



Image Analysis

Στο παρόν πρόγραμμα, γίνεται μία υλοποίηση της υπολογιστικής διαδικασίας του άρθρου «**Multimedia Retrieval through Unsupervised Hypergraph-based Manifold Ranking**» με στόχο την εξαγωγή των χαρακτηριστικών σε εικόνες και τον υπολογισμό του βαθμού ομοιότητας μεταξύ εικόνων.

Αρχικά γίνεται μετατροπή κάθε εικόνας ως ένα διάνυσμα μέσω της συνάρτησης ϵ και ύστερα γίνεται ο υπολογισμός της απόστασης κάθε ζεύγους εικόνων μέσω της συνάρτησης δ .

Έστω C μια συλλογή από εικόνες. Για κάθε εικόνα αυτής της συλλογής, υπολογίζεται μία λίστα κατάταξης t βασισμένη στο μέτρο ομοιότητας ρ . Κάθε λίστας κατάταξης έχει ταξινομημένες τις εικόνες κατά φθίνουσα σειρά με βάση την ομοιότητα. Δηλαδή, επειδή το μέτρο απόστασης είναι αντίστροφο του μέτρου ομοιότητας ρ , όσο μεγαλύτερο είναι το ρ μέσα σε μία λίστα κατάταξης, τόσο πιο «υψηλά» θα ταξινομηθεί.

Επειδή ο υπολογισμός κάθε ολόκληρης λίστας κατάταξης t έχει σημαντικό υπολογιστικό κόστος, υπολογίζονται για κάθε t μόνο οι L πιο σχετικές εικόνες με το L να είναι πολύ μικρότερο από τον αριθμό των πλήθους όλων των εικόνων. Με αυτόν τον τρόπο εντοπίζονται οι πλησιέστεροι γείτονες κάθε εικόνας.

Επομένως υλοποιείται το σύνολο T που περιέχει τις λίστες κατάταξης t για όλες τις εικόνες.

Υστερα γίνεται η υλοποίηση του Manifold Ranking. Ουσιαστικά υπολογίζεται ο βαθμός ομοιότητας κάθε ζεύγους εικόνων στο dataset ($T_r = f(T)$).

Για το Manifold ranking θα υλοποιηθεί το Hypergraph Manifold Ranking το οποίο απαρτίζεται από 5 στάδια:

- Κανονικοποίηση σειράς (Manifold Ranking):
Χρησιμοποιείται ώστε να κανονικοποιήσει τις αποστάσεις με σκοπό να βελτιώσει τη σειρά κατάταξης των εικόνων.
- Κατασκευή Υπεργράφου (Hypergraph Construction):
Μοντελοποιεί τους βαθμούς ομοιότητας ολόκληρου του dataset.

$$\text{Προκύπτει ο πίνακας } h(e_i, v_j) = \begin{cases} w(e_i, v_j), v_j \in e_i \\ 0, \text{αλλιώς} \end{cases}$$

- Υπολογισμός Ομοιότητας Υπερακμών (Hyperedge Similarities).
Υπολογίζει το βαθμό ομοιότητας μεταξύ εικόνων χρησιμοποιώντας τις υπερακμές.
- Υπολογισμός Καρτεσιανού Γινομένου μεταξύ στοιχείων των Υπερακμών (Cartesian Product of Hyperedge Elements).
Υπολογίζεται ώστε να μεγιστοποιήσει την πληροφορία που δίνουν οι υπερακμές για την ομοιότητα μεταξύ εικόνων.
- Υπολογισμός Ομοιότητας βάσει του κατασκευασμένου Υπεργράφου (Hypergraph – Based Similarity).
Ο υπολογισμός του Καρτεσιανού γινομένου μεταξύ των υπερακμών και οι ομοιότητες μεταξύ των υπερακμών συνδυάζονται ώστε να υπολογιστεί ο πίνακας ομοιότητας W βάσει του υπεργράφου, το οποίο μας δίνει νέες κατατάξεις t.

Έπειτα το νέο σύνολο T με τις νέες κατατάξεις t θα χρησιμοποιηθεί ώστε να προκύψει νέος τελικός πίνακας ομοιότητας W.

Έπειτα από T επαναλήψεις, θα δοθεί το τελικό σύνολο T_r .

Ανάλυση κώδικα :

Αρχικά, παίρνουμε τον φάκελο με όνομα coco_images και ορίζουμε ποιοι θα είναι οι μετασχηματισμοί που θα υλοποιηθούν στις εικόνες όπως τις διαστάσεις που χρειάζεται (224x224) ώστε να μετατραπούν σε tensors και να κανονικοποιηθούν. Κάθε τιμή από το mean και το std αναφέρονται σε ένα χρώμα με την εξής σειρά R,G και B.

Έπειτα δημιουργούμε μια λίστα με τα file paths.

```

11 # Ορισμός των μετασχηματισμών για τις εικόνες
12 transform = transforms.Compose([
13     transforms.Resize((224, 224)), # Μεγέθυνση εικόνων σε 224x224 για χρήση με ResNet
14     transforms.ToTensor(), # Μετατροπή της εικόνας σε tensor
15     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Κανονικοποίηση
16 ])
17
18
19 image_folder = datasets.ImageFolder(root='data/coco_images', transform=transform)
20 filenames = [image_folder.imgs[i][0] for i in range(len(image_folder))]# List of file paths
21

```

Έπειτα παίρνει 300 εικόνες ως dataset και το αποτέλεσμα εκχωρείται στη λίστα limited_filenames, γίνεται εξαγωγή των εικόνων και των labels και ύστερα μετατρέπονται οι εικόνες σε tensors (υλοποίηση συνάρτησης ε).

```

22 limited_image_folder = torch.utils.data.Subset(image_folder, range(300))
23 limited_filenames = filenames[:300]
24 print("Αντιστοίχιση αριθμών εικόνων και file-paths:")
25 for i in range(300):
26     print(i, " ", limited_filenames[i])
27
28 # Εξαγωγή των εικόνων και των labels
29 images, labels = zip(*[(data[0], data[1]) for data in limited_image_folder])
30
31 # Μετατροπή των εικόνων σε tensor
32 images_tensor = torch.stack(images)

```

Φορτώνεται το προ-εκπαιδευμένο μοντέλο ResNet50 και στη συνέχεια μέσω της συνάρτησης hook προσαρμόζουμε το μοντέλο ώστε να λάβουμε το flattening layer.

