστικά του θορύβου της αρχικής εικόνας.

Ο θόρυβος τύπου αλατιού-πιπεριού (salt and pepper) είναι γνωστός και ως impulse noise. Παρουσιάζεται ως αραιά τυχαία τοποθετημένα λευκά και μαύρα pixel. Οι τιμές αυτών σε κανάλι διαβάθμισης του γκρι είναι αντίστοιχα 255 και 0. Αυτό, έχει ως αποτέλεσμα την εισαγωγή πολλών ακραίων τιμών (outliers) συγκριτικά με τις τιμές των στοιχείων που έχουμε. Επομένως, το φίλτρο που θα επιλέξουμε για την αποθορυβοποίηση θα πρέπει να είναι ανθεκτικό σε αυτές τις ακραίες τιμές διατηρώντας ταυτόχρονα όσο καλύτερα γίνεται τις ακμές των αντικειμένων.

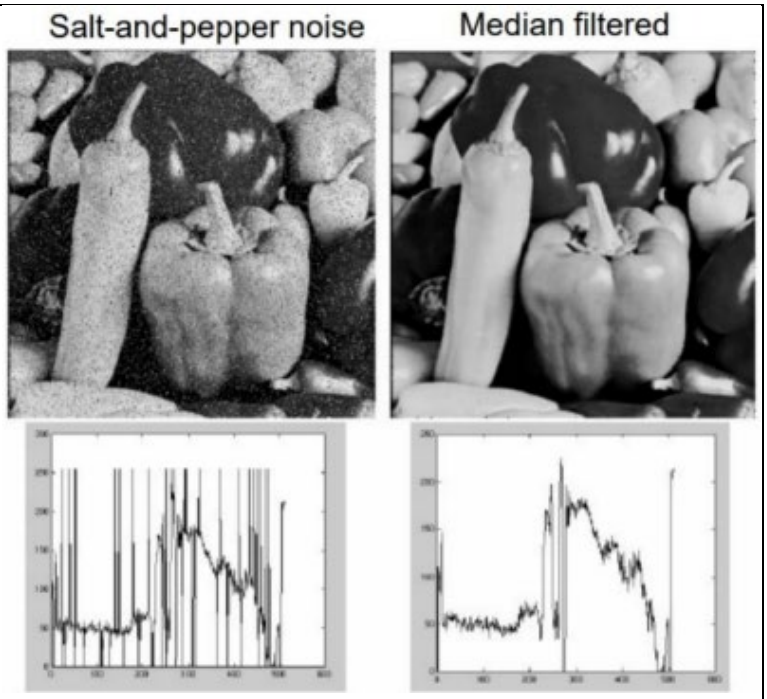
Μερικά φίλτρα κατάλληλα για τέτοια αποθορυβοποίηση είναι:

- ⇨ Median filter
- ⇨ Gaussian filter
- ⇨ Mean filter
- ⇨ Opening
- ⇨ Closing

Εφόσον καλούμαι να υλοποιήσω μόνη μου τον κώδικα για το φίλτρο που θα χρησιμοποιήσω, επέλεξα να εφαρμόσω το median filter, καθώς έχει ιδιαιτέρως καλά αποτελέσματα και είναι εύκολο στην υλοποίησή του.



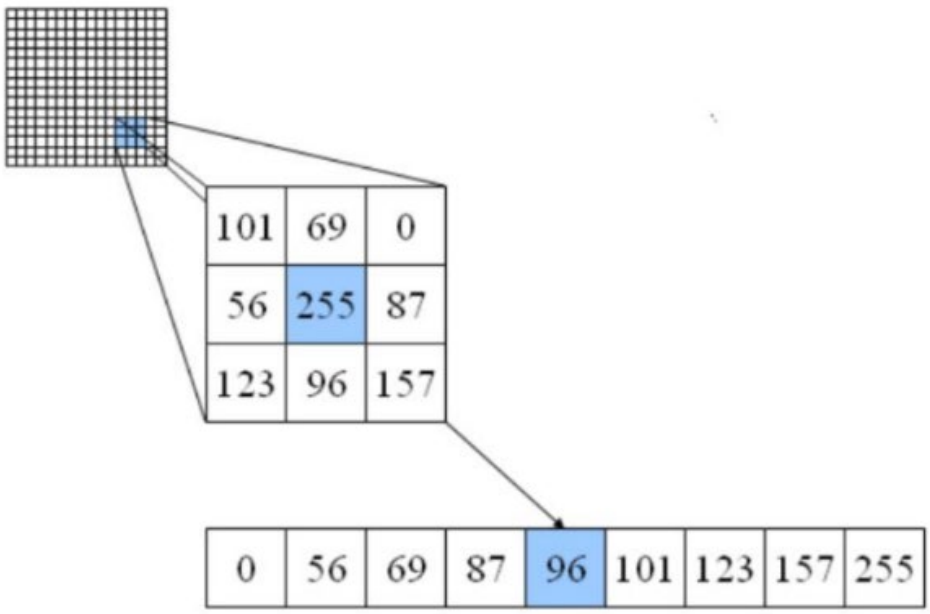
Εικόνα 3: Σύγκριση αποτελεσμάτων με χρήση φίλτρων median και mean



Εικόνα 4: Αποτέλεσμα με χρήση median filter

Λειτουργία Median Filter

Σύμφωνα με την θεωρία η τιμή του κάθε pixel αντικαθίσταται από την μέση τιμή όλων των pixels ενός τετραγωνικού παραθύρου (kernel) περιττών διαστάσεων με κέντρο το pixel του οποίου θα αλλάξει η τιμή.



Εικόνα 5: Λειτουργία median filter

Αξίζει να σημειωθεί ότι παρουσιάζει μεγαλύτερη ανθεκτικότητα στις ακραίες τιμές (outliers) σε σχέση με τον mean filter.

Υλοποίηση του φίλτρου

Αρχικά δημιουργώ ένα αντίγραφο της αρχικής εικόνας όπου θα αποθηκεύω τις τιμές του φίλτρου. Με αυτόν τον τρόπο, οι επόμενες τιμές δεν επηρεάζονται από την εφαρμογή του φίλτρου στις προηγούμενες. Σε περίπτωση που η εικόνα μου είχε έντονο θόρυβο θα μπορούσα να παραλείψω αυτό το βήμα για ένα καλύτερο αποτέλεσμα.

Όσο μεγαλύτερη είναι η διάσταση του παραθύρου, τόσο πιο θολό είναι το αποτέλεσμα του φιλτραρίσματος. Εφόσον θέλω να διατηρήσω όσο περισσότερη πληροφορία μπορώ, επιλέγω το ελάχιστο μέγεθος. Το παράθυρό μου είναι 3X3.

Για να έχω πρόσβαση σε κάθε εικονοστοιχείο της εικόνας δημιουργώ δύο δομές επανάληψης. Στην λίστα temp προς εύρεση της median τιμής προσθέτω τα εικονοστοιχεία τα οποία ανήκουν τόσο στο παράθυρο όσο και στην εικόνα, δηλαδή δεν βγαίνουν εκτός αυτής. Για την υλοποίηση αυτού του ελέγχου αξιοποιούνται 2 δομές ελέγχου.

