

**Εργασία Συστήματα Πολυμέσων 2018-2019**

Μέλη ομάδας :

Π16097 ΔΙΟΝΥΣΗΣ ΝΙΚΑΣ,

Π16195 ΝΗΣΙΩΤΗΣ ΜΑΡΙΝΟΣ

Περιεχόμενα

[Περιεχόμενα 2](#_Toc11602691)

[Άσκηση 6.16 4](#_Toc11602692)

[1.1 Εκφώνηση άσκησης 4](#_Toc11602693)

[1.2 Περιγραφή συστήματος/ψευδοκώδικας 5](#_Toc11602694)

[1.3 Υλοποίηση 7](#_Toc11602695)

[1.4 Αποτελέσματα 9](#_Toc11602696)

[1.5 Συμπεράσματα 10](#_Toc11602697)

[Άσκηση 6.17 11](#_Toc11602698)

[2.1 Εκφώνηση άσκησης 11](#_Toc11602699)

[2.2 Περιγραφή συστήματος/ψευδοκώδικας 12](#_Toc11602700)

[2.3 Υλοποίηση 13](#_Toc11602701)

[2.4 Αποτελέσματα 14](#_Toc11602702)

[2.5 Συμπεράσματα και απαντήσεις 14](#_Toc11602703)

[Άσκηση 8.17 16](#_Toc11602704)

[3.1 Εκφώνηση άσκησης 16](#_Toc11602705)

[3.2 Περιγραφή συστήματος/ψευδοκώδικας 18](#_Toc11602706)

[3.3 Υλοποίηση 23](#_Toc11602707)

[3.4 Αποτελέσματα 24](#_Toc11602708)

[3.5 Συμπεράσματα 25](#_Toc11602709)

[Άσκηση 8.18 27](#_Toc11602710)

[4.1 Εκφώνηση άσκησης 27](#_Toc11602711)

[4.2 Περιγραφή συστήματος/ψευδοκώδικας 28](#_Toc11602712)

[4.3 Υλοποίηση 31](#_Toc11602713)

[4.4 Αποτελέσματα 33](#_Toc11602714)

[4.5 Συμπεράσματα 34](#_Toc11602715)

[5 Βιβλιογραφία και βιβλιοθήκες που χρησιμοποιήθηκαν 35](#_Toc11602716)

[5.2 Βιβλιοθήκες Python 35](#_Toc11602717)

[5.3 Βιβλιογραφία 37](#_Toc11602718)

[5.4 ΒΙΒΛΙΟΓΡΑΦΙΑ ΔΙΑΔΙΚΤΥΟΥ 37](#_Toc11602719)

# Άσκηση 6.16

## Εκφώνηση άσκησης

*«Σε αυτήν την προγραμματιστική άσκηση, θα υλοποιήσετε έναν απλό κωδικοποιητή μήκους διαδρομής, ο οποίος επεξεργάζεται δεδομένα εικόνας ή οποιονδήποτε δισδιάστατο πίνακα τιμών, κβαντίζοντας αρχικά τις τιμές εισόδου και στη συνέχεια εφαρμόζοντας σε αυτές κωδικοποίηση μήκους διαδρομής. Η κβάντιση θα τονίσει το φαινόμενο της επανάληψης τιμών. Το πρόγραμμά σας πρέπει να δέχεται ως είσοδο μια εικόνα αποχρώσεων του γκρι (ή μια έγχρωμη εικονα RGB) και μια τιμή που θα ρυθμίζει τις τιμές επιπέδων κβάντισης.»*

## Περιγραφή συστήματος/ψευδοκώδικας

Για την υλοποίηση και την επίλυση της άσκησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python 3.

Για να μπορεί να γίνει η εκτέλεση του προγράμματος θα πρέπει να υπάρχουν εγκατεστημένες στο σύστημα οι εξής βιβλιοθήκες: PIL & numpy.

**Ψευδοκώδικας**

**Συνάρτηση κβάντισης**(τιμή\_κβάντισης, πίνακας\_με\_τα\_pixels):

Για 1 μέχρι το ύψος της εικόνας:

*Πίνακας\_κβάντισης* = διαίρεση των pixels με την τιμή\_κβάντισης

Εμφάνιση του *πίνακα\_κβάντισης*

Εμφάνιση της επεξεργασμένης εικόνας

*Επιστροφή του πίνακα\_κβάντισης*

Συνάρτηση κωδικοποίησης(πίνακας\_κβάντισης):

*Κωδικοποίσηση* σε string τον πίνακα\_κβάντισης

Όσο i < ύψος της εικόνας:

| Αν το i ==0 τότε:

| Προηγούμενο = προηγούμενο\_pixel\_της στήλης[i, j-1]

| Αλλιώς :

| Προηγούμενο = προηγούμενο\_pixel\_της\_γραμμής [i-1, j]

| Όσο j <πλάτος της εικόνας:

| | Παρόν = πίνακας\_κβάντισης [i, j]

| | Αν το παρόν == προηγούμενο:

| | *Κωδικοποιήση* των στοιχείων σε string

| | προηγούμενο=παρόν

| | Πήγαινε στο επόμενο στοιχείο κατά πλάτος (στήλη)

| Πήγαινε στο επόμενο στοιχείο κατά ύψος (γραμμή)

*Επιστροφή κωδικοποίηση*

Συνάρτηση αποκωδικοποίησης (αποτέλεσμα\_κωδικοποίησης):

τέλος\_γραμμής = ψευδής

*αποκωδικοποίηση* = πίνακας\_μηδενικών (ύψος, πλάτος)

κωδικοποίηση\_πίνακα = αποτέλεσμα\_κωδικοποίησης(διαχωρισμός ‘|’)

παρον = 0

προηγούμενη\_στήλη = 0

Για i μέχρι κωδικοποίηση\_πίνακα:

| i = διαχωρισμός (‘-’)

| εύρος\_χρωμάτων = i[1] διαχωρισμός (‘;’)

| στήλη\_εκίννησης = int (εύρος\_χρωμάτων[0]) #έλεγχος από που ξεκινάει η επόμενη στήλη

| γραμμή\_εκκίνησης = int(εύρος\_χρωμάτων [1]) #έλεγχος από που ξεκινάει η επόμενη γραμμή

| τιμές\_χρώματος = i[0] διαχωρισμός(‘;’)

| #έλεγχος αν υπάρχουν στοιχεία στην επόμενη στήλη

| Αν παρόν <= πλάτος & στήλη> προηγούμενη\_στήλη:

| | στήλη = προηγούμενη\_στήλη

| | γραμμή\_εκκίνησης = πλάτος

| | τέλος\_γραμμής = αληθής

|

| Όσο παρόν < γραμμή\_εκκίνησης:

| | *αποκωδικοποιήση* [στήλης][παρόν] = πίνακα(τιμές\_χρώματος)

| | Αν τέλος\_γραμμής & εύρος\_χρωμάτων[1] != 0 & και παρόν ==πλάτος -1

| | | παρόν = 0

| | | γραμμή\_εκκίνησης = int(εύρος\_χρωμάτων[1])

| | | στήλη = int([0])

| | | τέλος\_γραμμής = ψευδής

| | αλλιώς:

| | | παρόν +=1

| | προηγούμενη\_στήλη = στήλη

*Επιστροφή αποκωδικοποίηση*

Συνάρτηση (από)-κβάντισης(τιμή\_κβάντιστης, αποτέλεσμα\_αποκωδικοποίησης):

Για 1 μέχρι το ύψος της εικόνας:

*Πίνακας\_(από)-κβάντισης* = πολλαπλασιασμός των pixels με την τιμή\_κβάντισης

εικόνα = δημιουργία\_εικόνας από τον *πίνακας\_(από)-κβάντισης*

Εμφάνιση εικόνας

*Επιστροφή πίνακας\_(από)-κβάντισης*

**Κυρίως Πρόγραμμα**

Φόρτωση εικόνας

Ύψος και πλάτος εικόνας

Πίνακας\_με\_τα\_pixels

Εμφάνιση της εικόνας πρωτού την επεξεργασία

Πίνακας\_κβάντισης = συνάρτηση κβάντισης(τιμή κβάντισης, πίνακας\_με\_τα \_pixels)

Αποτέλεμσα\_κωδικοποίησης = συνάρτηση κωδικοποίησης(πίνακας\_κβάντισης)

Αποτέλεσμα\_αποκωδικοποίησης = συνάρτηση αποκωδικοποίησης (αποτέλεσμα\_κωδικοποίησης)

Νέα\_είκονα = συνάρτηση (από)-κβάντισης(τιμή κβάντισης, απότελεσμα\_αποκωδικοποίησης)

Εικόνα = δημιουργία\_εικόνας(Νέα\_εικόνα)

Εμφάνιση Εικόνα

Εγγραφή αποτελεσμάτων σε αρχείο .txt

## Υλοποίηση

Αρχικά φορτώνουμε την εικόνα, και τις διαστάσεις της. Λαμβάνουμε τα εικονοστοιχεία (pixels) της εικόνας και τα αποθηκεύουμε σε μορφή πίνακα. Με την συνάρτηση quantization, διαιρούμε όλα τα εικονοστοιχεία με την τιμή κβάντισης και στρογγυλοποιούμε προς τα κάτω. Η συνάρτηση quantization δέχεται ως παραμέτρους, την τιμή κβάντισης και τον πίνακα με τα εικονοστοιχεία της αρχικής εικόνας. Επιστρέφει ως αποτέλεσμα τον νέο κβαντισμένο πίνακα.