Εκχωρούνται στο μοντέλο οι εικόνες και τέλος γίνεται εκχώρηση των αποτελεσμάτων στη λίστα extracted_features.

```

34 # Φόρτωση του pre-trained ResNet50 μοντέλου
35 resnet_model = torchvision.models.resnet50(pretrained=True)
36
37 features = []
38
39 # usage
40 def hook(module, input, output):
41     flattened = output.view(output.size(0), -1)
42     features.append(flattened.clone().detach())
43
44 hook_handle = resnet_model.avgpool.register_forward_hook(hook)
45
46
47 with torch.no_grad():
48     resnet_model.eval()
49     _ = resnet_model(images_tensor)
50
51 hook_handle.remove()
52
53 extracted_features = torch.cat(features, dim=0)

```

Έχοντας μετατρέψει κάθε εικόνα σε μορφή διανύσματος (συνάρτηση ε), γεμίζουμε μία λίστα με όνομα distances. Κάθε γραμμή του πίνακα αντιπροσωπεύει μία εικόνα και κάθε στοιχείο της κάθε γραμμής είναι μία υπολίστα η οποία έχει 2 στοιχεία: το πρώτο είναι ο αριθμός μίας οποιαδήποτε εικόνας του dataset και το δεύτερο στοιχείο αντιπροσωπεύει την ευκλείδεια απόσταση της εικόνας αυτής από το πρώτο στοιχείο της γραμμής (υλοποίηση συνάρτησης δ).

Στη συνέχεια υπολογίζεται η λίστα κατάταξης $T=\{t_1, t_2, \dots\}$, με κάθε t να είναι μία λίστα κατάταξης για κάθε εικόνα του dataset.

Υπολογίζουμε πρώτα τα t_q

Η κάθε γραμμή του πίνακα που αναφέρθηκε αντιστοιχεί σε κάθε ένα από τα t . Ταξινομείται κατά αύξουσα σειρά η κάθε γραμμή του πίνακα ξεχωριστά με βάση το δεύτερο στοιχείο της κάθε υπο-λίστας που είναι η τιμή της απόστασης του εκάστοτε ζεύγους εικόνων. Αυτό οφείλεται στο γεγονός πως το μέτρο της απόστασης είναι αντίστροφο του μέτρου ομοιότητας: $t_q(i) < t_q(j) \rightarrow p(o_q, o_i) \geq p(o_q, o_j)$.

Έτσι, έχει υπολογιστεί η λίστα T .

```

55 distances = [[0 for _ in range(len(extracted_features))] for _ in range(len(extracted_features))]
56
57 for i,feature_tensor_v1 in enumerate(extracted_features):
58     for j,feature_tensor_v2 in enumerate(extracted_features):
59         distances[i][j] = [j, torch.norm(feature_tensor_v1 - feature_tensor_v2)]
60
61 distance_normalized = copy.deepcopy(distances)
62
63 for i in range(len(extracted_features)):
64     distances[i].sort(key=lambda x : x[1])

```

Στη συνέχεια, εφόσον έχουμε πάρει ένα αντίγραφο της λίστας με όνομα `distance_normalized`, καλείται η μέθοδος `repeater(extracted_features, distances, distance_normalized, 1)` η οποία ουσιαστικά θα υλοποιήσει το κύριο μέρος της εργασίας, δηλαδή το Manifold Ranking.

Η `distances` φέρει ό,τι και η `distance_normalized` με τη διαφορά πως η `distances` έχει ταξινομημένη την κάθε γραμμή της κατά αύξουσα σειρά με βάση την απόσταση της κάθε εικόνας (δεύτερο στοιχείο κάθε υπο-λίστας της κάθε γραμμής). Επίσης παίρνει ως είσοδο έναν counter για να μετρήσει πόσες φορές έχει εκτελεστεί η συνάρτηση `repeater`.

```
174 result_sorted, result = repeater(extracted_features, distances, distance_normalized, counter: 1)
```

Όταν κληθεί η μέθοδος `repeater`, θα καλέσει τη μέθοδο `get_neighbours(extracted_features, distances)` η οποία θα κρατήσει για κάθε εικόνα τους 4 πλησιέστερους γείτονές της, βρίσκοντας έτσι τη λίστα $N_4(q)$ με q μία από τις εικόνες του dataset.

```

2 usages
69 def get_neighbours(extracted_features, distances):
70     neighbours = [[0 for _ in range(4)] for _ in range(len(extracted_features))]
71
72     for i in range(len(extracted_features)):
73         for j in range(4):
74             neighbours[i][j] = distances[i][j][0]
75
76     return neighbours

```

Η μέθοδος `get_neighbours` γεμίζει με μηδενικά τη λίστα `neighbours` και στη συνέχεια γεμίζει τον πίνακα με τις κατάλληλες τιμές, ώστε να εξασφαλιστεί η λίστα με τους 4 πλησιέστερους γείτονες για κάθε εικόνα.

Υλοποίηση του Manifold Ranking μέσω του Hypergraph Manifold Ranking.

