ΜΕΛΗ

Όνομα	Επώνυμο	ΑΜ
Άννα	Πετρίδου	1115201600135
Μιχάλης	Ανδρουλάκης	1115201600004

Το αρχειο μας αποτελειται απο ενα φακελο project3 και απο 4 υποφακελους,τους Α,Β,Γ,Δ οπου εχουν τα αντιστοιχα κομματια μεσα τους και τον υποφακελο Functions_for_both_algorithms που περιεχει κωδικα της πρωτης εργασιας και χρησιμοποιειται απο τα Α,Β και Δ κομματια.

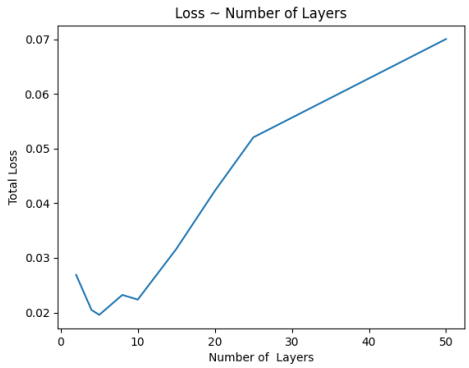
Α' ΜΕΡΟΣ

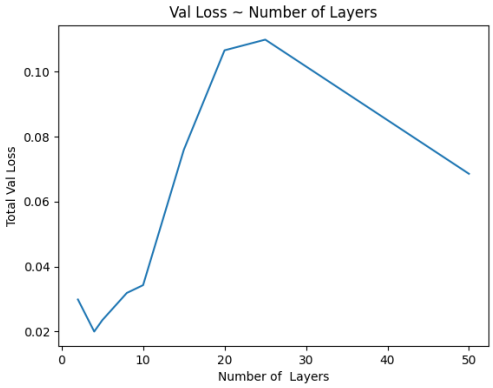
Το Α μέρος το υλοποιήσαμε σε python. Χρησιμοποιώντας το πρότυπο του autoencoder της προηγουμενης εργασιας, φτιαξαμε εναν νεο autoencoder που παιρνει τις εξης υπερπαραμετρους: number of filters, number of hidden layers, filter size dimension d(τετραγωνη διασταση οπως στην προηγουμενη εργασια dxd), number of epochs, batch size, learning rate και latent dimension με default=10. Προσθεσαμε Flatten layer στον encoder και FC και Reshape layer στον decoder. Επειτα καναμε μια σειρα πειραματων για να επιλεξουμε τις ιδανικες υπερπαραμετρους. Τα αποτελεσματα των πειραματων φαινονται παρακατω.

Οι μετρησεις εγιναν με τις εξης σταθερες παραμετρους, κρατωντας τις σταθερες και αλλαζοντας μια καθε φορα:

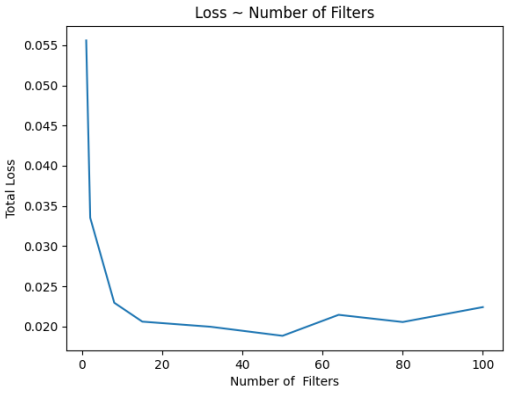
filters:32, layers:5, filtersize:3 ,epochs:10, batch: 100, lerning rate: 0.001, latent dimensions: 10

