ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ

ΑΝΑΛΥΣΗ ΕΙΚΟΝΑΣ

Ακαδημαϊκό έτος 2022-2023

*ΒΙΤΑΚΗΣ ΑΘΑΝΑΣΙΟΣ - **Π19247***

*ΑΥΓΕΡΙΝΟΣ ΧΡΗΣΤΟΣ - **Π19020***

*ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ - **Π19130***

Contents

Αναλυτική περιγραφή αλγόριθμου 3
Προγραμματιστική υλοποίηση 5
Διεξαγωγή χαρακτηριστικών μέσω resnet18 11
Εκτέλεση κώδικα 12
Αποτέλεσμα 13
Relevance score for each image 13
Μέτρηση ακρίβειας αλγορίθμου 15
Υλοποίηση 15
Αποτέλεσμα 15

Αναλυτική περιγραφή αλγόριθμου

Το άρθρο " **Multimedia Retrieval through Unsupervised Hypergraph-based Manifold Ranking**", IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 28, NO. 12, DECEMBER 2019 " παρουσιάζει μια νέα μέθοδο για την ανάκτηση πολυμέσων που βασίζεται στο unsupervised hypergraph-based manifold ranking. Η προτεινόμενη μέθοδος είναι σε θέση να χειρίζεται διάφορους τύπους δεδομένων πολυμέσων, όπως εικόνες, βίντεο και σήματα ήχου, και είναι σε θέση να ανακτά αποτελεσματικά σχετικά αποτελέσματα για ένα δεδομένο ερώτημα.

Η προτεινόμενη μέθοδος λειτουργεί κατασκευάζοντας πρώτα έναν υπεργράφο που αναπαριστά τις σχέσεις μεταξύ διαφορετικών σημείων δεδομένων πολυμέσων. Κάθε σημείο δεδομένων αναπαρίσταται ως κόμβος στο υπεργράφημα και οι ακμές σταθμίζονται με βάση την ομοιότητα μεταξύ των σημείων δεδομένων. Το υπεργράφημα στη συνέχεια μετατρέπεται σε γράφημα εφαρμόζοντας τον μετασχηματισμό Hyperedge-to-Node, ο οποίος συνδυάζει τους κόμβους που συνδέονται από τον ίδιο υπερακμή σε έναν μόνο κόμβο.

Στη συνέχεια, η μέθοδος υπολογίζει τον πίνακα συνάφειας(**relevance**) με βάση το γράφημα Laplacian του μετασχηματισμένου γραφήματος. Ο πίνακας συνάφειας χρησιμοποιείται στη συνέχεια για τον υπολογισμό της πολλαπλής βαθμολογίας κατάταξης για κάθε σημείο δεδομένων, η οποία αντικατοπτρίζει τη συνάφειά του με το ερώτημα. Οι βαθμολογίες κατάταξης λαμβάνονται με την επίλυση ενός συνόλου γραμμικών εξισώσεων χρησιμοποιώντας τη μέθοδο power iteration.

Συνοπτικά ακολουθείτε ο εξής αλγόριθμος:
1. Κανονικοποίηση Σειράς Κατάταξης (**Rank Normalization**)
2. Κατασκευή Υπεργράφου (**Hypergraph Construction**)
3. Υπολογισμός Ομοιότητας Υπερακμών (**Hyperedge Similarities**)
4. Υπολογισμός Καρτεσιανού Γινομένου μεταξύ των στοιχείων των Υπερακμών (**Cartesian Product of Hyperedge Elements**)
5. Υπολογισμός Ομοιότητας βάσει του κατασκευασμένου Υπεργράφου (**Hypergraph – Based Similarity**)

Αντίστοιχα και εμείς ακολουθήσαμε τα βασικά βήματα της αλγοριθμικής προσέγγισης και κατασκευάσαμε τον παρακάτω αλγόριθμο:

