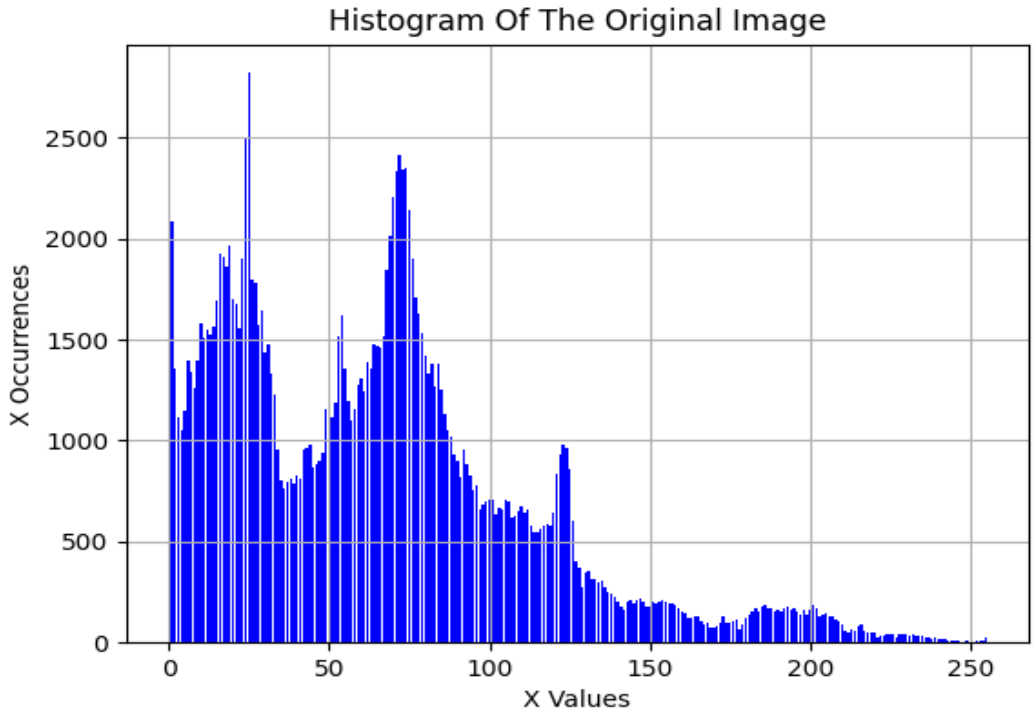
Χρήστος-Αλέξανδρος Δαρδαμπούνης ΑΕΜ : 10335

Εργασία 1: Βελτίωση εικόνων μέσω εξισορρόπησης ιστογράμματος

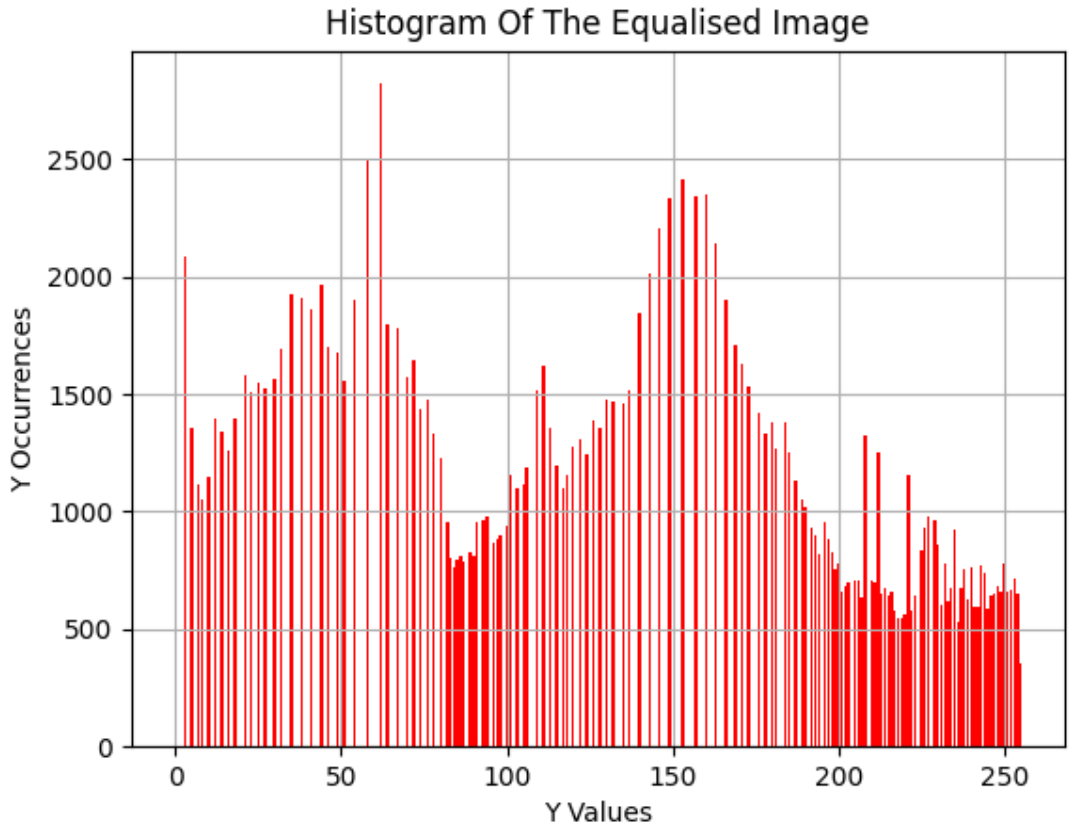
Στο πρώτο ερώτημα της συγκεκριμένης εργασίας πρέπει να κάνουμε ολική εξισορρόπηση ιστογράμματος της εικόνας που μας δίνεται στην εκφώνηση. Η εικόνα μας αρχικά λαμβάνει *L=256* διακριτές τιμές οι οποίες αντιπροσωπεύουν τις τιμές της φωτεινότητας της εικόνας. Σκοπός μας είναι να εξισορροπήσουμε το αρχικό ιστόγραμμα της grayscale εικόνας που μας δίνεται. Αρχικά έχουμε την εξής εικόνα με το παραπάνω ιστόγραμμα :



