Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα
Παραδοτέο 3
Γοργογιάννης Ορέστης sdi1700024
Μπαρτσώκας Θεόδωρος sdi1700096

Ερώτημα Α:

Το νευρωνικό δίκτυο κατασκευάστηκε με παρόμοιο τρόπο με αυτό του παραδοτέου 2. Αξιοσημείωτες διαφοροποιήσεις είναι η χρήση conv2DTranspose layers έναντι απλού conv2D στον decoder. Επίσης, στο τελευταίο layer του encoder υπάρχει ένα flatten layer και ένα fully connected layer που απλοποιεί την εικόνα σε έναν πίνακα-στήλη κάποιου latent dimension, όπως αναφερόταν στην εκφώνηση.

Το πρόγραμμα, αφού εκπαιδεύσει το δίκτυο, διαβάζει τα αρχεία δεδομένων και τα κωδικοποιεί στο latent space. Κανονικοποιεί τις τιμές σε 2 byte, μεταξύ των τιμών 0 και 2^16-1 για να χωράει σε 2 Bytes με όσο το δυνατόν μικρότερη απώλεια πληροφορίας. Τέλος, αποθηκεύει τα 2 νέα αρχεία, τα οποία χρησιμοποιούνται στα επόμενα ερωτήματα.

Για την υλοποίηση του δικτύου βασιστήκαμε στο σχήμα που μας δώθηκε στο φροντιστήριο.

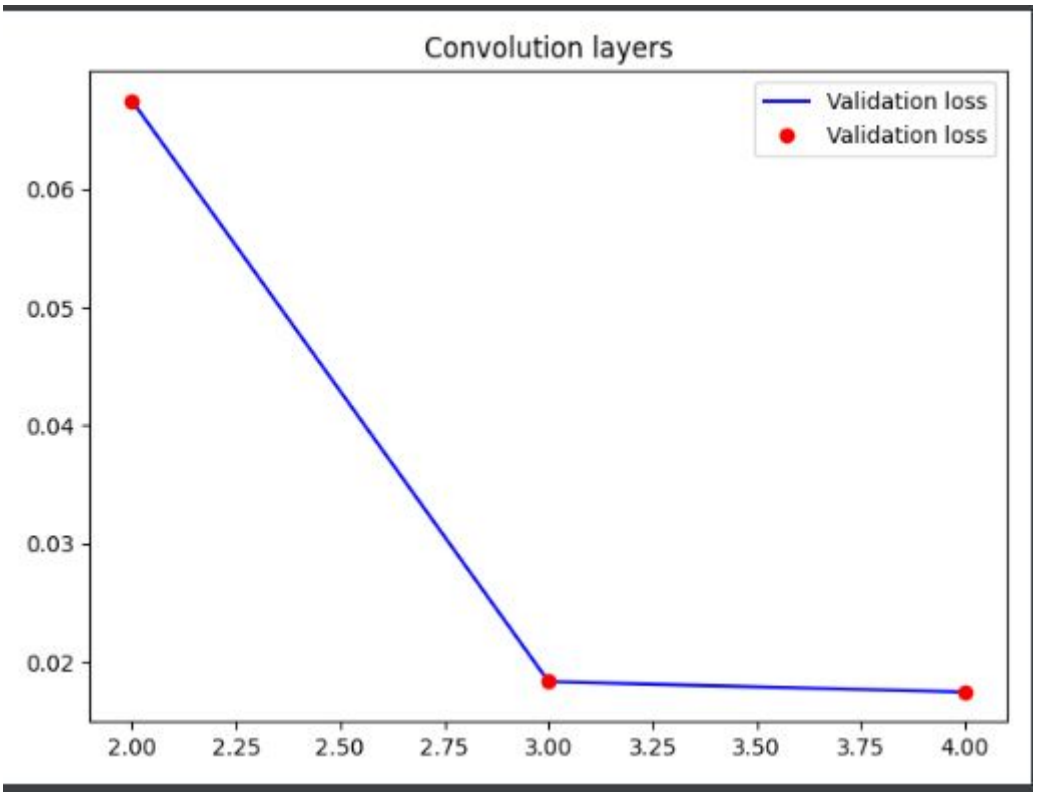
Όπως φαίνεται παρακάτω, οι ιδανικές υπερπαράμετροι για τον autoencoder κυμαίνονται στα 3 convolutional layers διότι έχουν παρόμοια συμπεριφόρα με τα 4 και άνω, όμως με μικρότερο χρόνο εκαπαίδευσης.Οι αριθμοί των συνελκτικών φίλτρων στους οποίους καταλήξαμε είναι (32,64,128). Η χρήση μεγαλύτερου αριθμού βοηθούσε πολύ λίγο και κρίθηκε μη απαραίτητη, καθώς συνέβαλλε στο overfitting.

