

## Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα 2020

### Εργασία 3

**Κανέλλης Παντελεήμων 1115201600055**

**Μπόννη Τηλέμαχος 1115201300118**

Github link: [Project2020\\_HW3](#)

Google Colab links: [reduce](#), [EMD](#)

**ΣΗΜΑΝΤΙΚΟ:** Χρησιμοποιήθηκε το εργαλείο **Google Colab** για την παραγωγή των αποτελεσμάτων, για λόγους ταχύτητας και μόνο. Ο κώδικας που παραδώσαμε στο Github έχει την ίδια λειτουργικότητα με τον κώδικα του colab, με την μόνη διαφορά ότι το colab δεν δέχεται command line arguments, οπότε τα paths είναι hardcoded στον κώδικα.

### Γενική Περιγραφή

Διανυσματική αναπαράσταση εικόνα σε χώρο χαμηλότερης διάστασης με χρήση νευρωνικού δικτύου αυτοκωδικοποίησης. Αναζήτηση και συσταδοποίηση των εικόνων στο νέο χώρο και σύγκριση με προσεγγιστική και εξαντλητική αναζήτηση και συσταδοποίηση στον αρχικό χώρο (διάστασης 784).

### Κατάλογος αρχείων κώδικα και περιγραφή

Τα αρχεία κώδικα παραδίδονται όλα κάτω από ένα κοινό directory και για κάθε ερώτημα γίνεται η χρήση των αντίστοιχων εντολών όπως περιγράφονται στο τέλος του README.

Για την παραγωγή των εκτελέσιμων των ερωτημάτων Β και Δ γίνεται η χρήση του Makefile και της εντολής make στο command line.

## Μέρος A - Reduce

Έγινε η απαιτούμενη προσθήκη των Flatten και Dense στρωμάτων στον Encoder και ενός Dense και Reshape στον Decoder για να καθίσταται δυνατή η δημιουργία του latent vector.

Πραγματοποιήθηκαν πολλαπλά πειράματα για την εύρεση του κατάλληλου μοντέλου για τον autoencoder για το οποίο δεν προέκυπτε overfitting ή underfitting.

Μετά την αναγνώριση του καλύτερου μοντέλου, αυτό αποθηκεύτηκε και σχεδιάστηκε η ροή των δεδομένων από το dataset και το queryset μέσα από τον encoder και η εξαγωγή του latent vector διάστασης 1x10 για κάθε εικόνα.

Αυτά αποθηκεύονται στα αρχεία out\_dataset.bin και out\_queryset.bin που αποτελούν τα αρχεία εισόδου για το δεύτερο μέρος αυτής της εργασίας.

### Layers:

Πειραματικά έγινε η επιλογή να υπάρχουν σταθερά 2 maxPooling layers και διαφορετικός αριθμός από Conv2D για τα οποία και κάναμε τις παρατηρήσεις. Παρατηρούμε ότι με την αύξηση των συνολικών layers γίνεται ταυτόχρονα αύξηση του χρόνου εκτέλεσης αλλά παράγονται καλύτερα μοντέλα.

Μετά από πολλαπλές δοκιμές προκύπτει ότι τα layers με την καλύτερη απόδοση είναι 6.

### Epoch:

Παρατηρούμε ότι για πολύ μικρό epoch (10 και κάτω) παρουσιάζεται underfitting καθώς δεν γίνεται επαρκής εκπαίδευση του νευρωνικού. Ενώ για περισσότερα από 60, ο χρόνος αυξάνεται δραματικά. Παρολαυτά αποφασίσαμε να κάνουμε μια εκτέλεση με 100 epochs και να την αποθηκεύσουμε για περαιτέρω χρήση.

### Batchsize:

Η καλύτερη επιλογή για batchsize σύμφωνα με τις δοκιμαστικές εκτελέσεις ήταν 512.

### **Τελικά αποτελέσματα και επιλογή τελικών βέλτιστων μοντέλων autoencoder:**

Τόσο σύμφωνα με τις παρατηρήσεις που προκύπτουν και που έχουν περιγραφεί παραπάνω, όσο και με την παρατήρηση των διαγραμμάτων που προέκυψαν από τις πειραματικές δοκιμές, παρατηρούμε ότι τα παρακάτω μοντέλα παρουσιάζουν τα καλύτερα αποτελέσματα με αυτό που προτείνεται για το reduce να είναι το **layers: 6, epoch: 100, batch size: 512**.

