ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάθημα Προπτυχιακών Σπουδών:
Ανάλυση Εικόνας
Ακαδημαϊκό έτος: 2023 - 2024
Εξάμηνο: 7ο
Απαλλακτική Εργασία

Ομάδα Εργασίας:	**Υπεύθυνος Καθηγητής:**
Θεόδωρος Κοξάνογλου Π20094 Απόστολος Σιαμπάνης Π20173 Αλέξανδρος Χόλης Π20217	Διονύσιος Σωτηρόπουλος Γιώργος Τσιχριτζής

Περιεχόμενα

Εκφώνηση

ΑΝΑΚΤΗΣΗ ΕΙΚΟΝΩΝ ΜΕ ΒΑΣΗ ΤΟ ΠΕΡΙΕΧΟΜΕΝΟ

(**Content – Based Image Retrieval**)

Στόχος της συγκεκριμένης υπολογιστικής εργασίας είναι ανάπτυξη γραφοθεωρητικών αλγορίθμων για την ανάκτηση εικόνων με βάση το περιεχόμενο. Τα βασικά βήματα της προτεινόμενης αλγοριθμικής προσέγγισης έχουν ως ακολούθως:

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**)

Λεπτομερής περιγραφή της παραπάνω αλγοριθμικής διαδικασίας μπορείτε να βρείτε στο άρθρο με τίτλο "**Multimedia Retrieval through Unsupervised Hypergraph-based Manifold Ranking**", IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 28, NO. 12, DECEMBER 2019. Το άρθρο θα αναρτηθεί στα έγγραφα του μαθήματος.

Ζητούμενα:

1. Να παρουσιάσετε μια αναλυτική περιγραφή της υπολογιστικής διαδικασίας που παρουσιάζεται στο άρθρο ενσωματώνοντάς την στην τεκμηρίωση της εργασίας σας. (**20% του συνολικού βαθμού**)
2. Να αναπτύξετε κώδικα σε Matlab ή Python για την προγραμματιστική υλοποίηση των παραπάνω υπολογιστικών βημάτων. (**20% του συνολικού βαθμού**)
3. Το σύνολο των αντικειμενικών χαρακτηριστικών για την διανυσματική αναπαράσταση της κάθε εικόνας να εξαχθεί με την χρήση ενός προεκπαιδευμένου νευρωνικού δικτύου (**squeezenet**, **googlenet**, **resnet18**, **resnet50**, **resnet101**, κλπ.). Συγκεκριμένα, μπορείτε να χρησιμοποιήσετε ως αντικειμενικά χαρακτηριστικά της κάθε εικόνας έξοδο που αντιστοιχεί σε κάποιο από τα ενδιάμεσα κρυφά επίπεδα των προαναφερθέντων νευρωνικών δικτύων. (**20% του συνολικού βαθμού**)
4. Να παρουσιάσετε παραδείγματα της ορθής εκτέλεσης του κώδικάς σας. Χαρακτηρίστε κάποιες από τις εικόνες της βάσης ως εικόνες στόχο (**target images**) και παρουσιάστε μια λίστα με τις συναφέστερες εικόνες της βάσης. (**20% του συνολικού βαθμού**)
5. Προτείνετε μια συστηματική διαδικασία για την μέτρηση της ακρίβειας του συγκεκριμένου αλγορίθμου και παρουσιάσετε τα αποτελέσματά της. (**20% του συνολικού βαθμού**)

Μπορείτε να εργαστείτε σε ομάδες των 2 ή 3 φοιτητών.

Περιγραφή Paper

Σύμφωνα με την εκφώνηση της υπολογιστικής εργασίας ζητούμενο είναι η ανάπτυξη ενός γραφοθεωρητικού αλγορίθμου για την ανάκτηση εικόνων με βάση το περιεχόμενο τους, η διαδικασία αυτή βασίζεται στο paper **Multimedia Retrieval through Unsupervised Hypergraph-based Manifold Ranking**. Ο αλγόριθμος που αναλύεται στο paper αυτό ονομάζεται "**Log-Based Hypergraph of Ranking Reference**" ή αλλιώς εν συντομία **LHRR** και αποτελείται από πέντε βήματα τα οποία είναι και ζητούμενα για την υλοποίηση της εργασίας.