1. Import necessary libraries: **itertools**, **torch**, **transforms**, **numpy**, **rankdata**, **random**, **vutils**, and **os**.
2. Load a pre-trained ResNet18 model using **models.resnet18(pretrained=True)**.
3. Set the model to evaluation mode with **model.eval()**.
4. Define a set of image transforms using **transforms.Compose** to resize and normalize the input images.
5. Load an image dataset using **ImageFolder** and apply the image transforms defined earlier to the dataset.
6. Retrieve the class names from the dataset using **dataset.classes**.
7. Select a random subset of the dataset and store its indices in **subset_indices**.
8. Retrieve the feature vectors for all images in the subset by passing each image through the ResNet18 model and appending the resulting feature vector to the list **features**.
9. Convert the list of feature vectors to a numpy array **features**.
10. ***Normalize the feature vectors by rank*** using **np.apply_along_axis(rankdata, axis=1, arr=features, method='average') / len(features)**.
11. ***Construct a hypergraph*** by computing the similarity matrix and adjacency matrix.
12. Compute the ***hyperedge similarities.***
13. Compute the **relevance** score for each image by iterating through the ***hyperedge elements and their combinations***.
14. Sort the images by relevance score in descending order.
15. Compute the relevance score for each base image.
16. Sort the base images by relevance score in descending order.
17. Compute the accuracy of the model by comparing the class of each base image with the class of the most relevant image.
18. Load the images and their relevance scores using **vutils.make_grid**.
19. Save the images and their relevance scores to a file using **vutils.save_image**.
20. Display the images and their relevance scores using **os.startfile**.

Προγραμματιστική υλοποίηση

Ο κώδικας φορτώνει ένα προεκπαιδευμένο μοντέλο ResNet18 από το torchvision.models, το οποίο είναι ένα μοντέλο βαθιού νευρωνικού δικτύου που μπορεί να χρησιμοποιηθεί για ταξινόμηση εικόνων. Το μοντέλο έχει ρυθμιστεί σε λειτουργία αξιολόγησης χρησιμοποιώντας model.eval(), το οποίο απενεργοποιεί τα επίπεδα κανονικοποίησης εγκατάλειψης και παρτίδας στο μοντέλο.

Επίσης ορίζουμε έναν μετασχηματισμό που χρησιμοποιείται για την προεπεξεργασία της εικόνας εισόδου. Αυτός ο μετασχηματισμός αλλάζει το μέγεθος της εικόνας σε 256x256 pixel, περικόπτει την κεντρική περιοχή μεγέθους 224x224 pixel, μετατρέπει την εικόνα σε tensor PyTorch και κανονικοποιεί την εικόνα χρησιμοποιώντας τις τιμές μέσης και τυπικής απόκλισης που παρέχονται. Οι μέσες τιμές και οι τιμές std που χρησιμοποιούνται για την κανονικοποίηση είναι τυπικές τιμές που χρησιμοποιούνται για προεκπαιδευμένα μοντέλα.

```python
import itertools

import torch
from torchvision import models, transforms
import numpy as np
from scipy.stats import rankdata
import random
import torchvision.utils as vutils
import os

# Load ResNet18 model
from torchvision.datasets import ImageFolder

model = models.resnet18(pretrained=True)

# Set model to evaluation mode
model.eval()

# Define the transform to preprocess the input image
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

```

Έπειτα φορτώνει ένα σύνολο δεδομένων εικόνας από τον κατάλογο 'images\Images\' και εφαρμόζει έναν μετασχηματισμό για την προεπεξεργασία των εικόνων. Στη συνέχεια επιλέγει ένα τυχαίο υποσύνολο του συνόλου δεδομένων με καθορισμένο μέγεθος 500 εικόνων. Τα ονόματα κλάσεων για τις εικόνες εξάγονται από το σύνολο δεδομένων. Στη συνέχεια, ο κώδικας επιλέγει μερικές βασικές εικόνες που θα επισημανθούν ως εικόνες στόχου, προσδιορίζοντας τους δείκτες τους στο υποσύνολο.