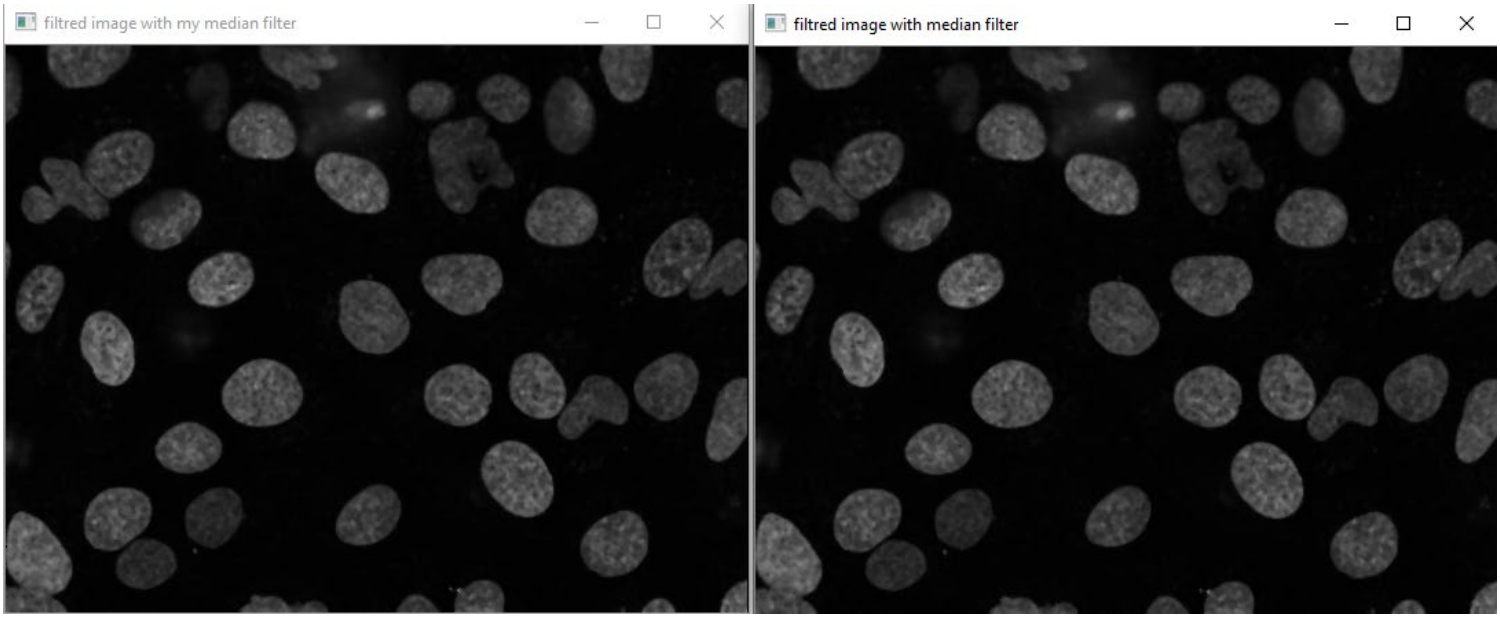
Τέλος, ταξινομώ την λίστα σε αύξοντα σειρά και επιλέγω το μεσαίο στοιχείο.

Ο κώδικας:

```
def median_filter(data, filter_size):
    border = filter_size // 2

    median = np.copy(data)

    rows = len(data)
    colums = len(data[0])

    for img_rows in range(rows):
        for img_colums in range(colums):
            temp = []
            for kernel_rows in range(filter_size):
                if img_rows + kernel_rows - border < 0 or img_rows + kernel_rows - border >
rows - 1:
                    #for kernel_colums in range(filter_size):
                        next
                else:
                    if img_colums + kernel_rows - border < 0 or img_colums + border > colums
- 1:
                        temp.append(0)
                    else:
                        for kernel_colums in range(filter_size):
                            temp.append(data[img_rows + kernel_rows - border][img_colums +
kernel_colums - border])

            temp.sort()
            median[img_rows][img_colums] = temp[len(temp) // 2]
    return median
```

```
# --------    Filter Application    -------- #
arr = np.array(img)
img = median_filter(arr, 3)
```

Επιπλέον γίνεται σύγκριση και των αποτελεσμάτων της χρήσης του “Median Filtering” της OpenCV με την δική μου συνάρτηση. Το αποτέλεσμα είναι ικανοποιητικό.



Εικόνα 6: Σύγκριση αποτελεσμάτων

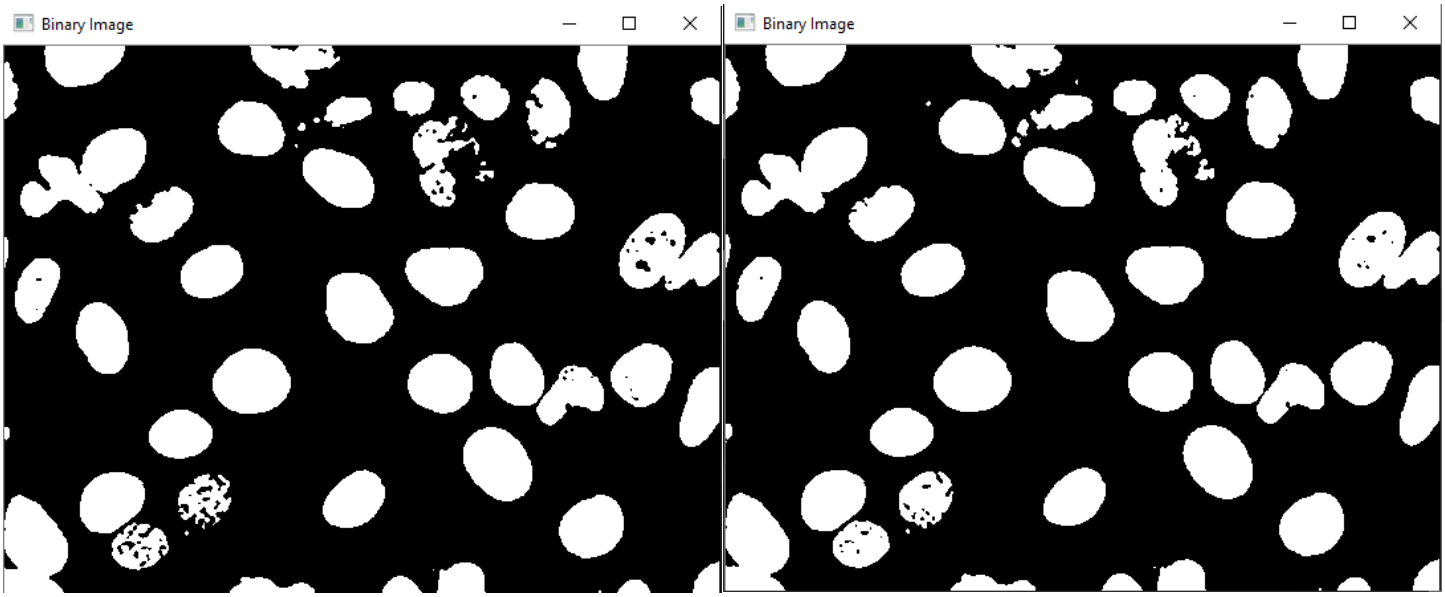
Μέτρηση κυττάρων

Δημιουργία δυαδικής εικόνας

Για να μπορέσουμε να εντοπίσουμε και να μετρήσουμε τα εικονιζόμενα κύτταρα πρέπει να πραγματοποιήσω ορισμένους μορφολογικούς μετασχηματισμούς. Η υλοποίηση αυτών προϋποθέτει πρώτα να μετατρέψω την αποθορυβοποιημένη εικόνα σε δυαδική (binary image).