Για την εκτέλεση του αλγορίθμου απαιτείται πρώτα να εξαχθεί ένα σύνολο αντικειμενικών χαρακτηριστικών για την περιγραφή της διανυσματικής αναπαράστασης όλων των πολυμεσικών αντικειμένων (multimedia objects), όπως είναι τα βίντεο και οι εικόνες. Στο σημείο αυτό λοιπόν η πρόταση του paper είναι η χρήση μιας μεθόδου εξαγωγής χαρακτηριστικών (feature extracting function), η οποία συμβολίζεται με **ε**. Η συγκεκριμένη διαδικασία στο δικό μας αρχείο κώδικα γίνεται μέσω ενός προεκπαιδευμένου μοντέλου (pretrained model) της βιβλιοθήκης torchvision, στο οποίο έχουμε αφαιρέσει στην ουσία το output layer -που για το μοντέλο resnet50 ονομάζεται fully connected (fc) layer- με αποτέλεσμα να μπορούμε να εξάγουμε τα χαρακτηριστικά (features). Στο σημείο αυτό, απαιτείται η χρήση ενός κατάλληλου μέτρου ομοιότητας - θέλουμε δηλαδή μια μετρική στο χώρο αναπαράστασης των εικόνων - η οποία θα βασίζεται στην ομοιότητα **ρ** μεταξύ δύο αντικειμένων, η οποία ομοιότητα αυτή βασίζεται σε μία μέθοδο υπολογισμού απόστασης και συμβολίζεται με **δ**. Ο συνηθέστερος τρόπος υπολογισμού του **δ**, είναι η Ευκλείδεια απόσταση όπου και είναι ως μέτρο αντιστρόφως ανάλογη της ομοιότητας των δύο αντικειμένων.

Το αρχικό σύνολο (dataset) των εικόνων συμβολίζεται ως $\mathbf{C = \{o_1, o_2, \ldots, o_n\}}$ και με $\mathbf{n = |C|}$ συμβολίζεται το πλήθος των objects (ο) που βρίσκονται στο dataset αυτό. Επίσης, θέτουμε ως $\mathbf{o_q}$ την εικόνα στόχος (query image) και στόχος του αλγορίθμου είναι να βρεθεί το πλήθος **k** πλησιέστερων objects. Το παραπάνω για να επιτευχθεί, γίνεται η χρήση μιας λίστας $\mathbf{Tq = \{\tau_q(1), \tau_q(2), \ldots, \tau_q(n)\}}$, η οποία αποτελεί μια ταξινομημένη λίστα (Ranked List) όπου περιλαμβάνει μόνο τα πρώτα **L** ομοίοτερα objects της εικόνας στόχος $\mathbf{o_q}$. Η ταξινόμηση της λίστας χρησιμοποιεί το similarity function **ρ**. Στην ταξινομημένη αυτή λίστα ισχύει ότι αριστερότερα βρίσκονται οι περισσότερο όμοιες εικόνες και όσο προχωράμε προς τα δεξιά βρίσκονται οι λιγότερο όμοιες. Επομένως σε μία λίστα **Tq**, όπου το $\mathbf{o_i}$ προηγείται του $\mathbf{o_j}$ αυτό που ισχύει είναι ότι: $\mathbf{Tq(i) < Tq(j)}$ και επομένως $\mathbf{\rho(O_q, O_i) \geq \rho(O_q, O_j)}$. Στην υλοποίηση μας, για τον υπολογισμό της ομοιότητας έγινε η χρήση του αντίστροφου της Ευκλείδειας απόστασης, σύμφωνα με την υπόδειξη του καθηγητή στη σχετική διάλεξη του μαθήματος. Τέλος, ορίζονται τα γειτονικά σύνολα **N(q, k)**. Ο λόγος όπου το paper προτείνει να διατηρηθούν μόνο τα πρώτα **L** ομοιότερα, είναι πως η εξαντλητική αξιολόγηση της ομοιότητας είναι υπολογιστικά επαχθής, έτσι προκύπτει το υποσύνολο $\mathbf{CL \subset C}$.

Στο σημείο αυτό θα αναλύσουμε τα βήματα του αλγορίθμου:

1. **Κανονικοποίηση σειράς κατάταξης (Rank Normalization):**
   Όταν θέτουμε μια εικόνα στόχο $\mathbf{o_i}$, πραγματοποιείται μια διαδικασία κανονικοποίησης στην αντίστοιχη λίστα $\mathbf{T_i}$ υπολογίζοντας το νέο μετρικό ομοιότητας $\mathbf{\rho_n(i, j) = 2L - (T_i(j)}$

+ $T_j(i)$) με σκοπό την βελτίωση της συμμετρίας των γειτονικών συνόλων **N(q, k)**. Τέλος, κάθε λίστα ταξινομείται.

