1η Εργασία

Ονοματεπώνυμο: Εμμανουηλίδης Κωνσταντίνος

Αριθμός Μητρώου: 57315

Ζητούμενο 1ο

Αρχικά, εισάγουμε τις βιβλιοθήκες που θα χρησιμοποιηθούν στη συγκεκριμένη άσκηση. Η βιβλιοθήκη OpenCV παρέχει χρήσιμες συναρτήση για την επεξεργασία εικόνων. Η βιβλιοθήκη Numpy χρησιμοποιείται για αριθμητικές πράξεις πινάκων, ενώ η βιβλιοθήκη math περιέχει πιο εξειδικευμένες μαθηματικές συναρτήσεις.

```
#Importing libraries
import cv2            #OpenCV
import numpy as np    #Numpy for arithmetic operations with arrays
import math           #Used for mathematical operations
```

Έχοντας εισάγει τις απαραίτητες βιβλιοθήκες, επόμενο βήμα αποτελεί η εισαγωγή/ανάγνωση της εικόνα που θα τεθεί σε επεξεργασία:

```
filename = 'N5.png'
#filename = 'NF5.png'

# Read image
img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
```

Στη συνέχεια, επιθυμούμε να εξάγουμε τον θόρυβο τύπου “Salt and Pepper”. Γι’ αυτό το σκοπό κρίνεται καταλληλότερη η χρήση του φίλτρου ενδιάμεσης τιμής (median filter).

Το φίλτρο ενδιάμεσης τιμής εισάγει σε κάθε pixel της εικόνας τιμή ίση με την ενδιάμεση τιμή από αυτές που περιέχονται στο χρησιμοποιούμενο kernel. Πλεονεκτεί σημαντικά σε σχέση με ένα γραμμικό(blur) ή ένα Gaussian φίλτρο σε εικόνες με θόρυβο τύπου “Salt and Pepper”, μιας και παρουσιάζει ανθεκτικότητα σε τοπικές απότομες μεταβολές(robustness to outliers) και ταυτόχρονα διατηρεί σε μεγάλο βαθμό τις γωνίες της εικόνας. Αντίθετα, ένα Gaussian φίλτρο παίρνοντας το σταθμισμένο μέσο όρο των τιμών των γειτονικών pixels οδηγεί στην εξομάλυνση των γωνιών και δεν ανταποκρίνεται τόσο καλά σε τοπικές απότομες μεταβολές(outliers).

Η υλοποίηση του φίλτρου ενδιάμεσης τιμής, λαμβάνοντας υπόψη τα σημεία που βρίσκονται στο περίγραμμα της εικόνας, φαίνεται παρακάτω:

```python
#Function for median_filtering
def median_filter(img, filter_size):
    temp = []
    indexer = filter_size // 2
    window = [
        (i, j)
        for i in range(-indexer, filter_size-indexer)
        for j in range(-indexer, filter_size-indexer)
    ]
    index = len(window) // 2
    for i in range(len(img)):
        for j in range(len(img[0])):
            img[i][j] = sorted(
                0 if (
                    min(i+a, j+b) < 0
                    or len(img) <= i+a
                    or len(img[0]) <= j+b
                ) else img[i+a][j+b]
                for a, b in window
            )[index]
    return img
```

```python
# Denoising image with median filter
img = median_filter(img,5)
```

Στη συνέχεια, αφού έχουμε περάσει την εικόνα από το παραπάνω φίλτρο τη μετατρέπουμε σε κλίμακα του γκρι και κατόπιν σε δυαδική, ώστε σε κάθε pixel να έχουμε μία από τις δύο διακριτές τιμές(0 ή 255).