Τα διανύσματα χαρακτηριστικών για όλες τις εικόνες στο υποσύνολο υπολογίζονται χρησιμοποιώντας το μοντέλο ResNet18. Για κάθε εικόνα στο υποσύνολο, ο κώδικας προσθέτει μια διάσταση παρτίδας στον tensor εικόνας και τον περνά μέσα από το μοντέλο ResNet18. Το προκύπτον διάνυσμα χαρακτηριστικών στη συνέχεια προσαρτάται σε μια λίστα διανυσμάτων χαρακτηριστικών. Τέλος, η λίστα των διανυσμάτων χαρακτηριστικών μετατρέπεται σε έναν numpy πίνακα.

```
# Load the image dataset and apply the transform
dataset = ImageFolder('images\\Images', transform=transform)

# Get the class names from the image dataset
class_names = dataset.classes

# Print the class names
print(class_names)

# Select a random subset of the dataset
subset_size = 3000
subset_indices = random.sample(range(len(dataset)), subset_size)
subset = torch.utils.data.Subset(dataset, subset_indices)

# Select some base images to mark as target images
target_indices = [10, 15, 27, 30]

# Compute the feature vectors for all images in the dataset using ResNet18
features = []
for image, _ in subset:
    # Add batch dimension to image tensor
    image = image.unsqueeze(0)

    # Pass the image through the ResNet18 model
    with torch.no_grad():
        feature = model(image).squeeze().numpy()

    # Append the feature vector to the list of features
    features.append(feature)

```

Το παρακάτω μπλοκ κώδικα υλοποιεί την κατασκευή ενός υπεργράφου με βάση τα διανύσματα χαρακτηριστικών που υπολογίστηκαν στο προηγούμενο βήμα.
Πρώτον, τα διανύσματα χαρακτηριστικών κανονικοποιούνται ανά κατάταξη χρησιμοποιώντας τη συνάρτηση rankdata() από το scipy.stats. Αυτή η συνάρτηση εκχωρεί τάξεις σε κάθε διάνυσμα χαρακτηριστικών σε όλες τις διαστάσεις και, στη συνέχεια, οι προκύπτουσες τάξεις διαιρούνται με τον συνολικό αριθμό χαρακτηριστικών για να ληφθούν κανονικοποιημένες τάξεις.
Στη συνέχεια, το υπεργράφημα κατασκευάζεται με τον υπολογισμό του πίνακα ομοιότητας μεταξύ ζευγών διανυσμάτων χαρακτηριστικών, χρησιμοποιώντας το dot product των διανυσμάτων χαρακτηριστικών που κανονικοποιούνται κατά σειρά διαιρούμενο με το εξωτερικό γινόμενο των κανόνων L2 τους. Αυτός ο πίνακας ομοιότητας ορίζεται στη συνέχεια ορίζοντας όλες τις τιμές κάτω από τη διάμεση ομοιότητα σε μηδέν και όλες τις άλλες σε ένα, με αποτέλεσμα έναν πίνακα γειτνίασης. Η διαγώνιος της μήτρας γειτνίασης έχει μηδενιστεί για την αποφυγή αυτο-βρόχων.
Στη συνέχεια, οι ομοιότητες υπεράκρων υπολογίζονται με επανάληψη σε όλα τα ζεύγη κόμβων που μοιράζονται έναν κοινό γείτονα στον πίνακα γειτνίασης. Για κάθε ζεύγος κόμβων, υπολογίζεται ένα βάρος με βάση το γινόμενο των ομοιοτήτων τους με τον κοινό γείτονα και το βάρος προστίθεται στις αντίστοιχες εγγραφές στον πίνακα ομοιότητας υπεράκρων.
Τέλος, όλες οι εικόνες στο υποσύνολο επιλέγονται δημιουργώντας μια λίστα δεικτών από 0 έως len(subset)-1.

```
# Convert the list of features to a numpy array
features = np.array(features)

# Normalize the features by rank
rank_features = np.apply_along_axis(rankdata, axis=1, arr=features, method='average') / len(features)

# Construct the hypergraph
n, d = rank_features.shape
similarity_matrix = np.dot(rank_features, rank_features.T) / np.outer(np.linalg.norm(rank_features, axis=1),
                                                                      np.linalg.norm(rank_features, axis=1))
similarity_matrix = np.maximum(similarity_matrix, similarity_matrix.T)

threshold = np.median(similarity_matrix)
adj_matrix = (similarity_matrix >= threshold).astype(float)
np.fill_diagonal(adj_matrix, 0)

# Compute the hyperedge similarities
hyperedge_similarities = np.zeros((n, n))
for i in range(n):
    for j in range(i + 1, n):
        if adj_matrix[i, j] == 1:
            common_neighbors = np.intersect1d(np.where(adj_matrix[i, :] == 1), np.where(adj_matrix[j, :] == 1))
            if len(common_neighbors) > 0:
                for k in common_neighbors:
                    weight = np.sqrt(similarity_matrix[i, k] * similarity_matrix[j, k])
                    hyperedge_similarities[i, j] += weight
                    hyperedge_similarities[j, i] += weight
# Select all images in the subset
subset_indices = list(range(len(subset)))
```