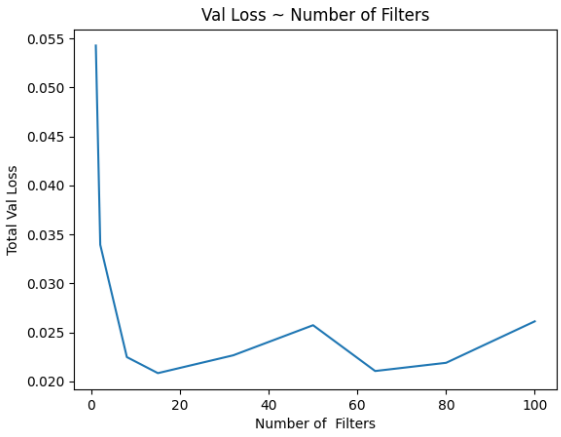
Στα



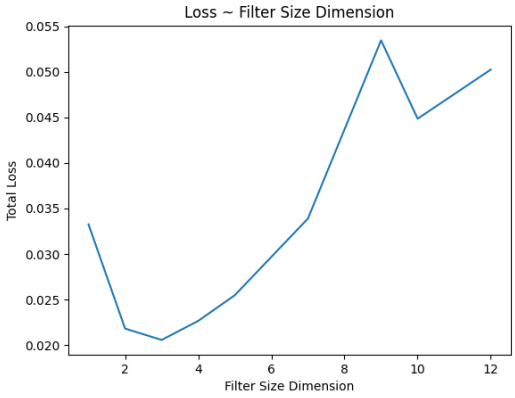


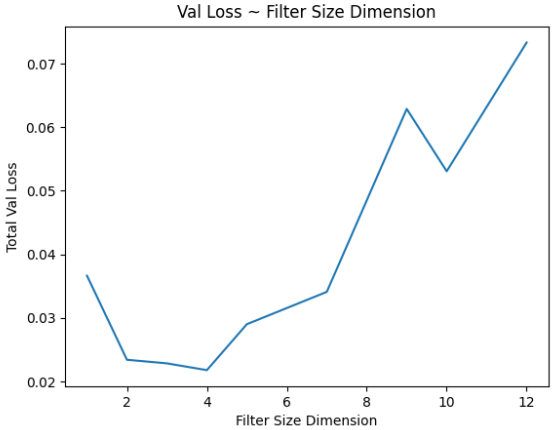
layers παρατηρουμε πως η ιδανικη τιμη ειναι γυρω στο 10, καθως μετα αυξανονται και το val loss και το loss.



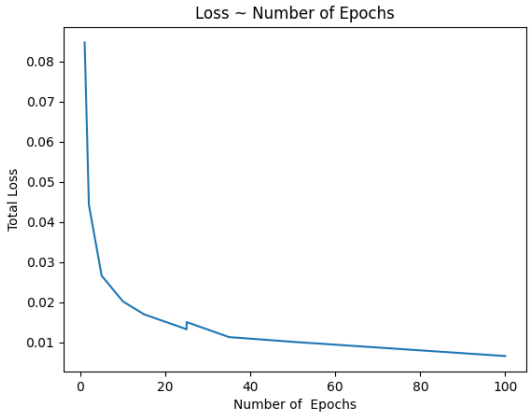


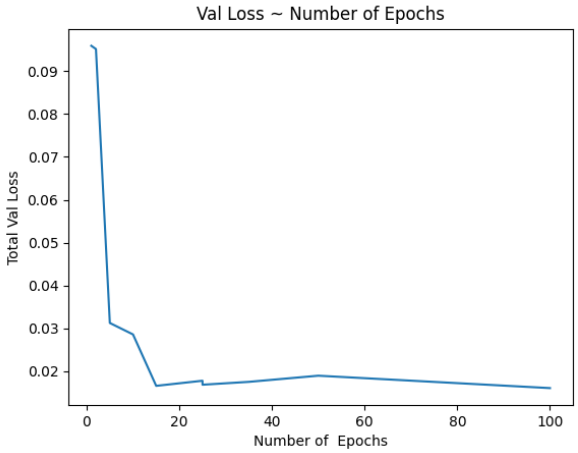
Για τα φιλτρα παρατηρουμε πως το val loss πιανει μια κατωτατη τιμη γυρω στο 15.