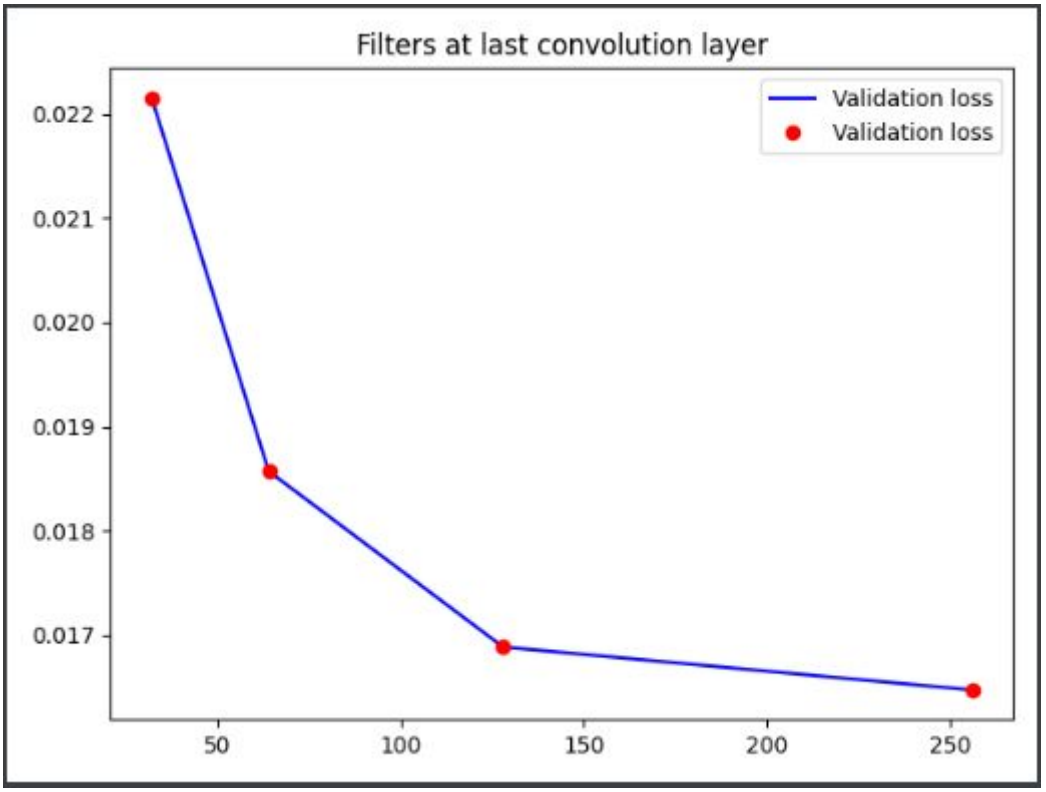
Ιδανικότερο Batch size ήταν το 32, όμως η διαφορά με το 64 ή το 128 ήταν της τάξης του 10^-3, ενώ μειωνόταν σημαντικά ο χρόνος εκπαίδευσης με μεγαλύτερο size. Το kernel dimension διαφέρει ελάχιστα μεταξύ του 2x2 και 3x3. Για το καλύτερο μοντέλο μας επιλέξαμε το 3x3. Τέλος, όπως φαίνεται από την εικόνα με το γράφημα loss vs epochs, με εκπαίδευση σε 40 εποχές, το δίκτυο μάθαινε επαρκώς, δίχως να παρατηρείται overfitting.

