Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Incorporating Visual Features for Image Retrieval via Scene Graphs

Diploma Thesis

by

Dimitrios Georgoulopoulos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Συστημάτων Τεχνητής Νοημοσύνης και Μάθησης

Incorporating Visual Features for Image Retrieval via Scene Graphs

Diploma Thesis

by

Dimitrios Georgoulopoulos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουλίου, 2025.

.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Μιχάλης Βαζιργιάννης
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ecole Polytechnique

Αθήνα, Ιούλιος 2025

.....

Δημητριος Γεωργουλοπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Αφιέρωση

Θα ήθελα να αφιερώσω την παρούσα διπλωματική στη μνήμη της μητέρας μου, που με γέμιζε θάρρος και δύναμη. Ιδιαίτερη ευγνωμοσύνη οφείλω στον πατέρα μου, που μου μετέδωσε το πάθος για γνώση, καθώς και στις αδερφές μου, Καίτη και Ιωάννα, στα ξαδέλφια μου και στους φίλους μου, για την αμέριστη ψυχολογική στήριξή τους.

Περίληψη

Η ανάκτηση εικόνων με βάση εικόνα είναι η διαδικασία εύρεσης των πιο όμοιων εικόνων σε μία εικόνα ερώτηση. Ένας πρακτικός τρόπος αναπαράστασης του υψηλού σημασιολογικού περιεχομένου των εικόνων είναι μέσω γράφων σκηνής, οι οποίοι περιέχουν τα αντικείμενα και τις σχέσεις τους για τις εικόνες. Με την τροποποίηση των γράφων σκηνής των εικόνων με οπτικές πληροφορίες και την επεξεργασία τους μέσω νευρωνικών δικτύων γράφων, μπορούμε να εξάγουμε ισχυρές αναπαραστάσεις για αυτούς που μπορούν να χρησιμοποιηθούν για το πρόβλημα της ανάκτησης εικόνων με βάση εικόνα.

Παρόλα αυτά, πολλά σύνολα δεδομένων με γράφους σκηνής κατασκευασμένους από ανθρώπους και τις εικόνες τους είναι συχνά μη συνεκτικά και γεμάτα με αντικείμενα που δεν έχουν σημασία για τη σημασιολογική περιγραφή της εικόνας. Επιπρόσθετα, νευρωνικά δίκτυα γράφων που χρησιμοποιούν μηχανισμούς προσοχής συχνά αγνοούν τις ακμές των γράφων — πληροφορίες για τις σχέσεις μεταξύ των αντικειμένων — οι οποίες μπορεί να είναι κρίσιμες για τον διαχωρισμό των εικόνων. Σε αυτή τη διατριβή, αντιμετωπίζουμε αυτά τα ζητήματα. Πρώτον, προτείνουμε ένα μοντέλο πρόβλεψης σημαντικότητας, που χρησιμοποιεί κωδικοποιητές Transformer και ένα multi-head attention στρώμα με εκπαιδευόμενες ερωτήσεις (queries), για τον προσδιορισμό μιας τιμής σημαντικότητας για τα αντικείμενα και τις σχέσεις τους, μέσω των ονομάτων και των οπτικών τους πληροφοριών. Έπειτα φιλτράρουμε τους γράφους σκηνής, κρατώντας μόνο τα πιο σημαντικά αντικείμενα και ακμές. Δεύτερον, προτείνουμε ένα στρώμα «Edge-Aware GATv2» που ενσωματώνει τις πληροφορίες στις ακμές των γράφων τόσο στον υπολογισμό των συντελεστών προσοχής όσο και στα μηνύματα που ανταλλάσσονται μεταξύ των κόμβων. Εκπαιδεύουμε ύστερα αυτό το μοντέλο ώστε ή σύγκριση της ομοιότητας δύο εικόνων μέσω των παραγόμενων ενσωματώσεων τους , να αντιστοιχεί με την πραγματική ομοιότητα των εικόνων, μετρούμενη από τη σημασιολογική ομοιότητα των προτάσεων που τις περιγράφουν.

Αξιολογούμε τη μέθοδό μας ποσοτικά — συγκρίνοντάς την με τυπικές αρχιτεκτονικές GNN, εκδοχές με αφαίρεση στοιχείων (ablated versions) και εναλλακτικές μεθόδους, χρησιμοποιώντας μετρικές ανάκτησης και κατάταξης — και ποιοτικά, παρουσιάζοντας ενδεικτικά αποτελέσματα του μοντέλου μας. Οι φιλτραρισμένοι, πολυτροπικοί γράφοι σκηνής, επεξεργασμένοι με το Edge-Aware GATv2, παράγουν ενσωματώσεις που περιέχουν πλούσιες πληροφορίες για τα αντικείμενα, την οπτική τους αναπαράσταση και τις σχέσεις τους. Σε όλες τις μετρικές, η μέθοδός μας υπερέχει άλλων GNN και τεχνικών ανάκτησης, δείχνοντας την αποτελεσματικότητά της για την ανάκτηση εικόνων με βάση εικόνα.

Λέξεις-κλειδιά — Ανάκτηση εικόνων βάσει εικόνας,Νευρωνικά δίκτυα γράφων,Γράφοι σκηνής,Ομοιότητα γράφων,Δίκτυα προσοχής σε γράφους,GATv2

Abstract

Image-to-image retrieval involves finding the most relevant images given a query image. A practical way to capture high-level semantic information in images is through scene graphs, which include objects and their relationships. Of course, raw visual features of the objects and the overall image are also critical for retrieval. By augmenting scene graphs with visual features and processing them with graph neural networks, we can derive powerful representations that effectively address the image-to-image retrieval problem.

However, existing datasets of human-annotated scene graphs and their corresponding images are often inconsistently labeled and cluttered with objects that are irrelevant to the image's main content. Moreover, attention-based graph neural networks typically overlook edge attributes—information about the relationships between objects—that can be crucial for distinguishing one image from another. In this thesis, we address these shortcomings in two ways. First, we introduce an Importance Prediction Module that leverages Transformer encoders and a multi-head attention mechanism with learned queries to find an importance score for each object and relation by combining semantic and visual cues. We then prune the scene graph, retaining only the most significant nodes and edges. Second, we propose an Edge-Aware GATv2 layer, which integrates edge features directly into both the attention computation and the messages exchanged between nodes. We train this model so that the resulting graph embeddings align with semantic similarity as measured by image captions.

We evaluate our approach and model quantitatively—comparing it against standard GNN architectures, ablated variants, and alternative methods using retrieval and ranking metrics—and qualitatively, by presenting illustrative examples. Our filtered, multimodal scene graphs processed with Edge-Aware GATv2 produce embeddings that capture rich information about objects, their visual appearance, and their relationships. Across all metrics, our method outperforms competing GNN variants and retrieval techniques, demonstrating its effectiveness for image-to-image retrieval.

Keywords — Image-to-image retrieval,Graph Neural Networks,Scene Graphs,Graph Similarity,Graph attention networks, GATv2,

Ευχαριστίες

Η υποστήριξη πολλών ανθρώπων ήταν ιδιαίτερα σημαντική για την περάτωση αυτής της διπλωματικής εργασίας. Ευχαριστώ πολύ τον επιβλέποντα μου , κ. Στάμου Γεώργιο , για την συστηματική επιστημονική καθοδηγηση του .Ειμαι ευγνώμον στον Νίκο Χάιδο και την Αγγελική Δημητρίου για την πολύτιμη βοήθεια που μου προσέφεραν σε κάθε στάδιο της εκπόνησης της διατριβής μου καθώς και τις ιδέες που μοιράστηκαν μαζί μου .Επιπλέον, ευχαριστώ όλους τους καθηγητές του Εθνικού Μετσόβιου Πολυτεχνείου που μου μετέδωσαν τις απαραίτητες γνώσεις για την επιτυχή διεξαγωγή της έρευνάς μου.

Γεωργουλόπουλος Δημήτριος, Ιούλιος 2025

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρετικό Υπόβαθρο

Πολλές τεχνολογίες που χρησιμοποιούν συστήματα τεχνητής νοημοσύνης έχουν φέρει τα τελευταία χρόνια επανάσταση σε πολλές βιομηχανίες και αποτελούν κρίσιμο κομμάτι πολλών εφαρμογών καθημερινής χρήσης. Κάποια παραδείγματα τεχνητής νοημοσύνης στην καθημερινή μας ζωή περιλαμβάνουν αυτοοδηγούμενα αυτοκίνητα, συστήματα κατανόησης και παραγωγής γλώσσας, ιατρικές εφαρμογές, προβλέψεις για τον καιρό, χρηματιστήριο κ.ά.

Ένα αναδυόμενο είδος νευρωνικών δικτύων που τα τελευταία χρόνια έχει κερδίσει έδαφος είναι τα νευρωνικά δίκτυα γράφων, τα οποία ξεπερνούν τις συνήθεις προκλήσεις που παρουσιάζουν οι δομές γράφων και βρίσκουν εφαρμογές σε πολλά προβλήματα όπου τα δεδομένα έχουν την μορφή γράφων, όπως στα συστήματα προτάσεων, τα κοινωνικά δίκτυα κ.ά.

Ο κύριος στόχος αυτής της διπλωματικής εργασίας είναι η ανάκτηση εικόνας με βάση άλλη εικόνα, δηλαδή η αναζήτηση της πιο όμοιας εικόνας βάσει τόσο του σημασιολογικού νοήματος όσο και του οπτικού περιεχομένου της, με βάση μια εικόνα-ερώτημα. Το πρόβλημα αυτό έχει πολλές εφαρμογές σε συστήματα προτάσεων και έχει μελετηθεί αρκετά από τη βιβλιογραφία. Στην εργασία θα χρησιμοποιήσουμε γράφους σκηνής, οι οποίοι παρέχουν μια σημασιολογική περιγραφή των αντικειμένων μιας εικόνας και των σχέσεων μεταξύ τους, σε συνδυασμό με οπτικές πληροφορίες, για μια πιο ακριβή επίλυση του προβλήματος.

Ειδικότερα, θα δοκιμάσουμε διαφορετικά είδη νευρωνικών δικτύων γράφων, τα οποία θα επεξεργάζονται τις πληροφορίες των γράφων σκηνής και τα οπτικά χαρακτηριστικά και θα παράγουν εκφραστικές αναπαραστάσεις των εικόνων. Επίσης, θα προτείνουμε ένα δίκτυο που θα εντοπίζει τα πιο σημαντικά αντικείμενα και τις σχέσεις μιας εικόνας, και θα προτείνουμε τροποποιήσεις στα νευρωνικά δίκτυα γράφων για καλύτερη αναπαράσταση και επεξεργασία των γράφων σκηνής.

1.2 Αλγόριθμοι Ομαδοποίησης

Οι αλγόριθμοι ομαδοποίησης είναι μια μη επιβλεπόμενη τεχνική μηχανικής μάθησης, στην οποία τα δεδομένα εισόδου — συχνά αναπαριστώμενα με τη μορφή ενσωματώσεων — ομαδοποιούνται σε ομάδες με βάση κάποιο μέτρο ομοιότητας (συνήθως τη γεωμετρική απόσταση στον χώρο των δεδομένων). Με αυτόν τον τρόπο μπορούμε να εντοπίσουμε κρυφές δομές στα δεδομένα και να αναγνωρίσουμε φυσικές ομαδοποιήσεις σε αυτά κ.ά. Στο σημείο αυτό θα περιγράψουμε έναν αλγόριθμο που θα χρησιμοποιήσουμε στην εργασία μας.

Ο αλγόριθμος Jenks natural breaks [37] είναι ένας αλγόριθμος ομαδοποίησης βασισμένος στην κατανομή, που στοχεύει στο να ελαχιστοποιήσει τη διασπορά μεταξύ των σημείων που ανήκουν στην ίδια κλάση και να μεγιστοποιήσει τη διασπορά μεταξύ διαφορετικών κλάσεων. Ο αλγόριθμος λαμβάνει ως είσοδο τον αριθμό των κλάσεων και, με χρήση δυναμικού προγραμματισμού, υπολογίζει αποδοτικά τις βέλτιστες θέσεις διαχωρισμού . Ως κριτήριο χρησιμοποιεί το άθροισμα των τετραγωνικών αποκλίσεων από τους μέσους όρους των κλάσεων και

επιλέγει τον διαχωρισμό με τη μικρότερη τιμή. Επειδή η πολυπλοκότητά του είναι περίπου $O(n^2 \cdot k)$, εφαρμόζεται κυρίως σε σχετικά μικρά σύνολα δεδομένων και σε περιορισμένο αριθμό κλάσεων εξόδου.

1.3 Μετρικές

Στο σημείο αυτό θα διατυπώσουμε τις μετρικές με τις οποίες αξιολογούνται πολλά μοντέλα μηχανικής μάθησης, καθώς και ειδικότερα μετρικές κατάταξής με τις οποίες αξιολογούνται συνήθως συστήματα ανάκτησης.

1.3.1 Μετρικές Ταξινόμησης

- **Ακρίβεια (Accuracy)** Ο λόγος των σωστών προβλέψεων προς τον συνολικό αριθμό των προβλέψεων:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

όπου TP = αληθώς θετικά, TN = αληθώς αρνητικά, FP = ψευδώς θετικά, FN = ψευδώς αρνητικά.

- **Ακρίβεια (Precision)** Ο λόγος των αληθώς θετικών προς το άθροισμα των αληθώς θετικών και ψευδώς θετικών:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

- **Ανάκληση (Recall)** Ο λόγος των αληθώς θετικών προς το άθροισμα των αληθώς θετικών και ψευδώς αρνητικών:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

- **Δείκτης F_1 ($\mathbf{F_1}$ Score)** Ο αρμονικός μέσος της ακρίβειας και της ανάκλησης:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

1.3.2 Μετρικές κατάταξης

1. **NDCG (Κανονικοποιημένο Εκπτωτικό Αθροιστικό Κέρδος)** Η μετρική μετρά την ποιότητα της κατάταξης συγκρίνοντας την κατάταξη που προβλέπει ένα μοντέλο με την ιδανική κατάταξη. Έστω $\text{rel}(i)$ η συνάφεια του αντικειμένου στη θέση i της προβλεπόμενης λίστας και $\text{rel}^*(i)$ η συνάφεια στην ιδανική κατάταξη. Τότε:

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{\text{rel}(i)}{\log_2(i+1)}, \quad \text{IDCG@}k = \sum_{i=1}^{k} \frac{\text{rel}^*(i)}{\log_2(i+1)}, \quad \text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}.$$

όπου DCG το εκπτωτικό αθροιστικό κέρδος της προβλέψιμης ταξινόμησης του μοντέλου μας , και IDCG το εκπτωτικό αθροιστικό κέρδος της ιδανικής κατάταξης με βάση την βασική αλήθεια [89] .

2. **Μέσος Αντίστροφος Βαθμός (MRR)** Χρησιμοποιείται συχνά σε συστήματα προτάσεων και ανάκτησης. Μετρά τη θέση του πρώτου σχετικού αντικειμένου στα αποτελέσματα:

$$\text{MRR} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{\text{rank}_n},$$

όπου N είναι ο αριθμός των ερωτημάτων και rank_n η θέση του πρώτου σχετικού αντικειμένου στην n-οστή λίστα.

3. **Μέσος Όρος Ακρίβειας (MAP)** Συνυπολογίζει το πλήθος των σχετικών αντικειμένων που ανακτήθηκαν και τη θέση τους στη λίστα προβλέψεων:

$$\text{MAP@}K = \frac{1}{U} \sum_{u=1}^{U} \text{AP@}K_u,$$

όπου

$$\text{AP@}K_u = \frac{1}{N_u}\sum_{k=1}^{K}\text{Precision}(k)\times\text{rel}(k),$$

και N_u ο αριθμός των σχετικών αντικειμένων για το u-οστό ερώτημα και rel(k) ,0 αν το αντικείμενο στην θέση k ειναι μη σχετικό και 1 αν είναι σχετικό .

1.3.3 Μετρικές παλινδρόμησης

1. **Μέσο Απόλυτο Σφάλμα (MAE)** Η μέση απόλυτη διαφορά μεταξύ των προβλεπόμενων τιμών $\hat{y}_i$ και των πραγματικών y_i:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|.$$

2. **Μέσο Τετραγωνικό Σφάλμα (MSE)** Η μέση τετραγωνική διαφορά μεταξύ των προβλεπόμενων τιμών $\hat{y}_i$ και των πραγματικών y_i:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

1.3.4 Μετρικές συσχέτισης

1. **Συντελεστής συσχέτισης Spearman** Χρησιμοποιείται για να συγκρίνει τις κατατάξεις προβλέψεων με τις πραγματικές κατατάξεις. Δείχνει κατά πόσο η σχέση μεταξύ δύο μεταβλητών μπορεί να περιγραφεί από μια μονότονη συνάρτηση. Έστω R_i η θέση του στοιχείου i στην πρώτη λίστα και S_i η θέση του στη δεύτερη. Ορίζουμε

$$d_i = R_i - S_i.$$

Ο συντελεστής συσχέτισης Spearman δίνεται από

$$\rho_s = 1 - \frac{6\sum_i d_i^2}{n\,(n^2-1)},$$

όπου n ο αριθμός των στοιχείων.

2. **Συσχέτιση απόστασης** Μετρά τόσο τις γραμμικές όσο και τις μη γραμμικές εξαρτήσεις μεταξύ δύο τυχαίων μεταβλητών (π.χ. διανυσμάτων). Χρησιμοποιείται όταν δύο μεταβλητές έχουν ισχυρή μη γραμμική σχέση, αλλά ο συντελεστής Pearson (r) είναι περίπου 0, επειδή μετρά μόνο γραμμικές εξαρτήσεις [81].

1.4 Γράφοι

Οι γράφοι είναι μια σημαντική δομή δεδομένων που περιγράφει αντικείμενα και σχέσεις, όπως οι χάρτες, τα συστήματα προτάσεων, τα κοινωνικά δίκτυα, τα δίκτυα παραπομπών, το Διαδίκτυο (π.χ. Wikipedia, όπου όροι και λέξεις συνδέονται με άλλους όρους ή URLs).

Ένας συνηθισμένος ορισμός ενός γράφου είναι το μη διατεταγμένο ζεύγος

$$G = (V, E),$$

όπου V το σύνολο των κορυφών (vertices ή nodes) και

$$E \subseteq \{\{x, y\} \mid x, y \in V,\, x \neq y\}$$

το σύνολο των ακμών (links μεταξύ κόμβων). Επειδή οι ακμές είναι μη διατεταγμένα ζεύγη, η σειρά των στοιχείων δεν έχει σημασία .Επίσης στους πιο απλούς ορισμούς , όπως αυτός , δεν επιτρέπονται πολλαπλές ακμές μεταξύ των ίδιων κόμβων.

Οι ακμές μπορούν επίσης να οριστούν ως διατεταγμένα ζεύγη (u, v) με $u, v \in V$. Σε αυτή την περίπτωση ο γράφος είναι *κατευθυνόμενος* (directed), καθώς $(u, v) \neq (v, u)$.Αντιθέτως αν η ακμή (u, v) ταυτίζεται με την (v, u), τότε ο γράφος είναι *μη κατευθυνόμενος* (undirected).

1.5 Transformers

Το 2017, ερευνητική ομάδα της Google (Vaswani et al. [83]) δημοσίευσε το άρθρο «Attention Is All You Need», στο οποίο εισήγαγαν την αρχιτεκτονική Transformer.

Το κύριο χαρακτηριστικό του Transformer είναι η χρήση του μηχανισμού προσοχής για τον εντοπισμό των πιο χρήσιμων και σημαντικών τμημάτων μιας ακολουθίας. Στο σημείο αυτό θα περιγράψουμε τα βασικά στοιχεία της μονάδας κωδικοποίησης (encoder) της αρχιτεκτονικής Transformer, δίνοντας τις κεντρικές ιδέες της, καθώς για την εργασία μας χρησιμοποιούμε κυρίως αυτή τη μονάδα.

Προετοιμασία δεδομένων

Διαχωρισμός σε tokens (tokenization): Αρχικά διαχωρίζουμε τις εισόδους του μοντέλου σε tokens. Δηλαδή, κάθε είσοδος μετατρέπεται σε αναπαράσταση ενσωμάτωσης (embedding).