2. **Κατασκευή υπεργραφήματος (Hypergraph Construction):**
Ένας υπεργράφος είναι μια γενίκευση ενός παραδοσιακού γράφου αποτελούμενο από ακμές ως μη κενά υποσύνολα του συνόλου **V** των κορυφών. Σε ένα γράφημα της μορφής **G = (V, E, w)**, το **V** είναι ένα σύνολο κορυφών, το **E** το σύνολο των υπερακμών $U_{e\in E}$ **= V** και το **W**, για κάθε υπερακμή e_i που ανήκει στο **V,** είναι μια θετική μετρική βάρους $W(e_i) \geq 0$ σε ένα σύνολο κορυφών $e_i = \{u_1, u_2, \ldots, u_n\}$ και ορίζεται για κάθε εικόνα **o** βάση το γειτονικό της σύνολο **k**. Στον αλγόριθμο μας, κάθε υπερακμή αντιστοιχίζεται σε μια εικόνα/κόμβο, δηλαδή ο πρώτος κόμβος στο σύνολο κορυφών της e_i είναι το v_i. Η υπερακμή e_i εφάπτεται με μια κορυφή V_j στο **V** όταν το V_j ανήκει στο e_j, η σχέση αυτή κωδικοποιείται με τη δυαδική μήτρα **|E| x |V|**, όπου **1** ανήκει και όπου **0** δεν ανήκει. Σε κάποιες συγκεκριμένες περιπτώσεις η παραπάνω υλοποίηση δημιουργεί έναν βαθμό ασάφειας έτσι μπορεί να γίνει η χρήση μιας πιθανοτικής αναπαράστασης - ομοίως υλοποιήθηκε και στο δικό μας αλγόριθμο. Έστω $o_x \in N(i,k)$ ένας γείτονας του o_i και $o_j \in N(x,k)$ ένας γείτονας του o_x .Το μέτρο συμμετοχής $r(e_i,u_j)$ της κορυφής u_j στην υπερακμή e_i υπολογίζεται ως εξής: $r(e_i,u_j) = \Sigma_{ox\in N(i,k)\ \wedge\ oj\in N(x,k)}\ w_p(i,x) \times w_p(x,j)$ όπου $w_p(i,x)$ είναι μια συνάρτηση που αναθέτει ένα βάρος σχετικότητας στο o_x με βάση την θέση του στην ταξινομημένη λίστα T_i. Εικόνες οι οποίες είναι κοντά στο χώρο των αντικειμενικών χαρακτηριστικών προβλέπεται πως θα ανήκουν στην ίδια κλάση. Το βάρος που ανατίθεται στην εικόνα o_x με βάση τη θέση της στην λίστα T_i υπολογίζεται ως εξής: $w_p(i,x) = 1 - \log_k \tau_i(x)$. Η συνάρτηση αυτή θέτει ένα μέγιστο βάρος ίσο με 1 στην πρώτη θέση της λίστας (που είναι η ίδια η εικόνα στόχος) και παρουσιάζει μια ταχεία μείωση για τις πρώτες ταξινομημένες θέσεις. Σκοπός είναι να ανατεθούν υψηλές τιμές βαρών στις πρώτες θέσεις όπου η αποτελεσματικότητα των ταξινομημένων λιστών είναι ανώτερη. Με βάση τις τιμές $r(e_i,u_j)$ ορίζουμε το incidence matrix που αποτελεί απεικόνιση του υπεργραφήματος ως εξής: $H(e_i,v_j) = r(e_i,v_j)$, αν $v_j \in e_i$ ή 0 σε διαφορετική περίπτωση. Ακόμη πρέπει να οριστεί ένα βάρος $W(e_i)$ για κάθε υπερακμή e_i το οποίο κωδικοποιεί την εμπιστοσύνη των σχέσεων που έχουν δημιουργηθεί ανάμεσα στις κορυφές των υπερακμών. Μια πολύ αποτελεσματική υπερακμή αναμένει να περιέχει πολλές κορυφές, κάτι το οποίο έρχεται σε άμεση συσχέτιση με τις υψηλές τιμές του $h(e_i,\cdot)$. Με αυτό το τρόπο το $W(e_i)$ μπορεί να οριστεί ως εξής: $w(e_i) = \Sigma_{j\in Nh(i,k)}\ H(i,j)$.

3. **Υπολογισμός ομοιότητας υπερακμών (Hyperedge Similarities):**
Στο βήμα αυτό αρχικά πρέπει να υπολογιστεί ο πίνακας ομοιότητας ανά ζεύγη (pairwise similarity matrix) **S**. Ο υπολογισμός της βασίζεται σε δύο διαφορετικές υποθέσεις. Η <u>πρώτη υπόθεση</u> είναι πως παρόμοιες εικόνες παρουσιάζουν παρόμοιες ταξινομημένες λίστες και επομένως και παρόμοιες υπερακμές. Εφόσον όλη η πληροφορία για την σχετικότητα των υπερακμών έχει κωδικοποιηθεί στο incidence matrix, ένα μέτρο ομοιότητας για δύο υπερακμές e_i, e_j μπορεί να οριστεί ως εξής: $S_h = HH^T$. Η <u>δεύτερη υπόθεση</u>, είναι πως παρόμοιες εικόνες αναμένεται να αναφέρονται από τις ίδιες υπερακμές και έτσι προκύπτει ο πίνακας: $S_u = H^TH$. Έτσι, αφού και οι δύο μήτρες ομοιότητας περιέχουν την σχετική πληροφορία μπορεί να υπολογιστεί ο

πίνακας ομοιότητας ανά ζεύγη **S** μέσω του Hadamard Product (Element-wise multiplication): $\mathbf{S = S_h \bullet S_u}$.