Ενώ η αρχική εικόνα σε grayscale είναι η εξής :

Αν στην παραπάνω εικόνα εφαρμόσουμε τον μετασχηματισμό εξισορρόπησης ιστογράμματος το αποτέλεσμα που θα πάρουμε θα είναι το εξής :

Ενώ το αντίστοιχο ιστόγραμμα που θα προκύψει από την παραπάνω εξισορρόπηση θα είναι:



Το συγκεκριμένο ιστόγραμμα προέκυψε για όλες τις τιμές της φωτεινότητας της εικόνας, παρόλο που στην εξισορροπημένη εικόνα δεν εμφανίζονται όλες και αυτό εξαιτίας του γεγονός ότι στον

μετασχηματισμό υπάρχει το round το οποίο αντιστοιχεί αρκετές τιμές του x στο ίδιο y. Για αυτό τον λόγο κιόλας παρατηρούμε αρκετούς μηδενισμούς στο ιστόγραμμα.

Όπως παρατηρούμε από την εξισορρόπηση ιστογράμματος όλες οι τιμές πλέον της φωτεινότητας της εικόνας είναι κατανεμημένες ομοιόμορφα, όπως και το περιμέναμε. Επίσης, παρατηρούμε πλέον πως τα σημεία της αρχικής εικόνας που ήταν περισσότερα φωτεινά πως τώρα με την εξισορρόπηση είναι ακόμα περισσότερα φωτεινά και πως γενικά ολόκληρη η εικόνα μας μετά την εξισορρόπηση του ιστογράμματος έχει γίνει πιο φωτεινή γεγονός που πιθανόν οφείλεται στο ότι όλες η πλειονότητα των τιμών της φωτεινότητας των pixels έχουν συγκεντρωθεί γύρω από μια μεγαλύτερη τιμή φωτεινότητας από εκείνη της αρχικής εικόνας. Επίσης, επαληθεύεται τώρα πλέον ότι οι στάθμες της εικόνας εξόδου s είναι ισοπίθανες, ενώ ακόμα παρατηρούμε ότι η εικόνα εξόδου θα έχει μικρότερο πλήθος τιμών από το αντίστοιχο της αρχικής εισόδου που οφείλεται, όπως αναφέρεται και στην εκφώνηση της άσκησης, στην στρογγυλοποίηση καθώς πολλές διαδοχικές στάθμες εισόδου απεικονίζονται στην ίδια στάθμη εξόδου.

Στην συνέχεια φαίνεται ο σχετικός κώδικας σε Python ο οποίος χρησιμοποιήθηκε για την δημιουργία του μετασχηματισμού της εικόνας καθώς επίσης και ο κώδικας ο οποίος πραγματοποιεί την εξισορρόπηση ιστογράμματος σε ολόκληρη την εικόνα :

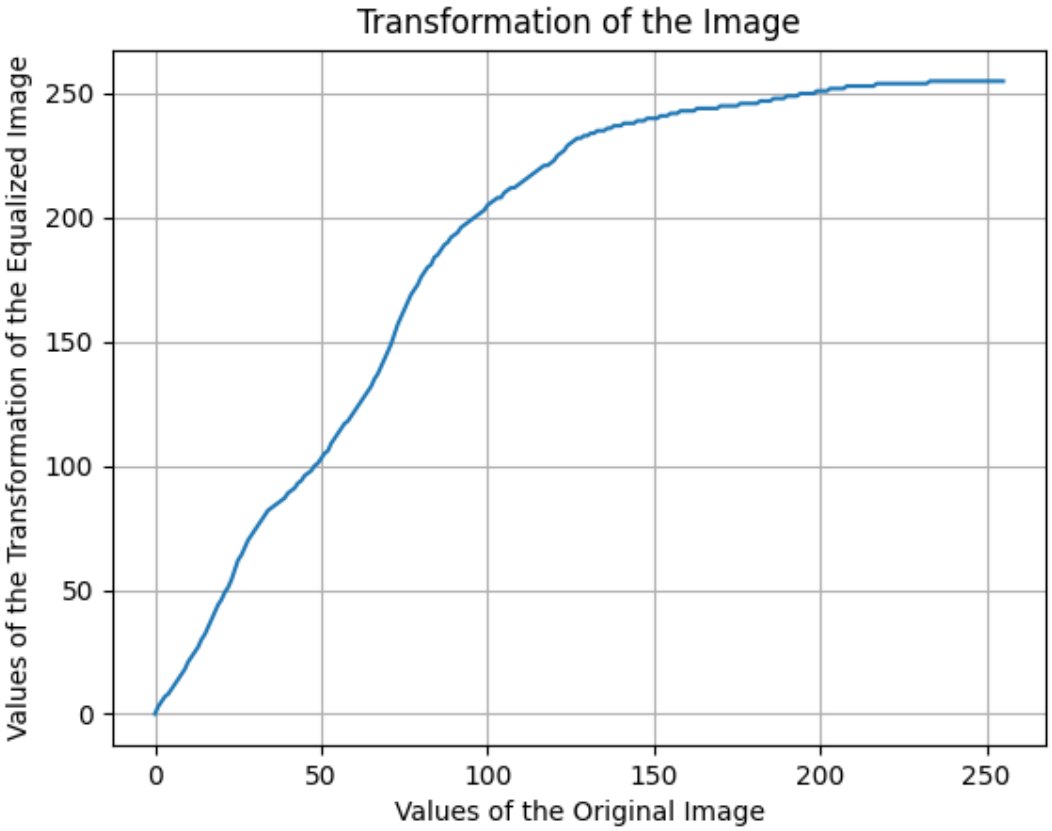
```python
def get_equalization_transform_of_img(img_array):
    appearances = list(element for sublist in img_array for element in sublist) # concatanate the 2D matrix into a single list
    L = 256
    cnt  = [0] * L # list to count the appearances of all the elements in the image
    prob = [0] * L # find the probability of each value prob(x_i)

    for i in appearances:
        cnt[i] += 1 # increment the counter for each appearance of an element

    mySum = sum(cnt) # create the denominator to calculate the cummulative pdf

    prob = list(i/mySum for i in cnt) # calculation of the cummulative pdf
    # creating a list with the probability of an element appearing in the image

    v = [0] * L
    y = [0] * L
    v[0] = prob[0]

    for i in range(1, len(prob)):
        v[i] = v[i-1] + prob[i] # creating the vk vector

    v = np.array(v)
    y = np.round((v-v[0])/(1-v[0])*(L-1)) # creating the transformation that is going to be applied to the image

    equalization_transform = np.array(y, dtype=np.uint8) # changing its type to the desired return type as mentioned in the project

    return equalization_transform
```