Κωδικοποιήσεις θέσης (positional encodings): Επειδή οι transformer επεξεργάζονται όλα τα tokens ταυτόχρονα, οι ενσωματώσεις των tokens πρέπει να περιέχουν πληροφορίες για τη θέση τους στην ακολουθία. Για το σκοπό αυτό, προσθέτουμε σε κάθε token ένα διανύσμα θέσης—συνήθως παραγόμενο από ημιτονικές και συνημιτονικές συναρτήσεις—που υποδεικνύει στο μοντέλο τη θέση του κάθε token.

1.5.1 Κωδικοποιητής (Encoder)

Στον κωδικοποιητή εισάγεται ολόκληρη η ακολουθία που θέλουμε να επεξεργαστεί ο Transformer (token embeddings + positional encodings) παράλληλα. Κάθε επίπεδο κωδικοποιητή (layer)—το οποίο μπορεί να επαναλαμβάνεται πολλές φορές για μεγαλύτερη εκφραστικότητα—περιέχει:

1. **Multi-Head Self-Attention:** Σε αυτό το υπο-επίπεδο κάθε ενσωμάτωση token προβάλλεται σε τρία διανύσματα ερώτησης (query), κλειδιού (key) και τιμής (value). Έπειτα χωρίζουμε κάθε προβολή σε h κεφαλές, κάθε μία διαστάσεως d_k. Για κάθε κεφαλή υπολογίζουμε:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)\, V.$$

   Κάθε διάνυσμα ερώτησης συγκρίνεται με τα διανύσματα κλειδιού για να προκύψουν "βαθμολογικά σκόρ" (relevance scores) . Η διαίρεση με $\sqrt{d_k}$ κανονικοποιεί τα σκόρ αυτά και η softmax τα μετατρέπει σε συντελεστές προσοχής με άθροισμα 1. Σχηματίζουμε μετά ένα σταθμισμένο άθροισμα (weighted sum) των τιμών (values) και, αφού το επαναλάβουμε για όλες τις κεφαλές, η ενώσεις (concatenation) των εξόδων προβάλλονται στην αρχική διάσταση.

2. **Πρόσθεση και Κανονικοποίηση:** Προσθέτουμε residual συνδεση (δηλαδή προσθέτουμε στο αρχικό token την έξοδο του μοντέλου ώστε να μην αλλάξει ραγδαία η αναπαράστασή του χάνοντας πιθανόν πληροφορίες για αυτό) και εφαρμόζουμε ένα στρώμα κανονικοποίησης της εξόδου (normalization layer).

3. **Feed-Forward Δίκτυο:** Κάθε token εξόδου περνάει από ένα ανεξάρτητο MLP με δύο γραμμικά στρώματα και μία μη γραμμική συνάρτηση (π.χ. ReLU, GELU) , αφού έχει συλλέξει πληροφορίες (context) από τον μηχανισμό αυτο-προσοχής (self-attention).

4. **Επανάληψη Πρόσθεσης και Κανονικοποίησης:** Προσθέτουμε και πάλι residual συνδεση και εφαρμόζουμε ένα στρώμα κανονικοποίησης της εξόδου (normalization layer). Με αυτό ολοκληρώνεται ένα επίπεδο κωδικοποιητή, το οποίο μπορεί να επαναληφθεί πολλές φορές ώστε το δίκτυό μας να συλλέξει ακόμα πιο σύνθετες πληροφορίες (context) για κάθε token.

1.6 Νευρωνικά Δίκτυα Γράφων

Ένα μεγάλο πλήθος προβλημάτων και δεδομένων μπορεί να αναπαρασταθεί με γράφους. Συνηθισμένες αρχιτεκτονικές νευρωνικών δικτύων— όπως τα συνελικτικά (convolutional) δίκτυα—δεν μπορούν να εφαρμοστούν απευθείας σε γράφους, γιατί η δομή τους δεν είναι ούτε πλέγμα ούτε ακολουθία (όπως απαιτούν τα RNNs και οι Transformers).

Συγκεκριμένα, μια συνάρτηση που εφαρμόζεται σε γράφους πρέπει να δίνει τα ίδια αποτελέσματα ανεξαρτήτως της σειράς με την οποία εμφανίζονται οι κόμβοι στα σύνολα (V, E). Με άλλα λόγια, οι διεργασίες στους γράφους πρέπει να είναι *permutation invariant* (πχ για την αναπαράσταση σε επίπεδο γράφου) και *permutation equivariant*. Έστω $V = \{u_1, u_2, \dots\}$ και P ένας πίνακας αναδιάταξης (permutation matrix) που αλλάζει απλώς τη σειρά των κόμβων. Τότε η συνάρτηση f για επεξεργασία γράφου πρέπει να ικανοποιεί:

- **Permutation invariance:**
$$f(P\,K\,P^\top) = f(K).$$
- **Permutation equivariance:**
$$f(P\,K\,P^\top) = P\,f(K).$$

1.6.1 Κατηγορίες Νευρωνικών Δικτύων Γράφων

Στη συνέχεια θα αναλύσουμε τη λειτουργία ορισμένων βασικών νευρωνικών δικτύων γράφων. Γενικά, τα νευρωνικά δίκτυα γράφων μπορούν να χωριστούν σε δύο κατηγορίες:

1. **Βασισμένα στο Φάσμα (spectral-based):** Τα φίλτρα που «αναμειγνύουν» πληροφορίες μεταξύ κόμβων ορίζονται με βάση τα ιδιοδιανύσματα του λαπλασιανού πίνακα του γράφου.
2. **Βασισμένα στο Χώρο (spatial-based):** Κάθε κόμβος λαμβάνει πληροφορίες απευθείας από τους γειτονικούς του. Είναι τα δίκτυα που χρησιμοποιούνται ευρύτερα στην πράξη, καθώς είναι πιο αποδοτικά σε υπολογιστικό κόστος.

1.6.2 Νευρωνικά Δίκτυα Μετάβασης Μηνυμάτων

Τα Νευρωνικά Δίκτυα Μετάβασης Μηνυμάτων (Message Passing Neural Networks – MPNNs) [27] παρέχουν ένα γενικό πλαίσιο για μάθηση σε δομές γράφων, με την επαναληπτική ενημέρωση των αναπαραστάσεων των κόμβων μέσω «μετάδοσης μηνυμάτων» από τους γειτονικούς κόμβους. Αυτά τα δίκτυα είναι σχεδιασμένα ώστε να είναι permutation equivariant (στην ενημέρωση κόμβων) και, όταν χρησιμοποιούν για έξοδο λειτουργίες (π.χ. sum ή mean pooling), permutation invariant (σε επίπεδο ολόκληρου του γράφου).

Σε κάθε στρώμα k, κάθε κόμβος u έχει τρέχουσα κρυφή αναπαράσταση $h_u^{(k)}$. Η ενημέρωση από το στρώμα k στο στρώμα $k+1$ γίνεται ως εξής:

$$m_{N(u)}^{(k)} = \mathrm{AGGREGATE}^{(k)}\big(\{h_v^{(k)} \mid v \in N(u)\}\big),$$
$$h_u^{(k+1)} = \mathrm{UPDATE}^{(k)}\big(h_u^{(k)},\, m_{N(u)}^{(k)}\big),$$

όπου

- $N(u)$ είναι το σύνολο των γειτόνων του κόμβου u,
- $\mathrm{AGGREGATE}^{(k)}$ είναι μια συνάρτηση που σέβεται την αναδιάταξη (π.χ. άθροισμα ή μέσος όρος, ακολουθούμενα από νευρωνικό δίκτυο),
- $\mathrm{UPDATE}^{(k)}$ είναι ένα διαφορίσιμο νευρωνικό δίκτυο που συνδυάζει τη "τωρινή" αναπαράσταση $h_u^{(k)}$ με το μήνυμα $m_{N(u)}^{(k)}$ για να παράγει τη νέα $h_u^{(k+1)}$.

Στοίβα πολλών τέτοιων στρωμάτων επιτρέπει σε κάθε κόμβο να λαμβάνει σταδιακά πληροφορίες από απομακρυσμένους κόμβους (μέσω k hops για k στρώματα). Οι τελικές αναπαραστάσεις των κόμβων μπορούν να χρησιμοποιηθούν για προβλήματα επιπέδου κόμβου, ενώ με κατάλληλη έξοδο, μπορούν να επιλύσουν και προβλήματα επιπέδου γράφου ή ακμής.

1.6.3 Συνελικτικά Δίκτυα Γράφων

Το αρχικό συνελικτικό δίκτυο γράφων που πρότεινε ο Kipf & Welling [42] συνδέει τα δίκτυα γράφων που βασίζονται στο φάσμα και στο χώρο. Δείχνει ότι ένα ΣΔΓ (συνελικτικό δίκτυο γράφων), που είναι μια προσέγγιση πρώτης τάξης των τοπικών φασματικών φίλτρων, είναι ισοδύναμο με μια συνάρτηση μετάβασης μηνυμάτων (MPNNs). Συγκεκριμένα, η διαδικασία είναι η εξής:

Ανάλυση Fourier του γράφου: Τα ιδιοδιανύσματα U της κανονικοποιημένης λαπλασιανής $L = I_N - D^{-1/2} A\, D^{-1/2}$ παίζουν τον ρόλο των ημιτόνων και συνημιτόνων σε ένα γράφο. Μετατρέποντας το σήμα x στην ιδιοβάση $U^\top x$ (που ορίζει την μετατροπή Fourier σε γράφους), βλέπουμε πόσο κάθε "συχνότητα" του γράφου—που ορίζεται από τις ιδιοτιμές λ_i του L—συμμετέχει στο σήμα.

Φασματικά φίλτρα: Ένα φίλτρο στην παραπάνω βάση είναι η συνάρτηση $g_\theta(\Lambda) = \mathrm{diag}(g_\theta(\lambda_1), \dots, g_\theta(\lambda_N))$, και εφαρμόζοντας ένα τέτοιο φίλτρο σε ένα σήμα, ενισχύει ή καταστέλει κάποια ιδιοσυχνότητα, δίνοντας:

$$g_\theta * x = U\, g_\theta(\Lambda)\, U^\top x.$$

Διαφορετικές επιλογές για το $g_\theta(\Lambda)$ ελέγχουν πόσο ισχυρά θα αναμειχθούν πληροφορίες από τους γειτονικούς κόμβους (Τα φίλτρα χαμηλής διέλευσης εκτελούν εξομάλυνση γειτνίασης, τα φίλτρα υψηλής διέλευσης τονίζουν τις έντονες διαφορές και ούτω καθεξής).

Κόλπο Chebyshev (Chebyshev trick): Ο υπολογισμός του U είναι υπολογιστικά ακριβός, γι' αυτό ο Hammond et al. (2011) προσεγγίζει το $g_\theta(\Lambda)$ με ενα ανάπτυγμα σε Chebyshev πολυώνυμα μήκους K: $g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta'_k\, T_k(\tilde{\Lambda})$, όπου $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$. Έτσι προκύπτει $g_\theta * x \approx \sum_{k=0}^{K} \theta'_k\, T_k(\tilde{L})\, x$, το οποίο είναι K-localized και υπολογίζεται σε $O(|E|)$ χρόνο.

Προσέγγιση πρώτης τάξης και renormalization: Θέτοντας $K = 1$, $\lambda_{\max} = 2$ και $\theta'_0 = \theta'_1 = 1$ για μείωση των παραμέτρων, παίρνουμε για το φίλτρο $(I_N + D^{-1/2} A\, D^{-1/2})\, x$. Ο Kipf & Welling επανακανονικοποιούν (renormilize) την σχέση σε $\tilde{D}^{-1/2}\, \tilde{A}\, \tilde{D}^{-1/2}$x με $\tilde{A} = A + I_N$ για να αποφύγουν αριθμητικές αστάθειες, ετσι παίρνουν τον τελικό κανόνα για την ανανέωση των αναπαραστάσεων:

$$H^{(\ell+1)} = \sigma\big(\tilde{D}^{-1/2}\, \tilde{A}\, \tilde{D}^{-1/2}\, H^{(\ell)}\, W^{(\ell)}\big).$$

Γενικά, ένα συνελικτικό δίκτυο γράφων μαθαίνει πόσο και ποιές απο τις φασματικές συνιστώσες του γράφου , να διατηρήσει για κάθε κόμβο . Οι τελικές αναπαραστάσεις των κόμβων περιέχουν πληροφορίες τόσο για τις αναπαραστάσεις των γειτονικών κόμβων όσο και για τη δομή και τη συνδεσιμότητα του γράφου.

1.6.4 Graph Isomorphism Network (GIN)

Ο σκοπός του GIN [95] είναι να φτιάξει ένα νευρωνικό δίκτυο γράφων με διακριτική ικανότητα αντίστοιχη του τεστ Weisfeiler–Lehman. Το GIN ανανεώνει την αναπαράσταση κάθε κόμβου παίρνοντας την ενσωμάτωση της προηγούμενης αναπαράστασής του και το άθροισμα των αναπαραστάσεων των γειτονικών κόμβων, και μετά περνάει αυτό το αποτέλεσμα από ένα *multi layer perceptron* (MLP). Συγκεκριμένα, στο στάδιο k η ενημέρωση γίνεται ως εξής:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\big((1 + \varepsilon^{(k)})\, h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}\big),$$

όπου $\varepsilon^{(k)}$ είναι είτε μια σταθερά, είτε παράμετρος που μαθαίνει το μοντέλο.

Παρόλα αυτά, σε πολλές πρακτικές εφαρμογές— ιδιαίτερα όταν οι κόμβοι έχουν πλούσιες αναπαραστάσεις αρχικά— το γεγονός ότι το δίκτυό μας είναι αποδεδειγμένα όσο δυνατό όσο το τεστ 1-WL δεν μεταφράζεται απαραίτητα σε καλύτερη απόδοση του μοντέλου. Μοντέλα όπως το GAT και το GCN μπορεί να γενικεύουν καλύτερα σε δεδομένα πολλών εφαρμογών.

1.6.5 Graph Attention Network (GAT)

Εμπνευσμένα από τον μηχανισμό προσοχής (self-attention) των Transformers, τα GAT [84] τροποποιούν τη μετάδοση μηνυμάτων στα νευρωνικά δίκτυα γράφων προσθέτοντας έναν μηχανισμό προσοχής που επιτρέπει την απόδοση διαφορετικού βάρους σε κάθε γειτονικό κόμβο. Προηγούμενα δίκτυα συγκέντρωναν (aggregate) ομοιόμορφα μηνύματα, ενώ τα GAT μαθαίνουν μεταβλητούς συντελεστές προσοχής α_{ij} που δίνουν έμφαση σε ορισμένους κόμβους και αγνοούν άλλους.

Συγκεκριμένα, για κόμβο i και κάθε γείτονά του $j \in N(i)$, υπολογίζονται πρώτα οι συντελεστές

$$e_{ij} = \text{LeakyReLU}\big(a^\top [\, Wh_i \,\|\, Wh_j \,]\big), \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i)} \exp(e_{ik})},$$

όπου το e_{ij} είναι ο μη κανονικοποιημένος συντελεστής προσοχής μεταξύ των κόμβων i και j, με χρήση ενός εκπαιδευόμενου διανύσματος a και ενός πίνακα βαρών W, και η α_{ij} η κανονικοποιημένη του μορφή.

Η ενημέρωση της αναπαράστασης του κόμβου i γίνεται με:

$$h'_i = \sigma\Big(\sum_{j \in N(i)} \alpha_{ij} \, Wh_j \Big),$$

όπου σ είναι μια μη γραμμική ενεργοποίηση (π.χ. ELU ή ReLU).

Τέλος, μπορούμε να εφαρμόσουμε multi-head attention με K κεφαλές για περισσότερη εκφραστικότητα του μοντέλου μας:

$$h'_i = \sigma\Big(\tfrac{1}{K} \sum_{k=1}^{K} \sum_{j \in N(i)} \alpha_{ij}^{(k)} \, W^{(k)} h_j \Big).$$

1.6.6 GATv2

Σε μια δημοσίευση που ακολούθησε το αρχικό GAT, οι συγγραφείς έδειξαν ότι τα GAT στρώματα έχουν περιορισμούς στον τρόπο που υπολογίζουν σε ποιους κόμβους πρέπει να δοθεί προσοχή. Συγκεκριμένα, το κλασικό GAT δεν επιτρέπει σε κάθε κόμβο να δώσει μοναδική προσοχή σε διαφορετικούς γείτονες: όλοι οι κόμβοι δίνουν προτεραιότητα στον ίδιο γείτονα. Αυτό συμβαίνει διότι ο συντελεστής προσοχής μπορεί να γραφτεί ως

$$e(h_i, h_j) = \text{LeakyReLU}\big(a_1^\top W h_i + a_2^\top W h_j\big),$$

όπου $a = [a_1 \,\|\, a_2]$. Επειδή ο όρος $a_2^\top W h_j$ εξαρτάται μόνο από τον j, υπάρχει πάντα ένα $j_{\max}$ με μέγιστη τιμή $a_2^\top W h_{j_{\max}}$. Λόγω της μονότονης LeakyReLU και της softmax, ο κόμβος $j_{\max}$ λαμβάνει πάντα το μεγαλύτερο βάρος προσοχής για κάθε i που συνδέεται μαζί του.

Για να ξεπεράσουν αυτό το πρόβλημα, οι συγγραφείς πρότειναν το GATv2 [10], που απλώς αναδιατάσσει τις πράξεις στον υπολογισμό του συντελεστή προσοχής. Συγκεκριμένα, εφαρμόζεται πρώτα η LeakyReLU και μετά ο πολλαπλασιασμός με το εκπαιδευόμενο διάνυσμα a:

$$e(h_i, h_j) = a^\top \text{LeakyReLU}\big(W[\, h_i \,\|\, h_j \,]\big).$$

Έτσι, ο παράγοντας $\text{LeakyReLU}(W[\, h_i \,\|\, h_j \,])$ εξαρτάται ταυτόχρονα από h_i και h_j, και ο συντελεστής προσοχής υπολογίζεται δυναμικά για κάθε ζεύγος κόμβων.

1.7 Προτεινόμενο Μοντέλο

1.7.1 Συνεισφορά

Οι συνεισφορές αυτής της διπλωματικής εργασίας μπορούν να συνοψιστούν ως εξής:

- Χρησιμοποιούμε γράφους σκηνής — από το σύνολο δεδομένων Panoptic Scene Graph Generation — και οπτικές πληροφορίες για τα αντικείμενα και τη συνολική εικόνα, ώστε να πραγματοποιήσουμε ανάκτηση της πιο όμοιας εικόνας βάσει εικόνας-ερωτήματος. Οι γράφοι σκηνής παρέχουν σημασιολογικές πληροφορίες για τα αντικείμενα και τις σχέσεις τους επιτρέποντας μας σε συνδιασμό με οπτικές πληροφορίες να σχηματίσουμε πλούσιες αναπαραστάσεις για τις εικόνες μέσω γράφων.
- Χρησιμοποιούμε έναν κωδικοποιητή transformer σε συνδυασμό με ένα επίπεδο multi-head attention με εκπαιδευόμενα ερωτήματα, για την κατασκευή ενός μοντέλου που, λαμβάνοντας τα ονόματα των αντικειμένων και των σχέσεών τους, καθώς και τις οπτικές πληροφορίες και μια συνολική αναπαράσταση της εικόνας, προβλέπει τα πιο σημαντικά αντικείμενα και σχέσεις στον γράφο της εικόνας. Μετά, εφαρμόζουμε έναν προτεινόμενο αλγόριθμο φιλτραρίσματος που αφαιρεί τους μη σημαντικούς κόμβους των γράφων.

- Δοκιμάζουμε διάφορα νευρωνικά δίκτυα γράφων και προτείνουμε μια τροποποιημένη έκδοση του στρώματος GATv2, η οποία εισάγει πληροφορίες για τις σχέσεις των αντικειμένων στον υπολογισμό του συντελεστή προσοχής και στην παραγωγή μηνυμάτων μεταξύ των κόμβων.
- Προτείνουμε βελτιστοποιήσεις στη ροή επεξεργασίας (pipeline) και στην αρχιτεκτονική μας, και εξηγούμε ποσοτικά και ποιοτικά τη διαισθητική (intuition) και το κίνητρό μας πίσω από κάθε απόφαση, καθώς και την επίδρασή τους στο μοντέλο μας.