Η δομή του Autoencoder που επιλέχθηκε και παραδίδεται ως μοντέλο .h5 στην εργασία είναι η παρακάτω:

Layer (type)	Output Shape	Param #
input_16 (InputLayer)	[None, 28, 28, 1]	0
conv2d_101 (Conv2D)	(None, 28, 28, 32)	320
batch_normalization_87 (Batch Normalization)	(None, 28, 28, 32)	128
max_pooling2d_31 (MaxPooling)	(None, 14, 14, 32)	0
conv2d_102 (Conv2D)	(None, 14, 14, 64)	18496
batch_normalization_88 (Batch Normalization)	(None, 14, 14, 64)	256
max_pooling2d_32 (MaxPooling)	(None, 7, 7, 64)	0
conv2d_103 (Conv2D)	(None, 7, 7, 128)	73856
batch_normalization_89 (Batch Normalization)	(None, 7, 7, 128)	512
flatten_15 (Flatten)	(None, 6272)	0
dense_30 (Dense)	(None, 10)	62730
dense_31 (Dense)	(None, 6272)	68992
reshape_15 (Reshape)	(None, 7, 7, 128)	0
conv2d_104 (Conv2D)	(None, 7, 7, 128)	147584
batch_normalization_90 (Batch Normalization)	(None, 7, 7, 128)	512
up_sampling2d_29 (UpSampling)	(None, 14, 14, 128)	0
conv2d_105 (Conv2D)	(None, 14, 14, 64)	73792
batch_normalization_91 (Batch Normalization)	(None, 14, 14, 64)	256
up_sampling2d_30 (UpSampling)	(None, 28, 28, 64)	0
conv2d_106 (Conv2D)	(None, 28, 28, 32)	18464
batch_normalization_92 (Batch Normalization)	(None, 28, 28, 32)	128
conv2d_107 (Conv2D)	(None, 28, 28, 1)	289

## Μέρος Β - Search

Η βασική υλοποίηση γίνεται στο lsh.cpp που πρακτικά είναι η επέκταση του παραδοτέου της εργασίας 1. Όλα τα απαραίτητα αρχεία από την εργασία 1 έχουν μεταφερθεί και ξανα παραδίδονται με τις κατάλληλες προσαρμογές που απαιτούνταν για την ανάγνωση των out\_dataset.bin και out\_queryset.bin που περιέχουν δεδομένα σε unsigned short int με little endian. Έγινε σχεδιαστικά η επιλογή για little endian για αυτά τα αρχεία δεδομένων καθώς αυτό εξυπηρέτούσε την ευκολότερη υλοποίηση με τις λιγότερες αλλαγές κώδικα.

Το lsh.cpp καλεί τις συναρτήσεις απαραίτητες για την εύρεση του κοντινότερου γείτονα με LSH και στη συνέχεια ανοίγει και διαβάζει μέσω κατάλληλων συναρτήσεων τα αποτελέσματα από δυο αρχεία .txt (exact\_results\_original και exact\_results\_reduced) τα οποία περιέχουν τα

αποτελέσματα για το Exact τόσο στον αρχικό χώρο όσο και στον νεο. Η σχεδιαστική αυτή επιλογή έγινε καθώς ο χρόνος εκτέλεσης του exact ήταν υπερβολικά μεγάλος.

Αν απαιτείται τότε με κατάλληλη προσαρμογή των αρχείων εισόδου και εξόδου μπορούν να παραχθούν εκ νέου τα μέσω των εκτελέσιμων αρχείων exact\_nn\_original και exact\_nn\_reduced.

Όλα τα αποτελέσματα παρουσιάζονται στο results\_partB.txt που συμπεριλαμβάνεται στα παραδοτέα της εργασίας.

Τα αποτελέσματα από την εκτέλεση δείχνουν ότι το Reduced είναι σαφώς πιο αποτελεσματικό σε σχέση με το LSH ενώ ταυτόχρονα ο χρόνος εκτέλεσης είναι κατά μικρή μονάδα παραπάνω, εμφανώς μικρότερος από τον True.