4. **Υπολογισμός καρτεσιανού γινομένου μεταξύ των υπερακμών (Cartesian Product of Hyperedge Elements):**
   Για να εξαχθούν οι σχέσεις ζευγαριών απευθείας από ένα σύνολο των στοιχείων που έχουν ως όρισμα τις υπερακμές, πρέπει να υπολογιστεί το καρτεσιανό τους γινόμενο. Για να επιτευχθεί αυτό κάθε υπερακμή συνδέει ένα σύνολο κορυφών, με αποτέλεσμα να μεγιστοποιήσουν την πληροφορία της σχετικότητας τους. Το καρτεσιανό γινόμενο δηλαδή ορίζεται ως εξής: $\mathbf{e_q \ x \ e_i = \{(u_x,u_y):u_x \in e_q \text{^} u_y \in e_i\}}$. Ας υποθέσουμε ότι το $\mathbf{e_q^2}$ δηλώνει το καρτεσιανό γινόμενο μεταξύ στοιχείων από την ίδια υπερακμή, $\mathbf{e_q}$ που προκύπτει ως: $\mathbf{e_q \ x \ e_q = e_q^2}$. Έτσι, για κάθε ζεύγος κορυφών που ανήκουν στο καρτεσιανό γινόμενο των υπερακμών ορίζεται μια σχέση ομοιότητας ανά ζεύγη. Στη συνέχεια ξανά υπολογίζεται η συνάρτηση ομοιότητας **ρ** με βάση το $\mathbf{w(e_q)}$ η οποία κωδικοποιεί την εμπιστοσύνη της υπερακμής $\mathbf{e_q}$ που ορίζει την συσχέτιση. Ο βαθμός συμμετοχής των $\mathbf{u_i}$ και $\mathbf{u_j}$ είναι: $\mathbf{\rho(e_q,u_i,u_j) = w(e_q) \ x \ H(e_q,u_i) \ x \ H(e_q,u_j)}$. Το μέτρο ομοιότητας στο οποίο βασίζεται το καρτεσιανό γινόμενο υπολογίζεται μέσω μιας μήτρας **C**, η οποία λαμβάνει υπόψη τις σχέσεις που υπάρχουν σε όλες τις υπερακμές. Ο λόγος αυτών των υπολογισμών βασίζεται στην πιθανότητα συνύπαρξης των κορυφών $\mathbf{u_i}$ και $\mathbf{u_j}$ σε διαφορετικές υπερακμές. Κάθε θέση στη μήτρα **C** υπολογίζεται ως εξής: $\mathbf{C(i,j) = \Sigma_{eq \in E \text{^} (ui,uj) \in eq^2} \rho(u_i,u_j)}$.

5. **Υπολογισμός ομοιότητας βάσει του κατασκευασμένου υπεργράφου (Hypergraph - based Similarity):**
   Το ζεύγος ομοιότητας που ορίστηκε βασισμένο στις υπερακμές και το καρτεσιανό γινόμενο παρέχει διακριτή και συμπληρωματική πληροφορία για την πολλαπλότητα των δεδομένων. Αυτή, εξάγεται από τον υπολογισμό της μήτρας συγγένειας **W** που συνδυάζει τις μήτρες **C** και **S**: $\mathbf{W = C \bullet S}$. Σύμφωνα με τον βαθμό συγγένειας που υπολογίστηκε στο **W**, μια διαδικασία κατηγοριοποίησης (ranking), μπορεί να εφαρμόζεται με σκοπό να δημιουργηθεί μια νέα ταξινομημένη λίστα **T**. Η διαδικασία είναι η ακόλουθη: Θεωρώντας την τωρινή επανάληψη ως $\mathbf{T^{(0)}}$, την αρχική λίστα που υπολογίστηκε βάση των αρχικών αντικειμενικών χαρακτηριστικών. Θα μπορούσαμε να πούμε, πως η $\mathbf{T^{(t+1)}}$ μπορεί να υπολογιστεί βάσει του $\mathbf{W^{(T)}}$. Μετά από ένα προκαθορισμένο αριθμό επαναλήψεων της λίστας **T**, υπολογίζεται το τελικό ταξινομημένο σύνολο των λιστών $\mathbf{T_r}$.

