Ψηφιακή Επεξεργασία Εικόνας
-Εργασία 3: Image segmentation

Α. Ντελόπουλος

Άνοιξη 2025

Εισαγωγικά

Στην εργασία αυτή θα υλοποιήσετε τα ακόλουθα:

1. Αναπαράσταση εικόνων σαν γράφους
2. Image segmentation με τη μέθοδο *spectral clustering*
3. Image segmentation με τη μέθοδο *normalized cuts* ή αλλιώς n-cuts με την αναδρομική και τη μη-αναδρομική μέθοδο

Μαζί με την εκφώνηση θα βρείτε και το βοηθητικό αρχείο `dip_hw_3.mat` το οποίο περιλαμβάνει τα δεδομένα που θα χρησιμοποιήσετε σε κάθε ερώτημα, όπως εικόνες εισόδου και affinity πίνακες.

1 Εικόνες ως γράφοι

Κατασκευάστε τη ρουτίνα `image_to_graph(.)` η οποία δέχεται σαν είσοδο μια εικόνα με C κανάλια και επιστρέφει τον affinity πίνακα που περιγράφει ένα *μη-κατευθυντικό* γράφο $G = (V, E)$. Πιο συγκεκριμένα:

```
def image_to_graph(
    img_array: np.ndarray,
) -> np.ndarray:
```

Ορίσματα:

- `img_array`: `numpy array` τύπου `dtype=float`, διαστάσεων $[M, N, C]$ και τιμές στο διάστημα $[0, 1]^C$, που αναπαριστά την εικόνα εισόδου C καναλιών

Έξοδοι:

- `affinity_mat`: ένας δισδιάστατος `numpy array` τύπου `dtype=float` και διαστάσεων $[MN, MN]$, που αναπαριστά τον αντίστοιχο affinity πίνακα

Λειτουργία: η συνάρτηση δέχεται την εικόνα εισόδου και σχηματίζει τον τετράγωνο (και συμμετρικό) affinity πίνακα που περιγράφει το γράφο εκείνης.

Θεωρήστε πως κάθε pixel της εικόνας αποτελεί ένα vertex (ή node) του τελικού γράφου. Επιπλέον, οι τιμές των βαρών των ακμών του γράφου θα υπολογίζονται ως $\mathbf{A}(i, j) = \frac{1}{e^{d(i,j)}}$, όπου $d(i, j)$ η Ευκλείδεια απόσταση της φωτεινότητας των καναλιών μεταξύ του i-οστού και του j-οστού pixel.

Ο γράφος που θα παράγει η ρουτίνα `image_to_graph()` θα πρέπει να είναι fully-connected, δηλαδή για κάθε ζεύγος κορυφών i,j θα πρέπει να υπάρχει η αντίστοιχη ακμή $e_{i,j}$ με μη-μηδενικό βάρος.

2 Spectral Clustering

Στη δεύτερη ενότητα της εργασίας θα υλοποιήσετε τη μέθοδο Spectral Clustering. Τα βήματα της μεθόδου περιγράφονται παρακάτω:

1. Δεδομένης μιας εικόνας εισόδου, κατασκευάστε ένα μη κατευθυντικό γράφο σύμφωνα με τις προδιαγραφές της Ενότητας 1. Έστω $\mathbf{W}$ ο affinity πίνακας που περιγράφει τον γράφο.
2. Υπολογίστε το Λαπλασιανό πίνακα $\mathbf{L}$ ως:
$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$
Ο διαγώνιος πίνακας $\mathbf{D}$ ορίζεται ως: $\mathbf{D}(i,i) = \sum_j \mathbf{W}(i,j)$.
3. Λύστε το πρόβλημα ιδιοτιμών $\mathbf{Lx} = \lambda\mathbf{x}$, και υπολογίστε τις k **μικρότερες** ιδιοτιμές καθώς και τα k ιδιοδιανύσματα που αντιστοιχούν σε αυτές τις ιδιοτιμές.
4. Σχηματίστε τον πίνακα $\mathbf{U} \in \mathbb{R}^{n\times k}$ που περιέχει τα ιδιοδιανύσματα $\mathbf{u_1}, \ldots, \mathbf{u_k}$ σαν στήλες. Για $i = 1, \ldots, n$, έστω $\mathbf{y_i} \in \mathbb{R}^k$ το διάνυσμα που αντιστοιχεί στην i-οστή γραμμή του $\mathbf{U}$.
5. Ομαδοποιήστε τα σημεία $(\mathbf{y_i})_{i=1,\ldots,n}$ με τον αλγόριθμο k-means στα clusters $C_1, \ldots, C_k$.