Μετά υπολογίζει τη βαθμολογία συνάφειας για κάθε εικόνα στο υποσύνολο και κάθε βασική εικόνα (επιλεγμένη ως στόχοι). Αρχικά κάνει βρόχο σε κάθε εικόνα στο υποσύνολο και υπολογίζει τη βαθμολογία συνάφειάς της. Η βαθμολογία συνάφειας υπολογίζεται επαναλαμβάνοντας όλα τα ζεύγη εικόνων που μοιράζονται τουλάχιστον ένα υπεράκρο με τη δεδομένη εικόνα και αθροίζοντας το γινόμενο των ομοιοτήτων υπεράκρων τους. Η βαθμολογία συνάφειας που προκύπτει αποθηκεύεται σε μια λίστα πλειάδων που περιέχει το ευρετήριο εικόνας και τη βαθμολογία συνάφειάς του.

Στη συνέχεια, η λίστα των εικόνων στο υποσύνολο ταξινομείται με φθίνουσα σειρά με βάση τη βαθμολογία συνάφειάς τους. Η ταξινομημένη λίστα εκτυπώνεται στην κονσόλα, εμφανίζοντας το ευρετήριο εικόνας και τη βαθμολογία συνάφειάς του.

```
90  # Compute the relevance score for each image
91  image_scores = []
92  for image_idx in subset_indices:
93      hyperedge_elements = np.where(adj_matrix[image_idx, :] == 1)[0]
94      relevance_score = 0
95      # Calculates Cartesian product of the lists contained in edge lists
96      for i, j in itertools.product(hyperedge_elements, repeat=2):
97          if i != j and adj_matrix[i, j] == 1:
98              relevance_score += hyperedge_similarities[i, j]
99      image_scores.append((image_idx, relevance_score))
100
101 # Sort the images by relevance score in descending order
102 sorted_images = sorted(image_scores, key=lambda x: x[1], reverse=True)
103
104 # Print the sorted list of images and their relevance scores
105 for image in sorted_images:
106     print(f"Image index: {image[0]}, Relevance score: {image[1]}")
107
108
109 # Compute the relevance score for each base image
110 target_scores = []
111 for target_idx in target_indices:
112     hyperedge_elements = np.where(adj_matrix[target_idx, :] == 1)[0]
113     relevance_score = 0
114     for i, j in itertools.product(hyperedge_elements, repeat=2):
115         if i != j and adj_matrix[i, j] == 1:
116             relevance_score += hyperedge_similarities[i, j]
117     target_scores.append((target_idx, relevance_score))
118
```

Αυτός ο κώδικας υπολογίζει την ακρίβεια του αλγορίθμου στην αντιστοίχιση των πιο σχετικών εικόνων με τις εικόνες-στόχους.

