Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

3η προγραμματιστική εργασία

Κίμων Σοφτάς: sdi1700145

Παπαδοπούλου Ευθυμία: sdi1600128

Αθήνα 2020

Περιεχόμενα

A. Image Autoencoder

Ξεκινήσαμε τους πειραματισμούς μας με την δομή του βέλτιστου autoencoder τον οποίο βρήκαμε στη 2η εργασία του μαθήματος. Ο βέλτιστος encoder σύμφωνα με τα πειράματά μας είχε την εξής δομή:

- 4 convolutional layers
- Filter_size = 3x3
- 32 Filters/Layer
- Epochs [100,300]
- Batch size = 64

Για τον decoder χρησιμοποιήσαμε “mirrored” δομή και τις ίδιες υπερπαραμέτρους.

Για τις ανάγκες του ερωτήματος προσθέσαμε 2 νέα layers στον encoder:

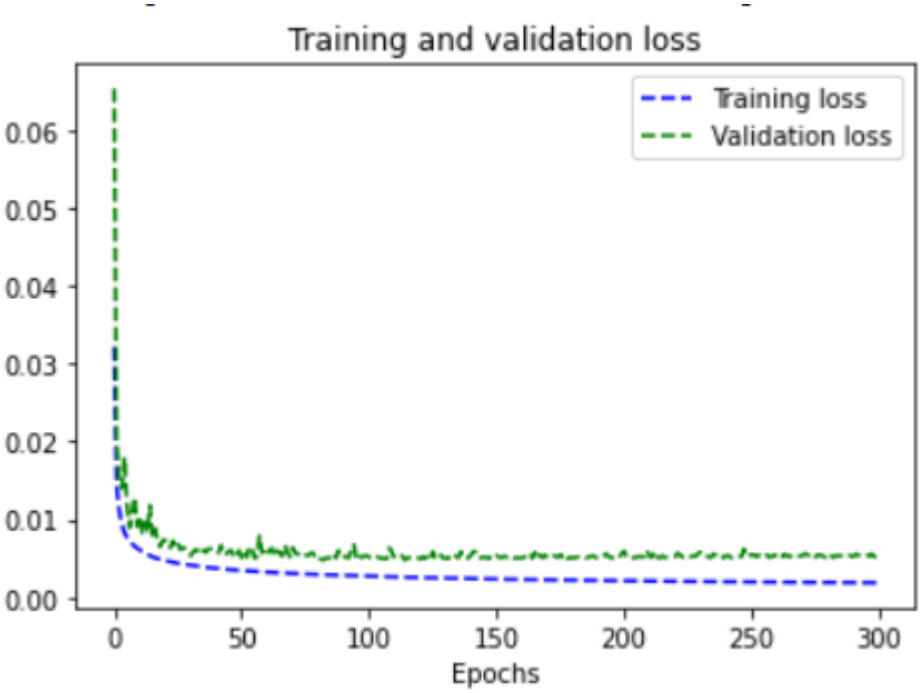
- 1 Flatten layer, το οποίο μετατρέπει την εικόνα σε διάνυσμα το οποίο τροφοδοτούμε στη συνέχεια σε ένα fully connected layer
- 1 Dense layer, το οποίο μας δίνει μια υπερσυμπυκνωμένη πληροφορία για την αρχική μας εικόνα. Ουσιαστικά μας δίνει ένα πολύ μικρότερο διάνυσμα από αυτό που μας έδωσε το flatten layer. Είναι ουσιαστικά ένα στρώμα συμπίεσης. Η καταλληλότερη διάσταση αυτού του διανύσματος για τα δεδομένα μας παρουσιάζεται παρακάτω.

Δύο νέα layers προσθέσαμε και στη δομή του decoder:

- 1 Dense layer, που πρακτικά αποσυμπιέζει την εικόνα που συμπιέσαμε στον encoder. Παραδείγματος χάριν, αν το input στο flatten layer ήταν εικόνα με διάσταση 7x7x64, στο παρών layer θα θέσουμε την παράμετρο unit ίση με 3.136.
- 1 Reshape layer ώστε να μπορέσουμε το διάνυσμα του dense layer, που προηγήθηκε, να το φέρουμε σε μορφή που μπορούν να επεξεργαστούν τα convolutional layers που έπονται.

Σημείωση: Μια επιπλέον διαφορά του νέου decoder με εκείνου που χρησιμοποιήσαμε στην προηγούμενη εργασία είναι πως αντί για Conv2D layers χρησιμοποιούμε Conv2DTranspose layers. Το convolutional 2D transpose layer εκτελεί μια λειτουργία αντίστροφης συνέλιξης. Επιθυμούμε να επαναφέρουμε την εικόνα στις αρχικές διαστάσεις. Με το απλό convolutional layer η εικόνα μας θα έχει ακριβώς το ίδιο σχήμα που είχε αρχικά, όμως οι τιμές στους πίνακες των επιπέδων δεν θα έχουν επαναφερθεί στη σωστή θέση. Για αυτό και το transpose layer είναι υπεύθυνο να τοποθετήσει τις τιμές ακριβώς στην ίδια θέση που ήταν προηγουμένως.

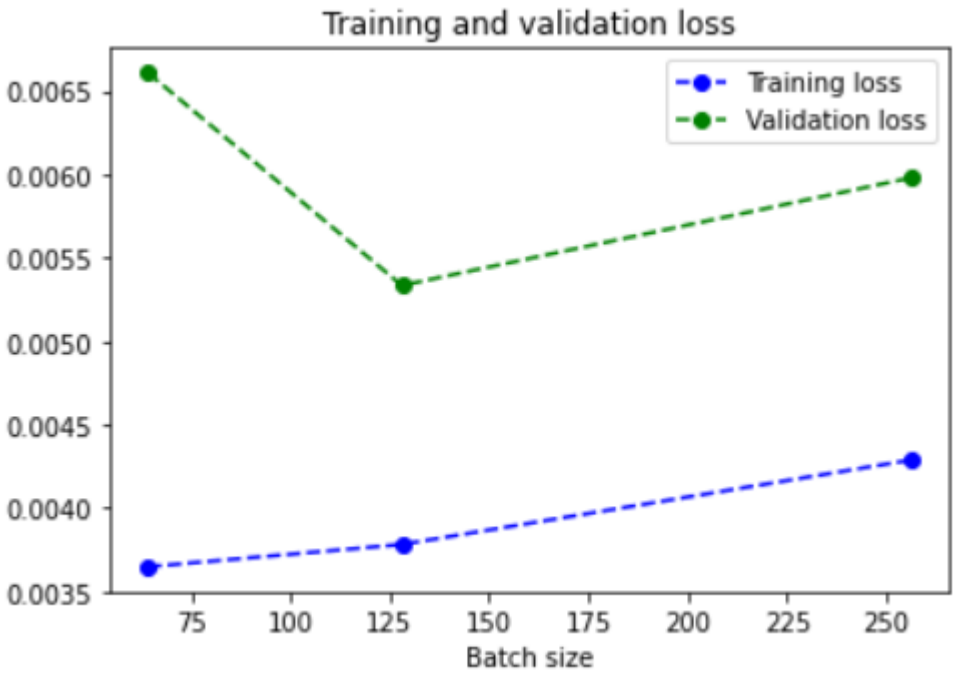
4 conv. Layers --- 3x3 filter size --- 32 filters/layer --- 64 batch size