Εφόσον η εικόνα μου είναι ήδη σε grayscale, πρέπει να επιλέξω μία τιμή του γκρι η οποία ονομάζεται κατώφλι (threshold). Κάθε pixel με τιμή μεγαλύτερη από το κατώφλι παίρνει την τιμή 255 και γίνεται λευκό, ενώ κάθε pixel με τιμή μικρότερη από το κατώφλι παίρνει την τιμή 0 και γίνεται μαύρο. Η τιμή του threshold αποφασίζεται τυχαία, οπότε μπορώ να βρω την κατάλληλη τιμή με try and error ή με την χρήση κάποιου αλγορίθμου όπως ο Otsu's Thresholding που υπάρχει στην OpenCV. Αυτός ο αλγόριθμος επεξεργάζεται την εικόνα της εισόδου, αποκτά το histogram της εικόνας, την κατανομή των pixels και υπολογίζει την ζητούμενη τιμή αυτόματα.

Ο αλγόριθμος έδωσε τιμή κατωφλιού 34 αλλά με την μέθοδο του try and error παρατήρησα ότι η τιμή 30 δίνει καλύτερα αποτελέσματα και για αυτό και προτιμήθηκε.



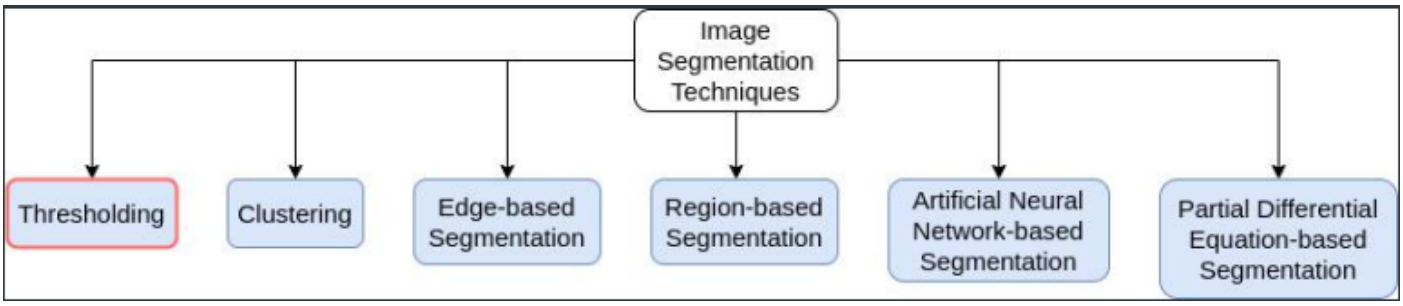
Εικόνα 7: (αριστερά) Με Otsu, τιμή κατωφλιού 34 *(δεξιά) Με try and error, τιμή κατωφλιού 30*

Παρατήρησα ότι όσο μεγαλύτερη η τιμή κατωφλιού τόσο πιο έντονα εξαλείφονται τμήματα των κυττάρων, ενώ όσο μικρότερη είναι η τιμή κατωφλιού τόσο περισσότερος ανεπιθύμητος θόρυβος παρουσιάζεται στην δυαδική εικόνα.

Ο κώδικας:

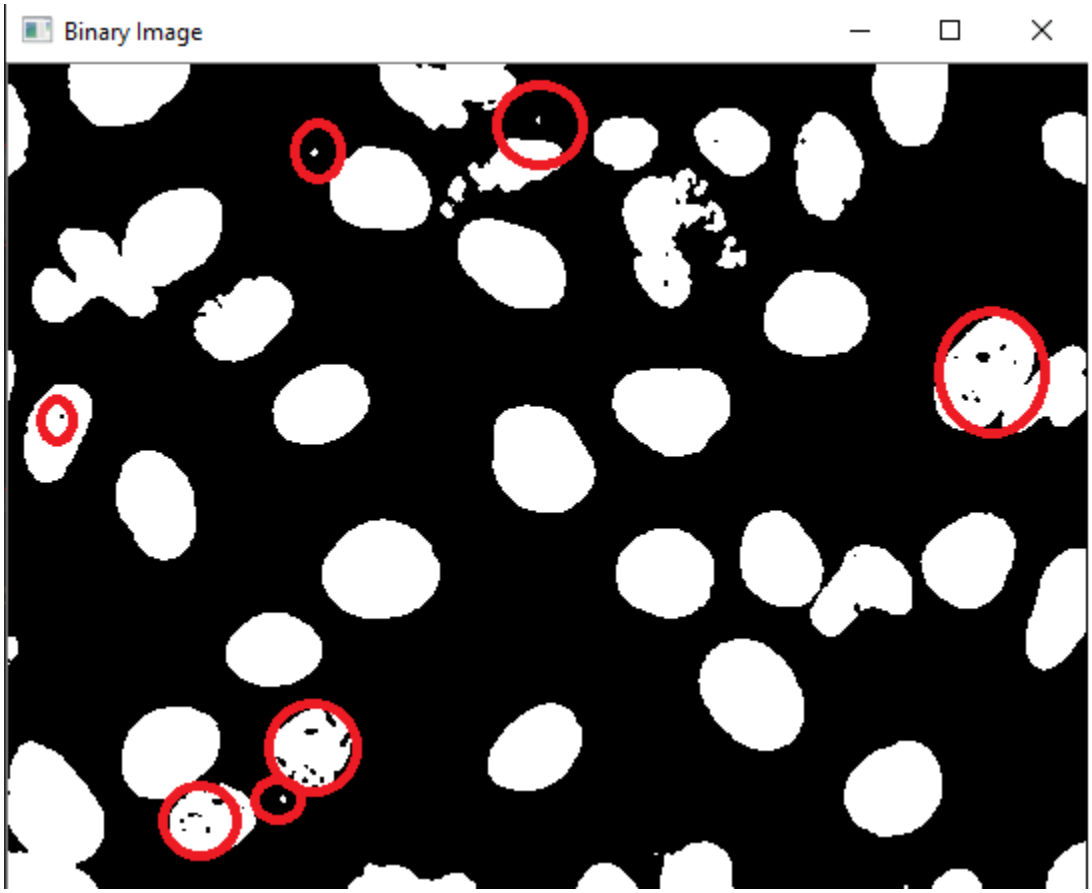
```
# --------    Binary Image    -------- #
#thresh, img_binary = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
_, img_binary = cv2.threshold(img, 30, 255, cv2.THRESH_BINARY)
cv2.imshow('Binary Image', img_binary)
```

Η μέθοδος που αναλύθηκε είναι μία από τις τεχνικές του Image Segmentation.



Επιλογή συνδυασμού μορφολογικών μετασχηματισμών

Η δυαδική μου εικόνα παρατηρώ ότι έχει ορισμένα σημεία τα οποία μπορεί να θεωρηθούν ως θόρυβος και να αλλοιώσουν το τελικό αποτέλεσμα. Χαρακτηριστικό παράδειγμα είναι ορισμένες περιοχές άσπρων εικονοστοιχείων σε διάφορα σημεία και μαύρες περιοχές στο εσωτερικό των κυττάρων, όπως υποδεικνύεται στην ακόλουθη εικόνα.