Αρχικά βρίσκω όλα τα στοιχεία του δισδιάστατου πίνακα της εικόνας και τα βάζω όλα σε μια λίστα . Μετά με την λίστα cnt βρίσκω το πλήθος των εμφανίσεων του κάθε στοιχείου και με αυτό τον τρόπου σχηματίζω την αθροιστική συνάρτηση πιθανότητας και στην συνέχεια το διάνυσμα u_k με το οποίο βρίσκω τον μετασχηματισμό της εικόνας .

```python
def perform_global_hist_equalization(img_array):

    T = get_equalization_transform_of_img(img_array) # get the global transformation of the image
    s = T[img_array] # apply the transformation to the img_array data s = T[r]
    # array slicing, numpy performs the mapping of the transformation
    equalized_img = np.array(s, dtype=np.uint8)
    return equalized_img
```

Αφού βρω τον ολικό μετασχηματισμό της εικόνας, τον εφαρμόζω στα pixel της εικόνας (χρησιμοποιώντας το array slicing και άρα εφαρμόζοντας τον κατάλληλο μετασχηματισμό στο αντίστοιχο pixel) και στην συνέχεια επιστρέφω την εξισορροπημένη εικόνα.

Τέλος ο μετασχηματισμός που προκύπτει για την εξισορρόπηση του ιστογράμματος της συνολικής εικόνας είναι ο εξής :



Από το παραπάνω διάγραμμα επιβεβαιώνεται και πάλι η παρατήρησή μας, ότι δηλαδή η εικόνα εξόδου θα έχει μικρότερο πλήθος τιμών από το αντίστοιχο της αρχικής εισόδου που οφείλεται, όπως αναφέρεται και στην εκφώνηση της άσκησης, στην στρογγυλοποίηση καθώς πολλές διαδοχικές στάθμες εισόδου απεικονίζονται στην ίδια στάθμη εξόδου.

Adaptive Histogram Equalization

Όλα τα παραπάνω αφορούσαν την περίπτωση στην οποία εφαρμόσαμε ολική εξισορρόπηση ιστογράμματος. Από την παραπάνω διαδικασία παρατηρήσαμε ότι η τελική εικόνα, με την εφαρμογή του μετασχηματισμού εξισορρόπησης σε όλη της την έκταση, γίνεται πιο φωτεινή και αυτό ίσως καθιστά σε ορισμένες περιπτώσεις δύσκολο τον διαχωρισμό των αντικειμένων που βρίσκονται στην εικόνα. Για τον λόγο αυτό επιλέγουμε να εφαρμόσουμε πλέον τον μετασχηματισμό εξισορρόπησης, όχι σε ολόκληρη την εικόνα αλλά σε contextual regions τα οποία έχουν διαστάσεις 64x48 το καθένα όπως αναφέρεται στην εκφώνηση της άσκησης. Ωστόσο, επειδή αυτή η επιλογή των διαστάσεων των contextual regions δεν διαιρεί επακριβώς την εικόνα, επιλέγω να αφαιρέσω κάποιο κομμάτι της εικόνας προκειμένου να μπορώ να εφαρμόσω την μέθοδο για τις διαστάσεις των contextual regions που μου ζητούνται. Συνεπώς, το αποτέλεσμα αυτής της επιλογής θα είναι να