Όπως και αναμέναμε, όσο αυξάνονται οι εποχές για τις οποίες εκπαιδεύεται ο autoencoder του προγράμματός μας και οι 2 απώλειές μειώνονται. Αν οι υπερπαράμετρος epochs παίρνει κάποια τιμή στο διάστημα [150-300] τα αποτελέσματα για το είδος των δεδομένων που διαχειριζόμαστε είναι πολύ ικανοποιητικά. Επειδή όμως οι εποχές είναι αυτές που καθορίζουν την ταχύτητα του προγράμματος μας συνεχίσαμε τους πειραματισμούς μας για 150 εποχές. Επιπλέον, είναι σημαντικό να αναφέρουμε πως δεν παρατηρείται το φαινόμενο του overfitting για αυτό και δεν χρειάστηκε να προσθέσουμε Dropout layers .

Σημείωση: Στον τίτλο αναφέρονται οι υπερπαράμετροι του *encoder* για τις οποίες έγιναν τα πειράματα.

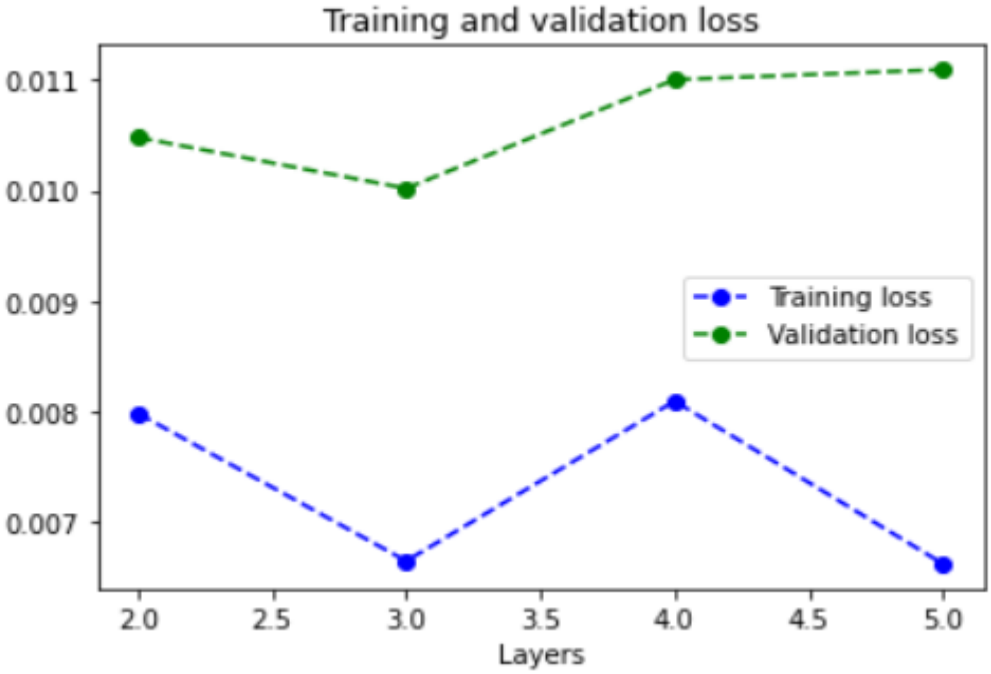
4 conv. layers --- 3x3 filter size --- 32 filters/layer --- 150 epochs



Τα αποτελέσματα που απεικονίζονται στη γραφική παράσταση, τα οποία προέκυψαν από την εκπαίδευση του autoencoder για batch size = 64, 128,256 είναι αρκετά αναμενόμενα. Ο λόγος που περιμέναμε να αυξάνονται οι απώλειες και για το training και για το validation loss είναι επειδή το batch size καθορίζει το πόσο συχνά θα γίνονται διορθώσεις των βαρών στα layers του CNN με βάση τους μέσους όρους σφάλματος. Παραδείγματος χάριν, για ένα training set των 60.000 εικόνων, αν batch size= 512 γίνονται 117 αναθεωρήσεις σε κάθε εποχή ενώ για batch size=32 γίνονται αντίστοιχα 1875 αναθεωρήσεις(!). Η διαφορά είναι σημαντική και αυτό αποτυπώνεται στη γραφική παράσταση. Θα μπορούσαν βέβαια οι ευθείες να είναι ακόμα πιο απότομες, αλλά το υψηλό πλήθος εποχών για το οποίο εκπαιδεύτηκε ο autoencoder αποτρέπει κάτι τέτοιο.

