Ψηφιακή Επεξεργασία Εικόνας - Εργασία 3: Image segmentation

Βογιατζής Χαρίσιος ΑΕΜ: 9192

July 4, 2025

Abstract

Σε αυτήν την εργασία υλοποιούμε και συγκρίνουμε τρεις αλγορίθμους για *segmentation* εικόνων: *spectral clustering*, *non-recursive* και *recursive Normalized Cuts*.

1 Εισαγωγή

Στόχος μας είναι η κατάτμηση εικόνων (Image Segmentation) με χρήση:

- **Spectral Clustering**.
- **Non-recursive Normalized Cuts**.
- **Recursive Normalized Cuts**.

2 Υλοποίηση

2.1 Συναρτήσεις

Οι συναρτήσεις έχουν δημιουργηθεί στο αρχείο functions.py.

- `image_to_graph(img_array: np.ndarray) -> np.ndarray`:
  Μετασχηματίζει την εικόνα σε έναν πλήρως συνδεδεμένο γράφο, ο οποίος αναπαριστάται από τον πίνακα afinity ο υπολογισμός του οποίου φαίνεται παρακάτω:

```
# Reshape the image array to a list of pixels
# Each row is a pixel, and columns are the channel values
pixel_list = img_array.reshape(-1, c)
# Calculate the pairwise Euclidean distances between all pixels
p1 = pixel_list[:, np.newaxis, :]
p2 = pixel_list[np.newaxis, :, :]
diff = p1 - p2
dist_sq = np.sum(diff**2, axis=-1)
distances = np.sqrt(dist_sq)
# Calculate the affinity matrix using A[i,j] = exp(-d(i,j))
affinity_map = np.exp(-distances)
```

- `spectral_clustering(affinity_mat: np.ndarray, k: int) -> np.ndarray`
  Δεδομένου του afinity πίνακα affinity_mat υπολογίζει το Λαπλασιανό πίνακα L και υπολογίζει τις k μικρότερες ιδιοτιμές και ιδιοδιανύσματα. Στη συνέχεια χρησιμοποιεί k-means για να δημιουργήσει clusters.

```
# Calculate the Laplacian matrix L = D - W
D = np.diag(np.sum(affinity_mat, axis=1))
L = D - affinity_mat

# Solve the eigenvalue problem for the k smallest eigenvalues
# We ask for k+1 eigenvectors and discard the first one (trivial
    eigenvector)
# 'SM' for smallest magnitude eigenvalues
eigenvalues, eigenvectors = eigs(L, k=k, which='SM')
```

```
    # The eigenvectors are in the columns of the returned array
    # We need to take the real part as they can be complex
    U = np.real(eigenvectors)

    # Apply K-Means clustering
    # Each row of U is a data point to be clustered
    kmeans = KMeans(n_clusters=k, random_state=1, n_init=10)
    cluster_idx = kmeans.fit_predict(U)
```

- `n_cuts(affinity_mat: np.ndarray, k: int) -> np.ndarray`
  Λειτουργεί παρόμοια με το spectral clustering με τη διαφορά ότι λύνει το γενικευμένο πρόβλημα $Lx = \lambda Dx$.

```
    # Calculate the diagonal matrix D
    D = np.diag(np.sum(affinity_mat, axis=1))

    # Calculate the Laplacian matrix L
    L = D - affinity_mat

    # Solve the generalized eigenvalue problem L*x = l*D*x for the k
       smallest eigenvalues.
    # 'SM' - Smallest Magnitude.
    try:
        eigenvalues, eigenvectors = eigs(L, k=k, M=D, which='SM')
    except Exception as e:
        # Add a small value to the diagonal of D to avoid singularity
           issues
        D_reg = D + np.eye(D.shape[0]) * 1e-9
        eigenvalues, eigenvectors = eigs(L, k=k, M=D_reg, which='SM')

    # The eigenvectors are in the columns, take the real part
    U = np.real(eigenvectors)

    # Cluster the rows of U using K-Means
    kmeans = KMeans(n_clusters=k, random_state=1, n_init=10)
    cluster_idx = kmeans.fit_predict(U)
```