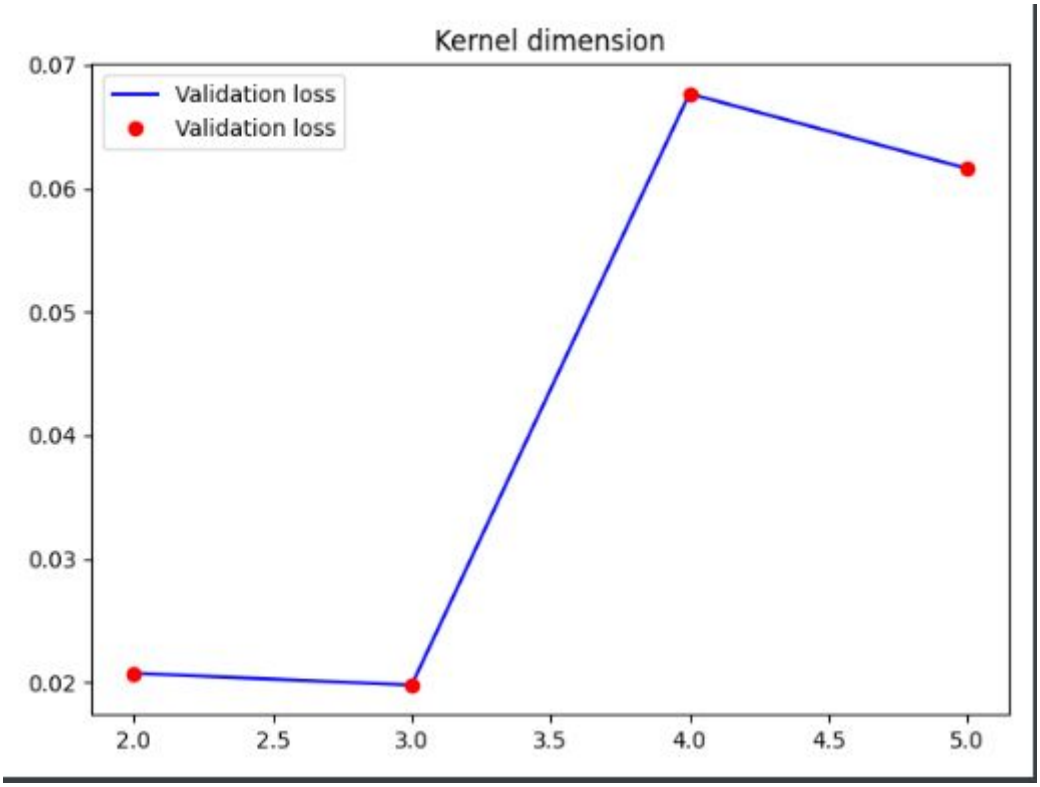
Στο καλύτερο μοντέλο το οποίο καταθέσαμε με την εργασία χρησιμοποιήσαμε:

