ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

Ανάκτηση περιεχομένου εικόνας με βάση την κατάτμηση

Του φοιτητή
Καραγίαννη Αντώνη
Αρ. Μητρώου: 03 2438

Επιβλέπων καθηγητής
Κωνσταντίνος Διαμαντάρας

Θεσσαλονίκη 2009

Αφιέρωση

Θα ήθελα να αφιερώσω αυτή την πτυχιακή εργασία στους γονείς μου που με την αγάπη και υποστήριξη τους όλο αυτό τον καιρό έκαναν το όνειρο μου για σπουδές στην πληροφορική πραγματικότητα. Θα ήθελα να την αφιερώσω ακόμα, στην γλυκιά μου Ruka που με την ύπαρξη της σαν όμορφη ηλιαχτίδα μου έδινε το κουράγιο να συνεχίζω να δουλεύω.

Ευχαριστίες

Ευχαριστώ τον καθηγητή μου κ. Κ. Διαμαντάρα που καθ' όλη την διάρκεια της έρευνας μου σε αυτή την πτυχιακή εργασία μου παρείχε την σωστή βοήθεια και καθοδήγηση καταφέρνοντας πάντα να με οδηγεί μακριά από εμπόδια και αδιέξοδα.

Περιεχόμενα

Κατάλογος Κώδικα

Κατάλογος Εικόνων

Πρόλογος

Τις προηγούμενες δεκαετίες οι άνθρωποι συνήθιζαν να βγάζουν φωτογραφίες με φιλμ τις οποίες τύπωναν και κρατούσαν. Τα τελευταία χρόνια όμως, η ψηφιακή τεχνολογία έχει εισβάλει στη ζωή μας. Ο αριθμός εικόνων που βρίσκονται στο διαδίκτυο καθώς και αυτών που έχει ο κάθε ένας μας στον προσωπικό του υπολογιστή είναι πολύ μεγάλος. Οι συνθήκες αυτές δημιουργούν την ολοένα αυξανόμενη ανάγκη για αποτελεσματική ταξινόμηση και αναζήτηση τους. Με γνώμονα αυτά θα προσπαθήσουμε να δημιουργήσουμε ένα μοντέλο το οποίο θα επεξεργάζεται εικόνες κάνοντας κατάτμηση και στη συνέχεια με την βοήθεια των νευρωνικών δικτύων και άλλων τεχνικών εμπνευσμένων από τον τρόπο αντίληψης του ανθρώπου, θα προσπαθεί να ανακτήσει το περιεχόμενο τους. Αναλύοντας το μοντέλο αυτό θα προσπαθήσουμε να ξεπεράσουμε τις οποιεσδήποτε δυσκολίες ώστε να πετύχουμε μια λειτουργικότητα την οποία θεωρούμε ικανοποιητική.

Κεφάλαιο 1

Εισαγωγή

1.1. Εισαγωγή

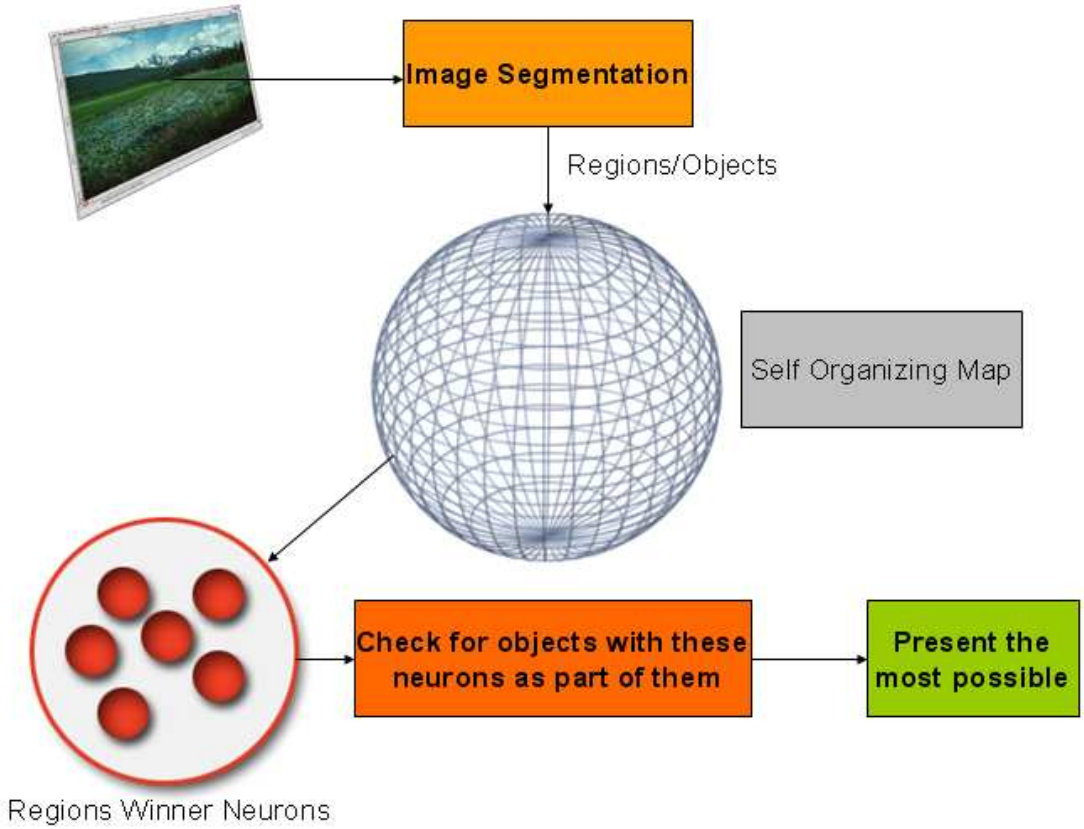
Τα τελευταία χρόνια ταυτόχρονα με την αύξηση των χρηστών του παγκόσμιου ιστού έχουμε αύξηση και στις εικόνες που φιλοξενούνται σε αυτό. Οι ανάγκες για αναζήτηση και ανάκτηση μιας εικόνας έχουν αυξηθεί και δεν μπορούν πλέον να εξαρτούνται από το όνομα που έχει το αρχείο μιας εικόνας. Λόγω αυτής της ανάγκης έχουν προταθεί πολλοί αλγόριθμοι που μπορεί πολλές φορές να χρησιμοποιούν λέξεις κλειδιά, δείγματα εικόνων, ιστογράμματα και διάφορες άλλες τεχνικές για να ανακτούν εικόνες με βάση το περιεχόμενο τους. Το πρόβλημα αυτό έχει αποδεχτεί κάθε άλλο παρά απλό, και έχει αναγκάσει μεγάλες εταιρίες όπως ο γίγαντας της αναζήτησης, Google, να ξοδέψουν πάρα πολλές ώρες έρευνας στον τομέα αυτό χωρίς να έχουν όμως ακόμα τα επιθυμητά αποτελέσματα που επιθυμούν. Αυτό δεν σημαίνει όμως ότι δεν έχει υπάρξει καθόλου πρόοδος. Όταν το πρόβλημα γίνεται συγκεκριμένο, για παράδειγμα στην αναγνώριση προσώπων ή χαρακτήρων γραφής, τότε τα βήματα προόδου που παρουσιάζονται είναι αλματώδης. Εμείς παρόλα αυτά δεν θέλουμε να παρουσιάσουμε ένα σύστημα ειδικευμένο σε κάτι αλλά ένα άλλο γενικευμένο μοντέλο που προσπαθεί να μιμηθεί την ανθρώπινη αντίληψη. Θα μελετήσουμε τις διάφορες πτυχές του προβλήματος προσπαθώντας να τις εφαρμόσουμε και να καταγράψουμε πόσο εφικτές είναι στην πράξη.

Αν λάβουμε υπόψη και τις πέντε αισθήσεις, την όραση, την οσμή, την γεύση και την αφή, αναμφισβήτητα η όραση είναι αυτή που ο άνθρωπος έχει ανάγκη περισσότερο από τις υπόλοιπες και αυτή που του παρέχει τα περισσότερα δεδομένα από αυτά που λαμβάνει. Με κάθε ματιά τα δεδομένα που ταξιδεύουν από τα μάτια προς τον εγκέφαλο δεν παρέχουν μόνο megabits από πληροφορίες αλλά πιθανότατα τα δεδομένα της συνεχούς όρασης να ξεπερνούν τα 10 megabits ανά δευτερόλεπτο. Πολλά από αυτά τα δεδομένα όπως είναι φυσικό είναι περιττά και συμπιέζονται από τον οπτικό φλοιό, ώστε τα πιο σημαντικά κέντρα του εγκεφάλου να ερμηνεύσουν ένα μικρότερο ποσοστό αυτών. Παρόλα αυτά τα δεδομένα που λαμβάνουν τα κέντρα αυτά είναι τουλάχιστον δυο φορές μεγαλύτερα από αυτά των υπολοίπων αισθήσεων[12].

Το να βλέπουμε δεν είναι μια απλή διαδικασία, αντίθετα η όραση δημιουργήθηκε μέσα από εκατομμύρια χρόνια εξέλιξης έτσι ώστε να μην υπερφορτώνει το εγκέφαλο μας με άχρηστες πληροφορίες που σε κάποια ώρα ανάγκης θα μας έκαναν πολύ αργούς στις αντιδράσεις μας. Ως αποτέλεσμα αυτού η άγνοια μας για την διαδικασία που ακολουθεί η ανθρώπινη όραση είναι ακόμα πολύ μεγάλη. Ο άνθρωπος όμως είναι εφευρετικός και τα τελευταία χρόνια χρησιμοποιεί τους υπολογιστές για τον βοηθούν. Για τις απλές εργασίες και διαδικασίες μέχρι τώρα τα καταφέρνει μια χαρά, αλλά για τις υπόλοιπες που θεωρούνται πιο πολύπλοκες οι μηχανές πρέπει να αποκτήσουν και αυτές το πλεονέκτημα της όρασης.

Και φυσικά σαν να μην έφταναν όλα τα παραπάνω, ένα ακόμα προτέρημα της ανθρώπινης όρασης είναι το ότι ο εγκέφαλος μας διαθέτει κάπου 10^{10} νευρώνες που μάλιστα μερικοί από του οποίους διατηρούν περισσότερες από 10000 συνάψεις με άλλους νευρώνες[18]. Αν υποθέσουμε ότι ένας νευρώνας συμπεριφέρεται σαν ένας μικροεπεξεργαστής τότε θα είχαμε ένα απέραντο υπολογιστή που όλα του τα στοιχεία θα μπορούν να λειτουργούν ταυτόχρονα. Οι περισσότεροι υπολογιστές σήμερα αποτελούνται από λιγότερα των 100 εκατομμυρίων επεξεργαστικών στοιχείων, πράγμα που κάνει την όραση που ο εγκέφαλος κάνει με μια ματιά να μοιάζει απίθανη να εκτελεστεί από υπολογιστή[12]. Υπάρχει ακόμα και το πρόβλημα της οργάνωσης και του προγραμματισμού αυτού του συστήματος. Ο εγκέφαλος μέσα από την εξέλιξη μπορεί να αλλάζει ακόμα και αυτό δυναμικά.

Εμείς έχοντάς αυτές τις δυσκολίες στο μυαλό θα προσπαθήσουμε να δημιουργήσουμε το μοντέλο που θεωρούμε ως ιδανικό και μετά θα κάνουμε όλες τις απαραίτητες αλλαγές ώστε αυτό να μπορεί να γίνει λειτουργικό και όσον το δυνατό πιο αξιόπιστο. Αν μπορούσαμε δηλαδή να δημιουργήσουμε ένα σύστημα που αναγνωρίζει αντικείμενα σε εικόνες με βάση τον κάποιο αλγόριθμο τότε αυτό ίσως να το θεωρούσαμε ιδανικό. Στην εικόνα και τον αλγόριθμο της επόμενης σελίδας μπορούμε να δούμε τα βασικά σημεία του ιδανικού μοντέλου όπως το φανταζόμαστε εμείς, και αμέσως μετά θα προσπαθήσουμε να τα αναλύσουμε περεταίρω.



Εικόνα 1.1: Το ιδανικό μοντέλο.

1. *Διαχωρισμός της εικόνας στις περιοχές των αντικειμένων.*
2. *Εξαγωγή δεδομένων περιοχής:*
    - *Μορφή ή σχήμα.*
    - *συσχετίσεις σε μέγεθος και απόσταση ανάμεσα στα αντικείμενα.*
    - *χρώμα περιοχής.*
3. *Εισαγωγή όλων των περιοχών στον αυτοοργανούμενο χάρτη και εύρεση νικητή νευρώνα για κάθε περιοχή.*
4. *Ανάκτηση αντικειμένου βάση τους νικητές νευρώνες και της συσχετίσεις τους.*
5. *Προβολή αντικειμένου.*

Κώδικας 1.1: Ο αλγόριθμος του ιδανικού μοντέλου.

Αυτό που κάνουμε δηλαδή είναι να περιγράφουμε μια εικόνα με βάση τα αντικείμενα της και τις συσχετίσεις τους και αμέσως μετά να ψάχνουμε πιο από τα αντικείμενα που έχουμε ήδη εξετάσει και γνωρίζουμε, έχει αυτά τα χαρακτηριστικά.

1.2. Κατάτμηση/Ανάλυση Εικόνας

Το πρώτο κομμάτι που θα προσπαθήσουμε να προσεγγίσουμε είναι αυτό της κατάτμησης της εικόνας μας. Του διαχωρισμού της δηλαδή σε κομμάτια που όλα μαζί αποτελούν τα αντικείμενα που παρουσιάζονται σε αυτήν. Το κομμάτι αυτό είναι ίσως το πιο σημαντικό καθώς τα δεδομένα που θα συλλεχθούν αποτελούν το σημείο γύρω από το οποίο θα αναπτυχθεί όλο μας το σύστημα. Λόγω του μεγάλου αριθμού εικόνων που πιθανόν να χρειαστεί να κάνουμε κατάτμηση, θα προσπαθήσουμε να αναπτύξουμε ένα αυτόνομο αλγόριθμο που θα μπορεί να εκτελεί τις κατατμήσεις στις εικόνες που θα επιλέξουμε, με ταχύτητα και χωρίς την συνεχή επίβλεψη μας.

Προσπαθώντας να καταλάβουμε τον τρόπο με τον οποίο λειτουργεί η ανθρώπινη όραση θα αναπτύξουμε τον αλγόριθμο μας που θα προσπαθήσει να ισορροπήσει ανάμεσα στην ανθρώπινη αντίληψη και στην αντικειμενικότητα του υπολογιστή κρατώντας πάντα στο μυαλό μας την ορθότητα των αποτελεσμάτων και την σταθερότητα με σεβασμό στις διάφορες παραμέτρους, και την μεγάλη ποικιλία σε εικόνες[2].

Η προσέγγιση μας βασίζεται στην ομαδοποίηση περιοχών που μοιάζουν μεταξύ τους σύμφωνα σε ένα σύνολο προκαθορισμένων κριτηρίων[4] και μεγαλώνουν επαναληπτικά συνδυάζοντας τις με μικρότερες περιοχές[1]

Τον αλγόριθμο μας για την κατάτμηση εικόνας θα τον παρουσιάσουμε αναλυτικά στο επόμενο κεφάλαιο. Τώρα θα προσπαθήσουμε να καθορίσουμε τα δεδομένα που θέλουμε να εξάγουμε από μια εικόνα ώστε να μπορούμε να κάνουμε αναζήτηση του περιεχόμενου της. Τα χαρακτηριστικά που εξάγουμε από κάθε περιοχή της εικόνας είναι:

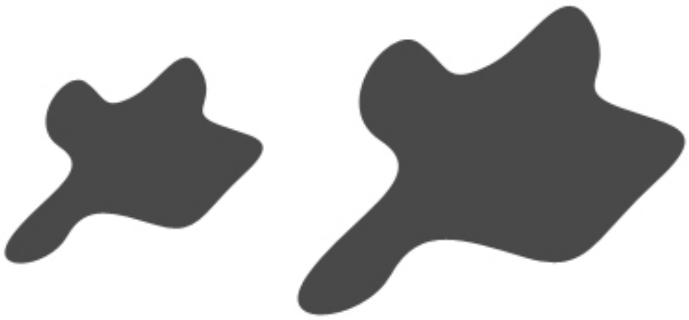
- το χρώμα
- η μορφή

- η τοποθεσία τους
- το μέγεθος

Το μέγεθος και η τοποθεσία μιας περιοχής βρίσκετε πάντα σε σχέση με την εικόνα.

Το χρώμα αναφέρεται στην RGB τιμή που έχει περιοχή μετά την κατάτμηση. Επειδή ο αλγόριθμος μας λειτουργεί βάση την ομοιογένεια του χρώματος που υπάρχει σε κάθε περιοχή στην ουσία η RGB τιμή του, είναι η μέση τιμή σε κάθε κανάλι χρώματος που την υπολογίζουμε από όλα τα pixels αυτής.

Όταν αναφερόμαστε στην μορφή μιας περιοχής εννοούμε το σχήμα της. Εξάγουμε όλες τις περιοχές και κρατάμε την μορφή τους στις διαστάσεις που εμείς θα ορίσουμε μέσα από τον αλγόριθμο μας. Αυτό το κάνουμε για αποκλείσουμε ότι ένα σχήμα που στην ουσία είναι το ίδιο με κάποιο άλλο, φαίνετε ως διαφορετικό λόγω του διαφορετικού μεγέθους που μπορεί να έχει.



Εικόνα 1.2: Ίδιο σχήμα με διαφορετικό μέγεθος.

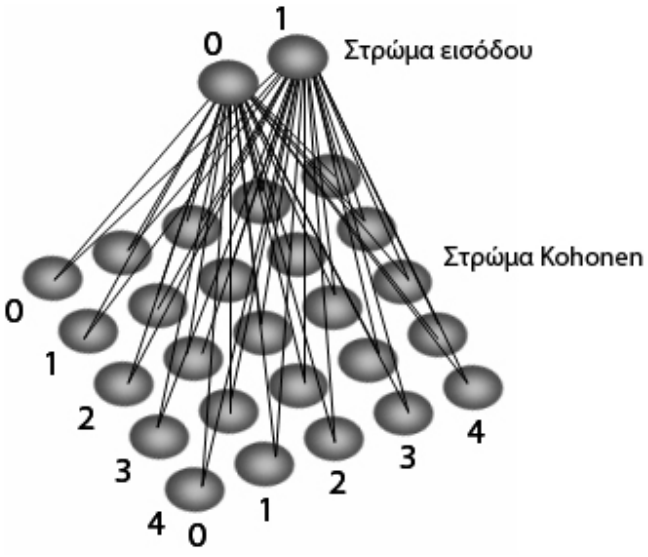
Όπως για παράδειγμα φαίνεται και στην εικόνα πιο πάνω έχουμε τα ίδια σχήματα με διαφορετικό μέγεθος. Αν αυτά τα φυλάσσαμε με το μέγεθος τους σε pixels όπως καταγράφετε σε μια εικόνα τότε θα αποτύχουμε να αναγνωρίσουμε ότι στην ουσία πρόκειται για την ίδια μορφή.

Τα τελευταία δύο χαρακτηριστικά που εξάγουμε είναι το μέγεθος και η τοποθεσία που έχουν τα αντικείμενα. Και τα δύο αυτά χαρακτηριστικά για μια περιοχή είναι πάντα σε σχέση με τις υπόλοιπες περιοχές της εικόνας. Δηλαδή το μέγεθος μιας περιοχή θα το υπολογίσουμε σε σχέση με μια άλλη, δηλαδή για παράδειγμα μια περιοχή έχει μέγεθος ½ σε σχέση με κάποια άλλη και απόσταση από αυτήν στα ¾ του μεγέθους της. Αρχικά τα χαρακτηριστικά αυτά κατά την κατάτμηση θα υπολογίζονται ως προς το μέγεθος της εικόνας και αργότερα,

ανάλογα με τις ανάγκες του αλγόριθμου μας θα υπολογίζονται σε σχέση με τις υπόλοιπες περιοχές της εικόνας.

1.3. Αυτοοργανούμενος Χάρτης (Self Organizing Map)

Μετά την κατάτμηση θα πρέπει να εισάγουμε τα δεδομένα μας μέσα σε ένα αυτοοργανούμενο χάρτη. Οι αυτοοργανούμενοι χάρτες είναι ένα εργαλείο που μπορεί να ταξινομεί πολύπλοκα δεδομένα ανάλογα με τις συσχετίσεις που παρουσιάζουν μεταξύ τους. Έχουμε επιλέξει αυτό τον τύπο νευρωνικού δικτύου λόγω της ικανότητας του να προσαρμόζεται στα δεδομένα που λαμβάνει χωρίς να έχει κάποια επίβλεψη. Αυτό που θα θέλαμε είναι να κατηγοριοποιήσουμε τις περιοχές μας και να προσδιορίσουμε ένα αρμόδιο νευρώνα για κάθε περιοχή που μπορεί να προκύψει. Αυτός ο τύπος δικτύου παρέχει μια μέθοδο εκπαίδευσης χωρίς επίβλεψη που είναι γνωστή ως ανταγωνιστική μάθηση, και είναι ιδιαίτερα χρήσιμη όταν γίνεται ανάλυση δεδομένων ανάμεσα στα οποία δεν είναι γνωστές οι συσχετίσεις[16]. Σε αυτό το δίκτυο δεν υπάρχουν στόχοι για να επιτευχθούν, όπως επίσης δεν υπάρχει σφάλμα για να ελαχιστοποιηθεί. Αυτά είναι και τα χαρακτηριστικά που το κάνουν να ξεχωρίζει από τις μεθόδους που λειτουργούν με επιβλεπόμενη μάθηση[16].



Εικόνα 1.3: Τα στρώματα εισόδου και Kohonen στο δίκτυο SOM

1.4 Ανάκτηση και προβολή αντικειμένου

Μετά την εισαγωγή των περιοχών στον αυτοοργανούμενο χάρτη έχουμε όλους τους νικητές νευρώνες. Όπως είπαμε ο κάθε νευρώνας ειδικεύεται στο να αντιπροσωπεύει χρώμα και το σχήμα μιας περιοχής. Σε αυτό το κομμάτι πρέπει να βρούμε τις συσχετίσεις που έχουν οι νευρώνες μεταξύ τους. Ας θεωρήσουμε την παρακάτω εικόνα:

Εικόνα 1.4: Ιδανικό μοντέλο. (αριστερά: εικόνα, δεξιά: περιοχές μοντέλου).

Αφού διαχωρίσουμε την εικόνα του Duffy Duck στις περιοχές που καταλαμβάνουν τα αντικείμενα που βλέπουμε πιο πάνω, παίρνουμε για παράδειγμα τους νικητές νευρώνες για τις περιοχές 1 και 2. Αν θέλουμε να βρούμε τις συσχετίσεις τους πρέπει να παρατηρήσουμε ότι, το σχήμα τους είναι το περίγραμμα τους, το μέγεθος του 2 σε σχέση με το 1 να είναι περίπου στο 1/3, η απόσταση του 2 από το κέντρο του 1 περίπου είναι μια φορά το μέγεθος του 2 και το χρώμα τους είναι να μαύρο και πορτοκαλί αντίστοιχα. Κάνοντας την παραδοχή ότι μια εικόνα θα έχει παρόμοιους νικητές νευρώνες με παρόμοιες συσχετίσεις, ψάχνω να βρω σε πιο από τα αντικείμενα που έχουμε εξετάσει κατά την εκπαίδευση του μοντέλου έχει τα περισσότερα κοινά στοιχεία για να το παρουσιάσω.

1.5 Τι έπεται

Η ανάπτυξη ενός συστήματος με τέτοια πολυπλοκότητα σίγουρα θα επιφέρει αρκετά απρόβλεπτα αποτελέσματα. Για αυτό τον λόγο, τώρα που είδαμε σε γενικές γραμμές το ιδανικό μας μοντέλο, θα προσπαθήσουμε αρχικά να το αναπτύξουμε χρησιμοποιώντας την γλώσσα προγραμματισμού Java. Στην συνέχεια θα δούμε τις ανάγκες και τα προβλήματα που θα προκύψουν και θα προσπαθήσουμε τελικά να φτιάξουμε ένα λειτουργικό μοντέλο που θα είναι όσο πιο κοντά γίνεται στο ιδανικό.

Στο δεύτερο κεφάλαιο θα παρουσιάσουμε ενδελεχώς την κατάτμηση εικόνας. Παρουσιάζουμε τη σημασιολογία και το μοντέλο μιας εικόνας και κάνουμε μία θεωρητική ανάλυση στον αλγόριθμο κατάτμησης που θα χρησιμοποιήσουμε, τον Split n’ Merge. Αφού αναλύσουμε όλες τις θεωρητικές πτυχές του αλγόριθμού μας, υλοποιούμε το πρόγραμμα κατάτμησης και παρουσιάζουμε κάποια πειραματικά αποτελέσματα.

Στο τρίτο κεφάλαιο παρουσιάζουμε τους αυτοοργανούμενους χάρτες. Κάνουμε μια αναδρομή στα νευρωνικά δίκτυα και αναλύουμε τις θεωρητικές πτυχές των αυτοοργανούμενων χαρτών. Στη συνέχεια προσαρμόζουμε τον τύπο αυτό νευρωνικού δικτύου στο μοντέλο μας και υλοποιούμε βήμα-βήμα τους αλγόριθμους εκπαίδευσης και ανάκλησης.

Στο τέταρτο κεφάλαιο εξετάζουμε την ανθρώπινη μνήμη και αντίληψη αντικειμένων. Με βάση αυτές δημιουργούμε τις συσχετίσεις που αποτελούν τις αναμνήσεις και υλοποιούμε τον κώδικα εκπαίδευσης και ανάκλησης τους.

Στο πέμπτο κεφάλαιο δοκιμάζουμε το ιδανικό και αξιολογούμε τα αποτελέσματα για να προκύψει στη συνέχεια το μοντέλο συσχετίσεων και τελικά τελικό μας μοντέλο που είναι το πιο λειτουργικό.

Στο έκτο κεφάλαιο εξετάσουμε μερικά χαρακτηριστικά που θα ήταν καλό να προσαρτηθούν στο μοντέλο μας. Εξετάζουμε τρόπους βελτίωσης της κατάτμησης, τα γραφικά vector, τρόπους βελτίωσης των συσχετίσεων και την σχετική ανατροφοδότηση.