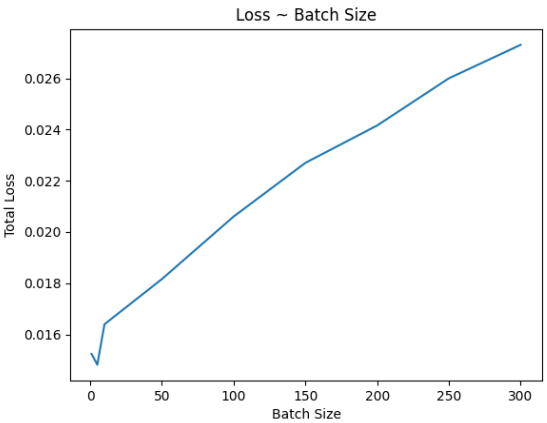


Για το μεγεθος των φιλτρων εχουμε ως βελτιστη τιμη το 4 οπως παρατηρουμε και απ το val loss και κοντα και το loss.

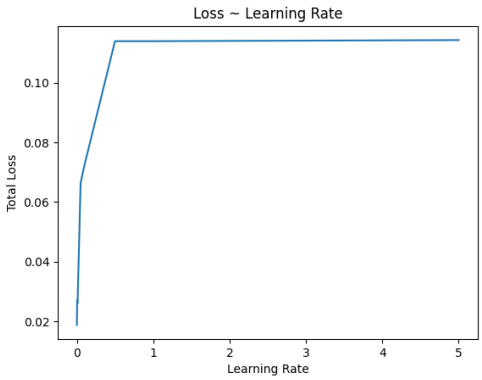


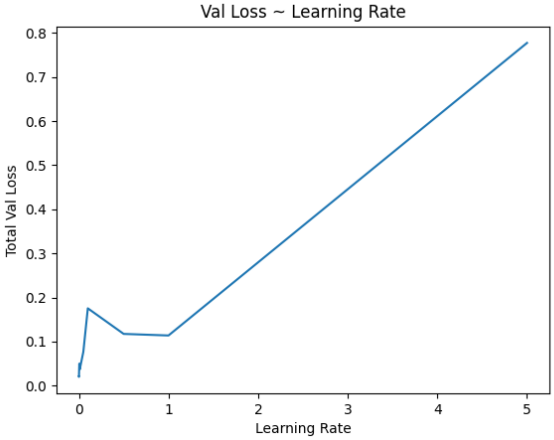


Για τις εποχες, παρατηρουμε οτι οσο περισσοτερες ειναι οι εποχες τοσο λιγοτερο το loss και val loss. Οποτε θα επιλεγαμε μια αρκετα μεγαλη τιμη.

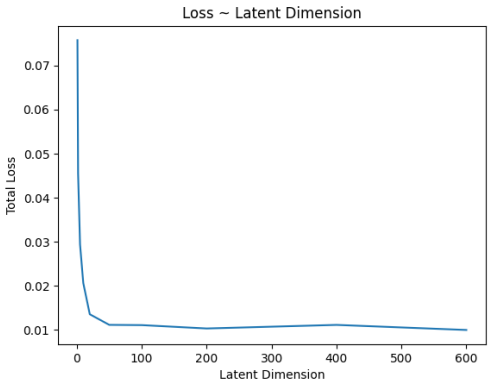


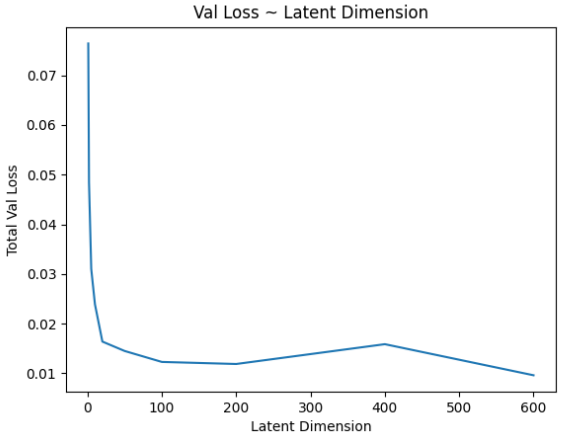
Για τα batches, παρατηρουμε πως το Loss πιανει και στις δυο περιπτωσεις μια κατωτατη τιμη γυρω στο 1-2.