- ➔ tReduced: 1131951034
- ➔ tLSH: 86746949
- ➔ tTrue: 4965924233
- ➔ Approximation Factor for Reduced: 1.18201
- ➔ Approximation Factor for LSH: 1.4261

### Μέρος Γ - Earth Mover's Distance

Εδώ υλοποιείται ο αλγόριθμος Earth Mover's Distance (EMD), ακριβώς όπως τις διαφάνειες. Η υλοποίηση έγινε με χρήση **Python**. Για την παραγωγή των signatures, η αρχική εικόνα 28x28 διαστάσεων χωρίζεται σε υπο-εικόνες (clusters) των AxA, όπου A διαιρετής του 28 και > 1 (δηλαδή έχουμε clusters των 2x2, 4x4, 7x7, 14x14). Ο χρήστης επιλέγει αναγκαστικά μία από αυτές τις διαμερίσεις. Στην συνέχεια υπολογίζονται οι αποστάσεις από κάθε cluster των queries σε κάθε cluster από τα inputs μαζί με τα βάρη. Φτιάχνονται οι πίνακες περιορισμών και λύνεται το πρόβλημα γραμμικού προγραμματισμού με την χρήση της scipy.

Για να γίνουν περισσότερο κατανοητοί οι πίνακες των περιορισμών, θα χρησιμοποιηθεί ένα απλό παράδειγμα με n = 4, άρα θα έχουμε συνολικά 16 μεταβλητές. Το πρόβλημα για n = 4 εκφράζεται μαθηματικά ως εξής:

$$\min \sum_{i=1}^4 \sum_{j=1}^4 d_{ij} * f_{ij} \quad , \text{ με περιορισμούς:}$$

$f_{ij} \geq 0$  , δηλαδή

$$f_{11}, f_{12}, f_{13}, f_{14}, f_{21}, f_{22}, f_{23}, f_{24}, f_{31}, f_{32}, f_{33}, f_{34}, f_{41}, f_{42}, f_{43}, f_{44} \geq 0$$

$$\sum_{i=1}^4 f_{ij} = w_i, 1 \leq i \leq 4 \quad \text{Δηλαδή:}$$

$$f_{i1} + f_{i2} + f_{i3} + f_{i4} = w_i, 1 \leq i \leq 4$$

$$i=1: f_{11} + f_{12} + f_{13} + f_{14} = w_1$$

$$i=2: f_{21} + f_{22} + f_{23} + f_{24} = w_2$$

$$i=3: f_{31} + f_{32} + f_{33} + f_{34} = w_3$$

$$i=4: f_{41} + f_{42} + f_{43} + f_{44} = w_4$$

Η εξίσωση θα πρέπει να γραφτεί σαν ένας πίνακας με 16 τιμές (αφού έχουμε 16 μεταβλητές για  $n = 4$ ). Κάθε αριθμός δηλώνει τον συντελεστή που πολλαπλασιάζεται η μεταβλητή. Πχ αν είχαμε 2 μεταβλητές με εξίσωση  $3x + 4y = 7$ , ο πίνακας θα ήταν  $[3,4]$ . Κάθε εξίσωση αποτελεί διαφορετικό πίνακα. Σε αυτό το παράδειγμα, ο πρώτος περιορισμός που αναπτύχθηκε παραπάνω έχει ως αποτέλεσμα τους παρακάτω πίνακες:

$$[1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0]$$

$$[0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0]$$

$$[0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0]$$

$$[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1]$$

Ο δεύτερος περιορισμός αναπτύσσεται στις εξισώσεις:

$$\sum_{j=1}^4 f_{ij} = w'_j, 1 \leq j \leq 4 \quad \text{Δηλαδή:}$$

$$f_{1j} + f_{2j} + f_{3j} + f_{4j} = w'_j, 1 \leq j \leq 4$$

$$j=1: f_{11} + f_{21} + f_{31} + f_{41} = w'_1$$

$$j=2: f_{12} + f_{22} + f_{32} + f_{42} = w'_2$$

$$j=3: f_{13} + f_{23} + f_{33} + f_{43} = w'_3$$

$$j=4: f_{14} + f_{24} + f_{34} + f_{44} = w'_4$$

Οι παραπάνω περιορισμοί σε μορφή πινάκων είναι ως εξής:

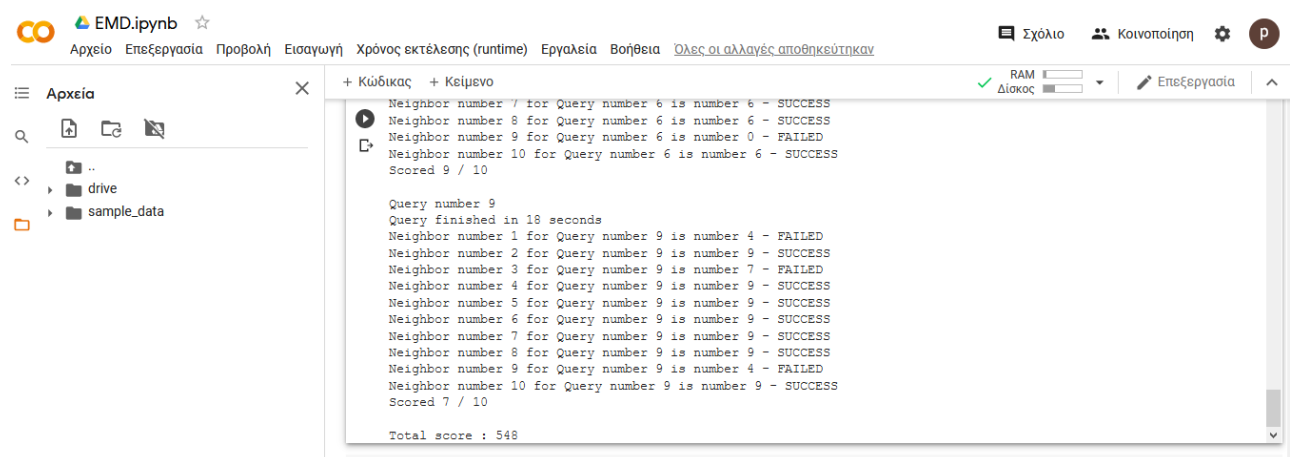
```
[1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0]
[0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0]
[0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0]
[0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1]
```

Ο πίνακας των βαρών θα είναι:

```
[w1,w2,w3,w4,w'1,w'2,w'3,w'4]
```

Ο συνένωση των πινάκων περιορισμών δίνει τον πίνακα Aeq, ενώ ο πίνακας των βαρών ονομάζεται beq. Έτσι δημιουργείται η ισότητα  $Aeq = beq$  και δίνεται στην scipy για επίλυση (ο πίνακας των αποστάσεων που θέλουμε να ελαχιστοποιήσουμε ονομάζεται c στην scipy).

Τα αποτελέσματα παρήχθησαν για 600 inputs και 100 queries, επειδή ο EMD είναι αρκετά αργός, ακόμα και για 7x7 cluster size. Είναι λογικό ότι είναι αργός, αφού για κάθε εικόνα έχει να λύσει  $4 \times 4 = 16$  προβλήματα γραμμικού προγραμματισμού. Άρα για 100 queries λύνει 1600 τέτοια προβλήματα. Για κάθε query βρέθηκαν οι πρώτοι 10 γείτονες όπως ζητήθηκε. Τα αποτελέσματα ήταν στο όριο, με ρεκόρ επιτυχίας **54,8%** (548 / 1000). Το παρακάτω στιγμιότυπο που προήλθε από το google colab, επιβεβαιώνει αυτό το αποτέλεσμα:



```
EMD.ipynb
Αρχείο Επεξεργασία Προβολή Εισαγωγή Χρόνος εκτέλεσης (runtime) Εργασία Βοήθεια Όλες οι αλλαγές αποθηκεύτηκαν
Αpxεία
> drive
> sample_data

+ Κώδικας + Κείμενο
Neighbor number 7 for Query number 6 is number 6 - SUCCESS
Neighbor number 8 for Query number 6 is number 6 - SUCCESS
Neighbor number 9 for Query number 6 is number 0 - FAILED
Neighbor number 10 for Query number 6 is number 6 - SUCCESS
Scored 9 / 10

Query number 9
Query finished in 18 seconds
Neighbor number 1 for Query number 9 is number 4 - FAILED
Neighbor number 2 for Query number 9 is number 9 - SUCCESS
Neighbor number 3 for Query number 9 is number 7 - FAILED
Neighbor number 4 for Query number 9 is number 9 - SUCCESS
Neighbor number 5 for Query number 9 is number 9 - SUCCESS
Neighbor number 6 for Query number 9 is number 9 - SUCCESS
Neighbor number 7 for Query number 9 is number 9 - SUCCESS
Neighbor number 8 for Query number 9 is number 9 - SUCCESS
Neighbor number 9 for Query number 9 is number 4 - FAILED
Neighbor number 10 for Query number 9 is number 9 - SUCCESS
Scored 7 / 10

Total score : 548
```