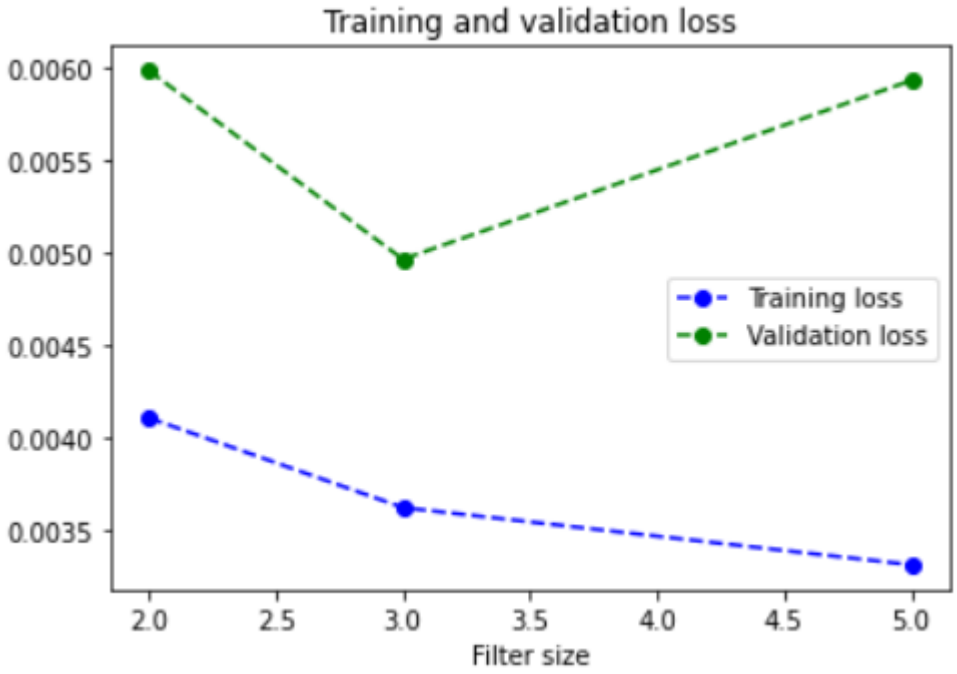
Συνεπώς επιλέγουμε ένα σχετικά μικρό batch size ίσο με 64. Θα μπορούσαμε ενδεχομένως να επιλέγαμε και την τιμή 32. Εκτός όμως ότι αποδίδει πολύ καλά το πρόγραμμα για batch size=64, το batch size είναι μια επιπλέον υπερπαράμετρος που παίζει ρόλο στο χρόνο εκπαίδευσης. Τέλος, η πολύ συχνή διόρθωση των εσωτερικών «αγνώστων-βαρών» των επιπέδων μπορεί να οδηγήσει και σε overfitting.

3x3 filter size --- 32 filters/layer --- 150 epochs --- 64 batch size



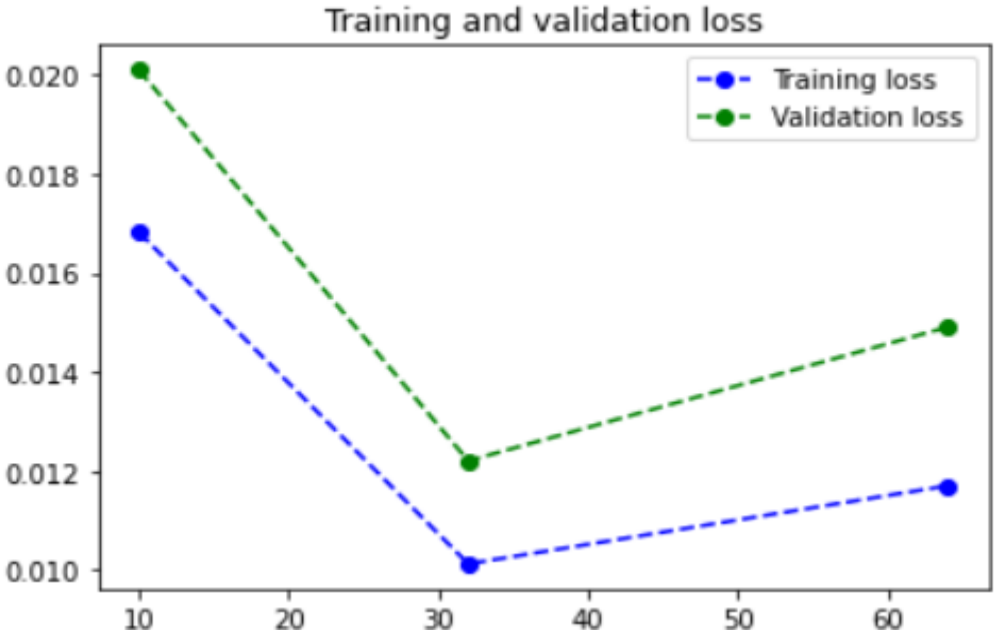
Σύμφωνα με τη γραφική παράσταση, καθώς και τα συμπεράσματα και τις παρατηρήσεις μας από την προηγούμενη εργασία, το καλύτερο trade και τις μικρότερες απώλειες τις έχουμε όταν το συνολικό πλήθος των convolutional layers είναι 4 για τον encoder και συνεπώς και για τον decoder. Οπότε και για το νέο νευρωνικό δίκτυο αυτοκωδικοποίησης εικόνων με bottleneck στρώματα θα χρησιμοποιήσουμε 4 convolutional layers.

4 conv. layers --- 32 filters/layer --- 150 epochs --- 64 batch size



Όπως διαπιστώνουμε, το κατάλληλο μέγεθος φίλτρου που εφαρμόζουν τα convolutional στρώματα του autoencoder είναι 3x3. Για μεγαλύτερο filter size είναι λογικό να αυξάνεται το loss καθώς η σημαντική πληροφορία, τα χρήσιμα χαρακτηριστικά κάθε εικόνας εντοπίζονται συνήθως τοπικά οπότε είναι λογικό να πρέπει να επεξεργαζόμαστε λίγα pixels κάθε φορά. Για αυτό και πειραματιστήκαμε μόνο με μικρά filter/kernel μεγέθη. Επιπλέον, ο λόγος που επιλέγουμε συνήθως σε τέτοιες περιπτώσεις περιττού μεγέθους φίλτρα είναι επειδή τα pixels των layers είναι συμμετρικά ως προς το pixel εξόδου. Αν χάσουμε αυτή τη συμμετρία θα πρέπει να λάβουμε υπόψη τις παραμορφώσεις μεταξύ των επιπέδων.