A. Υλοποίηση του Rank Normalization.

Υποθέτοντας πως υπάρχει κάποιο κανονικοποιημένο μέτρο ομοιότητας μεταξύ όλων των ζευγών εικόνων, σε μία διπλή loop, βρίσκουμε κάθε υπαρκτό συνδυασμό $t_i(q_j)$ και $t_j(q_i)$. Υποθέτοντας πως είναι συμμετρικά, δηλαδή $P_n(O_i, O_j) = P_n(O_j, O_i)$, εφαρμόζουμε τον τύπο $p_n(o_i, o_j) = 2 * L - (t_i(j) + t_j(i))$ και αντικαθιστούμε την απόσταση κάθε συνδυασμού εικόνων με την τιμή που προκύπτει από αυτόν τον τύπο.

Στη συνέχεια, κάνουμε ταξινόμηση ξανά ώστε να υπάρχει κατάταξη.

```
85     # A) --- Rank Normalization ---
86
87     L = len(extracted_features)
88
89     for i in range(L):
90         for j in range(L):
91             positions_v1 = [k for k, sublist in enumerate(distances[i]) if sublist[0] == j]
92             positions_v2 = [l for l, sublist in enumerate(distances[j]) if sublist[0] == i]
93             distance_normalized[i][j][1] = 2 * L - (positions_v1[0] + positions_v2[0])
94
95     for i in range(len(extracted_features)):
96         distance_normalized[i].sort(key=lambda x: x[1], reverse=True)
```

B. Υλοποίηση του Hypergraph Construction.

Στον πίνακα `neighbors_normalized`, εφόσον έχουν βρεθεί οι 4 πλησιέστεροι γείτονες για κάθε εικόνα με βάση τα κανονικοποιημένα μέτρα ομοιότητας, υποθέτουμε πως κάθε εικόνα αποτελεί έναν κόμβο και κάθε γραμμή του πίνακα αποτελεί μία υπερακμή, δηλαδή μια γειτονιά.

Σε κάθε υπερακμή μπορούν να συμπεριληφθούν παραπάνω από 2 στοιχεία.

Σε κάθε συνδυασμό μιας εικόνας με κάθε έναν από τους τέσσερις πλησιέστερους γείτονες της, εκχωρούμε τον αριθμό 1, δηλώνοντας πως ανήκουν στην ίδια υπερακμή. Επομένως προκύπτει ο πίνακας H του κώδικα.

Με αυτόν τον τρόπο δηλώνεται πως κάθε υπερακμή συνδέεται με έναν κόμβο όταν ο κόμβος ανήκει σε αυτήν.

```
101    # B) --- Hypergraph Construction ---
102
103    Hb = [[0 for _ in range(len(extracted_features))] for _ in range(len(extracted_features))]
104    neighbors_normalized = get_neighbours(extracted_features, distance_normalized)
105
106    for i in range(len(extracted_features)):
107        for j in range(4):
108            Hb[i][neighbors_normalized[i][j]] = 1
```

Τώρα θέλουμε να αντικαταστήσουμε αυτές τις συνδέσεις που έχουν τιμή 1, με βάρη, με κάθε βάρος να ανήκει ανάμεσα στο διάστημα [0,1]. Αυτό θα γίνει υπολογίζοντας τη σχέση: $w(i,j) = 1 - \log_{k+1}[t_i(j)]$ με $k = 4$ (όσοι οι γείτονες)

Κάθε υπερακμή έχει ένα βάρος μεγαλύτερο ή ίσο του μηδενός, το οποίο δείχνει πόσο σημαντική είναι η σχέση αυτής της υπερκαμής.

```

114     H = copy.deepcopy(Hb)
115
116     for i in range(L):
117         for j in range(L):
118             if H[i][j] == 1:
119                 positions_v1 = [k for k, sublist in enumerate(distance_normalized[i]) if sublist[0] == j]
120
121                 H[i][j] = 1 - math.log(positions_v1[0] + 1, base=5)

```

Δεδομένου πως κάθε υπερακμή έχει 4 κόμβους, υπολογίζεται το συνολικό βάρος κάθε υπερακμής με τον εξής τρόπο:

```

124     w = []
125     for i in range(L):
126         w.append(sum(H[i]))

```

Γ. Υλοποίηση του Hyperedges Similarities.

Υπολογίζουμε τον ανάστροφο του πίνακα H_array (πίνακας H_array_transpose).

Υπολογίζουμε το βαθμό ομοιότητας μεταξύ όλων των ζευγαριών υπερακμών που μπορεί να προκύψουν. Αυτό υλοποιείται από τον πολλαπλασιασμό των πινάκων $Sh = H_array * H_array_transpose$.

Έπειτα υπολογίζουμε το βαθμό ομοιότητας μεταξύ όλων των ζευγαριών κόμβων που μπορεί να προκύψουν. Αυτό υλοποιείται από τον πολλαπλασιασμό πινάκων $Sv = H_array_transpose * H_array$.

Έπειτα συνδυάζουμε τους δύο αυτούς υπολογισμούς κάνοντας τον εξής κανονικό πολλαπλασιασμό : $S = Sh * Sv$.

```

133     # C) --- Hyperedge Similarities ---
134
135     H_array = np.array(H)
136     H_array_transpose = H_array.T
137
138     Sh = np.dot(H_array, H_array_transpose)
139     Sv = np.dot(H_array_transpose, H_array)
140
141     S = Sh * Sv

```

Δ. Υλοποίηση του Cartesian Product of Hyperedge Elements.