δημιουργηθεί μια εικόνα η οποία πλέον θα αποτελείται από 49 contextual regions με διαστάσεις 64x48 το καθένα ενώ πλέον η εικόνα μας θα έχει διαστάσεις 448x336. Η διαδικασία αυτή με την οποία βρίσκω τον μετασχηματισμό του κάθε contextual region γίνεται με την ζητούμενη συνάρτηση

```python
def calculate_eq_transformations_of_regions(img_array, region_len_h, region_len_w):


    dim = img_array.shape # get the dimensions of the image array
    rows, cols = dim

    heightPart = cols // region_len_h # integer division to get the number of vertical regions
    widthPart =  rows // region_len_w # integer division to get the number of horizontal regions
    regionHeight = region_len_h
    regionWidth = region_len_w
    regions = []

    for i in range(heightPart):
        for j in range(widthPart):
            wStart = i * region_len_w
            hStart = j * region_len_h
            regions.append((wStart, hStart)) # creating the tuple that is going to be the key for our dictionary
            # I am firstly iterating over the horizontal axis because Python numpy arrays and Images have the rows and
            # columns switched


myDictionary ={} # initializing the Dictionary


for i in range(len(regions)):
    A = img_array[regions[i][0] : regionWidth, regions[i][1] : regionHeight] # getting a region from the whole image
    regionHeight += region_len_h # this is to increment the step for the vertical axis
    # first I create the row regions and then I move on to the next column

    if(regions[i][1] == region_len_h * (heightPart - 1)):
        regionWidth += region_len_w
        regionHeight = region_len_h

    myDictionary[(regions[i][1], regions[i][0])] = np.array(get_equalization_transform_of_img(A), dtype = np.uint8)
    # adding the transformation of every region of the original Image to the dictionary
    # with it's corresponding key

return myDictionary
```

Στην συνέχεια, στο αρχείο demo.py καλώ την συγκεκριμένη συνάρτηση και αφότου λάβω το συγκεκριμένο λεξικό προσπαθώ έχοντας τον μετασχηματισμό του κάθε contextual region να τον εφαρμόσω και να «κολλήσω» τα contextual regions δημιουργώντας εν τέλει την συνολική εικόνα η οποία πλέον αποτελείται από 49 contextual regions που το καθένα έχει τον δικό του μετασχηματισμό. Παρακάτω φαίνεται λίγο ο κώδικας με τον οποίο κάνω την διαδικασία που περιέγραψα ενώ ακόμα παραθέτω επίσης και την εικόνα που θα προκύψει καθώς και το αντίστοιχο ιστόγραμμα της :

Ο κώδικας για με τον οποίο κάνω την προαναφερθείσα διαδικασία στο demo.py είναι ο εξής :

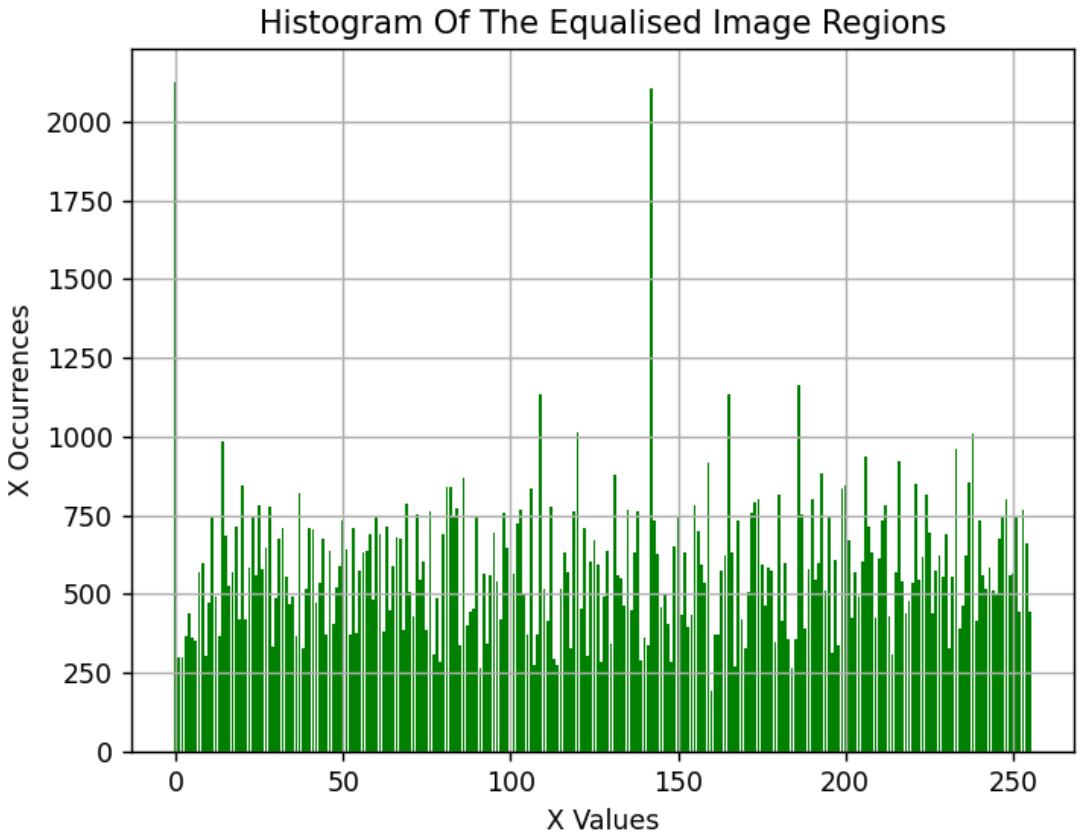
```python
region_len_h = 64
region_len_w = 48
Dict = calculate_eq_transformations_of_regions(img_array, region_len_h, region_len_w)
newDict = {}
regionHeight = 64
regionWidth = 48

rows, cols = img_array.shape # rows, cols


regions = []
heightPart = cols // regionHeight
widthPart = rows // regionWidth



for i in range(heightPart):
    for j in range(widthPart):
        hStart = i * regionWidth
        wStart = j * regionHeight
        regions.append((hStart, wStart))
```

```
for i in range(len(regions)):
    B = img_array[regions[i][0] : regionWidth, regions[i][1] : regionHeight]
    regionHeight += region_len_h

    if (regions[i][1] == (heightPart - 1) * region_len_h):
        regionWidth += region_len_w
        regionHeight = region_len_h

    newDict[(regions[i][1], regions[i][0])] = np.array(Dict[(regions[i][1], regions[i][0])][B], dtype = np.uint8)

smallArray = [value for value in newDict.values()] # create a list of 49 arrays with dimensions 48x64
rows = [np.concatenate(smallArray[i*7:(i+1)*7], axis = 1) for i in range(widthPart)] # create a rows list of 7 arrays per element
image = np.concatenate(rows, axis=0) # concatenate all the arrays to create the final image
```