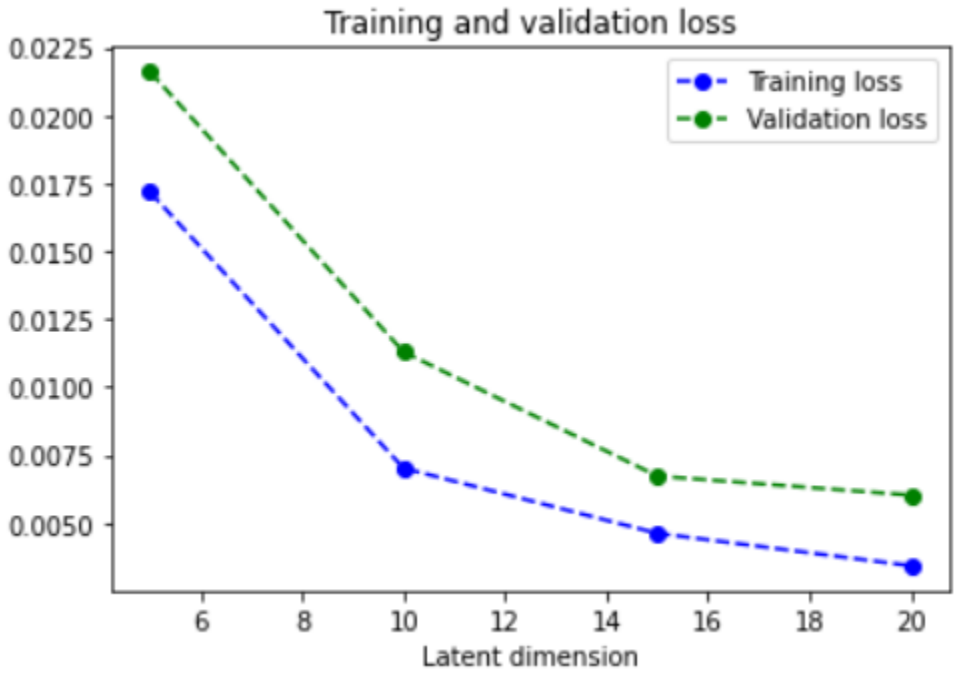
4 conv. layers --- 3x3 filter size --- 50 epochs --- 64 batch size



Το παραπάνω πείραμα είχε σαν υπερπαράμετρο το πλήθος των φίλτρων σε κάθε convolutional layer του autoencoder, και συνεπώς και σε κάθε convolutional transpose layer από τη στιγμή που ακολουθούμε πάντα mirrored δομή στον decoder.

Το συγκεκριμένο πείραμα έγινε για ίδιο αριθμό φίλτρων σε όλα τα layers. Παρόλο που φαίνεται πως τα 32 φίλτρα είναι η βέλτιστη επιλογή, πειραματιστήκαμε και για διαφορετικό πλήθος φίλτρων μέσα στον ίδιο autoencoder. Εκεί που καταλήξαμε είναι πως η βέλτιστη τιμή της υπερπαραμέτρου είναι τα φίλτρα να αυξάνονται όσο «μπαίνουμε βαθύτερα» στο CNN. Οπότε καταλήξαμε να χρησιμοποιούμε 32,64,128 και 256 φίλτρα για τα convolutional layers με τη σειρά που αναγράφονται.

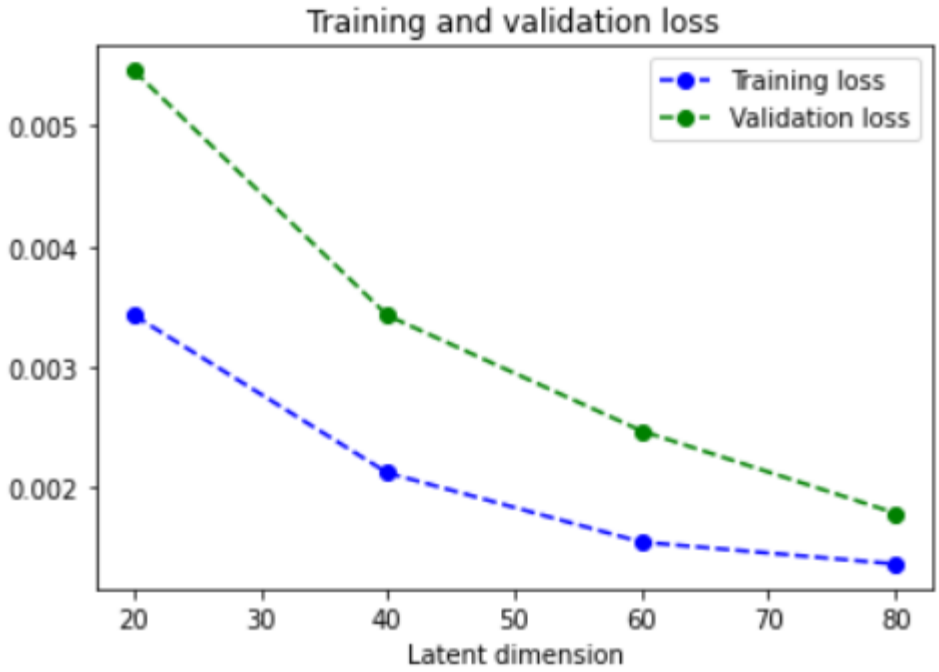
Πείραμα για latent dimension



Όλα τα προηγούμενα πειράματα πραγματοποιήθηκαν για latent dimension ίση με 10. Όμως η latent dimension αποτελεί μια νέα υπερπαράμετρο για την οποία πρέπει να πειραματιστούμε ώστε να επιλέξουμε αυτή που ταιριάζει καλύτερα στο πρόβλημα που καλούμαστε να λύσουμε.

Από την παραπάνω εικόνα, καθώς και από άλλα πειράματα που πραγματοποιήσαμε, είναι εμφανές πως όσο μεγαλώνει η latent dimension τόσο πέφτει το training και το validation loss. Το αποτέλεσμα είναι δικαιολογημένο καθώς αν έχουμε θέσει μια μεγάλη latent διάσταση σημαίνει πως η εικόνα μας δεν θα συμπιεστεί αρκετά, οπότε στο τελικό διάνυσμα, πριν την αποκωδικοποίηση, έχουμε κρατήσει περισσότερη πληροφορία για την αρχική μας εικόνα.

Το ερώτημα είναι γιατί δεν κρατάμε μια μεγάλη τιμή, έστω το 100; Αν κρατήσουμε μια τόσο μεγάλη τιμή χάνεται το νόημα της άσκησης και αποκλίνουμε από το ζητούμενο καθώς ο βασικός μας στόχος είναι η όσο το δυνατόν μεγαλύτερη μείωση των αρχικών διαστάσεων της εικόνας ώστε να επιτευχθεί καλύτερη απόδοση στην αναζήτηση.



Το παραπάνω διάγραμμα είναι ακόμα μια επιβεβαίωση των ισχυρισμών που κάναμε παραπάνω.