Με την συνάρτηση encoder, υλοποιούμε τον κωδικοποιητή μήκους διαδρομής, όπου δέχεται ως είσοδο τον πίνακα που επέστρεψε η συνάρτηση quantization. Η κωδικοποίηση έχει γίνει ως εξής: διαβάζουμε τον πίνακα με τις κβαντισμένες τιμές, ανά γραμμή σημειώνοντας το αρχικό pixel. Όσο οι κβαντισμένες τιμές είναι ίδιες (λόγω της κβάντισης εμφανίζεται συχνά το φαινόμενο της επανάληψης) πηγαίνουμε στο επόμενο εικονοστοιχείο. Μόλις λάβουμε τιμή διαφορετική από την προηγούμενη τότε, σημειώνουμε ως τέλος με το σύμβολο ‘|’, και κρατάμε την θέση στην οποία εμφανίστηκε η νέα τιμή. Αυτή η διαδικασία επαναλαμβάνεται όσο είναι το μέγεθος της εικόνας. Μόλις τελειώσει η διαδικασία, το αποτέλεσμα αποθηκεύεται με την μορφή ‘string’ και το επιστρέφει.

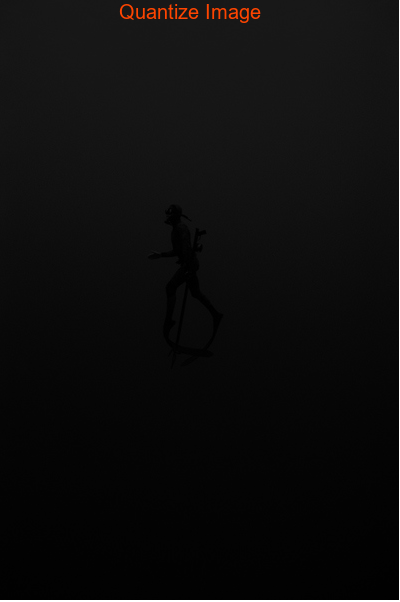
Ο αποκωδικοποιητής λαμβάνει ως παράμετρο το αποτέλεσμα που επέστρεψε η συνάρτηση encoder. Δημιουργούμε ένα πίνακα μηδενικών 3 διαστάσεων για τιμές RGB, στον οποίο θα αποθηκεύσει τις τιμές μετά την αποκωδικοποίηση. Αναθέτουμε την παράμετρο σε πίνακα όπου γίνεται ο διαχωρισμός βάση του συμβόλου ‘|’, δηλαδή σε κάθε σημείο στο οποίο άλλαζε η τιμή του εικονοστοχείου. Διαχωρίζουμε τον πίνακα με το σύμβολο ‘-‘, όπου δείχνει πόσα εικονοστοιχεία έχουν την ίδια τιμή κβάντισης, π.χ. *11;11;11-10;130,* σημαίνει πως από την θέση 10 μέχρι 130 έχουν τις RGB τιμές 11,11,11. Μόλις τελειώσει η διαδικασία αποκωδικοποίησης η συνάρτηση επιστρέφει τον πίνακα με τις RGB τιμές.

Αφού έχουμε κάνει την αποκωδικοποιήση, αυτό που μένει είναι να δημιουργήσουμε τον πίνακα με τα εικονοστοιχεία που είχε η αρχική εικόνα. Αυτό μπορεί να επιτευχθεί αν πολλαπλασιάσου με την ίδια τιμή, με την οποία κάναμε την κβάντιση. Προφανώς και το αποτέλεσμα θα είναι διαφορετικό, αφού κατά την διαδικασία της κβάντισης τα δεκαδικά μέρη τα οποία είχαν προκύψει από την διαίρεση χάθηκαν, και δεν υπάρχει τρόπος ανάκτησης τους σε αυτό το στάδιο. Επομένως θα υπάρχει μια μικρή διαφορά της αρχικής εικόνας με της κβαντισμένης.

Το αποτέλεσμα της κωδικοποιήσης το αποθηκεύουμε σε ένα ένα αρχείο .txt με το όνομα ‘*apotelesmata.txt’.*

## Αποτελέσματα

Η εικόνα πριν την κβαντίση: Κβαντισμένη εικόνα:



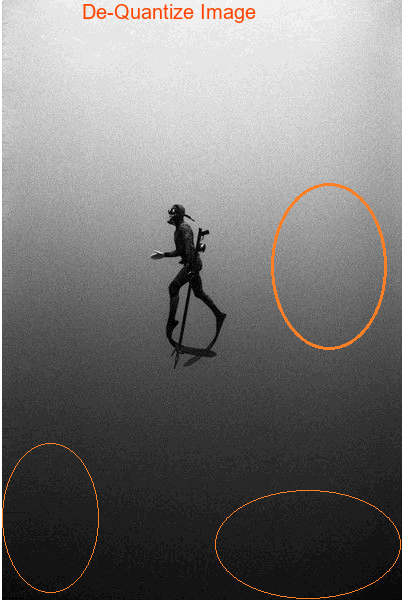


Εικόνα από την επαναφόρα της κβάντισης:



## Συμπεράσματα

Κβαντίζοντας την εικόνα ρίξαμε τα επίπεδα του γκρι, με αποτέλεσμα οι τιμές της κβαντισμένης εικόνας να είναι πιο κοντά στο μηδέν, για αυτό και έχει μειωθεί η ένταση του άσπρου. Αντιστρέφοντας την διαδικασία για να πάρουμε την αρχική εικόνα, λαμβάνουμε μια εικόνα η οποία δεν είναι ακριβώς όπως η αρχική, αλλά διαφορετική (τα σημεία έχουν κυκλωθεί στην τελική εικόνα). Αυτό συμβαίνει γιατί δεν μπόρεσαν να επανέλθουν ακριβώς οι αρχικές τιμές, αλλά με μικρή διαφορά προς τα κάτω. Στα κυκλώμενα σημεία παρατήρουμε την μεγαλύτερη διαφορά, που υπάρχει μεταξύ της αρχικής και της τελικής εικόνας. Οι κύκλοι δείχουν ότι το σημείο δεν είναι το ίδιο λείο σε σχέση με το αρχικό, παρά υπάρχει μια αισθητή διαφορά.



Τα ερωτήμα του βιβλίου!!!!

# Άσκηση 6.17

## Εκφώνηση άσκησης

*«Στην άσκηση αυτή, θα εφαρμόσετε τεχνικές DPCM για να συμπιέσετε βίντεο. Αν και αυτός δεν είναι ο πιο αποδοτικός τρόπος για τη συμπίεση βίντεο, αποτελεί εντούτοις μια καλή εισαγωγή στη συμπίεση βίντεο που θα ακολουθήσει στα επόμενα κεφάλαια. Το βίντεο μπορεί να οριστεί ως ακολουθία πλαισίων. Υπάρχει μεγάλη συνάφεια των δεδομένων από πλαίσιο σε πλαίσιο – δεν μεταβάλλονται δηλαδή όλα τα εικονοστοιχεία από πλαίσιο σε πλαίσιο, όπως για παράδειγμα συμβαίνει με τα εικονοστοιχεία του παρασκηνίου. Αυτό ακριβώς το γεγονός θα εκμεταλλευτείτε σε αυτήν την άσκηση. Εδώ σας ζητείτε η συγγραφή προγράμματος το οποίο θα διαβάσει μια ακολουθία πλαισίων βίντεο ως είσοδο, μαζί με μία παράμετρο κβάντισης. Το καθήκον σας είναι η υλοποίηση ενός αλγορίθμου παρόμοιου με τον DPCM, που θα αφήνει το πρώτο πλαίσιο ως έχει, αλλά θα υπολογίζει τις διαφορές μεταξύ διαδοχικών πλαισίων και θα κβαντίζει κάθε τιμή της διαφοράς των εικονοστοιχείων με τρόπο εξαρτώμενο από την τιμή κβάντισης που δέχεται ως είσοδο.»*

## Περιγραφή συστήματος/ψευδοκώδικας

Για την υλοποίηση και την επίλυση της άσκησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python 3.