Ανάλυση Κώδικα

Programming Language: Python
Libraries: pytorch,torchvision,numpy,math,matplotlib,math,os
Dataset: ImageNetDogs
Features Extraction: resnet50

Αρχικά επιλέγεται η συσκευή στην οποία θα εκτελεστεί ο αλγόριθμος, καθώς στις συγκεκριμένες διεργασίες μια gpu αποδίδει καλύτερα και εφόσον ένα σύστημα διαθέτει είναι καλό να χρησιμοποιηθεί.

```
# Set the processing device
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Επειδή το συγκεκριμένο dataset δεν επιτρέπει την λήψη του μέσω διαδικτύου, έχουμε διατηρήσει ένα δείγμα περίπου 5000 εικόνων. Την φορτώνουμε με την μέθοδο **load_dataset**. Από αυτές, σε κάθε εκτέλεση του προγράμματος, επιλέγουμε ένα τυχαίο δείγμα εικόνων πλήθους **limit**. (στη φωτογραφία το limit είναι 400)

```
1 usage   thkox +1
def load_dataset(transform, limit=400, shuffle=True):
    """
    Load Stanford Dogs dataset and preprocess the images with a specified limit
    http://vision.stanford.edu/aditya86/ImageNetDogs/
    """
    dataset = ImageFolder( root: 'images_dataset', transform=transform)
    if shuffle:
        sample_list_index = random.sample(range(len(dataset)), limit)
        dataset = torch.utils.data.Subset(dataset, sample_list_index)
    else:
        dataset = torch.utils.data.Subset(dataset, range(limit))

    return dataset
```

Στη συνέχεια φορτώνουμε ένα προεκπαιδευμένο μοντέλο της επιλογής μας (**resnet-50**) με τη μέθοδο **load_pre_trained_model**. Ορίζουμε το μοντέλο ότι είναι για λόγους testing και απενεργοποιούμε το training mode.

```python
def load_pre_trained_model(select_device):
    """
    Load the pre-trained model
    """
    model_set = models.resnet50(weights='IMAGENET1K_V1')

    # Set the model to evaluation mode to avoid updating the running statistics of batch normalization layers
    model_set.eval()

    # Remove the classification (fully connected) layer
    model_set = torch.nn.Sequential(*(list(model_set.children())[:-1]))

    # set processing device
    model_set.to(select_device)

    return model_set
```

Εκτυπώνουμε στην οθόνη τη πρώτη εικόνα του dataset, η οποία θα είναι και το query object του προγράμματος, χρησιμοποιώντας την μέθοδο **show_image**.

```python
2 usages  thkox
def show_image(image_index, save_image=False, image_name=""):
    image, _ = data_loader[image_index]
    image = image.squeeze().permute(1, 2, 0).numpy()
    plt.imshow(image)
    plt.axis('off')
    if not os.path.exists("retrieved_images"):
        os.makedirs("retrieved_images")
    if save_image and image_name != "":
        plt.savefig("retrieved_images/" + image_name + ".png")
    elif save_image:
        plt.savefig("retrieved_images/" + str(image_index) + ".png")
    plt.show()
```

Έχοντας το φορτώσει το μοντέλο και το δείγμα εικόνων στη βάση, το επόμενο βήμα είναι ο υπολογισμός των χαρακτηριστικών των εικόνων και την αποθήκευση των χαρακτηριστικών αυτών σε μία λίστα **features_list** με την μέθοδο **calculate_features**.

```python
def calculate_features(dataset_images):
    """
    Calculate the features of the images using the pre-trained model
    """
    features_list = []
    for ds_image, _ in dataset_images:
        # Add batch dimension to image tensor
        ds_image = ds_image.unsqueeze(0)

        # Move the image tensor to the same device as the pre-trained model
        ds_image = ds_image.to(device)

        # Pass the image through the ResNet18 model
        with torch.no_grad():
            feature = pre_trained_model(ds_image).squeeze().cpu().numpy()  # Convert feature to numpy array

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

    # Make them 1D
    for i in range(len(features_list)):
        features_list[i] = features_list[i].reshape(features_list[i].size)

    return features_list
```

Log-based Hypergraph of Ranking References (LHRR)

Initial Similarity

Η συνάρτηση **calculate_similarity,** υπολογίζει τα σκορ ομοιότητας μεταξύ εικόνων βασιζόμενη στην αντίστροφη Ευκλείδεια απόσταση των χαρακτηριστικών τους. Η λίστα των χαρακτηριστικών μετατρέπεται σε NumPy array και δημιουργείται μια κενή λίστα **similarity_scores_list,** στην οποία αποθηκεύονται τα σκορ ομοιότητας. Τα αποτελέσματα αποθηκεύονται στην λίστα, ως λίστες, όπου κάθε είσοδος περιλαμβάνει ένα tuple με τον δείκτης της εικόνας και το αντίστοιχο σκορ ομοιότητας. Τέλος, επιστρέφεται η λίστα των σκορ ομοιότητας για όλες τις εικόνες.