Δοκιμάσαμε και τον αλγόριθμο Exact Neighbor για τις 10 διαστάσεις (επίσης για 100 queries και 600 inputs για να είναι δίκαιη η σύγκριση). Το πρόγραμμα αναπτύχθηκε με C++ και παράξαμε ένα txt αρχείο όπου κάθε 10 γραμμές έχουν τους γείτονες για καθένα από τα 100 queries. Το txt αρχείο αναλύθηκε με Python και βγήκαν συμπεράσματα. Το ποσοστό επιτυχίας ήταν **71,1%**, πολύ καλύτερα από το EMD. Η παρακάτω εικόνα επιβεβαιώνει αυτό το αποτέλεσμα: (Να σημειωθεί ότι η εξαντλητική αναζήτηση στις 10 διαστάσεις βγάζει καλύτερα αποτελέσματα για 60,000 inputs με ποσοστό 94,9%, απλά ο EMD θα ήταν απαγορευτικά αργός για 60,000 inputs και πρέπει το μέτρο σύγκρισης να είναι το ίδιο).

## Μέρος Δ - Clustering

Έγινε η συσταδοποίηση των εικόνων στις 784 και στις 10 διαστάσεις, με την χρήση του παραδοτέου της 1ης εργασίας. Στην συνέχεια, παράχθηκε το classification αρχείο από το παραδοτέο της 2ης εργασίας και χρησιμοποιήθηκε για την παραγωγή της αντικειμενικής συνάρτησης και Silhouette.

Παρακάτω δίνονται τα αποτελέσματα της objective function για κάθε cluster για τον αρχικό χώρο (784 διαστάσεις):

### ORIGINAL SPACE

CLUSTER-1 value of Objective Function: 2.60066e+07  
CLUSTER-2 value of Objective Function: 2.71525e+07  
CLUSTER-3 value of Objective Function: 1.12521e+07  
CLUSTER-4 value of Objective Function: 9.15228e+06  
CLUSTER-5 value of Objective Function: 3.31829e+07  
CLUSTER-6 value of Objective Function: 2.96715e+07  
CLUSTER-7 value of Objective Function: 1.40276e+07  
CLUSTER-8 value of Objective Function: 2.28763e+07  
CLUSTER-9 value of Objective Function: 3.61477e+07  
CLUSTER-10 value of Objective Function: 1.85872e+07

Silhouette Total Average: -0.102386

Να σημειωθεί ότι οι τιμές της αντικειμενικής συνάρτησης διαφέρουν ανά εκτέλεση, επειδή η αρχικοποίηση επιλέγει τυχαία τα κεντροειδή της κατανομής, σύμφωνα με τον αλγόριθμο αρχικοποίησης kmeans++. Χρησιμοποιήθηκε  $k = 10$ , και ο αλγόριθμος Loyd's. Εδώ φαίνεται ότι οι εικόνες δεν είναι πολύ καλά κατανεμημένες στα 10 clusters, επειδή οι τιμές της αντικειμενικής συνάρτησης διαφέρουν αρκετά μεταξύ των clusters. Επίσης η χαμηλή τιμή silhouette δηλώνει κακό clustering.

Παρακάτω δίνονται τα αποτελέσματα της objective function για κάθε cluster για τον νέο χώρο (10 διαστάσεις):

#### **NEW SPACE**

CLUSTER-1 value of Objective Function: 2.43208e+07  
CLUSTER-2 value of Objective Function: 1.8778e+07  
CLUSTER-3 value of Objective Function: 1.26166e+07  
CLUSTER-4 value of Objective Function: 1.69328e+07  
CLUSTER-5 value of Objective Function: 1.96182e+07  
CLUSTER-6 value of Objective Function: 2.67221e+07  
CLUSTER-7 value of Objective Function: 1.00813e+07  
CLUSTER-8 value of Objective Function: 2.92981e+07  
CLUSTER-9 value of Objective Function: 1.71893e+07  
CLUSTER-10 value of Objective Function: 2.7425e+07

Silhouette Total Average: -0.13536

Αν και η οι τιμές της objective function είναι περισσότερο ομοιόμορφα κατανεμημένες στα clusters, η χαμηλή τιμή silhouette δείχνει πάλι κακό clustering.