Για να μπορεί να γίνει η εκτέλεση του προγράμματος θα πρέπει να υπάρχουν εγκατεστημένες στο σύστημα οι εξής βιβλιοθήκες: cv2 & numpy.

**Ψευδοκώδικας**

**|**

**|**

**Απαραίτητες ρυθμίσεις για την ανάγνωση και εγγραφή βίντεο με τη βοήθεια της βιβλιοθήκης cv2**

**|**

**|**

Όσο υπάρχουν πλαίσια από το βίντεο για να διαβάσουμε:

| Εικόνα = βιντεο.διαβασεΕπόμενηΕικόνα()

| Εικόνες.πρόσθεσε(εικόνα)

| Πλήθος\_εικόνων += 1

Εικόνες = Εικόνες.numpyArray()

Διαστάσεις\_εικόνας = Εικόνες.διαστάσεις()

Εμφάνισε(«Δώσε παράμετρο κβάντισης»)

Δώσε(Παραμετρο\_κβάντισης)

Για i από 1 μέχρι Πλήθος\_εικόνων:

| Πίνακας\_διαφορών(i) = Εικόνες(i) – Εικόνες(i-1)

| Κβαντισμένες\_εικόνες(i) = Στρογγυλοποίησε(Πίνακας\_διαφορών(i)/ Παράμετρο\_κβάντισης)

|

Για i από 1 μέχρι Πλήθος\_εικόνων -1:

**|** Ανακατασκευασμένες εικόνες(i) = Κβαντισμένες εικόνες(i) + | Κβαντισμένες εικόνες (i+1)

| Γράψε εικόνα στο αρχείο

## Υλοποίηση

* Ανάγνωση βίντεο

Αρχικά φορτώνουμε το βίντεο, και παίρνουμε τις διαστάσεις του. Μετά δηλώνουμε αντίστοιχα τις ιδιότητες και τη κωδικοποίηση του βίντεο που θα δημιουργήσουμε και τις διαστάσεις. Μετά αρχίζουμε και διαβάζουμε το βίντεο πλαίσιο προς πλαίσιο και για κάθε πλαίσιο το βάζουμε προσωρινά σε μία λίστα ως λίστα των αριθμητικών τιμών των pixels.

* Υπολογισμός διαφορών διαδοχικών πλαισίων

Αφού διαβάσουμε όλα τα πλαίσια του βίντεο μετατρέπουμε τη λίστα που αναφέραμε σε numpy array για να μας βοηθήσει στις πράξεις μεταξύ πινάκων. Έχοντας πια τα πλαίσια του βίντεο σε πίνακες numpy array για κάθε πλαίσιο παίρνουμε διαδοχικά αυτό και το αμέσως προηγούμενο του και υπολογίζουμε το πλαίσιο διαφορών τους και αποθηκεύουμε το αποτέλεσμα στην θέση που βρισκόταν το πλαίσιο που ελέγχαμε εκείνη την στιγμή.

* Κβάντιση των τιμών των πλαισίων

Έπειτα υπολογίζουμε τις κβαντισμένες τιμές κάθε πλαισίου διαφορών διαιρώντας κάθε κβαντισμένο πλαίσιο με τη παράμετρο κβάντισης και στρογγυλοποιώντας τις πιθανές δεκαδικές τιμές που προκύπτουν.

* Ανακατασκευή των πλαισίων

Τέλος ανακατασκευάζουμε κάθε πλαίσιο εικόνας από τα κβαντισμένα πλαίσια εικόνων προσθέτοντας το κάθε κβαντισμένο πλαίσιο με το επόμενο διαδοχικό του και «γράφοντας» το στο αρχείο βίντεο που αναφέραμε στην αρχή.

## Αποτελέσματα

Η εικόνα πριν την κβαντίση: Κβαντισμένη εικόνα:

Εικόνα από την επαναφόρα της κβάντισης:

## Συμπεράσματα και απαντήσεις

Βλέποντας το αρχικό βίντεο και το συμπιεσμένο βίντεο και συγκρίνοντας μεταξύ τους, βλέπουμε ότι στο συμπιεσμένο βίντεο έχοντας κβαντίσει το κάθε πλαίσιο ρίξαμε τα επίπεδα των χρωμάτων (RGB), με αποτέλεσμα οι τιμές των κβαντισμένων πλαισίων να είναι πιο κοντά στο μηδέν, για αυτό και έχει μειωθεί η ένταση της φωτεινότητας. Αντιστρέφοντας την διαδικασία για να πάρουμε την αρχική εικόνα σε κάθε πλαίσιο, λαμβάνουμε μια εικόνα η οποία δεν είναι ακριβώς όπως η αρχική, αλλά διαφορετική και λίγο υποφωτισμένη. Αυτό συμβαίνει γιατί δεν μπόρεσαν να επανέλθουν ακριβώς οι αρχικές τιμές, αλλά με μικρή διαφορά προς τα κάτω. Καθώς ανεβάζουμε τη παράμετρο κβάντισης οι συνέπειες αυτές θα είναι όλο και πιο φανερές καθώς και το μέγεθος του αρχείου θα μικραίνει όλο και περισσότερο λόγο της συμπίεσης.

Ο λόγος συμπίεσης που είχαμε με αυτά τα δεδομένα ήταν ανάμεσα στο [0.7-1.8] με παράμετρο κβαντισης Κ = [1,8].

# Άσκηση 8.17

## Εκφώνηση άσκησης

*«Στην άσκηση αυτή, θα υλοποιήσετε την τεχνική της αντιστάθμισης κίνησης και θα μελετήσετε πως επηρεάζει τα σφάλματα πρόβλεψης. Υποθέστε ότι το πρώτο θα είναι πάντα ένα πλαίσιο Ι και ότι το υπόλοιπα πλαίσια θα είναι τύπου Ρ. Η υπόθεση αυτή στην περίπτωση σύντομων ακολουθιών βίντεο που επεξεργάζεστε, που έχουν μήκος το πολύ 100 πλαισίων.»*

*1ο μέρος άσκησης:*

*«Στο πρώτο μέρος της άσκησης υποθέστε ότι θέλετε να προβλέπετε ολόκληρα Ρ πλαίσια και όχι κατά τμήματα. Η πρόβλεψη κάθε ολόκληρου πλαισίου γίνεται με βάση το προηγούμενο πλαίσιο. Υλοποιήστε μια διαδικασία που δέχεται είσοδο δύο πλαίσια, υπολογίζει τη διαφορά τους και επιστρέφει ένα πλαίσιο σφαλμάτων. Δεν υπολογίζετε διάνυσμα κίνησης. Να προβάλετε τα πλαίσια σφαλμάτων.»*

*2ο μέρος άσκησης:*

*«Στο δεύτερο θα υλοποιήσετε τεχνική πρόβλεψης κίνησης, η οποία υπολογίζει διανύσματα κίνησης ανά μπλοκ. Κάθε μπλόκ έχει το τυπικό MPEG μέγεθος 16Χ16. Υλοποιήστε μία συνάρτηση που δέχεται είσοδο δύο πλαίσια: ένα πλαίσιο αναφοράς, το οποίο θα χρησιμοποιηθεί κατά την αναζήτηση των διανυσμάτων κίνησης, και ένα πλαίσιο στόχο, το οποίο θα προβλεφθεί. Διαιρέστε το πλαίσιο στόχο σε μακρομπλόκ μεγέθους 16Χ16. Εάν το πλάτος και ύψος του πλαισίου δεν είναι πολλαπλάσια του 16, συμπληρώστε κατάλληλα το πλαίσιο με μαύρα εικονοστοιχεία. Για κάθε μπλοκ στο πλαίσιο-στόχο, ανατρέξτε στην αντίστοιχη θέση στο πλαίσιο αναφοράς και βρείτε την περιοχή που δίνει το καλύτερο ταίριασμα, όπως έχει εξηγηθεί στο κείμενου του κεφαλαίου. Χρησιμοποιήστε τη μετρική SAD σε περιοχή αναζήτησης που προκύπτει για k=16, έτσι ώστε τα διανύσματα κίνησης να έχουν μέγεθος το πολύ 16 εικονοστοιχείων ως προς κάθε κατεύθυνση. Με βάση το μπλοκ πρόβλεψης, υπολογίστε το μπλοκ σφαλμάτων ως τη διαφορά μεταξύ του αρχικού μπλόκ και του προβλεφθέντος. Αφού αυτή η διαδικασία ολοκληρωθεί για όλα τα μπλοκ, θα προκύψει ένα πλαίσιο σφαλμάτων. Να γίνει η προβολή όλων των πλαισίων σφαλμάτων»*

## Περιγραφή συστήματος/ψευδοκώδικας

Για την υλοποίηση και την επίλυση της άσκησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python 3.