Βέλτιστες υπερπαράμετροι:

Μετά από πολλούς πειραματισμούς και συνδυασμούς υπερπαραμέτρων καταλήξαμε ότι οι βέλτιστες υπερπαράμετροι για το πρόβλημα που καλούμαστε να επιλύσουμε είναι:

- ✓ Πλήθος συνελικτικών στρωμάτων: **4**
- ✓ Μεγέθους συνελικτικών φίλτρων: **3x3**
- ✓ Πλήθος συνελικτικών φίλτρων ανά στρώμα: **32/64/128/256**
- ✓ Πλήθος εποχών εκπαίδευσης (epochs): **150**
- ✓ Μέγεθος δέσμης (batch size): **64**
- ✓ Διάστασης συμπίεσης: **10-20**

Σημείωση: Όλα τα παραπάνω πειράματα, για λόγους ταχύτητας, έγιναν στο Google Colab.

Αρχεία-Εκτέλεση:

- Το αρχείο του ερωτήματος έχει όνομα **reduce.py**
- Το αρχείο αυτό βρίσκεται στο φάκελο με όνομα: A
- Η εντολή εκτέλεσης είναι αυτή που αναγράφεται στην εκφώνηση

Β. Σύγκριση αναζήτησης γειτόνων σε διαφορετικούς διανυσματικούς χώρους

Για τις ανάγκες του ερωτήματος, αναζητήσαμε τον κοντινότερο γείτονα για κάθε εικόνα του query set με τους παρακάτω τρόπους:

1. Εξαντλητική αναζήτηση στο νέο χώρο *10-διαστάσεων*
2. Εξαντλητική αναζήτηση στον αρχικό χώρο *784-διαστάσεων*
3. Προσεγγιστική αναζήτηση με LSH στον αρχικό χώρο

Οι χρόνοι που παρουσιάζονται παρακάτω είναι οι μέσοι χρόνοι εύρεσης του κοντινότερου γείτονα ενός query**.**

Latent dimension = 10

Για 10.000 εικόνες από το queryset και 10.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**8.39 ms**	**1.00114**
True-784	**5.92 ms**	**---**
Reduced-10	**0.167 ms**	**1.27179**

Για 10.000 εικόνες από το queryset και 30.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**15.1 ms**	**1.001**
True-784	**17.4 ms**	**---**
Reduced-10	**0.31 ms**	**1.32716**

Για 10.000 εικόνες από το queryset και 60.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**27.3 ms**	**1.00312**
True-784	**36.4 ms**	**---**
Reduced-10	**0.71 ms**	**1.36164**

Παρατηρώντας προσεκτικά το output των παραπάνω αναζητήσεων διαπιστώσαμε πως το id, του κοντινότερου γείτονα μιας εικόνας από το queryset, σχεδόν πάντα είναι ίδιο στην Lsh αναζήτηση και στην αναζήτηση στον αρχικό χώρο. Αυτό προφανώς συμβαίνει και για την απόσταση του για αυτό και ο approximation factor του Lsh είναι τόσο καλός. Ο χρόνος βέβαια του Lsh θα μπορούσε να ήταν ακόμα καλύτερος αλλά στην $1^{η}$ εργασία επιλέξαμε παραμέτρους που καθυστερούν λίγο παραπάνω την εκτέλεση της αναζήτησης αλλά δίνουν καλύτερα αποτελέσματα.

Όσον αφορά τη σύγκριση της εξαντλητικής αναζήτησης στον αρχικό και στο νέο χώρο, το id του κοντινότερου γείτονα που δίνει η αναζήτηση στο νέο χώρο σπάνια συμπίπτει με το id που μας δίνει η αναζήτηση στον αρχικό. Όμως, η απόσταση μεταξύ των 2 αυτών γειτόνων είναι τις περισσότερες φορές ικανοποιητική αν αναλογιστούμε

Latent dimension = 20

Για 10.000 εικόνες από το queryset και 10.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**7 ms**	**1.00075**
True-784	**9.7 ms**	**---**
Reduced-10	**0.26 ms**	**1.26918**

Για 10.000 εικόνες από το queryset και 30.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**22.7 ms**	**1.00128**
True-784	**17.3 ms**	**---**
Reduced-10	**0.77 ms**	**1.30695**

Για 10.000 εικόνες από το queryset και 60.000 από το dataset

	Χρόνος	*Approximation Factor*
LSH	**33 ms**	**1.00222**
True-784	**35 ms**	**---**
Reduced-10	**1.55 ms**	**1.32399**

Σύγκριση αναζήτησης με διαφορετική latent dimension

Για 10.000 εικόνες από το queryset και 60.000 από το dataset

	Χρόνος dimension 10	*Χρόνος dimension 20*	*Approximation Factor dimension 10*	*Approximation Factor dimension 20*
LSH	**27.3 ms**	**30 ms**	**1.00312**	**1.00222**
True-784	**36.4 ms**	**35 ms**	**---**	**---**
Reduced-10	**0.71 ms**	**1.55 ms**	**1.36164**	**1.32399**

Όπως και αναμέναμε ο χρόνος αναζήτησης στο νέο χώρο, του κοντινότερου γείτονα μιας εικόνας, παραμένει εξαιρετικά χαμηλός και για latent dimension=20. Ο approximation factor βελτιώνεται ελαφρώς όταν γίνεται αναζήτηση στο νέο χώρο με 20 διαστάσεις, το οποίο είναι απολύτως λογικό από τη στιγμή που στις 20 διαστάσεις κρατάμε την διπλάσια πληροφορία για την αρχική εικόνα από ότι στις 10 διαστάσεις. Προφανώς όσο αυξάνεται η latent dimension θα μειώνεται το κλάσμα προσέγγισης.

Όσον αφορά τη σύγκριση του Lsh με την αναζήτηση στο νέο χώρο, το κλάσμα προσέγγισης (approximation factor) στο νέο χώρο είναι αισθητά μεγαλύτερο από εκείνο του Lsh. Στην περίπτωση του Lsh είναι εξαιρετικά χαμηλό και για τα 3 διαφορετικά πλήθη του dataset για τα οποία τρέξαμε την αναζήτηση, που σημαίνει πως για τις περισσότερες εικόνες του query set έβρισκε σχεδόν πάντα τον ίδιο γείτονα με την εξαντλητική αναζήτηση.