1.7.2 Σύνολα δεδομένων

Χρησιμοποιούμε τις εικόνες που βρίσκονται τόσο στο PSG σύνολο δεδομένων (Panoptic Scene Graph Generation dataset) όσο και στο MS-COCO σύνολο δεδομένων. Το πρώτο σύνολο δεδομένων μας παρέχει γράφους σκηνής για κάθε εικόνα, καθώς και τις συντεταγμένες των αντικειμένων στην εικόνα (bounding boxes), ενώ το δεύτερο σύνολο δεδομένων μας παρέχει πέντε προτάσεις για κάθε εικόνα που την περιγράφουν σημασιολογικά.

1.7.3 Σύνολο Γράφων Δεδομένων (Graph Dataset)

Για την κατασκευή των γράφων που θα χρησιμοποιήσουμε παρακάτω, αρχικά εφαρμόζουμε τον προτασιακό (sentence) και τον οπτικό (vision) Transformer του μοντέλου OpenCLIP_ViT_H_14_laion2b_s32b_b79k σε κάθε αντικείμενο της εικόνας. Συγκεκριμένα, για κάθε αντικείμενο εξάγουμε μια προτασιακή ενσωμάτωση διαστάσεως 1024 για το όνομά του και μια οπτική ενσωμάτωση διαστάσεως 1024 για το τμήμα της εικόνας που αντιστοιχεί σε αυτό. Στη συνέχεια, επεκτείνουμε την οπτική ενσωμάτωση προσθέτοντας πέντε κανονικοποιημένους αριθμούς που αντιστοιχούν στις συντεταγμένες και το εμβαδόν του πλαισίου περιορισμού (bounding box), σχηματίζοντας έτσι οπτική ενσωμάτωση διαστάσεως 1029. Η τελική αναπαράσταση κάθε αντικειμένου προκύπτει από τη συνένωση της προτασιακής και της οπτικής ενσωμάτωσης, με διαστατικότητα 2053. Για τις σχέσεις μεταξύ αντικειμένων χρησιμοποιούμε μόνο προτασιακή ενσωμάτωση, καθώς δεν υπάρχουν για αυτές πλαίσια περιορισμού. Επιπλέον, σε κάθε γράφο προσθέτουμε μια γενική οπτική ενσωμάτωση ολόκληρης της εικόνας, η οποία συνδέεται με όλους τους άλλους κόμβους μέσω ακμών χωρίς πληροφορία (zero embedding). Τέλος, διαχωρίζουμε το σύνολο δεδομένων σε σετ εκπαίδευσης (train), επικύρωσης (validation) και δοκιμής (test), τα οποία παραμένουν τα ίδια για όλα τα μοντέλα που παρουσιάζονται στη συνέχεια.

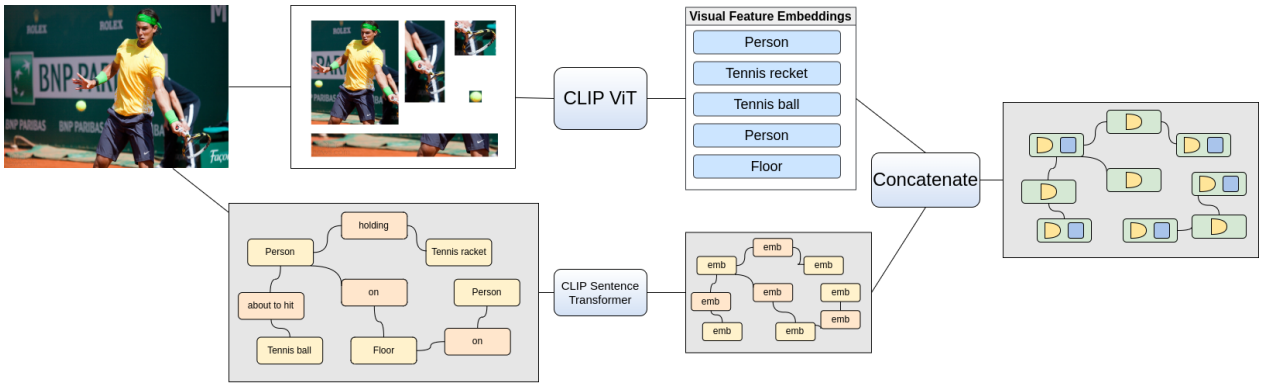


Figure 1.7.1: Για τον σχηματισμό των γράφων χρησιμοποιούμε τα ονόματα των αντικειμένων και των σχέσεων μεταξύ τους από το PSG σύνολο δεδομένων, καθώς και τη δομή του γράφου. Για τις οπτικές πληροφορίες των αντικειμένων και της εικόνας χρησιμοποιούμε τον οπτικό Transformer του μοντέλου CLIP. Έπειτα, για τα αντικείμενα του γράφου ενώνονται οι προτασιακές και οι οπτικές ενσωματώσεις τους.

1.7.4 Μοντέλο πρόβλεψης σημαντικότητας

Βασική αλήθεια

Ως «βασική αλήθεια» για τις τιμές σημαντικότητας των αντικειμένων του γράφου χρησιμοποιούμε τη μέση τιμή του εσωτερικού γινομένου των προτασιακών ενσωματώσεων των ονομάτων των αντικειμένων με αυτές των πέντε προτάσεων που περιγράφουν την εικόνα. Αντίστοιχα, για τριπλέτες (αντικείμενο_1, σχέση, αντικείμενο_2) σχηματίζουμε τη φράση «αντικείμενο_1 σχέση αντικείμενο_2» από τα ονόματα του PSG συνόλου δεδομένων,

υπολογίζουμε την προτασιακή ενσωμάτωση της φράσης αυτής και ξανά παίρνουμε τη μέση τιμή του εσωτερικού γινομένου με τις ενσωματώσεις των προτάσεων.

Για παράδειγμα, αν ένας γράφος σκηνής περιέχει τα αντικείμενα «άνθρωπος» και «γάτα» με τη σχέση «κοιτάζει» και μία από τις προτάσεις που περιγράφουν την εικόνα είναι «Μια γυναίκα κοιτάζει δύο γάτες», τότε η βαθμολογία σημαντικότητας για τον «άνθρωπο» είναι το εσωτερικό γινόμενο της ενσωμάτωσης «άνθρωπος» με τις προτάσεις (μέσος όρος εσωτερικών γινομένων σε όλες τις προτάσεις), αντίστοιχα για τη «γάτα», και για την τριπλέτα «άνθρωπος κοιτάζει γάτα» υπολογίζουμε το εσωτερικό γινόμενο της ενσωμάτωσης της φράσης «άνθρωπος κοιτάζει γάτα» με τις ενσωματώσεις των προτάσεων που περιγράφουν την εικόνα.

Στόχος του μοντέλου

Εκπαιδεύουμε ένα μοντέλο που για κάθε γράφο προβλέπει ποια αντικείμενα και ποιες τριπλέτες θεωρούνται σημαντικές. Το μοντέλο λαμβάνει ως είσοδο την ενσωμάτωση του αντικειμένου 1 (προτασιακή || οπτική), την προτασιακή ενσωμάτωση της σχέσης μεταξύ των αντικειμένων, την ενσωμάτωση του αντικειμένου 2 (προτασιακή || οπτική), μια συνολική προτασιακή ενσωμάτωση που περιγράφει ολόκληρο τον γράφο και μια συνολική οπτική ενσωμάτωση της εικόνας. Όταν θέλουμε να προβλέψουμε τη σημαντικότητα μόνο ενός αντικειμένου αντί τριπλέτας, αντικαθιστούμε την ενσωμάτωση της σχέσης και του αντικειμένου 2 με διανύσματα μηδενικών τιμών.

Συνολική προτασιακή ενσωμάτωση για τον γράφο

Για να παράγουμε τη συνολική προτασιακή ενσωμάτωση του γράφου κάθε εικόνας, ξεκινάμε σχηματίζοντας έναν γράφο που περιλαμβάνει μόνο τα ονόματα των αντικειμένων και των σχέσεων (χωρίς οπτικές πληροφορίες). Κάθε κόμβος στον γράφο φέρει την προτασιακή ενσωμάτωση του αντίστοιχου αντικειμένου ή σχέσης. Έπειτα, μετατρέπουμε κάθε σχέση σε ξεχωριστό κόμβο, δημιουργώντας ένα διμερές γράφο (bipartite graph) μεταξύ κόμβων-αντικειμένων και κόμβων-σχέσεων. Εκπαιδεύουμε έναν μη επιβλεπόμενο αλγόριθμο InfoGraph στο σετ εκπαίδευσης, και συλλέγουμε τις ενσωματώσεις που παράγει για κάθε γράφο κατά τη διάρκεια της εκπαίδευσης. Για τα σετ επικύρωσης και δοκιμής, χρησιμοποιούμε τις προβλεφθείσες ενσωματώσεις του ίδιου αλγορίθμου. Η τελική αυτή ενσωμάτωση κωδικοποιεί πληροφορίες για τη συνολική δομή του γράφου, καθώς και για όλα τα αντικείμενα και τις σχέσεις που περιέχει.

Αρχιτεκτονική

Πριν τις εισάγουμε στο μοντέλο πρόβλεψης σημαντικότητας , παιρνάμε καθέ μία απο τις 5 ενσωματώσεις που περιγράψαμε παραπάνω απο ένα γραμμικό στρώμα προβολής για να τα προβάλουμε στην ίδια διάσταση . Μετα παιρνάμε αυτά τα tokens απο ένα σύστημα κωδικοποιητή transformer , ο οποίος διαμέσου του μηχανισμού προσοχής που διαθέτει , ανανεώνει κάθε ενσωμάτωση δεδομένων με βάση τις άλλες, δίνοντας έμφαση στα πιο σημαντικά σημεία καθε μίας . Ετσι ο κωδικαποιητής tranformer παράγει 5 ανανεωμένα token ένα για κάθε είσοδο . Μετα παιρνάμε τα ανανεωμένα token απο ένα multi head attention στρωμα με εκπαιδευόμενες ερωτήσεις (queries) , για την παραγωγή ακόμα πιο εκφραστικών ενσωματώσεων . Τελικά κάνουμε mean pooling (μέση τιμή) τις ενσωματώσεις αυτές και παιρνάμε την συνολική ενσωμάτωση απο ένα μικρο MLP για να πάρουμε τελικά μια αριθμιτική τιμή σημαντικότητας για το αντικείμενο ή την τριπλέτα .Η συνολική αρχιτεκτονική φαίνετε στο σχημα 1.7.2

Εκπαίδευση

Για την εκπαίδευση του μοντέλου πρόβλεψης σημαντικότητας κατασκευάζουμε ένα σύνολο δεδομένων από το σετ εκπαίδευσης. Για κάθε αντικείμενο σε κάθε εικόνα περιλαμβάνουμε ένα δείγμα της μορφής (ενσωμάτωση αντικειμένου 1 (προτασιακή || οπτική), μηδενικό διάνυσμα, μηδενικό διάνυσμα, γενική οπτική ενσωμάτωση, γενική προτασιακή ενσωμάτωση γράφου) και την «βασική αλήθεια» της σημαντικότητας του αντικειμένου. Παρομοίως, για κάθε τριπλέτα σχηματίζουμε δείγματα της μορφής (ενσωμάτωση αντικειμένου 1, ενσωμάτωση σχέσης, ενσωμάτωση αντικειμένου 2, γενική οπτική ενσωμάτωση, γενική προτασιακή ενσωμάτωση γράφου) με την αντίστοιχη «βασική αλήθεια». Το μοντέλο εκπαιδεύεται ώστε να ελαχιστοποιεί το μέσο τετραγωνικό σφάλμα (mean squared error) μεταξύ των προβλεπόμενων τιμών σημαντικότητας και της «βασικής αλήθειας» για κάθε αντικείμενο ή τριπλέτα.

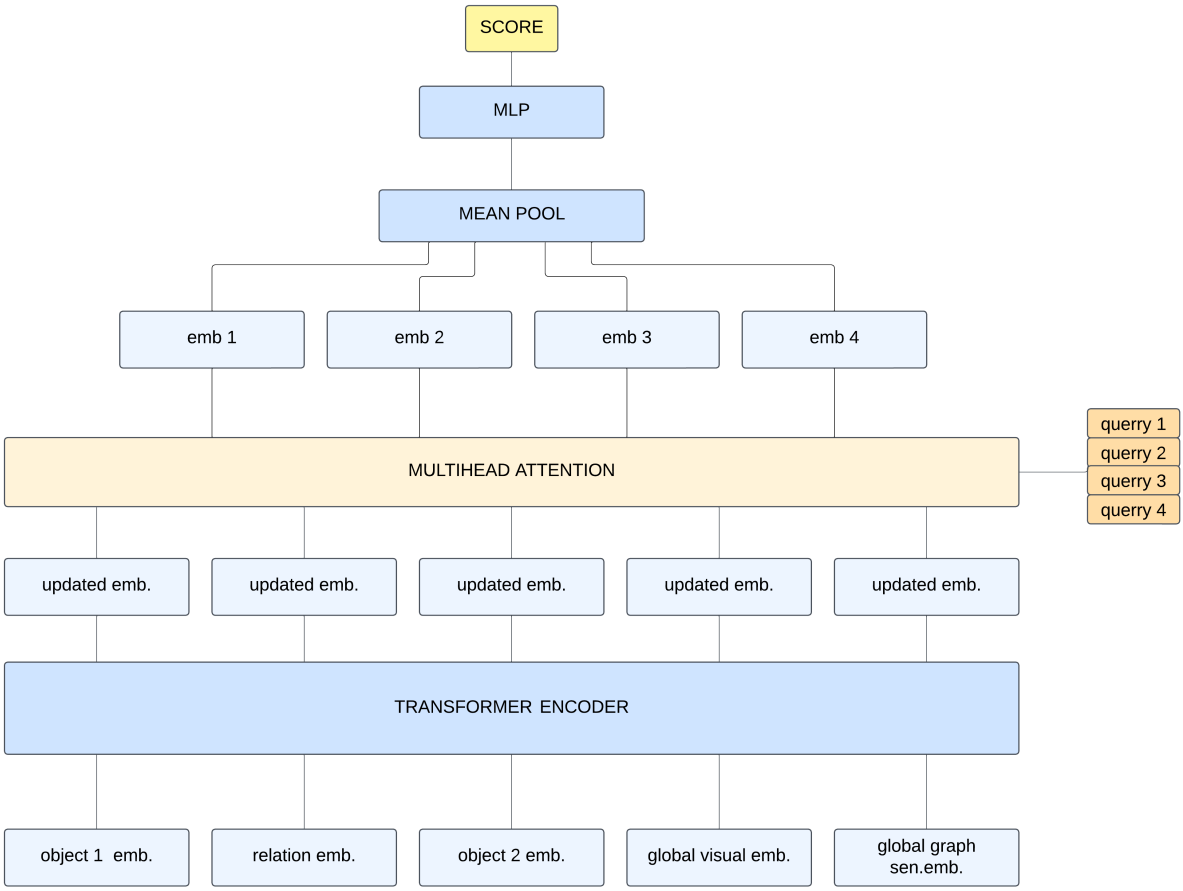


Figure 1.7.2: Η αρχιτεκτονική του μοντέλου πρόβλεψης σημαντικότητας: ως είσοδο λαμβάνει πέντε ενσωματώσεις (αντικείμενο 1, σχέση, αντικείμενο 2, γενική οπτική ενσωμάτωση εικόνας, γενική προτασιακή ενσωμάτωση γράφου). Αρχικά περνά αυτές μέσω ενός κωδικοποιητή Transformer, ο οποίος ανανεώνει τις ενσωματώσεις. Στη συνέχεια, οι ανανεωμένες ενσωματώσεις χρησιμοποιούνται ως κλειδιά (keys) και τιμές (values) σε ένα στρώμα multi-head attention με εκπαιδευόμενα ερωτήματα (queries). Τέλος, υπολογίζεται ο μέσος όρος των παραγόμενων ενσωματώσεων και αυτός τροφοδοτείται σε ένα μικρό MLP που δίνει την τελική τιμή σημαντικότητας.

1.8 Εκπαίδευση Νευρωνικού Δικτύου Γράφου

Βασική αλήθεια

Για την εκπαίδευση του νευρωνικού δικτύου γράφου, ως «βασική αλήθεια» της ομοιότητας μεταξύ δύο εικόνων παίρνουμε τη μέση τιμή της ομοιότητας μεταξύ κάθε δυάδας από τις πέντε προτάσεις που περιγράφουν την κάθε εικόνα (άρα 25 ζεύγη για κάθε ζεύγος εικόνων). Η «βασική αλήθεια» για τη σημαντικότητα των αντικειμένων και των τριπλετών σε μια εικόνα ορίζεται όπως περιγράφηκε στην προηγούμενη ενότητα.

Κλάδεμα γράφων

Πριν περάσουμε τους γράφους από το νευρωνικό δίκτυο γράφων για να παράξουμε μια γενική ενσωμάτωση, κλαδεύουμε κάποιους κόμβους και ακμές. Κατά τη διάρκεια της εκπαίδευσης, για κάθε εικόνα στο σετ εκπαίδευσης χρησιμοποιούμε τη βασική αλήθεια της σημαντικότητας των αντικειμένων για να σχηματίσουμε μια λίστα με αυτά. Αρχικά εφαρμόζουμε ένα όριο τιμής (threshold): όποιο αντικείμενο έχει τιμή σημαντικότητας πάνω από αυτό, χαρακτηρίζεται σημαντικό (label 1). Στη συνέχεια τρέχουμε τον αλγόριθμο Jenks natural breaks με δύο κλάσεις στις τιμές σημαντικότητας και θεωρούμε σημαντικά (label 1) τα αντικείμενα που ομαδοποιούνται στην κλάση με τις υψηλότερες τιμές. Τέλος, το σύνολο των σημαντικών αντικειμένων ορίζεται ως η ένωση των δύο συνόλων. Αντίστοιχα, εφαρμόζουμε την ίδια διαδικασία για να βρούμε τις σημαντικές τριπλέτες (αντικείμενο 1, σχέση, αντικείμενο 2) της εικόνας.

Στη συνέχεια, φιλτράρουμε κάθε γράφο στο σετ εκπαίδευσης ως εξής: κρατάμε τους κόμβους–αντικείμενα που έχουν ήδη χαρακτηριστεί σημαντικά και όλους τους κόμβους που συμμετέχουν σε κάποια σημαντική τριπλέτα. Επιπλέον, ο κόμβος με τη γενική οπτική ενσωμάτωση του γράφου θεωρείται πάντα σημαντικός και παραμένει μετά το φιλτράρισμα, μαζί με όλες τις ακμές του προς τους επιλεγμένους κόμβους. Ένα παράδειγμα της διαδικασίας φαίνεται στην Εικόνα 1.8.1 . Δοκιμάσαμε επίσης μια έκδοση του αλγορίθμου κλαδέματος που διατηρεί μόνο τα πλέον σημαντικά αντικείμενα, αλλά η απόδοση του μοντέλου ήταν ελαφρώς χαμηλότερη.

Η χρήση του αλγορίθμου Jenks προέκυψε από την παρατήρηση ότι οι τιμές σημαντικότητας των αντικειμένων σε διαφορετικές εικόνες παρουσίαζαν μεγάλη διασπορά. Έτσι ένα απλό όριο (ως κριτήριο της σημαντικότητας) δεν θα ήταν αρκετά συνεπές για όλες τις εικόνες. Επιπλέον, επειδή ο Jenks είναι αλγόριθμος συσταδοποίησης βασισμένος στην κατανομή των τιμών, μπορεί να βρει «φυσικά» σύνορα ανάμεσα σε δύο κλάσεις (υψηλές και χαμηλές τιμές), προσαρμόζοντάς τα σε κάθε εικόνα. Παρόμοια, και αλγόριθμοι όπως ο k-means πιθανόν να λειτουργούν ικανοποιητικά.

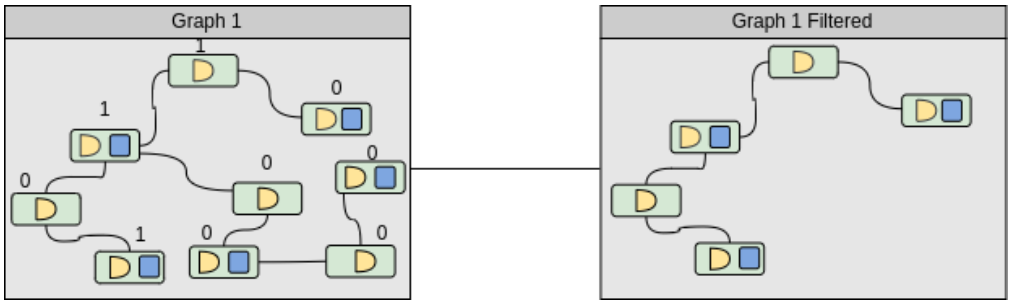


Figure 1.8.1: Στο κλάδεμα των γράφων κρατάμε μόνο κόμβους/αντικείμενα που είτε είναι σημαντικά είτε αποτελούν μέρος μιας σημαντικής τριπλέτας.