Για να μπορεί να γίνει η εκτέλεση του προγράμματος θα πρέπει να υπάρχουν εγκατεστημένες στο σύστημα οι εξής βιβλιοθήκες: scipy, numpy & cv2.

**Ψευδοκώδικας**

**ΘΕΛΕΙ ΝΑ ΒΑΛΕΙΣ ΚΑΙ ΤΟΝ SAD ΣΤΟ ΔΥΕΤΕΡΟ ΜΕΡΟΣ!!!!!**

**1ο μέρος Άσκησης**

Ρυθμίσεις και ιδιότητες βίντεο

|

| αντικείμενο για ανάγνωση βίντεο ()

| ύψος πλαισίου ()

| μήκος πλαισίου ()

| κωδικοποίση βίντεο ()

| αρχείο για εγγραφή του βίντεο ()

|

Τέλος ρυθμίσεων και ιδιοτήτων του βίντεο

Προηγούμενο πλαίσιο = Διάβασε πρώτο πλαίσιο Ι ()

Προηγούμενο πλαίσιο = Μετατροπή σε ασπρόμαυρο (Προηγούμενο πλαίσιο)

Όσο υπάρχουν πλαίσια Ρ για διάβασμα:

| Τωρινό πλαίσιο = Διάβασε επόμενο πλαίσιο Ρ ()

| Αν δεν υπάρχει επόμενο πλαίσιο

| | ΕΞΟΔΟΣ από loop

| Αλλιως

| | Τωρινό πλαίσιο = Μετατροπή σε ασπρόμαυρο (Τωρινό πλαίσιο)

| Υπολόγισε πλαίσιο σφαλμάτων με το προηγούμενο πλαίσιο()

| Γράψε στο καινούριο βίντεο το πλαίσιο σφαλμάτων ()

| Προηγούμενο πλαίσιο = Τωρινό πλαίσιο

Υπολόγισε εντροπία του καινούριου βίντεο ()

**2ο μέρος Άσκησης**

***Μέθοδοι / συναρτήσεις προγράμματος***

**Συνάρτηση χωρισμού** **σε μακρομπλόκς**(πλήθος\_γραμμών, πλήθος\_στηλών, πίνακας\_εικόνας):

Μπλοκς = []

Για k από 0 μέχρι το (πλάτος\_εικόνας - πλήθος\_γραμμών+1) με βήμα = πλήθος\_γραμμών :

| Για j μέχρι (ύψος\_εικόνας - πλήθος\_στηλών+1) με βήμα = πλήθος\_στηλών:

| μακρομπλόκ = πίνακας [ k:k+πλήθος\_γραμμών, c:c+πλήθος\_στηλών]

| Μπλοκς.πρόσθεσε(μακρομπλόκ)

μετατροπή του Μπλοκς σε πίνακα numpy array

*Επιστροφή πίνακα Μπλοκς*

**Συνάρτηση προσθήκη\_μαύρων\_εικονοστοιχείων**(εικόνα):

Ύψος\_εικόνας

Πλάτος\_εικόνας

προσθήκη\_μαύρων\_εικονοστοιχείων = μετατροπή\_σε\_πίνακα(μαύρο\_pixel() \* (((πλάτος-1 // 16+1)\* 16)- πλάτος))

Για i μέχρι μέγεθος\_εικόνας

| γραμμή = μετατροπή\_σε\_πίνακα(i, προσθήκη\_μαύρων\_εικονοστοιχείων,

| άξονας\_γραμμών)

| προσωρινή\_εικόνα = γραμμή

προσθήκη\_μαύρης\_γραμμής = μετατροπή\_σε\_πίνακα(μαύρο\_pixel() \*

(((πλάτος-1 // 16+1) \* 16)- πλάτος))

Για j μέχρι (((πλάτος-1 // 16+1) \* 16)- πλάτος)):

προσωρινή\_εικόνα = προσθηκη\_μαύρης\_γραμμής

*Επιστροφή μετατροπή\_σε\_πίνακα(προσωινή\_εικόνα)*

**Συνάρτηση μαύρο\_εικονστοιχείο**():

*Επιστροφή [0]*

**Συνάρτηση ιεραρχική\_αναζήτηση(**πίνακα):

πίνακα1 = μετατροπή\_σε\_πίνακα()

x, y, = μορφή\_πίνακα /\* να έχουμε τις συντεταγμένες σε μορφή πίνακα\*/

*πίνακα2* = [] /\*αρχικοποιήση του πίνακα2 σαν λίστα\*/

Για i από 0 μέχρι x με\_βήμα 2:

| Για j από 0 μέχρι y με\_βήμα 2:

| | *πίνακα2* = πίνακα1([x], [y]) /\*ο πίνακα2 είναι σε μορφή λίστας \*/

*πίνακα2* = μετατροπή\_σε\_πίνακα()

*πίνακα2* = σχηματισμός(x/2, y/2) /\* για να αποκτήσει τις διαστάσεις που θέλουμε \*/

*Επιστροφή πίνακα2*

**Συνάρτηση ανάλυση\_σε\_επίπεδα** (εικόνα ανφοράς, εικόνα στόχος):

εικόνα ανφοράς = μετατροπή\_σε\_πίνακα()

εικόνα στόχος = μετατροπή\_σε\_πίνακα()

Για i μέχρι 2: //τα επίπεδα

| ιεραρχική\_εικόνα1 = ιεραρχική\_εικόνα1 + [ιεραρχική\_αναζήτηση(εικόνα αναφοράς)]

| εικόνα αναφοράς = ιεραρχική\_αναζήτηση(εικόνα αναφοράς)

| ιεραρχική\_εικόνα2 = ιεραρχική\_εικόνα2 + [ιεραρχική\_αναζήτηση(εικόνα στόχος)]

| εικόνα στόχος = ιεραρχική\_αναζήτηση(εικόνα στόχος)

**Συνάρτηση βελτιστοποίηση\_Επιπέδων**(blocks\_move, ιεραρχική\_εικόνα1, ιεραρχική\_εικόνα2):

size\_block = [8, 16]

Για k μέχρι 2:

| /\* διαίρεση της μικρής εικόνας σε blocks μεγέθους size\_block \* size\_block \*/

| εικόνα1 = blocks(size\_block [k], size\_block [k], ιεραρχική\_εικονα1[size\_block [1- k])

| εικόνα2 = blocks(size\_block [k], size\_block [k], ιεραρχική\_εικονα2[size\_block [1- k])

| blocks\_χωρίς\_κίνηση=[]

| Για i μέχρι το μέγεθος(blocks\_move):/\*ελέγχουμε μόνο τα blocks στα οποία βρήκαμε κίνηση\*/

| | Αν εύρεση\_κίνησης (εικόνα1[blocks\_move[i]] – εικονα2[blocks\_move[i]]):

| | | Συνέχισε

| | Αλλιώς:

| | | blocks\_χωρίς\_κίνηση = blocks\_χωρίς\_κίνηση [i] /\* αν δεν υπάρχει κίνηση

| | | βάζουμε το block στην λίστα\*/

| blocks\_move =[αν κάποιο block\_χωρίς\_κίνηση δεν είναι στη λίστα blocks\_moves,αφαίρεσε το]

*Επιστροφή εικόνα1, εικόνα2, blocks\_move*

**Συνάρτηση εύρεση\_κίνησης**(πίνακα):

πίνακα1 = μετατροπή\_σε\_πίνακα()

x, y = μορφή\_πίνακα

πλήθος\_μηδενικών\_στοιχείων = x \* y – μη\_μηδενικά\_στοιχεία(πίνακα1)

Αν πλήθος\_μηδενικών\_στοιχείων >= 0.8 \*x \*y:

Επέστρεψε 0

Αλλιώς:

Επέστρεψε 1

**Συνάρτηση ανακατασκευή\_εικόνας**(x, y, εικόνα2):

ρ =1

Για i μέχρι x:

| έξοδος = μετατροπή\_σε\_πίνακα(εικόνα2 [i \*y]) /\* για να πάρουμε τις διαστάσεις

| της κάθε εικόνας\*/

| Για j μέχρι (y-1):

| | έξοδος = ένωσε(έξοδο, εικόνα2[ρ], άξονα\_στηλών)

| | ρ +=1

| ρ +=1

| Αν i == 0:

| ανακατασκευασμένη\_εικόνα = έξοδο

| Αλλιώς:

| ανακατασκευασμένη \_εικόνας = ένωσε((εμφάνιση\_εικονας, έξοδο), άξονας\_γραμμών)

*Επιστροφή ανακατασκευασμένη \_εικόνας*

***Κυρίως πρόγραμμα***

Ρυθμίσεις και ιδιότητες βίντεο

|

| αντικείμενο για ανάγνωση βίντεο ()

| ύψος πλαισίου ()

| μήκος πλαισίου ()

| κωδικοποίση βίντεο ()

| αρχείο για εγγραφή του βίντεο ()

|

Τέλος ρυθμίσεων και ιδιοτήτων του βίντεο

Προηγούμενο πλαίσιο = Διάβασε πρώτο πλαίσιο Ι ()