Έχοντας δύο υπερακμές e_q και e_l , το καρτεσιανό γινόμενο μεταξύ τους είναι:

$$e_q \times e_l = \{(Vx, Vy) : V_x \in e_q \text{ και } V_y \in e_l\}$$

Θα υπολογιστεί το μέτρο ομοιότητας βάσει του Cartesian product για κάθε ζευγάρι κόμβων ανεξάρτητα από την υπερακμή που ανήκουν:

```
143     # D) --- Cartesian Product of Hyperedge Elements ---
144
145     C = [[0 for _ in range(len(extracted_features))] for _ in range(len(extracted_features))]
146
147     for i in range(L):
148         for j in range(L):
149             p = 0
150             for k in range(L):
151                 p += w[k] * H[k][i] * H[k][j]
152             C[i][j] = p
```

Με $w[k] * H[k][i] * H[k][j]$ να είναι η σχέση ομοιότητας για το εκάστοτε ζεύγος κόμβων $(v_i, v_j) \in e_q^2$

Ε. Υλοποίηση του Hypergraph-Based Similarity.

Στη συνέχεια θα υπολογιστεί ο πίνακας συνολικού βάρους W_{matrix} για κάθε ζευγάρι κόμβων:

Αυτό θα γίνει πολλαπλασιάζοντας τους πίνακες C και S .

```
154     W_matrix = C * S
```

Εφόσον έχει υπολογιστεί ο τελικός πίνακας που ποσοτικοποιεί τη συγγένεια/ομοιότητα μεταξύ οποιουδήποτε ζεύγους εικόνων, προκύπτει μια νέα λίστα κατάταξης T .

Στη συνέχεια υπολογίζεται η δισδιάστατη λίστα $W_{matrix_modified_sorted}$ η οποία σε κάθε στοιχείο της έχει μία υπο-λίστα με 2 στοιχεία: το πρώτο είναι μια οποιαδήποτε εικόνα του dataset και το δεύτερο στοιχείο είναι ο βαθμός συγγένειας/ομοιότητας που έχει σε σχέση με την εικόνα που αντιστοιχεί στη συγκεκριμένη γραμμή.

Δημιουργείται ένα αντίγραφο της λίστας αυτής, και στη συνέχεια κάθε γραμμή του δισδιάστατου πίνακα ταξινομείται με βάση το δεύτερο στοιχείο της κάθε υπο-λίστας.

```
164     W_matrix_modified = copy.deepcopy(W_matrix_modified_sorted)
165
166     for i in range(L):
167         W_matrix_modified_sorted[i].sort(key=lambda x: x[1], reverse=True)
168
169     print(f" ----- Η repeater εκτελέστηκε για {counter}η φορά... : -----")
```

Η μέθοδος επιστρέφει τη λίστα αυτή ως νέα λίστα Τ αλλά και το αντίγραφό της, το οποίο το λάβαμε πριν προβούμε στην ταξινόμησή της.

Έπειτα θα κληθεί εκ νέου η συνάρτηση repeater, με είσοδο τη νέα λίστα κατάταξης Τ ταξινομημένη (result_sorted) και την result με τη διαφορά πως η result_sorted έχει ταξινομημένη την κάθε γραμμή της κατά φθίνουσα σειρά με βάση το μέτρο ομοιότητας της κάθε εικόνας (δεύτερο στοιχείο κάθε υπο-λίστας της κάθε γραμμής).

Η συνάρτηση repeater θα εφαρμοστεί άλλες 6 φορές, προκύπτοντας με αυτόν τον τρόπο ο τελικός πίνακας κατάταξης Tr.

```
176     for i in range(6):
177         result_sorted, result = repeater(extracted_features, result_sorted, result, i+2)
```

Στη συνέχεια, επιλέγονται τυχαία 2 έως 6 εικόνες στόχοι (τυχαίος αριθμός πλήθους εικόνων στόχων στο διάστημα [3,6]).

```
179     times = random.randint( a: 3, b: 8)
180
181
182     target_images = []
183     target_images_numbers = []
184     for i in range(times):
185         target_images_number = random.randint( a: 0, len(result)-1)
186         print("Για Target Image την " + str(target_images_number) + " και σχετικές εικόνες: ")
187         target_images.append(result_sorted[target_images_number])
188         target_images_numbers.append(target_images_number)
189         for i in range(4):
190             print(result_sorted[target_images_number][i])
191     print("\n")
```

Για τον υπολογισμό της ακρίβειας του αλγορίθμου θα χρησιμοποιηθεί ο Μέσος Όρος των Μέτρων Ομοιότητας.

Ο τύπος υπολογίζει τον μέσο όρο της σχετικότητας για κάθε εικόνα στόχο με τύπο:

Μέσος Όρος των Μέτρων Ομοιότητας

$= \frac{1}{n} \sum_{i=1}^n$ Μέτρο Ομοιότητας i , με n τον αριθμό των 4 πλησιέστερων γειτόνων.

Όσο μεγαλύτερη είναι η τιμή, τόσο πιο σχετικές είναι και οι 4 εικόνες/γείτονες που συσχετίστηκαν με την εκάστοτε εικόνα στόχο.