```python
def calculate_similarity(features_of_all_images):
    """
    Calculate the similarity scores between images based on the inverse Euclidean distance of their features.
    Similarity score = 1 / (Euclidean distance)
    """
    features_of_all_images = np.array(features_of_all_images)
    similarity_scores_list = []

    for i in range(len(features_of_all_images)):
        distances = np.linalg.norm(features_of_all_images - features_of_all_images[i], axis=1)
        scores = 1 / np.where(distances == 0, 0.00001, distances)  # Avoid division by zero
        similarity_scores_list.append(list(enumerate(scores)))

    return similarity_scores_list
```

Rank Normalization

Η συνάρτηση **rank_normalization,** πραγματοποιεί την κανονικοποίηση βαθμολογίας σε λίστες ομοιότητας μεταξύ εικόνων. Η συνάρτηση, χρησιμοποιεί την λίστα **similarity_lists** όπου περιέχει σε λίστες τα σκορ ομοιότητας. Στην συνέχεια, υπολογίζονται οι βαθμοί (**ranks**).

```
def rank_normalization(similarity_lists):
    """
    ---Rank Normalization---

    Use similarity scores to sort
    """
    normalized_similarity_scores = []
    L = len(similarity_lists[0])  # Length of each similarity list

    for i in range(len(similarity_lists)):
        ranks = []
        for j in range(len(similarity_lists[i])):
            rank = 2 * L - (similarity_lists[i][j][1] + similarity_lists[j][i][1])
            ranks.append((j, rank))
        # sort the ranks based on the rank (the second value of the tuple) and append to the
        # normalized_similarity_scores list
        normalized_similarity_scores.append(sorted(ranks, key=lambda x: x[1]))

    return normalized_similarity_scores
```

Hypergraph Construction

Όταν ολοκληρωθεί η διαδικασία εξομάλυνσης των χαρακτηριστικών, επόμενο στάδιο είναι η δημιουργία ενός υπεργραφήματος. Το υπεργράφημα δημιουργείται μέσω της μεθόδου **get_hypergraph_construction**.

```python
def get_hypergraph_construction(similarity_scores, k=5):
    """
    ---Hypergraph Construction---

    Create hyperedges as lists
    """
    hyperedges = []
    for i in range(len(similarity_scores)):
        hyperedge = []
        for j in range(k):
            hyperedge.append(similarity_scores[i][j][0])
        # Add the hyperedge to the hyperedges
        hyperedges.append(hyperedge)
    return hyperedges
```

Στη συνέχεια υπολογίζουμε τις σχέσεις μεταξύ των ακμών, μέσω της μεθόδου **create_edge_associations** και τις αποθηκεύουμε στο υπεργράφημα που έχουμε δημιουργήσει.

```python
def create_edge_associations(hyperedges, k=5):
    """
    ---Create Edge Associations---

    Create edge associations based on the hyperedges
    """
    associations = np.zeros((len(hyperedges), len(hyperedges)))
    for i, e in enumerate(hyperedges):
        for j in range(len(hyperedges)):
            if j in e:
                position = e.index(j) + 1  # Get the position of the node in the hyperedge
                associations[i][j] = 1 - math.log(position, k + 1)  # Calculate the weight
            else:
                associations[i][j] = 0
    return associations
```

Έχοντας το σχεδιασμένο υπεργράφημα, μπορούμε εύκολα να υπολογίσουμε τα βάρη μεταξύ των ακμών, δηλαδή των φωτογραφιών.

```python
def create_edge_weights(hyperedges, edge_associations):
    """
    ---Hyperedge Weight---

    Calculate the weights of the hyperedges
    """
    weights = []
    for i, e in enumerate(hyperedges):
        sum = 0
        for h in e:
            sum += edge_associations[i][h]
        weights.append(sum)
    return weights
```

Hyperedge Similarities

Η συνάρτηση, **get_hyperedges_similarities** υπολογίζει τις ομοιότητες μεταξύ υπερακμών (hyperedges) μέσω Hadamard product σε πίνακες.

```python
def get_hyperedges_similarities(incidence_matrix):
    """
    ---Hyperedges Similarities---

    Compute the Hadamard product of the pairwise similary matrix
    """
    Similarity_matrix_h = incidence_matrix @ incidence_matrix.T  # Matrix multiplication
    Similarity_matrix_u = incidence_matrix.T @ incidence_matrix  # Matrix multiplication
    Similarity_matrix = np.multiply(Similarity_matrix_h, Similarity_matrix_u)  # Hadamard product
    return Similarity_matrix
```

Cartesian Product of Hyperedge Elements

Η συνάρτηση, **get_cartesian_product_of_hyperedge_elements**, εκτελεί τρις βασικές λειτουργίες: Υπολογισμός καρτεσιανού γινομένου, Οι Βαθμοί σχέσεων μεταξύ ζευγών κορυφών, Βαθμοί ομοιότητας βασισμένοι στο καρτεσιανό γινόμενο.