- `calculate_n_cut_value(full_W: np.ndarray, partition_nodes_indices: np.ndarray, labels: np.ndarray) -> float`
  Υπολογίζει τη μετρική Ncut σύμφωνα με τις προδιαγραφές.

```
    cluster_A_local_indices = np.where(labels == 0)[0]
    cluster_B_local_indices = np.where(labels == 1)[0]

    if len(cluster_A_local_indices) == 0 or len(cluster_B_local_indices) ==
        0:
        return float('inf')  # Invalid cut

    # Map local indices to global indices
    cluster_A_global_indices = partition_nodes_indices[
        cluster_A_local_indices]
    cluster_B_global_indices = partition_nodes_indices[
        cluster_B_local_indices]

    # Calculate associations using the full affinity matrix
    assoc_A_A = full_W[np.ix_(cluster_A_global_indices,
        cluster_A_global_indices)].sum()
    assoc_B_B = full_W[np.ix_(cluster_B_global_indices,
        cluster_B_global_indices)].sum()
    assoc_A_V = full_W[cluster_A_global_indices, :].sum()
    assoc_B_V = full_W[cluster_B_global_indices, :].sum()

    term1 = assoc_A_A / assoc_A_V if assoc_A_V != 0 else 0
```

```
        term2 = assoc_B_B / assoc_B_V if assoc_B_V != 0 else 0
        n_assoc = term1 + term2

        return 2 - n_assoc
```

- `n_cuts_recursive(affinity_mat: np.ndarray, T1: int, T2: float) -> np.ndarray`
  Στην αναδρομική εκδοχή του αλγορίθμου χρησιμοποιούμε μία ουρά για να χωρίσουμε αναδρομικά τα partitions. Η αναδρομή σταματάει με βάση δύο μετρικές, την T1 και την T2 και η μέθοδος αυτή μας επιτρέπει να βρίσκουμε το βέλτιστο αριθμό από clusters (*υπό την προϋπόθεση ότι έχουμε επιλέξει τις κατάλληλες υπερπαραμέτρους T1, T2).
  Αναλυτικά δείτε το καλά σχολιασμένο source code.

3 Demos

3.1 Demo Script 1 (`demo1.py`)

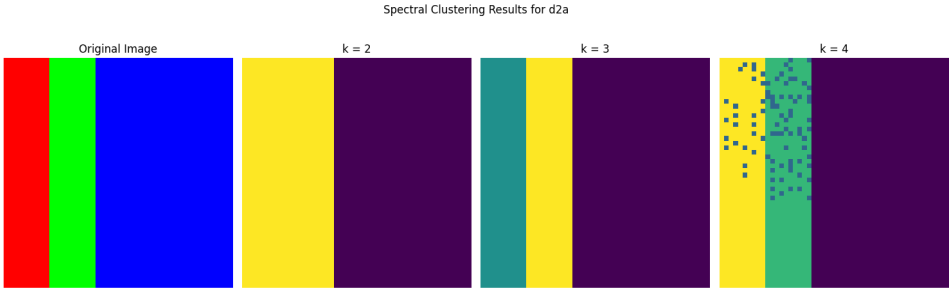
Το demo1 παρουσιάζει τη λειτουργία της spectral_clustering() και τα αποτελέσματα του παρουσιάζονται εδώ:

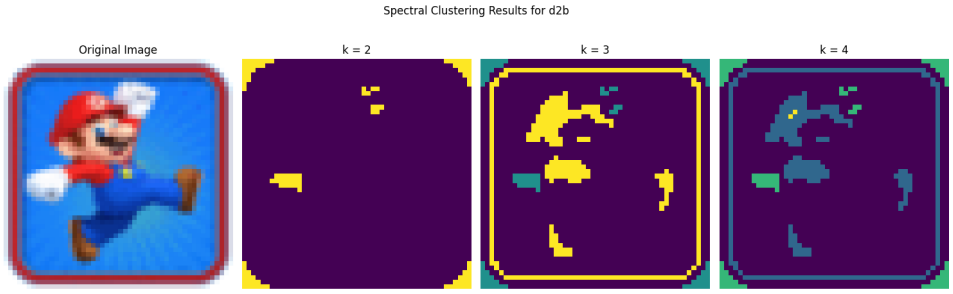
```
--- k = 2 ---
Cluster labels:
[1 1 1 1 0 0 0 0 0 0 0 0]
--------------
--- k = 3 ---
Cluster labels:
[1 1 1 1 0 0 0 0 2 2 2 2]
--------------
--- k = 4 ---
Cluster labels:
[0 0 0 0 1 1 1 3 3 2 2 2]
--------------
```

Είναι ξεκάθαρο ότι ανάλογα με την επιλογή του k έχουμε και τα αντίστοιχα clusters.
Και στις 3 περιπτώσεις το ένα cluster είναι κοινό, για k=3 φαίνεται ότι εντοπίζεται ένα νέο cluster που προηγουμένως δεν ήταν δυνατόν να εντοπιστεί, ενώ για k=4 φαίνεται ότι το επιπλέον cluster έχει δημιουργηθεί "δανειζόμενο" στοιχεία από δύο άλλα cluster.