Ξεκινά με την προετοιμασία μιας μεταβλητής correct_count για να παρακολουθείτε πόσες φορές η πιο σχετική εικόνα έχει την ίδια κλάση με την εικόνα προορισμού. Στη συνέχεια, κάνει βρόχους πάνω από κάθε εικόνα στόχο και την αντίστοιχη πιο σχετική εικόνα, οι οποίες ταξινομήθηκαν κατά βαθμολογία συνάφειας με φθίνουσα σειρά.

Για κάθε ζεύγος στόχων και σχετικών εικόνων, εξάγει τις ετικέτες κλάσεων από το αντικείμενο δεδομένων χρησιμοποιώντας τους δείκτες τους. Εάν οι δύο εικόνες έχουν την ίδια ετικέτα κλάσης, η μεταβλητή correct_count αυξάνεται.

Τέλος, η ακρίβεια υπολογίζεται ως η αναλογία του correct_count προς τον συνολικό αριθμό των εικόνων-στόχων και εκτυπώνεται

```
# Print the sorted list of base images and their relevance scores
for target in sorted_targets:
    print(f"Image index: {target[0]}, Relevance score: {target[1]}")

# Compute accuracy
'''This modification checks if the class of the target image
 is the same as the class of the most relevant image, and only
 increments the correct count if they match. The final accuracy
 is computed as the ratio of correct predictions to the total number of target images.'''
correct_count = 0
for target, (idx, score) in zip(target_indices, sorted_targets):
    # Get the class of the target image
    target_class = dataset.classes[target]

    # Get the class of the most relevant image
    relevant_class = dataset.classes[idx]

    if target_class == relevant_class:
        print(target_class)
        print(relevant_class)
        correct_count += 1

accuracy = correct_count / len(target_indices)
print(f"Accuracy: {accuracy}")


# Load the images and their relevance scores
'''This will open a separate window to display each image,
 along with the corresponding relevance score for the base images.'''
target1_idx = sorted_targets[0][0]
```

Εμφάνιση κάθε εικόνας, μαζί με την αντίστοιχη βαθμολογία συνάφειας για τις βασικές εικόνες.

```
target2_idx = sorted_targets[1][0]
target3_idx = sorted_targets[2][0]
target4_idx = sorted_targets[3][0]


best_relevance_idx1 = np.argmax(hyperedge_similarities[target1_idx])
best_relevance_idx2 = np.argmax(hyperedge_similarities[target2_idx])
best_relevance_idx3 = np.argmax(hyperedge_similarities[target3_idx])
best_relevance_idx4 = np.argmax(hyperedge_similarities[target4_idx])

# Load the base images
base_image1, _ = subset[target1_idx]
base_image2, _ = subset[target2_idx]
base_image3, _ = subset[target3_idx]
base_image4, _ = subset[target4_idx]

# Load the best relevance score image
best_relevance_image1, _ = subset[best_relevance_idx1]
best_relevance_image2, _ = subset[best_relevance_idx2]
best_relevance_image3, _ = subset[best_relevance_idx3]
best_relevance_image4, _ = subset[best_relevance_idx4]

# Display the images
images = [base_image1, base_image2, base_image3, base_image4, best_relevance_image1,
          best_relevance_image2, best_relevance_image3, best_relevance_image4]
grid = vutils.make_grid(images, nrow=4, padding=5, normalize=True)
vutils.save_image(grid, 'images.png')

# Open the saved image file
os.startfile('images.png')
```

Διεξαγωγή χαρακτηριστικών μέσω resnet18

```
# Load ResNet18 model
from torchvision.datasets import ImageFolder

model = models.resnet18(pretrained=True)

# Set model to evaluation mode
model.eval()