Εικόνα 8: Ενδεικτικά σημεία που θεωρούνται ως θόρυβος στην δυαδική μου εικόνα

Για την αντιμετώπιση των περισσότερων μαύρων τρυπών θα αξιοποιήσω τον μορφολογικό μετασχηματισμό closing. Σύμφωνα με την θεωρία, ο συνδυασμός αυτός υλοποιείται με την διαδοχική εφαρμογή των μορφολογικών τελεστών της διαστολής (dilation) και της διάβρωσης (erosion).

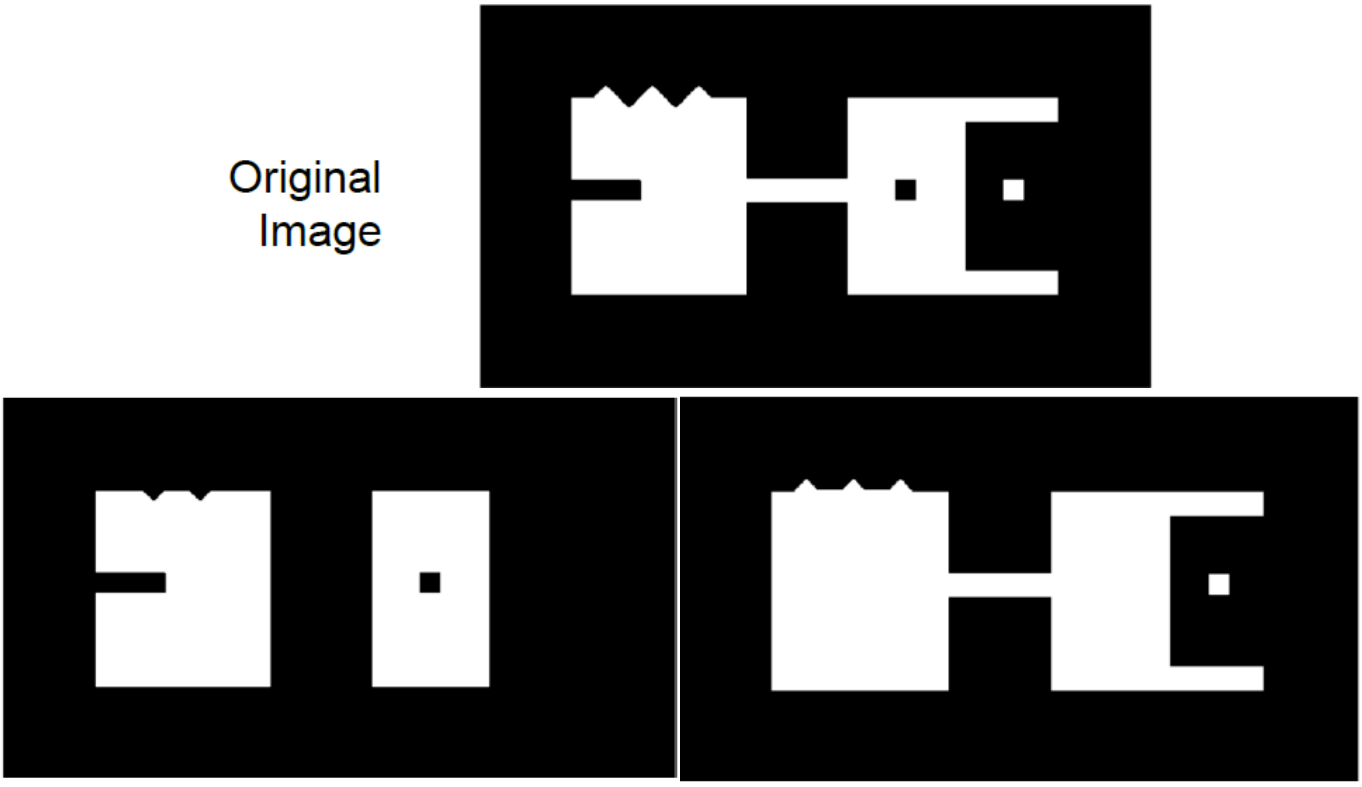
Υπάρχει ένα παράθυρο, structuring element, το οποίο μπορεί να έχει διάφορα σχήματα και μεγέθη. Αυτό εφαρμόζεται σε όλη την εικόνα και ανάλογα τον μετασχηματισμό

προσδιορίζει την τιμή του pixel στο οποίο εφαρμόζεται. Έπειτα από αρκετούς πειραματισμούς κατέληξα στο συμπέρασμα ότι το καλύτερο structing element είναι μία έλλειψη ακτίνων 3 και 2.

Στο erosion η τιμή του pixel στο οποίο εφαρμόζεται το structuring element γίνεται 255 (λευκό) μόνο αν όλα τα pixels στο structuring element έχουν την τιμή αυτή, δηλαδή αν βρίσκεται ολόκληρο μέσα στο αντικείμενο, αλλιώς παίρνει την τιμή 0. Πρακτικά, το αποτέλεσμα μιας τέτοιας πράξης είναι η διάβρωση των αντικειμένων και η εξαφάνιση των λευκών κουκίδων διαστάσεων μικρότερων αυτών του structuring element.

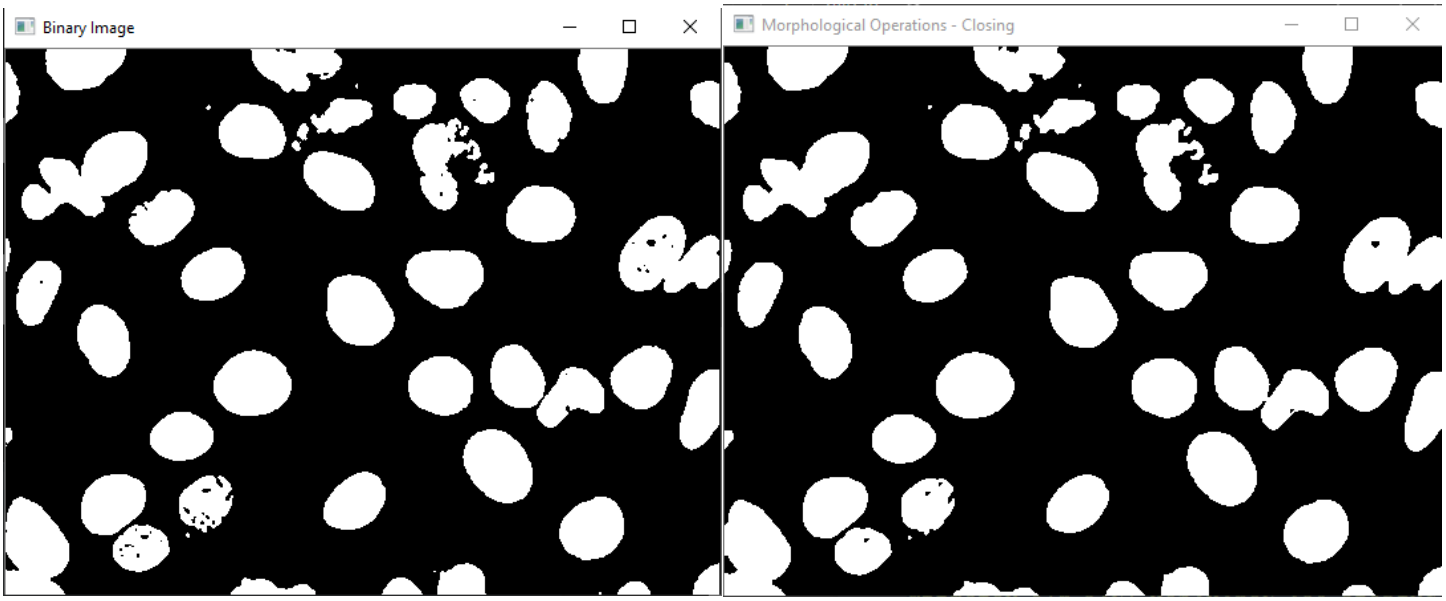
Στο dilation, η διαδικασία είναι ίδια μόνο που αυτή τη φορά το pixel θα γίνει λευκό (255) αν έστω ένα από τα γύρω pixels αυτού στο οποίο εφαρμόζεται το structuring element είναι λευκό, δηλαδή αν τουλάχιστον ένα εικονοστοιχείο του structuring element βρίσκεται εντός του αντικειμένου. Πρακτικά, το αποτέλεσμα μιας τέτοιας πράξης είναι η διαστολή των αντικειμένων.