Σαν κατώφλι(threshold) για την μετατροπή στην κλίμακα του γκρι έχουμε τη δυνατότητα επιλογής δύο τιμών, του 65 ή του 75.Σε περίπτωση χρησιμοποίησης της τιμής 65 λαμβάνουμε την μικρότερη τιμή κατωφλίου που δεν οδηγεί στη συνένωση κυττάρων που ήταν αρχικά διαφορετικά. Ωστόσο, στο κέντρο της εικόνας υπάρχουν δύο κύτταρα που με χρήση της τιμής 65 παρουσιάζονται ενωμένα και για την αποτελεσματική εύρεση των ζητούμενων τιμών της άσκησης θα χρειαστεί να τα μεταχειριστούμε ξεχωριστά με τα υπόλοιπα. Πιο συγκεκριμένα, αν επιλέξουμε ως τιμή κατωφλίου την τιμή 65 πιθανώς να χρειαστεί να βρούμε όλα τα ζητούμενα για όλα τα κύτταρα εκτός των δυο αυτών, στη συνέχεια να κάνουμε διάβρωση(erosion) μόνο του ενός από τα δύο για να βρούμε τα ζητούμενα για το άλλο και τέλος να επαναλάβουμε τη διαδικασία για το άλλο. Για την αποφυγή όλης αυτής της διαδικασίας μπορεί να χρησιμοποιηθεί ως τιμή κατωφλίου το 75, επιτυγχάνοντας έτσι το διαχωρισμό των δύο κεντρικών κυττάρων και διατηρώντας ταυτόχρονα τα μεγέθη όλων των κυττάρων πολύ κοντά στα αρχικά. Τέλος, η επιλογή μεγαλύτερης τιμής κατωφλίου δεν συνίσταται, καθώς θα είχε ως συνέπεια την αλλοίωση των κυττάρων, ενώ η επιλογή μικρότερης τιμής θα οδηγούσε στη συνένωση διαφορετικών κυττάρων.

Κώδικας μετατροπής σε grayscale image:

```
# Convert to binary image
th, img_th = cv2.threshold(img, 75, 255, cv2.THRESH_BINARY)     #using 75 as threshold
#th, img_th2 = cv2.threshold(img, 65, 255, cv2.THRESH_BINARY);  #using 65 as threshold
```

Στη συνέχεια, πραγματοποιούμε κλείσιμο(“closing”) για να γεμίσουμε με άσπρο τα «κενά» που υπάρχουν στο σώμα ορισμένων κυττάρων. Για τη διαδικασία αυτή χρησιμοποιούμε δομικό στοιχείο(“structuring element”), το μέγεθος του οποίου εξαρτάται από την τιμή κατωφλίου που χρησιμοποιήσαμε νωρίτερα. Σε περίπτωση επιλογής του 65 ως τιμής κατωφλίου χρησιμοποιούμε δομικό στοιχείο μεγέθους 3x3, καθώς έχουμε «κενά» μικρού μεγέθους στην εικόνα τα οποία θέλουμε να «γεμίσουμε». Σε περίπτωση επιλογής του 75 ως τιμής κατωφλίου χρησιμοποιούμε δομικό στοιχείο μεγέθους 5x5, καθώς το 3x3 αφήνει μικρά μαύρα στίγματα μέσα στο άσπρο κύτταρο. Επιπροσθέτως, η χρήση δομικού στοιχείου μεγαλύτερου μεγέθους θα

λάμβανε υπόψη περισσότερα pixels της εικόνας στη διαδικασία closing και θα οδηγούσε στη συνένωση κυττάρων, πράγμα μη επιθυμητό.

```
# Closing image for extracting a small point seemingly like a cell
strel = np.ones((5,5), np.uint8)
img_th = cv2.morphologyEx(img_th, cv2.MORPH_CLOSE, strel)
```

Εν συνεχεία, βρίσκουμε το περίγραμμα κάθε κυττάρου χρησιμοποιώντας τη συνάρτηση findContours() της βιβλιοθήκης numpy:

```
# Finding contours of the cells
img_temp,contours,hierarchy= cv2.findContours(img_th, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Η συνάρτηση findContours() βρίσκει τα περιγράμματα των αντικειμένων που υπάρχουν στην εικόνα και τα επιστρέφει στη μορφή πολυδιάστατου πίνακα. Περισσότερες λεπτομέρειες για τον αλγόριθμο που χρησιμοποιείται στη συγκεκριμένη συνάρτηση μπορούν να βρεθούν στη σχετική βιβλιογραφία στο τέλος της εργασίας.

Με τον ακόλουθο τμήμα κώδικα αφαιρούμε τα κύτταρα που περιλαμβάνουν pixel που βρίσκονται στο περίγραμμα της εικόνας, ενώ ταυτόχρονα διατηρούμε τα υπόλοιπα:

```
#<----------------------Removing the cells that are in the borders of the image---------------------
count = 0
to_be_deleted = []
print("Initial len of contours",len(contours))
for i in range(len(contours)):
    for j in range(len(contours[i])):
        if (contours[i][j][0][0] == 0  ):
                to_be_deleted.append(i)
                break
        if (contours[i][j][0][0] == 806 ):
            to_be_deleted.append(i)
            break
        if (contours[i][j][0][1] == 0):
                    to_be_deleted.append(i)
                    break
        if (contours[i][j][0][1] == 564):
            to_be_deleted.append(i)
            break
```

Έτσι, καταλήγουμε να τυπώσουμε στην οθόνη το μέγεθος του πίνακα contours που αποτελεί τον αριθμό από τα κύτταρα που βρίσκονται στην εικόνα, χωρίς αυτά που είναι στο περίγραμμά της:

```
#Printing number of contours
print("Number of contours", len(contours))
```

Ζητούμενο 2ο

Για τον υπολογισμό του αριθμού των pixels κάθε κυττάρου της εικόνας θα γίνει χρήση της συνάρτησης contourArea() που παρέχεται από τη βιβλιοθήκη OpenCV. Πιο συγκεκριμένα, η παραπάνω συνάρτηση κάνει χρήση του τύπου του Green σε διακριτή μορφή και με χρήση πληροφορίας για τα σύνορα του κάθε αντικειμένου πραγματοποιεί τον υπολογισμό του αριθμού των pixels του. Λόγω της χρήσης του τύπου του Green το αποτέλεσμα της συνάρτησης είναι μη ακέραιος αριθμός, γεγονός που αποκαλύπτει και τη χρήση πληροφορίας για το περίγραμμα και όχι για το εσωτερικό κάθε κυττάρου και οδηγεί στη χρήση της συνάρτησης ceiling(math.ceil()) για την μετατροπή του αποτελέσματος σε ακέραιο.

Η συνάρτηση εφαρμόζεται με όρισμα το περίγραμμα κάθε κυττάρου και στο τέλος τυπώνεται ο πίνακας με τον αριθμό των pixels κάθε κυττάρου:

```
#Measuring area of every cell with contourArea
area = []
for i in range(len(contours)):
    area.append(math.ceil(cv2.contourArea(contours[i])))
print("Number of pixels in each cell",area)
```

Ζητούμενο 3ο

Αρχικά, θα χρειαστεί να υπολογίσουμε το περιβάλλον κουτί(bounding box) κάθε αντικειμένου. Για το σκοπό αυτό χρησιμοποιούμε τη συνάρτηση boundingRect() της βιβλιοθήκης OpenCV.

Η συγκεκριμένη συνάρτηση υπολογίζει τις συνταταγμένες ενός ορθογωνίου παραλληλογράμμου στο οποίο θα περικλειόταν το ζητούμενο κύτταρο, επιστρέφοντας τις συντεταγμένες της πάνω αριστερά γωνίας του ορθογωνίου, το πλάτος και το ύψος του.

Τα παραπάνω στοιχεία αποθηκεύονται στους πίνακες x,y,w,h αντίστοιχα και στη συνέχεια προβάλλουμε στην οθόνη τα ορθογώνια πάνω στην αρχική εικόνα:

```
# Declaration of lists used to save the coordinates of bounding boxes
x = []
y = []
w = []
h = []

# Finding the bounding boxes
for i in range(len(contours)):
    temp1,temp2,temp3,temp4 = cv2.boundingRect(np.asarray(contours[i]))
    x.append(temp1)
    y.append(temp2)
    w.append(temp3)
    h.append(temp4)
    cv2.rectangle(img_th,(x[i],y[i]),(x[i]+w[i],y[i]+h[i]),120,2)