Προηγούμενο πλαίσιο = Μετατροπή σε ασπρόμαυρο (Προηγούμενο πλαίσιο)

Αρχικοποίηση ιεραρχικης εικόνας 1 ως GLOBAL

Αρχικοποίηση ιεραρχικης εικόνας 2 ως GLOBAL

Όσο υπάρχουν πλαίσια Ρ για διάβασμα:

| Τωρινό πλαίσιο = Διάβασε επόμενο πλαίσιο Ρ ()

| Αν δεν υπάρχει επόμενο πλαίσιο

| | ΕΞΟΔΟΣ από loop

| Αλλιως

| | Αν μέγεθος προηγούμενου πλαισίου δεν διαρείται με το 16:

| | | **προσθήκη\_μαυρων\_εικονστοιχείων**(προηγούμενο πλαίσιο)

| | Αν μέγεθος τωρινού πλαισίου δεν διαρείται με το 16:

| | | **προσθήκη\_μαυρων\_εικονστοιχείων**(τωρινό πλαίσιο)

Τωρινό πλαίσιο = Μετατροπή σε ασπρόμαυρο (Τωρινό πλαίσιο)

| Ιεραρχική εικόνα 1 = [Προηγούμενο πλαίσιο]

| Ιεραρχική εικόνα 2 = [Τωρινό πλαίσιο]

| **Ανάλυση\_σε\_επίπεδα(**Προηγούμενο πλαίσιο, Τωρινό πλαίσιο)

| Μπλόκ\_κίνησης = []

| Για i από 0 μέχρι μέγεθος Προηγούμενου πλαισίου:

| | Αν **εύρεση\_κίνησης(**Προηγούμενο πλαίσιο[i] – Τωρινό πλαίσιο[i]):

| | | Μπλόκ\_κίνησης.πρόσθεσε(i)

| Εικόνα 1, Εικόνα 2, Μπλόκ\_κίνησης = **Βελτιστοποίηση\_επιπέδων**(Μπλόκ κίνησης, Ιεραρχική εικόνα 1, Ιεραρχική εικόνα 2)

| Για i από 0 μέχρι μέγεθος Μπλόκ κίνησης:

| | προβλεπόμενο Μπλόκ = **SAD**(Εικόνα 1, Εικόνα 2, Μπλόκ κίνησης[i])

| | Εικόνα 1[Μπλόκ κίνησης[i]] = Εικόνα 1[προβλεπόμενο Μπλόκ]

| Πλαίσια σφαλμάτων = Εικόνα 2 – Εικόνα 1

| **ανακατασκευή\_εικόνας(**Πλαίσια σφαλμάτων**)**

| Γράψε στο καινούριο βίντεο την ανακατασκευασμένη εικόνα των πλαισίων σφαλμάτων ()

| Προηγούμενο πλαίσιο = Τωρινό πλαίσιο

Υπολόγισε εντροπία του καινούριου βίντεο ()

## Υλοποίηση

**Πρώτο μέρος**

Για την υλοποίηση του πρώτου μέρους της άσκησης ξεκινήσαμε διαβάζοντας ένα ένα τα πλαίσια του βίντεο και απλά για κάθε δυο διαδοχικά πλαίσια τα μετατρέπαμε σε δύο πίνακες με τις τιμές για τα pixel των εικόνων αντίστοιχα και αφαιρούσαμε αυτούς τους δύο πίνακες για να υπολογίσουμε τα πλαίσια σφαλμάτων, γράφοντας στο καινούριο βίντεο κάθε φορά το κάθε πλαίσιο σφαλμάτων.

**Δεύτερο μέρος**

Για την υλοποίηση του δεύτερου μέρους του προβλήματος χρησιμοποιήθηκε η τεχνική ιεραρχικής αναζήτησης και η μετρική SAD(Sum absolute differences). Θα μπορούσε να χρησιμοποιηθεί και η τεχνική λογαριθμικής αναζήτησης αλλά θα έπαιρνε πολύ περισσότερο χρόνο για κάθε πλαίσιο λόγω της ανάλυσης των βίντεο και επιπλέον μας βοηθάει και στην επίλυση της επόμενης άσκησης.

* Ανάγνωση βίντεο και έλεγχος διαίρεσης με το 16

Πιο συγκεκριμένα πάλι ξεκινάμε διαβάζοντας ένα ένα τα πλαίσια του βίντεο ελέγχοντας στην αρχή αν η εικόνα μπορεί να διαιρεθεί σε μακρομπλόκ μεγέθους 16Χ16 και αν όχι γεμίζουμε τις υπολειπόμενες θέσεις του πίνακα τιμών του πλαισίου με μηδενικά (δηλαδή μαύρα pixel) για να μπορεί να διαιρεθεί.

* Τεχνική Ιεραρχικής αναζήτησης και ανάλυσης

Στη συνέχεια μετατρέπουμε τα πλαίσια σε ασπρόμαυρα για ευκολότερες πράξεις και αρχικοποιούμε ως πρώτο επίπεδο της ιεραρχικής μας ανάλυσης τις εικόνες στις αρχικές τους μορφές. Συνεχίζουμε καλώντας τη συνάρτηση που κάνει ανάλυση των εικόνων σε επίπεδα, δηλαδή για κάθε δύο διαδοχικά πλαίσια τα αναλύουμε σε επίπεδα οπού σε κάθε επίπεδο μειώνουμε στο μισό την ανάλυση, το μέγεθος των μακρομπλόκ αλλά και την παράμετρο k που καθορίζει την περιοχή αναζήτησης που χρησιμοποιείται στην SAD. Έτσι ενώ ξεκινάμε με την εικόνα στη πλήρες μορφή της μετά την χωρίζουμε σε μακρομπλόκ 16Χ16 και την αποθηκεύουμε ως δεύτερο επίπεδο όπως ζητείται και μετά σε 8Χ8 κ.ο.κ μέχρι να φτάσουμε στο 4Χ4 που είναι το μικρότερο που θέλουμε και το υψηλότερο επίπεδο στην ιεραρχική αναζήτηση. Με αυτή την τεχνική επιτυγχάνουμε πολύ γρήγορες πράξεις και αναζήτηση του προβλεπόμενου μπλόκ στην SAD καθώς τα μακρομπλόκ στο υψηλότερο επίπεδο έχουν μικρότερο μέγεθος και η περιοχή αναζήτησης είναι μικρότερη.

* Εύρεση διανυσμάτων κίνησης

Έτσι στο υψηλότερο επίπεδο γίνεται μια πλήρης αναζήτηση και βρίσκονται τα διανύσματα κίνησης, όπου τα υπολογίζουμε βρίσκοντας τη διαφορά ανάμεσα σε δυο μακρομπλόκ στις αντίστοιχες θέσεις κάθε μακρομπλόκ και έπειτα για κάθε από αυτά ελέγχουμε αν το ποσοστό των μηδενικών στο μακρομπλόκ διαφορών είναι μεγαλύτερο του 90% δηλαδή δεν υπήρχει αξιοσήμαντη κίνηση, ενώ αν υπήρχε κίνηση αποθηκεύουμε τη θέση του μακρομπλόκ για να τη ξαναχρησιμοποιήσουμε στο κατώτερο επίπεδο. Έπειτα προχωράμε στο κατώτερο επίπεδο βελτιώνοντας στην ουσία τα μπλόκ και τα διανύσματα κίνησης που υπήρχε κίνηση στο υψηλότερο επίπεδο και τα ξαναελέγχουμε στο χαμηλότερο επίπεδο αυτή τη φορά.

* Εφαρμογή μετρικής SAD

Έχοντας υπολογίσει πλέον όλα τα διανύσματα κίνησης για κάθε μακρομπλόκ επιλέγουμε τα μπλόκ που εντοπίσαμε κίνηση και εφαρμόζουμε την μετρική SAD για να βρούμε πιο μακρομπλόκ του πλαισίου αναφοράς αντιστοιχεί καλύτερα στο πλαίσιο στόχο ανατρέχοντας κάθε φορά στην αντίστοιχη θέση και υπολογίζοντας κάθε φορά το άθροισμα της απόλυτης διαφοράς του κάθε μακρομπλόκ του πλαισίου αναφοράς με τα αντίστοιχα γειτονικά μακρομπλόκ του μακρομπλόκ του πλαισίου στόχου στην αντίστοιχη θέση, με μέγιστη απόσταση κάθε φορα το μήκος του μακρομπλόκ και με το ιδανικότερο μακρομπλόκ να είναι αυτό που έχει το μικρότερο αποτέλεσμα (δηλαδή αν έχουμε k = 16 τότε όλα τα μακρομπόκ έχουν μήκος και πλάτος 16Χ16 άρα θα υπολογίσουμε το άθροισμα τον απόλυτων διαφορών με τα μακρομπλόκ του πλαισίου στόχου που βρίσκονται σε μέγιστη απόσταση ίση με 16 προς όλες τις κατευθύνσεις από το μακρομπλόκ του πλαισίου στόχου στην αντίστοιχη θέση με το μακρομπλόκ του πλαισίου ανφοράς που ελέγχουμε και θα πάρουμε το ως το ιδανικότερο μακρομπλόκ αυτό που μας δίνει το μικρότερο αποτέλεσμα καθώς δηλώνει ότι υπάρχουν λιγότερες διαφορές με το πλαίσιο αναφοράς από τα άλλα.)