3.2 Demo Script 2 (demo2.py

Το demo2 παρουσιάζει τη λειτουργία της spectal_clustering σε συνδιασμό με την image_to_graph() καθώς και τα αποτελέσματα του segmentation στις εικόνες.

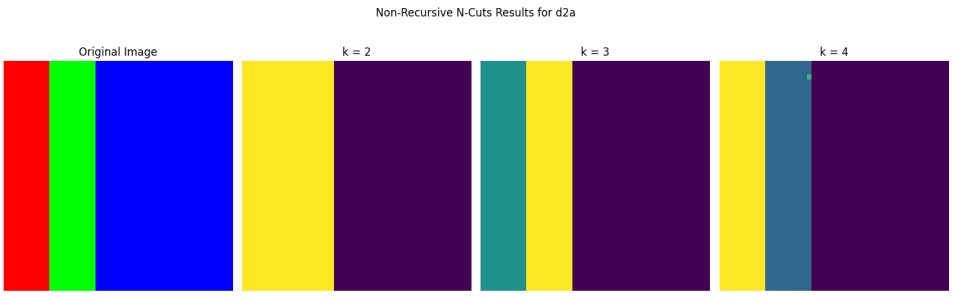


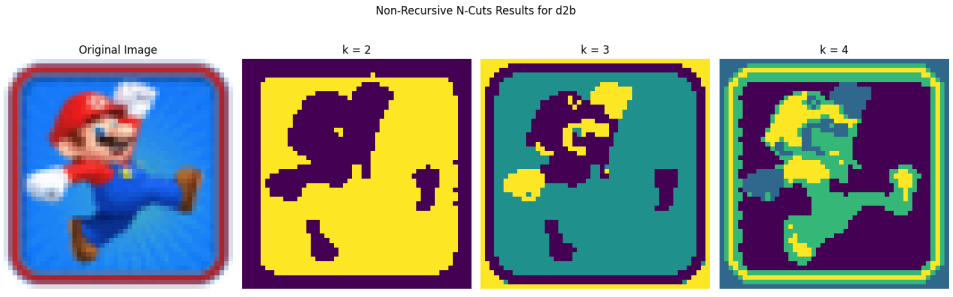


Όπως παρατηρούμε στην πρώτη εικόνα με τα τρία χρώματα το segmentation είναι καλό μόνο όταν k=3, ενώ στην δεύτερη και πιο περίπλοκη εικόνα το segmentation βελτιώνεται για μεγαλύτερα k.
Επιπλέον, για την πρώτη εικόνα βλέπουμε ότι ο αλγόριθμος συνένωσε τις δύο μικρές περιοχές (k=2) και προσπάθησε να δημιουργήσει ένα έξτρα cluster με pixels που δεν θα έπρεπε να χωριστούν (overclustering).

3.3 Demo Script 3a (demo3a.py)

Το demo3a παρουσιάζει τη λειτουργία της μη αναδρομικής Normalized Cuts και τα αποτελέσματα της στις εικόνες.