```python
def get_cartesian_product_of_hyperedge_elements(edge_weights, edge_associations, hyperedges):
    """
    ---Cartesian Product of Hyperedge Elements---

    Calculate cartesian product of the hyperedge elements.
    Then calculate the membership degrees of the hyperedges based on weights.
    Finally, calculate the matrix C.
    """
    membership_degrees = [{} for _ in range(len(hyperedges))]
    matrix_c = np.zeros((len(hyperedges), len(hyperedges)))
    for i, e in enumerate(hyperedges):
        eq_ei = np.transpose(np.meshgrid( *xi: e, e)).reshape(-1, 2)
        for (vertices1, vertices2) in eq_ei:
            membership_degrees[i][(vertices1, vertices2)] = edge_weights[i] * edge_associations[i][vertices1] * \
                                                            edge_associations[i][vertices2]
            matrix_c[vertices1][vertices2] += membership_degrees[i][(vertices1, vertices2)]
    return matrix_c
```

Hypergraph-Based Similarity

Η συνάρτηση **get_hypergraph_based_similarity**, υπολογίζει το τελικό **affinity matrix**, βασισμένο σε υπεργράφους.

```python
def get_hypergrapgh_based_simalarity(matrix_c, hyperedges_similarities):
    """
    ---Hypergraph-based similarity---

    Computes final affinity matrix.
    """
    affinity_matrix = np.multiply(matrix_c, hyperedges_similarities)
    return affinity_matrix
```

Εδώ, πραγματοποιείται ανάκτηση εικόνων με βάση τα **similarity_scores**, για μια συγκεκριμένη εικόνα (query object/image).

```python
# Retrieve the images
query_image_index = 0
retrieved_images = []
for (i, score) in similarity_scores[query_image_index]:
    if score != 0:
        retrieved_images.append((i, score))
retrieved_images = sorted(retrieved_images, key=lambda x: x[1], reverse=True)

print("Retrieved images: ", len(retrieved_images))
```

Επιστρέφουμε όλες τις φωτογραφίες που εμφανίζουν ομοιότητες με το query object/image.

```python
def calculate_accuracy(retrieved_images, query_image_label):
    """
    Calculate the accuracy of the retrieved images
    """
    count = 0
    for i in range(len(retrieved_images)):
        image_index, score = retrieved_images[i]
        image, label = data_loader[image_index]
        print("Image index: {:<5} | Image label: {:<5} | Score: {:.6f}".format( *args: image_index, label, score))
        if label == query_image_label:
            count += 1

    return (count / len(retrieved_images)) * 100
```

Υπολογίζουμε την ακρίβεια του αλγορίθμου με την μέθοδο **calculate_accuracy**. Χρησιμοποιούμε το label της πρώτης φωτογραφίας και το συγκρίνουμε με τα labels των πρώτων 5 όμοιων φωτογραφιών που επέστρεψε ο αλγόριθμος. Η ακρίβεια ισούται με το ποσοστό των όμοιων labels με το label του query image/object προς το άθροισμα όλων των φωτογραφιών που συγκρίναμε, δηλαδή 5.

Εκτέλεση Παραδειγμάτων Προγράμματος

Ακολουθούν εικόνες εκτέλεσης προγράμματος:

```
Image Analysis: Final Exam
Device:  cuda
Dataset size:  600
Extracted features:  600  features.
Similarity scores calculated.
Number of iterations:  9
-----------Iteration:  1 ------------
Rank normalization completed.
Hypergraph construction completed.
Edge associations created.
Edge weights created.
Hyperedges similarities calculated.
Cartesian product of hyperedge elements completed.
Affinity matrix calculated.
Similarity scores updated.
------Iteration:  1  completed.------
```

Επαναλαμβάνεται η βασική δομή του αλγορίθμου συνολικά 9 φορές.

Ο υπολογισμός της ακρίβειας του συγκεκριμένου αλγορίθμου μπορεί να διαφέρει από εκτέλεση σε εκτέλεση, καθώς κατά την εκτέλεση επιλέγεται τυχαία κάθε φορά ένα υποσύνολο του αρχικού dataset, όπως επίσης και ένα τυχαίο query object/image.

Πρώτη Περίπτωση: Η ακρίβεια είναι μικρότερη του 100%

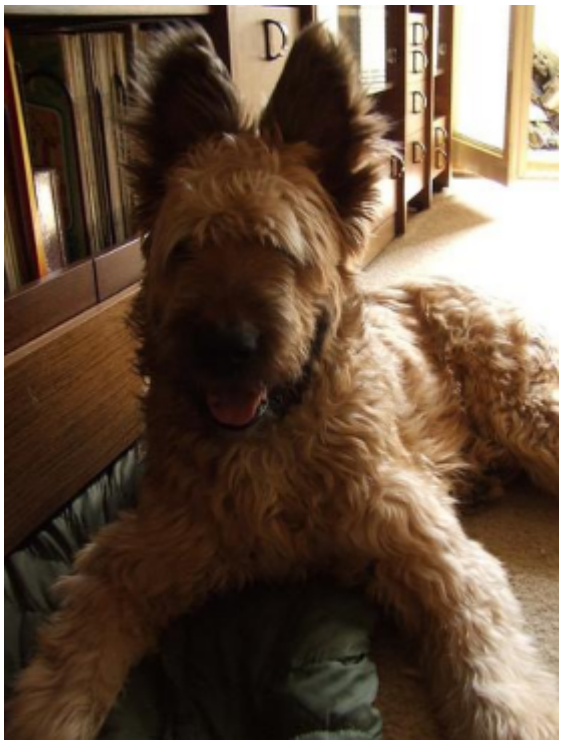
```
Retrieved images:  8
The top 5 retrieved images saved to the retrieved_images folder.
Image index: 0     | Image label: 0     | Score: 5.888818
Image index: 510   | Image label: 0     | Score: 4.510605
Image index: 501   | Image label: 28    | Score: 2.452110
Image index: 222   | Image label: 0     | Score: 0.850223
Image index: 89    | Image label: 20    | Score: 0.019623
Accuracy: 60.0%
```

Όταν τελειώσει η επανάληψη επιστρέφει τον αριθμό κατάταξης, την κατηγορία και το βαθμό ομοιότητας των 5 πρώτων φωτογραφιών με την φωτογραφία query object.

Εικόνες

Αρχική Εικόνα	
	**Index:** 0 **Label:** 23

Εικόνες που επιστρέφει ο αλγόριθμος	
	**Index:** 0 **Label:** 23 **Score:** 6.281548
	**Index:** 0 **Label:** 19 **Score:** 4.469404
	**Index:** 0 **Label:** 19 **Score:** 2.170430

	**Index:** 0 **Label:** 23 **Score:** 0.279432
	**Index:** 0 **Label:** 19 **Score:** 0.197154

Δεύτερη Περίπτωση: Η ακρίβεια είναι 100%

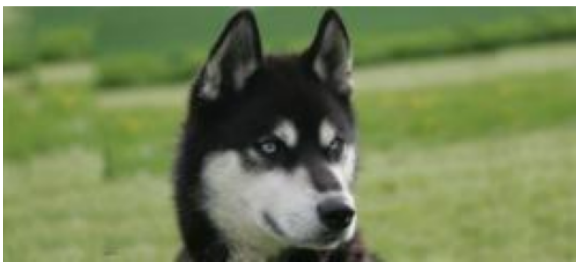
```
Retrieved images:  7
The top 5 retrieved images saved to the retrieved_images folder.
Image index: 0     | Image label: 30    | Score: 20.216243
Image index: 232   | Image label: 30    | Score: 12.432255
Image index: 491   | Image label: 30    | Score: 6.398597
Image index: 591   | Image label: 30    | Score: 2.754038
Image index: 436   | Image label: 30    | Score: 1.558267
Accuracy: 100.0%
```

Υπάρχουν και περιπτώσεις που ο αλγόριθμος έχει 100% ακρίβεια και όλες οι φωτογραφίες που επιστρέφει ανήκουν στην ίδια κατηγορία με την query image/object.

Εικόνες

Αρχική Εικόνα	
	**Index:** 0 **Label:** 30

Εικόνες που επιστρέφει ο αλγόριθμος	
	**Index:** 0 **Label:** 30 **Score:** 20.216243
	**Index:** 232 **Label:** 30 **Score:** 12.432255

	**Index:** 491 **Label:** 30 **Score:** 6.398597
	**Index:** 591 **Label:** 30 **Score:** 2.754038
	**Index:** 436 **Label:** 30 **Score:** 1.558267

Πηγές

Guimarães Pedronette, D. C., Valem, L. P., Almeida, J., & Torres, R. S. (2019). "Multimedia Retrieval through Unsupervised Hypergraph-based Manifold Ranking."

PyTorch Documentation. Retrieved January 2024 from https://pytorch.org/docs/stable/index.html

Torchvision Documentation. Retrieved January 2024 from https://pytorch.org/vision/stable/index.html

ImageNet Dataset. Retrieved January 2024 from http://vision.stanford.edu/aditya86/ImageNetDogs/menu_frame.html