Στην συνέχεια φαίνεται η σχετική εικόνα που παράγεται από την εφαρμογή του μετασχηματισμού σε κάθε contextual region και την ένωση τους :

Από την συγκεκριμένη εικόνα μπορούμε να παρατηρήσουμε αρκετές διαφορές σε σχέση με την εικόνα που προκύπτει από την ολική εξισορρόπηση ιστογράμματος. Αρχικά, αυτό που παρατηρούμε είναι ότι αυτή η εικόνα δεν είναι τόσο φωτεινή όσο εκείνη της ολικής εξισορρόπησης και τα αντικείμενα είναι εύκολα διακριτά. Αυτό οφείλεται στο γεγονός ότι σε κάθε contextual region εφαρμόζουμε τον δικό του μετασχηματισμό και όχι έναν global ο οποίος δεν εξισώνει όλες τις τιμές της φωτεινότητας σε μια τιμή. Επομένως είναι εύκολο να διακρίνουμε πλέον τα αντικείμενα της εικόνας. Ωστόσο, η δεύτερη διαφορά που παρατηρούμε είναι ότι μπορούμε να διακρίνουμε τα contextual regions στην μετασχηματισμένη εικόνα και αυτό οφείλεται στο γεγονός ότι εφαρμόζουμε στο καθένα ξεχωριστά τον δικό του μετασχηματισμό οπότε στο σημείο που αλλάζει το contextual region αλλάζει η φωτεινότητα του κάθε region και για αυτό είναι ευδιάκριτος ο διαχωρισμός.

Τέλος, παρουσιάζω το σχετικό ιστόγραμμα της εικόνας :

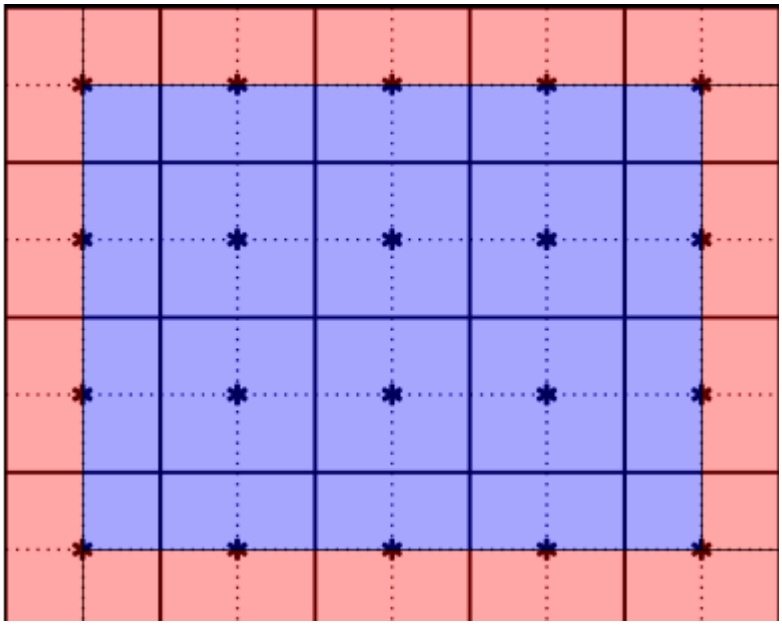


Στην παραπάνω εικόνα φαίνεται το ιστόγραμμα της εικόνας που προκύπτει από την ένωση των μετασχηματισμένων contextual regions από το οποίο μπορούμε να παρατηρήσουμε πως οι τιμές της φωτεινότητας είναι περισσότερο ομοιόμορφα κατανεμημένες σε σχέση με την ολική εξισορρόπηση ιστογράμματος. Εν τέλει, παρατηρούμε πως τελικά με αυτό τον τρόπο έχουμε πετύχει μια καλύτερη αναπαράσταση της εικόνας καθότι πλέον μπορούμε να διακρίνουμε όλα τα αντικείμενα χωρίς κάποια να φαίνονται ιδιαίτερα φωτεινά όπως στην περίπτωση της ολικής εξισορρόπησης. Ωστόσο, όπως αναφέρθηκε και προηγουμένως, στην εικόνα με τον χωρισμό σε contextual regions είναι εμφανής ο χωρισμός της κάθε περιοχής και αυτό διότι σε κάθε region εφαρμόσαμε και διαφορετικό μετασχηματισμό οπότε υπάρχει μια ασυνέχεια της τιμής της φωτεινότητας στα όρια του κάθε contextual region.