cv2.imshow('calc',img_th)
cv2.waitKey(0)
```

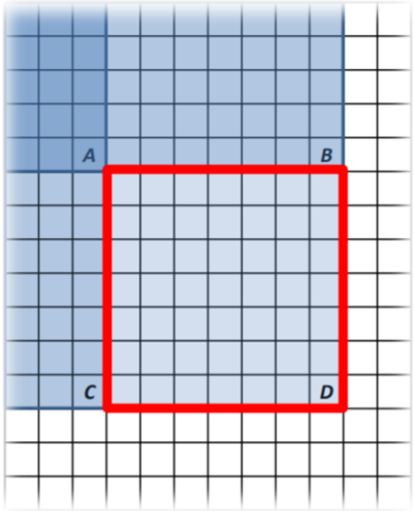
Για τη μέτρηση της μέσης τιμής διαβάθμισης του γκρι των εικονοστοιχείων που περιέχονται στα περιβάλλοντα κουτιά (bounding boxes) των αντικειμένων με τέτοιο τρόπο ώστε η ταχύτητα εκτέλεσης του υπολογισμού να είναι ανεξάρτητη του μεγέθους του αντικειμένου θα χρησιμοποιήσουμε την τεχνική με τα αθροίσματα τετραγώνων.

Ειδικότερα, υπολογίζουμε τα αθροίσματα των pixels μικρότερων τμημάτων της εικόνας και έχοντας τα αποθηκευμένα σε έναν πίνακα(integral image) μπορούμε ανά πάσα στιγμή να υπολογίσουμε το άθροισμα των pixels σε οποιαδήποτε περιοχή της εικόνας ανεξαρτήτως του μεγέθους της. Δημιουργούμε, λοιπόν, τον πίνακα(integral image), κάθε pixel του οποίου είναι το άθροισμα των γειτονικών pixels που βρίσκονται πάνω και αριστερά από αυτό το pixel. Έτσι, υλοποιούμε ουσιαστικά την παρακάτω αναδρομική σχέση:

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

Στην περίπτωση του παρακάτω σχήματος η ζητούμενη περιοχή D υπολογίζεται ως εξής:

SumOf(D) = A + D – B – C



Προγραμματιστικά γίνεται χρήση της συνάρτησης integral() της βιβλιοθήκης OpenCV. Πιο συγκεκριμένα, η παραπάνω συνάρτηση επιστρέφει έναν πίνακα όπου σε κάθε pixel περιέχεται το άθροισμα των γειτονικών pixel του που βρίσκονται άνω και αριστερά του στην αρχική εικόνα.

Στη συνέχεια, με χρήση επανάληψης for δημιουργούμε τους πίνακες A,B,C,D που περιέχουν τις τιμές που θα εισαχθούν στον παραπάνω τύπο για να πάρουμε το ζητούμενο αποτέλεσμα:

```
# Declaration of lists used in computing the mean value of grayscale pixels
A = []
B = []
C = []
D = []
gray_mean_value = []

# Computing the mean value of grayscale pixels
for i in range(len(contours)):
    A.append( integral_list[y[i]][x[i]] )
    D.append(integral_list[y[i]+h[i]][x[i]+w[i]])
    B.append(integral_list[y[i]][x[i]+w[i]])
    C.append(integral_list[ y[i]+h[i]][x[i]])
```

Τέλος, διαιρούμε το υπολογισθέν κάθε φορά άθροισμα με τον αριθμό των pixel του περιβάλλοντος κουτιού(bounding box) για να πάρουμε την μέση τιμή της διαβάθμισης του γκρι των εικονοστοιχείων που περιέχονται στα περιβάλλοντα κουτιά (bounding boxes) και το τυπώνουμε στην οθόνη:

```
    #Division with number of pixels in bounding box
    gray_mean_value.append((A[i] + D[i] - B[i] - C[i]) / (w[i] * h[i]))

# Printing the mean value
print("Mean value of grayscale in each box", gray_mean_value)
```

Σχετική Βιβλιογραφία/Πηγές

1) Διαφάνειες Μαθήματος «Όραση Υπολογιστών», 7ου Εξαμήνου ΗΜΜΥ ΔΠΘ

Καθηγητή Ιωάννη Πρατικάκη

2) A Discrete Version of Green's Theorem, *GREGORY Y. TANG,* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-4, NO. 3, MAY 1982

(http://people.csail.mit.edu/tieu/notebook/imageproc/tang82.pdf)

3) A discrete Green's Theorem

(https://demonstrations.wolfram.com/ADiscreteGreensTheorem/)

4) Median Filtering with Python and OpenCV

(https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1)

5) OpenCV Documentation για τη συνάρτηση integral()

(https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#integral)

6) OpenCV Documentation για τις συναρτήσεις contourArea() και boundingRect()

(https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html)

7) Median Filter Implementation

(https://codereview.stackexchange.com/questions/191974/median-filter-implementation-in-python)