Εκπαίδευση

Περνάμε κάθε κλαδεμένο γράφο S από ένα νευρωνικό δίκτυο γράφου (GNN) για να πάρουμε μια ενσωμάτωση $g(S) \in \mathbb{R}^d$. Κατά τη διάρκεια της εκπαίδευσης παίρνουμε ζεύγη γράφων (S_i, S_j) και μετράμε την ομοιότητα των ενσωματώσεων που προκύπτουν μέσω του GNN, μέσω του εσωτερικού γινομένου τους:

$$f(S_i, S_j) = \langle g(S_i),\, g(S_j) \rangle,$$

και ζητάμε από το δίκτυό μας η τιμή αυτή να είναι ίδια με την ομοιότητα των δύο εικόνων:

$$s(I_i, I_j) = \frac{1}{25} \sum_{k=1}^{5} \sum_{\ell=1}^{5} c_{i,k}\, c_{j,\ell},$$

όπου $c_{i,k}$ είναι η ενσωμάτωση μίας από τις προτάσεις που περιγράφουν την εικόνα I_i. Έτσι ελαχιστοποιούμε το τετραγωνικό σφάλμα

$$L_{ij} = \big(f(S_i, S_j) - s(I_i, I_j)\big)^2,$$

και εκπαιδεύουμε το μοντέλο ώστε οι ενσωματώσεις που παράγει το GNN να καθρεφτίζουν τη σημασιολογική ομοιότητα των εικόνων βάσει των προτάσεών τους. Το συνολικό pipeline φαίνεται στην Εικόνα 1.8.2.

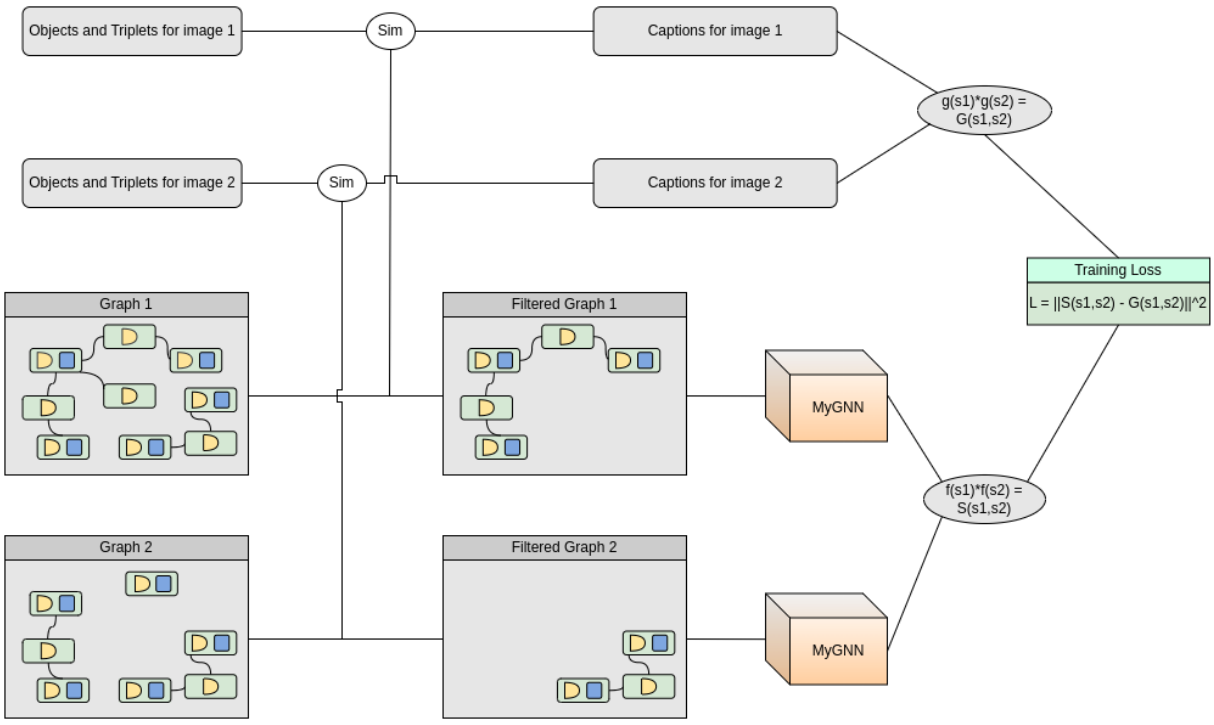


Figure 1.8.2: Η συνολική διαδικασία εκπαίδευσης του μοντέλου μας. Αρχικά φιλτράρουμε τους γράφους, έπειτα τους περνάμε από ένα νευρωνικό δίκτυο γράφων που παράγει μια συνολική ενσωμάτωση για κάθε γράφο. Τέλος, συγκρίνοντας (μέσω MSE) την ομοιότητα δύο γράφων με την πραγματική ομοιότητα βάσει της βασικής αλήθειας, εκπαιδεύουμε το μοντέλο μας.

1.8.1 Πρόταση Νευρωνικού Δικτύου Γράφου

Ένα συνηθισμένο μειονέκτημα πολλών αρχιτεκτονικών νευρωνικών δικτύων γράφων (π.χ. GAT, GIN κ.λπ.) είναι ότι κατά την ανταλλαγή μηνυμάτων δεν αξιοποιούνται οι ενσωματώσεις των σχέσεων στις ακμές, με αποτέλεσμα να χάνεται πληροφορία που θα μπορούσε να ενισχύσει την εκφραστικότητα του μοντέλου. Στους γράφους μας κάθε ακμή φέρει πλούσιες πληροφορίες για τη σχέση μεταξύ των αντικειμένων (π.χ. «άνθρωπος—παίζει με—σκύλο»). Ωστόσο, προηγούμενες μεθόδοι ανάκτησης εικόνας βάσει εικόνας είτε παραβλέπουν εντελώς αυτές τις πληροφορίες [99] είτε τις χρησιμοποιούν μόνο στο τελικό στάδιο του αλγορίθμου ανάκτησης [90], αντί να τις ενσωματώνουν κατά τη διάρκεια της μετάδοσης μηνυμάτων στο GNN.

Προτείνουμε μια τροποποίηση του GATv2 συνελικτικού δικτύου που αξιοποιεί τις ενσωματώσεις σχέσεων στις ακμές τόσο για τον υπολογισμό του συντελεστή προσοχής όσο και για τη μετάδοση μηνυμάτων από κόμβο σε κόμβο. Προηγούμενες προσπάθειες ενσωμάτωσης των πληροφοριών ακμών στο GAT χρησιμοποιούσαν τις ακμές μόνο στον υπολογισμό του συντελεστή προσοχής e_{ij} (με concatenation $[z_i \| e_{ij} \| z_j]$) και προαιρετικά

εισήγαγαν τις ενσωματώσεις ακμών μόνο στο τελευταίο στρώμα του GNN [13]. Αντίθετα, εμείς εισάγουμε τις ενσωματώσεις ακμών από το πρώτο στρώμα της αρχιτεκτονικής, ώστε τα επόμενα στρώματα να μάθουν περισσότερο εκφραστικές αναπαραστάσεις, γνωρίζοντας τόσο τις ενσωματώσεις κόμβων όσο και ακμών. Επιπλέον, χρησιμοποιούμε πράξη αθροίσματος αντί για την ένωση των ενσωματώσεων (concatenation) τόσο στη μετάδοση μηνυμάτων όσο και στον υπολογισμό του συντελεστή προσοχής. Επειδή οι κόμβοι αντικειμένων φέρουν προτασιακές και οπτικές ενσωματώσεις ενώ οι ενσωματώσεις ακμών είναι μόνο προτασιακές, αρχικά διαχωρίζουμε για κάθε γείτονα j τις ενσωματώσεις z_j σε z_j^{text} και z_j^{vis}. Στο προτασιακό και οπτικό κομμάτι εφαρμόζουμε γραμμικά στρώματα προβολής:

$$z_j^{\text{text}\prime} = W_t\, z_j^{\text{text}}, \quad z_j^{\text{vis}\prime} = W_v\, z_j^{\text{vis}}, \quad e'_{ij} = W_e\, e_{ij},$$

όπου $W_t \in \mathbb{R}^{d_{\text{out}} \times d_{\text{text}}}$, $W_v \in \mathbb{R}^{d_{\text{out}} \times d_{\text{vis}}}$, $W_e \in \mathbb{R}^{d_{\text{out}} \times d_{\text{edge}}}$. Έπειτα, για κάθε γείτονα j σχηματίζουμε μία μεικτή ενσωμάτωση

$$z'_j = \left(z_j^{\text{text}\prime} + e'_{ij}\right) \,\|\, z_j^{\text{vis}\prime},$$

ενώ ο κόμβος i διατηρεί την αρχική του ενσωμάτωση $z_i = z_i^{\text{text}} \| z_i^{\text{vis}}$. Τέλος, εφαρμόζουμε τον κλασικό μηχανισμό προσοχής του GATv2 πάνω στις νέες αυτές μεικτές ενσωματώσεις:

$$e_{ij} = a^\top \text{LeakyReLU}\big(W_{\text{att}}\,[\,z_i \| z'_j\,]\big), \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i)} \exp(e_{ik})}, \quad z'_i = \sigma\Big(\sum_{j \in N(i)} \alpha_{ij}\, W_m\, z'_j\Big).$$

Με αυτόν τον τρόπο οι ενσωματώσεις των κόμβων μετά το πρώτο στρώμα περιέχουν ήδη χρήσιμες πληροφορίες για τα αντικείμενα και τις σχέσεις τους. Δεδομένου ότι οι ενσωματώσεις ακμών δεν ανανεώνονται σε αυτό το στρώμα, τα επόμενα στρώματα χρησιμοποιούν κλασικά GATv2 στρώματα χωρίς περαιτέρω χρήση των ακμών. Μια αναπαράσταση του μηχανισμού μετάδοσης μηνυμάτων φαίνεται στο Σχήμα 1.8.3.

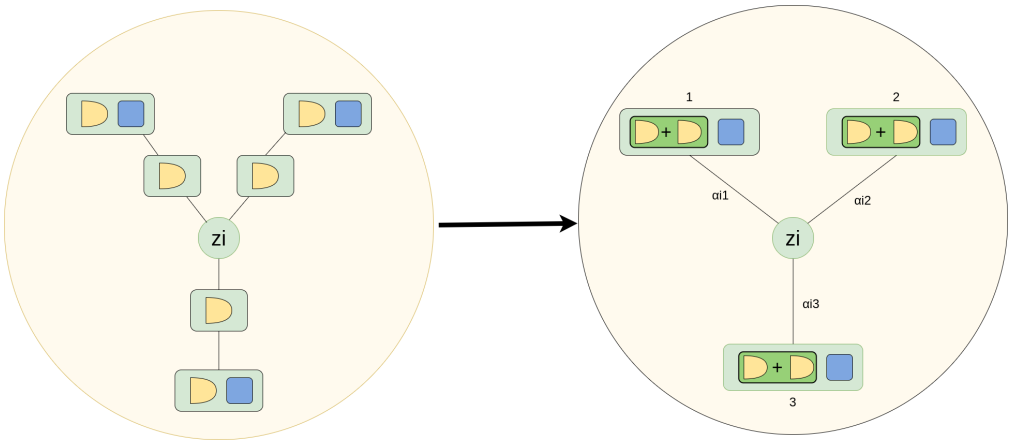


Figure 1.8.3: Η διαδικασία μετάδοσης μηνυμάτων που χρησιμοποιούμε στο πρώτο στρώμα του μοντέλου μας. Προσθέτουμε στο προτασιακό μέρος της ενσωμάτωσης των κόμβων, που παράγουν μηνύματα για άλλους κόμβους, την προτασιακή ενσωμάτωση των σχέσεων μεταξύ τους. Με αυτόν τον τρόπο οι ενσωματώσεις που προκύπτουν από το στρώμα αυτό περιέχουν πληροφορίες τόσο για τα αντικείμενα όσο και για τις σχέσεις μεταξύ τους.

1.8.2 Inference

Κατά τη διάρκεια του inference, για κάθε εικόνα–γράφο στο σετ δοκιμής υπολογίζουμε πρώτα τις τιμές σημαντικότητας για κάθε αντικείμενο και τριπλέτα χρησιμοποιώντας το μοντέλο πρόβλεψης σημαντικότητας. Έπειτα, όπως κατά την εκπαίδευση, επιλέγουμε τα σημαντικά αντικείμενα και τριπλέτες εφαρμόζοντας (i) ένα όριο τιμής και (ii) τον αλγόριθμο Jenks natural breaks, και παίρνουμε την ένωση των αποτελεσμάτων. Με αυτή τη μάσκα φιλτραρίσματος κλαδεύουμε τον γράφο, αφαιρώντας μη σημαντικούς κόμβους και ακμές. Τελικά, περνάμε τον κλαδεμένο γράφο από το νευρωνικό δίκτυο γράφου για να λάβουμε μια συνολική ενσωμάτωση, και η προβλεπόμενη ομοιότητα δύο εικόνων δίνεται από το εσωτερικό γινόμενο των αντίστοιχων ενσωματώσεών τους. Η διαδικασία inference φαίνεται στο Σχήμα 1.8.4.

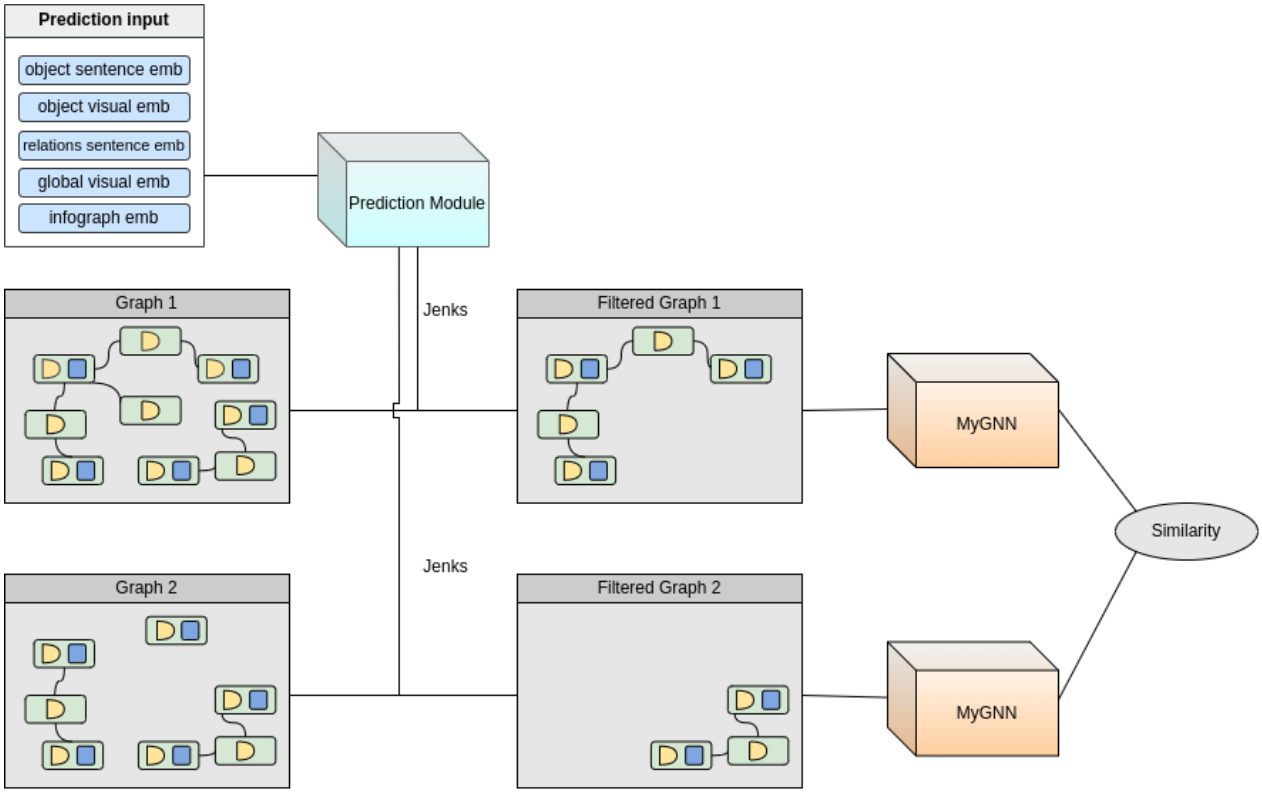


Figure 1.8.4: Κατά τη διάρκεια της πρόβλεψης, αρχικά φιλτράρουμε τους γράφους βάσει των προβλέψεων για σημαντικά αντικείμενα και τριπλέτες από το μοντέλο πρόβλεψης σημαντικότητας. Στη συνέχεια, περνάμε τους φιλτραρισμένους γράφους από ένα νευρωνικό δίκτυο γράφων εκπαιδευμένο όπως περιγράψαμε παραπάνω. Τέλος, η ομοιότητα δύο γράφων–εικόνων δίνεται από το εσωτερικό γινόμενο των αντίστοιχων ενσωματώσεων.

1.9 Πειραματικό Μέρος

1.9.1 Σύνολο Δεδομένων

Για το πειραματικό μέρος χρησιμοποιήσαμε εικόνες που υπάρχουν ταυτόχρονα στο PSG σύνολο δεδομένων και στο MS-COCO. Το PSG dataset παρέχει γράφους σκηνής υψηλότερης ποιότητας σε σχέση με το Visual Genome (είναι ουσιαστικά μια «καθαρότερη» έκδοση), ενώ το MS-COCO μας δίνει πέντε προτάσεις για κάθε εικόνα που περιγράφουν το περιεχόμενό της με σημασιολογικό τρόπο. Περιοριστήκαμε στις κοινές εικόνες και των δύο συνόλων, ώστε να έχουμε διαθέσιμες προτάσεις για τον υπολογισμό της «βασικής αλήθειας» της ομοιότητας μεταξύ εικόνων, καθώς και γράφους σκηνής για αυτές.

Έπειτα χωρίζουμε τυχαία το σύνολο δεδομένων σε 80 % για το σετ εκπαίδευσης, 12 % για το σετ επικύρωσης (validation) και 8 % για το σετ δοκιμής (test). Συγκεκριμένα, καταλήγουμε με 37 352 εικόνες για εκπαίδευση, 5 603 για επικύρωση και 3 736 για δοκιμή.

Οι γράφοι σκηνής περιέχουν συγκεκριμένες κατηγορίες αντικειμένων και σχέσεων. Για να τις επεξεργαστούμε, κατασκευάζουμε τις προτασιακές ενσωματώσεις τους χρησιμοποιώντας τον προτασιακό (sentence) Transformer του μοντέλου OpenCLIP_ViT_H_14_laion2b_s32b_b79k. Επιπλέον, χρησιμοποιούμε τον οπτικό (vision) Transformer του ίδιου μοντέλου για να λάβουμε οπτικές ενσωματώσεις των αντικειμένων στους γράφους.

Για την βασική τιμή της ομοιότητας δύο εικόνων παίρνουμε τις πέντε προτάσεις που περιγράφουν κάθε εικόνα και τις μετατρέπουμε σε ενσωματώσεις χρησιμοποιώντας το μοντέλο `all-mpnet-base-v2` από τη βιβλιοθήκη `sentence-transformers` της Python. Όπως αναφέραμε παραπάνω, υπολογίζουμε την ομοιότητα για κάθε ζεύγος προτάσεων ($5 \times 5 = 25$ συγκρίσεις) και παίρνουμε τον μέσο όρο αυτών για να προκύψει η τελική τιμή ομοιότητας (ground-truth).