Αξίζει να σημειωθεί ότι η διάβρωση και η διαστολή παραμορφώνουν το αντικείμενο, δηλαδή αλλάζουν τις διαστάσεις του, μη επιθυμητή κατάσταση. Για την αντιμετώπιση του προβλήματος αυτού εφαρμόζεται διαδοχικά τόσο ο ένας όσο και ο άλλος μετασχηματισμός, όπως ακριβώς ειπώθηκε και κατά την διάρκεια της αντίστοιχης διάλεξης. Ανάλογα με την σειρά που θα εφαρμοστούν οι δύο προαναφερόμενοι μετασχηματισμοί, έχουμε opening (πρώτα erosion και μετά dilation) ή closing (πρώτα dilation και μετά erosion).



Εικόνα 9: (πάνω) Αρχική εικόνα, (κάτω αριστερά) Αποτέλεσμα opening, (κάτω δεξιά) Αποτέλεσμα closing. Και στις δύο περιπτώσεις οι διαστάσεις του αντικειμένου παραμένουν αναλλοίωτες.

Το closing ουσιαστικά πραγματοποιεί «γέμισμα» των τρυπών που εμφανίζονται στο εσωτερικό του αντικειμένου (λευκά pixels) μικρότερων διαστάσεων του structing element, δηλαδή αντιμετωπίζει τον μαύρο θόρυβο στο εσωτερικό των κυττάρων.



Εικόνα 10: (αριστερά) Δυαδική εικόνα (δεξιά) Εικόνα μετά το closing

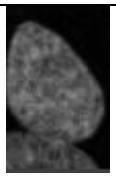
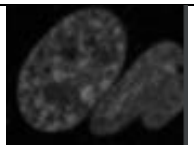
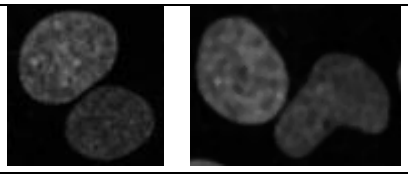
Ο κώδικας:

```
# --------    Morphological Operations    -------- #
# CLOSING
strel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2,2) )
img_close = cv2.morphologyEx(img_binary, cv2.MORPH_CLOSE, strel, iterations=2)
cv2.imshow('Morphological Operations - Closing', img_close)
```

Παρατηρώ ότι η πλειοψηφία των μαύρων περιοχών έχει αντιμετωπιστεί αποτελεσματικά, το μέγεθος των κυττάρων δεν έχει παραμορφωθεί σε μη επιθυμητό βαθμό και τα ορισμένα άσπρα σημεία παραμένουν.

Παρατηρώ ότι ορισμένα κύτταρα είναι ενωμένα:

A. δύο στην κάτω αριστερή γωνία λόγω τοποθεσίας τους το ένα πάνω στο άλλο ήδη από την αρχική εικόνα
B. δύο στο μέσο δεξιά τμήμα λόγω μικρού κενού μεταξύ τους το οποίο εξαφανίζεται κατά την μετατροπή σε δυαδική μορφή
C. δύο στο κάτω αριστερά τμήμα και δύο στο μέσο δεξιά λόγω των μορφολογικών μετασχηματισμών και του median φίλτρου.

Ενωμένα Κύτταρα	Στην τελική εικόνα	Στην αρχική αποθορυβοποιημένη εικόνα
A		
B		
C		

Προσπάθησα να διαχωρίσω τα κύτταρα της κατηγορίας B και C αλλά αλλοιώνονταν αρκετά τα υπόλοιπα, οπότε προτίμησα να τα αφήσω ως έχει, ώστε να έχω περισσότερα σωστά αποτελέσματα. Μία λύση του προβλήματος, θα ήταν να χωρίσω την αρχική μου εικόνα σε μικρότερες, να τις μεγεθύνω και να δώσω κάθε μία ως ξεχωριστή είσοδο στον αλγόριθμό μου.

Μέτρηση αριθμού κυττάρων

Για την μέτρηση των κυττάρων της εικόνας χρησιμοποιώ, σύμφωνα με την δεύτερη υπόδειξη, την συνάρτηση connectedComponents της OpenCV, η οποία σαρώνει την εικόνα και ομαδοποιεί τα εικονοστοιχεία με βάση την συνδεσιμότητά τους. Όλα τα εικονοστοιχεία μιας συνδεδεμένης περιοχής μοιράζονται παρόμοιες τιμές έντασης και συνδέονται με κάποιον τρόπο μεταξύ τους. Η συνάρτηση επιστρέφει έναν αριθμό που δηλώνει πόσα διαφορετικά connected components βρέθηκαν και έναν πίνακα όπου η τιμή των pixels κάθε connected components έχει ένα συγκεκριμένο αριθμό. Η αρίθμηση ξεκινάει από το 1.

Για να μην λαμβάνω ψευδή στοιχεία μετρώντας θόρυβο ως υποπεριοχή αποφασίζω να ορίσω ως όριο διαστάσεων το 11. Έτσι, υποπεριοχή θεωρείται ο,τι έχει διάσταση πλάτους και ύψους μεγαλύτερη από 11 εικονοστοιχεία. Το αποτέλεσμα, όπως θα φανεί στην συνέχεια, είναι ικανοποιητικό. Η τιμή του ορίου 11 εικονοστοιχείων επιλέχτηκε ύστερα από αρκετούς πειραματισμούς.

Τέλος, κατά την επανάληψη ελέγχεται αν κάθε pixel είναι λευκό.