Κεφάλαιο 2

Κατάτμηση Εικόνας

2.1. Εισαγωγή

Όπως καθορίζεται στην ψυχολογία μετά το κίνημα του Gestalt η ικανότητα της ομαδοποίησης αντικειμένων παίζει ένα πολύ σημαντικό ρόλο στην ανθρώπινη αντίληψη[1]. Αν και στον άνθρωπο αυτή η ομαδοποίηση έχει ένα βασικό και σημαντικό χαρακτήρα, στον τομέα των υπολογιστών και ιδιαίτερα στον τομέα της ανάλυσης εικόνας παραμένει ένα αγκάθι προς το οποίο τα τελευταία χρόνια γίνονται βήματα βελτίωσης. Η όραση, όπως είναι ευρέως αποδεκτό είναι ένα πρόβλημα λογικής που βασίζεται σε συμπεράσματα. Με αυτή την αποδοχή το πρόβλημα της ομαδοποίησης μπορεί να αντιστοιχηθεί με την μετατροπή των pixels μιας εικόνας σε οπτικά ουσιαστικές περιοχές και αντικείμενα[1].

Ένας αλγόριθμος κατάτμησης θα πρέπει να καταφέρνει να ισορροπεί ανάμεσα στην ανθρώπινη αντίληψη και στην αντικειμενικότητα του υπολογιστή πληρώντας τα παρακάτω[2]:

1. *Ορθότητα*: Η ικανότητα παραγωγής κατατμήσεων που συμφωνούν με την ανθρώπινη αντίληψη.
2. *Σταθερότητα με σεβασμό στις επιλογές των παραμέτρων*: Η ικανότητα σταθερής κατάτμησης με την χρήση ενός εύρους παραμέτρων.
3. *Σταθερότητα με σεβασμό στην μεγάλη ποικιλία εικόνας*: Η ικανότητα παραγωγής κατατμήσεων με σταθερή ορθότητα μέσα από ποικιλία εικόνων με διαφορετικά αντικείμενα.

Υπάρχει ένας σημαντικός αριθμός μεθόδων κατάτμησης της εικόνας σε ομάδες[3] ο οποίος συνήθως εστιάζει σε κάτι και διαφέρει ανάλογα με το πρόβλημα προς επίλυση. Το δυσκολότερο έργο απ’ όλα όμως είναι αυτό της κατάτμησης και ομαδοποίησης αντικειμένων με χαρακτηριστικά τα οποία δεν καθορίζονται ευκρινώς η θεωρούνται μη σημαντικά δηλαδή, τις εικόνες που περιέχουν τυχαία αντικείμενα και λαμβάνονται κάτω από μη ελεγχόμενο περιβάλλον. Σε αυτού του τύπου την κατάτμηση θα επικεντρωθούμε και εμείς.

Οι αλγόριθμοι κατάτμησης γενικά βασίζονται σε ένα από τα δύο βασικά χαρακτηριστικά στοιχεία έντασης μιας περιοχής της εικόνας, την ασυνέχεια και την ομοιότητα[4]. Στην πρώτη κατηγορία η ομαδοποίηση βασίζετε στις απότομες

αλλαγές έντασης που παρουσιάζονται όπως είναι αυτές στα άκρα ενός αντικειμένου. Η δεύτερη προσέγγιση βασίζεται στην ομαδοποίηση περιοχών που μοιάζουν σύμφωνες με ένα σύνολο προκαθορισμένων κριτηρίων[4] οι οποίες μεγαλώνουν επαναληπτικά συνδυάζοντας μικρότερες περιοχές η pixels, pixels τα οποία από μόνα τους αποτελούν μια στοιχειώδες περιοχή[1].

Σε αυτό το κεφάλαιο εμείς θα εξετάσουμε μια στρατηγική η οποία βασίζεται στην δεύτερη κατηγορία και συγκεκριμένα στους αλγόριθμους διεύρυνσης και συγχώνευσης περιοχών. Οι αλγόριθμοι διεύρυνσης και συγχώνευσης περιοχών, όπως έχουμε αναφέρει και πιο πάνω για όλες τις τεχνικές που βασίζονται στην ομαδοποίηση με ομοιότητα, χρησιμοποιούν για την διεύρυνση τους προκαθορισμένα κριτήρια, ένα τεστ δηλαδή για να αποφασίσουν την συγχώνευση περιοχών. Ένα κατηγόρημα σύγκρισης χρησιμοποιεί αυτά τα κριτήρια και έτσι κτίζει την κατατετμημένη σε ομάδες εικόνα στην βάση σχεδόν πάντα τοπικών αποφάσεων[1].

Στόχος μας είναι η δημιουργία ενός μοντέλου το οποίο θα δημιουργεί κατατετμημένες εικόνες και θα παρέχει όλες τις θεωρητικές και πρακτικές προσεγγίσεις που παρέχει ένα αξιόπιστο σύστημα κατάτμησης εικόνας, λαμβάνοντας υπόψη και την επέκταση του στην λειτουργία ενός συστήματος αναζήτησης εικόνας βάση του περιεχομένου της.

2.2. Η Σημασιολογία και το Μοντέλο μιας εικόνας

Μια παρατηρούμενη εικόνα ***I*** περιέχει |***I***| pixels, το κάθε ένα από τα οποία περιέχει μεταβλητές για το Κόκκινο-Πράσινο-Μπλε (GRB, Red-Green-Blue). Κάθε μια από αυτές τις μεταβλητές μπορεί να πάρει τιμές από το σύνολο { 1, 2, 3,…,g}, και στην πράξη έχουμε g=256[1].

Το ***I*** είναι μια τέλεια παρατήρηση ενός σκηνικού ***I**** που ξέρουμε. Σε ένα ***I**** οι περιοχές αναπαρίστανται από θεωρητικά αντικείμενα που μοιράζονται τις ίδιες ιδιότητες ομοιογένειας:

- Μέσα σε κάθε περιοχή και κάθε δεδομένο κανάλι χρώματος που ανήκει στο {R, G, B}, όλα τα pixels έχουν την ίδια απόκλιση από το κανάλι χρώματος.

- Σε γειτονικές περιοχές τα pixels είναι διαφορετικά τουλάχιστον σε ένα κανάλι χρώματος του {R, G, B}.

Το ***I*** το ανακτούμε από το ***I**** παίρνοντας ένα δείγμα για κάθε pixel και τη τιμή RGB που θα πάρει.

2.3. Θεωρητική Ανάλυση και αλγόριθμοι

2.3.1 Κριτήριο συγχώνευσης

Γενικά, το κριτήριο συγχώνευσης μπορεί να εντοπίζει την ομοιογένεια μιας περιοχής βασιζόμενο στην ένταση, τις μεγάλες εναλλαγές, το χρώμα, την υφή, την κίνηση, την μορφή και το μέγεθος[5][6]. Στην προσέγγιση που ακολουθούμε εμείς εξετάζουμε το χρώμα και τις μεγάλες εναλλαγές που δημιουργούν αντιθέσεις. Το δεύτερο το παρουσιάζουμε αμέσως μετά στην ενότητα 3.2.

Όπως αναφέραμε πιο πάνω, για να μπορούμε να πούμε ότι ένα pixel ανήκει σε μια περιοχή πρέπει να έχει την ίδια απόκλιση σε κάθε κανάλι χρώματος της. Εμείς μέσα στον αλγόριθμο μας αυτή την απόκλιση θα μπορούμε να την καθορίσουμε μέσω μιας ανεκτικότητας (tolerance) η οποία όταν ξεπεραστεί τότε το συγκεκριμένο pixel δεν θα συγχωνεύεται με την περιοχή. Τα κανάλια χρώματος της περιοχής θα το ορίσουμε ως τον μέσο όρο των καναλιών των pixels που περιέχει κάθε περιοχή. Δηλαδή για κάθε κανάλι χρώματος θα πρέπει να ισχύει:

$$|pixels(x,y,r) - regionAvg(r)| < tolerance$$
$$|pixels(x,y,g) - regionAvg(g)| < tolerance$$
$$|pixels(x,y,b) - regionAvg(b)| < tolerance$$

2.3.2 Αντιμετώπιση μεγάλων αντιθέσεων

Σε περιοχές όπου παρουσιάζεται μεγάλη αντίθεση όπως είναι για παράδειγμα το γρασίδι παρουσιάζονται πάρα πολλές μικρές περιοχές που για να καταφέρουμε να τις συγχωνεύσουμε χρειαζόμαστε μεγάλη ανεκτικότητα πράγμα που μπορεί να μας οδηγήσει σε μια ανεπιθύμητη υπερσυγχώνευση. Για να αντιμετωπίσουμε το φαινόμενο αυτό εισάγαμε στον αλγόριθμο μας μια

μεταβλητή, την bumpEffect. Στόχος μας δηλαδή είναι να ισορροπήσουμε μεταξύ των μικρών τοπικών αντιθέσεων και της συνολικής περιοχής όπως την αντιλαμβάνεται το ανθρώπινο μυαλό[7].

Εικόνα 2.1: Κατάτμηση χωρίς, και με ανεκτικότητα στις μικρές περιοχές.

Στην εικόνα 1, κάνουμε κατάτμηση της φωτογραφίας στα αριστερά. Στην μέση παρουσιάζεται κατάτμηση χωρίς ανεκτικότητα στις μεγάλες αντιθέσεις, ενώ στα δεξιά κάνουμε χρήση της bumpEffect. Η κατάτμηση στα δεξιά βλέπουμε ότι μας δίνει ένα σαφώς μικρότερο αριθμό μικρών περιοχών. Με άλλα λόγια η bumpEffect μας βοηθά να αυξάνουμε την ανεκτικότητα αν η περιοχή που εξετάζουμε είναι αρκετά μικρή. Δηλαδή σύμφωνα με τον τύπο που χρησιμοποιούμε ισχύει:

tolerance = tolerance + (bumpEffect - (regionSize*bumpEffect))

Αυτό που κάνουμε είναι να προσθέτουμε στην ανεκτικότητα μας ένα ποσοστό της μεταβλητής bumpEffect που εξαρτάται από το μέγεθος της περιοχής μας. Το regionSize όπως φαίνεται ποιο πάνω καθορίζει το ποσοστό αυτό υπολογίζοντας την αναλογία της περιοχής μας μέσα στην εικόνα με μέγιστο ποσοστό βοήθειας στο 25% ολόκληρης της εικόνας.

regionSize=elements/(w*h*4)

2.4. Ο αλγόριθμος μας: Split n' Merge

Δεν μπορεί κανείς να αρνηθεί ότι η επίτευξη ενός καλού αλγόριθμου κατάτμησης δεν εξαρτάται μόνο από την σωστή λειτουργία του. Οι καλοί αλγόριθμοι κατάτμησης κρίνονται εκτός από την λειτουργία τους, για την ταχύτητα επίτευξης των αποτελεσμάτων τους[8]. Ο αλγόριθμος που χρησιμοποιούμε είναι σχετικά απλός και γρήγορος. Ξεκινάμε υποθέτοντάς ότι κάθε pixel από μόνο του αποτελεί μια περιοχή. Παίρνοντας τα pixels αυτά λοιπόν ένα ένα εξετάζουμε με το κριτήριο συγχώνευσης αν τα γειτονικά pixels ανήκουν στην ίδια περιοχή και αναλόγως τα συγχωνεύουμε. Μετά το τέλος της διαδικασίας αυτής παίρνουμε τις τελικές περιοχές που δημιουργήθηκαν και εξετάζουμε αν υπάρχει μια δυνατότητα συγχώνευσης ανάμεσα τους. Παρακάτω παρουσιάζονται οι βασικές αρχές του αλγόριθμου σε ψευδοκώδικα.

```
1. Όσο w<ImageWidth, w++
2.     Όσο h<ImageHeight, h++
3.         if( ! isInSegment(w,h)
               createNewRegion
4.         else insertPixelToRegion
5.         if(isInSegment(w-1,regionStart,RegionEnd)
               MergeWhithRegionAbove
6. Οσο region1<FinalRegions.lengh region1++
7.     Οσο region2<FinalRegions.lengh region2++
8.         if(region1~region2 & isInSegment(region1,region2))
9.             mergeSegments(region1,region2)
```

Κώδικας 2.1: Ο αλγόριθμος της Split n' Merge.

2.4.1 Ο αλγόριθμος μας σε σύγκριση με τον αλγόριθμο Region Growing

Ο αλγόριθμος που παρουσιάζουμε είναι στην ουσία μια παραλλαγή ενός αλγορίθμου Region Growing. Οι αλγόριθμοι Region Growing παίρνουν μια περιοχή και την μεγαλώνουν συγχωνεύοντας σε αυτήν pixels που πληρούν το κριτήριο συγχώνευσης. Πολλές φορές οι αλγόριθμοι αυτής της κατηγορίας

χρησιμοποιούν την τεχνική των σπόρων. Στην τεχνική των σπόρων η κατάτμηση μέσα στην εικόνα μας ξεκινά από βασικά σημεία(σπόρους) και συνεχίζει την επέκταση στις περιοχές γύρω τους. Οι σπόροι επιλέγονται με κριτήρια που αρμόζουν στο πρόβλημα μας. Την δική μας υλοποίηση του αλγόριθμου Region Growing την αποκαλούμε Split n' Merge λόγω του ότι αρχικά χωρίζει την εικόνα σε περιοχές που αντιστοιχούν μια για κάθε pixel, και αμέσως μετά τις επεκτείνει κάνοντας συγχώνευση των pixels που είναι συμβατά μεταξύ τους.

Ο αλγόριθμος Region Growing όπως παρουσιάζεται στο βιβλίο Digital image Processing των Rafael C. Gonzalez και Richard E. Woods[4] έχει ως εξής:

1. $\bigcup_{i=1}^{n} R_i = R$
2. *R_i είναι μια συνδεδεμένη περιοχή, i = 1,2,3,…,n*
3. *$R_i \cap Rj = \emptyset$ για i και j, I ≠ j*
4. *$P(R_i)$ = TRUE για i = 1,2,3,…,n*
5. *$P(R_i \cup Rj)$ = FALSE για I ≠ j*

Το R αναπαριστά ολόκληρη την περιοχή που καταλαμβάνει η εικόνα. Η διαδικασία κατάτμησης είναι αυτή που χωρίζει το R σε n υποπεριοχές, $R_1, R_2, \ldots, R_n$, έτσι ώστε να πληρούνται οι παραπάνω συνθήκες. Στο (4) και (5) το P είναι μια συνάρτηση που μας επιστρέφει TRUE ή FALSE.

Η συνθήκη (1) μας υποδεικνύει ότι για να τελειώσει η κατάτμηση πρέπει κάθε pixel να ανήκει σε κάποια περιοχή. Η συνθήκη (2) μας υποδεικνύει ότι τα pixels σε μια περιοχή είναι γειτονικά και έχουν κάποια κοινά στοιχεία όπως το χρώμα. Η συνθήκη (3) μας λέει ότι οι περιοχές δεν πρέπει να έχουν κοινά σημεία. Η συνθήκη (4) υποδεικνύει τις ιδιότητες που πρέπει να πληρούν τα pixels σε μια περιοχή. Τέλος η συνθήκη (5) μας δίνει τις διαφορές που έχουν οι περιοχές *R_i και Rj σε σχέση με το P.*

Στο [4] επίσης αναφέρεται, ότι όπως το υπονοεί και το όνομα, Region Growing (Επέκταση Περιοχής) είναι η διαδικασία της ομαδοποίησης των pixels ή υποπεριοχών σε μεγαλύτερες περιοχές σύμφωνα με προκαθορισμένα κριτήρια. Όταν το πρόβλημα είναι συγκεκριμένο η βασική προσέγγιση που ακολουθείται συνήθως είναι, αυτή των σπόρων που αρμόζουν στις ιδιότητες μας[4].

Ο αλγόριθμος μας βασίζεται σε μεγάλο βαθμό στον αλγόριθμο του [4] αλλά λόγω της μη καθορισμένης και μη συγκεκριμένης φύσης της κατάτμησης που υλοποιούμε έχει την διαφορά ότι δεν χρησιμοποιούμε την τεχνική των σπόρων. Επειδή δεν υπάρχουν πληροφορίες για το πρόβλημα μας, κάθε pixel αντιμετωπίζεται με τα ίδια κριτήρια που θα το οδηγήσουν όσο προχώρα η διαδικασία σε μια περιοχή χωρίς να επιλέγονται αρχικά κάποιοι σπόροι. Στα υπόλοιπα σημεία που υποδεικνύουν οι συνθήκες που αναφέραμε πιο πάνω δεν έχουμε καμιά διαφοροποίηση. Την συνθήκη (4) υλοποίει το κριτήριο συγχώνευσης ανάλογα με την απόκλιση που έχει το κάθε pixel από το κάθε κανάλι χρώματος της περιοχής. Εδώ μπαίνει μία ακόμα διαφορά στον αλγόριθμο μας. Το κριτήριο συγχώνευσής μας μπορεί να γίνεται μεταβλητό και να έχει την ικανότητα να αντιμετωπίζει μεγάλες αντιθέσεις οι οποίες παρουσιάζονται σε κάποιες περιοχές. Το κριτήριο συγχώνευσης, την αντιμετώπιση μεγάλων αντιθέσεων και τον αλγόριθμο μας συνολικά τα παρουσιάζουμε στην ενότητα 3.1, 3.2 και 4 αντίστοιχα.

2.5. Υλοποίηση Κώδικα σε Java

Το ονομα της κλάσης που υλοποιεί την κατάτμηση σε Java είναι **SplitAndMerge**. Κάνουμε χρήση του πίνακα **imageArray** ο οποίο κράτα τα δεδομένα της εικόνας σε τρισδιάστατη μορφη (ύψος,πλατος,κανάλια) και του πίνακα regCheck που κρατά τον αριθμό της περιοχής που ανήκει κάθε pixel. Οι μεταβλητές **tolerance, regionNo, width, height, bands, bumpEffect** κρατάνε αντίστοιχα την ανεκτικότητα, αριθμό περιοχής, πλάτος και ύψος της εικόνας, αριθμός καναλιών και την ανεκτικότητα σε μεγάλες αντιθέσεις. Γίνετε επίσης χρήση της κλάσης **Regions regionsAvg** που κρατά και υπολογίζει πληροφορίες όπως το χρώμα κάθε περιοχής και τον αριθμό των pixels που βρίσκονται σε αυτήν. Πριν παρουσιάσουμε ολόκληρο τον κώδικα κατάτμησης, παρουσιάζουμε μερικές μεθόδους οι λειτουργίες των οποίων ειναι αρκετα σημαντικές. Στην λειτουργικότητα της κλάσης Regions δεν θα επεκταθούμε γιατί θεωρούμε ότι οι λειτουργίες της είναι πιο κοντά σε αυτές των δομών δεδομένων παρά στον αλγόριθμο μας.

2.5.1. Κριτήριο συγχώνευσης για τα pixels.

Για να καθορίσουμε αν ένα pixel μπορει να συγχωνευθεί με μια περιοχή χρησημοποιούμε την boolean μέθοδο **isInSegment** που φαίνετε πιο κάτω.

```
private  boolean isInSegment(int w,int h,int regionNo){
    double diffr; double diffg; double diffb;
    double []regAvg=regionsAvg.getRegion(regionNo);
    int elements=(int)regAvg[3];
    double regionSize=elements/(width*height*4);
    double bEffect=bumpEffect-(regionSize*bumpEffect);

    if(regCheck[w][h]==-1 || regionNo==-1)
        return false;
      else{
        diffr=Math.abs(imageArray[w][h][0] - regAvg[0]);
        diffg=Math.abs(imageArray[w][h][1] - regAvg[1]);
        diffb=Math.abs(imageArray[w][h][2] - regAvg[2]);
        if(diffr<tolerance+bEffect &&
          diffg<tolerance+bEffect &&
          diffb<tolerance+bEffect)
           return true;
        else
           return false;
      }}
```

Κώδικας 2.2: Μέθοδος isInSegment, κριτήριο συγχώνευσης pixels.

Η **isInSegment** δέχεται το w και h συμβολίζουν το πλάτος και ύψος όπου βρίσκεται το pixel που θέλουμε να ελέγξουμε αν βρίσκετε στην περιοχή regionNo. Αρχικά ανακτούμε τον αριθμό στοιχείων που περιέχονται μέχρι τώρα στην περιοχή μας και υπολογίζουμε το μέγεθος του bumpEffect που θα προστεθεί στο tolerance. Όπως έχουμε προβλέψει, στα pixels τα οποία βρίσκονται στο τέλος μιας περιοχής δίνουμε τον αριθμό περιοχής -1 για να μπορούμε να κρατήσουμε το περίγραμμά της ξεχωριστό. Επειδή τώρα τα περιγράμματα όλων των

περιοχών έχουν τον ίδιο αριθμό, το -1, πρέπει να σιγουρευτούμε με τον παρακάτω έλεγχο ότι τα pixels που βρίσκονται δίπλα σε αυτά δεν θα συγχωνευθούν μαζί τους.

```
if(regCheck[w][h]==-1 || regionNo==-1)
        return false;
```

Αμέσως μετά υπολογίζουμε την διαφορά που έχει το κανάλι χώματος του pixel από τον μέσο όρο αυτού της περιοχής και αν αυτό είναι μικρότερο από την τελική τιμή της ανεκτικότητας τοτε η μέθοδος μας επιστρέφει true για να μπορεί να ανάψει το πράσινο φως στην συγχώνευση. Διαφορετικά σε κάθε άλλη περίπτωση επιστρέφει false.

2.5.2. Κριτήριο συγχώνευσης μεταξύ περιοχών.

Το κριτήριο συγχώνευσης μεταξύ περιοχών έχει την ίδια λειτουργία με αυτή του κριτηρίου συγχώνευσης μεταξύ των pixels. Η κύρια διαφορά του όπως φαίνετε πιο κάτω, είναι ότι στην είσοδο δέχεται τον αριθμό των δύο περιοχών για τις οποίες θα γίνει ο έλεγχος.

```
private  boolean isInSegment(int regionNo1,int regionNo2){
    if(regionNo1==-1 || regionNo2==-1)
       return false;
    double diffr; double diffg; double diffb;
    double []regAvg1=regionsAvg.getRegion(regionNo1);
    double []regAvg2=regionsAvg.getRegion(regionNo2);

    int elements=(int)regAvg1[3];
    double regionSize=elements/(width*height);
    double bEffect1=bumpEffect-(regionSize*bumpEffect);
    elements=(int)regAvg2[3];
    regionSize=elements/(width*height*4);
    double bEffect2=bumpEffect-(regionSize*bumpEffect);
    double bEffect=(bEffect1+bEffect2)/2;

    diffr=Math.abs(regAvg1[0] - regAvg2[0]);
    diffg=Math.abs(regAvg1[1] - regAvg2[1]);
```

```
        diffb=Math.abs(regAvg1[2] - regAvg2[2]);
        if(diffr<tolerance+bEffect &&
                diffg<tolerance+bEffect &&
                diffb<tolerance+bEffect)
            return true;
        else
            return false;
    }
```

Κώδικας 2.3: Μέθοδος isInSegment, κριτήριο συγχώνευσης περιοχών.

Πρώτα απ' όλα ξεκινάμε κάνουμε ένα έλεγχο για να βεβαιωθούμε ότι οι δύο περιοχές δεν ανήκουν στο περίγραμμα. Δήλαδη ότι μια από τις περιοχές δεν έχει τιμή -1. Μετά τον έλεγχο, αν καμία περιοχή δεν ανήκει στο περίγραμμα, τότε ανακτούμε την μέση τιμή κάθε καναλιού χρώματος και για τις δύο περιοχές. Και τις τοποθετούμε στους πίνακες regAvg1 και regAvg2.

```
double []regAvg1=regionsAvg.getRegion(regionNo1);
double []regAvg2=regionsAvg.getRegion(regionNo2);
```

Η ανεκτικότητα στις αντιθέσεις στην περίπτωση αυτή που έχουμε δύο περιοχές, υπολογίζεται με τον ίδιο ακριβώς τρόπο και παίρνουμε τον μέσο όρο των δυο.

```
double bEffect=(bEffect1+bEffect2)/2;
```

Τέλος, υπολογίζουμε την διαφορά τους σε κάθε κανάλι χρώματος και αν αυτή είναι μικρότερη από την ανεκτικότητα μας τότε μπορεί να γίνει η συγχώνευση.

```
if(diffr<tolerance+bEffect &&  diffg<tolerance+bEffect && diffb<tolerance+bEffect)
        return true;
```

2.5.3. Συγχώνευση περιοχών.

Τέλος πριν δούμε την μέθοδο κατάτμησης θα εξετάσουμε την μέθοδο συγχώνευσης. Η μέθοδος που υλοποιεί την συγχώνευση είναι σχετικά απλή. Παίρνει ως είσοδο τον αριθμό δύο περιοχών region1 και region2. Πρώτα θα καλέσει την μέθοδο mergeRegions της κλάσης Regions η οποία υπολογίζει τον νέο μέσο όρο για κάθε κανάλι χρώματος στην συγχωνευμένη περιοχή και αφαιρει την δεύτερη περιοχή που πια δεν χρησιμοποιείται, από την μνήμη. Μετά την

κλήση της μεθόδου αυτής ελέγχουμε όλα pixels της εικόνας μας και σε όσα έχουν αριθμό περιοχής αυτό της region2 δίνουμε αριθμό περιοχής αυτό της region1. Ο κώδικας φαίνεται πιο κάτω.

```
private void mergeSegments(int region1,int region2){
    regionsAvg.mergeRegions(region1, region2);
    for(int i=0; i<width; i++)
      for(int j=0; j<height; j++)
        if(regCheck[i][j]==region2) regCheck[i][j]=region1;
  }
```

Κώδικας 2.4: Μέθοδος mergeSegments για συγχώνευση περιοχών.