Επομένως, με μια loop υπολογίζουμε τον τύπο και στη συνέχεια παίρνουμε τον μέσο όρο όλων των αποτελεσμάτων που προέκυψαν για κάθε εικόνα στόχο, δίνοντας το συνολικό μέσο αποτέλεσμα.

```
195     print("\n")
196     print("Οι βαθμολογίες των target images είναι: ")
197     total_avg_sum = 0
198     for i in range(len(target_images)):
199         sum = 0
200         for j in range(4):
201             sum+=target_images[i][j][1]
202         avg_sum=sum/4
203         print("Για την Target Image " + str(target_images_numbers[i]) + ": " + str(avg_sum))
204         total_avg_sum+=avg_sum
205     print("Total score is: "+str(total_avg_sum/len(target_images)))
```

Ακολουθούν ενδεικτικά screenshots από το output του προγράμματος :

```
C:\Users\theofanis\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\theofanis\Desktop\εξόμνω ζ'ΑΝΑΛΥΣΗ_ΕΙΚΟΝΑΣ\image_Analysis_Prerelease\Image_Analysis_Prerelease\main_prerelease.py"
Αντιστοίχιον οριζόντιων εικόνων και file-paths:
  0  data\coco_images\ipynb_checkpoints\000000000030.jpg
  1  data\coco_images\ipynb_checkpoints\000000000049.jpg
  2  data\coco_images\ipynb_checkpoints\000000000061.jpg
  3  data\coco_images\ipynb_checkpoints\000000000064.jpg
  4  data\coco_images\ipynb_checkpoints\000000000071.jpg
  5  data\coco_images\ipynb_checkpoints\000000000074.jpg
  6  data\coco_images\ipynb_checkpoints\000000000077.jpg
  7  data\coco_images\ipynb_checkpoints\000000000086.jpg
  8  data\coco_images\ipynb_checkpoints\000000000094.jpg
  9  data\coco_images\ipynb_checkpoints\000000000109.jpg
 10  data\coco_images\ipynb_checkpoints\000000000118.jpg
 11  data\coco_images\ipynb_checkpoints\000000000113.jpg
 12  data\coco_images\ipynb_checkpoints\000000000127.jpg
 13  data\coco_images\ipynb_checkpoints\000000000136.jpg
 14  data\coco_images\ipynb_checkpoints\000000000149.jpg
 15  data\coco_images\ipynb_checkpoints\000000000151.jpg
 16  data\coco_images\ipynb_checkpoints\000000000165.jpg
 17  data\coco_images\ipynb_checkpoints\000000000192.jpg
 18  data\coco_images\ipynb_checkpoints\000000000201.jpg
 19  data\coco_images\ipynb_checkpoints\0000000008350.jpg
 20  data\coco_images\ipynb_checkpoints\0000000008351.jpg
 21  data\coco_images\ipynb_checkpoints\0000000016478.jpg
 22  data\coco_images\ipynb_checkpoints\000000016526.jpg
 23  data\coco_images\ipynb_checkpoints\000000016610.jpg
 24  data\coco_images\ipynb_checkpoints\000000024221.jpg
 25  data\coco_images\ipynb_checkpoints\000000024664.jpg
 26  data\coco_images\ipynb_checkpoints\000000024762.jpg
 27  data\coco_images\ipynb_checkpoints\000000024778.jpg
 28  data\coco_images\ipynb_checkpoints\000000024779.jpg
 29  data\coco_images\ipynb_checkpoints\000000032773.jpg
 30  data\coco_images\ipynb_checkpoints\000000032809.jpg
 31  data\coco_images\ipynb_checkpoints\000000032810.jpg
 32  data\coco_images\ipynb_checkpoints\000000032892.jpg
 33  data\coco_images\ipynb_checkpoints\000000032907.jpg
 34  data\coco_images\ipynb_checkpoints\000000032949.jpg
 35  data\coco_images\ipynb_checkpoints\000000041079.jpg
 36  data\coco_images\ipynb_checkpoints\000000041085.jpg
 37  data\coco_images\ipynb_checkpoints\000000041103.jpg
 38  data\coco_images\ipynb_checkpoints\000000041132.jpg
 39  data\coco_images\ipynb_checkpoints\000000049283.jpg
 40  data\coco_images\ipynb_checkpoints\000000049327.jpg
 41  data\coco_images\ipynb_checkpoints\000000049371.jpg
 42  data\coco_images\ipynb_checkpoints\000000057353.jpg
 43  data\coco_images\ipynb_checkpoints\000000057361.jpg
 44  data\coco_images\ipynb_checkpoints\000000057362.jpg
 45  data\coco_images\ipynb_checkpoints\000000057445.jpg
 46  data\coco_images\ipynb_checkpoints\000000057460.jpg
 47  data\coco_images\ipynb_checkpoints\000000057503.jpg
 48  data\coco_images\ipynb_checkpoints\000000057571.jpg
 49  data\coco_images\ipynb_checkpoints\000000065567.jpg
 50  data\coco_images\ipynb_checkpoints\000000065584.jpg
 51  data\coco_images\ipynb_checkpoints\000000065594.jpg
 52  data\coco_images\ipynb_checkpoints\000000065604.jpg
 53  data\coco_images\ipynb_checkpoints\000000065632.jpg
 54  data\coco_images\ipynb_checkpoints\000000065643.jpg
 55  data\coco_images\ipynb_checkpoints\000000065668.jpg
 56  data\coco_images\ipynb_checkpoints\000000073729.jpg
 57  data\coco_images\ipynb_checkpoints\000000073861.jpg
 58  data\coco_images\ipynb_checkpoints\000000073864.jpg
