Εργασία 3

– Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα –

Τίτλος: «Neural Networks and clustering of images»

Διανέλλος Στέφανος – 1115201500036
Χιόνης Κουφάκος Αριστοφάνης – 1115201500177

Για οδηγίες μεταγλώττισης και εκτέλεσης των προγραμμάτων παρακαλώ δείτε το README.md στο root directory. Μέσα στον φάκελο src/ θα βρείτε 4 φακέλους a, b, c, d που περιέχουν τον κώδικα για το κάθε ερώτημα αντίστοιχα: output files και πειράματα.

Ακολουθούν πειράματα και παρατηρήσεις:
Α)
Αρχικά πειραματιστήκαμε με διαφορετικές υπερ-παραμέτρους και διαφορετικά μοντέλα, γύρω από το μοντέλο που μας δόθηκε μέσω του e-class αλλά και του μοντέλου που αναπτύξαμε στην $2^η$ εργασία. Παρατηρήσαμε ότι το loss ήταν πολύ μεγάλο, αλλά αυτό συμπίπτει με αυτό που περιμέναμε διότι από 784 διαστάσεις πάμε σε 10, οπότε λογικό είναι να υπάρχει κάποια απώλεια πληροφορίας. Βάσει των αποτελεσμάτων παρατηρήσαμε ότι το loss παραμένει σχετικά σταθερό και έτσι αποφασίσαμε να συνεχίσουμε με τη δομή που μας δόθηκε στο e-class. Προσπαθήσαμε να βάλουμε και Dropout layer για να μειώσουμε το overfitting, αλλά το loss δεν άλλαξε. Τον κώδικα τον τρέξαμε χρησιμοποιώντας το Google Colab και το αντίστοιχο notebook μπορεί να βρεθεί στο παραδοτέο στον φάκελο a.Autoencoder. Τα πειράματα μας για διάφορες τιμές των υπερπαραμέτρων:

Number of filters: 6
Filter size: (3,3)
Number of layers: 11
Number of epochs: 10
Batch size: 64
Latent Dimension: 10
Loss: 7234
Val_loss: 7204

Number of filters: 6
Filter size: (3,3)
Number of layers: 11
Number of epochs: 30
Batch size: 64
Latent Dimension: 10
Loss: 7210
Val_loss: 7205

Number of filters: 6
Filter size: (3,3)
Number of layers: 11

Number of epochs: 50 (Σταμάτησε στα 45 epochs λόγω του early stopping)
Batch size: 64
Latent Dimension: 10
Loss: 7224
Val_loss: 7206

Number of filters: 6
Filter size: (3,3)
Number of layers: 11
Number of epochs: 10
Batch size: 128
Latent Dimension: 10
Loss: 7201
Val_loss: 7204

Number of filters: 6
Filter size: (3,3)
Number of layers: 11
Number of epochs: 50 (Σταμάτησε στα 31 epochs λόγω του early stopping)
Batch size: 128
Latent Dimension: 10
Loss: 7237
Val_loss: 7204

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 10
Batch size: 128
Latent Dimension: 10
Loss: 7233
Val_loss: 7203

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 30
Batch size: 128
Latent Dimension: 10
Loss: 7226
Val_loss: 7203

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 50

Batch size: 128
Latent Dimension: 10
Loss: 7239
Val_loss: 7203

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 10
Batch size: 64
Latent Dimension: 10
Loss: 7214
Val_loss: 7203

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 30
Batch size: 64
Latent Dimension: 10
Loss: 7227
Val_loss: 7203

Number of filters: 8
Filter size: (3,3)
Number of layers: 11
Number of epochs: 50
Batch size: 64
Latent Dimension: 10
Loss: 7232
Val_loss: 7203

B)
Εδώ δεν είχαμε κάποια πειράματα να τρέξουμε, ακολουθήσαμε τις οδηγίες της εκφώνησης και εκτελέσαμε όλα όσα ζήταγε. Τα αποτελέσματα μπορούν να βρεθούν στον φάκελο b.LSH, ενδεικτικά για w=400, K=4, L=5:

tReduced: 4.00782
tLSH: 0.504088
tTrue: 168.074
Approximation Factor LSH: 2.0311
Approximation Factor Reduced: 1.30739

Παρατηρούμε ότι ο brute force αλγόριθμος στον νέο διανυσματικό χώρο δίνει καλύτερα αποτελέσματα απ'ότι ο LSH στον αρχικό χώρο αλλά είναι σημαντικά πιο αργός (~8 φορές).

Γ)
Παρακάτω παραθέτουμε μερικά πειράματα που εκτελέσαμε για το ερώτημα Γ.