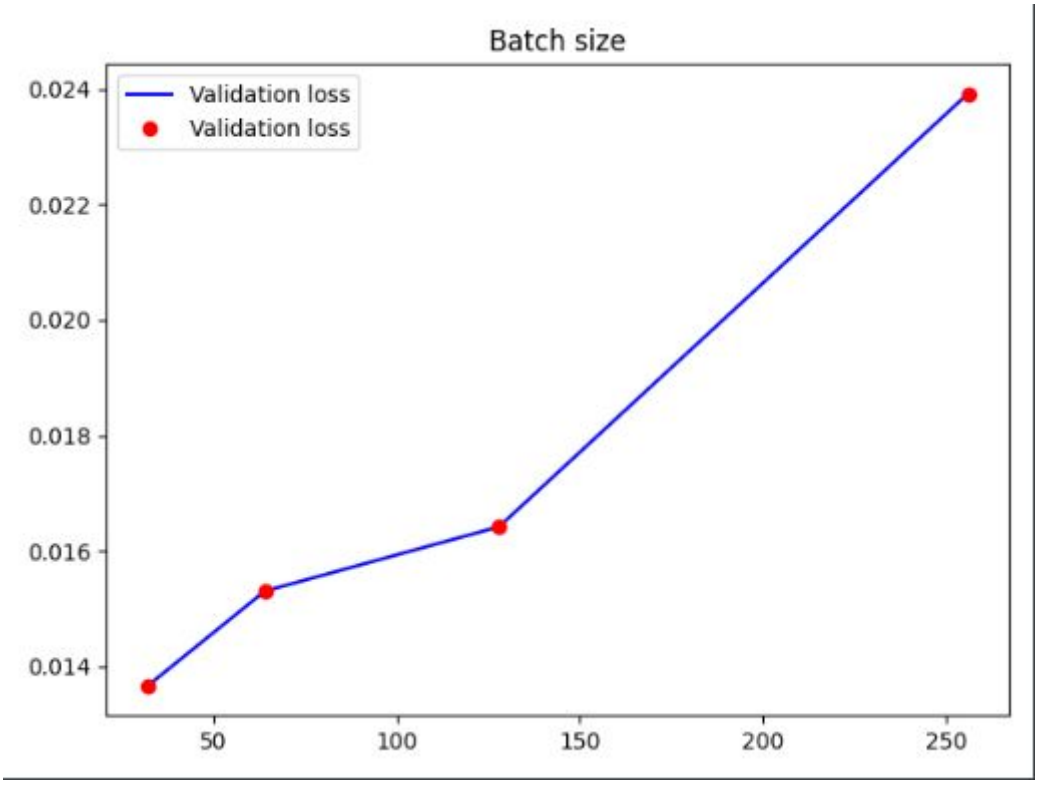
3 convolutional layers (32,64,128)
Batch size 128
40 training epochs
3x3 kernel dimensions