Επομένως, υλοποιήστε τη συνάρτηση:

```
def spectral_clustering(
    affinity_mat: np.ndarray,
    k: int
) -> np.ndarray:
```

Ορίσματα:

- `affinity_mat`: Ο τετράγωνος και συμμετρικός `numpy array` διαστάσεων $[MN, MN]$, που αναπαριστά την εικόνα εισόδου ως γράφο.
- `k`: Το πλήθος των clusters που επιθυμούμε να σχηματιστούν.

Έξοδοι:

- `cluster_idx`: ένας μονοδιάστατος `numpy array` τύπου `dtype=float` και μήκους MN, με τις ετικέτες που δείχνουν σε ποιο cluster ανήκει ο κάθε κόμβος του γράφου εισόδου.

Λειτουργία: η συνάρτηση δέχεται σαν είσοδο έναν affinity πίνακα που περιγράφει έναν μη-κατευθυντικό γράφο $G = (V, E)$ και τον αριθμό των clusters, και επιστρέφει τις ετικέτες των clusters στις οποίες ανήκουν οι κορυφές του γράφου.

Η μη-επιβλεπώμενη διαδικασία ομαδοποίησης k-means θα πρέπει να γίνει με τη χρήση της έτοιμης υλοποίησης από το πακέτο `sklearn` ως εξής:

```
from sklearn.cluster import KMeans

# Set the random_state parameter to 1, to ensure reproducibility across experiments.
# Assume 'k' refers to the desired number of clusters
kmeans = KMeans(n_clusters = k, random_state=1)

# X refers to the input data matrix
kmeans.fit(X)
```

```
# obtain the cluster labels for each input data point
labels = kmeans.labels_
```

TIP: Ο υπολογισμός των ιδιοδιανυσμάτων στον κώδικά σας θα πρέπει να γίνεται με τη χρήση της έτοιμης υλοποίησης `scipy.sparse.linalg.eigs`. Μελετήστε καλά το επίσημο documentation, καθώς οι affinity πίνακες που θα κατασκευαστούν δύνανται να έχουν μεγάλες διαστάσεις, ο υπολογισμός *όλων* των ιδιοδιανυσμάτων θα είναι από **πολύ** χρονοβόρος έως απαγορευτικός.

2.1 Demo 1

Για το πρώτο demo της ενότητας 2 καλείστε να παρουσιάσετε την λειτουργία της ρουτίνας `spectral_clustering(.)`. Για τους σκοπούς του demo, σας δίνεται ένας προ-κατασκευασμένος affinity πίνακας (μεταβλητή `d1a` του βοηθητικού αρχείου `dip_hw_3.mat`). Επιπλέον, για τον λόγο του οτι ο αλγόριθμος k-means έχει ως πρώτο βήμα την τυχαία αρχικοποίηση των k κέντρων του, θέσατε `random_state = 1` στην αρχικοποίηση του `KMeans(.)`, έτσι ώστε να μπορείτε να επαναλάβετε το πείραμα με το ίδιο random seed και κατ' επέκταση να έχετε τα ίδια αποτελέσματα ταξινόμησης στα k clusters. Παρουσιάστε τις ετικέτες που προκύπτουν για παραμέτρους $k = 2$, $k = 3$ και $k = 4$ (σύνολο 3 πειράματα). Παρουσιάστε/σχολιάστε τα αποτελέσματα στην αναφορά σας.