Ποια είναι η καλύτερη μέθοδος τελικά; Αρχικά εξαρτάται από το πρόβλημα που καλούμαστε να λύσουμε και υπό ποιες συνθήκες. Αν παραδείγματος χάριν καλούμαστε να βρούμε τους κοντινότερους γείτονες για ένα πρόβλημα που απαιτεί κυρίως ταχύτητα σίγουρα η αναζήτηση στον νέο χώρο είναι η κατάλληλη με μέτρια αποτελέσματα. Να σημειώσουμε κιόλας πως τα παραπάνω αποτελέσματα που συγκρίναμε είναι για 60.000 πιθανούς γείτονες. Αν όμως είχαμε ένα πρόβλημα που απαιτεί κυρίως ακρίβεια στην εκτίμηση και είχαμε ένα σύνολο με 10.000 πιθανούς γείτονες (και 10.000 queries) η Lsh θα χρειαζόταν 8.39 ms/query κατά μέσο όρο και συνεπώς 1,3 λεπτά για όλα τα queries και μια εξαιρετική ακρίβεια.

Αρχεία-Εκτέλεση:

Αρχεία εισόδου

- *input file new space:* input_new_space_10.txt (latent dimension=10)
- *input file new space:* input_new_space_20.txt (latent dimension=20)
- *query file new space:* query_new_space_10.txt (latent dimension=10)
- *query file new space:* query_new_space_20.txt (latent dimension=20)

- Μέσα στο φάκελο με όνομα B υπάρχουν όλα τα αρχεία που απαιτούνται και για τις 3 μεθόδους αναζήτησης
- Μεταγλώττιση αρχείων με την εντολή **make**
- Η εντολή εκτέλεσης είναι αυτή που αναγράφεται στην εκφώνηση
- Το output της εκτέλεσης με 10.000 εικόνες από το query set, 60.000 εικόνες από το dataset και latent dimension =10 βρίσκεται μέσα στο φάκελο. Το όνομα του αρχείου είναι: *output_file_B*.

Γ. Earth Mover's Distance

Αρχικά θέλαμε να επαληθεύσουμε αν το άθροισμα της συνολικής φωτεινότητας κάθε εικόνας παραμένει σταθερό. Όμως, όπως φαίνεται στην παρακάτω εικόνα δεν ισχύει κάτι τέτοιο:

```
The brightness for 1 image is:     27525.0
The brightness for 2 image is:     31095.0
The brightness for 3 image is:     19443.0
The brightness for 4 image is:     17135.0
The brightness for 5 image is:     23214.0
The brightness for 6 image is:     29601.0
The brightness for 7 image is:     17646.0
The brightness for 8 image is:     35867.0
The brightness for 9 image is:     10874.0
The brightness for 10 image is:    21904.0
The brightness for 11 image is:    28548.0
The brightness for 12 image is:    14250.0
The brightness for 13 image is:    35761.0
The brightness for 14 image is:    28443.0
The brightness for 15 image is:    11622.0
The brightness for 16 image is:    25296.0
The brightness for 17 image is:    24635.0
The brightness for 18 image is:    27106.0
The brightness for 19 image is:    13589.0
The brightness for 20 image is:    17738.0
```

Για αυτό κάθε βάρος/signature κάθε cluster το διαιρέσαμε με το συνολικό άθροισμα των βαρών των cluster της συγκεκριμένης εικόνας. Έτσι, πλέον όλες οι εικόνες έχουν το ίδιο άθροισμα φωτεινότητας, ίσο με 1.

Για την επίλυση γραμμικού προβλήματος βελτιστοποίησης, με στόχο την εύρεση της EMD απόστασης μεταξύ δύο εικόνων, χρησιμοποιήσαμε τα *or tools* που παρέχονται από την Google για Python.

Πλήθος υποεικόνων: 16, μέγεθος: 7 x 7

Αποτελέσματα πειραμάτων:

Query set images	*Dataset images*	*Χρόνος(λεπτά)*	*Ορθότητα*
20	100	**0.4**	***39.5%***
20	2.000	**9**	***69%***
20	4.000	**18**	***69%***
100	100	**1.9**	**37.4%**
100	1.000	**22**	**60.3%**
1.000	100	**22**	**34.2%**

Συμπεράσματα: Είναι προφανές από τα παραπάνω αποτελέσματα πως η αύξηση της ακριβείας εξαρτάται από το πόσες εικόνες εξετάζουμε ως πιθανούς γείτονες. Επειδή, τα πειράματα ήταν ιδιαίτερα χρονοβόρα και ήταν ξεκάθαρο από τα αρχή πως για να αυξήσουμε την ορθότητα θα πρέπει να επιλέξουμε ένα μεγάλο πλήθος εικόνων από το dataset, κρατήσαμε ένα αρκετά χαμηλό πλήθος εικόνων από το query set. Για μεγάλο πλήθος εικόνων από το dataset η ορθότητα είναι ικανοποιητική, όμως ο χρόνος θεωρείται απαγορευτικός καθώς για 1.000 εικόνες φτάνει τα 22 λεπτά.

Κάτι ενδιαφέρον είναι το γεγονός πως ενώ κρατήσαμε σταθερό το πλήθος των queries σε 20 και αυξήσαμε τους πιθανούς γείτονες από 2.000 σε 4.000 η ορθότητα παρέμεινε σταθερή (και προφανώς ο χρόνος διπλασιάστηκε).

Ο λόγος που δεν περιμέναμε μεγαλύτερα ποσοστά ορθότητας είναι επειδή ο αριθμός των clusters στον οποίο «σπάσαμε» την εικόνα μας θεωρείται σχετικά μικρός για το μέγεθος της. Όσο πιο μεγάλο το πλήθος των υποεικόνων τόσες περισσότερες και οι «μεταφορές μάζας» μεταξύ προμηθευτή και καταναλωτή. Αυτό σημαίνει τόσο πιο ακριβής θα είναι η μεταφορά από cluster σε cluster οπότε τόσο πιο ακριβής θα είναι και η εκτίμηση της EMD απόστασης.

Τα παρακάτω πειράματα έγιναν με σκοπό τη σύγκριση τους με τα πειράματα για 49 clusters, για τα οποία λόγω του ότι παίρνουν πάρα πολύ χρόνο δεν πειραματιστήκαμε με μεγαλύτερες παραμέτρους.