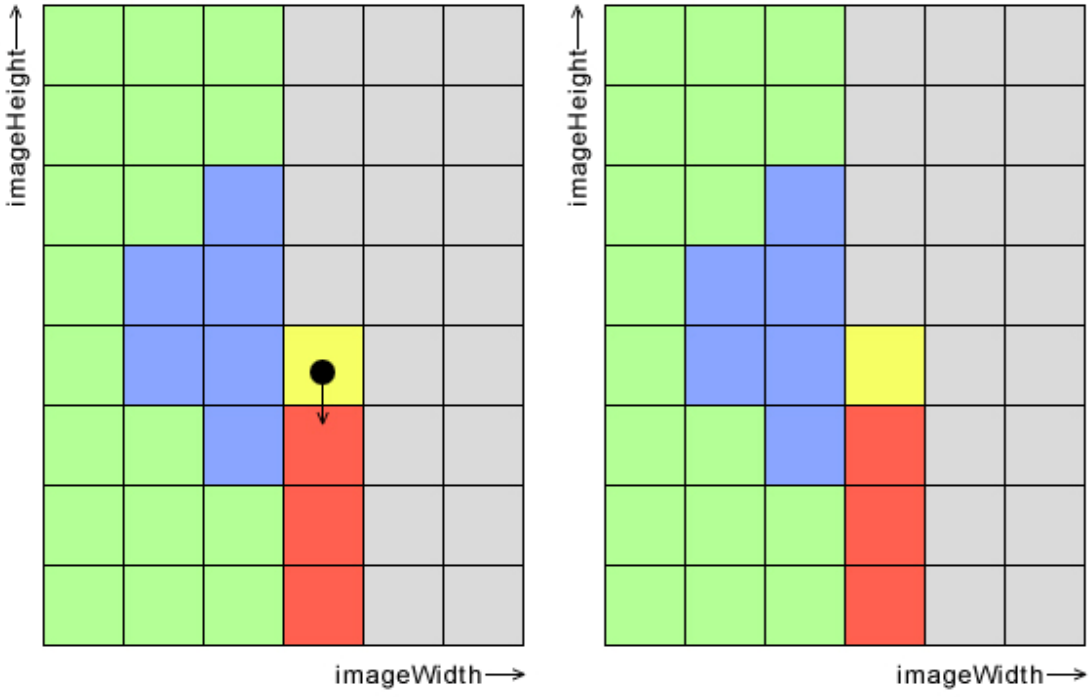
2.5.4. Η κατάτμηση της εικόνας.

Στη μέθοδο κατάτμησης υλοποιούμε τον αλγόριθμο που αναφέραμε πιο πάνω. Σαρώνουμε την εικόνα κατά το ύψος των pixels και τα ελέγχουμε όλα μέχρι το πλάτος της εικόνας. Όπως είπαμε μια περιοχή μπορεί να είναι ένα pixel από μόνο του. Άρα αρχίζουμε θέτοντας για αρχική περιοχή στο συγκεκριμένο πλάτος το pixel που αρχίζει στην θέση ένα του ύψους, δηλαδή hStart=1. Μετά θα δημιουργήσουμε μια καινούρια περιοχή στην κλάση Regions μέσω του αντικειμένου regionsAvg. Σε αυτήν θα εισάγουμε την τιμή κάθε καναλιού χρώματος για την περιοχή που μόλις δημιουργήσαμε. Ακολούθως θέτουμε στον πίνακα regCheck τον αριθμό περιοχής που έχει το pixel μας. Επειδή ακόμα βρισκόμαστε στην αρχή της εικόνας και εξετάζουμε το πρώτο pixel το τέλος της προηγούμενης περιοχής βρίσκετε στο μηδέν που βρίσκεται το περίγραμμα της εικόνας. Έτσι θα πρέπει να σημειώσουμε ότι η εικόνα από μόνη της αποτελεί μια περιοχή όπου μέσα της κρατά άλλες μικρότερες εμφωλευμένες περιοχές.

```
for(int w=1; w<width-1; w++){
        hStart=1;
        regionNo=regionsAvg.createRegion(imageArray[w][1][0]
                                         ,imageArray[w][1][1]
                                         ,imageArray[w][1][2]);
        regCheck[w][1]=regionNo;
        for(int h=1; h<=height-2; h++){
```

Κώδικας 2.5: Δημιουργία αρχικών περιοχών για κάθε πλάτος.

Αφού καθορίσουμε την περιοχή για το πρώτο pixel στο συγκεκριμένο πλάτος συνεχίζουμε κατά ύψος για να εξετάσουμε και τα υπόλοιπα. Χρησιμοποιώντας την μέθοδο isInSegment εξετάζουμε αν το pixel στο οποίο βρισκόμαστε ανήκει στην περιοχή που μέχρι τώρα δημιουργήσαμε. Αν αυτό δεν ανήκει στην περιοχή μας τότε θα κλείσουμε την περιοχή που δημιουργήθηκε και ua εξετάσουμε μια καινούρια. Θα ορίσουμε για αυτή την νέα περιοχή το pixel στο οποίο βρισκόμαστε ως αρχή. Ακόμα, θα δημιουργήσουμε μια καινούρια περιοχή στην κλάση Regions μέσω του αντικειμένου regionsAvg και θα δώσουμε στην νέα περιοχή τον αριθμό της δηλώνοντας την στον πίνακα regCheck. Στην εικόνα 2 στην επόμενη σελίδα, αριστερά με κίτρινο φαίνεται το pixel που ελέγχουμε και με κόκκινο η περιοχή που δημιουργήθηκε μέχρι τώρα. Δεξιά, μετά την αποτυχία της συγχώνευσης, το κίτρινο pixel αποτελεί την αρχή μιας καινούριας περιοχής.



Εικόνα 2.2: Έλεγχος και δημιουργία νέας περιοχής.

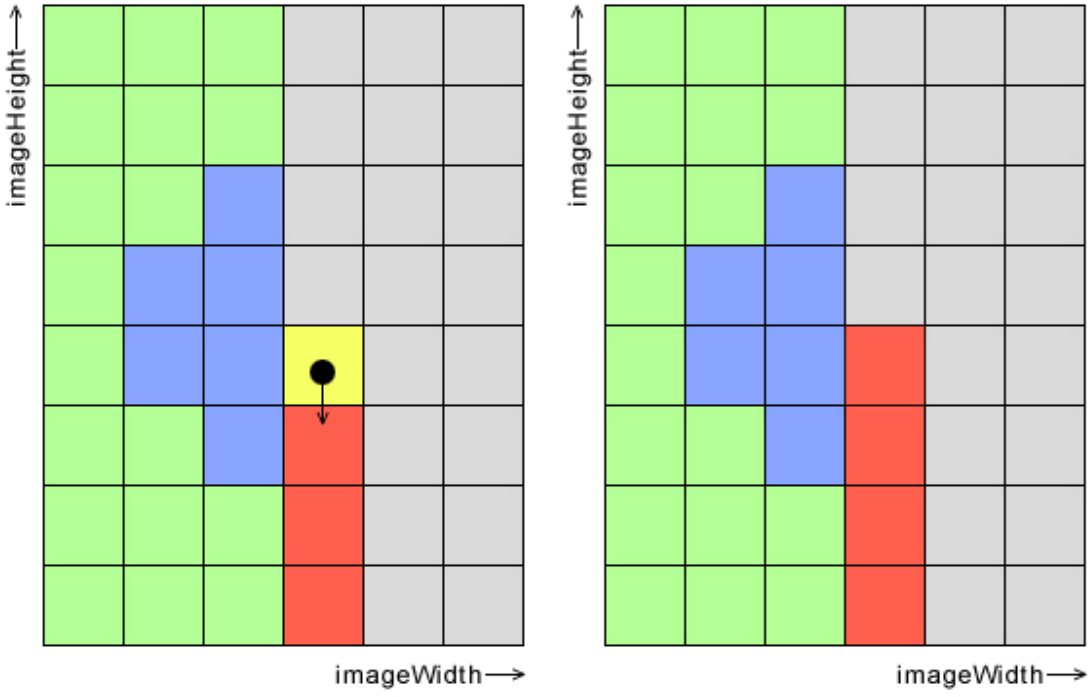
Εδώ θα πρέπει να σημειώσουμε ότι σε περίπτωση που βρισκόμαστε στο τέλος του ύψους της εικόνας απλά θα κλείσουμε την περιοχή που εξετάζουμε τώρα και

ο αλγόριθμος μας θα προχωρήσει στο επόμενο πλάτος. Όλα αυτά φαίνονται πιο κάτω στον κώδικα που ακολουθεί.

```
if(!isInSegment(w,h,regCheck[w][hStart],lastRegionH) || h==height-2){
        hStart =h;
        if(h!=height-2){
          regionNo=regionsAvg.createRegion(imageArray[w][h][0],
                                           imageArray[w][h][1],
                                           imageArray[w][h][2]);
          regCheck[w][h]=regionNo;  }
```

Κώδικας 2.6: Δημιουργία νέας περιοχής.

Όσα αναφέραμε πιο πάνω ελέγχουν την περίπτωση που το pixel το οποίο κοιτάμε δεν ανήκει στην περιοχή που εξετάζουμε. Στην άλλη περίπτωση που μπορεί να προκύψει, το pixel έχει τα ίδια χαρακτηριστικά με την περιοχή και έτσι μπορεί να γίνει συγχώνευση. Στην εικόνα πιο κάτω φαίνεται ο έλεγχος που γίνεται για το τρέχων pixel. Ο έλεγχος είναι θετικός και όπως φαίνεται στην εικόνα δεξιά το pixel συγχωνεύεται με την κόκκινη περιοχή.



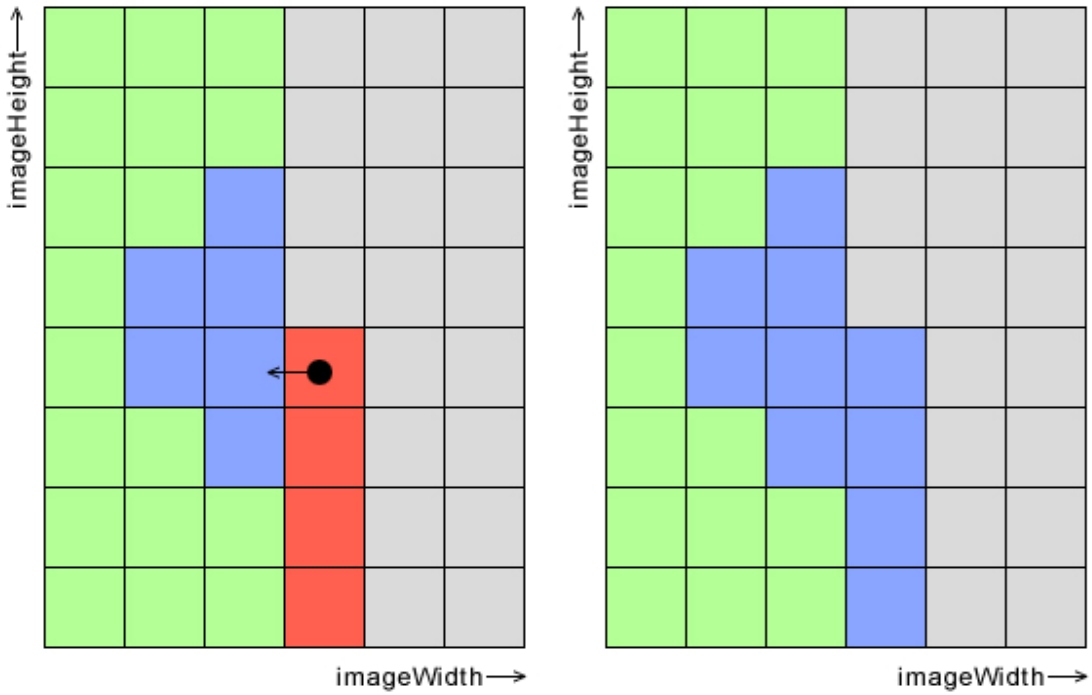
Εικόνα 2.3: Έλεγχος και συγχώνευση pixel με την περιοχή.

Για να γίνει αυτό καλούμε αρχικά την insertPixel της κλάσης Regions, και θέτουμε σε αυτή την τιμή κάθε καναλιού χρώματος του pixel και τον αριθμό της περιοχής μας. Αφού γίνουν αυτά τότε η συγχώνευση τελειώνει δίνοντας τον αριθμό περιοχής στον πίνακα regCheck στην θέση που βρίσκετε το pixel.

```
regionsAvg.insertPixel(imageArray[w][h][0], imageArray[w][h][1],
                       imageArray[w][h][2],regionNo);
regCheck[w][h]=regionNo;
```

Κώδικας 2.7: Συγχώνευση pixel με την τρέχουσα περιοχή.

Αμέσως μετά τον έλεγχο συγχώνευσης με την περιοχή πιο πίσω στο ύψος ένα ακόμα κομμάτι για το οποίο πρέπει να παρέχουμε κάποιο έλεγχο είναι αν το pixel αυτό μπορεί να συγχωνευτεί με την περιοχή που βρίσκετε δίπλα του σε πλάτος. Πιο κάτω, όπως φαίνεται στην εικόνα αριστερά βρισκόμαστε στο pixel με την κουκίδα και ελέγχουμε την συγχώνευση με την περιοχή στην οποία οδηγεί το βέλος.



Εικόνα 2.4: Έλεγχος και συγχώνευση μεταξύ περιοχών.

Αν μπορεί να γίνει συγχώνευση του pixel με την περιοχή τότε έχουμε την μπλε περιοχή όπως φαίνετε στην εικόνα 4 δεξιά. Όταν η συγχώνευση δεν είναι δυνατή τότε η κόκκινη περιοχή θα παραμείνει όπως είναι. Για να πραγματοποιηθούν αυτά πρέπει να βρισκόμαστε σε πλάτος μεγαλύτερο του 1 και ή περιοχή που βρίσκετε δίπλα δεν έχει ήδη τον ίδιο αριθμό με αυτή του pixel μας τότε χρησιμοποιούμε την μέθοδο isInSegment για να καθορίσουμε αν το pixel αυτό είναι συγχωνεύαμε. Αν ο έλεγχος είναι θετικός τότε καλούμε την mergeSegments για να κάνουμε την συγχώνευση.

```
if(w>1 &&
  regCheck[w][h]!=regCheck[w-1][h] &&
  isInSegment(w,h,regCheck[w-1][h])){
            mergeSegments(regCheck[w-1][h],regCheck[w][h]);
            regionNo=regCheck[w-1][h];
}
```

Κώδικας 2.8: Έλεγχος και συγχώνευση μεταξύ περιοχών.

Ο κώδικας των περιπτώσεων που εξετάσαμε πιο πάνω παραπείθετε ολοκληρωμένος πιο κάτω.

```
public void segmentationAvg(){
    int hStart;
   for(int w=1; w<width-1; w++){
     hStart=1;
      regionNo=regionsAvg.createRegion(imageArray[w][1][0],
                                       imageArray[w][1][1],
                                       imageArray[w][1][2]);
      regCheck[w][1]=regionNo;
      for(int h=1; h<=height-2; h++){
        if(!isInSegment(w,h,regCheck[w][hStart]) || h==height-2){
          hStart=h;
          if(h!=height-2){
            regionNo=regionsAvg.createRegion(imageArray[w][h][0],
                                             imageArray[w][h][1],
                                             imageArray[w][h][2]);
```

```
            regCheck[w][h]=regionNo;
        }
    }
    else{
        regionsAvg.insertPixel(imageArray[w][h][0], imageArray[w][h][1],
                               imageArray[w][h][2],regionNo);
        regCheck[w][h]=regionNo;
    }
    if(w>1 &&
            regCheck[w][h]!=regCheck[w-1][h] &&
            isInSegment(w,h,regCheck[w-1][h])){
                mergeSegments(regCheck[w-1][h],regCheck[w][h]);
                regionNo=regCheck[w-1][h];
    } } } }
```

Κώδικας 2.9: Η μέθοδος κατάτμησης segmentationAvg.

2.5.5. Συγχώνευση γειτονικών περιοχών.

Όπως αναφέραμε πιο πάνω στις βασικές αρχές του κώδικα μας, στο βήμα 8, μετά το τέλος της κατάτμησης προσπαθούμε να βρούμε επιπλέον συγχωνεύσεις μεταξύ περιοχών. Η mergeNeighbours είναι μια μέθοδος που κοιτά την κατατετμημένη πια εικόνα και ψάχνει για αυτές τις επιπλέον συγχωνεύσιμες περιοχές που κατά την εκτέλεση του αλγόριθμου μας δεν ανιχνευτήκαν. Η διαφορά της είναι ότι κοιτάει τις γειτονικές περιοχές συνολικά και όχι αν ένα pixel είναι συγχωνεύσιμο με μια περιοχή. Ουσιαστικά αυτό που κάνει είναι να εντοπίζει τα άκρα μιας περιοχής ελέγχοντας αν γειτονικά pixels μέσα στην εικόνα έχουν διαφορετικό αριθμό περιοχής. Αν εντοπιστούν τέτοια pixels τότε γίνετε ένας έλεγχος κατά πόσο οι περιοχές στις οποίες ανήκουν είναι συγχωνεύσιμες και αναλόγως αν ισχύει κάτι τέτοιο τότε θα προχωρήσει στην συγχώνευση. Ακόμα, στις παραμέτρους της παίρνει την elimSize η οποία δηλώνει ένα μέγεθος περιοχής το οποίο οι τελικές περιοχές θα πρέπει να ξεπερνούν. Αν αυτό δεν συμβαίνει τότε οι μικρότερες περιοχές θα συγχωνεύονται με τις γειτονικές τους. Αν η elimSize είναι μηδέν τότε θα συγχωνεύονται μονο οι

περιοχές που είναι συμβατές μεταξύ τους. Ο λόγος που εισάγουμε αυτό τον τρόπο συγχώνευσης είναι για να αποφύγουμε τις περιπτώσεις που έχουμε πολλές και πολύ μικρές περιοχές.

```
private void mergeNeighbours(int elimSize){
    for(int i=1; i<regCheck.length; i++)
        for(int j=1; j<regCheck[0].length; j++){
            if(regCheck[i][j]!=regCheck[i-1][j])
                if(isInSegment(regCheck[i-1][j],regCheck[i][j]) ||
                    regionsAvg.getRegion(regCheck[i][j])[3]<elimSize)
                    mergeSegments(regCheck[i-1][j],regCheck[i][j]);
            if(regCheck[i][j]!=regCheck[i][j-1])
                if(isInSegment(regCheck[i][j-1],regCheck[i][j]) ||
                    regionsAvg.getRegion(regCheck[i][j])[3]<elimSize)
                    mergeSegments(regCheck[i][j-1],regCheck[i][j]);
        }
    }
```

Κώδικας 2.10: Συγχώνευση γειτονικών περιοχών με την segmentationAvg.

2.6. Πειραματικά αποτελέσματα

2.6.1. Σύγκριση διάφορων ρυθμίσεων.

Αν και η ανθρώπινη αντίληψη είναι καθαρά υποκειμενική πιο κάτω παρουσιάζουμε τα αποτελέσματα της προσπάθειας μας αυτής ώστε να δημιουργήσουμε ένα σωστό αλγόριθμο κατάτμησης εικόνας. Στις δοκιμές που κάνουμε πιο κάτω χρησιμοποιούμε αυτόματο υπολογισμό της ανεκτικότητας από το πρόγραμμα μας. Η ανεκτικότητα αυτή υπολογίζετε αν πάρουμε το μισό της τυπικής απόκλισης που παρουσιάζεται στον μέσο όρο των pixels ολόκληρης της εικόνας για κάθε κανάλι χρώματος. Παρουσιάσουμε επίσης μια κατάτμηση που δεν παρέχει ανοχή στις μεγάλες αντιθέσεις και μια ακόμα που κάνει χρήση σταθερής ανεκτικότητας με τιμή 30.

Πρώτα θα παρουσιάσουμε την κατάτμηση μιας εικόνας με τριαντάφυλλα όπως φαίνεται πιο κάτω. Με μια πρώτη μάτια θα λέγαμε ότι καλύτερη κατάτμηση

γίνεται στην ρύθμιση με ανεκτικότητα 30 καθώς η κατάτμηση χωρίς ανεκτικότητα μας δίνει πολλές μικρές περιοχές, και η αυτόματη ρύθμιση δεν καταφέρνει να συγχώνευση το φόντο ολόκληρο σε μία περιοχή. Όμως όπως είπαμε, η ανθρώπινη αντίληψη είναι καθαρά αντικειμενική. Αυτό δηλαδή που σε εμάς φαίνεται σαν ένα ο αλγόριθμος μας το αντιλαμβάνεται διαφορετικά λόγο των διαφορετικών χαρακτηριστικών του.

Εικόνα 2.5: Κατάτμηση εικόνας με τριαντάφυλλα.

Αν δούμε τώρα την επόμενη δοκιμή μας όπου παρουσιάζει την κατάτμηση μίας εικόνας με ένα σκυλί θα φτάσουμε εύκολα στο συμπέρασμα ότι καλύτερα με την αυτόματη ρύθμιση φτάνουμε στο καλύτερο αποτέλεσμα. Όπως φαίνεται στην εικόνα 6 στην επόμενη σελίδα αν παρατηρήσουμε την κατάτμηση με ανεκτικότητα 30 τότε θα έχουμε μια μεγάλη ανεπιθύμητη υπερσυγχώνευση. Έτσι μπορούμε τώρα να διακρίνουμε ότι η ρύθμιση που στην προηγούμενη εικόνα μας έδωσε την καλύτερη κατάτμηση τώρα μας δίνει την χειρότερη. Λόγω της υποκειμενικότητας του ανθρώπινου μυαλού αν ψάχναμε μας μόνοι μας τις

ιδανικές ρυθμίσεις θα φτάναμε σίγουρα σε καλύτερο αποτέλεσμα που θα ερχόταν βέβαια σε αντίθεση με αυτό που προσπαθούμε να κάνουμε. Αυτό που θέλουμε είναι μία κατάτμηση εξ ολοκλήρου από το πρόγραμμα μας που θα μπορεί να φτάσει από μονό του σε αποτελέσματα χωρίς την πλήρη επίβλεψη μας.

Εικόνα 2.6: Κατάτμηση εικόνας με σκυλί.

2.6.2. Σύγκριση κατάτμησης με τον αλγόριθμο Statistical Region Merging.

Ο αλγόριθμος Statistical Region Merging[1] του Richard Nock και Frank Nielsen είναι κατά την γνώμη μας ένας πάρα πολύ καλός αλγόριθμος Region Growing και για αυτό ακριβώς τον λόγο τον επιλέξαμε για να συγκρίνουμε τα αποτελέσματα μας με αυτόν. Επιτυγχάνει πάρα πολύ καλά αποτελέσματα με μεγάλη ταχύτητα εκτέλεσης, και κάνει την επέκταση των περιοχών στην εικόνα χρησιμοποιώντας τεχνικές από την στατιστική για να βρει σε πια περιοχή ανήκει ένα pixel. Παρακάτω παρουσιάζουμε μια σειρά από κατατμήσεις που έχουμε με τον αλγόριθμο μας σε σύγκριση με τον αλγόριθμο Statistical Region Merging. Οι εικόνες που χρησιμοποιήσαμε φαίνονται στην εικόνα 7, στην επόμενη σελίδα.

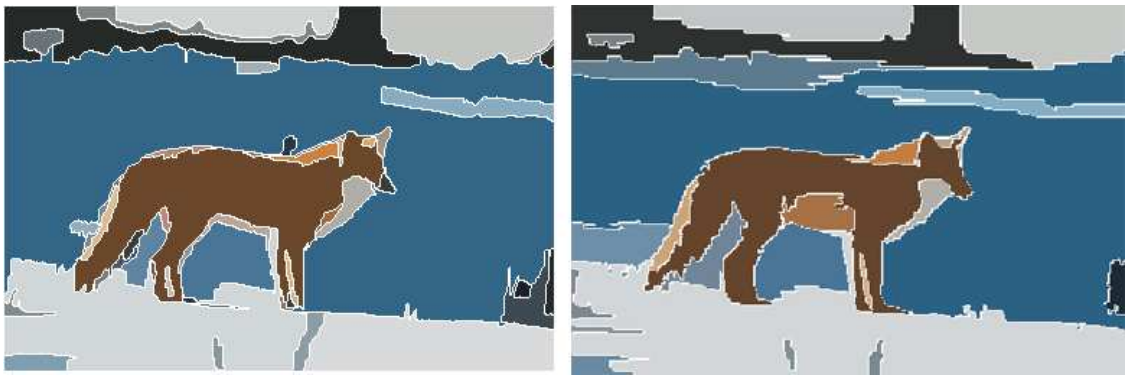
Εικόνα 2.7: Εικόνες για σύγκριση -(α)Τριαντάφυλλο (β)Αλεπού (γ) Λύκος (δ) Πλοίο

Σε όλες τις συγκρίσεις που κάνουμε, οι ρυθμίσεις στον δικό μας αλγόριθμο γίνονται με την αυτόματη επιλογή ανεκτικότητας όπως την παρουσιάσαμε πιο πάνω στην ενότητα 6.1. Στον αλγόριθμο Statistical Region χρησιμοποιούμε τις εξορισμού ρυθμίσεις όπως αυτές παρουσιάζονται στο [1].

Εικόνα 2.8: Σύγκριση (α) (αριστερά:Split n'Merge,δεξιά:Statistical Region Merging)

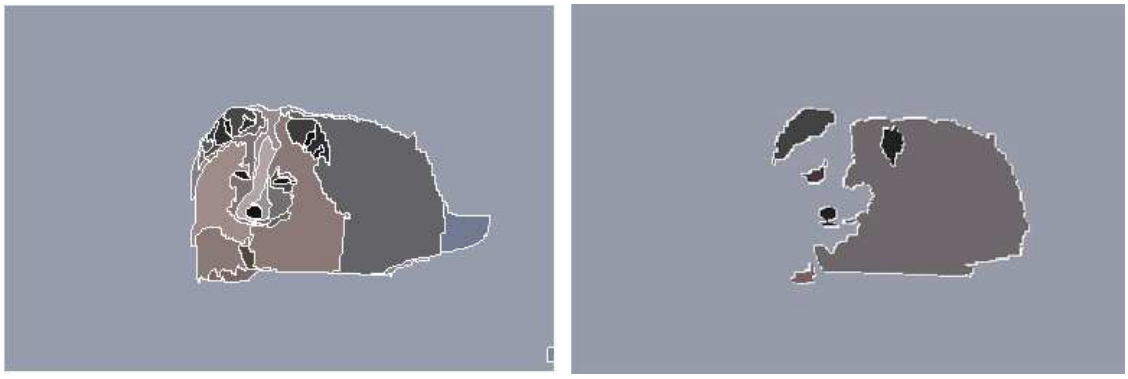
Πιο πάνω φαίνεται η πρώτη μας σύγκριση με την κατάτμηση της εικόνας (α) με το τριαντάφυλλο. Οι συγκρίσεις όπως φαίνονται μοιάζουν σε αρκετά μεγάλο βαθμό.