* Ανακτασκευή εικόνας πλαισίων σφαλμάτων με τα προβλεπόμενα μακρομπλόκ

Έχοντας υπολογίσει τα προβλεπόμενα μπλόκ με βάση την μετρική SAD για κάθε μακρομπλόκ αντικαθιστούμε σε κάθε πλαίσιο στη θέση που εντοπίζεται κίνηση το ήδη υπάρχον μπλόκ του πλαισίου αναφοράς με το αντίστοιχο προβλεπόμενο από το πλαίσιο στόχο που βρήκαμε πριν.

Μετά υπολογίζουμε τα πλαίσια σφαλμάτων μεταξύ των μακρομπλόκ του προβλεπόμενου πλαισίου αναφοράς και τα μαρκομπλόκ του πλαισίου στόχου αφαιρώντας κάθε μακρομπλόκ του πλαισίου στόχου με το αντίστοιχο μακρομπλόκ του προβλεπόμενου πλαισίου αναφοράς.

Τέλος καλούμε τη συνάρτηση ανακατασκευής της εικόνας που παίρνει όλα τα μακρομπλόκ 16Χ16 και τα επαναφέρει στο αρχικό μέγεθος «ενώνοντάς» τα, δίνοντας μας την πλήρης εικόνα στην ανάλυση που θέλουμε έχοντας πιά το ολοκληρωμένο πλαίσιο σφαλμάτων όπου το γράφουμε και στο τελικό αρχείο βίντεο

## Αποτελέσματα

Εδώ βλέπουμε αρχικά ένα πλαίσιο από το αρχικό βίντεοA picture containing indoor, athletic game, wall, sport

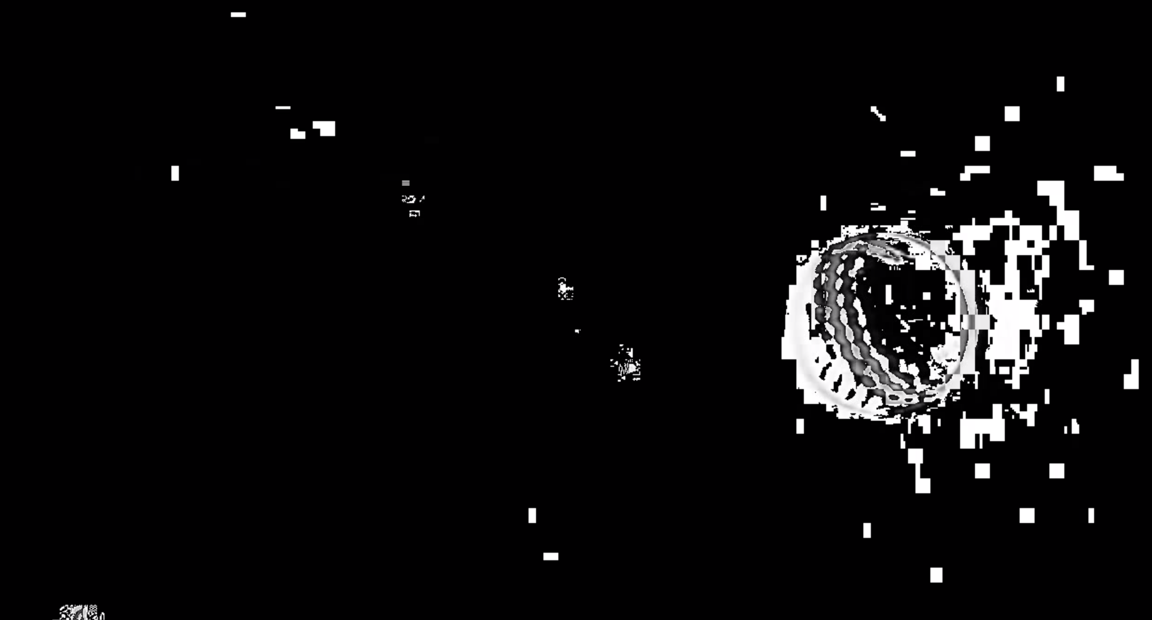
Description automatically generated

A black sign with white text

Description automatically generatedΤο αποτέλεσμα του πλαισίου σφαλμάτων χρησιμοποιώντας τον πρώτο τρόπο από το **πρώτο μέρος της άσκησης**A picture containing outdoor object

Description automatically generated

Το αποτέλεσμα του πλαισίου σφαλμάτων χρησιμοποιώντας τον δεύτερο τρόπο από το **δεύτερο μέρος της άσκησης**



A close up of a logo

Description automatically generated

## Συμπεράσματα

# Άσκηση 8.18

## Εκφώνηση άσκησης

*«Σε αυτήν την προγραμματιστική άσκηση, θα υλοποιήσετε έναν απλό κωδικοποιητή μήκους διαδρομής, ο οποίος επεξεργάζεται δεδομένα εικόνας ή οποιονδήποτε δισδιάστατο πίνακα τιμών, κβαντίζοντας αρχικά τις τιμές εισόδου και στη συνέχεια εφαρμόζοντας σε αυτές κωδικοποίηση μήκους διαδρομής. Η κβάντιση θα τονίσει το φαινόμενο της επανάληψης τιμών. Το πρόγραμμά σας πρέπει να δέχεται ως είσοδο μια εικόνα αποχρώσεων του γκρι (ή μια έγχρωμη εικονα RGB) και μια τιμή που θα ρυθμίζει τις τιμές επιπέδων κβάντισης.»*

## Περιγραφή συστήματος/ψευδοκώδικας

Για την υλοποίηση και την επίλυση της άσκησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python 3.

Για να μπορεί να γίνει η εκτέλεση του προγράμματος θα πρέπει να υπάρχουν εγκατεστημένες στο σύστημα οι εξής βιβλιοθήκες: PIL & numpy.

**Ψευδοκώδικας**

***Μέθοδοι / συναρτήσεις προγράμματος***

Συνάρτηση χωρισμού σε μακρομπλόκς(πλήθος\_γραμμών, πλήθος\_στηλών, πίνακας\_εικόνας):

Μπλοκς = []

Για k από 0 μέχρι το (πλάτος\_εικόνας - πλήθος\_γραμμών+1) με βήμα = πλήθος\_γραμμών :

| Για j μέχρι (ύψος\_εικόνας - πλήθος\_στηλών+1) με βήμα = πλήθος\_στηλών:

| μακρομπλόκ = πίνακας [ k:k+πλήθος\_γραμμών, c:c+πλήθος\_στηλών]

| Μπλοκς.πρόσθεσε(μακρομπλόκ)

μετατροπή του Μπλοκς σε πίνακα numpy array

*Επιστροφή πίνακα Μπλοκς*

Συνάρτηση προσθήκη\_μαύρων\_εικονοστοιχείων(εικόνα):

Ύψος\_εικόνας

Πλάτος\_εικόνας

προσθήκη\_μαύρων\_εικονοστοιχείων = μετατροπή\_σε\_πίνακα(μαύρο\_pixel() \* (((πλάτος-1 // 16+1)\* 16)- πλάτος))

Για i μέχρι μέγεθος\_εικόνας

| γραμμή = μετατροπή\_σε\_πίνακα(i, προσθήκη\_μαύρων\_εικονοστοιχείων,

| άξονας\_γραμμών)

| προσωρινή\_εικόνα = γραμμή

προσθήκη\_μαύρης\_γραμμής = μετατροπή\_σε\_πίνακα(μαύρο\_pixel() \*

(((πλάτος-1 // 16+1) \* 16)- πλάτος))

Για j μέχρι (((πλάτος-1 // 16+1) \* 16)- πλάτος)):

προσωρινή\_εικόνα = προσθηκη\_μαύρης\_γραμμής

*Επιστροφή μετατροπή\_σε\_πίνακα(προσωινή\_εικόνα)*

Συνάρτηση μαύρο\_εικονστοιχείο():

*Επιστροφή [0]*

Συνάρτηση ιεραρχική\_αναζήτηση(πίνακα):