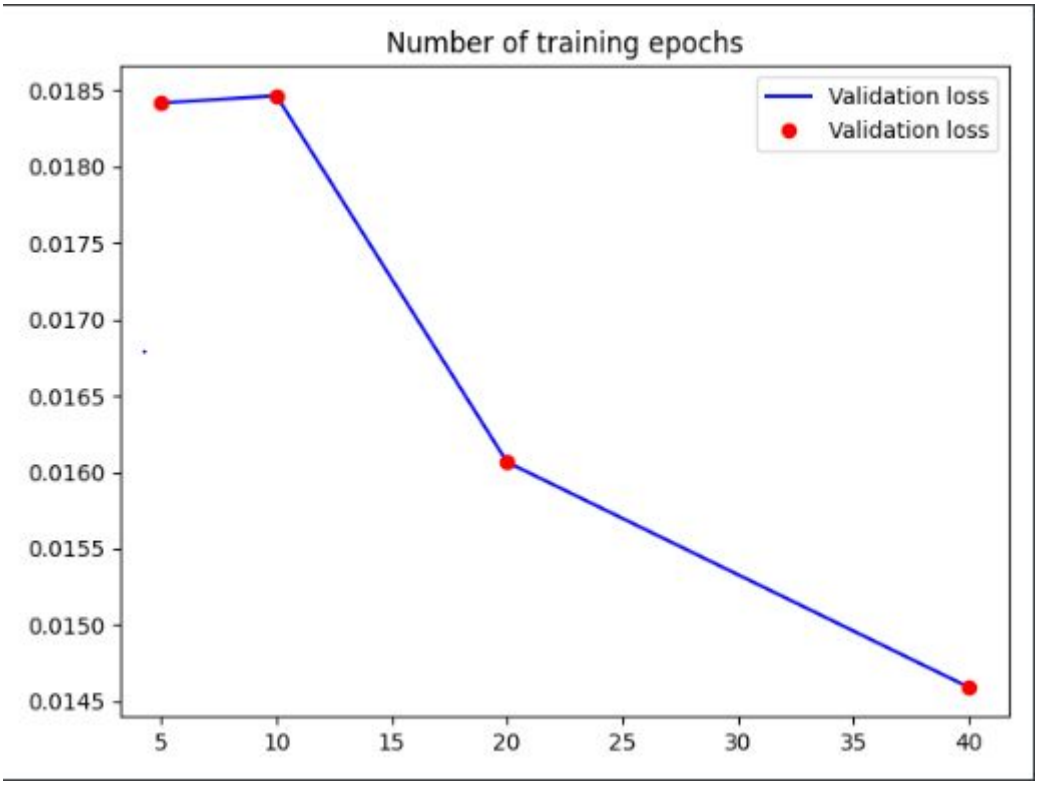
Όπως παρατηρείται από το τελευταίο plot (training and validation loss vs epochs) , στο μοντέλο αυτό δεν γίνεται Overfit.

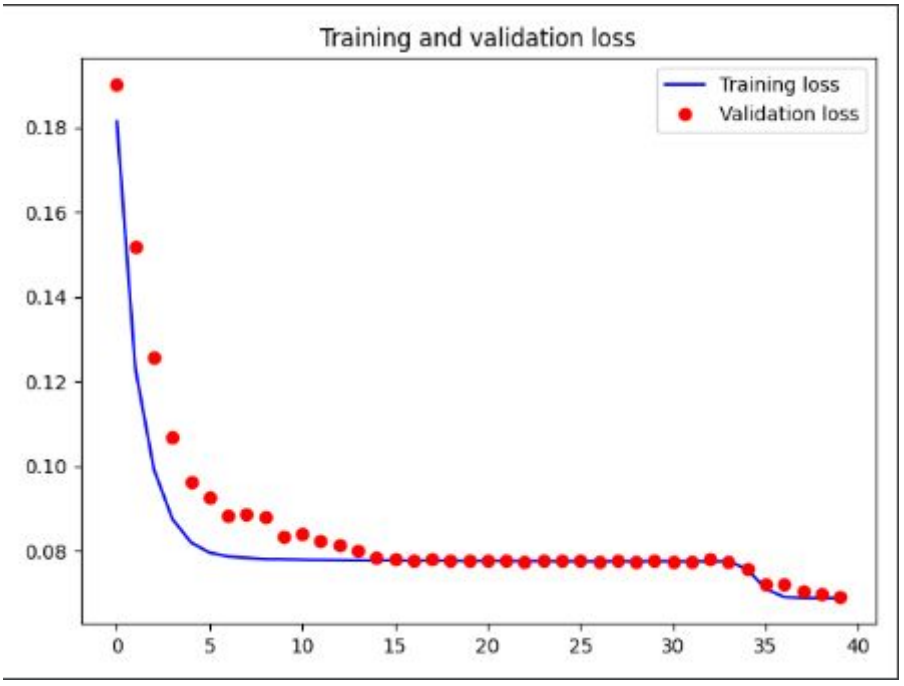

Convolution layers
Validation loss
Validation loss
0.06
0.05
0.04
0.03
0.02
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00


Filters at last convolution layer
Validation loss
Validation loss
0.022
0.021
0.020
0.019
0.018
0.017
50
100
150
200
250