Μπορείτε να διαβάσετε τα περιεχόμενα του `.mat` αρχείου ως εξής:

```
from scipy.io import loadmat

data = loadmat("dip_hw_3.mat")

d1a = data["d1a"]
```

Οι παραπάνω λειτουργίες θα πρέπει να βρίσκονται μέσα στο script `demo1.py`.

2.2 Demo 2

Για το δεύτερο demo της ενότητας 2 καλείστε να παρουσιάσετε την λειτουργία της `spectral_clustering(.)` σε συνδυασμό με την ρουτίνα `image_to_graph(.)`. Πιο συγκεκριμένα στο βοηθητικό `mat` αρχείο σας δίνονται 2 RGB εικόνες εισόδου με ονόματα μεταβλητών `d2a` και `d2b`. Μετατρέψτε κάθε εικόνα εισόδου στον αντίστοιχο γράφο (δηλαδή στον αντίστοιχο affinity πίνακα) με την χρήση της `image_to_graph(.)` και στην συνέχεια πραγματοποιήστε την διαδικασία spectral clustering. Επαναλάβετε το πείραμα για αριθμούς κέντρων $k = 2$, $k = 3$ και $k = 4$ και για τις 2 εικόνες (σύνολο 6 πειράματα).

Δείξτε τα αποτελέσματα της διαδικασίας clustering πάνω στην κάθε εικόνα εισόδου. Όπως και στο `demo1`, χρησιμοποιήστε `random_state = 1` για να ελέγξετε την τυχαιότητα της αρχικοποίησης του k-means. Σχολιάστε τα αποτελέσματα.

Οι παραπάνω λειτουργίες θα πρέπει να βρίσκονται μέσα στο script `demo2.py`.

3 Normalized-cuts

Για το τελευταίο κομμάτι της εργασίας θα κατασκευάσετε 2 εκδοχές (αναδρομική και μη-αναδρομική) της μεθόδου για image segmentation, με την ονομασία **normalized-cuts** ή **n-cuts**.

Αναφορικά με τη μη-αναδρομική εκδοχή, τα βήματα της μεθόδου είναι τα παρακάτω:

1. Δεδομένης μιας εικόνας εισόδου, κατασκευάστε ένα μη κατευθυντικό γράφο σύμφωνα με τις προδιαγραφές της Ενότητας 1. Έστω **W** ο affinity πίνακας που περιγράφει τον γράφο.

2. Υπολογίστε το μη-κανονικοποιημένο Λαπλασιανό πίνακα **L** ως $\mathbf{L} = \mathbf{D} - \mathbf{W}$. Ο διαγώνιος πίνακας **D** ορίζεται ως: $\mathbf{D}(i,i) = \sum_j \mathbf{W}(i,j)$.

3. Λύστε το **γενικευμένο** πρόβλημα ιδιοτιμών $\mathbf{Lx} = \lambda\mathbf{Dx}$, και υπολογίστε τις k **μικρότερες** ιδιοτιμές καθώς και τα k ιδιοδιανύσματα που αντιστοιχούν σε αυτές τις ιδιοτιμές.

4. Σχηματίστε τον πίνακα $\mathbf{U} \in \mathbb{R}^{n \times k}$ που περιέχει τα ιδιοδιανύσματα $\mathbf{u_1}, \ldots, \mathbf{u_k}$ σαν στήλες. Για $i = 1, \ldots, n$, έστω $\mathbf{y_i} \in \mathbb{R}^k$ το διάνυσμα που αντιστοιχεί στην i-οστή γραμμή του **U**.

5. Ομαδοποιήστε τα σημεία $(\mathbf{y_i})_{i=1,\ldots,n}$ με τον αλγόριθμο k-means στα clusters $C_1, \ldots, C_k$.