Μπορούμε να πούμε να πούμε ίσως ότι η κατάτμηση στα δεξιά από τον αλγόριθμο Statistical Region Merging είναι καλύτερη από την κατάτμηση του δικού μας αλγόριθμου. Σε αυτό το συμπέρασμα φτάνουμε αν παρατηρήσουμε ότι ο αλγόριθμος μας προβαίνει σε ένα μικρό βαθμό υπερσυγχώνευσης στα πέταλα του τριαντάφυλλου. Με τον αλγόριθμο Statistical Region Merging ξεχωρίζουμε τα αντικείμενα με μεγαλύτερη ακρίβεια αλλά, και στους δύο αλγόριθμους έχουμε μια αρκετά καλή κατάτμηση. Παρακάτω μπορούμε να δούμε τα αποτελέσματα της δεύτερης μας σύγκρισης στην εικόνα με την αλεπού.



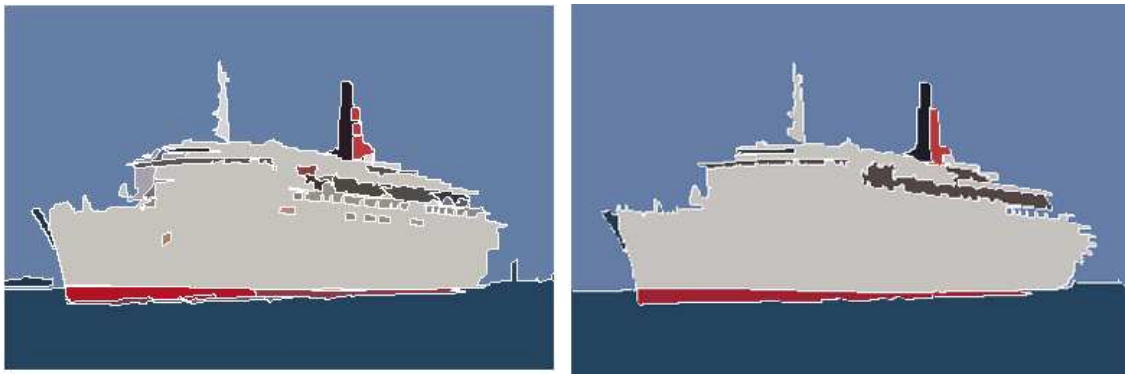
Εικόνα 2.9: Σύγκριση (β)(αριστερά:Split n'Merge, δεξιά:Statistical Region Merging)

Εδώ μπορούμε να πούμε ότι η καλύτερη κατάτμηση είναι αυτή στα αριστερά από τον δικό μας αλγόριθμο. Και πάλι οι δυο αλγόριθμοι παρουσιάζουν αποτελέσματα που είναι παρόμοια. Παρατηρούμε όμως ότι στον αλγόριθμο Statistical Region Merging έχουμε μια ανεπιθύμητη υπερσυγχώνευση στα πίσω πόδια της αλεπούς, καθώς και μία ακόμα περιοχή στην περιοχή της κοιλιάς η ύπαρξη της οποίας όμως εναπόκειται στην ανθρώπινη υποκειμενικότητα.



Εικόνα 2.10:Σύγκριση (γ)(αριστερά:Split n'Merge,δεξιά:Statistical Region Merging)

Η κατάτμηση τώρα της εικόνας 10 της προηγούμενης σελίδας παρουσιάζει την σύγκριση (γ) όπου υπάρχει ένας σχεδόν λευκός λύκος σε ένα λευκό χιονισμένο περιβάλλον. Εδώ φαίνεται ξεκάθαρα ότι η κατάτμηση με την μεγαλύτερη επιτυχία είναι αυτή του αλγόριθμου μας αφού, η κατάτμηση στα δεξιά αποτυγχάνει να καθορίσει ως ξεχωριστό αντικείμενο το κεφάλι του λύκου. Αυτό οφείλετε σε μεγάλο βαθμό στο ότι το χρώμα του κεφαλιού μοιάζει αρκετά με το χρώμα του χιονιού. Αν αλλάξουμε τις ρυθμίσεις στον αλγόριθμο Statistical Region Merging είναι πιθανόν ότι θα πετύχουμε μια καλή κατάτμηση αλλά αυτό που εμείς θέλουμε είναι ένας αλγόριθμος που προσαρμόζεται μόνος του, και προβαίνει σε κατάτμηση χωρίς επίβλεψη. Κάτι που επιτυγχάνουμε στον δικό μας αλγόριθμο κάνοντας χρήση της αυτόματης ρύθμισης της ανεκτικότητας. Πιο κάτω στην εικόνα 11 μπορούμε να δούμε τα αποτελέσματα της τελευταίας μας σύγκρισης για την εικόνα (δ).



Εικόνα 2.11:Σύγκριση (δ)(αριστερά:Split n'Merge,δεξιά:Statistical Region Merging)

Η κατατμήσεις των δύο αλγόριθμων μας δίνουν για ακόμα μια φορά παρόμοια αποτελέσματα με τον αλγόριθμο Statistical Region Merging να υπερισχύει ίσως για λίγο, ξεχωρίζοντας το πλοίο ολόκληρο σε ένα τμήμα, και τον δικό μας να κάνει ακόμα μια μικρή υπερσυγχώνευση στην πλώρη του πλοίου.

Βλέποντας τις τέσσερις συγκρίσεις που παρουσιάσαμε μπορούμε να πούμε ότι ο αλγόριθμος μας επιτυγχάνει πολύ καλά αποτελέσματα. Μπορεί ακόμα χωρίς επίβλεψη, από μόνος του δηλαδή να φτάνει σε αποτελέσματα που οδηγούν σε μια ασφαλή κατάτμηση.

2.7. Συμπεράσματα

Η κατάτμηση είναι πολύ σημαντική στα συστήματα ανάκτησης εικόνας βάσει περιεχομένου γιατί μπορεί να ενισχύσει σε πολύ μεγάλο βαθμό τις πληροφορίες που εξάγονται από μια εικόνα[9][10]. Αντίστοιχα η μορφή και τα χαρακτηριστικά ενός αντικειμένου σε μια εικόνα εξαρτώνται κατά πολύ από τον αλγόριθμο κατάτμησης[11]. Εμείς δημιουργήσαμε ένα αλγόριθμο που ομαδοποιεί τα αντικείμενα ανάλογα με τα κοινά χαρακτηρίστηκα που έχουν τα pixels τους σε κάθε κανάλι χρώματος. Επίσης αφού δημιουργηθεί μια πρώτη ιδέα των αντικειμένων επανεξετάζουμε για πιθανές συγχωνεύσεις μεταξύ τους. Όλα αυτά μπορούν να υλοποιηθούν με ένα πολύ απλό κριτήριο συγχώνευσης που το μόνο σφάλμα που μπορεί μερικές φορές να μας παρουσιαστεί είναι αυτό της υπερσυγχώνευσης. Στις περισσότερες περιπτώσεις όμως ο αλγόριθμος μας, μας παρέχει με μεγάλη ταχύτητα, αξιόπιστα αποτελέσματα που ικανοποιούν τους στόχους μας στην κατάτμηση εικόνας.

Κεφάλαιο 3

Αυτοοργανούμενοι Χάρτες

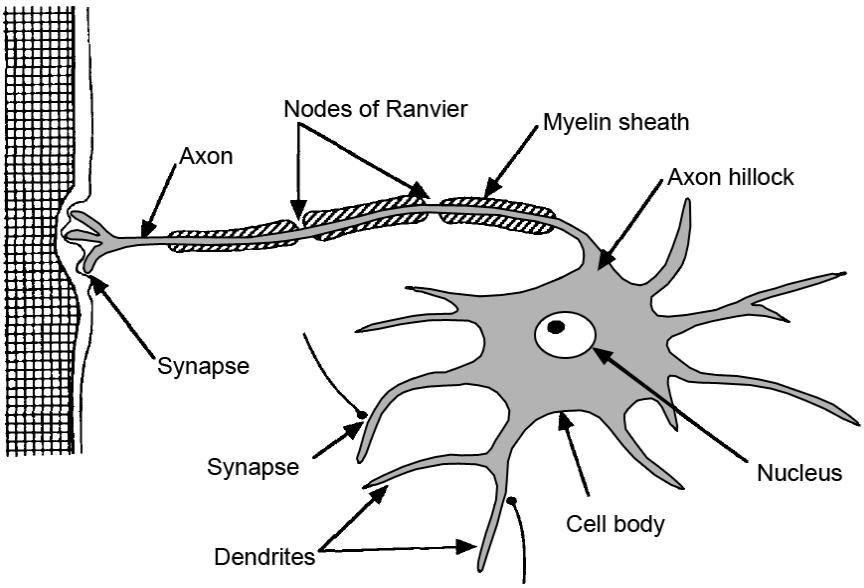
3.1. Εισαγωγή

Μετά την ανάλυση της εικόνας μέσω της κατάτμησης θα πρέπει να εισάγουμε τα δεδομένα μας μέσα σε ένα αυτοοργανούμενο χάρτη (Self Organizing Map). Οι αυτοοργανούμενοι χάρτες αναπτύχθηκαν από τον Teuvo Kohonen το 1982[13], γενικά μπορούμε να πούμε ότι πρόκειται για ένα σχετικά απλό εργαλείο που όμως μπορεί να ταξινομεί πολύπλοκα δεδομένα ανάλογα με τις συσχετίσεις που παρουσιάζουν μεταξύ τους. Έχουμε επιλέξει αυτό τον τύπο νευρωνικού δικτύου λόγω της ικανότητας του να προσαρμόζεται στα δεδομένα που λαμβάνει χωρίς να έχει κάποια επίβλεψη. Αυτό που θα θέλαμε είναι να κατηγοριοποιήσουμε τις περιοχές μας και να προσδιορίσουμε ένα αρμόδιο νευρώνα για κάθε περιοχή που μπορεί να προκύψει. Πριν πούμε περισσότερα όμως, πάμε να δούμε τις βασικές αρχές τω νευρωνικών δικτύων και να εξετάσουμε την λειτουργία του αυτοοργανούμενου χάρτη

3.1.1. Αναδρομή στα Νευρωνικά Δίκτυα

Τα νευρωνικά δίκτυα είναι γνωστά και ως συνδετικά μοντέλα που προσπαθούν να κάνουν χρήση μερικών γνωστών ή αναμενόμενων τεχνικών οργάνωσης του ανθρώπινου εγκεφάλου. Ένα νευρωνικό δίκτυο αποτελείτε από ένα αριθμό απλών, ανεξάρτητων επεξεργαστών, τους νευρώνες[15].

Στην αρχή η έρευνα σε αυτόν τον τομέα έγινε για νευροβιολογικά ενδιαφέροντα. Τα πρώτα πεδία έρευνας κινούνταν γύρω από ένα νευρώνα και τη μοντελοποίηση του, ή γύρω από τους αποκαλούμενους κανόνες μάθησης τροποποιώντας τα συνοπτικά βάρη. Το perceptron του Frank Rosenblatt ήταν το πρώτο που θεώρησε τις προοπτικές της επεξεργασίας και της αποθήκευσης πληροφοριών. Ως μια μηχανή που μαθαίνει δημιούργησε τεράστια αίσθηση στο χώρο ξεκινώντας έτσι μαζί του την “χρυσή εποχή” των νευρωνικών δικτύων την δεκαετία του 1960. Σύντομα όμως έγινε γνωστό ότι δεν μπορούσε να λύσει απλά προβλήματα, όπως για παράδειγμα να ξεχωρίσει τους άρτιους από τους περιττούς αριθμούς[15].



Εικόνα 3.1: Ένας απλοποιημένος νευρώνας του εγκεφάλου[17][18]**.**

Έτσι η έρευνα στο πεδίο υπέφερε από μια τεράστια οπισθοδρόμηση. αυτό ξεκίνησε το 1969 την “σκοτεινή εποχή” των νευρωνικών δικτύων. Το ενδιαφέρον και η εφορία επανήλθαν το 1985 όταν ανακαλύφθηκαν νέοι αλγόριθμοι μάθησης. Σήμερα μερικά νευρωνικά δίκτυα ονομάζονται παγκόσμιοι προσεγγιστές (universal aproximators) πού μπορούν να προσαρμοστούν σε οποιαδήποτε δεδομένα εισόδου, εξόδου και να προσεγγίσουν κάθε δεδομένο πρόβλημα. Διάφοροι τύποι νευρωνικών δικτύων μπορούν να λύσουν διάφορα προβλήματα όπως η αναγνώριση προτύπων, συμπλήρωση προτύπων, καθορισμός ομοιοτήτων μεταξύ προτύπων ή δεδομένων, και αυτόματη κατάταξη σε κατηγορίες (classification)[15].Αν η διαδικασία μάθησης καταφέρει να βρει ένα κατάλληλο συνδυασμό βαρών έτσι ώστε να λύνετε το δεδομένο πρόβλημα, τότε η λύση του περικλείεται στις συνδέσεις του δικτύου. Τα νευρωνικά δίκτυα για τους χρήστες τους λειτουργούν σαν “μαύρα κουτιά” και συνήθως δεν είναι δυνατόν να εξάγουμε από αυτά καθαρή γνώση. Τα νευρωνικά δίκτυα δηλαδή είναι ικανά να λύσουν δύσκολα προβλήματα αλλά δεν μας λένε πως το κάνουν. Παρόμοια προβλήματα προκύπτουν αν θέλουμε να χρησιμοποιήσουμε ήδη υπάρχουσα γνώση για τις συνδέσεις μεταξύ προτύπων εισόδου και εξόδου. Είναι αδύνατο να προσαρμόσουμε μια τέτοια γνώση αν θέλουμε να επιταχύνουμε τη

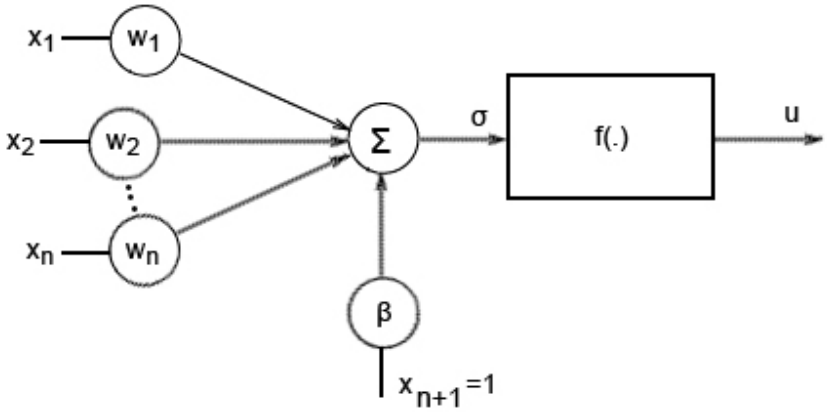
διαδικασία εκπαίδευσης. Ένα νευρωνικό δίκτυο πάντα πρέπει να μάθει από το μηδέν.

3.1.2. Ο Τεχνητός Νευρώνας

Ένας στατικός νευρώνας αποτελείτε από ένα αθροιστή, η έξοδος του οποίου είναι το σταθμισμένο άθροισμα των εισόδων του[14], δηλαδή:

$$\sigma = \sum_{i=1}^{n} w_i x_i + \beta = \sum_{i=1}^{n+1} w_i x_i \quad \text{και} \quad x_{n+1}=1,\ \beta = w_{n+1}$$

Όπου **w** είναι τα βάρη, **x** οι είσοδοι του νευρώνα και **β** η σταθερά πόλωσης. Θετική τιμή ενός βάρους αναπαριστά την διέγερση και η αρνητική την αποδιέγερση. Συνεπώς η απόλυτη τιμή του βάρους καθορίζει την ισχύ της σύνδεσης[14][17].



Εικόνα 3.2: Τεχνητός νευρώνας

Το σταθμισμένο άθροισμα ενεργοποιεί την συνάρτηση **f(.)** που έχει χαρακτήρα λογικής μονάδας κατωφλίου (threshold). Η έξοδος του νευρώνα σκανδαλίζεται όταν το σήμα **σ** περάσει το κατώφλι που ορίζεται από την πόλωση **β** [2]. Για την συνάρτηση εξόδου του νευρώνα υπάρχουν διάφορες παραλλαγές[14][17]:

1. Γραμμική σχέση(γραμμικός νευρώνας):

$$f(\sigma)=\sigma$$

2. Δυαδική(δίτιμη) σχέση κατωφλίου:

$$f(\sigma)=1 \text{ εάν } \sigma>0$$

$$f(\sigma)=0 \text{ εάν } \sigma\leq 0$$

3. Σιγμοειδής(Sigmoid):

$$f(\sigma)=1/1+e^{-\sigma} \in [0,1]$$

4. Σχέση με υπερβολική εφαπτομένη(hyperbolic tangent):

$$f(\sigma)= 1-e^{-\sigma} /1+e^{-\sigma}= ½ \in [0,1]$$

5. Σχέση τύπου Perceptron:

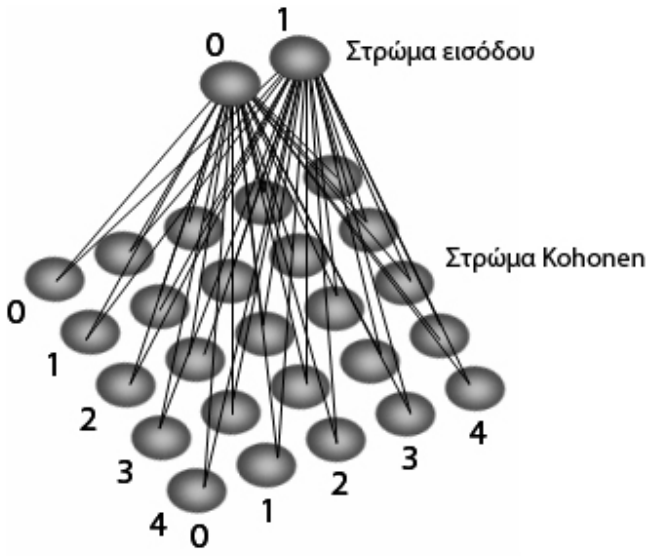
$$f(\sigma)=\sigma \text{ εάν } \sigma>0$$

$$f(\sigma)=0 \text{ εάν } \sigma\leq 0$$

3.1.3. Ανάλυση του Αυτοοργανούμενου χάρτη (Self Organizing Map)

Αυτός ο τύπος δικτύου παρέχει μια μέθοδο εκπαίδευσης χωρίς επίβλεψη που είναι γνωστή ως ανταγωνιστική μάθηση, και είναι ιδιαίτερα χρήσιμη όταν γίνεται ανάλυση δεδομένων ανάμεσα στα οποία δεν είναι γνωστές οι συσχετίσεις[16]. Σε αυτό το δίκτυο δεν υπάρχουν στόχοι για να επιτευχθούν, όπως επίσης δεν υπάρχει σφάλμα για να ελαχιστοποιηθεί. Αυτά είναι και τα χαρακτηριστικά που το κάνουν να ξεχωρίζει από τις μεθόδους που λειτουργούν με επιβλεπόμενη μάθηση[16].

Ένα νευρωνικό δίκτυο SOM(Self Organizing Map) λειτουργεί με δύο στρώματα, το στρώμα εισόδου και το στρώμα Kohonen[15][16][19]. Το στρώμα εισόδου αποτελείτε από κόμβους που διανέμουν τις τιμές των μεταβλητών των πρότυπων εισόδου σε κάθε κόμβο του στρώματος Kohonen όπως φαίνεται και στο σχήμα 3 πιο κάτω[16]. Το στρώμα Kohonen είναι μια συλλογή από κόμβους τοποθετημένους σε τετραγωνική ή εξαγωνική μορφή. Οι κόμβοι του στρώματος εισόδου είναι συνδεδεμένοι με κάθε κόμβο του στρώματος Kohonen μέσω βαρών που τροποποιούνται κατά την εκπαίδευση[16].

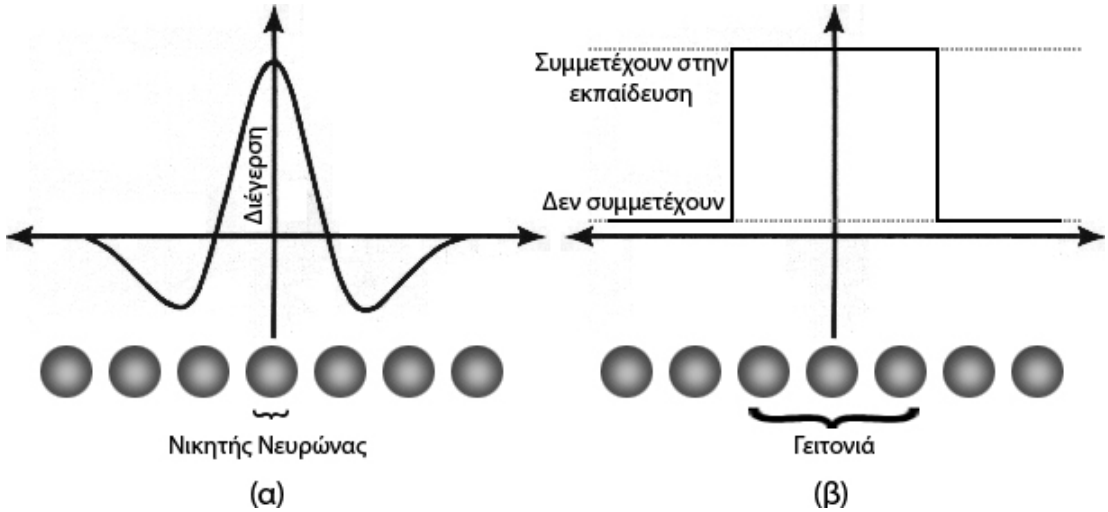


Εικόνα 3.3: Τα στρώματα εισόδου και Kohonen στο δίκτυο SOM

Σε σύγκριση με ένα βιολογικό νευρωνικό δίκτυο, το στρώμα Kohonen είναι ανάλογο με τον φλοιό του ανθρώπινου εγκέφαλου. Ο φλοιός του εγκεφάλου είναι σαν ένα ρούχο με μέγεθος περίπου ένα τετραγωνικό μέτρο και νευρώνες έξι στρωμάτων διπλωμένους έτσι ώστε να μπορούν να χωρέσουν στο κρανίο[16]. Παρόλο που η μηχανική και οι ενέργειες του φλοιού δεν είναι εντελώς κατανοητές υπάρχουν στοιχεία που μας λένε ότι υπάρχει μια πλευρική επέκταση στην αλληλεπίδραση ανάμεσα στους νευρώνες του. Με άλλα λόγια αν ένας νευρώνας στον εγκεφαλικό φλοιό διεγερθεί, τότε σε κάποιο βαθμό θα διεγερθούν και οι νευρώνες που τον περιβάλλουν. Μπορούμε να πούμε ότι όσο αυξάνεται η απόσταση μεταξύ των νευρώνων που γειτονεύουν τόσο μειώνεται η διέγερση, όπως φαίνεται και από την συνάρτηση του Mexican hat στο σχήμα 4^{α}[16].

Το στρώμα Kohonen είναι μια απλοποίηση του εγκεφαλικού φλοιού. Πριν ξεκινήσει η εκπαίδευση οι μεταβλητές των βαρών αρχικοποιούνται σε τυχαίες τιμές. Η εκπαίδευση γίνεται όταν παρουσιαστούν τα πρότυπα στο δίκτυο. Οι κόμβοι στο στρώμα Kohonen ανταγωνίζονται μεταξύ τους για να βρουν ποιος νευρώνας θα διεγερθεί. Αυτό που καθορίζει το νικητή κόμβο είναι η ευκλείδεια απόσταση ανάμεσα στα πρότυπα εισόδου και τα βάρη[16][19]. Νικητής κόμβος είναι αυτός που έχει την μικρότερη απόσταση. Η διαδικασία εκπαίδευσης

τροποποιεί τα βάρη του νικητή κόμβου και της γειτονίας του. Για να επιτευχτεί κάποια υπολογιστική απλοποίηση στη συμμετοχή που θα έχει ο νικητής νευρώνας και η γειτονιά του στην τροποποίηση των βαρών, η συνάρτηση του Mexican hat που παρατηρείται στους βιολογικούς νευρώνες απλοποιείτε σε μια βηματική συνάρτηση όπως φαίνεται στο σχήμα πιο κάτω. Εδώ όλοι οι κόμβοι στη γειτονιά του νικητή συμμετέχουν πλήρως στην διαδικασία μάθησης. Καθώς η διαδικασία εκπαίδευσης προχωρά το μέγεθος της γειτονιάς μειώνετε μέχρι να γίνει μηδέν, και έτσι με κάθε πρότυπο υπάρχει ένας μόνο νικητής. Το βήμα εκπαίδευσης επίσης μειώνεται κατά την διάρκεια της εκπαίδευσης[16].



Εικόνα 3.4: Συνάρτηση μεξικάνικου καπέλου και βηματική συνάρτηση

Το αποτέλεσμα της εκπαίδευσης είναι ένας χάρτης χαρακτηριστικών που αναγνωρίζει τις σχέσεις ανάμεσα στα πρότυπα εισόδου. Με άλλα λόγια κλάσεις με παρόμοια χαρακτηριστικά είναι τοποθετημένες η μία δίπλα στην άλλη. Ο χάρτης χαρακτηριστικών μπορεί να χρησιμοποιηθεί για την κατηγοριοποίηση καινούριων προτύπων εισόδου αναγνωρίζοντας τον νικητή νευρώνα απλά με την παρουσίαση του προτύπου στο δίκτυο.