πίνακα1 = μετατροπή\_σε\_πίνακα()

x, y, = μορφή\_πίνακα /\* να έχουμε τις συντεταγμένες σε μορφή πίνακα\*/

*πίνακα2* = [] /\*αρχικοποιήση του πίνακα2 σαν λίστα\*/

Για i από 0 μέχρι x με\_βήμα 2:

| Για j από 0 μέχρι y με\_βήμα 2:

| | *πίνακα2* = πίνακα1([x], [y]) /\*ο πίνακα2 είναι σε μορφή λίστας \*/

*πίνακα2* = μετατροπή\_σε\_πίνακα()

*πίνακα2* = σχηματισμός(x/2, y/2) /\* για να αποκτήσει τις διαστάσεις που θέλουμε \*/

*Επιστροφή πίνακα2*

Συνάρτηση ανάλυση\_σε\_επίπεδα (εικόνα ανφοράς, εικόνα στόχος):

εικόνα ανφοράς = μετατροπή\_σε\_πίνακα()

εικόνα στόχος = μετατροπή\_σε\_πίνακα()

Για i μέχρι 2: //τα επίπεδα

| ιεραρχική\_εικόνα1 = ιεραρχική\_εικόνα1 + [ιεραρχική\_αναζήτηση(εικόνα αναφοράς)]

| εικόνα αναφοράς = ιεραρχική\_αναζήτηση(εικόνα αναφοράς)

| ιεραρχική\_εικόνα2 = ιεραρχική\_εικόνα2 + [ιεραρχική\_αναζήτηση(εικόνα στόχος)]

| εικόνα στόχος = ιεραρχική\_αναζήτηση(εικόνα στόχος)

Συνάρτηση κίνηση\_blocks(blocks\_move, ιεραρχική\_εικόνα1, ιεραρχική\_εικόνα2):

size\_block = [8, 16]

Για k μέχρι 2:

| /\* διαίρεση της μικρής εικόνας σε blocks μεγέθους size\_block \* size\_block \*/

| εικόνα1 = blocks(size\_block [k], size\_block [k], ιεραρχική\_εικονα1[size\_block [1- k])

| εικόνα2 = blocks(size\_block [k], size\_block [k], ιεραρχική\_εικονα2[size\_block [1- k])

| blocks\_χωρίς\_κίνηση=[]

| Για i μέχρι το μέγεθος(blocks\_move):/\*ελέγχουμε μόνο τα blocks στα οποία βρήκαμε κίνηση\*/

| | Αν εύρεση\_κίνησης (εικόνα1[blocks\_move[i]] – εικονα2[blocks\_move[i]]):

| | | Συνέχισε

| | Αλλιώς:

| | | blocks\_χωρίς\_κίνηση = blocks\_χωρίς\_κίνηση [i] /\* αν δεν υπάρχει κίνηση

| | | βάζουμε το block στην λίστα\*/

| blocks\_move =[αν κάποιο block\_χωρίς\_κίνηση δεν είναι στη λίστα blocks\_moves,αφαίρεσε το]

| /\* ο κώδικας για την συγκεκριμένη εντολή βρίσκετε στη γραμμή 128\*/

*Επιστροφή εικόνα1, εικόνα2, blocks\_move*

Συνάρτηση εύρεση\_κίνησης(πίνακα):

πίνακα1 = μετατροπή\_σε\_πίνακα()

x, y = μορφή\_πίνακα

πλήθος\_μηδενικών\_στοιχείων = x \* y – μη\_μηδενικά\_στοιχεία(πίνακα1)

Αν πλήθος\_μηδενικών\_στοιχείων >= 0.8 \*x \*y:

Επέστρεψε 0

Αλλιώς:

Επέστρεψε 1

Συνάρτηση ανακατασκευή\_εικόνας(x, y, εικόνα2):

ρ =1

Για i μέχρι x:

| έξοδος = μετατροπή\_σε\_πίνακα(εικόνα2 [i \*y]) /\* για να πάρουμε τις διαστάσεις

| της κάθε εικόνας\*/

| Για j μέχρι (y-1):

| | έξοδος = ένωσε(έξοδο, εικόνα2[ρ], άξονα\_στηλών)

| | ρ +=1

| ρ +=1

| Αν i == 0:

| ανακατασκευασμένη\_εικόνα = έξοδο

| Αλλιώς:

| ανακατασκευασμένη \_εικόνας = ένωσε((εμφάνιση\_εικονας, έξοδο), άξονας\_γραμμών)

*Επιστροφή ανακατασκευασμένη \_εικόνας*

Συνάρτηση αρχικοποίησης(ς):

ιεραρχική\_εικόνα1 = [εικόνα[0]] /\* λίστα από εικονα[0]\*/

ιεραρχική\_εικόνα2 = [εικόνα[ς]] /\* λίστα από εικόνα [ς]\*/

αναλυση\_σε\_επίπεδα ([εικόνα[0], εικόνα[ς]) /\* εύρεση διαφορών 2 εικόνων, ρίχνοντας την

ανάλυση \*/

εικόνα1 = blocks(4, 4, ιεραρχική\_εικόνα[2]) /\* διαίρεση της εικόνας σε 4x4 blocks\*/

εικόνα2 = blocks(4, 4, ιεραρχική\_εικόνα[2])

blocks\_move = [] /\*θα αποθηκευτούν τα blocks στα οποία βρέθηκε κίνηση \*/

Για i μέχρι το μέγεθος(εικόνα1):

| Αν εύρεση\_κίνησης (εικόνα1[i]-εικόνα2[i]):

| | blocks\_move = blocks\_move + [i] /\* αν βρεθεί κίνηση την καταγράφει\*/

*Επιστροφή εικόνα1, εικόνα2, ιεραρχική\_εικόνα1, ιεραρχική\_εικόνα2, blocks\_move*

Συνάρτηση main (εικόνες):

Αν (εικόνες[0] ή εικόνες[1]) % 16 !=0 /\* αν οι εικόνες δεν διαιρούνται ακριβώς σε block 16x16

| προσθέτουμε μαύρα εικονοστοιχεία\*/

| Για i μέχρι το μέγεθος(εικόνες):

| εικόνες[i] = πρόσθεση\_μαύρων\_εικονοστοιχείων(εικόνες[i])

εικόνες = μετατροπή\_σε\_πίνακα(εικόνες)

Για ς μέχρι το μέγεθος(εικόνες):

| /\* αρχικποιήση των πινάκων με τις διαστάσεις, από την συνάρτηση αρχικοποίησης\*/

| εικόνα1, εικόνα2, ιεραρχική\_εικόνα1, ιεραρχική\_εικόνα2, blocks\_move = αρχικοποίηση(ς)

| /\* υπολογισμός των block που θα αντικατασταθούν, από την συνάρτηση κίνηση\_blocks\*/

| εικόνα1, εικόνα2, blocks\_move = κίνηση\_blocks(blocks\_move, ιεραρχική\_εικόνα1,

| ιεραρχική\_εικόνα2)

| Για i μέχρι το μέγεθος(blocks\_move):

| | εικόνα2[blocks\_move[i]] = εικόνα1[blocks\_move[i]] /\*αντικατάσταση κινούμενων block

| | με παρασκήνιο \*/

| Διαίρεση για μπλοκ μεγέθους 16\*16

| εικόνες[ς] = ανακατασκευή\_εικόνας(x, y, εικονα2)

*Επιστροφή εικόνες*

## Υλοποίηση

Αρχικά φορτώνουμε την εικόνα, και τις διαστάσεις της. Λαμβάνουμε τα εικονοστοιχεία (pixels) της εικόνας και τα αποθηκεύουμε σε μορφή πίνακα. Με την συνάρτηση quantization, διαιρούμε όλα τα εικονοστοιχεία με την τιμή κβάντισης και στρογγυλοποιούμε προς τα κάτω. Η συνάρτηση quantization δέχεται ως παραμέτρους, την τιμή κβάντισης και τον πίνακα με τα εικονοστοιχεία της αρχικής εικόνας. Επιστρέφει ως αποτέλεσμα τον νέο κβαντισμένο πίνακα.

Με την συνάρτηση encoder, υλοποιούμε τον κωδικοποιητή μήκους διαδρομής, όπου δέχεται ως είσοδο τον πίνακα που επέστρεψε η συνάρτηση quantization. Η κωδικοποίηση έχει γίνει ως εξής: διαβάζουμε τον πίνακα με τις κβαντισμένες τιμές, ανά γραμμή σημειώνοντας το αρχικό pixel. Όσο οι κβαντισμένες τιμές είναι ίδιες (λόγω της κβάντισης εμφανίζεται συχνά το φαινόμενο της επανάληψης) πηγαίνουμε στο επόμενο εικονοστοιχείο. Μόλις λάβουμε τιμή διαφορετική από την προηγούμενη τότε, σημειώνουμε ως τέλος με το σύμβολο ‘|’, και κρατάμε την θέση στην οποία εμφανίστηκε η νέα τιμή. Αυτή η διαδικασία επαναλαμβάνεται όσο είναι το μέγεθος της εικόνας. Μόλις τελειώσει η διαδικασία, το αποτέλεσμα αποθηκεύεται με την μορφή ‘string’ και το επιστρέφει.