```

```
# Load the image dataset and apply the transform
dataset = ImageFolder('images\\Images', transform=transform)
# Select a random subset of the dataset
subset_size = 750
subset_indices = random.sample(range(len(dataset)), subset_size)
subset = torch.utils.data.Subset(dataset, subset_indices)
```

```
# Compute the feature vectors for all images in the dataset using ResNet18
features = []
for image, _ in subset:
    # Add batch dimension to image tensor
    image = image.unsqueeze(0)

    # Pass the image through the ResNet18 model
    with torch.no_grad():
        feature = model(image).squeeze().numpy()

    # Append the feature vector to the list of features
    features.append(feature)

# Convert the list of features to a numpy array
features = np.array(features)

```

Αυτός ο κώδικας υπολογίζει τα διανύσματα χαρακτηριστικών για όλες τις εικόνες σε ένα σύνολο δεδομένων χρησιμοποιώντας το ResNet18, το οποίο είναι ένα προεκπαιδευμένο μοντέλο βαθιάς εκμάθησης για ταξινόμηση εικόνων. Τα διανύσματα χαρακτηριστικών υπολογίζονται περνώντας κάθε εικόνα μέσω του μοντέλου ResNet18 και εξάγοντας την έξοδο του τελευταίου πλήρως συνδεδεμένου στρώματος πριν από το επίπεδο ταξινόμησης. Αυτά τα διανύσματα χαρακτηριστικών αντιπροσωπεύουν μια αφηρημένη αναπαράσταση υψηλού επιπέδου του περιεχομένου κάθε εικόνας που μπορεί να χρησιμοποιηθεί για την ανάκτηση εικόνων. Τα διανύσματα χαρακτηριστικών αποθηκεύονται σε έναν numpy πίνακα για περαιτέρω επεξεργασία.

Εκτέλεση κώδικα

Θέτουμε κάποιες από τις εικόνες της βάσης ως εικόνες στόχο (**target images**)

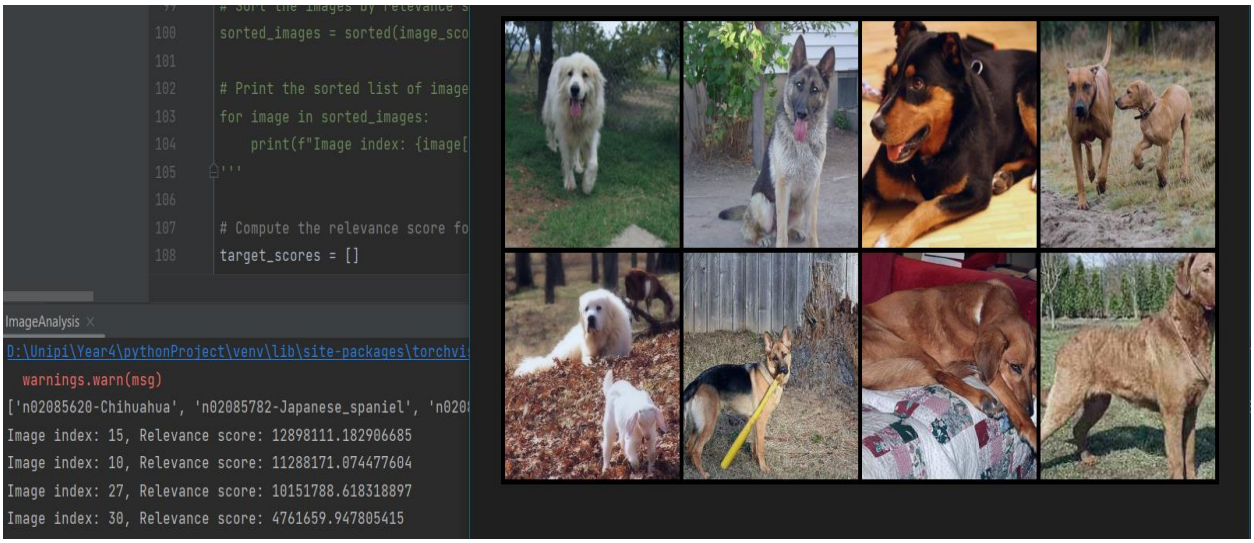
```
42  # Select some base images to mark as target images
43  target_indices = [10, 15, 27, 30]
```

```
107 # Compute the relevance score for each base image
108 target_scores = []
109 for target_idx in target_indices:
110     hyperedge_elements = np.where(adj_matrix[target_idx, :] == 1)[0]
111     relevance_score = 0
112     for i, j in itertools.product(hyperedge_elements, repeat=2):
113         if i != j and adj_matrix[i, j] == 1:
114             relevance_score += hyperedge_similarities[i, j]
115     target_scores.append((target_idx, relevance_score))
116
117 # Sort the base images by relevance score in descending order
118 sorted_targets = sorted(target_scores, key=lambda x: x[1], reverse=True)
119
120 # Print the sorted list of base images and their relevance scores
121 for target in sorted_targets:
122     print(f"Image index: {target[0]}, Relevance score: {target[1]}")
123
```