Kernel dimension
Validation loss
Validation loss
0.07
0.06
0.05
0.04
0.03
0.02
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Batch size
Validation loss
Validation loss
0.024
0.022
0.020
0.018
0.016
0.014
50
100
150
200
250





Τα πειράματα για το latent dimension έγιναν συγκρίνοντας τα αποτελέσματα του ερωτήματος β. Όπως ήταν προφανές, με αύξηση του latent dimension, αυξανόταν το accuracy στο reduced size. Δεν υπάρχει κάποιο Plot για τη σύγκριση αυτή, καθώς τα πειράματα πραγματοποιήθηκαν σε c++.

Παραδοτέο:

reduce.py : Εκπαίδευση autoencoder και δημιουργία αρχείων latent dimension

reducer.h5: Το καλύτερο μοντέλο που εκπαιδεύσαμε

Ερώτημα Β:

Τα αρχεία των δεδομένων στο νέο χώρο που παράγει το παραδοτέο του ερωτήματος Α διαβάζονται από το τροποποιημένο πρόγραμμα lsh. Στη συνέχεια, γίνεται εφαρμογή του αλγόριθμου πλησιέστερου γείτονα lsh του 1ου παραδοτέου, εξαντλητικού αλγορίθμου πλησιέστερου γείτονα στον νέο αλλά και στον αρχικό χώρο. Οι 2 προσεγγίσεις συγκρίνονται ως προς τον χρόνο και ως προς την ακρίβεια σχετικά με τον εξαντλητικό αλγόριθμο.

Τα αποτελέσματα των πειραμάτων με όγκο δεδομένων 5000 και αριθμό ερωτημάτων 1000 είναι τα εξής:

Χρόνοι:
- LSH: 1/30 του εξαντλητικού
- Reduced: 1/10 του εξαντλητικού

Approximation Factor:
- LSH: 1.4
- Reduced: 2.4

Να σημειωθεί ότι η ακρίβεια του reduced βελτιώνεται ριζικά αν χρησιμοποιηθεί ολόκληρο το dataset. Κάτι τέτοιο όμως είναι χρονοβόρο στα μηχανήματα που το δοκιμάσαμε και γι αυτό χρησιμοποιείται το υποσύνολο δεδομένων που αναφέραμε παραπάνω.

Παραδοτέα:
Όλα τα παραδοτέα της πρώτης εργασίας εκτός από αυτά του hypercube. Τα ονόματα και οι λειτουργικότητες των αρχείων παρέμειναν ως έχει.

Util.cpp: Το αρχείο αυτό περιλαμβάνει συναρτήσεις γενικής χρήσεως που χρησιμοποιούνται και από τα 3 προγράμματα που υλοποιήθηκαν. Τέτοιες συναρτήσεις είναι η μετρική manhattan, modular power, συνάρτηση για τον υπολογισμό των s και άλλες.

hash.cpp: Στο αρχείο αυτό βρίσκονται οι κλάσεις του hashtable για lsh και για hypercube. Επίσης, όλες οι μέθοδοι για τις εν λόγω κλασσεις.

lsh_func.cpp: Αυτό το αρχείο περιέχει τις συναρτήσεις που χρησιμοποιεί ο αλγόριθμος lsh και η main που τον υλοποιεί.

lsh_main.cpp: Το main πρόγραμμα που χρησιμοποιεί lsh για να υλοποιήσει τον αλγόριθμο K-Nearest neighbor με χρήση lsh, καθώς και τον εξαντλητικό αλγόριθμο στον νέο αλλά και αρχικό χώρο.

Τέλος, τα κατάλληλα αρχεία επικεφαλίδας υλοποιήθηκαν για τις δηλώσεις κλάσεων, struct, μεθόδων και συναρτήσεων.