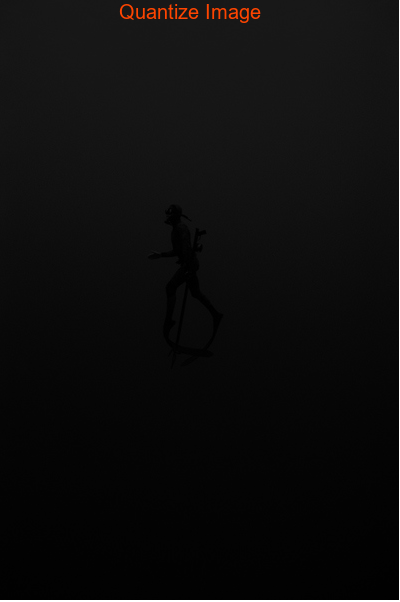
Ο αποκωδικοποιητής λαμβάνει ως παράμετρο το αποτέλεσμα που επέστρεψε η συνάρτηση encoder. Δημιουργούμε ένα πίνακα μηδενικών 3 διαστάσεων για τιμές RGB, στον οποίο θα αποθηκεύσει τις τιμές μετά την αποκωδικοποίηση. Αναθέτουμε την παράμετρο σε πίνακα όπου γίνεται ο διαχωρισμός βάση του συμβόλου ‘|’, δηλαδή σε κάθε σημείο στο οποίο άλλαζε η τιμή του εικονοστοχείου. Διαχωρίζουμε τον πίνακα με το σύμβολο ‘-‘, όπου δείχνει πόσα εικονοστοιχεία έχουν την ίδια τιμή κβάντισης, π.χ. *11;11;11-10;130,* σημαίνει πως από την θέση 10 μέχρι 130 έχουν τις RGB τιμές 11,11,11. Μόλις τελειώσει η διαδικασία αποκωδικοποίησης η συνάρτηση επιστρέφει τον πίνακα με τις RGB τιμές.

Αφού έχουμε κάνει την αποκωδικοποιήση, αυτό που μένει είναι να δημιουργήσουμε τον πίνακα με τα εικονοστοιχεία που είχε η αρχική εικόνα. Αυτό μπορεί να επιτευχθεί αν πολλαπλασιάσου με την ίδια τιμή, με την οποία κάναμε την κβάντιση. Προφανώς και το αποτέλεσμα θα είναι διαφορετικό, αφού κατά την διαδικασία της κβάντισης τα δεκαδικά μέρη τα οποία είχαν προκύψει από την διαίρεση χάθηκαν, και δεν υπάρχει τρόπος ανάκτησης τους σε αυτό το στάδιο. Επομένως θα υπάρχει μια μικρή διαφορά της αρχικής εικόνας με της κβαντισμένης.

Το αποτέλεσμα της κωδικοποιήσης το αποθηκεύουμε σε ένα ένα αρχείο .txt με το όνομα ‘*apotelesmata.txt’.*

## Αποτελέσματα

Η εικόνα πριν την κβαντίση: Κβαντισμένη εικόνα:



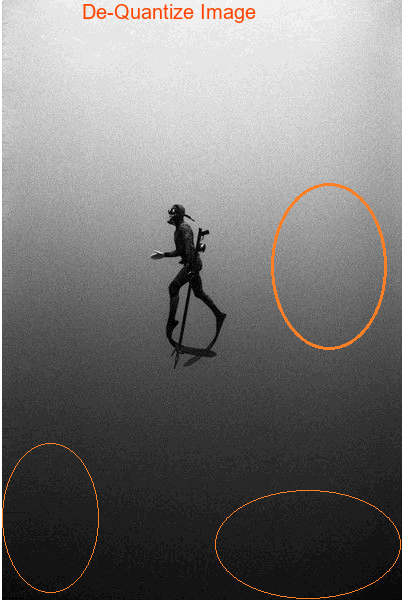


Εικόνα από την επαναφόρα της κβάντισης:



## Συμπεράσματα

Κβαντίζοντας την εικόνα ρίξαμε τα επίπεδα του γκρι, με αποτέλεσμα οι τιμές της κβαντισμένης εικόνας να είναι πιο κοντά στο μηδέν, για αυτό και έχει μειωθεί η ένταση του άσπρου. Αντιστρέφοντας την διαδικασία για να πάρουμε την αρχική εικόνα, λαμβάνουμε μια εικόνα η οποία δεν είναι ακριβώς όπως η αρχική, αλλά διαφορετική (τα σημεία έχουν κυκλωθεί στην τελική εικόνα). Αυτό συμβαίνει γιατί δεν μπόρεσαν να επανέλθουν ακριβώς οι αρχικές τιμές, αλλά με μικρή διαφορά προς τα κάτω. Στα κυκλώμενα σημεία παρατήρουμε την μεγαλύτερη διαφορά, που υπάρχει μεταξύ της αρχικής και της τελικής εικόνας. Οι κύκλοι δείχουν ότι το σημείο δεν είναι το ίδιο λείο σε σχέση με το αρχικό, παρά υπάρχει μια αισθητή διαφορά.



Τα ερωτήμα του βιβλίου!!!!

# Βιβλιογραφία και βιβλιοθήκες που χρησιμοποιήθηκαν

## 5.2 Βιβλιοθήκες Python

Στη διάρκεια της υλοποίησης της άσκησης χρησιμοποιήθηκαν γνώσεις και μέθοδοι από διάφορες έτοιμες βιβλιοθήκες της python που είναι επιστημονικού περιεχομένου και είναι φτιαγμένες για τέτοιου είδους προβλήματα.

Πιο συγκεκριμένα:

Για το ερώτημα 1 :

1)numPy

2)Pandas

3)SKLearn

4)tqdm

Για το ερώτημα 2 :

1)numPy

2)Matplot Lib

3)SciPy

4)tqdm

5)Pandas

Για το ερώτημα 3 :

1)numPy

2)Pandas

3)SKLearn (το sklearn.cluster για το Kmeans και το PCA από το sklearn.decomposition για τα γραφήματα )

4)SciPy (συγκεκριμένα απο το scipy.cluster.hierarchy και scipy.spatial.distance)

5)tqdm

Τα numPy, SciPy, SKLearn είναι βιβλιοθήκες με πιο πολύ μαθηματικές πράξεις και υλοποιήση αλγορίθμων

Το Pandas είναι για την καλύτερη οργάνωση και ανάλυση των δεδομένων μας σε Dataframes και την εύκολη μεταφορά και μετατροπή τους.

Το tqdm ([https://tqdm.github.io](https://tqdm.github.io/)) είναι για το μινιμαλιστικό user Interface και τα μυνήματα στη κονσόλα της Python για να ενημερώνουν το χρήστη για την εξέλιξη του προβλήματος.

Τέλος το MatplotLib είναι για την απεικόνιση και αναπαράσταση των γραφημάτων (όπως και στο Matlab/Ocatve)

## 5.3 Βιβλιογραφία

Είναι πολύ σημαντικό να σημειωθεί όμως ότι η έμπνευση και ο σχεδιασμός του μοντέλου BSAS υλοποιήθηκε βάση κυρίως των παραδειγμάτων από το Book-Software (συγκεκριμένα το παράδειγμα example742 αλλά και το mfiles που περιέχει τον BSAS) αλλά και από τη θεωρία και τους προτεινόμενους αλγορίθμους (ψευδοκώδικα) του βιβλίου στο Κεφάλαιο 11 (σελ. 542-548)

Το ίδιο ισχύει για την κατανόηση και επεξήγηση των αλγορίθμων K-means και Agglomerative/Ιεραρχικής ομαδοποίησης καθώς και των γραφημάτων PCA.

## 5.4 ΒΙΒΛΙΟΓΡΑΦΙΑ ΔΙΑΔΙΚΤΥΟΥ

Από το διαδίκτυο χρησιμοποιήσαμε τα εξής links και documentations για την επεξήγηση των αλγορίθμων αλλά και των χαρακτηριστικων τους:

Least Squares: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>

DataFrames:

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html>

PCA:

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

Clustering:

<https://scikit-learn.org/stable/modules/clustering.html>

Elbow method:

<https://pythonprogramminglanguage.com/kmeans-elbow-method/>

5-Fold:

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html>

Preprocessing Data:

<https://scikit-learn.org/stable/modules/preprocessing.html>