Στην συνέχεια, σαν βήμα 6, η μη-αναδρομική εκδοχή *ενώνει* σταδιακά τα k clusters που δημιουργήθηκαν. Στα πλαίσια αυτής της εργασίας όμως θα θεωρήσουμε πως η μη-αναδρομική εκδοχή της μεθόδου τελειώνει στη δημιουργία των k clusters (βήμα 5). Όπως μπορείτε να δείτε, η μόνη διαφορά μεταξύ της μεθόδου Spectral Clustering της Ενότητας 2 και της μη-αναδρομικής εκδοχής $n$cuts είναι το βήμα 3 και συγκεκριμένα το πρόβλημα των ιδιοτιμών που λύνουμε κάθε φορά.

Η αναδρομική εκδοχή της μεθόδου είναι μια υποπερίπτωση της μη-αναδρομικής για $k = 2$ [1] (δείτε το βήμα 3 παραπάνω), δηλαδή κάθε φορά χωρίζουμε τον γράφο σε **2** κομμάτια. Μετά από κάθε διχοτόμηση (βήματα 1 μέχρι και 5 με $k = 2$), αποφασίζουμε αν θα συνεχιστεί η διχοτόμηση των συγκεκριμένων κομματιών που προέκυψαν από την διαδικασία ομαδοποίησης (k-means συγκεκριμένα). Πιο συγκεκριμένα μπορείτε να πάρετε την απόφαση με τον παρακάτω τρόπο: Αν ο αριθμός των κόμβων είτε με ετικέτα 1 είτε 2 που προκύπτουν είναι μικρότερος από ένα κατώφλι T^1 **ή** αν η τιμή $Ncut(A,B)$ είναι μεγαλύτερη από ένα κατώφλι T^2, τότε η διχοτόμηση των συγκεκριμένων κομματιών που προέκυψαν σταματά. Σε διαφορετική περίπτωση κάθε ένα από τα δύο κομμάτια που προέκυψαν από την διαδικασία ομαδοποίησης χωρίζεται στα 2. Η διαδικασία συνεχίζει αναδρομικά και τερματίζει όταν κανένα κομμάτι δεν μπορεί να διχοτομηθεί για τους λόγους που αναφέρθηκαν παραπάνω.

Η μετρική $Ncut(A,B)$ για 2 ομάδες κόμβων με ετικέτες "A" και "B" (ή 1 και 0) ορίζεται ως εξής:

$$Ncut(A,B) = 2 - Nassoc(A,B) \tag{1}$$

με

$$Nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)} \tag{2}$$

και

$$assoc(A,V) = \sum_{u \in A, t \in V} \mathbf{W}(u,t) \tag{3}$$

Ουσιαστικά η μετρική $assoc(A,V)$ είναι το άθροισμα όλων των βαρών μεταξύ των κόμβων που ανήκουν στην ομάδα A (ή έχουν ετικέτα 1) προς όλους τους κόμβους του γράφου (V). Οι μετρικές $assoc(A,A)$, $assoc(B,V)$ και $assoc(B,B)$ ορίζονται αντίστοιχα. Με **W** συμβολίζουμε τον affinity πίνακα.

Κατασκευάστε λοιπόν την εξής συνάρτηση, που υλοποιεί τον κορμό της μεθόδου n -cuts.

```
def n_cuts(
    affinity_mat: np.ndarray,
    k: int
) -> np.ndarray:
```

[1] Στην πραγματικότητα το ιδιοδιάνυσμα που αντιστοιχεί στην **μικρότερη** ιδιοτιμή δεν μας προσφέρει καμία πληροφορία (είναι σχεδόν constant) και θα μπορούσαμε να το παραλείψουμε εξ' ολοκλήρου χωρίς να δουμε διαφορά στα αποτελέσματα. Για πρακτικούς όμως λόγους, στα πλαίσια της εργασίας θα το χρησιμοποιήσουμε.