Εκτέλεση:
$./search -d <input file original space> -i <input file new space> -q <query file original space> -s <query file new space> -k <int> -L <int> -o <output file>

Ερώτημα Γ:

Η μετρική EMD υλοποιήθηκε εξ'ολοκλήρου σε Python με χρήση της βιβλιοθήκης PULP, που ενδίκνειται για επίλυση προβλημάτων γραμμικού προγραμματισμού.

Το πρόγραμμα αρχικοποιεί το πρόβλημα, χωρίζοντας τις εικόνες σε clusters μεγέθους που ορίζεται από μια υπερπαράμετρο n (hard-coded. Για τα πειράματα, αλλάζαμε την μεταβλητή στη γραμμή 192 του αρχείου emd.py). Στη συνέχεια, καλέι έναν solver για να το λύσει και αποθηκεύει τα αποτελέσματά του σε μια λίστα, ή οποία γίνεται sort για να απομονωθούν τα 10 καλύτερα.

Η σύγκριση με τον εξαντλητικό αλγόριθμο 10-Nearest Neighbor έγινε με χρήση 100 εικόνων στα δεδομένα και 10 εικόνων ως ερωτήματα. Αυτό έγινε λόγω χρονικού περιορισμού της επίλυσης προβλήματος γραμμικού προγραμματισμού, καθώς αυξανόταν αρκετά με χρήση περισσότερων δεδομένων.

Cluster Size: 2x2
EMD : 0.82
MANHATTAN : 0.7

Cluster Size: 4x4
EMD : 0.66
MANHATTAN : 0.7

Cluster Size: 7x7
EMD : 0.47
MANHATTAN : 0.7

Cluster Size: 14x14
EMD : 0.36
MANHATTAN : 0.7

Αξίζει να σημειωθεί ότι με χρήση ολόκληρου του dataset (60.000 images), η εξαντλητική αναζήτηση (manhattan) έφτανε ακρίβεια >0.97.

Επίσης, δοκιμάσαμε να αλλάξουμε τα constraints για το πρόβλημα γραμμικού προγραμματισμού έτσι ώστε το άθροισμα των flows να είναι μικρότερο ή ίσο από τα βάρη και όχι μόνο ίσο. Αυτό είχε ως αποτέλεσμα σημαντική αύξηση στην απόδοση της μετρικής EMD. Πχ για cluster size 7x7, η ορθότητα ανέβαινε δραστικά, στο 0.67. Αυτό ίσως να οφείλεται στα πιο 'χαλαρά' constraints του προβλήματος, που δέχονται λύσεις οι οποίες κανονικά απορρίπτονται. Το αναφέρουμε ως παρατήρηση.

Παραδοτέα:
emd.py

Εκτέλεση:
$python emd.py -d <input file original space> -q <query file original space> -l1 <labels of input dataset> -l2 <labels of query dataset> -o <output file>

Ερώτημα Δ:

Το εκτελέσιμο του 2ου παραδοτέου τροποποιήθηκε για να αποθηκεύει τα αποτελέσματα της κατηγοριοποίησης που υλοποιεί σε ένα αρχείο. Το αρχείο είναι σε μορφή txt και αποθηκεύει τις προβλέψεις σε μορφή clusters.

Το αρχείο αυτό διαβάζεται από το τροποποιημένο εκτελέσιμο του 1ου παραδοτέου και στη συνέχεια υλοποιούνται άλλες 2 συσταδοποιήσεις. Η πρώτη είναι απλή Lloyd's στον αρχικό χώρο και η δεύτερη είναι lloyd's στον νέο χώρο. Μαζί με τη συσταδοποίηση του classification, συγκρίνονται ως προς τη σιλουέτα και την τιμή της συνάρτησης στόχου. Και οι 2 αυτές μετρικές υπολογίζονται στον αρχικό χώρο και στις 3 συσταδοποιήσεις.