3.1.4. Προσαρμογή του αυτοοργανούμενου χάρτη στο μοντέλο μας

Το μοντέλο μας, στο επίπεδο εισόδου του αυτοοργανούμενου χάρτη θα παίρνει ως παραμέτρους το χρώμα και το σχήμα μιας περιοχής όπως αυτή έχει καθοριστεί από την κατάτμηση. Στην έξοδο μας θα βρίσκετε η θέση του νικητή νευρώνα η οποία στην συνέχεια θα χρησιμοποιείτε από το πρόγαμα μας για να

φτιάξουμε μια σειρά αντικειμένων ή αναμνήσεων όπως τις αποκαλούμε, που θα αποτελούνται από αυτούς τους νικητές νευρώνες. Πιο κάτω φαίνετε ο τύπος της Ευκλείδειας απόστασης που θα χρησιμοποιηθεί για να βρεθεί ο νευρώνας που έχει την μικρότερη απόσταση από τα δεδομένα της εισόδου και είναι ο νικητής.

$$\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Για καλύτερη απόδοση τον αυτοοργανούμενο χάρτη τον έχουμε κάνει σφαιρικό. Όλες οι λειτουργείς του παραμένουν οι ίδιες όπως τις περιγράψαμε πιο πάνω με την διαφορά όμως ότι όλες οι άκρες του χάρτη συνδέονται μεταξύ τους. Σαν να βλέπουμε δηλαδή ένα χάρτη της γης με την Αμερική και την Ασία στις άκρες που όμως συνδέονται και είναι δυο γειτονικές περιοχές.

3.2. Υλοποίηση Αυτοοργανούμενου χάρτη σε Java

Παρακάτω παρουσιάζεται η υλοποίηση του αυτοοργανούμενου χάρτη όπως τον ορίσαμε στο ιδανικό μοντέλο. Θα δημιουργήσουμε ένα δίκτυο Kohonen SOM το οποίο στο στρώμα εισόδου θα παίρνει το σχήμα που είναι ένας πίνακας με 1600 pixels, και ένα χρώμα με δηλαδή τρεις τιμές για κόκκινο, πράσινο και μπλε. Το μέγεθος του στρώματος Kohonen, το αρχικό και τελικό βήμα εκπαίδευσης, το μέγεθος της γειτονιάς, καθώς και ο μέγιστος αριθμός εποχών θα καθορίζονται από τον χρήστη.

Ο κώδικας όπως σε όλη την εργασία είναι γραμμένος σε Java και θα επεκταθούμε σε αυτόν όπου είναι απαραίτητο. Στις παρακάτω παραγράφους θα επικεντρωθούμε στον κώδικα που υλοποιεί τις λειτουργίες του δικτύου και όπου αλλού κρίνουμε απαραίτητο ότι πρέπει να επεκταθούμε αφήνοντας μερικές φορές έξω μερικά σημεία τα οποία δεν θεωρούμε σημαντικά.

3.2.1. Δομή των δεδομένων μας στον αυτοοργανούμενο χάρτη

Τα δεδομένα που εξάγονται από τον κώδικα της κατάτμησης εισάγονται από τον αυτοοργανούμενο χάρτη είτε άμεσα είτε από την αποθηκευμένη τους μορφή σε XML από την κλάση *XMLReaderWriter* στο πακέτο *filesIO*. Κάθε κατατετμημένη εικόνα θα δημιουργήσει ένα αντικείμενο της κλάσης SomImage το

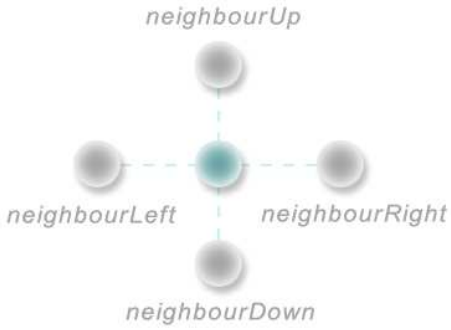
οποίο κρατάει σε μεταβλητές το την περιγραφή της εικόνας αν αυτή είναι διαθέσιμη, την ανεκτικότητα με την οποία έγινε η κατάτμηση, το μέγεθος που καταλαμβάνουν σε pixels συνολικά όλες οι περιοχές, καθώς και ένα *Vector<RegionDataSet>* με το όνομα *image* το οποίο περιέχει όλες τις περιοχές που προέκυψαν μετά την κατάτμηση.

Η *RegionDataSet* είναι η κλάση όπου κρατά τα δεδομένα για μια περιοχή. Κάθε περιοχή χαρακτηρίζεται από το χρώμα της σε όλα τα κανάλια χρώματος, την τοποθεσία της περιοχής μέσα στην εικόνα σε ύψος και σε πλάτος, το μέγεθος της σε pixels, και την μορφή της περιοχής σε ένα πίνακα 1600 θέσεων. Για τα παραπάνω με την σειρά που αναφέρθηκαν χρησιμοποιούμε τις μεταβλητές *red*, *green*, *blue*, *wOffset*, *hOffset*, *elements* και *shape[ ]*. Κατά την εκπαίδευση και ανάκληση του δικτύου δεν χρησιμοποιούνται όλα τα δεδομένα που παρουσιάσαμε πιο πάνω αλλά εμείς τα παρουσιάζουμε εδώ καθώς πιο θα τα χρειαστούμε σε επόμενα βήματα του μοντέλου μας.

Την κλάση *RegionDataSet* ουσιαστικά μπορούμε να την θεωρήσουμε ως τον νευρώνα στο στρώμα εισόδου καθώς είναι αυτή που συνδέεται με τους νευρώνες στο στρώμα Kohonen[19].

3.2.2. Οι νευρώνες στο στρώμα Kohonen

Το στρώμα Kohonen αποτελείτε από νευρώνες που παίρνουν τα δεδομένα από τις κλάσεις *RegionDataSet* και χρησιμοποιώντας τον τύπο της ευκλείδειας απόστασης υπολογίζουν την απόσταση τους από αυτά. Ο κάθε νευρώνας σε αυτό το επίπεδο έχει τέσσερις γείτονες, έναν στα δεξιά, έναν στα αριστερά, έναν πάνω και έναν κάτω.



Εικόνα 3.5: Ο νευρώνας και οι γείτονες του

Στην εφαρμογή μας οι νευρώνες είναι αντικείμενα της κλάσης *Neuron* και κάθε ένας από αυτούς έχει ως γείτονες αντικείμενα της ίδιας κλάσης τους νευρώνες *neighbourLeft*, *neighbourRight*, *neighbourUp*, *neighbourDown*. Επειδή οι νευρώνες αυτοί έχουν ιδιωτική πρόσβαση στην κλάση μας όταν θέλουμε να τους ανακτήσουμε θα χρησιμοποιούμε τις πιο κάτω μεθόδους:

```
public Neuron getNeighbourLeft(){return neighbourLeft;}
public Neuron getNeighbourRight(){return neighbourRight;}
public Neuron getNeighbourUp(){return neighbourUp;}
public Neuron getNeighbourDown(){return neighbourDown;}
```

Οι νευρώνες μας θα πρέπει να εκπαιδεύονται έτσι ώστε να μοιάζουν ο κάθε ένας σε διαφορετικό είδος περιοχής. Για αυτό τον λόγο κάθε νευρώνας θα αντιπροσωπεύεται από ένα σχήμα και ένα χρώμα όπου κατά την εκπαίδευση θα προσαρμόζετε στα χαρακτηριστικά μιας περιοχής. Το σχήμα και το χρώμα σε έναν νευρώνα αντιπροσωπεύονται από τους πίνακες *shape* και *color*. Μόλις ένας νευρώνας δημιουργηθεί οι δύο αυτοί πίνακες αρχικοποιούνται σε τυχαίες τιμές καλώντας την μέθοδο *initNeuron* που φαίνετε πιο κάτω.

```
private void initNeuron() {
        for(int i=0; i<color.length; i++)
                color[i]=Math.random()*255;

        for(int i=0; i<shape.length; i++)
        shape[i]=Math.random()*100;
}
```

Κώδικας 3.1: Μέθοδος αρχικοποίησης των χαρακτηριστικών του νευρώνα

Το χρώμα στον πίνακα *color* σε κάθε κανάλι χρώματος αρχικοποιείτε παίρνοντας τιμές από 0 ως 255 ενώ ο πίνακας *shape* των 1600 θέσεων για το σχήμα παίρνει τιμές από 0 ως 100. Μετά την αρχικοποίηση και κατά την εκπαίδευση αυτές οι τιμές αλλάζουν ανάλογα με τις περιοχές τις οποίες τους παρουσιάζουμε.

3.2.3. Υπολογισμός απόστασης της περιοχής από τον νευρώνα

Αυτό που κάνει την λειτουργία του αυτοοργανούμενου χάρτη να γίνεται χωρίς στόχους είναι η ανταγωνιστική μάθηση. Όταν παρουσιάσουμε μια περιοχή στο στρώμα Kohonen όλοι οι νευρώνες ανταγωνίζονται για το ποιος είναι καταλληλότερος να κερδίσει. Αυτό που καθορίζει τον νικητή σε αυτόν τον ανταγωνισμό είναι η απόσταση. Η απόσταση αποτελεί ένα κριτήριο ομοιότητας που εφαρμόζει τον τύπο της ευκλείδειας απόστασης στα κανάλια χρώματος και το σχήμα της περιοχής, δηλαδή η απόσταση βρίσκεται ως εξης:

$$\sqrt{\sum_{i=1}^{3}(color[i]-colorDataSet[i])^2} + \sqrt{\sum_{i=1}^{1600}(shape[i]-shapeDataSet[i])^2}$$

Η συνολική απόσταση είναι δηλαδή η απόσταση που έχει το χρώμα, συν την απόσταση που έχει το σχήμα της περιοχής. Στην Java την απόσταση την υπολογίζουμε με τις τρείς μεθόδους πιο κάτω.

```
public double distance(double shapeDataSet[],double[] colorDataSet){
   return shapeDistance(shapeDataSet) + colorDistance(colorDataSet);
}
```

Κώδικας 3.2: Μέθοδος υπολογισμού συνολικής απόστασης

Καλώντας αρχίκα την μέθοδο *distance* πού με την σειρά της καλεί την *shapeDistance* και *colorDistance* υπολογίζουμε την απόσταση. Αυτό που κάνει είναι να προσθέτει τις αποστάσεις που επιστρέφουν οι δύο μέθοδοι πιο κάτω.

```
public double shapeDistance(double shapeDataSet[]){
   double total=0;
   double dist;
   for(int i=0; i<shape.length; i++){
      total+=Math.pow(shape[i]-shapeDataSet[i],2);
   }
   dist=Math.sqrt(total);
   return dist;
}
```

Κώδικας 3.3: Μέθοδος υπολογισμού απόστασης σχήματος

```
public double colorDistance(double colorDataSet[]){
    double total=0;
    double dist;
    for(int i=0; i<color.length; i++){
        total+=Math.pow(color[i]-colorDataSet[i],2);
    }
    dist=Math.sqrt(total);

    return dist;
}
```

Κώδικας 3.4: Μέθοδος υπολογισμού απόστασης χρώματος

Η *shapeDistance* και *colorDistance* παίρνουν ως όρισμα ένα πίνακα, για το σχήμα τον *shapeDataSet* και για το χρώμα τον *colorDataset* για να υπολογίσει την απόσταση από τους αντίστοιχους πίνακες του νευρώνα. Αρχικά υπολογίζουν το άθροισμα της διαφοράς κάθε δείγματος του πίνακα δοσμένο στο τετράγωνο χρησιμοποιώντας την *Math.pow* της Java και ακολούθως βρίσκουμε την ρίζα αυτού του αθροίσματος χρησιμοποιώντας την *Math.sqrt*.

3.2.4. Εκπαίδευση νευρώνα

Όπως θα δούμε και πιο κάτω, κατά την διαδικασία εκπαίδευσης του αυτοοργανούμενου χάρτη παρουσιάζουμε τις διάφορες περιοχές στους νευρώνες του δικτύου και ψάχνουμε τον νικητή. Όταν ο νικητής βρεθεί τότε θα πρέπει να γίνει κάποια εκπαίδευση σε αυτόν και την γειτονιά του έτσι ώστε να μοιάζει περισσότερο στην περιοχή που έχει κερδίσει. Η εκπαιδευση ενός νευρώνα γίνεται με τις τρεις μεθόδους πιο κάτω.

```
public void train(double shapeDataSet[],double colorDataSet[],double beta){
    trainShape(shapeDataSet, beta);
    trainColor(colorDataSet,beta);
}
```

Κώδικας 3.5: Μέθοδος εκπαίδευσης νευρώνα

Η μέθοδος *train* δέχεται τα χαρακτηριστηκά της περιοχής που μόλις κέρδισε ο νευρώνας και με την σειρά της τα δίνει στις μεθόδους *trainShape* και *trainColor* όπου εκπαιδεύουν το σχήμα και το χρώμα του νευρώνα[16].

```
public void trainShape(double shapeDataSet[],double beta){
    double delta=0;
    for(int i=0; i<shape.length; i++){
        delta = beta * (shapeDataSet[i] - shape[i]);
        shape[i]+=delta;
    }
}
```

Κώδικας 3.6: Μέθοδος εκπαίδευσης σχήματος νευρώνα

```
public void trainColor(double colorDataSet[],double beta){
    double delta=0;
    for(int i=0; i<color.length; i++){
        delta = beta * (colorDataSet[i] - color[i]);
        color[i]+=delta;
    }
}
```

Κώδικας 3.7: Μέθοδος εκπαίδευσης χρώματος νευρώνα

Οι δύο μέθοδοι εκπαίδευσης παίρνουν εκτός από το χρώμα και το σχήμα της περιοχής ένα ακόμα όρισμα, το βήμα εκπαίδευσης *beta*. Το βήμα εκπαίδευσης καθορίζει το βαθμό στον οποίο θα μοιάζει ο νευρώνας στην περιοχή που του παρουσιάζεται. Κατά την διάρκεια της εκπαίδευσης του δικτύου το *beta* θα μειώνεται μέχρι να φτάσει σε μία πολύ μικρή τιμή που ορίζουμε εμείς. Αυτό που κάνουμε στην εκπαίδευση είναι να προσθέτουμε σε κάθε στοιχείο που έχει το σχήμα και το κάθε κανάλι χρώματος του νευρώνα την διαφορά του από την τιμή που έχει το αντίστοιχο στοιχείο της περιοχής πολλαπλασιάζοντας το με το βήμα εκπαίδευσης. Η τιμή που προστίθεται ονομάζεται δέλτα και βρίσκεται όπως πιο κάτω:

```
delta = beta * (colorDataSet[i] - color[i]);
```

3.2.5. Η γειτονιά στον αυτοοργανούμενο χάρτη

Οι αυτοοργανούμενοι χάρτες έχουν βασισμένες τις θεμελιώδες αρχές της λειτουργίας του φλοιού του εγκεφάλου. Στον φλοιό υπάρχει έντονη υ ύπαρξη της γειτνίασης ανάμεσα στους νευρώνες, δηλαδή νευρώνες που βρίσκονται κοντά ο ένας στον άλλο έχουν παρόμοια χαρακτηριστικά. Όσο η απόσταση ανάμεσα σε δύο νευρώνες αυξάνεται η ομοιότητα αυτή θα μειώνεται. Για να πετύχουμε αυτή την λειτουργία στον αλγόριθμο μας θα πρέπει κατά την εκπαίδευση να μην εκπαιδεύεται μόνο ο νικητής νευρώνας αλλά και η γειτονία του. Το μέγεθος της γειτονιάς μπορεί να είναι σχετικά μεγάλο αρχικά για να σχηματιστεί η γενική δομή του στρώματος Kohonen και σταδιακά να μειώνεται έτσι ώστε να μπορούν οι νευρώνες να εξειδικευτούν σε μια περιοχή. Παρακάτω βλέπουμε τις δύο μεθόδους που επιτρέπουν στους νευρώνες να ανακτούν ένα αριθμό γειτόνων προς τα αριστερά και προς τα πάνω όσο είναι το μέγεθος της γειτονιάς.

```
public Neuron goLeft(int i,int neigh,Neuron current){
   if(i==neigh) return current;
   else return goLeft(i+1,neigh,current.getNeighbourLeft());
}
```

Κώδικας 3.8: Μέθοδος ανάκτησης γειτονιάς νευρώνα προς τα αριστερά

```
public Neuron goUp(int i,int neigh,Neuron current){
   if(i==neigh) return current;
   else return goUp(i+1,neigh,current.getNeighbourUp());
}
```

Κώδικας 3.9: Μέθοδος ανάκτησης γειτονιάς νευρώνα προς τα πάνω

Αυτές οι δύο μέθοδοι λειτουργούν αναδρομικά ανακτώντας τον νευρώνα που βρίσκεται προς τα αριστερά ή προς τα πάνω αντίστοιχα μέχρι να φτάσουν στον νευρώνα που η απόσταση του είναι ίση με το μέγεθος της γειτονιάς. Και τις δύο αυτές μεθόδους θα τις χρησιμοποιήσουμε κατά την εκπαίδευση του δικτύου για να ανακτήσουμε τον νευρώνα που βρίσκεται πάνω αριστερά στην γειτονιά του νικητή, τον νευρώνα δηλαδή από τον οποίο θα αρχίσει η εκπαίδευση.

3.2.6. Αρχικοποίηση του αυτοοργανούμενου χάρτη

Αφού εξηγήσαμε τις βασικές λειτουργίες του νευρώνα τώρα θα εξετάσουμε πως αρχικοποιούνται όλοι νευρώνες για να σχηματιστεί ο χάρτης μας. Τους νευρώνες μας τους κρατάμε στον πίνακα *som[ ][ ]* στον οποίο μπορούμε να δώσουμε όποιο μέγεθος θέλουμε. Αφού δημιουργήσουμε τον πίνακα τότε καλούμε την *initSOM* για να σχηματίσουμε τις σχέσεις ανάμεσα στους νευρώνες. Δηλαδή να θέσουμε τον νευρώνα που βρίσκετε πάνω, κάτω, δεξιά και αριστερά ως γειτονικό.

```
public void initSOM(){
   for(int i=0; i<som.length; i++){
      for(int j=0; j<som[0].length; j++){
         som[i][j]=new Neuron(i,j);
         if(i>0){
            som[i][j].setNeighbourLeft(som[i-1][j]);
            som[i-1][j].setNeighbourRight(som[i][j]);
         }
         if(i==som.length-1){
            som[0][j].setNeighbourLeft(som[som.length-1][j]);
            som[som.length-1][j].setNeighbourRight(som[0][j]);
         }
         if(j>0){
            som[i][j].setNeighbourUp(som[i][j-1]);
            som[i][j-1].setNeighbourDown(som[i][j]);
         }
         if(j==som[0].length-1){
            som[i][0].setNeighbourUp(som[i][som[0].length-1]);
            som[i][som[0].length-1].setNeighbourDown(som[i][0]);
         }
      }
   }
}
```

Κώδικας 3.10: Μέθοδος αρχικοποίησης αυτοοργανούμενου χάρτη

Η μέθοδος αυτή εξετάζει κάθε νευρώνα του πίνακα *som* και αν αυτός δεν βρίσκεται στις άκρες, δηλαδή όταν το *i* και *j* είναι μεγαλύτερα του μηδέν και μικρότερα από το μήκος του πίνακα θέτει ως νευρώνα αριστερά και δεξιά για τον τρέχων *ij* αυτόν που βρίσκεται στο *i-1* και *i+1* αντίστοιχα. Για τον πάνω και κάτω τον *j-1* και *j+1*. Στην περίπτωση που βρισκόμαστε στην αρχή τότε ο νευρώνας πάνω ή αριστερά θα είναι αυτός που βρίσκεται στο τέλος. Το αντίστροφο συμβαίνει όταν βρισκόμαστε στο τέλος του πίνακα.

3.2.7. Ο αλγόριθμος και η υλοποίηση της εκπαίδευσης

Μέχρι τώρα ότι είδαμε ήταν γύρω από τους νευρώνες και τις λειτουργίες τους. Τώρα που έχουμε εξετάσει όλες τις σημαντικές πτυχές του νευρώνα ήρθε η ώρα να μιλήσουμε για την εκπαίδευση. Ό αλγόριθμος μας για την εκπαίδευση ακολουθεί την βασική αρχή των εποχών, έτσι εκπαίδευση στους αυτοοργανούμενούς χάρτες όπως άλλωστε συμβαίνει σε όλα τα νευρωνικά δίκτυα λαμβάνει χώρα σε κάποιο συγκεκριμένο αριθμό εποχών[15]. Σε κάθε εποχή παρουσιάζουμε τις περιοχές από διάφορες εικόνες στο δίκτυο μας και ψάχνουμε τον νικητή για να γίνει εκπαίδευση σε αυτόν και την γειτονιά του σύμφωνα με το βήμα εκπαίδευσης. Πιο κάτω φαίνεται αναλυτικά ο αλγόριθμος που θα υλοποιήσουμε.

1. *Για την εποχή epoch=0 μέχρι την μέγιστη εποχή*
    a. *Μείωση του βήματος εκπαίδευσης beta ανά εποχή*
    b. *Υπολογισμός γειτονιάς ανά εποχή*
    c. *Για όλες τις κατατετμημένες εικόνες*
        i. *Για όλες τις περιοχές της εικόνας*
            1. *Εξέτασε ένα-ένα όλους τους νευρώνες του δικτύου*
                a. *Αν ο τρέχων νευρώνας έχει την μικρότερη απόσταση κράτησε τον ως νικητή*
            2. *Πήγαινε στην αρχή της γειτονιάς του νικητή νευρώνα*
            3. *Για όλους τους νευρώνες της γειτονιάς*
                a. *Παρουσίασε την περιοχή και εκπαίδευσε τον νευρώνα*

Κώδικας 3.11: Αλγόριθμος εκπαίδευσης του αυτοοργανούμενου χάρτη

Τώρα που είδαμε τον αλγόριθμο του αυτοοργανούμενου χάρτη πάμε να εξετάσουμε και την υλοποίηση του στην εφαρμογή μας. Τις μέγιστές εποχές, το μέγεθος της γειτονιάς, το αρχικό και τελικό βήμα εκπαίδευσης, καθώς και τις εικόνες που θα συμμετάσχουν στην εκπαίδευση, τα ορίζουμε όλα εμείς στα ορίσματα τη μεθόδου *trainSom* όπως πιο κάτω:

```
trainSom(int maxEpochs,
        int neighbourhood,
        double betaInitial,
        double betaFinal,
        SomImage images[]){…
```

Κώδικας 3.12: Ορίσματα μεθόδου trainSom

Για να υλοποιήσουμε το βήμα *1.* της εκπαίδευσης μας το μόνο που έχουμε να κάνουμε είναι ένα βρόγχο επανάληψης *for*.

```
for(int epoch=0; epoch<maxEpochs; epoch++){
```

Για τα βήματα *1.a* και *1.b* πρέπει να υπολογίσουμε το βήμα εκπαίδευσης και το μέγεθος της γειτονιάς ανά εποχή. Το βήμα εκπαίδευσης *beta* θα μειώνεται σταδιακά μέχρι να φτάσει από την αρχική τιμή στην τελική και το μέγεθος της γειτονιάς *neigh* θα μειώνεται σταδιακά μέχρι να γίνει μηδέν όπου θα γίνεται εκπαίδευση μόνο στον νικητή νευρώνα.

```
beta=betaInitial-((epoch/maxEpochs)*(betaInitial-betaFinal));
neigh=neighbourhood -1 - (int)((epoch*neighbourhood)/maxEpochs);
```

Στα βήματα *1.c* και *1.c.i* δύο *for*, στο ένα θα περάσουμε τις εικόνες που επιλέξαμε για την εκπαίδευση και στο δεύτερο θα περνάμε μια μία τις περιοχές που δημιουργήθηκαν από την κατάτμηση.

```
for(int i=0; i<images.length; i++){
    for(int j=0; j<images[i].getRegionsNo(); j++){
        RegionDataSet regData=images[i].getRegion(j);
```

Το βήμα *1.c.i.1* λέει ότι πρέπει να εξετάσουμε ένα-ένα τους νευρώνες του δικτύου μας. Αφού οι νευρώνες μας βρίσκονται στον πίνακα *som* θα πρέπει να δημιουργήσουμε δύο επαναλήψεις, μια για κάθε διάσταση του πίνακα ώστε να τους ελέγξουμε.

```
for(int r=0; r<som.length; r++)
    for(int c=0; c<som[0].length; c++){
```

Για το βήμα *1.c.i.1.a* θα παίρνουμε τους νευρώνες και εξετάζουμε αν ή απόσταση τους είναι μικρότερη από τον τρέχων νικητή. Αυτό που χρειαζόμαστε είναι ένας έλεγχος που θα συγκρίνει τις αποστάσεις των δύο νευρόνων απο την περιοχή που εξετάζουμε. Δηλαδή:

```
if(som[r][c].distance(regData.getShape(),regData.getPercentColor())
        <winner.distance(regData.getShape(),regData.getPercentColor())){
                winner=som[r][c];
}
```

Όταν ο νικητής βρεθεί, όπως λέει και στο βήμα *1.c.i.2* θα πρέπει να μεταβούμε στην αρχή της γειτονιάς όπου αρχίζει η εκπαίδευση. Για να το κάνουμε αυτό χρησιμοποιούμε τις μεθόδους *goUp* και *goLeft* της κλάσης *Neuron* που παρουσιάσαμε νωρίτερα.

```
neighLeftTop=winner.goLeft(0,neigh,winner);
neighLeftTop=neighLeftTop.goUp(0,neigh,neighLeftTop);)
```

Τώρα που βρισκόμαστε στην αρχή της γειτονιάς μπορούμε να μεταβούμε στο βήμα *1.c.i.3* όπου θα δημιουργήσουμε ένα for που θα περιλαμβάνει όλους τους νευρώνες μέσα στη γειτονιά όπου θα εκπαιδευτούν. Για αυτό θα χρησιμοποιήσουμε ένα νευρώνα, το τρέχον που θα αλλάζει κάθε φορά και θα τον ονομάσουμε *curr*. Θα χρησιμοποιούμε επίσης τις μεθόδους *getNeighbourRight* και *getNeighbourDown* της κλάσης *Neuron* για να μετακινούμαστε στους γειτονικούς νευρώνες.