Query set images	*Dataset images*	*Χρόνος(λεπτά)*	*Ορθότητα*
10	100	**0.22**	**41%**
10	200	**0.36**	**50%**
10	500	**1.1**	**57%**
10	1.000	**2.2**	**59%**
10	3.000	**6.8**	**59%**
20	100	**0.38**	**39.5%**
100	20	**0.4**	**15.7%**

Πλήθος υποεικόνων: 49, μέγεθος: 4 x 4

Αποτελέσματα πειραμάτων:

Query set images	*Dataset images*	*Χρόνος(λεπτά)*	*Ορθότητα*
10	100	**2.1**	**49%**
10	200	**4.41**	**61%**
10	500	**10.3**	**81%**
10	1.000	**20.8**	**83%**
10	3.000	**61**	**89%**
20	100	**5.1**	**50.5%**
100	20	**4.1**	**18.5%**

Συμπεράσματα: Για τις ίδιες παραμέτρους, με τα πειράματα των 16 clusters, τα νέα πειράματα έχουν σημαντικά μεγαλύτερα ποσοστά ορθότητας με το χρόνο όμως να έχει σχεδόν δεκαπλασιαστεί το οποίο παίζει τεράστιο ρόλο στα πειράματα μας και επομένως στο αν συμφέρει κάποιος να επιλέξει την EMD μετρική για τέτοιου είδους προβλήματα. Η ορθότητα βελτιώθηκε σημαντικά και για μόλις 10 εικόνες του query set έφτασε το 89%. Για να φτάσει όμως σε ένα τέτοιο ποσοστό χρειάστηκε μια ώρα οπότε το μεγάλο ποσοστό επιτυχίας δεν είναι αρκετά ενθαρρυντικό.

Εξαντλητική αναζήτηση στον αρχικό χώρο με μετρική Manhattan

Τα πρώτα πειράματα έγινα κυρίως για λόγους σύγκρισης με τις προηγούμενες μετρικές, για αυτό χρησιμοποιήσαμε μικρές παραμέτρους.

Query set images	*Dataset images*	*Χρόνος(λεπτά)*	*Ορθότητα*
10	100	**0.01**	**44%**
10	200	**0.03**	**48%**
10	500	**0.09**	**56%**
10	1.000	**0.18**	**60%**
10	3.000	**0.5**	**66%**
20	100	**0.03**	**43%**
100	20	**0.04**	**17%**

Query set images	*Dataset images*	*Χρόνος(λεπτά)*	*Ορθότητα*
20	100	**0.03**	***43%***
20	2.000	**0.7**	***66%***
20	4.000	**1.4**	***74.5%***
20	10.000	**3.5**	***76.5%***
20	20.000	**7**	***79%***
100	100	**0.2**	**37.3%**
100	1.000	**1.7**	**59.3%**
100	10.000	**17.5**	**69.7%**
1.000	100	**1.8**	**32.5%**
1.000	1.000	**14**	**51.7%**

Συμπεράσματα: Για την εξαντλητική αναζήτηση στον αρχικό χώρο με βάση τη μετρική Manhattan τα αποτελέσματα είναι πολύ πιο ενθαρρυντικά. Ο χρόνος σε σχέση με την EMD των 49 clusters είναι 100 φορές μικρότερος οπότε μπορούμε να χρησιμοποιήσουμε μεγαλύτερο πλήθος εικόνων και από τα δύο datasets προκειμένου να αυξήσουμε επιπλέον την ορθότητα.

Σύγκριση αποτελεσμάτων διαφορετικών μετρικών

Query set	*Data set*	*Ορθότητα EMD 16*	*Ορθότητα EMD 49*	*Ορθότητα Manhattan*
10	100	**41%**	**49%**	**44%**
10	200	**50%**	**61%**	**48%**
10	500	**57%**	**81%**	**56%**
10	1.000	**59%**	**83%**	**60%**
10	3.000	**59%**	**89%**	**66%**
20	100	**39.5%**	**50.5%**	**43%**
100	20	**15.7%**	**18.5%**	**17%**

Με μια πρώτη γρήγορη ματιά κάποιος μπορεί να διαπιστώσει πως η EMD μετρική με 49 clusters δίνει την καλύτερη ορθότητά σε όλες τις περιπτώσεις που αναγράφονται. Αυτό όμως που δεν φαίνεται στον παραπάνω πίνακα είναι οι χρόνοι εκτέλεσης. Ο χρόνος εκτέλεσης της EMD-49 είναι 100 φορές μεγαλύτερος από αυτόν της μετρικής Manhattan και 10 φορές μεγαλύτερος από την EMD-16. Συμπερασματικά ο χρόνος που απαιτείται για να επιτευχθεί μια απόδοση της τάξης του 90% που είναι ένα πολύ καλό ποσοστό αγγίζει την μία ώρα το οποίο θεωρούμε απαγορευτικό, ειδικά από τη στιγμή που ψάχναμε τους γείτονες μόνο 10 εικόνων του query set.

Όσον αφορά τη σύγκριση των μεθόδων EMD-16 και Manhattan στον αρχικό χώρο είναι εύκολο κάποιος να επιλέξει. Η ορθότητα της Manhattan είναι λίγο καλύτερη και ο χρόνος εκτέλεσής της υποδεκαπλάσιος.

Αν για το παρόν πρόβλημα έπρεπε να επιλέξουμε μια από τις 2 μετρικές θα επιλέγαμε την εξαντλητική αναζήτηση στον αρχικό χώρο.

Αρχεία-Εκτέλεση:

- Μπορεί να χρειαστεί να εκτελεστεί η εντολή: **pip install ortools**
- Το πρόγραμμα εκτελείται με τα flags που αναφέρονται στην εκφώνηση με τη μόνη διαφορά πως αντί για ./search εκτελείται με την εντολή **python serach.py**
- Το αρχείο search.py βρίσκεται στο φάκελο με όνομα: C

Σημείωση: Όλα τα παραπάνω πειράματα για λόγους ταχύτητας καθώς και για να έχει νόημα η σύγκριση των χρόνων μεταξύ διαφορετικών μετρικών έγιναν στο *Google Colab*.

Δ. Συσταδοποίηση

Σ1 → συσταδοποίηση k-medians των εικόνων στο νέο χώρο(των 10 διαστάσεων)

Σ2→ συσταδοποίηση στον αρχικό χώρο

Σ3→ συσταδοποίηση μετά από κατηγοριοποίηση των εικόνων από τον classifier

Τα αποτελέσματα παρακάτω προέκυψαν μετά από συσταδοποίηση, σε 10 clusters των 60.000 εικόνων του dataset.