1.9.2 Λεπτομέρειες μοντέλου πρόβλεψης σημαντικότητας

Ο βασικός σχεδιασμός του μοντέλου πρόβλεψης σημαντικότητας περιγράφεται στην Ενότητα 1.7.4 και φαίνεται στο Σχήμα 1.7.2. Για τη γενική προτασιακή ενσωμάτωση του γράφου που δίνουμε στο μοντέλο ως είσοδο (εκτός άλλων), εκπαιδεύσαμε ένα μη επιβλεπόμενο μοντέλο InfoGraph για 120 εποχές, ρυθμίζοντας την έξοδό του σε διάσταση 1024. Έπειτα, κάθε είσοδος στο μοντέλο προβάλλεται σε διάσταση 1536 μέσω ενός γραμμικού στρώματος — μεγαλύτερη ή μικρότερη διάσταση για αυτές τις προβολές μείωνε την απόδοση του μοντέλου μας.

Για τον κωδικοποιητή του Transformer βρήκαμε ότι τρία στρώματα, το καθένα με 32 κεφαλές προσοχής, έδιναν τα καλύτερα αποτελέσματα. Μετά το τελευταίο στρώμα του κωδικοποιητή, οι ανανεωμένες ενσωματώσεις περνούν από ένα επιπλέον multi-head attention στρώμα με εκπαιδευόμενα ερωτήματα (queries). Σε αυτό το στρώμα χρησιμοποιούμε 4 εκπαιδευόμενα ερωτήματα και πάλι 32 κεφαλές προσοχής. Στη συνέχεια ενώνονται οι τέσσερις προκύπτουσες ενσωματώσεις με λήψη του μέσου όρου τους και η προκύπτουσα ενσωμάτωση τροφοδοτείται σε ένα μικρό MLP δύο στρωμάτων με συναρτήσεις ενεργοποίησης GeLU. Το πρώτο στρώμα του MLP μειώνει τη διάσταση από 1536 σε 768, ενώ το δεύτερο από 768 σε 1, που είναι η τελική τιμή-αποτέλεσμα της σημαντικότητας.

Εκπαίδευση

Εκπαιδεύσαμε ένα ενιαίο μοντέλο να εντοπίζει τόσο τα σημαντικά αντικείμενα όσο και τις σημαντικές τριπλέτες ενός γράφου. Συγκεκριμένα, το σύνολο δεδομένων εκπαίδευσης περιέχει για κάθε γράφο όλες τις τριπλέτες του (αντικείμενο 1, σχέση, αντικείμενο 2) και όλα τα μονά αντικείμενα ως δείγματα της μορφής (αντικείμενο 1, 0, 0). Η επιλογή αυτή έγινε επειδή διαπιστώσαμε ότι ένα ενιαίο μοντέλο που χειρίζεται ταυτόχρονα τριπλέτες και μονά αντικείμενα είχε συνολικά καλύτερη απόδοση.

Λεπτομέρειες εκπαίδευσης

Εκπαιδεύσαμε το μοντέλο μας για 10 εποχές χρησιμοποιώντας τον βελτιστοποιητή Adam με ρυθμό εκμάθησης 1×10^{-5} και συνάρτηση σφάλματος , το μέσο τετραγωνικό σφάλμα (mean squared error). Στο σετ δεδομένων χρησιμοποιήσαμε batch size 32, τέσσερις διεργασίες εργασίας (worker threads) και τυχαίο ανακάτεμα των δειγμάτων (shuffling) σε κάθε εποχή. Για να περιορίσουμε το overfitting εφαρμόσαμε dropout 0.2 σε κάθε στρώμα του κωδικοποιητή Transformer. Επιπλέον, χρησιμοποιήσαμε χρονοπρογραμματιστή του ρυθμού εκμάθησης (learning rate scheduler) που πολλαπλασιάζει το ρυθμό εκμάθησης επί 0.9 μετά από κάθε εποχή και early stopping με υπομονή τριών εποχών.

Βασική αλήθεια και μετρικές

Το μοντέλο μας προβλέπει αριθμητικές τιμές $s \in [0, 1]$ που αναπαριστούν τη σημαντικότητα ενός αντικειμένου ή τριπλέτας. Η μάσκα που καθορίζει ποια αντικείμενα και τριπλέτες είναι σημαντικά σε μια εικόνα προκύπτει σε δύο βήματα:

1. **Όριο τιμής:** κάθε αντικείμενο ή τριπλέτα με $s \geq 0.4$ θεωρείται σημαντικό.

2. **Jenks natural breaks:** εφαρμόζουμε τον αλγόριθμο Jenks με δύο κλάσεις στις τιμές s. Αν το ελάχιστο σκορ της «σημαντικής» κλάσης και το μέγιστο σκορ της «μη-σημαντικής» κλάσης διαφέρουν κατά λιγότερο από 0.1, τότε ενοποιούμε τις δύο κλάσεις και θεωρούμε όλα τα δείγματα σημαντικά.

Το τελικό σύνολο αντικειμένων (και τριπλετών) που στη μάσκα χαρακτηρίζονται σημαντικά είναι η ένωση αυτών που επιλέχθηκαν και με τις δύο μεθόδους.

Για κάθε εικόνα αξιολογούμε τις συνεχείς προβλέψεις του μοντέλου με τις τιμές της «βασικής αλήθειας» χρησιμοποιώντας τους συντελεστές Spearman's r, Distance Correlation, MSE και MAE. Αντίστοιχα, για τις δυαδικές ετικέτες (μάσκα) συγκρίνουμε τις προβλέψεις με τη μάσκα της «βασικής αλήθειας» χρησιμοποιώντας τις μετρικές Accuracy, Precision, Recall και F1-score.

Για παράδειγμα, για μια εικόνα με επτά αντικείμενα, οι ground-truth τιμές σημαντικότητας και οι αντίστοιχες δυαδικές ετικέτες είναι

$$[0.1266, 0.1266, 0.4529, 0.4529, 0.3555, 0.0772, 0.0389] \longrightarrow [0, 0, 1, 1, 1, 0, 0],$$

ενώ οι προβλέψεις του μοντέλου μας είναι

$$[0.2829, 0.0723, 0.4650, 0.4475, 0.3467, 0.0323, 0.0413] \longrightarrow [1, 0, 1, 1, 1, 0, 0].$$

Ποσοτική Ανάλυση

Στην ενότητα αυτή θα αξιολογήσουμε την αποδοτικότητα του μοντέλου πρόβλεψης σημαντικότητας. Συγκρίνουμε για τον λόγο αυτό δύο μοντέλα: το πρώτο χρησιμοποιεί μετά τα στρώματα κωδικοποιητή Transformer ένα στρώμα multi-head attention με εκπαιδευόμενα ερωτήματα και το δεύτερο όχι. Επίσης αξιολογούμε την ικανότητα των μοντέλων μας ξεχωριστά για τα αντικείμενα και τις τριπλέτες.

Model	Spearman	Avg Dist. Corr.	Avg MSE	Avg MAE
Με εκπαιδευόμενες ερωτήσεις(queries)	**0.6655**	**0.8556**	**0.0100**	**0.0755**
Χωρίς εκπαιδευόμενες ερωτήσεις(queries)	0.6596	0.8529	0.0102	0.0763

Model	Acc.	Prec.	Rec.	F1
Με εκπαιδευόμενες ερωτήσεις(queries)	**0.8267**	**0.8445**	0.8356	**0.8400**
Χωρίς εκπαιδευόμενες ερωτήσεις(queries)	0.8214	0.8340	**0.8390**	0.8365

Table 1.1: Προβλέψεις σημαντικότητας τριπλετών για αντικείμενα στο σετ δοκιμής.

Model	Spearman	Avg Dist. Corr.	Avg MSE	Avg MAE
Με εκπαιδευόμενες ερωτήσεις(queries)	**0.5829**	**0.7690**	**0.0093**	**0.0694**
Χωρίς εκπαιδευόμενες ερωτήσεις(queries)	0.5813	0.7671	0.0094	0.0699

Model	Acc.	Prec.	Rec.	F1
Με εκπαιδευόμενες ερωτήσεις(queries)	**0.7891**	**0.6866**	0.6818	**0.6842**
Χωρίς εκπαιδευόμενες ερωτήσεις(queries)	0.7814	0.6707	**0.6826**	0.6766

Table 1.2: Προβλέψεις σημαντικότητας για αντικείμενα στο σετ δοκιμής.

Αλγόριθμος ένωσης Σε αυτό το σημείο αξιολογούμε τον αλγόριθμο κλαδέματος που περιγράψαμε παραπάνω στο να βρίσκει τα σημαντικά αντικείμενα. Αν και μια ακρίβεια στο 81 % μπορεί να φαίνεται μικρή, σημειώνουμε ότι η βασική αλήθεια για τη σημαντικότητα των αντικειμένων έχει θόρυβο και λάθη σε κάποιες περιπτώσεις λόγω του τρόπου υπολογισμού της. Για παράδειγμα, αν δύο αντικείμενα στον γράφο έχουν την ίδια ετικέτα («άνθρωπος»), και τα δύο θα λάβουν την ίδια τιμή σημαντικότητας, ενώ υπαρχει περίπτωση να είναι μόνο ο ένας "ανθρωπος" σημαντικός στην εικόνα , όχι και οι δύο.

Model	**Acc.**	**Prec.**	**Rec.**	**F1**
Με εκπαιδευόμενες ερωτήσεις(queries)	**0.8127**	**0.8095**	0.8109	**0.8102**
Χωρίς εκπαιδευόμενες ερωτήσεις(queries)	0.8051	0.7964	**0.8123**	0.8042

Table 1.3: Προβλέψεις σημαντικότητας αντικειμένων, μετά την ένωση των προβλεπόμενων ως σημαντικών αντικειμένων με εκείνα που συμμετέχουν σε μια σημαντική τριπλέτα.

Ποιοτική Ανάλυση

Στο σημείο αυτό παρουσιάζουμε μερικά παραδείγματα προβλέψεων του μοντέλου πρόβλεψης σημαντικότητας. Στους γράφους επισημαίνουμε με πράσινο χρώμα τα αντικείμενα που προβλέφθηκαν ως σημαντικά και με κόκκινο αυτά που προβλέφθηκαν μη σημαντικά. Αντίστοιχα, οι τριπλέτες που χαρακτηρίστηκαν σημαντικές σημειώνονται με πράσινο στο χρώμα της σχέσης, ενώ οι μη σημαντικές με κόκκινο.

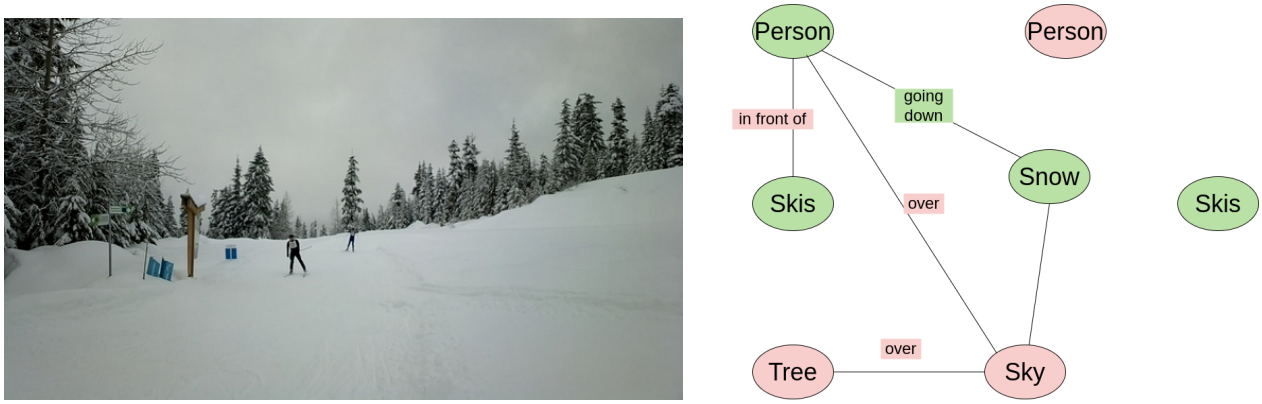


Figure 1.9.1: Παράδειγμα μιας εικόνας με τον γράφο της, όπου φαίνονται οι προβλέψεις του μοντέλου μας.

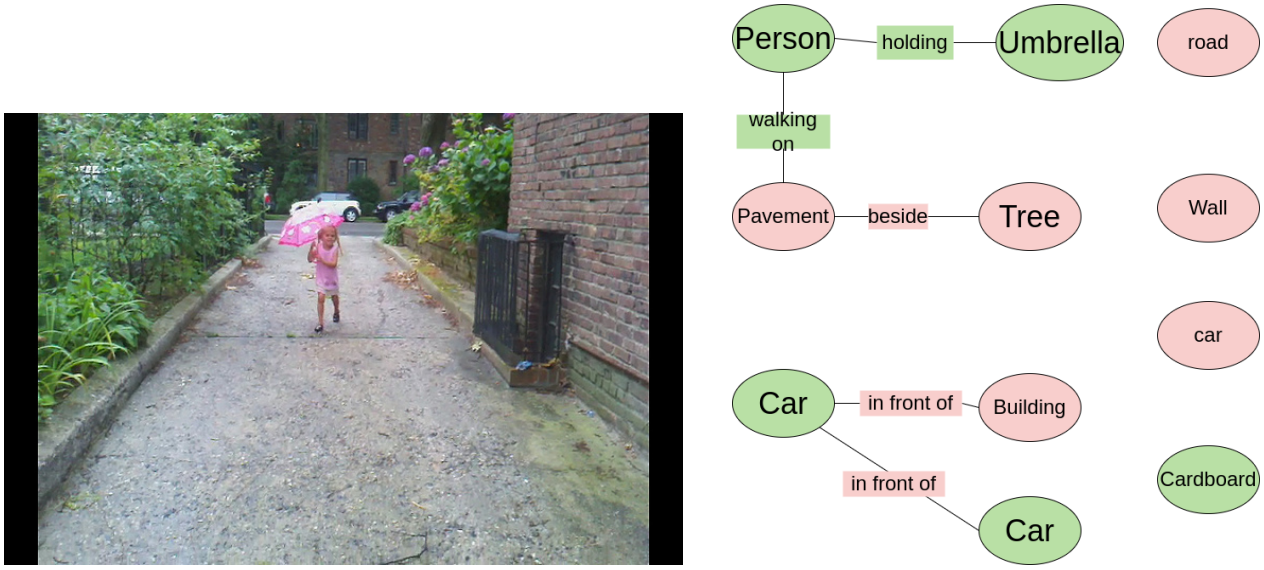


Figure 1.9.2: Παρατηρούμε ότι παρόλο που οι περισσότερες προβλέψεις του μοντέλου μας είναι διαισθητικά σωστές, υπάρχουν λάθη· π.χ. εδώ το "cardboard" προβλέφθηκε ως σημαντικό ενώ δεν φαίνεται στην εικόνα.

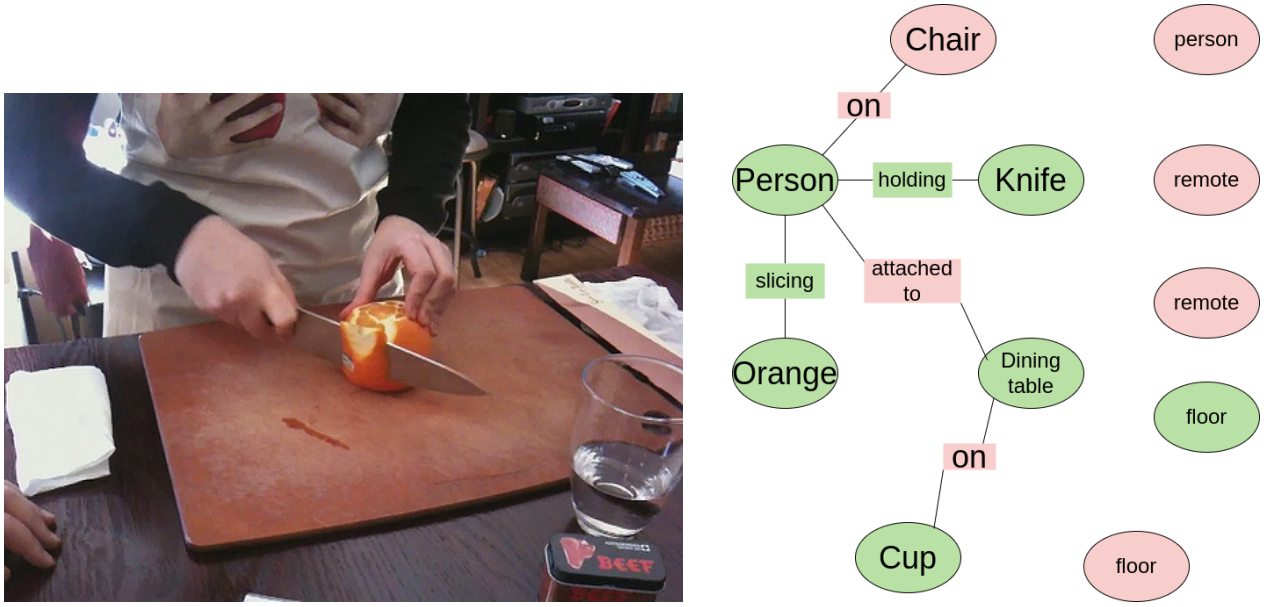


Figure 1.9.3: Το μοντέλο μας αξιολογεί σωστά στην εικόνα τα αντικείμενα που είναι σημαντικά διαισθητικά, καθώς και τις σημαντικές ενέργειες σε αυτή (slicing, holding).

1.9.3 Λεπτομέρειες Νευρωνικού Δικτύου Γράφων

Αρχιτεκτονική

Όπως περιγράψαμε στην Ενότητα 1.8.1, το μοντέλο που προτείναμε — GATv2 με επίγνωση των ακμών (Edge Aware GATv2) — έχει ως πρώτο στρώμα το στρώμα που προτείναμε και κατόπιν χρησιμοποιεί άλλα δύο στρώματα απλού GATv2. Οι κόμβοι του γράφου φέρουν ενσωματώσεις διαστάσεων 2053 (1024 προτασιακή ενσωμάτωση + 1029 οπτική ενσωμάτωση), ενώ οι ακμές φέρουν ενσωματώσεις σχέσεων διαστάσεων 1024. Στο πρώτο στρώμα (Edge Aware GATv2) προβάλουμε το προτασιακό και οπτικό κομμάτι των ενσωματώσεων των κόμβων σε 1024 διαστάσεις το καθένα, μέσω ενός γραμμικού στρώματος προβολής.

Οι καλύτερες παραμέτροι για το μοντέλο μας ήταν τρία στρώματα, όπως περιγράψαμε παραπάνω, με διαστάσεις $[3072, 2048, 2048]$ αντίστοιχα. Αυτές οι διαστάσεις λειτουργούν σαν αποκωδικοποιητής στο πρώτο στρώμα (με μεγαλύτερη διάσταση αρχικά και μειωμένες στα επόμενα). Διαισθητικά, στο πρώτο στρώμα αυξάνουμε τη διάσταση επειδή το μοντέλο συγκεντρώνει πληροφορίες από κόμβους και σχέσεις μέσω του αθροίσματος· αυτή η επέκταση της διάστασης βοηθά το δίκτυο να απομονώσει τις βασικές σχέσεις και να φιλτράρει τον θόρυβο που ενδέχεται να εισάγει η πράξη του αθροίσματος. Επιπλέον, όπως συνηθίζεται σε GNNs και CNNs, δεν εφαρμόζουμε residual σύνδεση στο πρώτο στρώμα, ενώ στα δύο επόμενα στρώματα προσθέτουμε residual συνδέσεις για να αποτρέψουμε το oversmoothing των ενσωματώσεων των κόμβων.

Μετά το πρώτο στρώμα (το δικό μας, που ενσωματώνει τις πληροφορίες ακμών), περνάμε τις ανανεωμένες αναπαραστάσεις των κόμβων στο επόμενο στρώμα ακριβώς ως έχουν. Μετά το δεύτερο και το τρίτο στρώμα (απλά GATv2 στρώματα) εφαρμόζουμε συνάρτηση ενεργοποίησης ReLU στις ενσωματώσεις των κόμβων. Τελικά, παίρνουμε την τελική αναπαράσταση για τους γράφους ,παιρνώντας τις ενσωματώσεις των κόμβων που προκύπτουν απο τα στρώματα GNN που περιγράψαμε από ένα γραμμικό στρώμα προβολής σε 2048 διαστάσεις και υπολογίζουμε τον μέσο όρο αυτών ως τη συνολική τελική ενσωμάτωση για τον γράφο.