Για το learning rate παρατηρουμε πως η βελτιστη τιμη ειναι πολυ κοντα στο 0, δηλαδη το 0,0001-0,001.





Για τον αριθμο των νεων διαστασεων, παρατηρουμε πως οσο αυξανεται, τοσο μειωνεται το loss, οποτε θα μπορουσαμε να επιλεξουμε εναν αριθμο οπως το 600 αν και η χρηση του latent δεν εξυπηρετει το σκοπο αυτο.

Επισης τα δεδομενα χωριστηκαν οπως και στη προηγουμενη εργασια σε training set και validation set. Στη συνεχεια, αφου ειχαμε το μοντελο μας, κανονικοποιησαμε τις εικονες στο (0,1) και τις μετατρεψαμε απο τις αρχικες διαστασεις στις νεες (πχ 1x10). Επειτα γραψαμε το νεο αρχειο που δημιουργησαμε που ειναι 2 bytes.

Δημιουργησαμε το reduce.py η οποια καλωντας τις καταλληλες συναρτησεις συμπιεζει καταλληλα το χωρο στις διαστασεις που

δοθηκαν (Latent) και δημιουργει το νεο αρχειο οπως εξηγησαμε παραπανω.

Εκτελεση προγραμματος με την εντολη:

Για το reduce.py:

$python3.8 reduce.py -d t10k-images.idx3-ubyte -od t10k-images.idx3-uhalfword -q train-images.idx3-ubyte -oq train-images.idx3-uhalfword

Για τον autoencoder_v3.py που δημιουργει το μονετλο:

$python3.8 autoencoder_v3.py -d t10k-images.idx3-ubyte

Τα περιεχομενα του φακελου ειναι:

autoencoder_v3.py: δεχεται τις υπερπαραμετρους και δημιουργει το ζητουμενο μοντελο

autoencoding.py: δημιουργια μοντελου

encoder_decoder: υλοποιηση του ecnoder και decoder

graphs.py: για την εκπυτωση των γραφικων

inputOutputMNIST: για το διαβασμα και γραψιμο αρχειων MNIST

reading_and_converting.py: διαβασμα ορισματων

reduce.py: διαβαζει το αρχικο αρχειο κ χρησιμοποιωντας το μοντελο φτιαχνει το νεο αρχειο ανα 2 byte στο συμπιεσμενο χωρο

Β' ΜΕΡΟΣ

Το μερος αυτο εγινε σε c++ κωδικα.Για το Β μερος χρησιμοποιησαμε τα κομματια lsh και true απο τη 1η εργασια με μικροαλλαγες και επειτα υλοποιησαμε τον Reduced χωρο.
Απο το Α επειτα απο μετρησεις πηραμε τα 2 καλυτερα αρχεια εικονων,ενα train και ενα test, τα οποια τα εχουμε επισυναψει στον Β φακελο και ειναι τα train-images.idx3-uhalfword και t10k-images.idx3-uhalfword , τα οποια περιεχουν εικονες μεγεθους 10 διστασεων οπου η πληροφορια για το καθε pixel προκυπτει απο το διαβασμα 2 bytes απο το αρχειο και συνενωση τους.Βρηκαμε εξαντλητικα το καλυτερο γειτονα ολων των queries στον Reduced χωρο,και επειτα υπολογισαμε τη αποσταση του γειτονα που βρηκαμε με το query στον αρχικο χωρο.