- Πλήθος query images: 20
- Πλήθος input images: 200
- Πλήθος N γειτόνων: 10
  - Για τον EMD:
    Χρόνος: 913.85 seconds
    Average Correct Search Results EMD: 0.28
  - Για τον MANHATTAN:
    Χρόνος: 0.001 seconds
    Average Correct Search Results MANHATTAN: 0.565

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 1
  - Για τον EMD:
    Χρόνος: 260.44 seconds
    Average Correct Search Results EMD: 0.4
  - Για τον MANHATTAN:
    Χρόνος: 0.0002 seconds
    Average Correct Search Results MANHATTAN: 0.6

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 2
  - Για τον EMD:
    Χρόνος: 251.83 seconds
    Average Correct Search Results EMD: 0.35
  - Για τον MANHATTAN:
    Χρόνος: 0.0003 seconds
    Average Correct Search Results MANHATTAN: 0.65

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 5
  - Για τον EMD:
    Χρόνος: 218.67 seconds
    Average Correct Search Results EMD: 0.3
  - Για τον MANHATTAN:
    Χρόνος: 0.0002 seconds
    Average Correct Search Results MANHATTAN: 0.54

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 7
  - Για τον EMD:
    Χρόνος: 253.97 seconds
    Average Correct Search Results EMD: 0.25
  - Για τον MANHATTAN:
    Χρόνος: 0.0002 seconds
    Average Correct Search Results MANHATTAN: 0.5

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 10
  - Για τον EMD:
    Χρόνος: 230.22 seconds
    Average Correct Search Results EMD: 0.24
  - Για τον MANHATTAN:
    Χρόνος: 0.0005 seconds
    Average Correct Search Results MANHATTAN: 0.47

- Πλήθος query images: 10
- Πλήθος input images: 100
- Πλήθος N γειτόνων: 15
  - Για τον EMD:
    Χρόνος: 213.98 seconds
    Average Correct Search Results EMD: 0.20
  - Για τον MANHATTAN:
    Χρόνος: 0.0002 seconds
    Average Correct Search Results MANHATTAN: 0.39

Για την εκτέλεση του (Γ), αποφασίσαμε να γράψουμε python3 για τον EMD και C++ για τον Brute Force. Για να τρέξουμε το πρόγραμμα πρέπει να τρέξουμε το C++ πρόγραμμα και αυτό πριν τελειώσει εκτελεί με system τον EMD από το python αρχείο. Χρησιμοποιήσαμε το pywraplp από τα google ortools. Παρατηρήσαμε ότι ο EMD αργεί πάρα πολύ, και ο Brute force τρέχει πολύ πιο γρήγορα. Αρχίσαμε τα πειράματα με ένα πλήθος query images: 20, input images: 200 αλλά έκανε πολύ χρόνο για να τρέξει οπότε στα υπόλοιπα πειράματα δοκιμάσαμε λιγότερα query images. Επίσης, παρατηρήσαμε ότι πάντα ο Brute Force δίνει καλύτερα αποτελέσματα. Όσο λιγότεροι ήταν οι γείτονες που ψάχναμε τόσο καλύτερα αποτελέσματα έδιναν οι αλγόριθμοι. Τελικά, καταλάβαμε ότι ο EMD δεν ενδείκνυται για image similarity, αφού είναι 10^6 φορές πιο αργός και δίνει και χειρότερα αποτελέσματα.

Δ)
Αρχικά τρέξαμε τον αλγόριθμο σιλουέτας για τον νέο χώρο χωρίς να μεταφέρουμε πρώτα τα κεντροειδή από 10 -> 784 διαστάσεις. Αυτό έβγαλε πολύ καλά αποτελέσματα σε σχέση με αργότερα που βρήκαμε το median διάνυσμα για κάθε cluster και το ορίσαμε ως το centroid του cluster.

Ενδεικτικά:

- Σιλουέτα με δέκα διαστάσεις για centroids:

Silhouette: [0.254532, 0.238512, 0.206198, 0.243637, 0.210502, 0.251914, 0.199545, 0.216747, 0.403105, 0.315805, 0.25405]
Και clustering_time: 24.2387 seconds.

- Σιλουέτα με 784 διαστάσεις για centroids:

  Silhouette: [-0.0376777, 0.343408, -0.0163629, 0.0366741, 0.121258, 0.0175235, 0.215857, 0.0908704, 0.185348, 0.0614737, 0.101837]
  Και clustering_time: 287.831 seconds.

  Οπότε αν κρατήσουμε τα centroids στον νέο χώρο των 10 διαστάσεων το clustering είναι πάρα πολύ καλύτερο και γρηγορότερο (~10 φορές).
  Τώρα συγκρίνοντας τις 3 συσταδοποιήσεις (Σ1, Σ2, Σ3), με τον αλγόριθμο σιλουέτας, ο καλύτερος είναι:

- Σ2 (αρχικός χώρος)
- Σ1 (νέος χώρος)
- Σ3 (Clustering as classes)

Αλλά συγκρίνοντας τις συσταδοποιήσεις σύμφωνα με το objective function:

- Σ3
- Σ1
- Σ2

Βλέπουμε τεράστια διαφορά ανάλογα με τη μετρική που χρησιμοποιείται κάθε φορά.