Για κάθε τρέχων νευρώνα θα εκτελούμε την μέθοδο train όπως λέει και το βήμα *1.c.i.3.a*. Στην μέθοδο αυτή θα περάσουμε το χρώμα και το σχήμα της τρέχουσας περιοχής.

```
Neuron curr=neighLeftTop;
for(int r=0; r<=neigh*2; r++){
        for(int c=0; c<=neigh*2; c++){
                curr.train(regData.getShape(),regData.getPercentColor(),beta);
                curr=curr.getNeighbourRight();
         }
        neighLeftTop=neighLeftTop.getNeighbourDown();
       curr=neighLeftTop;
 }
```

Πιο κάτω μπορούμε να δούμε τον κώδικα της μεθόδου ολοκληρωμένο.

```
private void trainSom(int maxEpochs, int neighbourhood,
                        double betaInitial,double betaFinal,SomImage images[]){
        double beta,delta;
        int neigh;
        Neuron winner,neighLeftTop;

         for(int epoch=0; epoch<maxEpochs; epoch++){
                beta=betaInitial-((epoch/maxEpochs)*(betaInitial-betaFinal));
                neigh=neighbourhood-1-(int)((epoch*neighbourhood)
                                                        /maxEpochs);
                for(int i=0; i<images.length; i++){
                        for(int j=0; j<images[i].getRegionsNo(); j++){
                                RegionDataSet regData=images[i].getRegion(j);
                                winner=som[0][0];
                                //find winner neuron
                                for(int r=0; r<som.length; r++)
                                        for(int c=0; c<som[0].length; c++){
                                                if(som[r][c].distance(
                                                        regData.getShape(),
```

```
                                        regData.getPercentColor())
                                        <winner.distance(
                                        regData.getShape()
                                        ,regData.getPercentColor())){
                                         winner=som[r][c];
                                    }
                                }
                neighLeftTop=winner.goLeft(0,neigh,winner);
                neighLeftTop=neighLeftTop.goUp(0,neigh,neighLeftTop);
                Neuron curr=neighLeftTop;
                for(int r=0; r<=neigh*2; r++){
                        for(int c=0; c<=neigh*2; c++){
                                curr.train(
                                        regData.getShape(),
                                        regData.getPercentColor(),beta);
                                curr=curr.getNeighbourRight();
                         }
                         neighLeftTop=neighLeftTop.getNeighbourDown();
                        curr=neighLeftTop;
                }}}}}
```

Κώδικας 3.13: Η μέθοδος trainSom

3.2.8. Ο αλγόριθμος και η υλοποίηση της ανάκλησης

Η ανάκληση στους αυτοοργανούμενους χάρτες λειτουργεί με την ίδια ακριβώς λογική όπως η εκπαίδευση. Παρουσιάζουμε στο εκπαιδευμένο πλέον δίκτυο τις περιοχές από μία εικόνα και ψάχνουμε τον νικητή νευρώνα[16].

1. *Για όλες τις περιοχές της κατατετμημένης εικόνας*
    a. *Εξέτασε ένα-ένα όλους τους νευρώνες του δικτύου*
        i. *Αν ο τρέχων νευρώνας έχει την μικρότερη απόσταση κράτησε τον ως νικητή*
    b. *Καταχώρησε στην περιοχή τον νικητή νευρώνα*

Κώδικας 3.14: Αλγόριθμος ανάκλησης στον αυτοοργανούμενο χάρτη

Αφού βρεθεί ο νικητής, καταχωρείτε στην συγκεκριμένη περιοχή ως νικητής για να χρησιμοποιηθεί αργότερα από την εφαρμογή μας στην εύρεση των συσχετίσεων ανάμεσα στις περιοχές.

Για την υλοποίηση της ανάκλησης χρησιμοποιούνται οι μέθοδοι που είδαμε και στην εκπαίδευση με εξαίρεση την *setWinnerNeuron* της κλάσης *RegionDataSet* για να καταχωρηθεί ο νικητής. Πιο κάτω ακολουθεί ο κώδικας.

```
private void recall(SomImage image){
   Neuron winner=som[0][0];
   Neuron noMemory=null;
   boolean hasNoMem=false;
   RegionDataSet regData;
   for(int i=0; i<image.getRegionsNo(); i++){
      regData=image.getRegion(i);
      for(int r=0; r<som.length; r++){
         for(int c=0; c<som[0].length; c++){
            if(som[r][c].distance(regData.getShape(),regData.getPercentColor())
              <winner.distance(regData.getShape(),regData.getPercentColor()))
              {
                    winner=som[r][c];
              }
         }
      }
      regData.setWinnerNeuron(winner);
   }
}
```

Κώδικας 3.15: Αλγόριθμος ανάκλησης στον αυτοοργανούμενο χάρτη

3.3. Συμπεράσματα

Σε αυτό το κεφάλαιο δημιουργήσαμε έναν αυτοοργανούμενο χάρτη που παίρνει τις κατατετμημένων περιοχών και βρίσκει τους νικητές νευρώνες για την κάθε μία. Αργότερα οι νικητές αυτοί θα χρησιμοποιηθούν στην εύρεση συσχετίσεων που στοχεύει στην δημιουργία ενός προφίλ που θα καθορίζει τα

αντικείμενα. Αυτό το κομμάτι του μοντέλου μας είναι αλληλένδετο με την εύρεση των συσχετίσεων ανάμεσα στις περιοχές και για αυτό το λόγο θα παρουσιάσουμε τις διάφορες δοκιμές που έχουμε κάνει μόνο όταν ολοκληρωθούν και τα δύο μέρη.

Κεφάλαιο 4

Αντικείμενα και Συσχετίσεις

4.1. Εισαγωγή

Μετά την ανάλυση της εικόνας μέσω της κατάτμησης και την εισαγωγή των δεδομένων μας στον αυτοοργανούμενο χάρτη έχουμε ότι χρειαζόμαστε για να σχηματίσουμε ένα προφίλ συσχετίσεων για τα αντικείμενα. Η συσχετίσεις αυτές θα παίζουν τον ρόλο που έχει η μνήμη στον ανθρώπινο εγκέφαλο. Μνήμη καθορίζεται ως η ικανότητα του εγκεφάλου να ανακτά πληροφορίες που κατάφερε να μάθει κάποια προηγουμένη στιγμή και τις κρατάει σε ένα σύστημα εσωτερικής αποθήκευσης ώστε να τις ανακαλεί όποτε τις χρειαστεί[20]. Μνήμη εμείς στο πρόγραμμα μας αποκαλούμε την ικανότητα αποθήκευσης των διαφόρων ερεθισμάτων, δηλαδή χαρακτηριστικών που προκύπτουν με την παρουσίαση μιας εικόνας στο σύστημα μας. Στον άνθρωπο η μνήμη ακολουθεί την διαδικασία[20]:

1. κωδικοποίηση
2. διατήρηση
3. ανάκτηση

Αυτό ακριβώς θα προσπαθήσουμε να αντιγράψουμε κι εμείς.

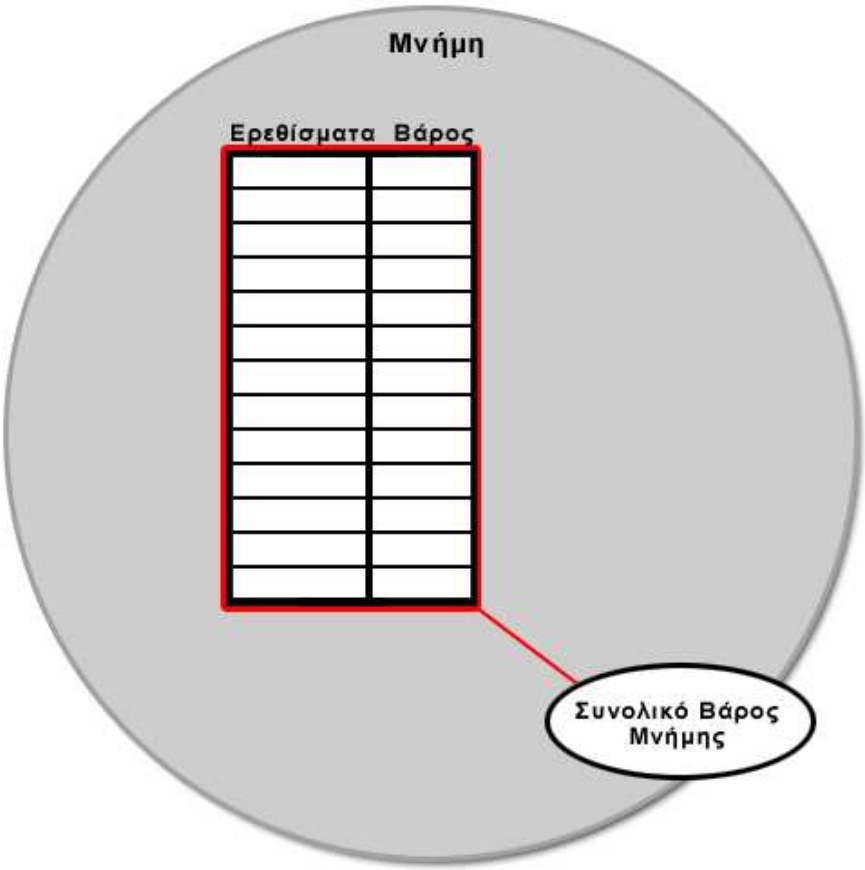
4.2. Η αντίληψη των αντικειμένων

Η αντίληψη αντικειμένων είναι η διαδικασία αναγνώρισης των ερεθισμάτων από μια εικόνα[21]. Τα ερεθίσματα που έχουμε εμείς στην διάθεση μας από τα προηγούμενα βήματα του μοντέλου μας είναι τα εξής:

- Η θέση των νικητών νευρώνων που αντιπροσωπεύουν το χρώμα και το σχήμα των περιοχών
- Το μέγεθος των περιοχών σε σχέση με τις άλλες
- Η θέση των περιοχών σε σχέση με τις άλλες

Δεδομένου ότι εικόνες που παρουσιάζουν το ίδιο αντικείμενο έχουν τα ίδια η παρόμοια χαρακτηριστικά τότε εμείς θα πρέπει εμείς να τα κωδικοποιήσουμε αυτά έτσι ώστε κάθε φορά που παρουσιάζουμε μια εικόνα στο δίκτυο μας να ενεργοποιεί αυτά τα ερεθίσματα και να μπορεί να επιλέξει σε πια ανάμνηση θα ενεργοποιήσει. Τα ερεθίσματα μας θα τα αποθηκεύουμε κωδικοποιημένα μέσα σε ένα μονοδιάστατο αυτοοργανούμενο χάρτη μνήμης, και θα τους δίνουμε με

κάποιο τρόπο αξιολόγησης ένα βάρος που αντιπροσωπεύει το πόσο σημαντική είναι η παρουσία τους μέσα στο αντικείμενο. Κάθε φορά που ανακαλείτε μια μνήμη θα ελέγχονται τα βάρη της με αυτά των αντικείμενων που κατάφεραν να αντιστοιχηθούν. Αν το συνολικό βάρος είναι κοντά σε αυτό που παρουσιάζει συνήθως το αντικείμενο αυτού του τύπου τότε η ανάμνηση θα ενεργοποιείται. Στην εικόνα πιο κάτω φαίνετε η αναπαράσταση μιας μνήμης όπως την περιγράψαμε.



Εικόνα 4.1: Θεωρητική αναπαράσταση μνήμης στο μοντέλο μας.

4.3. Υλοποίηση της μνήμης σε Java

Όλες οι αναμνήσεις που κρατάει το μοντέλο μας στην μνήμη του βρίσκονται στο αντικείμενο *memories* της κλάσης *Memories* που βρίσκεται στην κλάση *SOM*. Η *Memories* με την βοήθεια ενός vector παρέχει όλες τις μεθόδους δημιουργίας μιας ανάμνησης η ανάκτησης της είτε με το όνομα περιγραφής, είτε

με την θέση που βρίσκετε στο vector. Πιο κάτω βλέπουμε τις τέσσερεις μεθόδους της κλάσης Memories:

- *Memory getMemory(int index)*
- *Memory getMemory(String description)*
- *addMemory(Memory memory)*

Το vector της κλάσης *Memories* περιέχει αντικείμενα της κλάσης *Memory*. Η κλάση *Memory* που καθορίζεται με μια περιγραφή, είναι υπεύθυνη στο να κρατά τις συσχετίσεις μέσω του αντικειμένου relations της κλάσης *Relations*, καθώς και να υπολογίζει και το συνολικό βάρος της μνήμης έχοντας ένα αντικείμενο της κλάσης *MemoryNeuron*. Είναι υπεύθυνη επίσης για την εκπαίδευση των συσχετίσεων, την προσαρμογή του συνολικού βάρους στις συσχετίσεις και τον υπολογισμό της ομοιότητας της μνήμης με μια εικόνα που της παρουσιάζουμε. Όλα αυτά θα τα αναλύσουμε στις ενότητες που ακλουθούν.

4.3.1. Δημιουργία των συσχετίσεων μέσα στο μοντέλο μας

Όλες οι συσχετίσεις βρίσκονται όπως είπαμε στην κλάση *Relations*. Η *Relations* περιέχει 2000 αντικείμενα της κλάσης *Relation* -ένας αριθμός που επιλέξαμε εμείς- και ουσιαστικά είναι υπεύθυνη μόνο για την διαχείριση τους. Η λειτουργίες των συσχετίσεων κρύβονται στην κλάση *Relation* και η λειτουργία όλων συνολικά ως ένας μονοδιάστατος αυτοεπαινούμενος χάρτης βρίσκεται στην κλάση Memory. Οι μέθοδοι διαχείρισης της κλάσης Relations είναι οι παρακάτω:

- *Relations getRelations()*
- *Relation getRelation(int index)*
- *int getRelationsNo()*

Όλες οι συσχετίσεις δημιουργούνται και αρχικοποιούνται σε τυχαίες τιμές όταν δημιουργηθεί το αντικείμενο *relations* στην κλάση *Memories*.

Η κλάση Relation αποτελεί την συσχέτιση ανάμεσα σε δύο νευρώνες που αντιπροσωπεύουν με τη σειρά τους δύο περιοχές. Κρατάει επίσης και τα χαρακτηριστικά του πρώτου νευρώνα σε σχέση με τον δεύτερο, δηλαδή το μέγεθος και την θέση. Οι δύο νευρώνες στη συσχέτιση αντιπροσωπεύονται από την θέση τους στον αυτοοργανούμενο χάρτη, η θέση του πρώτου από τους δύο

αποθηκεύεται στις μεταβλητές *neuronW* και *neuronH* ενώ για τον δεύτερο στις μεταβλητές *relativeW* και *relativeH*. Το μέγεθος και τη θέση της πρώτης περιοχής σε σχέση με την δεύτερη την κρατάμε στις μεταβλητές *size* για το μέγεθος και τις *offsetW* και *offsetH* για την θέση. Όλες αυτές οι μεταβλητές αρχικά θα πάρουν μια τυχαία τιμή. Κάθε μία συσχέτιση έχει επίσης την μεταβλητή strength που κρατά το βάρος που έχει η συσχέτιση για το αντικείμενο.

Μία συσχέτιση μπορεί όμως να δημιουργηθεί και από δύο περιοχές αφού από αυτές θα προσαρμοστούν σιγά-σιγά οι συσχετίσεις που θα συμμετέχουν στο μοντέλο μας. Η θέση των νευρώνων ανακτάτε από αυτούς ενώ το για τα υπόλοιπα που είναι σχετικά με το μέγεθος της πρώτης περιοχής μπορούμε να τα υπολογίσουμε καλώντας την μέθοδο *findTransformation* της *RegionDataSet*. Η *RegionDataSet* απλά δέχεται ένα όρισμα που μπορεί να είναι οποιοδήποτε από τα παραπάνω και το επιστρέφει διαιρεμένο με το μέγεθος της περιοχής. Δηλαδή:

```
public double findTransformation(double regionParameter){
    return regionParameter/elements;}
```

Κώδικας 4.1: Η μέθοδος findTransformation.

Αν δούμε για παράδειγμα την εικόνα πιο κάτω του Daffy Duck βλέπουμε ότι μετά την κατάτμηση σχηματίζονται οι περιοχές στα δεξιά. Αν θέλουμε να βρούμε για τη συσχέτιση των περιοχών 1 και 2 παρατηρούμε ότι το μέγεθος του 2 σε σχέση με το 1 να είναι περίπου στο 1/3, η απόσταση του 2 από το κέντρο του 1 περίπου είναι μια φορά το μέγεθος του 2.

Εικόνα 4.2.: Δημιουργία συσχετίσεων στις περιοχές 1 και 2.

Στον κώδικα 4.2 μπορούμε να δούμε τον δομητή της κλάσης Relation που στα ορίσματα παίρνει δύο περιοχές και σχηματίζει την συσχέτιση.

```
public Relation(RegionDataSet region1,RegionDataSet region2){
    neuronW=region1.getWinnerNeuron().getPosW();
    neuronH=region1.getWinnerNeuron().getPosH();
    relativeW=region2.getWinnerNeuron().getPosW();
    relativeH=region2.getWinnerNeuron().getPosH();

    size=region1.findTransformation(region2.getElements());
    offsetW=region1.findTransformation(region2.getWOffset());
    offsetH=region1.findTransformation(region2.getHOffset());

    strength=1;
}
```

Κώδικας 4.2: Δημιουργία συσχέτισης από δύο περιοχές.

4.3.2. Ο υπολογισμός απόστασης στις συσχετίσεις

Όπως είπαμε οι συσχετίσεις μας θα οργανώνονται και θα προσαρμόζονται στο αντικείμενο που τους παρουσιάζουμε μέσω ενός μονοδιάστατου αυτοοργανούμενου χάρτη. Την οργάνωση αυτή που υποκινεί ο ανταγωνισμός ανάμεσα στους νευρώνες τον ρόλο των οποίων έχουν οι συσχετίσεις και ο νικητής καθορίζεται από την απόσταση. Η απόσταση στις συσχετίσεις δεν μπορεί παρά να καθορίζεται από τα χαρακτηριστικά τους άρα γίνεται ως εξής:

$$\sqrt{(size\text{-}sizeData)^2}+\sqrt{(offSet\text{-}offsetData)^2}+\sqrt{(neuronPos\text{-}neuronPosData)^2}+\sqrt{(relativePos\text{-}relativePosData)^2}$$

Η μέθοδος που υπολογίζει την απόσταση είναι η *relationDistance* που δέχεται στα ορίσματα της το μέγεθος του δικτύου στο στρώμα Kohonen του αυτοοργανούμενου χάρτη και μία συσχέτιση για να υπολογίσει την απόσταση σε σχέση με αυτήν.

```
public double relationDistance(Relation r,int netSize){
    double dist,sDist,offWDist,offHDist,neuDist,relDist,normal;
    normal=netSize/2;
    sDist=distance(size,r.getSize())*normal;
```

```
    offWDist=distance(offsetW,r.getOffsetW())*normal;
    offHDist=distance(offsetH,r.getOffsetH())*normal;
   neuDist=neuronDistance(neuronW,neuronH,
                          r.getNeuronW(),r.getNeuronH(),netSize);
    relDist= neuronDistance(relativeW,relativeH,
                          r.getRelativeW(),r.getRelativeH(),netSize);
    dist=sDist+offWDist+offHDist+relDist+neuDist;

    return dist;
  }
```

Κώδικας 4.3: Μέθοδος υπολογισμού απόστασης συσχετίσεων.

Η απόσταση όλων των χαρακτηριστικών εκτός από αυτήν της θέσης των δύο νευρώνων υπολογίζεται χρησιμοποιώντας τον κανονικό τύπο της ευκλείδειας απόστασης μέσω της μεθόδου πιο κάτω:

```
  public double distance(double val1,double val2){
    double dist;
    dist=Math.pow(val1-val2,2);
    dist=Math.sqrt(dist);

    return dist;
  }
```

Κώδικας 4.4: Μέθοδος υπολογισμού απόστασης δύο μεταβλητών.

Στην απόσταση τώρα των δύο νευρώνων πρέπει να λάβουμε υπόψη το γεγονός ότι ο αυτοοργανούμενος χάρτης είναι κυκλικός. Άρα η απόσταση των νευρώνων μπορεί να είναι μικρότερη αν την υπολογίσουμε χρησιμοποιώντας τις άκρες του χάρτη μας. Η μέθοδος *neuronDistance* δέχεται τις θέσεις δύο νευρώνων και το μέγεθος του δικτύου και λαμβάνοντας υπόψη το γεγονός αυτό υπολογίζει την απόσταση αυτή που στην ουσία δεν μπορεί να είναι μεγαλύτερη από το μισό του χάρτη.

```
private double neuronDistance(double n1w,double n1h,
                                        double n2w,double n2h,int netSize){

    //--------WIDHT--------
    double distW1=Math.pow(n1w-n2w,2);//Kanoniki apostasi

    double minW,maxW;//apostasi apo tis akres tou som
    if(n1w<n2w){minW=n1w; maxW=n2w;}
    else{maxW=n1w; minW=n2w;}
    double distW2=Math.pow((netSize+minW)-maxW,2);

    //--------HEIGHT--------
    double distH1=Math.pow(n1h-n2h,2);//Kanoniki apostasi

    double minH,maxH;//apostasi apo tis akres tou som
    if(n1h<n2h){minH=n1h; maxH=n2h;}
    else{maxH=n1h; minH=n2h;}
    double distH2=Math.pow((netSize+minH)-maxH,2);
    //----------------------

    double distW,distH;
    if(distW1<distW2)distW=distW1;
    else distW=distW2;

    if(distH1<distH2)distH=distH1;
    else distH=distH2;

    return Math.sqrt(distW+distH);
}
```

Κώδικας 4.5: Μέθοδος υπολογισμού απόστασης δύο νευρώνων.

Αρχικά πρέπει να βρούμε ποιος νευρώνας είναι σε μικρότερη θέση από τον άλλον και αμέσως μετά υπολογίζουμε την κανονική απόσταση και την απόσταση από τις άκρες του χάρτη για να επιλέγουμε αυτήν που είναι η μικρότερη

4.3.3. Η εκπαίδευση της συσχέτισης

Η εκπαίδευση της νικήτριας συσχέτισης που καθορίζεται από την μικρότερη απόσταση γίνεται από την μέθοδο *trainRelation* που δέχεται την συσχέτιση που παρουσιάσαμε στο δίκτυο, το βήμα εκπαίδευσης και το μέγεθος δικτύου που είναι απαραίτητο στην εκπαίδευση της θέσης των δύο νευρώνων.

```
public void trainRelation(Relation r,double beta,int netSize){
    size+=getDelta(size,r.getSize(),beta);
    offsetW+=getDelta(offsetW,r.getOffsetW(),beta);
    offsetH+=getDelta(offsetH,r.getOffsetH(),beta);
    neuronW+=neuronBoundsDelta(neuronW,r.getNeuronW(),netSize,beta);
    neuronH+=neuronBoundsDelta(neuronH,r.getNeuronH(),netSize,beta);
    relativeW+=neuronBoundsDelta(relativeW,r.getRelativeW(),netSize,beta);
    relativeH+=neuronBoundsDelta(relativeH,r.getRelativeH(),netSize,beta);
}
```

Κώδικας 4.6: Μέθοδος εκπαίδευσης συσχετίσεων.

Σε όλα τα χαρακτηριστικά η εκπαίδευση γίνεται προσθέτοντας το δέλτα που είναι ίσο με την διαφορά των δύο χαρακτηριστικών επί το βήμα εκπαίδευσης

```
private double getDelta(double value1,double value2,double beta){
    double delta;
    delta = beta * (value1 - value2);

    delta=cutDigits(delta);
    return delta;
}
```

Κώδικας 4.7: Υπολογισμός δέλτα.

Για τους νευρώνες όμως ισχύει και πάλι το ότι πρέπει να λάβουμε υπόψη το ότι ίσως η διαφορά από τα άκρα ίσως να είναι μικρότερη. Άρα το δέλτα για τους δύο νευρώνες υπολογίζεται από την μέθοδο *neuronBoundsDelta*. Και στις δύο μεθόδους η *cutDigits* κάνει μερική στρογγυλοποίηση στο δέλτα μας.

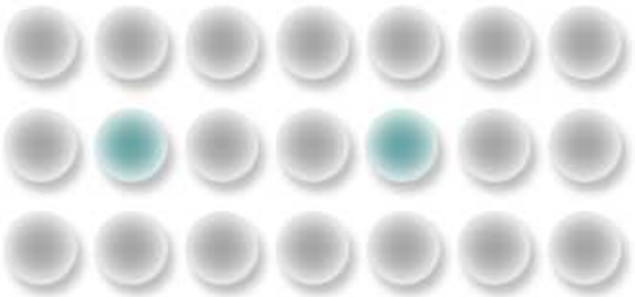
```
private double neuronBoundsDelta(double n1,double n2,int netSize,double beta){
    double d1,d2,nSize,dist,delta;
    if(n1<n2)nSize=netSize;
    else nSize=netSize-(netSize*2);
    d1=n1-n2;
    d2=d1+nSize;
    if(d1<d2)dist=d1;
    else dist=d2;

    delta = beta * dist;

    delta=cutDigits(delta);
    return delta;
}
```

Κώδικας 4.8: Υπολογισμός δέλτα για τους νευρώνες.

Εδώ η κανονική διαφορά υπολογίζεται από το *n1-n2* ενώ για τη διαφορά από τα όρια του χάρτη προσθέτουμε ή αφαιρούμε το μέγεθος του χάρτη από την κανονική διαφορά ανάλογα με το αν το *n1* ή το *n2* είναι μεγαλύτερο, δηλαδή.



Εικόνα 4.3: Υπολογισμός διαφοράς εκπαίδευσης νευρώνων.

Στην εικόνα 4.3 βλέπουμε ένα κομμάτι από ένα αυτοοργανούμενο χάρτη με μέγεθος 7. Αν υποθέσουμε ότι ο νευρώνας εκπαίδευσης *n1* είναι ο μπλε στα αριστερά και ο *n2* ο μπλε στα δεξιά τότε η κανονική απόσταση είναι 1-4=-3 και η απόσταση από τις άκρες. Στην περίπτωση που η θέση του n1 είναι μικρότερη από αυτήν του *n2*, όπως εδώ τότε για την διαφορά από τις άκρες προσθέτουμε

στην κανονική απόσταση το μέγεθος του χάρτη και έχουμε -3+7=4. Στην περίπτωση που ο νευρώνας εκπαίδευσης *n1* είναι αυτός στα δεξιά και ο *n2* στα αριστερά τότε η κανονική διαφορά είναι 4-1=3 και η διαφορά από τα άκρα γίνεται 4+7-(7*2)=-3 αφού το *n1* είναι μεγαλύτερο από το *n2*.