Για τα ενδεικτικα αποτελεσματα μας διατηρησαμε ολο το input file(60.000 εικονες) και ολο το query file (10.000 εικονες) και βρηκαμε:

Για k=4 και L=5:

tReduced: 327.008 secs
tLSH: 1439.16 secs
tTrue: 5849.43 secs
Approximation Factor LSH: 1.12371
Approximation Factor Reduced: 1.23968

Για k=4 και L=6:

tReduced: 328.551 secs
tLSH: 1812.94 secs
tTrue: 5447.67 secs
Approximation Factor LSH: 1.11478
Approximation Factor Reduced: 1.23968

Για k=5 και L=5:

tReduced: 315.035 secs
tLSH: 1352.87 secs
tTrue: 7623.44 secs

Approximation Factor LSH: 1.14763
Approximation Factor Reduced: 1.23968

Για k=5 και L=6:

tReduced: 317.238 secs
tLSH: 1860.103 secs
tTrue: 5335.53 secs
Approximation Factor LSH: 1.12804
Approximation Factor Reduced: 1.23968

Τα περιεχομενα του φακελα ειναι:

Hash.cpp - Hash.h : οπου βρισκεται η δομη του Lsh

Lsh.cpp - Lsh.h : οπου βρικονται οι συναρτησεις για το hashing του train file καθως και τον υπολογισμο των Lsh αλλα και των πραγματικων γειτονων των queries

Nearest_Neighbor.cpp - Nearest_Neighbor.h : οπου βρισκεται η συναρτηση για τον υπολογισμο του πραγματικου κοντινοτερου γειτονα ενος query

Reading.cpp - Reading.h : οπου βρισκονται οι συναρτησεις , για το διαβασμα των ορισματων της γραμμης εντολων , καθως και το διαβασμα αρχικου και νεο χωρου αρχειων

Reduced_Space.cpp - Reduced_Space.h : οπου βρισκεται η συναρτηση για την ευρεση του πραγματικα πλησιεστερου γειτονα στον νεο χωρο αλλα και μια συναρτηση για τον υπολογισμο των average σωστων εικονων που δινει ο Reduced χωρος για τις συγκρισεις με το Γ

Search.cpp : οπου εδω υπαρχει η main και καλουνται ολες οι υπολοιπες συναρτησεις

t10k-images.idx3-uhalfword : οπου ειναι το test αρχειο στις 10 διαστασεις που προεκυψε απο το Α

train-images.idx3-uhalfword : οπου ειναι το train αρχειο στις 10 διαστασεις που προεκυψε απο το Α

makefile : για να κανουμε compile το κωδικα μας

Εκτελεση του προγραμματος με την εντολη: ./search -d ../train-images.idx3-ubyte -q ../t10k-images.idx3-ubyte -i train-images.idx3-uhalfword -s t10k-images.idx3-uhalfword -o output.txt -k 5 -L 5

Γ' ΜΕΡΟΣ

Το μερος αυτο εγινε σε python κωδικα.Υλοποιησαμε τη μετρικη Manhattan που απλα μετατρεψαμε τη συναρτηση της c++ σε python αλλα και την EMD χρησιμοποιοντας το linprog της scipy βιβλιοθηκης.Για την EMD,επειτα απο κανονικοποιηση που καναμε στα βαρη των εικονων τις αποστασεις μεταξυ του καθε cluster input και query εικονας και επειτα φτιαχνουμε την linprog δηλαδη την objective function αλλα και τις εξισωσεις γεμιζοντας τα καταλληλα vectors.
Τα αποτελεσματα υστερα απο πειραματα που πηραμε ειναι :

Για clusters 2x2

Για 10 queries και 100 input:

Time_EMD : 8.61 secs και Average Correct EMD: 0.279999
Time_Manhattan : 7.55 secs και Average Correct Manhattan: 0.56

Για 10 queries και 1000 input:

Time_EMD : 78.76 secs και Average Correct EMD: 0.4
Time_Manhattan : 38.87 secs και Average Correct Manhattan: 0.73

Για 10 queries και 10000 input:

Time_EMD : 392.59 seconds και Average Correct EMD: 0.3299
Time_Manhattan : 341.52 και Average Correct Manhattan : 0.89

Για 100 queries και 100 input:

Time_EMD: 97.20 και Average Correct EMD: 0.23
Time_Manhattan : 74.40 και Average Correct Manhattan: 0.48

Για 100 queries και 1000 input:

Time_EMD : 413.96 secs και Average Correct EMD:0.314

Time_Manhattan : 349.12 secs και Average Correct Manhattan: 0.75

Για 100 queries και 10000 input:

Time_EMD : 4073.42 secs και Average Correct EMD:0.354
Time_Manhattan :3407.39 secs και Average Correct Manhattan:0.907

Για clusters 4x4:

Για 10 queries και 100 input:

Time_EMD : 48.60 secs και Average Correct EMD: 0.41

Time_Manhattan : 3.55 secs και Average Correct Manhattan: 0.56

Για 10 queries και 1000 input:

Time_EMD : 825.48 secs και Average Correct EMD: 0.59
Time_Manhattan : 36.27 secs και Average Correct Manhattan: 0.73

Για 10 queries και 10000 input:

Time_EMD : 5754.47 secs και Average Correct EMD: 0.68
Time_Manhattan : 340.90 secs και Average Correct Manhattan: 0.89

Για 100 queries και 100 input:

Time_EMD : 782.27 secs και Average Correct EMD: 0.37
Time_Manhattan : 36.96 secs και Average Correct Manhattan: 0.48

Για 100 queries και 1000 input:

Time_EMD : 6694.10 secs και Average Correct EMD: 0.602
Time_Manhattan : 353.80 secs και Average Correct Manhattan: 0.75

Για clusters 7x7:

Για queries 10 και input 100:

Time_EMD : 5329.80 secs και Average Correct EMD: 0.49
Time_Manhattan : 3.65 secs και Average Correct Manhattan 0.56

Για queries 100 και input 100:

Time_EMD : 44309.18 secs και Average Correct EMD: 0.47
Time_Manhattan : 35.65 secs και Average Correct Manhattan : 0.48

Για τα clusters 7x7 και 4x4 δε μπορεσαμε να κανουμε μεγαλυτερες μετρησεις καθως μας παιρνανε πανω απο μια μερα.

Εκτος απο αυτες τις μετρησεις εχουμε και μετρησεις συγκρισης μεταξυ του Β κομματιου του Reduced χωρου.Για τις συγκρισεις χρησιμοποιησαμε 1 πλησιεστερο γειτονα και στα 2 κομματια και μετρησαμε τους χρονους αλλα και τα αποτελεσματα. Για να κανουμε τις συγκρισεις αυτες φτιαξαμε μια συναρτηση στο Β που διαβαζει τα labels και για τα πρωτα 10 queries υπολογιζει το ποσοστο των σωστων labels που βρηκε η reduced μετρικη για N=1. Αναλογως για το Γ παλι κανοντας μετρησεις για N=1 βρηκαμε το ποσοστο των 10 πρωτων queries.Χρησιμοποιηθηκαν οι πρωτες 10000 εικονες του dataset.

Time_Reduced=317.65 seconds και average_reduced=0.9 δηλαδη 9/10 queries ο πλησιεστερος γειτονας του reduced χωρου εχει το ιδιο label με το query

Για clusters=4x4 time_EMD=5554.47 seconds και average_EMD=0.6

Για clusters=2x2 time_EMD=392.57 seconds και average_EMD=0.3299

Τα περιεχομενα του φακελου ειναι:

EMD.py : Που περιεχει την συναρτηση EMD_Distance για τον υπολογισμο της EMD αποστασης μεταξυ 2 εικονων