Τα πειράματα πραγματοποιήθηκαν με 5000 εικόνες ως δεδομένα και 1000 ως ερωτήματα. Τα αποτελέσματα των πειραμάτων είναι τα εξής:

NEW SPACE (Σ1)
clustering_time: 0.115691
Silhouette: [-0.00510937 , -0.00781798 , -0.0224679 , -0.01745 , -0.0161227 , -0.00267406 , -0.00609281 , -0.0108353 , -0.00887873 , -0.0117019 , -0.00990217]
Value of Objective Function: 211953438.0

ORIGINAL SPACE (Σ2)
clustering_time: 1.09878
Silhouette: [0.0636207 , 0.0640623 , 0.0710045 , 0.0614209 , 0.0755979 , 0.0636859 , 0.0679713 , 0.0644018 , 0.0734766 , 0.0743686 , 0.0683029]
Value of Objective Function: 171619279.0

CLASSES AS CLUSTERS (Σ3)
Silhouette: [-0.0131872 , -0.00211594 , -0.0207205 , -0.00645306 , -0.00945983 , -0.0185369 , -0.0108015 , -0.00320566 , -0.0160065 , -0.00967925 , -0.0106585]
Value of Objective Function: 211919632.0

Παρατηρούμε ότι ο αρχικός χώρος παράγει καλύτερη συσταδοποίηση ως προς τη σιλουέτα και τη συνάρτηση στόχο. Αυτό είναι λογικό, καθώς συγκρατεί περισσότερη πληροφορία από τον νέο χώρο. Ο χρόνος εκτέλεσής του είναι επίσης μεγαλύτερος και η διαφορά του από τους άλλους χρόνους θα είναι μεγαλύτερη αν χρησιμοποιηθεί ολόκληρο το dataset.

Μεταξύ των Σ1 και Σ3, παρατηρούνται παρόμοιες μετρικές απόδοσης. Αυτό οφείλεται λογικά στη χρήση μικρού dataset, όμως λόγω χρονικών περιορισμών και περιορισμών στο hardware, δεν ήταν δυνατή η χρήση μεγαλύτερων αρχείων. Θεωρητικά, περιμέναμε καλύτερη απόδοση μεταξύ των 2 να παρουσιάζει η Σ3, αν θεωρήσουμε ότι η καλή συσταδοποίηση ταυτίζεται με σωστή κατηγοριοποίηση από το νευρωνικό δίκτυο.

Κατά την μείωση της διάστασης των εικόνων χάνεται αρκετή πληροφορία, οπότε περιμένουμε το Σ1 να είναι το λιγότερο αποδοτικό απο τα 3.

Παραδοτέα

classification.py: Η υλοποίηση της κατηγοριοποίησης ίδια με την 2η εργασία.

good_autoencoder.h5: Ένας αυτοκωδικοποιητής για την εκπαίδευση του classifier

good_classifier.h5: Ένας καλά εκπαιδευμένος classifier για την εξέταση

kmeans_func.cpp: Αυτό το αρχείο περιέχει τις συναρτήσεις που χρησιμοποιεί ο αλγόριθμος kmeans και η main που τον υλοποιεί.

kmeans_main.cpp: Το main πρόγραμμα που χρησιμοποιεί kmeans clustering με απλό Lloyd's στον νέο και αρχικό χώρο και συγκρίνει τις 3 συσταδοποιήσεις.

Τέλος, τα κατάλληλα αρχεία επικεφαλίδας υλοποιήθηκαν για τις δηλώσεις κλάσεων, struct, μεθόδων και συναρτήσεων.

Εκτέλεση:
$./cluster -d <input file original space> -i <input file new space>
-n <classes from NN as clusters file> -c <configuration file> -o <output file>

Γενικές οδηγίες:

Μαζί με τα παραδοτέα, υπάρχει αρχείο makefile για την μεταγλώττιση των c++ αρχείων.

make για μεταγλώττιση όλων των αρχείων
make clean για τη διαγραφή όλων των εκτελέσιμων

Επίσης δίνονται config files για την εξέταση.