Για να αντιμετωπίσουμε αυτό το πρόβλημα πραγματοποιούμε εξισορρόπηση του κάθε pixel με βάση τις κατανομές που το περιβάλλουν. Αυτό γίνεται εντοπίζοντας τα 4 κοντινότερα contextual κέντρα που περιβάλλον το pixel. Αφού γίνει αυτό , και αφού προηγουμένως έχουν υπολογιστεί οι μετασχηματισμοί εξισορρόπησης του κάθε contextual region τότε η τιμή του pixel x απεικονίζεται στο y μέσω του μετασχηματισμού $y = (1-a)(1-b)T_{-,-}(x) + (1-a)bT_{+,-}(x) + a(1-b)T_{-,+}(x) + abT_{+,+}(x)$ όπου τα a, b αντιπροσωπεύουν τα διάφορα βάρη για την κάθε περιοχή δίνοντας της μεγαλύτερη ή μικρότερη σημασία. Εφαρμόζοντας αυτό τον μετασχηματισμό στα εσωτερικά σημεία της μπλε περιοχής θα γλιτώσουμε τις ασυνέχειες που εμφανίζονταν προηγουμένως όπου εφαρμόζαμε μόνο τον μετασχηματισμό σε κάθε contextual region ξεχωριστά.

Χρήστος-Αλέξανδρος Δαρδαμπούνης ΑΕΜ : 10335

Στα εξωτερικά σημεία της κόκκινης περιοχής ο μετασχηματισμός εξισορρόπησης που θα χρησιμοποιείται θα είναι του δικού του region χωρίς την χρήση παρεμβολής



Παρακάτω φαίνεται ο σχετικός κώδικας με τον οποίο υπολογίζω τα σημεία που θα βρίσκονται στο εσωτερικά της μπλε περιοχής και ποια θα βρίσκονται στα εξωτερικά της κόκκινης περιοχής. Σαν είσοδο για πίνακα εικόνας στην παρακάτω συνάρτηση δίνω την περικομμένη εικόνα διαστάσεων 448x336 έτσι ώστε να διαιρείται ακριβώς σε 49 contextual regions. Αυτή η παραδοχή δεν θα επηρεάσει το τελικό μας αποτέλεσμα καθότι η διγραμμική παρεμβολή θα εφαρμοστεί στο εσωτερικό της εικόνας και όχι στα εξωτερικά σημεία τα οποία θα κοπούν.

```
def perform_adaptive_hist_equalization(img_array, region_len_h, region_len_w):


    Dict = calculate_eq_transformations_of_regions(img_array, region_len_h, region_len_w)
    # receiving the dictionary created from the previous function

    dim = img_array.shape

    rows, cols = dim # rows = 360, cols = 480
    regions = list(Dict.keys()) # transforming the dictionary keys in to a list of tuples
    # which we are going to use later to create a dictionary
    regionHeight = region_len_h
    regionWidth = region_len_w
    heightPart = cols // region_len_h
    widthPart =  rows // region_len_w
    centers = {} # initializing the dictionary of the centers of each region
    k = 0
```

Αρχικά καλώ την συνάρτηση με την οποία έφτιαξα το dictionary που περιέχει τα contextual regions για να πάρω τα regions ενώ επίσης μέσα σε αυτή την συνάρτηση φτιάχνω ξανά το πλήθος των contextual regions. Επίσης αρχικοποιώ και ένα ακόμα dictionary το οποίο θα περιέχει τα κέντρα των contextual regions τα οποία θα μου χρειαστούν για την εφαρμογή της διγραμμικής παρεμβολής για τα εσωτερικά σημεία

```
    for i in range(heightPart): # I am iterating as many times as the numbers of region of row
        for j in range(widthPart):
            centers[regions[k]] = ((regionWidth - 1) / 2, (regionHeight - 1) / 2)
            # Here I create a dictionary which has as keys a tuple corresponding to the region which center we want to find
            # the value of each region is the center of each region which is found by dividing each coordinate in half
            regionHeight += 2*region_len_h
            k += 1
        regionHeight = region_len_h
        regionWidth += 2*region_len_w


    newDict = {}
    regionHeight = region_len_h
    regionWidth = region_len_w

    for key, value in centers.items():
        newDict[value] = Dict[key] # create a dictionary where every center is matched with the corresponding transformation

    centers = list(centers.values()) # create a list of tuples with the centers of each region
```

Χρήστος-Αλέξανδρος Δαρδαμπούνης ΑΕΜ : 10335

Στην συνέχεια μέσα σε μια διπλή for φτιάχνω ένα dictionary το οποίο θα έχει ως κλειδιά tuples που θα αντιπροσωπεύουν το κάθε contextual region ενώ το value κάθε κλειδιού θα είναι πλέον και αυτό ένα tuple μέσα στο οποίο θα περιέχονται οι συντεταγμένες του κέντρου του κάθε region. Στην συνέχεια φτιάχνω ένα ακόμα region μέσα από το οποίο ορίζω ότι τα κλειδιά του θα είναι τα κέντρα των regions και οι τιμές που θα αντιστοιχούν στα κλειδιά αυτά θα είναι ο αντίστοιχος μετασχηματισμός του κάθε τετραγώνου. Αυτό, θα μου χρειαστεί αργότερα καθότι μέσω αυτού, έχοντας εντοπίσει τα τέσσερα κοντινότερα κέντρα θα μπορώ να εφαρμόσω τον αντίστοιχο μετασχηματισμό

```
arr1 = [] # creating an empty list in which I am going to store the centers of the smaller dimension of the Image
arr2 = [] # creating an empty list in which I am going to store the centers of the greater dimension of the Image

for i in range(len(centers)):
    arr1.append(centers[i][0]) # Here I just the centers of each region to the above empty initialized lists
    arr2.append(centers[i][1])


arr1 = sorted(list(set(arr1))) # Since the number of the centers is going to appear multiple times
# I convert them to set's and then back to lists to get only one instance of each center and then I sort
# the list because it's going to be useful for the next step where I identify which pixels belong to the regions
# that are going to interpolated and which are not
arr2 = sorted(list(set(arr2)))


points = [] # I create an empty list of points which will represent which points have been interpolated
```