 59  data\coco_images\ipynb_checkpoints\000000073906.jpg
 60  data\coco_images\ipynb_checkpoints\000000082042.jpg
 61  data\coco_images\ipynb_checkpoints\000000082081.jpg
 62  data\coco_images\ipynb_checkpoints\000000090173.jpg
 63  data\coco_images\ipynb_checkpoints\000000090218.jpg
 64  data\coco_images\ipynb_checkpoints\000000090331.jpg
 65  data\coco_images\ipynb_checkpoints\000000098304.jpg
 66  data\coco_images\ipynb_checkpoints\000000098390.jpg
```

```
07  data/coco_images\ipynb_checkpoints\000000098471.jpg
08  data/coco_images\ipynb_checkpoints\000000098493.jpg
09  data/coco_images\ipynb_checkpoints\000000098502.jpg
10  data/coco_images\ipynb_checkpoints\000000106525.jpg
11  data/coco_images\ipynb_checkpoints\000000106545.jpg
12  data/coco_images\ipynb_checkpoints\000000106610.jpg
13  data/coco_images\ipynb_checkpoints\000000106621.jpg
14  data/coco_images\ipynb_checkpoints\000000106624.jpg
15  data/coco_images\ipynb_checkpoints\000000114720.jpg
16  data/coco_images\ipynb_checkpoints\000000114778.jpg
17  data/coco_images\ipynb_checkpoints\000000114858.jpg
18  data/coco_images\ipynb_checkpoints\000000114869.jpg
19  data/coco_images\ipynb_checkpoints\000000122913.jpg
20  data/coco_images\ipynb_checkpoints\000000122934.jpg
21  data/coco_images\ipynb_checkpoints\000000122953.jpg
22  data/coco_images\ipynb_checkpoints\000000131084.jpg
23  data/coco_images\ipynb_checkpoints\000000131087.jpg
24  data/coco_images\ipynb_checkpoints\000000131089.jpg
25  data/coco_images\ipynb_checkpoints\000000131101.jpg
26  data/coco_images\ipynb_checkpoints\000000131108.jpg
27  data/coco_images\ipynb_checkpoints\000000131115.jpg
28  data/coco_images\ipynb_checkpoints\000000131127.jpg
29  data/coco_images\ipynb_checkpoints\000000131152.jpg
30  data/coco_images\ipynb_checkpoints\000000131160.jpg
31  data/coco_images\ipynb_checkpoints\000000131162.jpg
32  data/coco_images\ipynb_checkpoints\000000131172.jpg
33  data/coco_images\ipynb_checkpoints\000000131174.jpg
34  data/coco_images\ipynb_checkpoints\000000131197.jpg
35  data/coco_images\ipynb_checkpoints\000000131208.jpg
36  data/coco_images\ipynb_checkpoints\000000131211.jpg
37  data/coco_images\ipynb_checkpoints\000000131215.jpg
38  data/coco_images\ipynb_checkpoints\000000131228.jpg
39  data/coco_images\ipynb_checkpoints\000000131245.jpg
40  data/coco_images\ipynb_checkpoints\000000139291.jpg
41  data/coco_images\ipynb_checkpoints\000000139355.jpg
42  data/coco_images\ipynb_checkpoints\000000139390.jpg
43  data/coco_images\ipynb_checkpoints\000000139416.jpg
44  data/coco_images\ipynb_checkpoints\000000139429.jpg
45  data/coco_images\ipynb_checkpoints\000000139475.jpg
46  data/coco_images\ipynb_checkpoints\000000147471.jpg
47  data/coco_images\ipynb_checkpoints\000000147576.jpg
48  data/coco_images\ipynb_checkpoints\000000147629.jpg
49  data/coco_images\ipynb_checkpoints\000000155652.jpg
50  data/coco_images\ipynb_checkpoints\000000155707.jpg
51  data/coco_images\ipynb_checkpoints\000000155749.jpg
52  data/coco_images\ipynb_checkpoints\000000155754.jpg
53  data/coco_images\ipynb_checkpoints\000000155800.jpg
54  data/coco_images\ipynb_checkpoints\000000157981.jpg
55  data/coco_images\ipynb_checkpoints\000000163866.jpg
56  data/coco_images\ipynb_checkpoints\000000163897.jpg
57  data/coco_images\ipynb_checkpoints\000000163928.jpg
58  data/coco_images\ipynb_checkpoints\000000164036.jpg
59  data/coco_images\ipynb_checkpoints\000000164040.jpg
60  data/coco_images\ipynb_checkpoints\000000164042.jpg
61  data/coco_images\ipynb_checkpoints\000000172096.jpg
62  data/coco_images\ipynb_checkpoints\000000172103.jpg
63  data/coco_images\ipynb_checkpoints\000000180287.jpg
64  data/coco_images\ipynb_checkpoints\000000180373.jpg
65  data/coco_images\ipynb_checkpoints\000000180445.jpg
66  data/coco_images\ipynb_checkpoints\000000188459.jpg
67  data/coco_images\ipynb_checkpoints\000000188540.jpg
68  data/coco_images\ipynb_checkpoints\000000188575.jpg
69  data/coco_images\ipynb_checkpoints\000000188639.jpg
70  data/coco_images\ipynb_checkpoints\000000188648.jpg
71  data/coco_images\ipynb_checkpoints\000000196610.jpg
72  data/coco_images\ipynb_checkpoints\000000196619.jpg
73  data/coco_images\ipynb_checkpoints\000000196623.jpg
74  data/coco_images\ipynb_checkpoints\000000196639.jpg
75  data/coco_images\ipynb_checkpoints\000000196644.jpg
```