Ο κώδικας:

```
num, val_pix = cv2.connectedComponents(eik)
for i in range(1, num):
    counter = 0
    regions = np.zeros(eik.shape, dtype=np.uint8)
    regions[val_pix == i] = 255
    x, y, w, h = cv2.boundingRect(regions)

    # to avoid noise, if the height and width of bounding rectangles are small enough
      they are considered noise
    if w < 11 or h < 11:
        counter += 1
        continue
    num_regions = i - counter
```

Υπολογισμός επιφάνειας κυττάρου και περιβάλλοντος κουτιού

Σχεδιασμός περιβάλλοντος κουτιού

Για να σχεδιαστεί το περιβάλλον κουτί απαιτούνται οι συντεταγμένες της επάνω αριστερά και της κάτω δεξιά γωνίας. Οι πληροφορίες αυτές παρέχονται από την συνάρτηση boundingRect της OpenCV, η οποία επιστρέφει τις συντεταγμένες της επάνω αριστερά γωνίας (x,y) καθώς και το πλάτος και το ύψος του κουτιού (w,h).

```
x, y, w, h = cv2.boundingRect(regions)
```

```
num_regions = i
print('---- Region Number ', str(num_regions), ": ----")
```

Αποφασίζω το περιβάλλον κουτί, καθώς και τα γράμματα του αύξοντα αριθμού των κυττάρων, να είναι κόκκινα (0, 0, 255). Υπενθυμίζω σε αυτό το σημείο ότι η εικόνα μου είναι σε μορφή grayscale οπότε δεν μπορεί να απεικονίσει χρώματα πέρα του μαύρου, του άσπορου και του γκρι. Για αυτό, δημιουργώ μία νέα εικόνα (img_final_rec) συγχωνεύοντας την αρχικώς αποθορυβοποιημένη εικόνα τρεις φορές κατά βάθος, ώστε να δημιουργηθούν αντίστοιχα frames για κάθε βασικό χρώμα της rgb παλέτας. Την εικόνα αυτή μαζί με τις διστάσεις του κουτιού δίνω σαν όρισμα στην συνάρτηση rectangle της OpenCV η οποία σχεδιάζει το περιβάλλον κουτί. Το αποτέλεσμα περνάει σαν όρισμα στην συνάρτηση putText της OpenCV η οποία σχεδιάζει τον μοναδικό αύξοντα αριθμό κάθε κυττάρου. Έτσι, τόσο το κουτί όσο και ο αριθμός σχεδιάζονται στην ίδια εικόνα.

```
img_final_rec = np.dstack([img, img, img])
```

```
color = (0,0,255)    #GBR
img_final_rec = cv2.rectangle(img_final_rec, (x, y), (x + w, y + h), color, 1)
img_final_text = cv2.putText(img_final_rec, str(num_regions), (x+10, y+h-10),
                             cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
```

Υπολογισμός επιφάνειας κυττάρου

Με σημείο αναφοράς ένα παράθυρο διαστάσεων της υποπεριοχής όπου εντοπίζεται το κύτταρο ερευνούνται τα λευκά σημεία (τιμή: 255), δηλαδή πόσα από τα εικονοστοιχεία της περιοχής είναι όντως κύτταρο. Όταν βρίσκεται λευκό εικονοστοιχείο, αυξάνεται η τιμή ενός μετρητή κατά μία μονάδα. Τελικώς, η επιφάνεια του κυττάρου είναι το περιεχόμενο του μετρητή.

```
# Calculating cells area:
cells_area_box = img_binary[y:y + h, x:x + w]
cells_area = 0
for y_i in range(1, cells_area_box.shape[0]):
    for x_i in range(1, cells_area_box.shape[1]):
        if cells_area_box[y_i][x_i] == 255:
            cells_area += 1
print("Area (px): ", cells_area)
```

Υπολογισμός επιφάνειας περιβάλλοντος κουτιού

Το εμβαδόν του περιβάλλοντος κουτιού προκύπτει από το γινόμενο των διαστάσεών του.

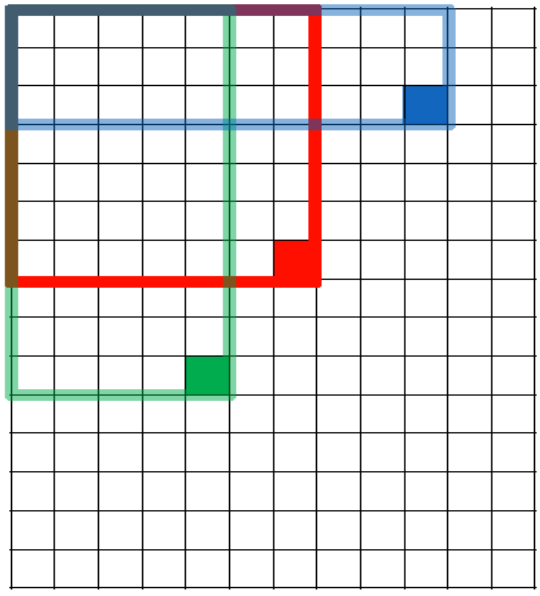
```
# Calculating the bounding box area with the dimensions of width and height
box_area = w * h
print("Bounding Box Area (px): ", box_area)
```

Υπολογισμός μέσης τιμής διαβάθμισης του γκρι

Πρακτικά ζητείται το άθροισμα όλων των εικονοστοιχείων κάθε περιοχής προς τα συνολικά της εικονοστοιχεία. Ζητείται ο υπολογισμός να είναι να είναι ανεξάρτητος του μεγέθους της περιοχής, δηλαδή η πολυπλοκότητά του να είναι O(1). Συνεπώς, είναι φανερή η ανάγκη χρήσης της συνάρτησης integral, η οποία θα υλοποιηθεί από την συγγραφέα σύμφωνα με την εκφώνηση.

Θεωρητική ανάλυση της integral

Η τιμή κάθε pixel της εικόνας integral προκύπτει από το άθροισμα όλων των τιμών των εικονοστοιχείων που βρίσκονται εντός ενός παραλληλογράμμου με άκρα το σημείο αναφοράς της εικόνας (συνήθως μία γωνία) και το προς εξέταση pixel. Στην παρούσα εργασία, εφόσον η αρίθμηση των περιεχομένων του πίνακα της εικόνας ξεκινάει από το επάνω αριστερό άκρο, αυτό θεωρείται και το σημείο αναφοράς μου.



Εικόνα 11: Παράδειγμα υπολογισμού της εικόνας integral

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

Εικόνα 12: Τύπος υπολογισμού της τιμής κάθε εικονοστοιχείου

Αξίζει να σημειωθεί ότι η τελική τιμή ορίζεται ως το άθροισμα των εικονοστοιχείων της αρχικής εικόνας με δείκτες μικρότερους από αυτής της integral, όχι μικρότερους ή ίσους. Συνεπώς, είναι αναγκαία η προσθήκη μίας επιπλέον σειράς στο επάνω και μίας επιπλέον στήλης στο δεξί μέρος, οι οποίες συγχωνεύονται κατάλληλα στον πίνακα της εικόνας. Με

αυτόν τον τρόπο, δεν χρειάζεται να ελέγχω αν ο δείκτης κάθε εικονοστοιχείου ικανοποιεί την προαναφερόμενη απαίτηση.

```
# building the extra row and colum
arr_v = np.zeros((h, 1))
arr_h = np.zeros((1, w + 1))

mat = np.hstack([arr_v, arr_img])
mat = np.vstack([arr_h, mat])
```

Προκειμένου να μην επηρεάζονται οι επόμενες τιμές από τις προηγούμενες δημιουργώ έναν μηδενικό πίνακα ίδιων διαστάσεων με την εικόνα, όπου αποθηκεύονται οι τιμές και δημιουργείται το τελικό αποτέλεσμα.

```
integral_image = np.zeros(mat.shape)
```

Με διπλή δομή επανάληψης με σημείο αναφοράς την γραμμή έχω πρόσβαση σε κάθε εικονοστοιχείο, την τιμή του οποίου προσθέτω στον μετρητή sum. Έτσι, σε κάθε εικονοστοιχείο περιέχεται το άθροισμα όλων των προηγούμενων της ίδιας γραμμής. Για να προσθέσω και το αποτέλεσμα των εικοστοιχείων της ίδιας στήλης, αρκεί στο άθροισμα του sum τελικώς να προσθέσω το περιεχόμενο της τιμής του εικονοστοιχείου από την εικόνα integral το οποίο βρίσκεται στην ίδια στήλη αλλά στην προηγούμενη γραμμή από το τρέχον. Για τα εικονοστοιχεία της πρώτης γραμμής της αρχικής εικόνας, προστίθεται το ουδέτερο στοιχείο της πρόσθεσης (0).