Δοκιμάζουμε επίσης διάφορες άλλες αρχιτεκτονικές νευρωνικών δικτύων γράφων χωρίς το προτεινόμενο στρώμα μας για συγκριτικούς σκοπούς. Συγκεκριμένα, για μοντέλα με στρώματα GAT και GATv2 βρήκαμε ότι οι καλύτερες διαστάσεις ήταν $[3072, 2048, 2048]$, ενώ για GCN και GIN ήταν $[2048, 2048, 2048]$, με τις υπόλοιπες λεπτομέρειες και ρυθμίσεις τόσο στο μοντέλο όσο και στην διαδικασία εκπαίδευσης του, να παραμένουν όπως περιγράφηκαν παραπάνω.

Λεπτομέρειες εκπαίδευσης

Εκπαιδεύσαμε τα μοντέλα μας για 60 εποχές χρησιμοποιώντας τον Adam optimizer με ρυθμό εκμάθησης 10^{-4} για τις πρώτες 20 εποχές. Στις υπόλοιπες εποχές εφαρμόσαμε scheduler που μειώνει το ρυθμό εκμάθησης κατά 10 % μετά από κάθε εποχή. Για τις παραμέτρους του Adam χρησιμοποιήσαμε τις προεπιλεγμένες τιμές ($\beta_1 = 0.9$, $\beta_2 = 0.999$). Το batch size ορίστηκε σε 32 και εφαρμόσαμε dropout 0.1 σε κάθε συνελικτικό στρώμα. Για να περιορίσουμε περαιτέρω την υπερεκπαίδευση (overfitting) χρησιμοποιήσαμε πρόωρη διακοπή (early stopping) με υπομονή (patience) 20 εποχών.

Αξιολόγηση μοντέλου

Το σετ δοκιμής περιλαμβάνει 3 736 εικόνες. Για κάθε εικόνα—ερώτημα— ταξινομούμε (ranking) τις υπόλοιπες εικόνες με βάση την ομοιότητά τους στην εικόνα-ερώτημα, υπολογισμένη ως εσωτερικό γινόμενο των ενσωματώσεων που παρήγαγαν τα νευρωνικά δίκτυα γράφων. Στη συνέχεια συγκρίνουμε τη λίστα κατάταξης του μοντέλου μας με αυτή που προκύπτει από την «βασική αλήθεια» της ομοιότητας εικόνων. Χρησιμοποιούμε τις μετρικές NDCG, MAP, MRR και Recall για την αξιολόγηση.

Όπως φαίνεται στους πίνακες, το μοντέλο μας με εισαγωγή πληροφοριών ακμών στο πρώτο στρώμα ξεπερνά τα υπόλοιπα μοντέλα. Υπερτερεί και του μοντέλου IRGS [99] , το οποίο δεν χρησιμοποιεί ούτε οπτικές πληροφορίες για τα αντικείμενα ούτε πληροφορίες για τις ακμές. Επιπλέον, παρατηρούμε ότι τα μοντέλα με μηχανισμούς προσοχής (GAT, GATv2) επιτυγχάνουν γενικά καλύτερη απόδοση σε σύγκριση με παραδοσιακά μοντέλα όπως το GCN και το GIN.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.8803**	**0.8926**	**0.8994**	**0.9081**	**0.9143**	**0.9185**	**0.9216**
MyGATv2	0.8779	0.8908	0.8983	0.9071	0.9134	0.9176	0.9206
MyGAT	0.8758	0.8903	0.8969	0.9060	0.9120	0.9164	0.9197
MyGCN	0.8690	0.8792	0.8865	0.8963	0.9030	0.9080	0.9116
MyGIN	0.8432	0.8613	0.8699	0.8815	0.8897	0.8950	0.8992
IRSGS GCN	0.7401	0.7502	0.7559	0.7647	0.7734	0.7817	0.7891
IRSGS GIN	0.6979	0.7120	0.7217	0.7341	0.7433	0.7513	0.7580

Table 1.4: NDCG@k για τα μοντέλα.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.9269**	**0.9070**	**0.8782**	**0.8558**	**0.8363**	**0.8184**
MyGATv2	0.9069	0.9257	0.9049	0.8760	0.8534	0.8336	0.8157
MyGAT	0.9020	0.9241	0.9031	0.8732	0.8499	0.8301	0.8125
MyGCN	0.8943	0.9119	0.8878	0.8563	0.8337	0.8138	0.7959
MyGIN	0.8506	0.8857	0.8600	0.8285	0.8061	0.7874	0.7706
IRSGS GCN	0.6603	0.7331	0.6968	0.6495	0.6202	0.5989	0.5825
IRSGS GIN	0.5803	0.6635	0.6322	0.5901	0.5640	0.5456	0.5303

Table 1.5: MAP@k για τα μοντέλα.

Model	MRR
Edge Aware GATv2	**0.9469**
MyGATv2	0.9448
MyGAT	0.9425
MyGCN	0.9347
MyGIN	0.9080
IRSGS GCN	0.7703
IRSGS GIN	0.7059

Table 1.6: MRR για τα μοντέλα.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.8802**	**0.8512**	**0.8060**	**0.7619**	**0.7144**	**0.6639**
MyGATv2	0.9069	0.8757	0.8488	0.8039	0.7597	0.7124	0.6616
MyGAT	0.9020	0.8722	0.8419	0.7989	0.7558	0.7092	0.6604
MyGCN	0.8943	0.8531	0.8236	0.7806	0.7377	0.6929	0.6458
MyGIN	0.8506	0.8199	0.7932	0.7500	0.7119	0.6687	0.6244
IRSGS GCN	0.6603	0.6111	0.5759	0.5325	0.5036	0.4803	0.4559
IRSGS GIN	0.5803	0.5402	0.5174	0.4839	0.4560	0.4316	0.4099

Table 1.7: Precision@k για τα μοντέλα.

Πειραματική Μελέτη Απαλοιφής (Ablation Study)

Στην ενότητα αυτή μελετάμε την επίδραση διαφορετικών στοιχείων του pipeline με μια πειραματική μελέτη απαλοιφής.Συγκεκριμένα συγκρίνουμε την απόδοση των παρακάτων "μειωμένων" εκδόσεων του μοντέλου μας:

- **-Κλάδεμα (Pruning):** χωρίς κλάδεμα των γράφων.
- **-Multi-head Attention:** χωρίς το στρώμα multi-head attention με εκπαιδευόμενα ερωτήματα στο μοντέλο πρόβλεψης σημαντικότητας.
- **-Εισαγωγή πληροφοριών ακμής (Edge Injection):** χωρίς εισαγωγή πληροφοριών ακμής στο μοντέλο μας.
- **-Γενική οπτική ενσωμάτωση (Global):** οι γράφοι δεν περιέχουν κόμβο με τη γενική οπτική ενσωμάτωση της εικόνας.
- **-Σημαντικότητα τριπλετών (Triplet Significance):** κατά το κλάδεμα θεωρούμε σημαντικά μόνο τα αντικείμενα που προβλέπονται σημαντικά.
- **-Σημαντικότητα αντικειμένων (Object Significance):** κατά το κλάδεμα θεωρούμε σημαντικά μόνο τα αντικείμενα που είναι μέρος κάποιας σημαντικής τριπλέτας.
- **-Οπτικές πληροφορίες (Visual Information):** οι γράφοι δεν περιέχουν καμία οπτική πληροφορία, ούτε για τα αντικείμενα ούτε για τον κόμβο με τη γενική οπτική ενσωμάτωση.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.8803**	**0.8926**	**0.8994**	**0.9081**	**0.9143**	**0.9185**	**0.9216**
-EdgeInjection	0.8779	0.8908	0.8983	0.9071	0.9134	0.9176	0.9206
-Global	0.8426	0.8572	0.8653	0.8743	0.8808	0.8856	0.8893
-TripletSignificance	0.8721	0.8848	0.8921	0.9006	0.9074	0.9120	0.9150
-ObjectSignificance	0.8035	0.8160	0.8234	0.8348	0.8424	0.8485	0.8529
-Pruning	0.8758	0.8909	0.8979	0.9065	0.9128	0.9170	0.9202
-MultiheadAttention	0.8791	0.8925	0.8992	0.9079	0.9141	0.9184	0.9214
-VisualInformation	0.7518	0.7719	0.7841	0.8002	0.8115	0.8206	0.8271

Table 1.8: NDCG@k για τις διαφορετικές εκδοχές του μοντέλου μας.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.9269**	**0.9070**	**0.8782**	**0.8558**	**0.8363**	**0.8184**
-EdgeInjection	0.9069	0.9257	0.9049	0.8760	0.8534	0.8336	0.8157
-Global	0.8555	0.8841	0.8581	0.8238	0.7979	0.7770	0.7586
-TripletSignificance	0.8953	0.9171	0.8954	0.8664	0.8426	0.8230	0.8053
-ObjectSignificance	0.7808	0.8279	0.7969	0.7559	0.7295	0.7088	0.6915
-Pruning	0.9071	0.9240	0.9030	0.8746	0.8520	0.8324	0.8146
-MultiheadAttention	0.9109	0.9262	0.9065	0.8776	0.8553	0.8358	0.8180
-VisualInformation	0.6777	0.7534	0.7234	0.6873	0.6645	0.6470	0.6326

Table 1.9: MAP@k για τις διαφορετικές εκδοχές του μοντέλου μας.

Model	MRR
Edge Aware GATv2	**0.9469**
-EdgeInjection	0.9448
-Global	0.9095
-TripletSignificance	0.9371
-ObjectSignificance	0.8587
-Pruning	0.9439
-MultiheadAttention	0.9462
-VisualInformation	0.7881

Table 1.10: MRR για τις διαφορετικές εκδοχές του μοντέλου μας.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.8802**	0.8512	**0.8060**	0.7619	**0.7144**	**0.6639**
-EdgeInjection	0.9069	0.8757	0.8488	0.8039	0.7597	0.7124	0.6616
-Global	0.8555	0.8153	0.7857	0.7372	0.6930	0.6487	0.6039
-TripletSignificance	0.8953	0.8655	0.8374	0.7899	0.7481	0.7021	0.6520
-ObjectSignificance	0.7808	0.7378	0.7048	0.6613	0.6210	0.5826	0.5429
-Pruning	0.9071	0.8724	0.8477	0.8012	0.7594	0.7120	0.6619
-MultiheadAttention	0.9109	0.8799	**0.8515**	0.8057	**0.7620**	0.7143	0.6637
-VisualInformation	0.6777	0.6509	0.6287	0.6000	0.5714	0.5435	0.5129

Table 1.11: Precision@k για τις διαφορετικές εκδοχές του μοντέλου μας.

Από τους παραπάνω πίνακες είναι φανερό ότι το μοντέλο μας υπερέχει όλων των άλλων εκδοχών σε όλες τις μετρικές. Παρατηρούμε όμως ότι σε ορισμένες μετρικές η απόδοσή του είναι αρκετά κοντινή με άλλες "μειωμένες" εκδοχές. Συγκεκριμένα, το μοντέλο μας είναι σχετικά κοντά στις εκδοχές χωρίς κλάδεμα των γράφων και χωρίς εισαγωγή πληροφοριών ακμών. Αυτό μπορεί να συμβαίνει επειδή οι οπτικές ενσωματώσεις των αντικειμένων του γράφου περιέχουν ήδη αρκετή πληροφορία, καθιστώντας τις ενσωματώσεις ακμών λιγότερο κρίσιμες. Επιπλέον, καθώς το μοντέλο CLIP που χρησιμοποιούμε για τις προτασιακές και οπτικές ενσωματώσεις έχει εκπαιδευτεί σε ζεύγη εικόνων–προτάσεων (mapping εικόνας και λεζάντας στον ίδιο χώρο), μπορεί να αντλεί τις βασικές σχέσεις μεταξύ των αντικειμένων μέσω του κόμβου με τη γενική οπτική πληροφορία για όλη την εικόνα, χωρίς να χρειάζεται να τις μάθει ειδικά κατά την εκπαίδευση. Παρά τη μικρή διαφορά σε ορισμένες "μειωμένες" εκδοχές, το προτεινόμενο μοντέλο διατηρεί σταθερά την κορυφαία απόδοση σε όλες τις μετρικές. Τέλος, η εκδοχή χωρίς κλάδεμα γράφων αποδίδει επίσης καλά, διότι σε αρχιτεκτονικές με μηχανισμούς προσοχής όπως GAT και GATv2 οι άχρηστες ενσωματώσεις κόμβων μπορούν να ανιχνεύονται αυτόματα· ωστόσο, η προσθήκη φιλτραρίσματος γράφων βελτιώνει περαιτέρω τα αποτελέσματα.

Από τα παραπάνω αποτελέσματα είναι προφανές ότι οι οπτικές πληροφορίες των αντικειμένων και ο κόμβος με τη γενική οπτική αναπαράσταση της εικόνας προσφέρουν σημαντική ώθηση στην απόδοση. Επιπλέον, ο αλγόριθμος κλαδέματος που επιλέξαμε για τους κόμβους του γράφου αποδίδει πολύ καλύτερα σε σχέση με απλούστερες εναλλακτικές μεθόδους.

Σε αυτό το σημείο δοκιμάζουμε το μοντέλο μας και δύο εκδοχές του — μία χωρίς κλάδεμα των γράφων και μία χωρίς εισαγωγή πληροφοριών ακμών — σε γράφους που δεν περιέχουν καθόλου οπτικές πληροφορίες. Αυτό δεν γίνεται για να βελτιστοποιηθεί η απόδοσή τους (φυσικά αποδίδουν χειρότερα), αλλά για να μετρήσουμε πιο καθαρά την επίδραση του κλαδέματος και της εισαγωγής πληροφοριών ακμών: σε αυτούς τους γράφους οι διαφορές είναι πιο εμφανείς.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.7518**	**0.7719**	**0.7841**	**0.8002**	**0.8115**	**0.8206**	**0.8271**
-EdgeInjection	0.7465	0.7604	0.7727	0.7882	0.7994	0.8084	0.8153
-Pruning	0.7475	0.7601	0.7683	0.7808	0.7893	0.7961	0.8017
-EdgeInjection,Pruning	0.7414	0.7529	0.7617	0.7741	0.7826	0.7897	0.7954

Table 1.12: NDCG@k για διαφορετικές εκδοχές του μοντέλου μας και γράφους χωρίς οπτικές πληροφορίες.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.6777**	**0.7534**	**0.7234**	**0.6873**	**0.6645**	**0.6470**	**0.6326**
-EdgeInjection	0.6724	0.7388	0.7057	0.6694	0.6463	0.6287	0.6143
-Pruning	0.6761	0.7416	0.7065	0.6659	0.6404	0.6210	0.6051
-EdgeInjection,Pruning	0.6510	0.7268	0.6947	0.6538	0.6286	0.6096	0.5943

Table 1.13: MAP@k για διαφορετικές εκδοχές του μοντέλου μας και γράφους χωρίς οπτικές πληροφορίες.

Model	MRR
Edge Aware GATv2	**0.7881**
-EdgeInjection	0.7798
-Pruning	0.7815
-EdgeInjection,Pruning	0.7658

Table 1.14: MRR για διαφορετικές εκδοχές του μοντέλου μας και γράφους χωρίς οπτικές πληροφορίες.

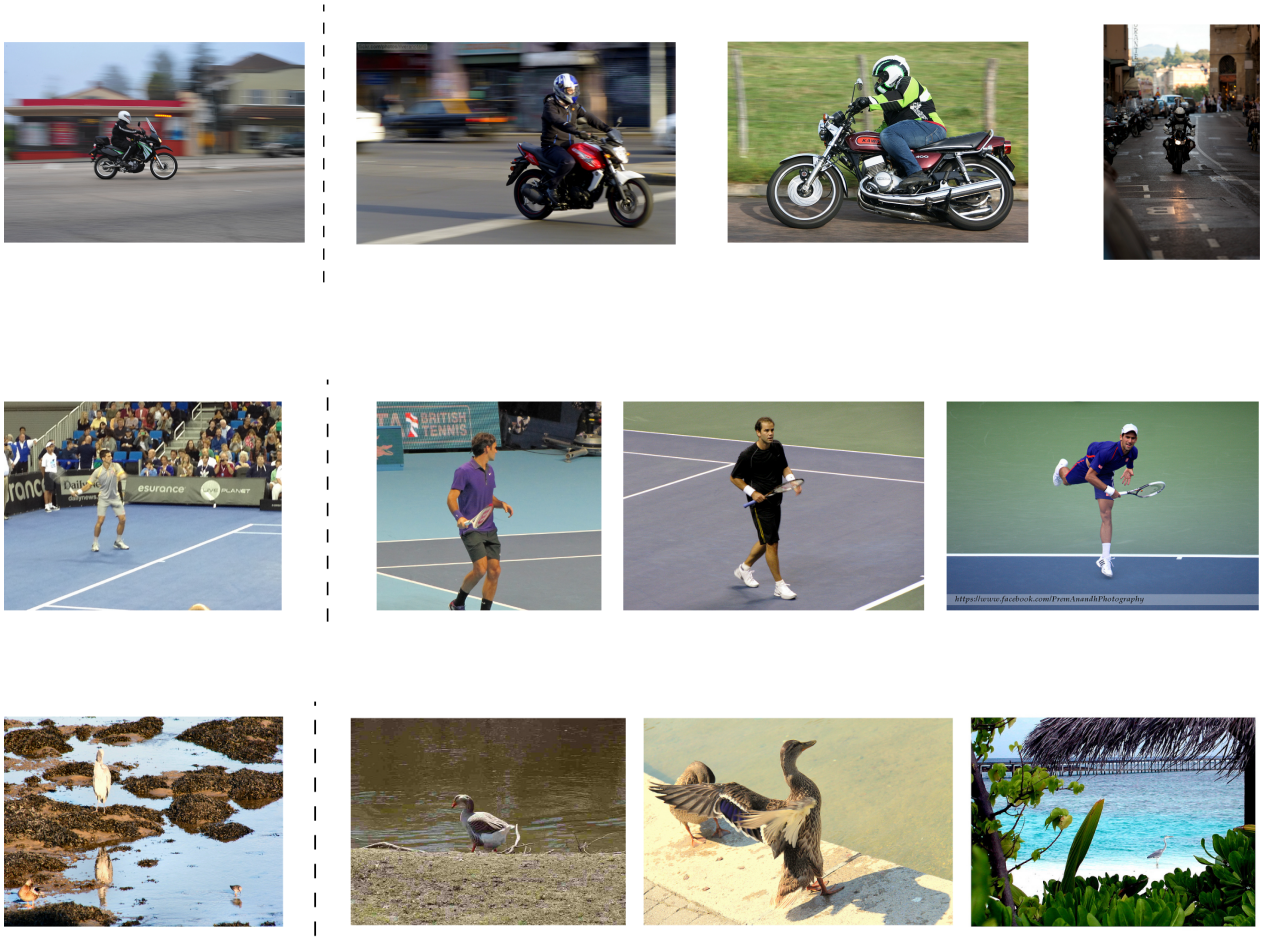
Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.6777**	**0.6509**	**0.6287**	**0.6000**	**0.5714**	**0.5435**	**0.5129**
-EdgeInjection	0.6724	0.6232	0.6069	0.5770	0.5496	0.5226	0.4958
-Pruning	0.6761	0.6278	0.6015	0.5637	0.5323	0.5027	0.4737
-EdgeInjection,Pruning	0.6510	0.6140	0.5875	0.5519	0.5208	0.4920	0.4641

Table 1.15: Precision@k για διαφορετικές εκδοχές του μοντέλου μας και γράφους χωρίς οπτικές πληροφορίες.

Από τους πίνακες αυτούς φαίνεται ξεκάθαρα ότι το μοντέλο μας βελτιώνεται με το κλάδεμα των γράφων και την εισαγωγή πληροφοριών για τις ακμές, όπως ακριβώς περιμέναμε.

1.9.4 Ποιοτικά Αποτελέσματα

Στην ενότητα αυτή παρουσιάζουμε κάποια ποιοτικά αποτελέσματα του μοντέλου μας. Συγκρίνουμε επίσης ενδεικτικά αποτελέσματα του μοντέλου με τη «βασική αλήθεια» και με άλλες εκδοχές του, για να τονίσουμε τα ιδιαίτερα χαρακτηριστικά του.