NEW SPACE

clustering_time: 0.104849

Silhouette: [-0.0544748, 0.0292119, -0.0102514, 0.0334528, 0.0433902, 0.0606233, 0.0234669, 0.0614837, 0.153236, -0.0640284, 0.0379951]

Value of Objective Function: 1,303,489,727

ORIGINAL SPACE

clustering_time: 6063.75

Silhouette: [0.0190897, 0.196904, 0.0682216, 0.0262252, 0.117014, 0.0496573, 0.0322323, 0.041053, 0.0870733, 0.131438, 0.0832784]

Value of Objective Function: 1,130,223,863

CLASSES AS CLUSTERS

Silhouette: [0.0962391, 0.472507, -0.0423962, 0.0400389, 0.0168539, -0.0577548, 0.0934797, 0.0732188, -0.000142955, 0.0404348, 0.0796033]

Value of Objective Function: 1,222,559,462

Αρχικά, οι τιμές για το Silhouette και η τιμή της objective function όσον αφορά τη συσταδοποίηση στον Σ1 και στον Σ2 είχε κάποιες μικρές διακυμάνσεις ανάμεσα σε διαφορετικές εκτελέσεις με τις ίδιες παραμέτρους, όπως το σύνολο των εικόνων και το πλήθος των clusters. Το οποίο βέβαια είναι λογικό και το οποίο είχαμε παρατηρήσει και στην 1η εργασία καθώς υπάρχει ο παράγοντας της τυχαιότητας στο ποια θα είναι τα αρχικά κέντρα των clusters από τα οποία εξαρτάται το ποιες εικόνες θα ενταχθούν σε αυτό το cluster. Στον χώρο Σ3 δεν παρατηρήσαμε κάτι παρόμοιο αφού δίναμε στο πρόγραμμα την ίδια κατηγοριοποίηση των εικόνων σε κάθε εκτέλεση. Να σημειώσουμε πως η κατηγοριοποίηση αυτή είχε accuracy = 99%.

Διακύμανση Silhouette στους τρεις χώρους:

- Σ1 (new space): 0.02-0.06
- Σ2 (original space): 0.07-0.1
- Σ3 (after classification): 0.08-0.1

Στον νέο χώρο, δηλαδή κατά τη συσταδοποίηση των εικόνων σε μόλις 10 διαστάσεις, η τιμή του Silhouette είναι η χαμηλότερη, δηλαδή γίνεται η χειρότερη συσταδοποίηση σε σχέση με τους άλλους 2 χώρους. Επίσης, στα περισσότερα πειράματα που πραγματοποιήσαμε η objective function ήταν μεγαλύτερη. Αυτό δικαιολογείται εύκολα από το γεγονός πως όσο χειρότερο είναι το clustering, δηλαδή όσο πιο χαμηλό είναι το Silhouette, τόσο πιο μακρινές μεταξύ τους θα είναι οι εικόνες του ίδιου cluster οπότε η objective function θα αυξάνεται.

Ο νέος χώρος, παρόλο που μας παρέχει μια υπερσυμπυκνωμένη πληροφορία για κάθε εικόνα και έχει γίνει μια καλή εκπαίδευση του autoencoder φαίνεται πως δεν αρκεί για μια πολύ καλή συσταδοποίηση. Βέβαια, αναλογιζόμενοι το χρόνο που χρειάστηκε θα ήταν μια καλή λύση σε ένα πρόβλημα που απαιτεί κυρίως ταχύτητα. Αυτό αποτυπώνεται στα αποτελέσματα μας καθώς το clustering στον Σ1 μόλις 0.1 δευτερόλεπτα ενώ ο Σ2 6063.75 (!) δευτερόπετα.

Στην πλειονότητα των πειραμάτων μας παρατηρήσαμε ότι τα Silhouette στον Σ2 και στον Σ3 είναι πολύ κοντά παρόλο που στον Σ3 έχει γίνει ένα πολύ καλό classification. Αυτό το γεγονός πιθανώς οφείλεται στο ότι για τον υπολογισμό του Silhouette βασιζόμαστε στην απόσταση μιας εικόνας από τις εικόνες στον ίδιο cluster με αυτή καθώς και την απόσταση της από τις εικόνες του αμέσως κοντινότερου cluster. Η κατηγοριοποίηση των εικόνων στον Σ3 δεν έγινε με βάση τις αποστάσεις τους, όπως στον Σ2, αλλά με βάση τα labels τους. Έτσι καταφέραμε να πετύχουμε μια εξαιρετική ακρίβεια. Όμως, μια εικόνα έστω με label 5, η οποία μοιάζει αρκετά με τον αριθμό 8 και η οποία είναι κακή και ακόμα και με το μάτι είναι δύσκολο να διακρίνει κάποιος ποιον αριθμό αναπαριστά, κατά τη συσταδοποίηση στον Σ3 θα μπει σε ένα cluster με

κεντροειδές μια εικόνα με αριθμό 5. Αυτές οι 2 εικόνες είναι αναμενόμενο να έχουν μεγάλη απόσταση μεταξύ τους και για αυτό το Silhouette στον Σ3 να μην είναι καλύτερο από αυτό στον Σ2 όπως και περιμέναμε. Επιπλέον, παρατηρήσαμε πως στις περισσότερες περιπτώσεις η objective function του Σ3 ήταν λίγο μεγαλύτερη σε σχέση με αυτή του Σ2.

Θεωρούμε πως με κριτήριο το Silhouette και την objective function ο ιδανικότερος χώρος για συσταδοποίηση των εικόνων της βιβλιοθήκης MNIST είναι ο αρχικός χώρος Σ2.

Ένα τελευταίο συμπέρασμα είναι πως αν επιθυμούμε να κατηγοριοποιήσουμε τις εικόνες ενός συνόλου η κατηγοριοποίηση τους, όπως έγινε στην 2η εργασία είναι πολύ πιο αποτελεσματική και γρήγορη από ότι η συσταδοποίηση τους με στόχο το classification.

Αρχεία-Εκτέλεση:

- Εντολή μεταγλώττισης: **make**
- Η εντολή εκτέλεσης είναι αυτή που αναγράφεται στην εκφώνηση
- input file new space: **output_dataset_file.txt**
- configuration file: **cluster.conf**
- classes from NN as clusters file: **classes**
- Όλα τα απαραίτητα αρχεία βρίσκονται στον φάκελο με όνομα: D