4.3.4. Αύξηση και μείωση του βάρους συσχέτισης

Η αυξομείωση του βάρους μιας συσχέτισης είναι μια πολύ απλή λειτουργία. Κάθε φορά που παρουσιάζουμε μια συσχέτιση στις συσχέτισης του αντικειμένου αυτή θα βρίσκει με ποια ταιριάζει και αν η περιγραφή των αντικειμένων είναι η ίδια τότε το βάρος θα αυξάνεται, αλλιώς σε περίπτωση που είναι διαφορετική θα μειώνεται. Η μέθοδοι για την αύξηση και μείωση είναι οι παρακάτω.

```
public void increaseStrenght(double distance){
    double increase=1/distance;
    strength+=increase;
}
public void decreaseStrenght(double distance){
    double decrease=1/distance;
    strength-=decrease;
}
```

Κώδικας 4.9: Αύξηση και μείωση βάρους.

Η ιδιαιτερότητα των δύο μεθόδων είναι ότι αυξάνουν το βάρος της συσχέτισης ανάλογα με την απόσταση που έχουν οι δύο μεταξύ τους. Αν η απόσταση είναι μεγάλη τότε η αύξηση είναι μικρή, και αν η απόσταση είναι μικρή τότε η αύξηση είναι μεγαλύτερη.

4.3.5. Εκπαίδευση αναμνήσεων

Η εκπαίδευση των αναμνήσεων ακολουθεί τον αλγόριθμο εκπαίδευσης ενός μονοδιάστατου αυτοοργανούμενου χάρτη. Για να υλοποιηθεί η εκπαίδευση σε μια εικόνα πρέπει να υπάρχει ένα βήμα εκπαίδευσης, το μέγεθος γειτονιάς και το μέγεθος του αυτοοργανούμενου χάρτη που χρησιμοποιήθηκε στην κατάτμηση.

Δηλαδή για κάθε εικόνα που του παρουσιάζουμε ο αλγόριθμος εκπαίδευσης έχει ως εξής:

1. *Για όλες τις περιοχές της εικόνας i=0*
    a. *Για όλες τις περιοχές της εικόνας j=i+1*
        i. *Δημιούργησε συσχέτιση ως προς την μεγαλύτερη περιοχή*
        ii. *Εξέτασε μια-μια όλες τις συσχετίσεις της ανάμνησης*
            1. *Αν η τρέχουσα συσχέτιση έχει την μικρότερη απόσταση κράτησε την ως πιο όμοια*
        iii. *Αν το αντικείμενο εκπαίδευσης δεν είναι το ίδιο με της ανάμνησης*
            1. *Μείωσε το βάρος συσχέτισης και τερμάτισε τον αλγόριθμο*
        iv. *Για την πιο όμοια συσχέτιση και τη γειτονιά της*
            1. *Εκπαίδευσε τις συσχετίσεις και αύξησε το βάρος τους*

Κώδικας 4.10: Αλγόριθμος εκπαίδευσης συσχετίσεων ανάμνησης

Τον αλγόριθμο αυτό υλοποιεί η μέθοδος *trainRelaitions* πιο κάτω σε συνδυασμό με την μέθοδο *trainRelations* της κλάσης SOM όπου για ένα αριθμό εποχών υπολογίζει το βήμα εκπαίδευσης και το μέγεθος της γειτονιάς πριν καλέσει της πρώτη θέτοντας όλα τα απαραίτητα ορίσματα.

```
public void trainRelations(SomImage image,double beta, int neigh,int netSize){
    double minDist;
    int closestRel=0;
    int rDown,rUp;
    RegionDataSet region1,region2;
    Relation relation;

    for(int i=0; i<image.getRegionsNo(); i++){
       region1=image.getRegion(i);
       for(int j=i+1; j<image.getRegionsNo(); j++){
          region2=image.getRegion(j);

          if(region1.getElements()>region2.getElements())
```

```
    relation=new Relation(region1,region2);
else
    relation=new Relation(region2,region1);

minDist=Double.MAX_VALUE;
closestRel=-1;
//find winner relation
for(int r=0; r<getRelationsNo(); r++){
    if(relation.relationDistance(relations.getRelation(r),netSize)
                                                        <minDist){
        closestRel=r;
        minDist=relation.relationDistance(relations.getRelation(r),netSize);
    }
}
if(!description.equals(image.getDescription())){
    relations.getRelation(closestRel).decreaseStrenght(minDist);
    break;
}

relations.getRelation(closestRel).trainRelation(relation,beta,netSize);
relations.getRelation(closestRel).increaseStrenght(minDist);
for(int r=1; r<=neigh; r++){
    rDown=closestRel-r;
    rUp=closestRel+r;

    if(rDown<0)rDown=getRelationsNo()+rDown;
    if(rUp>=getRelationsNo())rUp=rUp-getRelationsNo();
    relations.getRelation(rDown).trainRelation(relation,beta,netSize);
    relations.getRelation(rDown).increaseStrenght(minDist);

    relations.getRelation(rUp).trainRelation(relation,beta,netSize);
    relations.getRelation(rUp).increaseStrenght(minDist);
}}}}
```

Κώδικας 4.11: Μέθοδος εκπαίδευσης συσχετίσεων ανάμνησης

4.3.6. Νευρώνες ανάμνησης

Για να υπολογίσουμε το συνολικό βάρος που έχουν οι εικόνες όταν ενεργοποιήσουν μια ανάμνηση χρησιμοποιούμε ένα νευρώνα μνήμης. Οι νευρώνες μνήμης υλοποιούνται από την κλάση *MemoryNeuron* και ακολουθεί το γενικό μοντέλο εκπαίδευσης και απόστασης που είδαμε μέχρι τώρα.

Για την εκπαίδευσης χρησιμοποιούμε την train που παίρνει το βάρος του αντικειμένου εκπαίδευσης *str* και το βήμα εκπαίδευσης για να υπολογίζει το δέλτα όπου προστίθεται στο βάρος του νευρώνα.

```
public void train(double str,double beta){
   double delta;
   //train relation strenght
   delta = beta * (str - relationStrenght);
   relationStrenght+=delta;
}
```

Κώδικας 4.12: Εκπαίδευση νευρώνα ανάμνησης

Για τον υπολογισμό της απόστασης από την ανάμνηση καλούμε την μέθοδο *distance* όπου της δίνουμε το συνολικό βάρος που υπολογίσαμε και αυτή απλά υλοποιεί τον τύπο υπολογισμού της ευκλείδειας απόστασης.

```
public double distance(double str){
   double dist;
   dist=Math.pow(relationStrenght-str,2);
   dist=Math.sqrt(dist);
   return dist;
}
```

Κώδικας 4.13: Υπολογισμός απόστασης νευρώνα ανάμνησης

4.3.7. Εκπαίδευση συνολικού βάρους

Για την εκπαίδευση δίνουμε μια εικόνα του αντικειμένου της ανάμνησης και βρίσκουμε τις συσχετίσεις της. Αντιστοιχούμε τις συσχετίσεις και υπολογίζουμε το συνολικό του βάρος, ακολούθως το δίνουμε στον νευρώνα ανάμνησης για εκπαίδευση με την μέθοδο *train*. Ο αλγόριθμος πιο κάτω υλοποιεί την

εκπαίδευση για κάθε εικόνα που του παρουσιάζουμε παίρνοντας ένα βήμα εκπαίδευσης.

1. *Για όλες τις περιοχές της εικόνας i=0*
    b. *Για όλες τις περιοχές της εικόνας j=i+1*
        i. *Δημιούργησε συσχέτιση ως προς την μεγαλύτερη περιοχή*
        ii. *Εξέτασε μια-μια όλες τις συσχετίσεις της ανάμνησης*
            1. *Αν η τρέχουσα συσχέτιση έχει την μικρότερη απόσταση κράτησε την ως πιο όμοια*
        iii. *Πρόσθεσε στο συνολικό βάρος εκπαίδευσης το βάρος της πιο όμοιας συσχέτισης*
2. *Εκπαίδευσε τον νευρώνα ανάμνησης με το συνολικό βάρος εκπαίδευσης*

Κώδικας 4.14: Αλγόριθμος εκπαίδευσης νευρώνων ανάμνησης

Τον αλγόριθμο αυτό υλοποιεί η μέθοδος *trainNeuron* σε συνδυασμό με την μέθοδο *trainMemoryNeurons* της κλάσης SOM όπου για ένα αριθμό εποχών υπολογίζει το βήμα εκπαίδευσης πριν καλέσει της πρώτη θέτοντας όλα τα απαραίτητα ορίσματα.

```
public void trainNeuron(SomImage image,double beta,int netSize){
   int relationStrenght=0;
   double minDist;
   int relationPos;
   RegionDataSet region1,region2;
   Relation relation;

   for(int j=0; j<image.getRegionsNo(); j++){
      region1=image.getRegion(j);
      for(int k=j+1; k<image.getRegionsNo(); k++){
         region2=image.getRegion(k);

         if(region1.getElements()>region2.getElements())
            relation=new Relation(region1,region2);
         else
            relation=new Relation(region2,region1);
```

```
            relationPos=-1;
            minDist=Double.MAX_VALUE;
            for(int i=0; i<getRelationsNo(); i++)
               if(relations.getRelation(i).relationDistance(relation,netSize)
                                                              <minDist ){
                  relationPos=i;
                  minDist=relations.getRelation(i).relationDistance(relation,netSize);
               }
            relationStrenght+=relations.getRelation(relationPos).getStrenght();
         }
      }
      memoryNeuron.train(relationStrenght,beta);
   }
```

Κώδικας 4.15: Μέθοδος εκπαίδευσης νευρώνων ανάμνησης

4.3.8. Ανάκληση συνολικού βάρους και εύρεση νικήτριας ανάμνησης

Η μέθοδος αυτή είναι το τελευταίο λιθαράκι στο μοντέλο μας. Για την ανάκληση δίνουμε μια εικόνα αγνώστου αντικειμένου στην ανάμνηση και βρίσκουμε τις συσχετίσεις της. Αντιστοιχούμε τις συσχετίσεις και υπολογίζουμε το συνολικό του βάρος, ακολούθως το δίνουμε στον νευρώνα ανάμνησης για να υπολογίσει την απόσταση από την ανάμνηση με την μέθοδο *distance*. Νικήτρια ανάμνηση θα είναι αυτή που η εικόνα έχει την μικρότερη απόσταση από αυτήν. Ο αλγόριθμος υλοποίησης της ανάκλησης είναι ο ίδιος με της εκπαίδευσης απλά στο τελευταίο βήμα αντί να καλέσουμε την μέθοδο *train* καλούμε την μέθοδο *distance*.

```
   public double recallImage(SomImage image,int netSize){
      int relationStrenght=0;
      double minDist;
      int relationPos;
      RegionDataSet region1,region2;
      Relation relation;
```

```
        for(int j=0; j<image.getRegionsNo(); j++){
            region1=image.getRegion(j);
            for(int k=j+1; k<image.getRegionsNo(); k++){
                region2=image.getRegion(k);

                if(region1.getElements()>region2.getElements())
                    relation=new Relation(region1,region2);
                else
                    relation=new Relation(region2,region1);

                relationPos=-1;
                minDist=Double.MAX_VALUE;
                for(int i=0; i<getRelationsNo(); i++)
                    if(relations.getRelation(i).relationDistance(relation,netSize)
                                                                    <minDist ){
                        relationPos=i;
                        minDist=relations.getRelation(i).relationDistance(relation,netSize);
                    }
                relationStrenght+=relations.getRelation(relationPos).getStrenght();
            }
        }
        return memoryNeuron.distance(relationStrenght);
    }
```

Κώδικας 4.16: Μέθοδος ανάκλησης νευρώνων ανάμνησης

Για την ανάκλησης της καταλληλότερης μνήμης καλούμε την *stormRecall* της κλάσης *SOM* που δίνει σε όλες τις αναμνήσεις την εικόνα που θα γίνει η ανάκληση και επιλέγει την καταλληλότερη.

```
    public String stormRecall(SomImage image){
        recall(image);
        double minDist=Double.MAX_VALUE;
        int winner=0;
        for(int i=0; i<memories.memoriesNo(); i++){
            if(memories.getMemory(i).recallImage(image,netSize)<minDist){
```

```
            minDist=memories.getMemory(i).recallImage(image,netSize);
            winner=i;
          }
        }
        return memories.getMemory(winner).getDescription();
    }
```

Κώδικας 4.17: Μέθοδος ανάκλησης εικόνας

4.4. Συμπεράσματα

Σε αυτό το κεφάλαιο εξετάσαμε την αντίληψη αντικειμένων και σύμφωνα με αυτή προχωρήσαμε στην δημιουργία, εκπαίδευση και ανάκληση συσχετίσεων. Αυτό το κεφάλαιο είδαμε το τελευταίο κομμάτι του μοντέλου μας. Συμπληρώσαμε συνολικά τη διαδικασία και για την ανάκληση, αρχικά τοποθετήσαμε την κατατετμημένη εικόνα στον αυτοοργανούμενο χάρτη όπου βρίσκει τους νικητές νευρώνες κάθε περιοχής, στην συνέχεια δημιουργήσαμε τις συσχετίσεις, υπολογίσαμε το συνολικό βάρος και συγκρίνοντας το με κάθε ανάμνηση επιλέξαμε την καταλληλότερη. Στο επόμενο κεφάλαιο θα δοκιμάσουμε το μοντέλο μας και θα κάνουμε όποιες αλλαγές κρίνουμε απαραίτητες.

Κεφάλαιο 5

Δοκιμές και εξέλιξη του μοντέλου μας

5.1. Εισαγωγή

Σε αυτό το κεφάλαιο θα βάλουμε το μοντέλο μας στο μικροσκόπιο και θα το υποβάλουμε σε δοκιμές. Θα δημιουργήσουμε ένα σύστημα ανάκτησης εικόνας για cartoons και μέσα από αυτό θα αλλάξουμε τα σημεία του μοντέλου μας που νομίζουμε ότι υστερούν στην πράξη. Επιλέξαμε η δοκιμή μας να γίνει σε εικόνες από cartoons γιατί θεωρούμε ότι η πολυπλοκότητα που έχουν οι εικόνες του πραγματικού κόσμου είναι πολύ δύσκολο να προσεγγιστούν σε αυτό το σημείο στο οποίο βρίσκεται το μοντέλο μας. Μπορούμε όμως μέσα από τα cartoons να δοκιμάσουμε την θεωρία μας και να εξελίξουμε το μοντέλο μας ώστε να έρθει πιο κοντά σε αυτό που προσπαθούμε να πετύχουμε. Τις δοκιμές μας θα τις κάνουμε χρησιμοποιώντας στην εκπαίδευση εικόνες του Bugs Bunny, Duffy Duck, Pink Panther και Tweety.

Εικόνα 5.1: Οι χαρακτήρες εκπαίδευσης.

Σε κάθε ενότητα θα κάνουμε μια δοκιμή χρησιμοποιώντας στην εκπαίδευση 28 εικόνες 7 για κάθε χαρακτήρα και στην ανάκληση συνολικά 49 εικόνες των χαρακτήρων. Σε κάθε δοκιμή θα αξιολογούμε τα αποτελέσματα και θα αναλύουμε τα προβλήματα που υπάρχουν σε κάθε περίπτωση αν κάτι πάει λάθος.

5.2. Το ιδανικό μοντέλο

Το ιδανικό μοντέλο είναι αυτό που εμείς θεωρούμε ότι βρίσκεται πιο κοντά στον τρόπο που ο άνθρωπος αναλύει και επεξεργάζεται ότι βλέπει. Τώρα το

μόνο που μένει είναι να αρχίσουμε την δοκιμή και να δούμε πόσο λειτουργικό είναι στην πραγματικότητα.

5.2.1. Εκπαίδευση στο ιδανικό μοντέλο

Την εκπαίδευση στο ιδανικό μοντέλο την κάνουμε σε ένα δίκτυο με διαστάσεις 100*100 με εκπαίδευση στις 20 εποχές, αρχικό μέγεθος γειτονιάς 40 νευρώνες, αρχικό βήμα εκπαίδευσης 1 και τελικό 0.1.

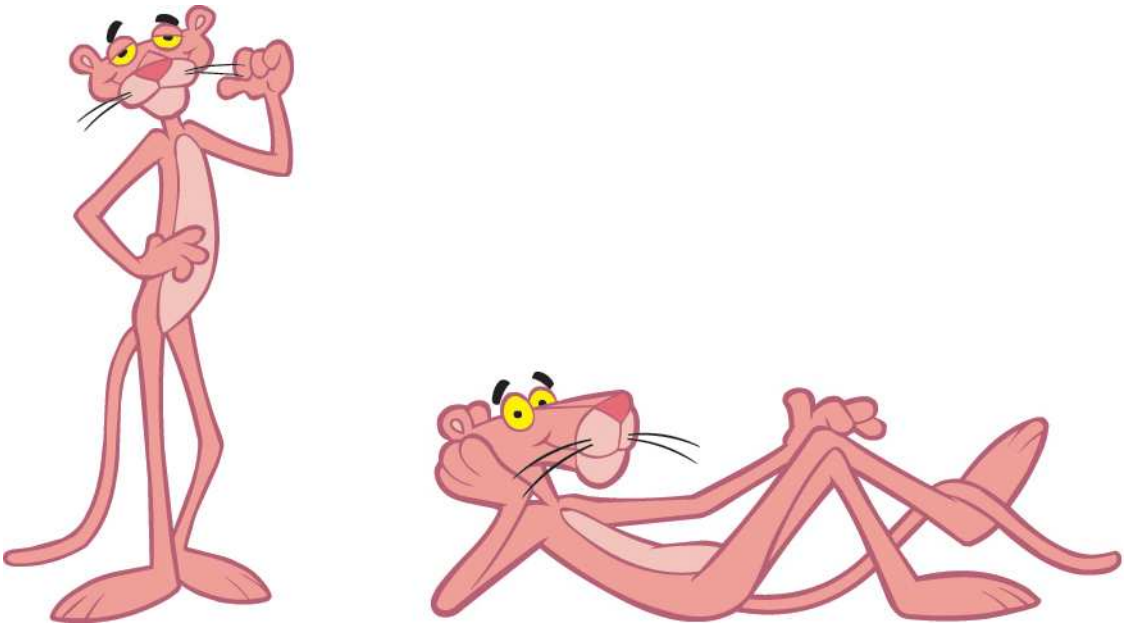
Πριν καν από το τέλος της εκπαίδευσης παρατηρούμε το πρώτο πρόβλημα. Η ταχύτητα εκπαίδευσης είναι παρά πολύ αργή με αποτέλεσμα να χρειάζεται πάνω από 10 ώρες για να ολοκληρωθεί. Αυτό οφείλεται κυρίως στην εκπαίδευση του δικτύου με το σχήμα αφού για κάθε υπολογισμό απόστασης ή εκπαίδευσης έχουμε εκτελέσεις στον πίνακα του σχήματος που έχει μέγεθος 40*40. Αυτό το γεγονός στην ουσία αυξάνει τον χρόνο εκτέλεσης κατά 3200 φορές.

5.2.2. Ανάκληση στο ιδανικό μοντέλο

Ακόμα και η ανάκληση είναι αργή αλλά αν αποφασίσουμε για λίγο να αφήσουμε αυτό το πρόβλημα και κοιτάξουμε τα αποτελέσματα της ανάκλησης βλέπουμε ότι έχουμε 22 λάθη στις 49 εικόνες. Έχουμε ένα αρκετά μικρό ποσοστό επιτυχίας πράγμα που δείχνει ότι εκτός από την ταχύτητα εκτέλεσης συμβαίνει και κάτι άλλο.

Το πρόβλημα μπορούμε να το εντοπίσουμε στο ότι μια εικόνα μπορεί να αλλάξει δραστικά και έτσι το σχήμα να βγει αρκετά εκτός του κανονικού πάρα πολύ εύκολα. Δηλαδή ας λάβουμε την εικόνα 5.2 παρακάτω ως παράδειγμα παρατηρούμε ότι στα αριστερά ο Ροζ Πάνθηρας είναι όρθιος ενώ στα δεξιά ξαπλώνει. Ενώ εμείς ως άνθρωποι μπορούμε να αντιληφτούμε χωρίς καμία απολύτως δυσκολία ότι πρόκειται για τον ίδιο χαρακτήρα, ο υπολογιστής προφανώς μπερδεύεται από κάτι και αυτό είναι η στάση του σώματος. Ενώ έχουμε εκπαιδεύσει το δίκτυο με διάφορες στάσεις για κάθε χαρακτήρα που δίνουν ο κάθε ένας τα δικά του ξεχωριστά σχήματα μετά από την κατάτμηση,

μπορούν πάρα πολύ εύκολα να προκύψουν στάσεις που θα δώσουν αποτελέσματα που θα μπερδέψουν τον αλγόριθμο μας.



Εικόνα 5.2: Ο Ροζ Πάνθηρας σε δύο στάσεις σώματος.

Η λύση θα μπορούσε να είναι τέτοια ώστε κάθε σχήμα της κατάτμησης να εξετάζεται σε κάθε μια από τις πιθανές περιστροφές του που ώμος θα μας αύξανε τον χρόνο εκτέλεσης κατά 180 φορές, όσες και οι όψεις περιστροφής.

5.2.3. Συμπεράσματα για το ιδανικό μοντέλο

Μετά τη δοκιμή που κάναμε στο μοντέλο μας βλέπουμε ότι είναι κάθε άλλο από ιδανικό και πόσο μάλλον ρεαλιστικό αφού ο χρόνος εκτέλεσης είναι εξαιρετικά μεγάλος. Είδαμε ακόμα το πρόβλημα που παρουσιάζεται με τα σχήματα που μπορεί να αποσυντονίσουν πολύ εύκολα τον αυτοοργανούμενο χάρτη. Θα μπορούσαμε να διορθώσουμε αυτό το πρόβλημα εξετάζοντας κάθε όψη του σχήματος αλλά αυτό θα μας μειώσει ακόμα περισσότερο την ταχύτητα και πρέπει να το λάβουμε σοβαρά υπόψη. Το θετικό μήνυμα είναι ότι τα λάθη μας είναι 22 πράγμα που σημαίνει ότι οι χαρακτήρες δεν είναι τυχαία επιλεγμένοι. Αυτό συμπεραίνεται από το γεγονός ότι για κάθε χαρακτήρα υπάρχουν τέσσερις

επιλογές άρα και 25% πιθανότητες επιτυχίας. Άρα στην περίπτωση μας ένα δείγμα που θα έμοιαζε ότι έχει τυχαία επιλογή θα έδινε περίπου δώδεκα σωστά αποτελέσματα ενώ εμείς έχουμε 27. Κρατώντας αυτό θα προσπαθήσουμε να πετύχουμε κάποιες βελτιώσεις.

5.3. Το μοντέλο συσχετίσεων

Το ιδανικό μοντέλο αποδείχτηκε προβληματικό στην επεξεργασία του σχήματος έτσι αποφασίσαμε να αφαιρέσουμε εντελώς. Είναι στην ουσία το ίδιο με το προηγούμενο αλλά με την διαφορά ότι το σχήμα δεν έχει κάποιο ρόλο σον υπολογισμό της απόστασης και την εκπαίδευση. Η διαφορά που δημιουργείτε, φαίνεται στις πιο κάτω μεθόδους που πια περιέχουν μόνο υπολογισμούς για το χρώμα της περιοχής.

```
public double distance(double[] colorDataSet){
    return colorDistance(colorDataSet);
}

public double colorDistance(double colorDataSet[]){
    double total=0;
    double dist;
    for(int i=0; i<color.length; i++){
        total+=Math.pow(color[i]-colorDataSet[i],2);
    }
    dist=Math.sqrt(total);
    return dist;
}
```

Κώδικας 5.1: Μέθοδοι για τον υπολογισμό απόστασης μοντέλου συσχετίσεων

```
public void train(double colorDataSet[],double beta){
    trainColor(colorDataSet,beta);
}

public void trainColor(double colorDataSet[],double beta){
    double delta=0;
```

```
        for(int i=0; i<color.length; i++){
            delta = beta * (colorDataSet[i] - color[i]);
            color[i]+=delta;
        }
    }
```

Κώδικας 5.2: Μέθοδος εκπαίδευσης νευρώνα μοντέλου συσχετίσεων

Το μοντέλο αυτό το ονομάσαμε μοντέλο συσχετίσεων καθώς αυτό που κρατάει τη δομή της εικόνας είναι οι συσχετίσεις της μνήμης των αντικειμένων. Πιο κάτω φαίνεται η δοκιμή που κάναμε στο μοντέλο συσχετίσεων.

5.3.1. Εκπαίδευση στο μοντέλο συσχετίσεων

Ο χρόνος εκπαίδευσης σε αυτό το μοντέλο έχει μειωθεί περισσότερο από το μισό σε σχέση με το προηγούμενο αλλά μπορούμε να πούμε ότι ακόμα παραμένει αρκετά ψηλά. Γι' αυτό τον λόγο την εκπαίδευση στο μοντέλο συσχετίσεων την κάνουμε και πάλι σε ένα δίκτυο με διαστάσεις 100*100 με εκπαίδευση μόνο 20 εποχές, αρχικό μέγεθος γειτονιάς 40 νευρώνες, αρχικό βήμα εκπαίδευσης 1 και τελικό 0.1.

Το κόστος σε χρόνο αυτή την φορά το κρατάει ο μεγάλος αριθμός συσχετίσεων καθώς ο αυτοοργανούμενος χάρτης τελειώνει την εκπαίδευση του σε μόλις πέντε λεπτά.

5.3.2. Ανάκληση στο μοντέλο συσχετίσεων

Τα αποτελέσματα μας είναι και πάλι τα ίδια, δηλαδή 22 λάθη στις 49 εικόνες πράγμα που δείχνει ότι το πρόβλημα του χαμηλού ποσοστού επιτυχίας δεν οφειλόταν μόνο στο σχήμα. Αντίθετα ο μεγάλος συσχετίσεων που έχουμε ίσως να δημιουργούν μεγαλύτερο πρόβλημα από αυτό του σχήματος. Ο αυτοοργανούμενος χάρτης που μεταξύ των άλλων έχει και την ιδιότητα να συμπιέζει τα δεδομένα του όμως εδώ ίσως το μέγεθος του να μην είναι αρκετά μεγάλο για να το κάνει αποδοτικά.