Από τα αποτελέσματα αυτά γίνεται εμφανής η σωστή αξιοποίηση τόσο της σημασιολογικής πληροφορίας από τους γράφους όσο και των οπτικών πληροφοριών των εικόνων. Στο πρώτο παράδειγμα , η εικόνα που ανακτά το μοντέλο μας είναι πολύ παρόμοια με την εικόνα–ερώτημα (και οι δύο έχουν θολό υπόβαθρο που υποδηλώνει κίνηση, τα ίδια αντικείμενα και παρόμοιες δράσεις). Στο δεύτερο παράδειγμα , όλες οι ανακτώμενες εικόνες έχουν μπλε γήπεδο, παρόλο που το σετ δοκιμής περιέχει και πράσινα γήπεδα με παρόμοιο σημασιολογικό περιεχόμενο. Η συνδυασμένη χρήση της σημασιολογικής πληροφορίας από τους γράφους φαίνεται ακόμη πιο καθαρά στα παραδείγματα 5–7. Στο παράδειγμα 6 , μάλιστα, τονίζουμε ότι παρότι η εικόνα–ερώτημα είναι έγχρωμη, το μοντέλο μας ανακτά μια ασπρόμαυρη εικόνα με αντίστοιχο σημασιολογικό χαρακτήρα.












Cupcakes Under Cover

1

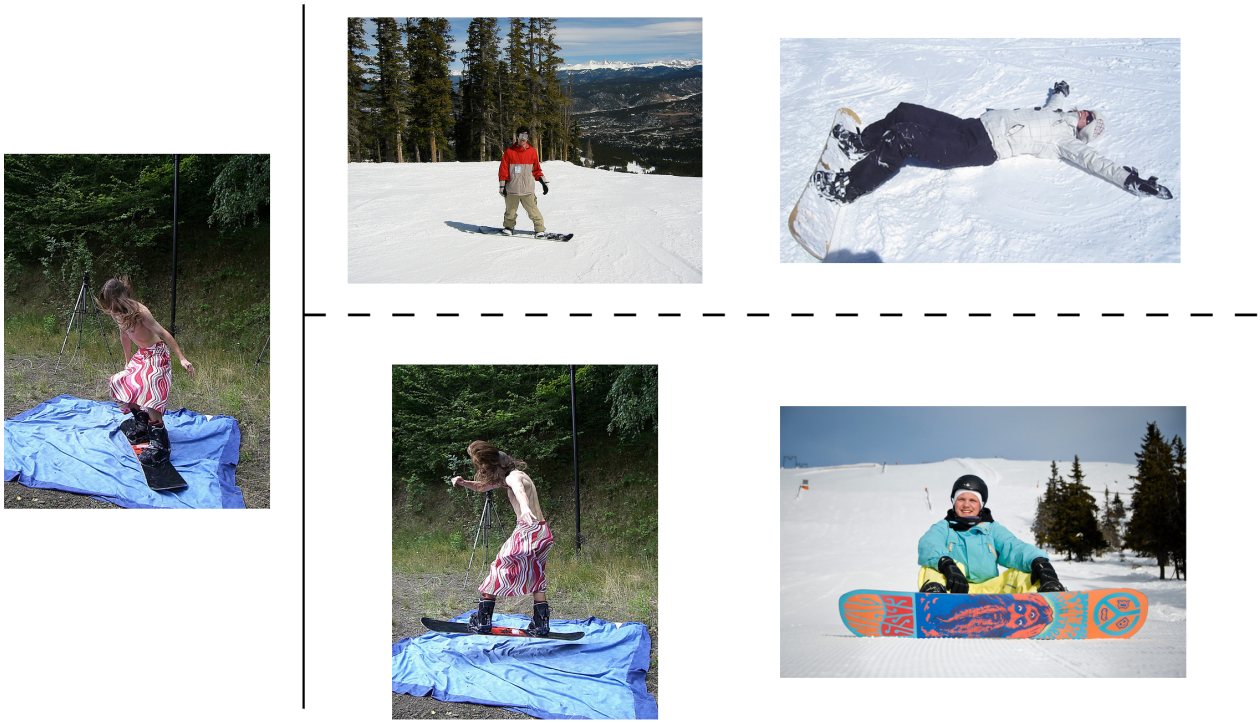



Figure 1.9.4: Στο παράδειγμα αυτό το μοντέλο μας ανακτά επιτυχώς την πιο όμοια εικόνα με βάση τις οπτικές πληροφορίες. Αριστερά φαίνεται η εικόνα–ερώτημα, δεξιά πάνω οι πιο όμοιες εικόνες με βάση τη «βασική αλήθεια», ενώ δεξιά κάτω οι εικόνες που ανακτήθηκαν από το μοντέλο μας.

Figure 1.9.5: Αριστερά φαίνεται η εικόνα–ερώτημα, δεξιά πάνω οι πιο όμοιες εικόνες με βάση τη «βασική αλήθεια», ενώ δεξιά κάτω οι εικόνες που ανακτήθηκαν από το μοντέλο μας.

Από τα παραπάνω παραδείγματα (1.9.4,1.9.5) φαίνεται ότι, παρόλο που σε αρκετές περιπτώσεις το μοντέλο μας ανακτά διαφορετικές εικόνες από τη «βασική αλήθεια», τα αποτελέσματά του παραμένουν αρκετά παρόμοια με την εικόνα–ερώτηση.

Στο σημείο αυτό παρουσιάζουμε ενδεικτικά παραδείγματα που αναδεικνύουν την ικανότητα του μοντέλου μας —λόγω της εισαγωγής πληροφοριών ακμών στο πρώτο στρώμα— να ανακτά εικόνες με πλουσιότερο σημασιολογικό χαρακτήρα σε σχέση με τις σχέσεις και τις δράσεις μεταξύ των αντικειμένων. Συγκεκριμένα, δείχνουμε την εικόνα–ερώτημα και τον κλαδεμένο γράφο της, καθώς και την εικόνα και τον γράφο που ανακτήθηκαν από το *Edge Aware GATv2* (με πληροφορίες ακμών) και, αντίστοιχα, από την εκδοχή που χρησιμοποιεί μόνο τα απλά στρώματα GATv2 (χωρίς το πρώτο στρώμα εισαγωγής πληροφοριών ακμών).

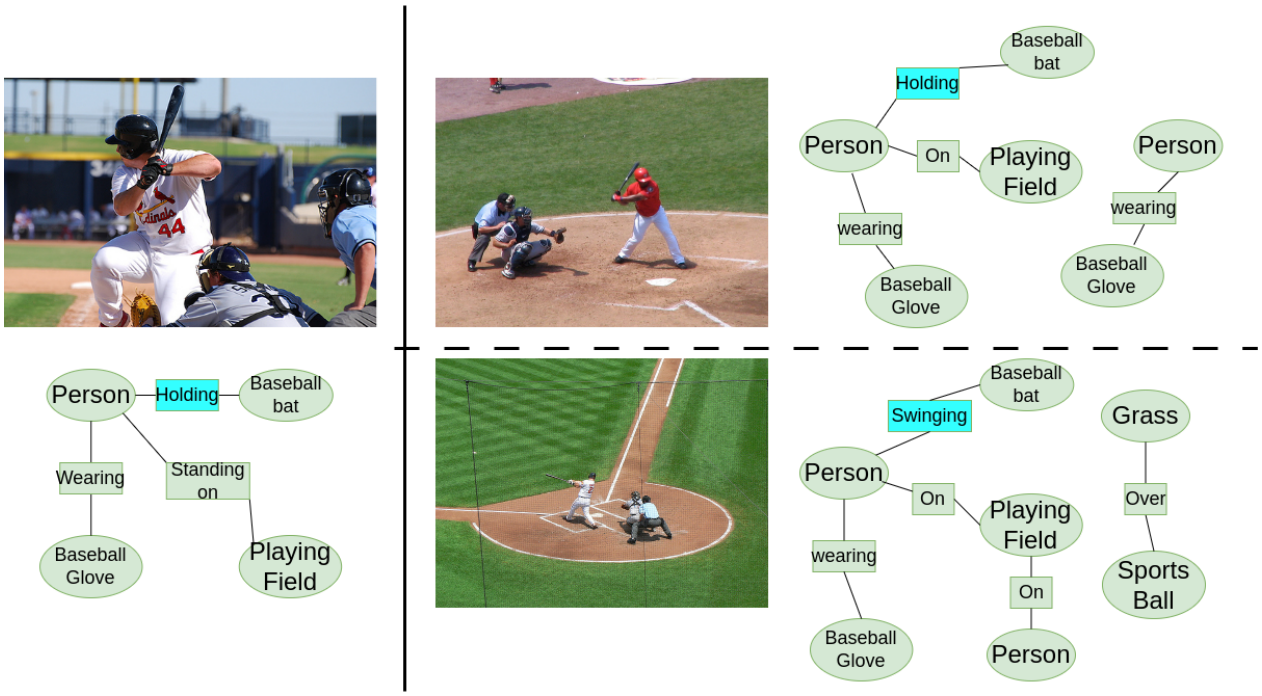


Figure 1.9.6: Στο παράδειγμα, το μοντέλο μας με εισαγωγή πληροφοριών ακμής ανιχνεύει σωστά ότι η σχέση στην εικόνα–ερώτηση είναι ο «άνθρωπος κρατάει (holding) το ρόπαλο», και όχι «βαράει (swinging) την μπάλα» με αυτό.

Παράλληλα, παρουσιάζουμε παραδείγματα για γράφους που δεν περιέχουν καθόλου οπτικές πληροφορίες, ώστε να αναδείξουμε ακόμη περισσότερο την ικανότητα του μοντέλου μας να ανιχνεύει σωστά τις σχέσεις των αντικειμένων μέσω των ενσωματώσεων ακμών. Σε αυτές τις περιπτώσεις, όπου οι κόμβοι φέρουν μόνο σημασιολογικές ενσωματώσεις, οι πληροφορίες στις ακμές αποκτούν ακόμα πιο κρίσιμο ρόλο στην ανάκτηση εικόνων.

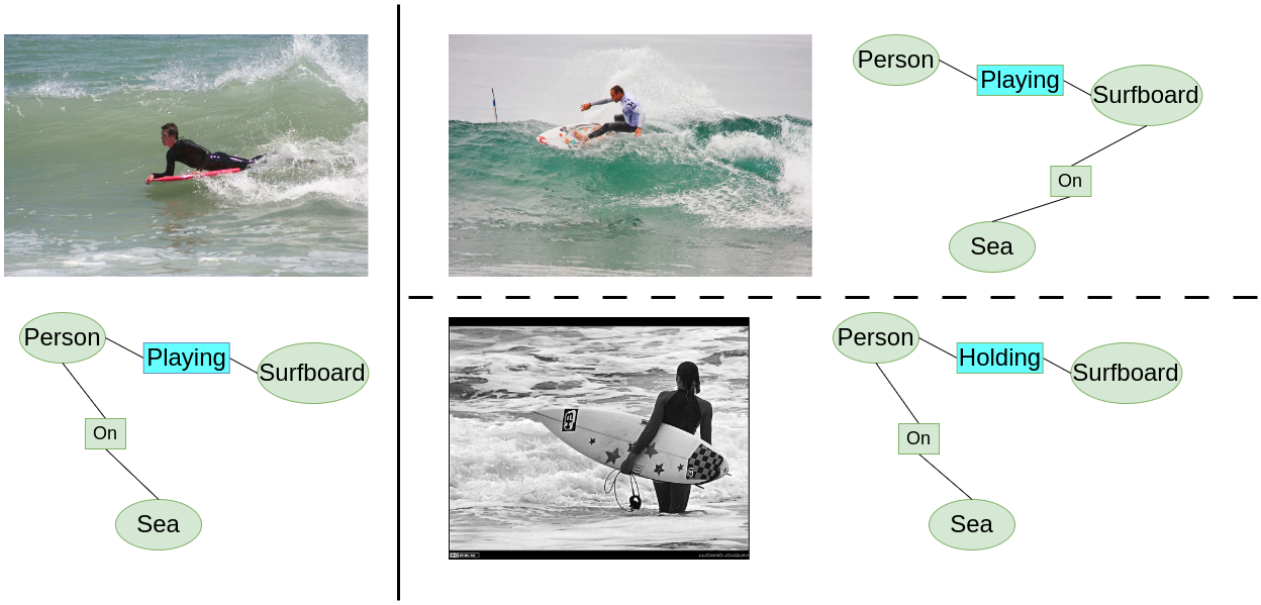


Figure 1.9.7: Στα αριστερά φαίνεται η εικόνα–ερώτημα. Πάνω δεξιά η εικόνα που ανέκτησε το καλύτερο μοντέλο μας (Edge Aware GATv2), το οποίο χρησιμοποιεί πληροφορίες για τις ακμές–σχέσεις. Κάτω δεξιά η εικόνα που ανακτά το ίδιο μοντέλο χωρίς εισαγωγή πληροφοριών για τις ακμές (GATv2).

Στο παραπάνω παράδειγμα (1.9.7) το μοντέλο μας με εισαγωγή πληροφοριών ακμής ανακτά εικόνες με τη σωστή σχέση μεταξύ των αντικειμένων.Το μοντέλο μας εντοπίζει ότι η σχέση ανάμεσα στον «άνθρωπο» και την «σανίδα του σερφ» στην εικόνα-ερώτημα είναι το «παίζει» (playing), ενώ το μοντέλο χωρίς πληροφορίες ακμών ανακτά εικόνα με τη σχέση «κρατάει» (holding).

1.10 Συμπεράσματα

Στην παρούσα διπλωματική εισάγαμε οπτικές πληροφορίες στην ανάκτηση εικόνων με χρήση γράφων σκηνής. Συγκεκριμένα, προτείναμε ένα μοντέλο πρόβλεψης σημαντικότητας το οποίο εκπαιδεύεται ώστε να διακρίνει τα σημαντικά αντικείμενα σε έναν γράφο–εικόνα με βάση τόσο τα ίδια τα αντικείμενα όσο και τις οπτικές πληροφορίες τους. Έπειτα, σχεδιάσαμε έναν αλγόριθμο κλαδέματος των γράφων που διατηρεί μόνο τα πλέον σημαντικά αντικείμενα και τις σχέσεις τους. Αυτά τα βήματα εμπνεύστηκαν από την παρατήρηση ότι το σύνολο δεδομένων Panoptic Scene Graph Generation περιέχει πολλά ασήμαντα αντικείμενα στις σημειώσεις του.

Εισάγαμε οπτικές πληροφορίες για τα αντικείμενα στους γράφους μας, καθώς και έναν κόμβο με γενική οπτική ενσωμάτωση ολόκληρης της εικόνας, κατασκευάζοντας έτσι πλούσιους σε πληροφορία γράφους για κάθε εικόνα. Για την επεξεργασία αυτών των γράφων προτείναμε και χρησιμοποιήσαμε μια τροποποιημένη έκδοση του GATv2 που ενσωματώσει τόσο στον υπολογισμό των συντελεστών προσοχής όσο και στη μετάδοση μηνυμάτων μεταξύ κόμβων τις ενσωματώσεις των ακμών–σχέσεων. Με αυτόν τον τρόπο, το δίκτυό μας είναι ικανό να παράγει πιο σημασιολογικά πλούσιες αναπαραστάσεις των γράφων.

Δοκιμάσαμε διάφορους τύπους νευρωνικών δικτύων γράφων (GCN, GIN, GAT, GATv2) και τη δική μας πρόταση (*Edge Aware GATv2*), αξιολογώντας την απόδοσή τους με μετρικές που χρησιμοποιούνται σε συστήματα πρότασης και ανάκτησης (MRR, MAP, NDCG, Precision). Το μοντέλο μας (*Edge Aware GATv2*) ξεπέρασε όλα τα υπόλοιπα σε κάθε μετρική, αν και σε ορισμένες περιπτώσεις οι διαφορές ήταν μικρές.Για να διερευνήσουμε περαιτέρω τις δυνατότητες και τους περιορισμούς του, παρουσιάσαμε ενδεικτικά παραδείγματα ανάκτησης όπου αναδείχθηκαν τόσο τα πλεονεκτήματα όσο και οι αδυναμίες του μοντέλου.

1.11 Μελλοντικές κατευθύνσεις

Τελειώνοντας αυτή την διατριβή, προτείνουμε κάποιες μελλοντικές ιδέες και κατευθύνσεις για τη βελτίωση της απόδοσης μοντέλων ανάκτησης εικόνας με βάση εικόνα.

- Αρχικά, ένα σημαντικό κομμάτι που μειώνει την αποδοτικότητα του μοντέλου μας είναι ο υπολογισμός της «βασικής αλήθειας» για τη σημαντικότητα αντικειμένων και τριπλετών σε μια εικόνα. Πιο αξιόπιστες τιμές σημαντικότητας σε ορισμένες λανθασμένες περιπτώσεις πιστεύουμε ότι θα βελτιώσουν τη συνολική απόδοση του μοντέλου μας.
- Ένας περιορισμός στην αναπαράσταση των σχέσεων στο PSG είναι ότι οι γράφοι που χρησιμοποιήθηκαν είναι μη κατευθυνόμενοι, με το ίδιο label και στις δύο διευθύνσεις. Για παράδειγμα, αν η σχέση μεταξύ του κόμβου «γραφείο» και του «ποτηριού» είναι «πάνω», τότε στην αντίστροφη διεύθυνση έχουμε επίσης «γραφείο»–«πάνω»–«ποτήρι», ενώ πιο σωστά θα ήταν να χρησιμοποιείται το label «κάτω». Η εισαγωγή κατευθυνόμενων ακμών με ορθά labels θα μπορούσε να βελτιώσει περαιτέρω την απόδοση του Edge Aware GATv2, το οποίο αξιοποιεί τις πληροφορίες στις ακμές.
- Παρόλο που το Edge Aware GATv2 εισάγει τις πληροφορίες ακμών μόνο στο πρώτο στρώμα και αυτές μεταφέρονται στα επόμενα, μια αρχιτεκτονική που χρησιμοποιεί επαναλαμβανόμενα τις ενσωματώσεις ακμών σε κάθε στρώμα — όπως γίνεται με τις ενσωματώσεις των κόμβων — θα πρέπει να ανανεώνει τόσο τις ενσωματώσεις των κόμβων όσο και των ακμών σε κάθε επίπεδο. Επιπλέον, η χρήση διαφορετικών συναρτήσεων για την παραγωγή της συνολικής ενσωμάτωσης γράφου εκτός της μέσης τιμής (π.χ. attention pooling) μπορεί να βελτιώσει το μοντέλο μας. Τέλος, η πρότασή μας (Edge Aware GATv2) θα πρέπει να αξιολογηθεί και σε διαφορετικά σύνολα δεδομένων ανάκτησης εικόνων.
- Επίσης, σε αυτή τη διατριβή δοκιμάσαμε τις κλασικές αρχιτεκτονικές νευρωνικών δικτύων γράφων—GCN, GAT, GIN και GATv2. Ωστόσο, πολλές σύγχρονες αρχιτεκτονικές που ενσωματώνουν πληροφορίες ακμών θα μπορούσαν να δοκιμαστούν για τον σκοπό μας. Επιπλέον, θα ήταν χρήσιμο να εξερευνήσουμε την απόδοση του pipeline με διαφορετικά μοντέλα για την εξαγωγή προτασιακών και οπτικών ενσωματώσεων των κόμβων, ώστε να βελτιωθεί η ποιότητα των ενσωματώσεων στους γράφους.

Chapter 2

Introduction

Artificial intelligence technologies have, in recent years, become a leading driver of technological innovation, transformed industries, and become a critical part of many widely used applications. A few examples of artificial intelligence in our everyday lives include self-driving vehicles, speech and language recognition and production systems, medical applications like cancer detection, pharmaceutical applications for molecular structure properties, recommender systems widely used in social media and on the web, weather prediction, stock market analysis, etc.

One emerging class of neural networks gaining traction in recent years is graph neural networks (GNNs), which are capable of handling data in graph form. GNNs overcome the challenges of graph-structured data, which—unlike images with a regular grid—cannot be processed easily by convolutional networks. This advancement has been crucial for domains naturally described by graphs, such as social networks, molecular structures, and even images, where scene graphs can represent objects along with their connections and relationships, providing rich semantic information.

The main focus of this thesis is image-to-image retrieval: retrieving the most relevant images, judged by both semantic and visual information, given a query image. Image retrieval has numerous applications in recommendation systems and has been explored from many angles. Traditional approaches—such as processing images with a convolutional network or a vision transformer and comparing their embeddings—ignore the rich semantic information contained in images. Instead, we will utilize scene graphs, which provide a structured semantic representation of images, alongside visual features of objects and the entire image to achieve a more comprehensive semantic and visual representation for retrieval.