Ορίσματα:

- `affinity_mat`: Ο τετράγωνος και συμμετρικός `numpy array` διαστάσεων $[MN, MN]$, που αναπαριστά την εικόνα εισόδου ως γράφο.
- `k`: Το πλήθος των clusters που επιθυμούμε να σχηματιστούν. Σημειώστε ότι στην αναδρομική περίπτωση, πάντα δίνεται `k = 2`

Έξοδοι:

- `cluster_idx`: ένας μονοδιάστατος `numpy array` τύπου `dtype=float` και μήκους MN, με τις ετικέτες που δείχνουν σε ποιο cluster ανήκει ο κάθε κόμβος του γράφου εισόδου.

Λειτουργία: η συνάρτηση δέχεται σαν είσοδο έναν affinity πίνακα που περιγράφει έναν μη-κατευθυντικό γράφο $G = (V, E)$ και τον αριθμό των clusters, και επιστρέφει τις ετικέτες των clusters στις οποίες ανήκουν οι κορυφές του γράφου, σύμφωνα με τα βήματα που περιγράφηκαν νωρίτερα.

Επιπλέον, κατασκευάστε την ρουτίνα `calculate_n_cut_value` η οποία υπολογίζει την σχετική μετρική της εξίσωσης 1 ειδικότερα για τις 2 ομάδες (clusters) που προκύπτουν από το βήμα 3 της περιγραφής της μεθόδου. Πιο συγκεκριμένα:

```
def calculate_n_cut_value(
    affinity_mat: np.ndarray,
    cluster_idx: np.ndarray
) -> float:
```

Ορίσματα:

- `affinity_mat`: Ο τετράγωνος και συμμετρικός `numpy array` διαστάσεων $[MN, MN]$, που αναπαριστά την εικόνα εισόδου ως γράφο.
- `cluster_idx`: Ένας μονοδιάστατος πίνακας μήκους MN, με τις ετικέτες που δείχνουν σε ποιο από τα 2 cluster ανήκει ο κάθε κόμβος του γράφου.

Έξοδοι:

- `n_cut_value`: η τιμή της μετρικής για τις 2 ομάδες κόμβων, σύμφωνα με τη Σχέση 1

Λειτουργία: η συνάρτηση δέχεται σαν είσοδο έναν affinity πίνακα που περιγράφει έναν μη-κατευθυντικό γράφο $G = (V, E)$, καθώς και τις ετικέτες των clusters στους οποίους ανήκουν οι κορυφές του γράφου, και υπολογίζει την τιμή της μετρικής Ncut για τη διαμέριση αυτή.

Τέλος, προχωρήστε στην υλοποίηση της:

```
def n_cuts_recursive(
    affinity_mat: np.ndarray,
    T1: int
    T2: float
) -> np.ndarray:
```

Ορίσματα:

- `affinity_mat`: Ο τετράγωνος και συμμετρικός `numpy array` διαστάσεων $[MN, MN]$, που αναπαριστά την εικόνα εισόδου ως γράφο.

- `T1`: Το threshold στο πλήθος των σημείων εντός ενός cluster, σύμφωνα με την περιγραφή αυτής της ενότητας.
- `T2`: Το threshold στην τιμή της μετρικής `n_cut` για τη διαμέριση του γράφου.

Έξοδοι:

- `cluster_idx`: ένας μονοδιάστατος `numpy array` τύπου `dtype=float` και μήκους MN, με τις ετικέτες που δείχνουν σε ποιο cluster ανήκει ο κάθε κόμβος του γράφου εισόδου.

Λειτουργία: η συνάρτηση δέχεται σαν είσοδο έναν affinity πίνακα που περιγράφει έναν μη-κατευθυντικό γράφο $G = (V, E)$, καθώς και τα thresholds T^1, T^2 και προχωρά στη διαδοχική διχοτόμηση του γράφου εισόδου χρησιμοποιώντας τις παραπάνω συναρτήσεις, σύμφωνα με όσα έχουν αναλυθεί προηγουμένως.