136 data\coco_images\ipynb_checkpoints\00000196675.jpg
137 data\coco_images\ipynb_checkpoints\00000196676.jpg
138 data\coco_images\ipynb_checkpoints\00000196686.jpg
139 data\coco_images\ipynb_checkpoints\00000196715.jpg
140 data\coco_images\ipynb_checkpoints\00000196723.jpg
141 data\coco_images\ipynb_checkpoints\00000196766.jpg
142 data\coco_images\ipynb_checkpoints\00000196773.jpg
143 data\coco_images\ipynb_checkpoints\00000196775.jpg
144 data\coco_images\ipynb_checkpoints\00000196785.jpg
145 data\coco_images\ipynb_checkpoints\00000196798.jpg
146 data\coco_images\ipynb_checkpoints\00000196811.jpg
147 data\coco_images\ipynb_checkpoints\00000196842.jpg
148 data\coco_images\ipynb_checkpoints\00000204800.jpg
149 data\coco_images\ipynb_checkpoints\00000204825.jpg
150 data\coco_images\ipynb_checkpoints\00000204887.jpg
151 data\coco_images\ipynb_checkpoints\00000204889.jpg
152 data\coco_images\ipynb_checkpoints\00000213215.jpg
153 data\coco_images\ipynb_checkpoints\00000221311.jpg
154 data\coco_images\ipynb_checkpoints\00000229387.jpg
155 data\coco_images\ipynb_checkpoints\00000229391.jpg
156 data\coco_images\ipynb_checkpoints\00000229419.jpg
157 data\coco_images\ipynb_checkpoints\00000229472.jpg
158 data\coco_images\ipynb_checkpoints\00000229478.jpg
159 data\coco_images\ipynb_checkpoints\00000229494.jpg
160 data\coco_images\ipynb_checkpoints\00000229500.jpg
161 data\coco_images\ipynb_checkpoints\00000229507.jpg
162 data\coco_images\ipynb_checkpoints\00000229511.jpg
163 data\coco_images\ipynb_checkpoints\00000229522.jpg
164 data\coco_images\ipynb_checkpoints\00000229530.jpg
165 data\coco_images\ipynb_checkpoints\00000229559.jpg
166 data\coco_images\ipynb_checkpoints\00000229564.jpg
167 data\coco_images\ipynb_checkpoints\00000237611.jpg
168 data\coco_images\ipynb_checkpoints\00000237643.jpg
169 data\coco_images\ipynb_checkpoints\00000237658.jpg
170 data\coco_images\ipynb_checkpoints\00000237764.jpg
171 data\coco_images\ipynb_checkpoints\00000237772.jpg
172 data\coco_images\ipynb_checkpoints\00000245842.jpg
173 data\coco_images\ipynb_checkpoints\00000245873.jpg
174 data\coco_images\ipynb_checkpoints\00000253965.jpg
175 data\coco_images\ipynb_checkpoints\00000253975.jpg
176 data\coco_images\ipynb_checkpoints\00000253981.jpg
177 data\coco_images\ipynb_checkpoints\00000253988.jpg
178 data\coco_images\ipynb_checkpoints\00000254045.jpg
179 data\coco_images\ipynb_checkpoints\00000254050.jpg
180 data\coco_images\ipynb_checkpoints\00000254060.jpg
181 data\coco_images\ipynb_checkpoints\00000254074.jpg
182 data\coco_images\ipynb_checkpoints\00000254132.jpg
183 data\coco_images\ipynb_checkpoints\00000254138.jpg
184 data\coco_images\ipynb_checkpoints\00000254139.jpg
185 data\coco_images\ipynb_checkpoints\00000262145.jpg
186 data\coco_images\ipynb_checkpoints\00000262146.jpg
187 data\coco_images\ipynb_checkpoints\00000262148.jpg
188 data\coco_images\ipynb_checkpoints\00000262161.jpg
189 data\coco_images\ipynb_checkpoints\00000262171.jpg
190 data\coco_images\ipynb_checkpoints\00000262175.jpg
191 data\coco_images\ipynb_checkpoints\00000262191.jpg
192 data\coco_images\ipynb_checkpoints\00000262197.jpg
193 data\coco_images\ipynb_checkpoints\00000262200.jpg
194 data\coco_images\ipynb_checkpoints\00000262207.jpg
195 data\coco_images\ipynb_checkpoints\00000262221.jpg
196 data\coco_images\ipynb_checkpoints\00000262228.jpg
197 data\coco_images\ipynb_checkpoints\00000262229.jpg
198 data\coco_images\ipynb_checkpoints\00000262235.jpg
199 data\coco_images\ipynb_checkpoints\00000262238.jpg
200 data\coco_images\ipynb_checkpoints\00000262239.jpg
201 data\coco_images\ipynb_checkpoints\00000262242.jpg
202 data\coco_images\ipynb_checkpoints\00000262260.jpg
203 data\coco_images\ipynb_checkpoints\00000262273.jpg
204 data\coco_images\ipynb_checkpoints\00000262274.jpg

```
274  data/coco_images\.ipynb_checkpoints\00000360510.jpg
275  data/coco_images\.ipynb_checkpoints\00000360528.jpg
276  data/coco_images\.ipynb_checkpoints\00000360570.jpg
277  data/coco_images\.ipynb_checkpoints\00000360595.jpg
278  data/coco_images\.ipynb_checkpoints\00000360604.jpg
279  data/coco_images\.ipynb_checkpoints\00000360624.jpg
280  data/coco_images\.ipynb_checkpoints\00000368676.jpg
281  data/coco_images\.ipynb_checkpoints\00000368702.jpg
282  data/coco_images\.ipynb_checkpoints\00000368852.jpg
283  data/coco_images\.ipynb_checkpoints\00000376868.jpg
284  data/coco_images\.ipynb_checkpoints\00000376870.jpg
285  data/coco_images\.ipynb_checkpoints\00000376984.jpg
286  data/coco_images\.ipynb_checkpoints\00000377005.jpg
287  data/coco_images\.ipynb_checkpoints\00000377008.jpg
288  data/coco_images\.ipynb_checkpoints\00000377012.jpg
289  data/coco_images\.ipynb_checkpoints\00000377044.jpg
290  data/coco_images\.ipynb_checkpoints\00000385066.jpg
291  data/coco_images\.ipynb_checkpoints\00000385078.jpg
292  data/coco_images\.ipynb_checkpoints\00000385139.jpg
293  data/coco_images\.ipynb_checkpoints\00000385145.jpg
294  data/coco_images\.ipynb_checkpoints\00000385168.jpg
295  data/coco_images\.ipynb_checkpoints\00000385186.jpg
296  data/coco_images\.ipynb_checkpoints\00000385204.jpg
297  data/coco_images\.ipynb_checkpoints\00000393223.jpg
298  data/coco_images\.ipynb_checkpoints\00000393224.jpg
299  data/coco_images\.ipynb_checkpoints\00000393227.jpg
```

```
----- H repeater εκτελέστηκε για 1η φορά... : -----
----- H repeater εκτελέστηκε για 2η φορά... : -----
----- H repeater εκτελέστηκε για 3η φορά... : -----
----- H repeater εκτελέστηκε για 4η φορά... : -----
----- H repeater εκτελέστηκε για 5η φορά... : -----
----- H repeater εκτελέστηκε για 6η φορά... : -----
----- H repeater εκτελέστηκε για 7η φορά... : -----
Για Target Image τnv 152 και σχετικές εικόνες:
[152, 5.2780975068250955]
[63, 3.5653993063519245]
[289, 0.5876841982835186]
[245, 0.07092374890105063]
```

```
Για Target Image τnv 259 και σχετικές εικόνες:
[259, 5.278097506825095]
[47, 2.989993886173021]
[169, 0.06475874951143833]
[195, 0.04318405702202762]
```