Manhattan.py : Που περιεχει τη συναρτηση Manhattan_Distance για τον υπολογισμο της Manhattan αποστασης 2 εικονων

Quicksort.py : Που περιεχει τις συναρτησεις quickSort και partition για για το sort και την ευρεση των 10 πλησιεστερων γειτονων

reading_and_converting.py : Που περιεχει τις συναρτησεις readingArguements για διαβασμα της γραμμης εντολων και file_normalization ωστε να κανονικοποιησουμε τα αρχεια

Writing.py : Που περιεχει τη συναρτηση για το γραψιμο του αποτελεσματος σε ενα αρχειο

search.py : Που περιεχει τη main

Εκτελεση προγραμματος με την εντολη: python3 search.py -d ../train-images.idx3-ubyte -q ../t10k-images.idx3-ubyte -l1 ../train-labels.idx1-ubyte -l2 ../t10k-labels.idx1-ubyte -o output.txt

Δ' ΜΕΡΟΣ

Για το Δ μερος καναμε αρχικα το Σ1, δηλαδη το clustering για τις εικονες στο νεο χωρο, διαβαζοντας το αρχειο που δημιουργησαμε στην Α ερωτηση, καταλληλα ανα 2 bytes. Αφου ολοκληρωσαμε το clustering, καναμε την αντιστοιχια των clusters με τα σημεια που προεκυψαν, στον αρχικο μας χωρο, υπολογισαμε τα νεα centroids και ετσι βρηκαμε και εκτυπωσαμε το silhouette. Επειτα καναμε το Σ2, το οποιο ειναι το clustering που ειχαμε και στη 1η εργασια, στον αρχικο χωρο και βρηκαμε και παλι το silhouette.

Τελος, παιρνοντας το αρχειο απ την κατηγοριοποιηση στη 2η εργασια, καναμε και παλι την αντιστοιχια και βρηκαμε τα αντιχτοιχα σημεια των clusters στον αρχικο χωρο. Πηραμε τις συντεταγμενες τους, υπολογισαμε τα νεα centroids και καναμε τελος silhouette.

Ενδεικτικα αποτελεσματα μετα απο μια εκτελεση στα 10000 εχουμε:

NEW SPACE

CLUSTER-0 {size: 1339, centroid:...

CLUSTER-1 {size: 642, centroid:...

CLUSTER-2 {size: 1073, centroid:...

CLUSTER-3 {size: 1060, centroid:..

CLUSTER-4 {size: 1008, centroid:...

CLUSTER-5 {size: 1408, centroid:...

CLUSTER-6 {size: 2324, centroid:...

CLUSTER-7 {size: 393, centroid:...

CLUSTER-8 {size: 221, centroid:...

CLUSTER-9 {size: 532, centroid:...

clustering_time: 1 sec

Silhouette:

[0.00469718,0.00214029,0.019,....,0,04157820]

ORIGINAL SPACE

CLUSTER-0 {size: 7551, centroid:...

CLUSTER-1 {size: 143, centroid:...

CLUSTER-2 {size: 572, centroid:...

CLUSTER-3 {size: 11, centroid:..

CLUSTER-4 {size: 91, centroid:...

CLUSTER-5 {size: 231, centroid:...

CLUSTER-6 {size: 4, centroid:...

CLUSTER-7 {size: 501, centroid:...

CLUSTER-8 {size: 286, centroid:...

CLUSTER-9 {size: 610, centroid:...

clustering_time: 26 sec

Silhouette:

[0.129548,0.143949,...,0.144479]

Εκτελεση προγραμματος με την εντολη:

$./cluster -i t10k-images.idx3-uhalfword -d t10k-images.idx3-ubyte -c config.txt -o output.txt -n class.txt

Τα περιεχομενα του φακελου ειναι ιδια με της προηγουμενης εργασιας. Ο,τι προστεθηκε, ηταν στα ηδη υπαρχοντα αρχεια.

Συνδεμος Github:https://github.com/Omada53/project3