3.1 Demo 3

Για το τελευταίο demo καλείστε να παρουσιάσετε την λειτουργία των ρουτινών `n_cuts`, `calculate_n_cut_value` και `n_cuts_recursive`. Για ακόμη μια φορά σαν εικόνες εισόδου χρησιμοποιήστε τις εικόνες του `demo2` (μεταβλητές `d2a` και `d2b`). Η τυχαιότητα των πειραμάτων που έγκειται στη χρήση του k-means, μπορεί να ελεγχθεί όπως και στις προηγούμενες περιπτώσεις.

Σκεφτείτε την διαδικασία διαχωρισμού σε 2 clusters σαν την δημιουργία ενός δυαδικού (unbalanced ενδεχομένως) δέντρου, όπου κάθε κόμβος του δέντρου κρατάει την πληροφορία των ετικετών σε σχέση με τον γονέα του.

3.1.1 Demo 3a

Αρχικά κατασκευάστε τους αντίστοιχους γράφους και ομαδοποιήστε τα nodes τους με την μη-αναδρομική εκδοχή της μεθόδου $n$cuts. Επαναλάβετε το πείραμα για αριθμούς κέντρων $k = 2$, $k = 3$ και $k = 4$ και για τις 2 εικόνες (σύνολο 6 πειράματα).

Δείξτε τα αποτελέσματα της διαδικασίας πάνω στην κάθε εικόνα εισόδου. Συγκρίνετε και σχολιάστε τα αποτελέσματα της μη-αναδρομικής μεθόδου $n$cuts σε σχέση με τα αποτελέσματα που προκύπτουν από την διαδικασία spectral clustering.

Οι παραπάνω λειτουργίες θα πρέπει να βρίσκονται μέσα στο script `demo3a.py`.

3.1.2 Demo 3b

Στη συνέχεια, και αφού κατασκευάσετε τους αντίστοιχους γράφους για τις 2 εικόνες εισόδου, εκτελέστε την αναδρομική μέθοδο n-cuts για **ένα βήμα**, δηλαδή σπάστε τον αρχικό γράφο σε 2 κομμάτια (σύνολο 2 πειράματα).

Δείξτε τα αποτελέσματα της διαδικασίας $n$cuts για ένα βήμα πάνω στην κάθε εικόνα εισόδου, καθώς και την τιμές των μετρικών $n$cut σε κάθε περίπτωση. Συγκρίνετε και σχολιάστε τα αποτελέσματα της μεθόδου n-cuts σε σχέση με τα αποτελέσματα που προκύπτουν από την διαδικασία spectral clustering για $k = 2$.

Οι παραπάνω λειτουργίες θα πρέπει να βρίσκονται μέσα στο script `demo3b.py`.

3.1.3 Demo 3c

Για το τελευταίο κομμάτι του demo καλείστε να παρουσιάσετε την **ολοκληρωμένη** (i.e. αναδρομική) εκτέλεση της μεθόδου $n$cuts. Για κατώφλια T^1 και T^2 χρησιμοποιήστε τις τιμές και 5 και 0.20 αντίστοιχα. Σαν εικόνες εισόδου χρησιμοποιήστε τις `d2a` και `d2b`.

Δείξτε τα αποτελέσματα της ολοκληρωμένης διαδικασίας $n$cuts πάνω στην κάθε εικόνα εισόδου. Συγκρίνετε και σχολιάστε τα αποτελέσματα της αναδρομικής μεθόδου $n$cuts σε σχέση με τα αποτελέσματα που προκύπτουν από την

διαδικασία spectral clustering και τα αποτελέσματα της μη-αναδρομικής μεθόδου $n$cuts για $k = 2$ και $k = 3$.