Στην συνέχεια φτιάχνω 2 πίνακες στους οποίους βάζω από μια φορά τα κέντρα με αύξουσα σειρά ενώ επίσης φτιάχνω και μια λίστα points μέσα στην οποία θα βάλω όλα τα σημεία τα οποία βρίσκονται στην εσωτερική μπλε περιοχή έτσι ώστε να μπορώ να τα ξεχωρίσω από τα outer points

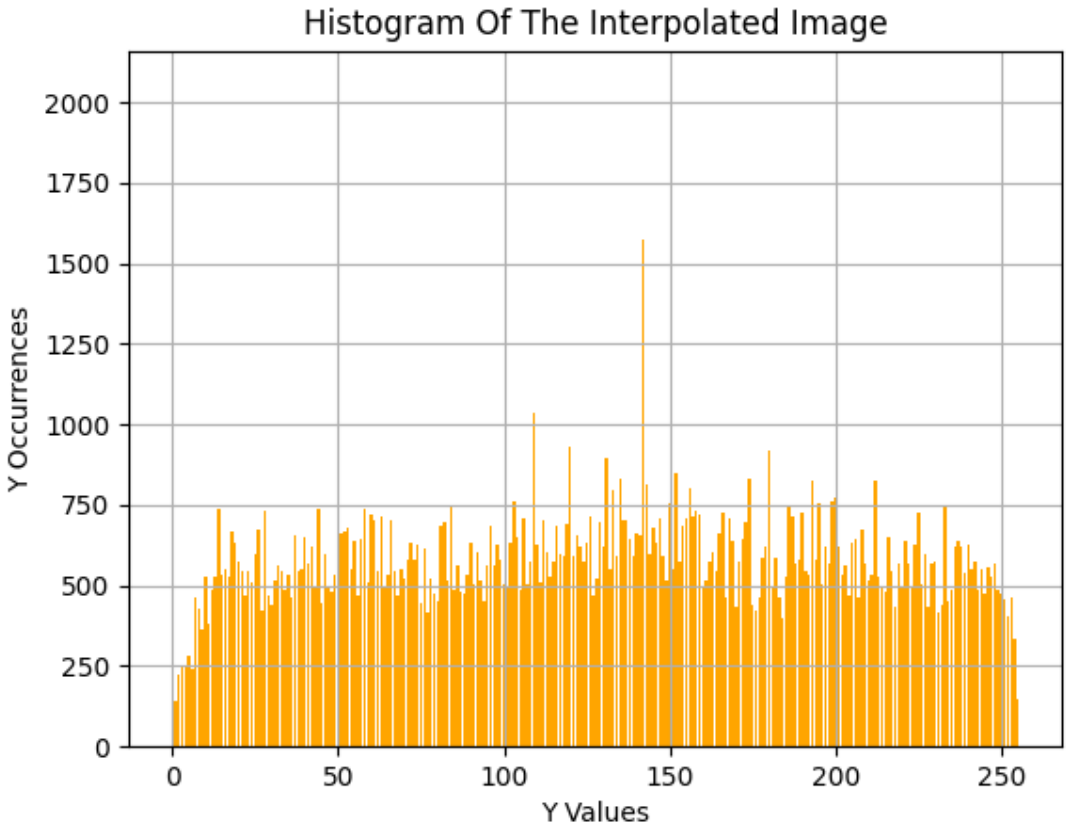
```
for i in range(rows):
    for j in range(cols):
        # I take every pixel of the picture and try to identify in which region it belongs
        for k in range(len(arr1) - 1):
            if i > arr1[k] and i < arr1[k+1]:
                # with this if statement I check if it belongs between two centers in the width dimension
                for m in range(len(arr2) - 1):
                    # if it does belong between two centers in the width dimension then I check if it belongs between
                    # any centers in the height dimension
                    if j > arr2[m] and j < arr2[m+1]:
                        # if it does indeed belong then I perform the interpolation
                        T1 = (arr1[k], arr2[m]) # the transformation of the upper left corner
                        T2 = (arr1[k], arr2[m+1]) # the transformation of the upper right corner
                        T3 = (arr1[k+1], arr2[m]) # the transformation of the lower left corner
                        T4 = (arr1[k+1], arr2[m+1]) # the transformation of the lower right corner
                        a = (i - arr1[k]) / (arr1[k+1] - arr1[k])
                        b = (j - arr2[m]) / (arr2[m+1] - arr2[m])
                        pixel = img_array[i][j]
                        img_array[i][j] = (1-a) * (1-b) * newDict[T1][pixel] + (1-a)*b*newDict[T2][pixel] + a*(1-b)*newDict[T3][pixel] + a*b*newDict[T4][pixel]
                        points.append((i, j)) # after I perform the interpolation I add the pixel to the points list
```