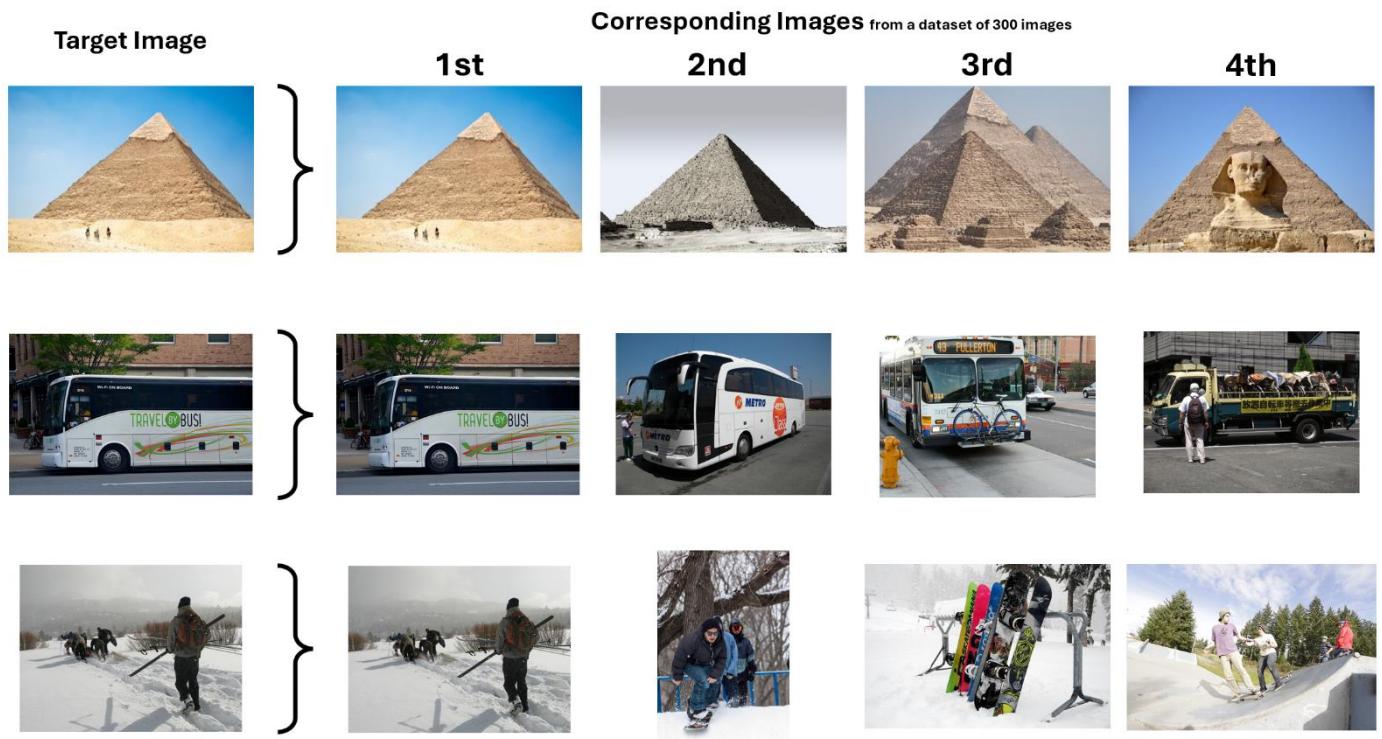
```
Για Target Image τnv 12 και σχετικές εικόνες:
[12, 4.358288668700613]
[33, 1.4120776602962206]
[185, 0.73636194688539]
[55, 0.1945723220835704]
```

```
Για Target Image τnv 208 και σχετικές εικόνες:
[208, 5.278097506825095]
[3, 3.5109443163418814]
[68, 0.5876841982835185]
[243, 0.022953371704799202]
```

Οι βαθμολογίες των target images είναι:

Για τnv Target Image 152: 2.375526190090397
Για τnv Target Image 259: 2.0940085498828958
Για τnv Target Image 12: 1.6753251494914485
Για τnv Target Image 208: 2.3499198482888235
Total score is: 2.123694934438391

Ακολουθούν αποτελέσματα του προγράμματος με την αναπαράσταση εικόνων :



To dataset των συνολικά 300 εικόνων που χρησιμοποιείται στην παρούσα δοκιμαστική εκτέλεση διαθέτει φωτογραφίες από τις εξής κατηγορίες:

- 1) Πυραμίδες – Αίγυπτος
- 2) Γάτες
- 3) Σκύλοι
- 4) Άνθρωποι (σε δραστηριότητες)
- 5) Ποδήλατα
- 6) Οχήματα