Σημειώνεται ότι ελέγχθηκε η σωστή λειτουργία της συνάρτησής μου συγκριτικά με την αντίστοιχη συνάρτηση της βιβλιοθήκης. Ο αντίστοιχος κώδικας βρίσκεται σε σχόλια στο τέλος.

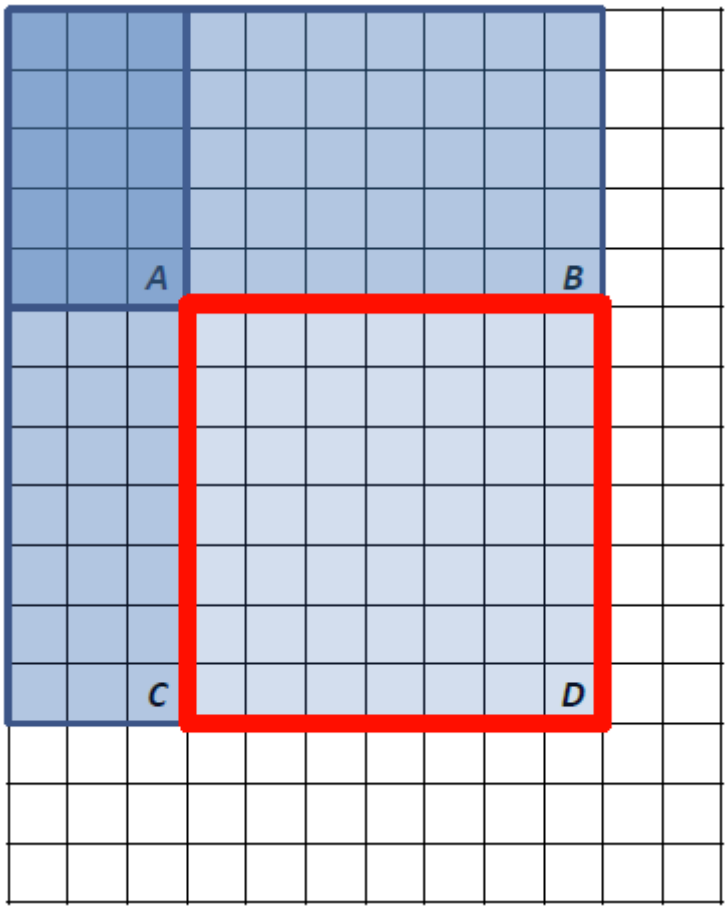
Ο κώδικας:

```
def my_integral(arr_img):
    h, w = len(arr_img), len(arr_img[0])

    """
    Integral image in OpenCV is defined as the sum of pixels in the original image with
    indices LESS THAN those of the integral image, not less than or equal to. Thus, an
    extra row and colum are needed. In this way there is no need to check if my indice
    is legal.
    For instance, if my rectagle covers the entire image, it may produce an index that
    falls out of array by 1. That is why the integral image is stored in a size that is
    by 1x1 larger than the original image.
    """

    # building the extra row and colum
    arr_v = np.zeros((h, 1))
    arr_h = np.zeros((1, w + 1))

    mat = np.hstack([arr_v, arr_img])
    mat = np.vstack([arr_h, mat])


    h, w = len(mat), len(mat[0])
    integral_image = np.zeros(mat.shape)

    # building integral image
    for y in range(1, h):  # rows
        sum = 0
        for x in range(1, w):  # colums
            sum += mat[y][x]
            integral_image[y][x] = sum + integral_image[y - 1][x]
```

```
    # checing if the output is correct
    # img2 = cv2.integral(img)
    # a = np.array_equal(img2, integral_image)

    return integral_image
```

Θεωρητική ανάλυση υπολογισμού της μέσης τιμής του γκρι

Για τον υπολογισμό της ζητούμενης τιμής κάθε υποπεριοχής, αρκεί να υπολογίσουμε τέσσερις αριθμούς, όπως φαίνεται στην παρακάτω εικόνα.

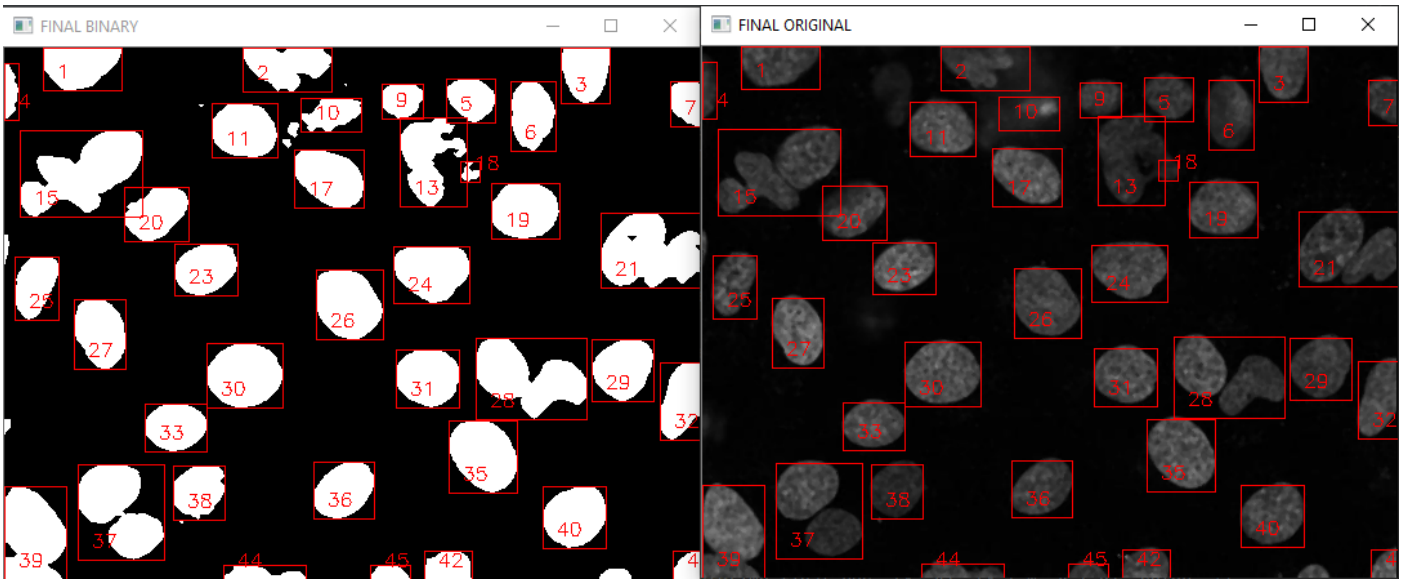


Γνωρίζοντας πλέον τις τιμές των A, B, C και D με βάση την εικόνα integral που δημιουργήσαμε προηγουμένως, μπορώ να υπολογίζω την μέση τιμή της διαβάθμισης του γκρι κάθε υποπεριοχής σύμφωνα με την πράξη:

$$A + D - B - C$$

Τελικό αποτέλεσμα και σχολιασμοί

Αποφάσισα να εκτυπώσω τα τετράγωνα και τον αύξοντα αριθμό των κυττάρων τόσο στην τελικώς επεξεργασμένη εικόνα όσο και στην αρχικώς αποθορυβοποιημένη.