Ο παραπάνω κώδικας υπολογίζει τη βαθμολογία συνάφειας για κάθε βασική εικόνα σε έναν αλγόριθμο κατάταξης πολλαπλών γραφημάτων(hypergraph-based manifold ranking). Ο αλγόριθμος κατάταξης πολλαπλών μορφών που βασίζεται σε υπεργράφο(hypergraph) εκχωρεί βαθμολογίες συνάφειας(relevance score) στις εικόνες με βάση την ομοιότητά τους με την εικόνα ερωτήματος(target image). Ο κώδικας επαναλαμβάνεται πρώτα μέσω των ευρετηρίων των εικόνων προορισμού και λαμβάνει τους δείκτες των στοιχείων υπεράκρων (δηλαδή, εικόνες που συνδέονται με την εικόνα προορισμού). Στη συνέχεια, υπολογίζει τη βαθμολογία συνάφειας για κάθε εικόνα στόχο ως το άθροισμα των ομοιοτήτων υπεράκρων μεταξύ όλων των ζευγών στοιχείων υπεράκρων. Στη συνέχεια, οι βαθμολογίες συνάφειας για όλες τις εικόνες-στόχους ταξινομούνται με φθίνουσα σειρά και εκτυπώνεται η ταξινομημένη λίστα, όπου εμφανίζεται ο δείκτης κάθε εικόνας στόχου και η αντίστοιχη βαθμολογία συνάφειας.

Αποτέλεσμα

Στην παρακάτω φωτογραφία βλέπουμε στις πάνω 4 εικόνες τα target images που θέσαμε και από κάτω τις best Relevance αντιστοιχίες τους. Επίσης το Relevance score κάθε αντιστοιχίας εμφανίζεται στην κονσόλα.