Στην συνέχεια μέσα σε μια διπλή for αφού τρέξω για όλα τα pixels της εικόνας, βρίσκω ποια από αυτά ανήκουν στο εσωτερικό της συγκρίνοντας αρχικά αν είναι μεταξύ δύο κέντρων στην διάσταση του πλάτους και στην συνέχεια άμα ισχύει αυτό τότε ελέγχω αν βρίσκεται το pixel και μεταξύ δύο κέντρων στην διάσταση του ύψους. Άμα ισχύουν ταυτόχρονα αυτές οι συνθήκες τότε το pixel είναι σίγουρα εσωτερικό και επομένως εφαρμόζω για αυτό την διγραμμική παρεμβολή. Επιπλέον, αφού είναι εσωτερικό σημείο το προσθέτω επίσης στην λίστα με τα εσωτερικά σημεία

```
for i in range(rows):
    for j in range(cols):
        # I try every of the picture and try to identify in which region it belongs
        if (i,j) in points:
            continue # if the pixel belongs in the interpolated pixels then I don't do anything and I continue the loop
        else:
            # if the picture doesn't belong in the interior interpolated pixels then it is an exterior point
            # and for that reason I have to define in which region it belongs so I can apply the proper transformation
            res1 = [result - i for result in arr1] # I create a list where every element represents the distance of the
            # pixel from each center in the width dimension
            res2 = [result - j for result in arr2] # I create a list where every element represents the distance of the
            # pixel from each center in the height dimension
            res1 = min(enumerate(res1), key=lambda x: abs(x[1]-0))
            # the above line of code is finding the index and the value in the res1 list which is closer to zero
            # where the lamda function takes the key pair (index, value) and returns absolute distance from zero
            res1 = res1[0] # with this way I get the proper index to define in which region my pixel belongs
            res2 = min(enumerate(res2), key=lambda x: abs(x[1]-0))
            res2 = res2[0]
            img_array[i][j] = newDict[(arr1[res1], arr2[res2])][img_array[i][j]]
            # after that I apply the proper transformation to every exterior pixel

return img_array
```

Τέλος, ξαναπερνάω όλα τα pixels της εικόνας και άμα κάποιο ανήκει στην λίστα με τα εσωτερικά, απλά συνεχίζω στο επόμενο, αλλιώς βρίσκω αρχικά την μικρότερη κατά απόλυτη τιμή απόσταση του από το κέντρο στην διάσταση του πλάτος αλλά και του ύψους έτσι ώστε να προσδιορίσω το exterior contextual region στο οποίο ανήκει το pixel και στην συνέχεια να εφαρμόσω τον απλό μετασχηματισμό του αντίστοιχου region. Ωστόσο, αν και ο τρόπος με τον οποίο το κάνω θα δουλέψει και θα δώσει τα σωστά αποτελέσματα, δεν είναι αρκετά χρονικά αποδοτικός, καθότι κάθε φορά ελέγχω άμα κάποιο σημείο βρίσκεται στα εσωτερικά, γεγονός που έχει σαν αποτέλεσμα να απαιτείται **πολύς χρόνος προκειμένου να ολοκληρωθεί η κλήση της συνάρτησης.** Παρακάτω, φαίνεται η εικόνα που προκύπτει μετά την εφαρμογή του adaptive histogram equalization :

Από την παραπάνω εικόνα βλέπουμε πως έχει βελτιωθεί σημαντικά η ποιότητα της και πως πλέον είναι πιο ευδιάκριτα όλα τα αντικείμενα που βρίσκονται μέσα σε αυτήν ενώ επίσης πλέον δεν είναι τόσο έντονες οι ασυνέχειες μεταξύ των contextual regions και πως στην περιοχή όπου εφαρμόσαμε την διγραμμική παρεμβολή δεν υπάρχουν καθόλου ασυνέχειες . Συνεπώς, με αυτό τον τρόπο κατορθώσαμε να βελτιώσουμε σημαντικά την ποιότητα της εικόνας , χωρίς να είναι ορισμένα σημεία πολύ φωτεινά και χωρίς να έχουμε ασυνέχειες μεταξύ των contextual regions. Τέλος παραθέτω και το σχετικό ιστόγραμμα της παραπάνω εικόνας :



Από το παραπάνω ιστόγραμμα παρατηρούμε πλέον πως οι τιμές της φωτεινότητας είναι πολύ καλύτερα ίσο-κατανεμημένες και πως το ιστόγραμμα είναι περισσότερο ισορροπημένο. Συνεπώς καταλήγουμε στο συμπέρασμα πως ο καλύτερος τρόπος για την εξισορρόπηση ιστογράμματος είναι ο adaptive καθότι μας δίνει πιο εξισορροπημένο ιστόγραμμα ενώ επίσης η τελική εικόνα που παράγει σαν αποτέλεσμα είναι πιο ευδιάκριτη και με πολύ λιγότερες ατέλειες από τις προηγούμενες.

(ΣΗΜΕΙΩΣΗ : ΓΙΑ ΝΑ ΤΡΕΞΕΙ Ο ΚΩΔΙΚΑΣ ΠΡΕΠΕΙ ΝΑ ΒΑΛΕΤΕ ΤΟ FILE PATH ΟΠΟΥ ΒΡΙΣΚΕΤΑΙ Η ΕΙΚΟΝΑ ΩΣΤΕ ΝΑ ΜΠΟΡΕΣΕΙ ΝΑ ΤΥΠΩΣΕΙ ΤΑ ΓΡΑΦΗΜΑΤΑ ΚΑΙ ΤΙΣ ΕΙΚΟΝΕΣ ΠΟΥ ΖΗΤΟΥΝΤΑΙ ΣΤΗΝ ΕΚΦΩΝΗΣΗ)