Οι παραπάνω λειτουργίες θα πρέπει να βρίσκονται μέσα στο script `demo3c.py`.

Αξιολόγηση & παραδοτέα

Κατά την υποβολή της εργασίας θα πρέπει να παραδώσετε μια αναφορά και τα αρχεία με τις συναρτήσεις:

- `image_to_graph.py`, με την αντίστοιχη συνάρτηση
- `spectral_clustering.py`, με την αντίστοιχη συνάρτηση
- `n_cuts.py`, με όλες τις σχετικές υλοποιήσεις

Επιπλέον, θα πρέπει να παραδώσετε και τα scripts `demo1.py`, `demo2.py`, `demo3a.py`, `demo3b.py` και `demo3c.py`, τα οποία θα εκτελούνται **χωρίς ορίσματα** και θα παρουσιάζουν τα ζητούμενα των ενοτήτων 2.1, 2.2 και 3.1. Τέλος, στην αναφορά σας θα πρέπει επίσης να παρουσιάσετε όποιες σχεδιαστικές επιλογές έχετε κάνει.

4 Για την υποβολή της εργασίας

Παραδώστε μία αναφορά με τις περιγραφές και τα συμπεράσματα που σας ζητούνται στην εκφώνηση. Η αναφορά θα πρέπει να επιδεικνύει την ορθή λειτουργία του κώδικά σας στην εικόνα που σας δίνεται και να παρουσιάζει και σχολιάζει τις εικόνες και τα διαγράμματα που παράγονται από το πρόγραμμα επίδειξης.

Ο κώδικας θα πρέπει να είναι σχολιασμένος ώστε να είναι κατανοητό τι ακριβώς λειτουργία επιτελεί (σε θεωρητικό επίπεδο, όχι σε επίπεδο κλήσης συναρτήσεων). Επίσης, ο κώδικας θα πρέπει να εκτελείται και να υπολογίζει τα σωστά αποτελέσματα για *οποιαδήποτε* είσοδο πληροί τις υποθέσεις της εκφώνησης, και όχι μόνο για την εικόνα που σας δίνεται.

Απαραίτητες προϋποθέσεις για την βαθμολόγηση της εργασίας σας είναι ο κώδικας να εκτελείται χωρίς σφάλμα (μόνο demos που εκτελούνται επιτυχώς θα βαθμολογηθούν), καθώς και να τηρούνται τα ακόλουθα:

- Υποβάλετε ένα και μόνο αρχείο, τύπου zip.
- Το όνομα του αρχείου πρέπει να είναι `AEM.zip`, όπου `AEM` είναι τα τέσσερα ή πέντε ψηφία του Α.Ε.Μ. του φοιτητή.
- Το προς υποβολή αρχείο πρέπει να περιέχει τα αρχεία κώδικα Python και το αρχείο `report.pdf` το οποίο θα είναι η αναφορά της εργασίας.
- Η αναφορά πρέπει να είναι ένα αρχείο τύπου PDF, και να έχει όνομα `report.pdf`.
- Όλα τα αρχεία κώδικα πρέπει να είναι αρχεία κειμένου τύπου UTF-8, και να έχουν κατάληξη `.py`.
- Το αρχείο τύπου zip που θα υποβάλετε δεν πρέπει να περιέχει κανέναν φάκελο.
- Μην υποβάλετε τις εικόνες που σας δίνονται για πειραματισμό.
- Μην υποβάλετε αρχεία που δεν χρειάζονται για την λειτουργία του κώδικά σας, ή φακέλους/αρχεία που δημιουργεί το λειτουργικό σας, πχ “Thumbs.db”, “.DS_Store”, “.directory”.
- Για την ονομασία των αρχείων που περιέχονται στο προς υποβολή αρχείο, χρησιμοποιείτε μόνο αγγλικούς χαρακτήρες, και όχι ελληνικούς ή άλλα σύμβολα, πχ “#”, “$”, “%” κλπ.