Εικόνα 13: (δεξιά) Τελικώς επεξεργασμένη εικόνα (αριστερά) Αρχικώς αποθορυβοποιημένη εικόνα

Αποφάσισα να αποθηκεύσω και τις δύο εικόνες. Για την αποθήκευση αξιοποιήθηκε η εντολή imwrite της OpenCV. Οι εικόνες αντίστοιχα έχουν όνομα img_final_binary και img_final_original.

Συγκεντρωτικά όλα τα αποτελέσματα είναι:

```
---- Region Number  1 : ----
Area (px):  1425
Bounding Box Area (px):  2013
Mean graylevel value in bounding box:  195.20864381520119
---- Region Number  2 : ----
Area (px):  1485
Bounding Box Area (px):  2346
Mean graylevel value in bounding box:  173.58695652173913
---- Region Number  3 : ----
Area (px):  1239
Bounding Box Area (px):  1634
Mean graylevel value in bounding box:  203.968788249694
---- Region Number  4 : ----
Area (px):  297
Bounding Box Area (px):  484
Mean graylevel value in bounding box:  182.29338842975207
---- Region Number  5 : ----
Area (px):  898
Bounding Box Area (px):  1292
Mean graylevel value in bounding box:  184.3421052631579
---- Region Number  6 : ----
Area (px):  1332
Bounding Box Area (px):  1890
Mean graylevel value in bounding box:  185.38095238095238
---- Region Number  7 : ----
Area (px):  593
Bounding Box Area (px):  805
Mean graylevel value in bounding box:  208.43478260869566
---- Region Number  9 : ----
Area (px):  634
```

```
Bounding Box Area (px):  864
Mean graylevel value in bounding box:  200.39930555555554
---- Region Number  10 : ----
Area (px):  765
Bounding Box Area (px):  1222
Mean graylevel value in bounding box:  167.98281505728315
---- Region Number  11 : ----
Area (px):  1631
Bounding Box Area (px):  2142
Mean graylevel value in bounding box:  202.26190476190476
---- Region Number  13 : ----
Area (px):  1874
Bounding Box Area (px):  3588
Mean graylevel value in bounding box:  137.52090301003344
---- Region Number  14 : ----
Area (px):  61
Bounding Box Area (px):  108
Mean graylevel value in bounding box:  177.08333333333334
---- Region Number  15 : ----
Area (px):  3622
Bounding Box Area (px):  6365
Mean graylevel value in bounding box:  147.19088766692852
---- Region Number  17 : ----
Area (px):  1760
Bounding Box Area (px):  2430
Mean graylevel value in bounding box:  189.62345679012347
---- Region Number  18 : ----
Area (px):  114
Bounding Box Area (px):  240
Mean graylevel value in bounding box:  133.875
---- Region Number  19 : ----
Area (px):  1742
Bounding Box Area (px):  2279
Mean graylevel value in bounding box:  202.52303641948222
---- Region Number  20 : ----
Area (px):  1404
Bounding Box Area (px):  2100
Mean graylevel value in bounding box:  174.97857142857143
---- Region Number  21 : ----
Area (px):  3065
Bounding Box Area (px):  4466
Mean graylevel value in bounding box:  183.74160322436185
---- Region Number  22 : ----
Area (px):  37
Bounding Box Area (px):  92
Mean graylevel value in bounding box:  171.84782608695653
---- Region Number  23 : ----
Area (px):  1431
Bounding Box Area (px):  1960
Mean graylevel value in bounding box:  192.42091836734693
---- Region Number  24 : ----
Area (px):  1916
Bounding Box Area (px):  2596
Mean graylevel value in bounding box:  194.7862095531587
---- Region Number  25 : ----
Area (px):  1139
Bounding Box Area (px):  1666
Mean graylevel value in bounding box:  181.22448979591837
---- Region Number  26 : ----
Area (px):  2099
Bounding Box Area (px):  2808
Mean graylevel value in bounding box:  193.97435897435898
---- Region Number  27 : ----
Area (px):  1609
Bounding Box Area (px):  2160
Mean graylevel value in bounding box:  194.31944444444446
---- Region Number  28 : ----
Area (px):  2905
Bounding Box Area (px):  5418
```

```
Mean graylevel value in bounding box:  141.3842746400886
---- Region Number  29 : ----
Area (px):  1641
Bounding Box Area (px):  2304
Mean graylevel value in bounding box:  184.83072916666666
---- Region Number  30 : ----
Area (px):  2212
Bounding Box Area (px):  2950
Mean graylevel value in bounding box:  195.70169491525425
---- Region Number  31 : ----
Area (px):  1677
Bounding Box Area (px):  2205
Mean graylevel value in bounding box:  200.76190476190476
---- Region Number  32 : ----
Area (px):  1300
Bounding Box Area (px):  1860
Mean graylevel value in bounding box:  188.78225806451613
---- Region Number  33 : ----
Area (px):  1316
Bounding Box Area (px):  1776
Mean graylevel value in bounding box:  197.28040540540542
---- Region Number  34 : ----
Area (px):  30
Bounding Box Area (px):  60
Mean graylevel value in bounding box:  195.5
---- Region Number  35 : ----
Area (px):  2148
Bounding Box Area (px):  2968
Mean graylevel value in bounding box:  187.8133423180593
---- Region Number  36 : ----
Area (px):  1472
Bounding Box Area (px):  2068
Mean graylevel value in bounding box:  187.4274661508704
---- Region Number  37 : ----
Area (px):  2790
Bounding Box Area (px):  4958
Mean graylevel value in bounding box:  148.68999596611536
---- Region Number  38 : ----
Area (px):  1119
Bounding Box Area (px):  1680
Mean graylevel value in bounding box:  191.25
---- Region Number  39 : ----
Area (px):  2702
Bounding Box Area (px):  3456
Mean graylevel value in bounding box:  202.76041666666666
---- Region Number  40 : ----
Area (px):  1748
Bounding Box Area (px):  2352
Mean graylevel value in bounding box:  194.06887755102042
---- Region Number  42 : ----
Area (px):  679
Bounding Box Area (px):  814
Mean graylevel value in bounding box:  212.70884520884522
---- Region Number  43 : ----
Area (px):  336
Bounding Box Area (px):  462
Mean graylevel value in bounding box:  199.25324675324674
---- Region Number  44 : ----
Area (px):  467
Bounding Box Area (px):  704
Mean graylevel value in bounding box:  155.390625
---- Region Number  45 : ----
Area (px):  245
Bounding Box Area (px):  341
Mean graylevel value in bounding box:  191.43695014662757
```