5.3.3. Συμπεράσματα για το μοντέλο συσχετίσεων

Παρατηρήσαμε ότι ακόμα και με τις αλλαγές δεν είχαμε κάποια βελτίωση στα αποτελέσματα. Από την άλλη ο χρόνος εκτέλεσης παραμένει ακόμα πολύ μεγάλος. Με αυτά στο μυαλό θα σχεδιάσουμε τις επόμενες αλλαγές που θα αντιμετωπίσουν όλα τα προβλήματα ταχύτητας. Βλέπουμε ακόμα ότι οι συσχετίσεις δεν λειτουργούν όπως θα τις θέλαμε και θα πρέπει με κάποιο τρόπο να τις βελτιώσουμε.

5.4. Το τελικό μοντέλο

Για να σχηματίσουμε το τελικό μοντέλο αποφασίσαμε να κάνουμε ριζικές αλλαγές που θα βελτιώσουν κατά κύριο λόγο την ταχύτητα του μοντέλου και θα βελτιώσουν τα αποτελέσματα.

Η πρώτη αλλαγή που κάναμε είναι να προσθέσουμε στον αυτοοργανούμενο χάρτη το μέγεθος της περιοχής και την αφαιρέσουμε από τις συσχετίσεις. Αυτό το κάναμε σε συνδυασμό με τις αλλαγές στην μνήμη που θα δούμε πιο κάτω για να βελτιώσουμε την ταχύτητα αλλά κάνουμε την παραδοχή ότι τα αντικείμενα μας θα καταλαμβάνουν πάντα το ίδιο μέγεθος pixels μέσα στην εικόνα. Αυτές οι αλλαγές επηρεάζουν τον υπολογισμό απόστασης και εκπαίδευση στον αυτοοργανούμενο χάρτη που γίνονται όπως πιο κάτω.

```
public double distance(double colorDataSet[],double sz){
    double total=0;
    double dist;
    total+=Math.pow(size-sz,2);
    for(int i=0; i<color.length; i++){
       total+=Math.pow(color[i]-colorDataSet[i],2);
    }
    dist=Math.sqrt(total);

    return dist;
  }
```

Κώδικας 5.3: Μέθοδος υπολογισμού απόστασης νευρώνα τελικού μοντέλου

```
public void train(double colorDataSet[],double sz,double beta){
   double delta;
   delta = beta * (size - sz);
   size+=delta;

   for(int i=0; i<color.length; i++){
      delta = beta * (colorDataSet[i] - color[i]);
      color[i]+=delta;
   }
}
```

Κώδικας 5.4: Μέθοδος εκπαίδευσης νευρώνα τελικού μοντέλου

Εκτός από αυτές τις αλλαγές στους νευρώνες έχουμε κάνει μια σειρά από αλλαγές στην μνήμη και τις συσχετίσεις. Για να αποφύγουμε τον μεγάλο αριθμό συσχετίσεων και περιττών δεδομένων αποφασίσαμε να αφαιρέσουμε τον δεύτερο νευρώνα από τις συσχετίσεις. Έτσι τώρα μια συσχέτιση περιέχει την θέση που κατέχει ο νευρώνας στον αυτοοργανούμενο χάρτη φτιάχνοντας έτσι ένα προφίλ στην μνήμη με τις περιοχές που συμμετέχουν στο αντικείμενο. Οι αλλαγές αυτές διαγράφονται στις μεθόδους εκπαίδευσης και ανάκλησης στις κλάσεις Relation και Memory.

```
public double relationDistance(Relation r,int netSize){
   double neuDist;

   neuDist=neuronDistance(r,netSize);
   return neuDist;
}

public void trainRelation(Relation r,double beta,int netSize){
   neuronW+=neuronBoundsDelta(neuronW,r.getNeuronW(),netSize,beta);
   neuronH+=neuronBoundsDelta(neuronH,r.getNeuronH(),netSize,beta);
}
```

Κώδικας 5.5: Απόσταση και εκπαίδευση συσχέτισης του τελικού μοντέλου

```
public void trainRelations(SomImage image,double beta, int neigh,int netSize){
    double minDist;
    int closestRel=0;
    int rDown,rUp;
    RegionDataSet region;
    Relation relation;

    for(int i=0; i<image.getRegionsNo(); i++){
        region=image.getRegion(i);
        relation=new Relation(region);
        minDist=Double.MAX_VALUE;
        closestRel=-1;
        //find winner relation
        for(int r=0; r<getRelationsNo(); r++){
            if(relation.relationDistance(relations.getRelation(r),netSize)<minDist){
                closestRel=r;
                minDist=relation.relationDistance(relations.getRelation(r),netSize);
            }
        }
        if(description.equals(image.getDescription()))
            relations.getRelation(closestRel).trainRelation(relation,beta,netSize);
        relations.getRelation(closestRel)
                .setStrenght(minDist,description.equals(image.getDescription()));
        for(int r=1; r<=neigh; r++){
            rDown=closestRel-r;
            rUp=closestRel+r;
            if(rDown<0)rDown=getRelationsNo()+rDown;
            if(rUp>=getRelationsNo())rUp=rUp-getRelationsNo();

            if(description.equals(image.getDescription())){
                relations.getRelation(rDown).trainRelation(relation,beta,netSize);
                relations.getRelation(rUp).trainRelation(relation,beta,netSize);
            }
```

```
            relations.getRelation(rDown)
                .setStrenght(minDist,description.equals(image.getDescription()));
            relations.getRelation(rUp)
                .setStrenght(minDist,description.equals(image.getDescription()));
        }
    }
}
```

Κώδικας 5.6: Εκπαίδευση στις συσχετίσεις του τελικού μοντέλου

```
public void trainNeuron(SomImage image,double beta,int netSize){
    int relationStrenght=0;
    double minDist;
    int relationPos;
    RegionDataSet region;
    Relation relation;

    for(int j=0; j<image.getRegionsNo(); j++){
        region=image.getRegion(j);
        relation=new Relation(region);

        relationPos=-1;
        minDist=Double.MAX_VALUE;
        for(int i=0; i<getRelationsNo(); i++)
            if(relations.getRelation(i).relationDistance(relation,netSize)<minDist ){
                relationPos=i;
                minDist=relations.getRelation(i).relationDistance(relation,netSize);
            }
        relationStrenght+=relations.getRelation(relationPos).getStrenght();
    }
    memoryNeuron.train(relationStrenght,beta);
}
```

Κώδικας 5.7: Εκπαίδευση νευρώνων μνήμης του τελικού μοντέλου

```
public double recallImage(SomImage image,int netSize){
    int relationStrenght=0;
    double minDist;
    int relationPos;
    RegionDataSet region;
    Relation relation;

    for(int j=0; j<image.getRegionsNo(); j++){
        region=image.getRegion(j);
        relation=new Relation(region);
        relationPos=-1;
        minDist=Double.MAX_VALUE;
        for(int i=0; i<getRelationsNo(); i++)
            if(relations.getRelation(i).relationDistance(relation,netSize)<minDist ){
                relationPos=i;
                minDist=relations.getRelation(i).relationDistance(relation,netSize);
            }
        relationStrenght+=relations.getRelation(relationPos).getStrenght();
    }
    return memoryNeuron.getPercentDistanse(relationStrenght);
}
```

Κώδικας 5.8: Ανάκληση εικόνας στο τελικό μοντέλο

5.4.1. Εκπαίδευση στο τελικό μοντέλο

Με τις αλλαγές που έχουμε κάνει ο χρόνος εκπαίδευσης σε αυτό το μοντέλο έχει πραγματικά μειωθεί πάρα πολύ πέφτοντας στα 10 μόλις λεπτά τα οποία θεωρούμε πάρα πολύ ικανοποιητικά. Την εκπαίδευση και σε αυτό το μοντέλο θα την κάνουμε για σκοπούς σύγκρισης και πάλι σε ένα δίκτυο με διαστάσεις 100*100 με εκπαίδευση μόνο 20 εποχές, αρχικό μέγεθος γειτονιάς 40 νευρώνες, αρχικό βήμα εκπαίδευσης 1 και τελικό 0.1. Τώρα όμως που ο χρόνος εκπαίδευσης έχει μειωθεί τόσο σίγουρα έχουμε την δυνατότητα καλύτερης εκπαίδευσης του δικτύου.

5.4.2. Ανάκληση στο μοντέλο συσχετίσεων

Εκτός από τον χρόνο εκπαίδευσης έχει μειωθεί και ο χρόνος ανάκλησης που παίρνει πιο μόνο μερικά δευτερόλεπτα. Τα αποτελέσματα μας έχουν βελτιωθεί επίσης δίνοντας μας 14 λάθη στις 49 εικόνες. Ο αυτοοργανούμενος χάρτης τώρα φαίνεται να λειτουργεί σωστά και καταφέρνει να οργανώνει σωστά τα δεδομένα του.

5.4.3. Συμπεράσματα για το μοντέλο συσχετίσεων

Οι αλλαγές που κάναμε μας έφεραν την λειτουργικότητα και αποδοτικότητα που περιμέναμε. Ως αλλαγή μπορεί προς το παρών να βελτιώσει την λειτουργικότητα μας αλλά πρακτικά σε εικόνες αγνώστου μεγέθους δεν μπορεί να λειτουργήσει σωστά. Κάνοντας δηλαδή αυτές τις αλλαγές θυσιάσαμε την ανεξαρτησία των δεδομένων εισόδου τα οποία τώρα πρέπει να έχουν κάποια συγκεκριμένη μορφή.

5.5. Συμπεράσματα

Αρχικά βάλαμε στο μικροσκόπιο μας το μοντέλο που θεωρούσαμε ιδανικό και δοκιμάσαμε την αποδοτικότητα του που δεν ήταν αυτό που περιμέναμε. Αργότερα κάναμε αλλαγές που μας βοήθησαν όμως σε κάποιο βαθμό μόνο στον χρόνο εκτέλεσης. Τελικά φτάσαμε να κάνουμε αλλαγές οι οποίες μας έφεραν στο τελικό μοντέλο το οποίο μας δίνει λειτουργικότητα και απόδοση πιο κοντά σε αυτό που ζητάμε. Για αυτό όμως χρειάστηκε να κάνουμε θυσίες που μας απομακρύνουν περισσότερο από την διαδικασία της ανθρώπινης όρασης και μας θέτουν σε σοβαρούς περιορισμούς. Σε μελλοντικές αλλαγές θα πρέπει να εξετάσουμε τομείς, ίσως αυτούς που αναφέρονται στο επόμενο κεφάλαιο ώστε να μας επιτρέψουν να αλλάξουμε μερικά πράγματα ριζικά και να έρθουμε πάλι πιο κοντά στο ιδανικό μοντέλο.

Κεφάλαιο 6

Σκέψεις για μελλοντική έρευνα και εξέλιξη

6.1. Εισαγωγή

Σε αυτό το κεφάλαιο θα εξετάσουμε μερικά χαρακτηριστικά που θα ήταν καλό να προσαρτηθούν στο μοντέλο μας. Θα δούμε τρόπους που θα μπορούσαν να βελτιώσουν την κατάτμηση η οποία αποτελεί τον θεμέλιο λίθο από τον οποίο παίρνουμε τα δεδομένα μας. Στην συνέχεια θα εξετάσουμε έναν εναλλακτικό τρόπο προσέγγισης στα σχήματα που πιστεύουμε ότι θα μας βοηθήσει σημαντικά σε ένα σημαντικό κομμάτι του μοντέλου μας που μοιάζει απαραίτητο αν θέλουμε να έχουμε καλύτερα αποτελέσματα. Θα προσπαθήσουμε να εξετάσουμε καλύτερα το πρόβλημα που μας οδήγησε στο να αλλάξουμε τις συσχετίσεις και βρούμε κάποια πιθανή λύση για μελλοντικές αλλαγές. Τέλος θα δούμε ένα τομέα ο οποίος από την στιγμή που θα έμπαινε στο μοντέλο μας θα μπορούσαν να δώσει το κάτι παραπάνω. Ο τομέας αυτός εξετάζει το ενδεχόμενο το μοντέλο μας να έχει την ικανότητα να μαθαίνει όχι μόνο κατά την εκπαίδευση αλλά και οποιαδήποτε άλλη στιγμή παίρνοντας την κατάλληλη τροφοδότηση από τον χρήστη και όχι μόνο.

6.2. Βελτίωση της κατάτμησης

Αν και η κατάτμηση που έχουμε αναπτύξει επιτυγχάνει αρκετά καλά αποτελέσματα αφήνει μερικά σημεία που δεν εξετάζει. Ας δούμε για παράδειγμα τη πιο κάτω εικόνα.

Εικόνα 6.1: Παράδειγμα με εικόνα του Willey Coyote.

Στην εικόνα 6.1 βλέπουμε τον Willey Coyote να κάθεται πάνω σε ένα σκαμπό και το πρόβλημα το εντοπίζουμε ακριβώς εκεί. Με τον δικό μας αλγόριθμο το σκαμπό μετά την κατάτμηση θα φαινόταν ως δύο κομμάτια λόγο της επικάλυψης του από το πόδι του Willey. Υπάρχουν μέθοδοι που λαμβάνουν αυτά τα γεγονότα υπόψη και μπορούν να φτάνουν σε καλύτερα συμπεράσματα. Ένας τέτοιος για παράδειγμα είναι ο αλγόριθμος κατάτμησης με γράφους. Σε αυτό τον αλγόριθμο κάθε pixel της εικόνας θεωρείτε ως ένας κόμβος. Όταν έχουμε δύο κόμβους συνδεδεμένους τους αποκαλούμε μια άκρη. Αυτό που κάνει ο αλγόριθμος είναι να ψάχνει τους κόμβους που βρίσκονται στις άκρες των αντικειμένων και να τους συνδέει για να φτιάξει κάποιο περίγραμμα[22]. Με αυτόν τον τρόπο μία σωστή υλοποίηση θα μπορούσε να λάβει υπόψη το πρόβλημα της επικάλυψης καθώς με την εισαγωγή κάποιας ορμής που ακολουθεί μια άκρη θα μπορούσε να δει ότι το αντικείμενο συνεχίζεται μετά την επικάλυψη ακόμα και αν δεν είναι άμεσα συνδεδεμένο.

Υπάρχουν πολλά ακόμα που θα μπορούσαν να λεχθούν για αυτόν τον τομέα όπως για παράδειγμα το πρόβλημα που δημιουργείτε όταν έχουμε έντονες σκιές. Υπάρχουν πολλές ιδέες και εξελίξεις στους αλγόριθμους κατάτμησης που θα μπορούσαν να βελτιώσουν αισθητά τα δεδομένα που εξάγουμε από την εικόνα. Είναι σίγουρα κάτι που πρέπει να ληφθεί πιο σοβαρά υπόψη και να μελετηθεί κατάλληλα ώστε να βελτιώσει το μοντέλο μας.

6.3. Γραφικά Vector

Τα γραφικά vector τυπικά χρησιμοποιούνται στον σχεδιασμό λογότυπων, σε αφαιρετικά σχέδια ή όπου αλλού απαιτητέ ακρίβεια. Εμείς μέσα από αυτά ευελπιστούμε ότι θα λύσουμε το πρόβλημα του σχήματος και από την άποψη της ταχύτητας αλλά και της προσαρμογής σε μεγάλες διαφορές όπως η περιστροφές.

Τα γραφικά vector δημιουργούνται μαθηματικά χρησιμοποιώντας στοιχεία όπως σημεία, γραμμές, καμπύλες για να δημιουργήσουν διάφορά σχήματα. Το βασικό τους πλεονέκτημα είναι ότι μπορούν να μεγαλώσουν όσο χρειαστεί χωρίς να χαλάσει καθόλου η ποιότητα τους κάτι που δεν συμβαίνει όταν η εικόνα μας

είναι φτιαγμένη από pixels[23]. Το στάνταρ πρότυπο που καθορίζει τα vector από την 3WC είναι το SVG.

Εικόνα 6.2: Παράδειγμα μεγέθυνσης vectors και pixels.

Εμείς θα θέλαμε να δοκιμάσουμε να προσεγγίζουμε το περίγραμμα του σχήματος των περιοχών που δημιουργεί η κατάτμηση χρησιμοποιώντας τα vectors. Έχουμε την πεποίθηση ότι λόγω της μαθηματικής τους φύσης θα είναι πιο εύκολο να επεξεργαστούν καταναλώνοντας έτσι λιγότερο χρόνο επεξεργασίας. Τα vector είναι σίγουρα κάτι που αξίζει μελέτης γιατί το σχήμα έχει ένα πολύ βασικό ρόλο στην ανθρώπινη όραση και πρέπει να βρούμε τρόπους ώστε να το προσεγγίσουμε σωστά και αποδοτικά.

6.4. Βελτίωση των συσχετίσεων

Προσεγγίζοντας τις συσχετίσεις μέσω του μονοδιάστατου αυτοοργανούμενου χάρτη στην μνήμη πετύχαμε να συμπιέσουμε αρκετά καλά τις συσχετίσεις. Το πρόβλημα μας ήταν στο ότι υπήρχαν πάρα πολλές συσχετίσεις που επαναλαμβάνουν το ίδιο πράγμα δημιουργώντας στην ουσία εκτός από τα περιττά δεδομένα ένα μεγάλο όγκο επεξεργασίας στα πρώτα δύο μοντέλα. Για κάθε εικόνα είχαμε ένα αριθμό συσχετίσεων ίσο περίπου με τον μισό αριθμό περιοχών στο τετράγωνο. Στην εικόνα 6.3 κάτω βλέπουμε το παράδειγμα

κατάτμησης με τον Duffy Duck για να εξετάσουμε τις περιοχές που δημιουργούνται και πώς θα θέλαμε τις εμείς τις συσχετίσεις να προκύψουν.

Εικόνα 6.3: Παράδειγμα συσχετίσεων σε τρείς περιοχές.

Στο μοντέλο μας όπως το υλοποιήσαμε για το ιδανικό και το μοντέλο συσχετίσεων θα είχαμε τις παρακάτω συσχετίσεις για τις τρείς περιοχές:

1. (1,2)
2. (1,3)
3. (2,3)

Επειδή οι συσχετίσεις γίνονται πάντα από την μεγαλύτερη συσχέτιση προς την μικρότερη αυτές που προκύπτουν είναι οι τρείς πιο πάνω. Ο αριθμός των συσχετίσεων θα αυξανόταν κατά πολύ αν είχαμε περισσότερες περιοχές.

Αυτό που θα θέλαμε να δημιουργήσουμε εμείς είναι ένας τρόπος ώστε οι συσχετίσεις να δημιουργούνται γύρω από μία μόνο περιοχή όπως παρακάτω:

1. (1,2)
2. (1,3)

Εδώ όλες οι συσχετίσεις δημιουργούνται γύρω από την πρώτη περιοχή. Η σκέψη μας είναι να βρεθεί μια περιοχή που κρίνεται ως η πιο σημαντική στο αντικείμενο και υπόλοιπες να θεωρούνται ως συσχετίσεις γύρω της. Θα πρέπει όμως αυτό να το δούμε προσεκτικά γιατί μπορεί να κρύβει παγίδες όπως την επιλογή

κάποιας περιοχής ως σημαντικής που όμως δεν έχει να κάνει με το αντικείμενο. Όπως θα ήταν για παράδειγμα μια μικρή περιοχή που ο αλγόριθμος κατάτμησης δεν κατάφερε να συγχώνευση σωστά. Οι συσχετίσεις πρέπει οπωσδήποτε να βελτιωθούν για να βελτιώσουν και την πρακτικότητα του μοντέλου μας.

6.5. Σχετική Ανατροφοδότηση

Η σχετική ανατροφοδότηση(Relevance Feedback) είναι μία μέθοδος που χρησιμοποιείτε στα συστήματα ανάκτησης πληροφοριών. Η βασική της ιδέα είναι να παίρνει τα δεδομένα που προέρχονται από τις αναζητήσεις των χρηστών και να τα αξιολογεί για μελλοντικές αναζητήσεις. Από την σχετική ανατροφοδότηση επιλέξαμε δύο μορφές που μας ενδιαφέρουν, την άμεση και την έμμεση[25].

Στην άμεση ανατροφοδότηση ο χρήστης που θα χρησιμοποιούσε το σύστημα μας θα έχει την δυνατότητα να αξιολογεί κατά πόσο τα αποτελέσματα του επέστρεψε η αναζήτηση του[24][25]. Δηλαδή αν οι εικόνες που βλέπει έχουν τα σωστά αποτελέσματα.

Στην έμμεση ανατροφοδότηση ο χρήστης θα κάνει κάποια αναζήτηση και θα του παρουσιάζονται τα αποτελέσματα που το μοντέλο μας θεωρεί σχετικά. Χωρίς να ερωτηθεί ο χρήστης, το σύστημα θα κρατά τις επιλογές του και θα τις αξιολογεί[25]. Δηλαδή θα θεωρεί ότι σαν σχετικές εικόνες θεωρήθηκαν αυτές που επέλεξε ο χρήστης.

Τα αποτελέσματα που θα παρθούν από τον χρήστη και στις δύο περιπτώσεις θα μπορούσαν να χρησιμοποιηθούν είτε σε μελλοντικές εκπαιδεύσεις είτε σε μία εκπαίδευση του συστήματος μας σε ζωντανό χρόνο. Η σχετική ανατροφοδότηση είναι μια σημαντική μέθοδος για το μοντέλο μας αφού θα μπορούσε να προσομοιώνει μια συνεχή διαδικασία μάθησης με δεδομένα που προέρχονται από την πιο αξιόπιστη πηγή που είναι ο άνθρωπος.

6.6. Συμπεράσματα

Μέσα από όλη αυτή την έρευνα μάθαμε αρκετά για την ανθρώπινη διαδικασία της όρασης και την υπεροχή της έναντι σε κάθε προσέγγιση που επιχειρείται από τους υπολογιστές. Μάθαμε ότι είναι μια διαδικασία που μπορεί

να προσεγγιστεί αλλά για να γίνει σωστά πολλές φορές χρειάζεται περισσότερη επεξεργαστική από αυτή που διαθέτουμε. Δημιουργήσαμε ένα λειτουργικό μοντέλο που όμως υπόκειται σε αρκετούς περιορισμούς. Πιστεύουμε όμως ότι δώσαμε μια καθοδήγηση μέσα από ένα μοντέλο που πρέπει να προσεγγιστεί. Δώσαμε επίσης σε αυτό το κεφάλαιο μερικούς τομείς έρευνας που θα έφερναν βελτιώσεις ίσως και πιο καλές από αυτές που θα αναμέναμε. Ο δρόμος της ανάκτησης δεν είναι στρωμένος με ροδοπέταλα αλλά οι ανάγκες που δημιουργούνται όλο και περισσότερο τον τελευταίο καιρό δείχνουν ότι πρέπει να υπάρξει κάποια σημαντική εξέλιξη.

Αναφορές

1. Richard Nock and Frank Nielsen - Statistical Region Merging. IEEE Transactions on pattern analysis and machine intelligence, vol. 26, no. 11, November 2004.
2. Caroline Pantofaru, Martial Hebert - A Comparison of Image Segmentation Algorithms.CMU-RI-TR-05-40, September 1, 2005.
3. Ranjith Unnikrishnan, Caroline Pantofaru, Martial Hebert - Toward Objective Evaluation of Image Segmentation Algorithms.IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 6, June 2007
4. Rafael C. Gonzalez, Richard E. Woods - Digital image Processing Second edition. Prentice Hall, 2002.
5. Besl and Jain - Segmentation Through Variable-Order Surface Fitting. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 2, pp. 167-192, 1988.
6. Robert M. Haralick, Linda G. Shapiro - Computer and Robot Vision (Volume II).Prentice Hall, 2002
7. Frank Nielsen, Richard Nock - On Region Merging: The Statistical Soundness of Fast Sorting, with Applications. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. II:19–26, 2003.
8. Ritendra Datta, Jia Li, James Z. Wang - Content-Based Image Retrieval – Approaches and Trends of the New Age. MIR’05, November 11-12, Singapore 2005.
9. Arnold W.M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, Ramesh Jain - Content-Based Image Retrieval at the End of the Early Years. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 12, December 2000.
10. Lifang Wu, Xun Liu, Lu Yui - Object recognition in complex backgrounds .ICSP'04 Proceedings, IEEE, 2004.

11. Yong Rui, Thomas S. Huang, Shish-Fu Chang - Image Retrieval: Current techniques, promising directions and open issues. Journal of Visual Communication an Image Representation, pp. 39-62, April 1999.

12. E.R. Davies - Machine Vision: Theory, Algorithms, Practicalities 2nd Edition, Academic Press, 1997.

13. Teuvo Kohonen – Self Organizing Maps, Second Edition, Springer, 1997.

14. Ροβέρτος – Ε. Κινγκ – Υπολογιστική Νοημοσύνη Στον Έλεγχο Συστημάτων, Εκδόσεις Π. Τραυλός – Ε. Κωσταράκη, 1998.

15. Delter Nauck, Frank Klawonn, Rudolf Kruse - Foundation of Neuro-Fuzzy Systems, John Wiley & Sons, 1997.

16. Joey Rogers – Object Oriented Neural Networks in C++, Academic Press, 1997.

17. Simon Haykin – Neural Networks, A Comprehensive Foundation, Second Edition, Prentice Hall, 1999.

18. Martin T. Hagan, Howard B. Demuth, Mark Beale – Neural Network Design, An International Thomson Publishing Company, 1996.

19. Judith Dayhoff – Neural Network Architectures, Van Nostrand Reinhold, 1990.

20. John A. Lucas - Encyclopedia Of the Human Brain Volume 2: Memory Overview, Academic Press, 2002.

21. Pepper Williams - Encyclopedia Of the Human Brain Volume 3: Object Perception, Academic Press, 2002.

22. Sokratis Makrogiannis, George Economou, Spiros Fotopoulos, Nikolaos G. Bourbakis - Segmentation of Color Images Using Multiscale Clustering and Graph Theoretic Region Synthesis, IEEE Transactions on Systems, MAN, and Cybernetics - Part A: Systems and Humans, vol. 35, no. 2, March 2005.

23. Ira Greenberg - Processing: Creative Coding and Computational Art, Springer, 2007.

24. Diane Kelly and Nicholas J. Belkin - Reading Time, Scrolling and Interaction: Exploring Implicit Sources of User Preferences for Relevance

Feedback During Interactive Information Retrieval, Sigir Poster Submition, 2001.

25. Ricardo Baeza-Yates, Berthier Ribeiro-Neto - Modern Information Retrieval, Addison-Wesley, 1999.