Relevance score for each image

Αντίστοιχα μπορούμε να βρούμε το relevance κάθε εικόνας σε σχέση με κάθε εικόνα θέτοντας σαν target image όλο το dataset(subset_indices)

```
# Compute the relevance score for each image
image_scores = []
for image_idx in subset_indices:
    hyperedge_elements = np.where(adj_matrix[image_idx, :] == 1)[0]
    relevance_score = 0
    # Calculates Cartesian product of the lists contained in edge lists
    for i, j in itertools.product(hyperedge_elements, repeat=2):
        if i != j and adj_matrix[i, j] == 1:
            relevance_score += hyperedge_similarities[i, j]
    image_scores.append((image_idx, relevance_score))

# Sort the images by relevance score in descending order
sorted_images = sorted(image_scores, key=lambda x: x[1], reverse=True)

# Print the sorted list of images and their relevance scores
for image in sorted_images:
    print(f"Image index: {image[0]}, Relevance score: {image[1]}")
```

```
Image index: 17, Relevance score: 11213.865255570447
Image index: 39, Relevance score: 10218.669554402668
Image index: 41, Relevance score: 9766.681710245384
Image index: 43, Relevance score: 9592.97514188329
Image index: 35, Relevance score: 9575.303615867757
Image index: 34, Relevance score: 9197.122868907742
Image index: 40, Relevance score: 9081.79762952345
Image index: 42, Relevance score: 8250.960360203371
Image index: 20, Relevance score: 8071.127184148616
Image index: 28, Relevance score: 8027.550279234002
Image index: 5, Relevance score: 7820.557023289375
Image index: 15, Relevance score: 7646.081492483452
Image index: 33, Relevance score: 7580.1585707066
Image index: 46, Relevance score: 7532.079809273635
Image index: 26, Relevance score: 7277.085835722612
Image index: 36, Relevance score: 7204.478751456016
Image index: 25, Relevance score: 7156.527771300713
Image index: 44, Relevance score: 7098.876708948507
Image index: 21, Relevance score: 7057.431711406669
Image index: 13, Relevance score: 7020.713639884829
```

```
Image index: 19, Relevance score: 6711.842323345775
Image index: 2, Relevance score: 6664.944424404308
Image index: 38, Relevance score: 6174.922055469968
Image index: 3, Relevance score: 5816.187147741542
Image index: 6, Relevance score: 5489.520897802633
Image index: 32, Relevance score: 5317.220918710037
Image index: 8, Relevance score: 5082.177936823635
Image index: 31, Relevance score: 4611.097809532209
Image index: 45, Relevance score: 4522.897922886136
Image index: 24, Relevance score: 4313.152534134524
Image index: 23, Relevance score: 4258.514495228891
Image index: 7, Relevance score: 3917.1806977544534
Image index: 48, Relevance score: 3633.2321432817675
Image index: 22, Relevance score: 3584.8528462305294
Image index: 18, Relevance score: 3374.434079629711
Image index: 29, Relevance score: 3325.3411821156183
Image index: 47, Relevance score: 3316.661897158278
Image index: 9, Relevance score: 2954.7162609130773
Image index: 1, Relevance score: 2846.36728895619
Image index: 37, Relevance score: 2623.9752199229106
```

Μέτρηση ακρίβειας αλγορίθμου

Ένας απλός τρόπος που σκεφτήκαμε για μετράμε την ακρίβεια είναι αρχικά να ελέγχει εάν η κλάση της εικόνας στόχου είναι η ίδια με την κατηγορία της πιο σχετικής εικόνας με βάση τη βαθμολογία συνάφειας που υπολογίστηκε νωρίτερα. Εάν ταιριάζουν, αυξάνει τη σωστή μέτρηση. Τέλος, η ακρίβεια υπολογίζεται διαιρώντας τη σωστή μέτρηση με τον συνολικό αριθμό των εικόνων-στόχων. Τέλος η ακρίβεια να εκτυπώνεται στην κονσόλα.

Υλοποίηση

```
# Compute accuracy
'''This modification checks if the class of the target image
is the same as the class of the most relevant image, and only increments the correct count if they match.
The final accuracy is computed as the ratio of correct predictions to the total number of target images.'''
correct_count = 0
for target, (idx, score) in zip(target_indices, sorted_targets):
    # Get the class of the target image
    target_class = dataset.classes[target]

    # Get the class of the most relevant image
    relevant_class = dataset.classes[idx]

    if target_class == relevant_class:
        print(target_class)
        print(relevant_class)
        correct_count += 1

accuracy = correct_count / len(target_indices)
print(f"Accuracy: {accuracy}")
```

Αποτέλεσμα

Στο παρακάτω παράδειγμα βλέπουμε ότι ορθά έβγαλε 75% ακεραιότητα καθώς τα το ένα ταίρι από τα τέσσερα είναι διαφορετική κλάση(ράτσα).Επίσης παρατηρούμε ότι το λανθασμένο ταίρι έχει το χαμηλότερο Relevance score.