Όσο αναφορά το αρχείο εξόδου της 2ης εργασίας, από κάθε cluster επιλέχθηκε ως κεντροειδές το σημείο που ανήκει στο cluster και απέχει την μικρότερη απόσταση από κάθε άλλο σημείο του ίδιου cluster. Παρακάτω δίνονται οι τιμές της objective function για κάθε cluster και η συνολική τιμή silhouette. (Να σημειωθεί ότι αυτά τα αποτελέσματα δεν είναι η τελική έξοδος. Τα αρχεία εξόδου περιέχουν αναλυτικές πληροφορίες για τα κεντροειδή και την silhouette για κάθε σημείο, απλά εδώ παρουσιάζουμε τις πιο χρήσιμες πληροφορίες).

#### **CLASSES AS CLUSTERS**

CLUSTER-1 value of Objective Function: 2.57657e+07  
CLUSTER-2 value of Objective Function: 1.30799e+07  
CLUSTER-3 value of Objective Function: 2.77968e+07  
CLUSTER-4 value of Objective Function: 2.37927e+07  
CLUSTER-5 value of Objective Function: 2.10789e+07  
CLUSTER-6 value of Objective Function: 2.27158e+07  
CLUSTER-7 value of Objective Function: 2.23418e+07  
CLUSTER-8 value of Objective Function: 1.90764e+07  
CLUSTER-9 value of Objective Function: 2.37796e+07  
CLUSTER-10 value of Objective Function: 2.05184e+07

Silhouette Total Average: 185.7

Εδώ βλέπουμε μια πολύ καλή κατανομή της αντικειμενικής συνάρτησης στα 10 clusters. Η μεγάλη τιμή της Silhouette δηλώνει μια πολύ καλή συσταδοποίηση, κάτι το οποίο είναι



αναμενόμενο, αφού το νευρωνικό δίκτυο της 2ης εργασίας σκόραρε ποσοστό αναγνώρισης 98,7%.

## Οδηγίες Χρήσης

Προτείνεται η χρήση των συνδέσμων στο colab λόγω της δυνατότητας χρήσης GPU που αυξάνει κατά πολύ την ταχύτητα εκτέλεσης.

Για να μπορέσει να γίνει το διάβασμα των αρχείων και η αποθήκευση των μοντέλων και των γραφικών παραστάσεων θα πρέπει να γίνει mount το google drive του χρήστη.

Τα παραδοτέα μέσω eclass ακολουθούν τον τρόπο εκτέλεσης που ζητάται από την εκφώνηση.

Για το **μέρος Α** της εργασίας δίνεται η εντολή από το command line:

```
$python3 reduce.py -d <dataset> -q <queryset> -od <output_dataset_file> -oq <output_query_file>
```

Δίνεται η επιλογή στον χρήστη να κάνει επανεκπαίδευση του μοντέλου αν θέλει ή να προχωρήσει με το default που έχει επιλεγεί ήδη από εμάς.

Η παραγωγή των αποτελεσμάτων αν δεν δοθούν arguments στο command line γίνεται σε default τοποθεσία με όνομα out\_dataset.bin και out\_queryset.bin

Για το **μέρος Β** της εργασίας δίνεται η εντολή από το command line:

```
./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>
```

Το εκτελέσιμο μπορεί να τρέξει με default παραμέτρους και μονοπάτια αρχείων με την εντολή

./search

Μαζί με τα αρχεία κώδικα και το makefile και τα αρχεία για τα exact nearest neighbors στον original (784) χώρο και τον reduced (10) που έχει ήδη τρέξει στο δικό μας μηχάνημα (2,5 ώρες περίπου), για την παραγωγή των αποτελεσμάτων.

Αν δεν δοθεί μονοπάτι για το output τότε αυτό δημιουργείται by default και αποθηκεύει τα αποτελέσματα ακριβώς στην μορφή που ζητείται στην εκφώνηση.

Για το **μέρος Γ** της εργασίας δίνεται η εντολή από το command line:

```
$python3 emd -d <input file original space> -q <query file original space> -l1 <labels of input dataset> -l2 <labels of query dataset> -o <output file>
```

Αποφασίσαμε να παράξουμε δύο εκτελέσιμα αντί να τα βάλουμε όλα στο ./search, οπότε η εντολή -EMD παραλείπεται.

Για το **μέρος Δ** της εργασίας δίνεται η εντολή από το command line:

```
$/cluster -d <input file original space> -i <input file new space>
```

```
-n <classes from NN as clusters file> -c <configuration file> -o <output file>
```