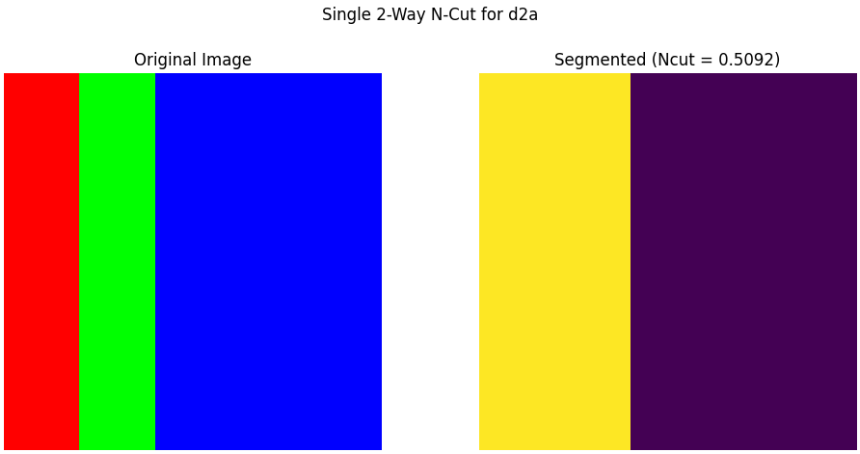


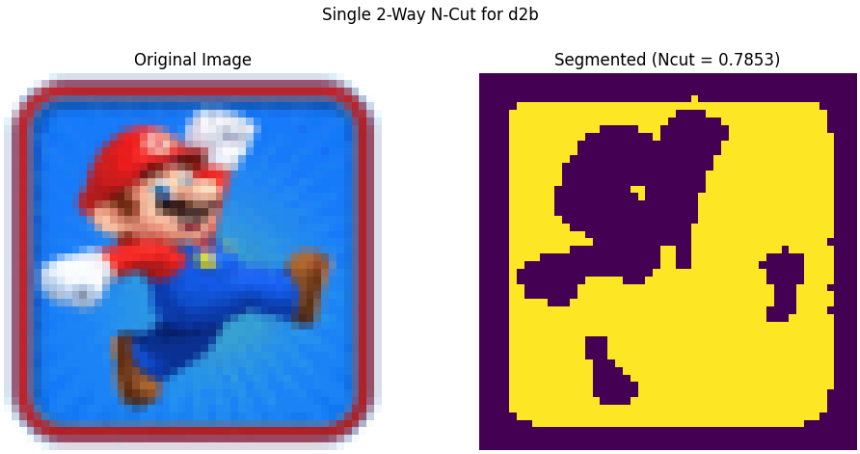


Παρατηρούμε ότι τα αποτελέσματα είναι καλύτερα και για τις δύο εικόνες καθώς το σφάλμα λόγω over-clustering στην 1 είναι πολύ περιορισμένο και το segmentation στην 2 είναι σαφώς ανώτερο.

3.4 Demo Script 3b (demo3b.py)

Το demo3b παρουσιάζει τη λειτουργία της αναδρομικής Normalized Cuts για ένα μόνο βήμα.

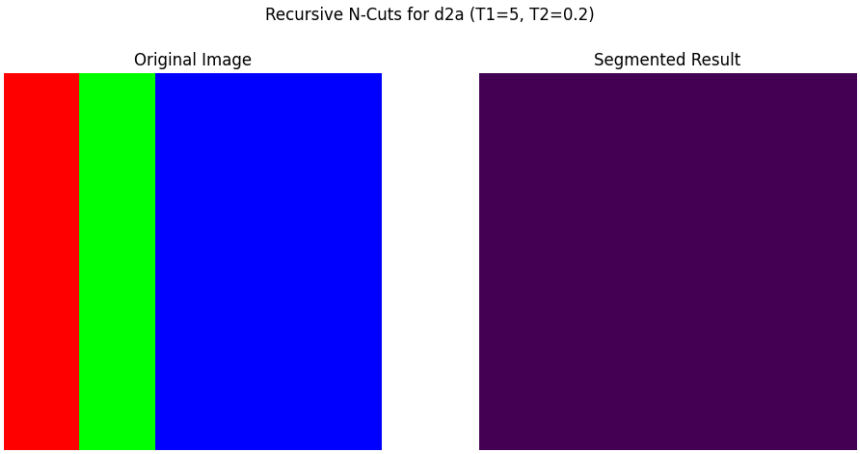




Παρατηρούμε ότι τα αποτελέσματα είναι τα ίδια με της μη αναδρομικής για k=2.

3.5 Demo Script 3c (demo3c.py)

Το demo3c παρουσιάζει την πλήρη λειτουργία της αναδρομικής Normalized Cuts.
Για τις τιμές των μετρικών-υπερπαραμέτρων που προτάθηκαν T1=5 και T2=0.2 ο αλγόριθμος δημιουργούσε ένα μόνο cluster όπως φαίνεται παρακάτω.



Ο λόγος που συμβαίνει αυτό είναι ότι φαίνεται ότι το T2=0.2 είναι αρκετά "αυστηρό" ως κριτήριο και σταματάει τη διαδικασία νωρίς.
Παρακάτω παρουσιάζονται τα αποτελέσματα για T1=5, T2=1.0 και T1=5, T2=1.4 για τις εικόνες 1 και 2 αντίστοιχα.

Παρατηρούμε ότι έχουμε σωστό segmentation και στις δύο εικόνες με εντοπισμό των βέλτιστων clusters. Ιδιαίτερα εντυπωσιακό είναι ότι στη 2η εικόνα ο αλγόριθμος εντοπίζει το μπλε παντελόνι του Mario από το επίσης μπλε background.

3.6 Σημείωση

Όπως αναφέρεται και στην αναφορά σας, το διάνυσμα που αντιστοιχεί στη μικρότερη ιδιοτιμή δεν μας προσφέρει καμία πληροφορία και συνεπώς αποφάσισα να το αφαιρέσω στην υλοποίηση μου.
Δεδομένου ότι η μία τιμή της T2 δεν μπορεί να γίνει generalized και άρα να χρησιμοποιηθεί για όλες τις εικόνες χρειάζεται ίσως να κάνουμε hyperparameter optimization ως προς την τιμή της, ανάλογα με την εικόνα που μας ενδιαφέρει.

4 Κώδικας

Μπορείτε να βρείτε τον κώδικα και στο Github.