Specifically, we will explore different GNN variants that, given fused information, namely scene graphs enriched with visual features, can generate representations of images suitable for semantic and visual image-to-image retrieval.We will also propose a model that determines the relative importance of each object and relation in the scene. Furthermore, since some literature already explores the technique of utilizing scene graphs for image retrieval, we will propose improvements and modifications to existing GNN layers and to our pipeline to achieve superior performance.

Specifically, the outline of this thesis is as follows:

- First, we discuss the theoretical background of machine learning concepts, covering general ideas in deep learning and artificial intelligence, and the technologies used in our work. We then focus on graph theory and each GNN variant we will use, exploring these networks theoretically, explain their key characteristics, advantages, and limitations, and discuss how existing methods address those limitations.
- Next, we discuss the task of image-to-image retrieval and review existing ideas in the literature, highlighting the motivations behind our contributions.
- Finally, we propose our pipeline for image-to-image retrieval. We introduce a module to predict the important parts of an image (objects and relationships) and a GATv2-inspired layer that incorporates edge information into message passing, thus enhancing the use of semantic information. We explain

the intuition behind our design decisions and analytically present our models' performance. We also explore different GNN variants, compare their results quantitatively and qualitatively, and highlight both the advances and limitations of our model.

Chapter 3

Machine Learning

Machine learning is a branch of computer science focused on creating algorithms and statistical models that learn from data, recognize patterns, and then make predictions or generate outputs on their own. In recent years, improvements in computational power, data availability, and algorithm design have turned machine learning into one of the fastest-growing and most influential areas of research. At its core, the aim is to build systems that improve their performance on a task by processing large amounts of data—much like humans improve with practice and experience.

The roots of machine learning trace back to early statistical methods and pattern-recognition techniques. Simple techniques—such as clustering algorithms, support vector machines, and decision trees—showed how computers could extract useful structure from data for tasks like grouping and classification. However, the field truly took off with the arrival of deep learning. This shift enabled breakthroughs in image recognition, speech processing, and natural language understanding that were previously out of reach.

Machine learning now includes a wide range of techniques and applications. Large language models produce human-level text, powering chatbots, translation tools, and code-generation assistants. Vision models—whether convolutional networks or transformer-based architectures—interpret images and video with superhuman accuracy. Generative models can create realistic images, audio, and even video from scratch. Graph neural networks capture relationships in complex structures like social networks or molecular graphs, and multimodal systems integrate insights across text, vision, and audio. Together, these advances are transforming industries, producing scientific discoveries, and incorporating artificial intelligence systems into everyday products.

Contents

3.1 Machine Learning Types

Machine learning algorithms can be grouped by how a model learns from data. Depending on the problem and the available dataset, different training paradigms and goals (loss functions) can be utilized to extract information. Different types of machine learning enable us to tackle a wide range of tasks using whatever data is available.

Unsupervised learning

The model learns from unlabeled data. It discovers structure by analyzing relationships between data points rather than relying on explicit labels. Common algorithms include clustering methods (e.g. k-means [58], DBSCAN [23]) and dimensionality-reduction techniques (e.g. PCA [35]).

Supervised learning

Each input is paired with a target output. The model learns to generalize from those examples so it can predict correct outputs on unseen inputs. This is the most used category, encompassing support vector machines [16] , decision trees [57], and many neural-network architectures.

Semi-supervised learning

A hybrid between supervised and unsupervised learning, where only a small portion of the data is labeled. Algorithms maximize learning by leveraging both labeled and unlabeled examples. Common applications include image classification when annotating every image is impractical.

Reinforcement learning

Rather than fixed input–output pairs, the model learns by interacting with an environment and receiving feedback (rewards or penalties). Widely used in robotics, game engines (e.g. AlphaGo [61]), and autonomous control.

Multitask learning

A single model is trained simultaneously on multiple related tasks, sharing representations to improve performance on each. For example, a neural network might learn to recognize both objects and their attributes in an image.

Transfer learning

Knowledge learned on one task or domain is transferred to improve learning on a different, but related, task. Pretrained language models (like BERT [18]) fine-tuned for sentiment analysis are a common example.

Few-shot / One-shot / Zero-shot learning

Models are designed to generalize from very few (few/one) or no (zero) labeled examples of a new class. Techniques often rely on metric learning or prompt-based adaptation in large pretrained models.

3.2 Embeddings

Embeddings are vector representations of data—objects, concepts, or relations—designed so that similar items lie close together in vector space. The key idea is that geometric proximity reflects semantic or structural similarity, allowing us to distinguish between different types of data. In other words, embeddings can summarize rich information about inputs, and by choosing different objectives we can emphasize the aspects most relevant to our task.

A familiar example is sentence embeddings, where sentences with similar meaning map to nearby vectors [69]. Likewise, image embeddings place visually or semantically related images close together. Popular models for

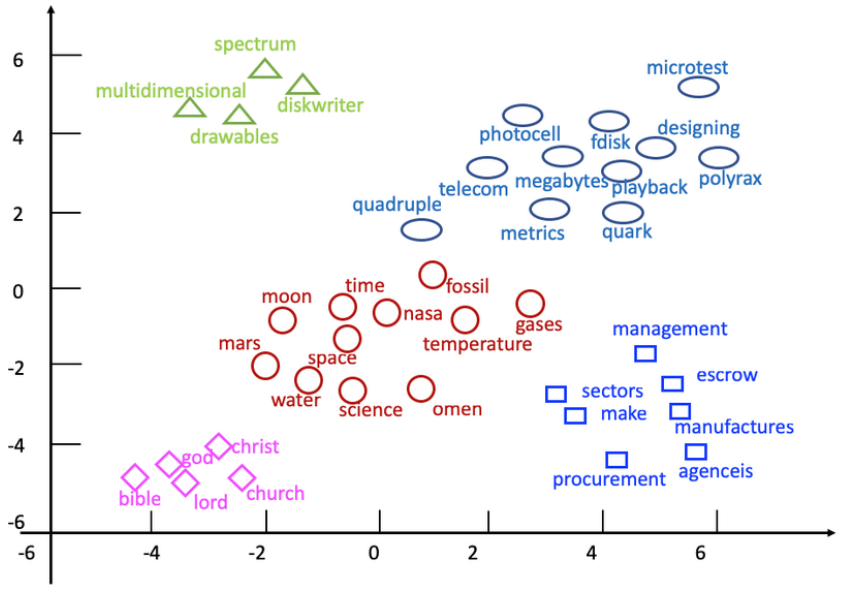


Figure 3.2.1: Example of the projection of word embeddings in two dimensions [51].

producing embeddings include Sentence-BERT [69] for text, and visual backbones like ResNet—or more recent vision-language models such as CLIP [67] and BLIP [54]—for images.

3.3 Clustering Algorithms

Clustering is an unsupervised machine-learning technique in which input data—often represented as embeddings—are grouped into clusters based on a similarity measure (typically geometric distance in the embedding space). By automatically assigning data points to clusters, we can uncover hidden structure, identify natural groupings, and even generate provisional labels for downstream tasks.

- **k-means:**The algorithm picks in the start some random points in the grid called centroids and assignes each data point to the centroid which is closer to it, then the position of the centroids is updated by finding the average position of all data points assigned to it . This process repeats until the data points assigned to each centroid does not change. As the process suggest k means is a centroid based clustering algorithm [58].

- **DBSCAN:** DBSCAN [23]is a density-based clustering algorithm , unlike k means where the clusters have spherical form the clusters forming from DBSCAN can have arbitrary shapes. It first finds core points which are points whith a sufficient number of neighbors within a radius specified by a hyperparameter e , then each of the points within this radius is assigned to the cluster of the core point and for each of the newly assigned points,if within a radius e exist another point not already assigned to a cluster then it is assigned also to the same cluster . DBSCAN also handles properly outliers because after processing all points any point that is not assigned to a cluster is labeled as noise .

- **Jenks natural breaks:** This is a distribution based clustering algorithm , that seeks to minimize the variance of data points within the same class and maximize the variance between classes. As k-means, jenks takes the output number of classes m as input and then calculates for each possible partition of the data points into m classes ,the sum of squared deviations from the class means . Then chooses the split with the lower value .As this process is very data intensive ,it is best used when we have small datasets and a low number of output classes [37].

3.4 Deep Learning

Deep learning can be defined as the process of transforming raw input data into successively more abstract—and ultimately meaningful—representations using neural networks. Mathematically, neural networks are based in the *Universal Approximation Theorem*, which states that a feedforward network with a single hidden layer of sufficient width can approximate any continuous function on a compact domain. Deep learning is a subset of machine learning that builds and trains neural networks—models loosely inspired by the layered, interconnected structure of neurons in the brain. In recent years, deep learning has come to dominate both research and applications in machine learning and artificial intelligence, achieving state-of-the-art results across fields such as computer vision, natural language processing, and robotics.

3.4.1 Perceptron

A perceptron [71]is a simple algorithm for binary classification. Given an input vector $x = (x_1, x_2, \ldots, x_D)$, each feature x_j is multiplied by a learnable weight w_j, and the weighted sum is passed through a Heaviside step (threshold) function:

$$y = \begin{cases} 1, & \text{if } \sum_{j=1}^{D} w_j\, x_j \;+\; b \;\geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

where b is a bias term. The output $y \in \{0, 1\}$ is the predicted class label. By replacing the step function with a linear activation, a perceptron can be generalized to produce a continuous scalar output for regression.

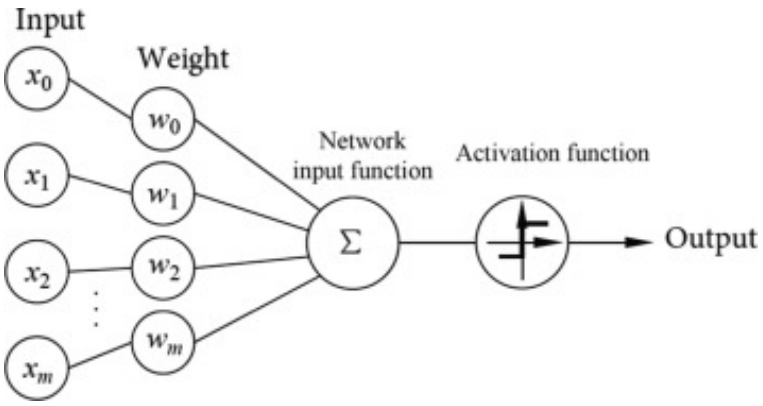


Figure 3.4.1: A simple perceptron model .

3.4.2 Multilayer Perceptron (MLP))

A multilayer perceptron [72] is a feedforward neural network that stacks multiple layers of perceptron-like units, both in width (multiple neurons per layer) and depth (multiple layers). Concretely, an MLP with L layers transforms an input x via:

$$\begin{aligned} h^{(1)} &= \sigma\big(W^{(1)}x + b^{(1)}\big), \\ h^{(2)} &= \sigma\big(W^{(2)}h^{(1)} + b^{(2)}\big), \\ &\vdots \\ h^{(L)} &= \sigma\big(W^{(L)}h^{(L-1)} + b^{(L)}\big), \end{aligned}$$

where each $W^{(\ell)}$ and $b^{(\ell)}$ are learnable weights and biases, and $\sigma(\cdot)$ is a nonlinear activation (e.g., ReLU, sigmoid, or tanh). The final layer's output $h^{(L)}$ can be passed to a softmax for classification or used directly in regression. By combining multiple layers, an MLP can learn complex, non-linear functions of its inputs.

The real strength of neural networks lies in their ability to learn complex, non-linear mappings. This capability arises because the activation functions between layers are non-linear. Furthermore, both the depth (number of layers) and the width (number of neurons per layer) of a network contribute to its expressive power, enabling it to approximate highly intricate functions. Some of the most popular non-linear activation functions are:

- **ReLU (Rectified Linear Unit):**
$$\mathrm{ReLU}(x) = \max(0,\, x).$$
It outputs zero for negative inputs and is linear for positive inputs. ReLU encourages sparse activations and is easy to compute, which helps deep networks train faster [62].

- **Sigmoid:**
$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$
This "logistic" function squashes any real input into $(0, 1)$. It was historically popular for binary classification and provides smooth gradients, but can suffer from vanishing gradients when $|x|$ is large [8].

- **Tanh (Hyperbolic Tangent):**
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$
Tanh squashes inputs into $(-1, 1)$ and is zero-centered, which often helps optimization converge more quickly than sigmoid.

- **GELU (Gaussian Error Linear Unit):**
$$\mathrm{GELU}(x) = x \;\cdot\; \Phi(x) \quad \text{where} \quad \Phi(x) = \tfrac{1}{2}\big[\,1 + \mathrm{erf}\big(\tfrac{x}{\sqrt{2}}\big)\big].$$
Equivalently (approximate): $x \cdot \sigma(1.702\,x)$. GELU weights inputs by their probability under a standard normal distribution, letting values pass through in a "soft," probabilistic manner rather than a hard cutoff. This often yields smoother gradients and slight performance gains in transformer architectures [32].

- **SiLU (Sigmoid Linear Unit, also called "Swish"):**
$$\mathrm{SiLU}(x) = x \;\cdot\; \sigma(x).$$
SiLU multiplies the input by its sigmoid. Unlike ReLU's abrupt zeroing, SiLU smoothly "gates" negative values, retaining a small nonzero slope. This smoothness can improve performance in deep networks, since gradients never fully vanish for $x < 0$ [68].

3.4.3 Convolutional Neural Networks (CNNs)

Convolutional neural networks are a class of deep learning models most commonly used for image processing, though they have also been applied successfully to video classification, speech recognition etc. For many years, CNNs were the de facto standard in computer vision before transformer-based architectures emerged.

A typical CNN consists of three main types of layers:

1. **Convolutional Layers:**
Each convolutional layer applies a set of learnable kernels (filters) to its input feature maps. A kernel is a small 2D matrix of weights that "slides" over the image (or previous feature map), computing a dot product at every spatial location. Intuitively, each kernel learns to detect a specific local pattern—edges, textures, or more complex shapes. Historically, fixed filters like Sobel operators computed image gradients; in a CNN, these kernels are learnable, allowing the network to discover intricate, data-specific features. After each convolution, a non-linear activation (commonly ReLU, $\max(0,\, x)$) is applied so the network can model complex, non-linear relationships.

2. **Pooling Layers:**
Pooling layers perform downsampling to reduce the spatial dimensions (height and width) of feature maps. By summarizing small neighborhoods (e.g., 2×2 windows), pooling layers increase translation invariance, reduce the number of parameters, and mitigate overfitting. Common choices include max-pooling (taking the maximum value in each window) and average-pooling (taking the average). Convolutional and pooling layers are typically interleaved several times, enabling the network to learn hierarchical features—from low-level edges in early layers to high-level object parts in deeper layers.

3. **Fully Connected (Dense) Layers:**
   After multiple rounds of convolution and pooling, the remaining feature maps are flattened into a one-dimensional vector. This vector passes through one or more fully connected layers to produce the final output—often a softmax-normalized vector for classification or a continuous embedding for regression or retrieval tasks.

By stacking convolutional and pooling layers [46], a CNN can learn increasingly abstract representations: early layers capture local edge features, middle layers capture textures or shapes, and deeper layers capture object-level concepts. The non-linear activations (e.g., ReLU) after each convolution are essential for modeling the complex patterns present in images.

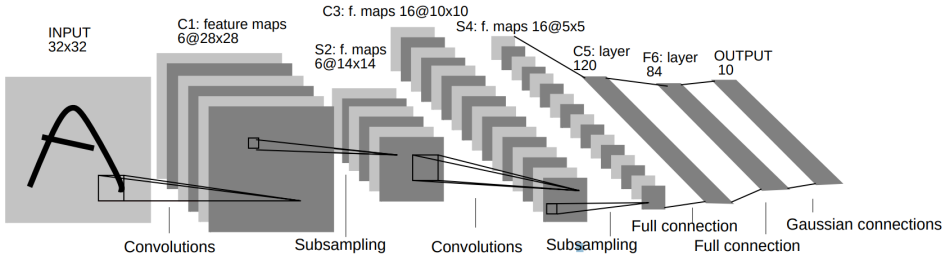


Figure 3.4.2: Architecture of one of the first cnn proposed, LeNet.[49]

3.4.4 Transformers

In 2017, a research team at Google (Vaswani *et al.*) published the now-influential paper *"Attention Is All You Need"*[83], which introduced the first Transformer model.

The main characteristic of the Transformer is its use of the attention mechanism to identify the most important parts of a sequence. Transformers replaced recurrent architectures such as LSTMs for sequential data. Whereas RNNs process one token at a time—carrying hidden states forward—Transformers are fed the entire input sequence (e.g., up to 512 tokens) in parallel. The model then learns to "attend" to the most relevant tokens for tasks like next-word prediction, translation, or other NLP applications.

Below we describe the basic Transformer architecture as introduced in that original paper, without diving into every detail, but still capturing the essential ideas.

Input Preparation

Tokenization and Embeddings First, we tokenize our inputs (e.g., splitting a sentence into words or subword tokens). Each token is then mapped to a learned embedding vector of size d_{model}. If our vocabulary has size V, we have an embedding matrix

$$E \in \mathbb{R}^{V \times d_{\text{model}}}.$$

Positional Encodings Because Transformers see all tokens at once, they need a way to know each token's position in the sequence. We add a positional encoding—often a sine/cosine–based vector—to each token embedding. Intuitively, this tells the model "which slot" each token occupies, so "word 3" doesn't get confused with "word 7."

Encoder

The encoder stack is fed the entire source sequence (token embeddings + positional encodings) in parallel. Each encoder layer (repeated N times) contains:

1. **Multi-Head Self-Attention**
   In this sublayer, every token's embedding is linearly projected into three vectors: a *query*, a *key*, and a

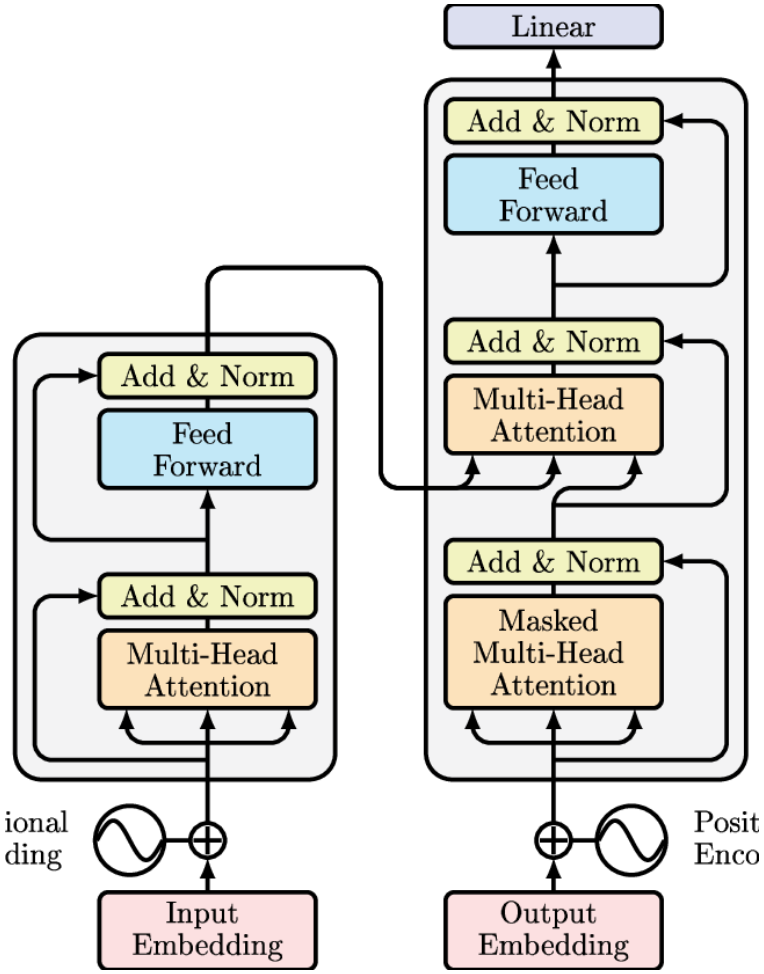


Figure 3.4.3: The Transformer architecture.[83]

value. We split those projections into h "heads," each of dimension d_k. Within each head, we compute

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\Big(\tfrac{Q\,K^{\intercal}}{\sqrt{d_k}}\Big)\,V.$$

Intuitively, each query vector (for one token) is compared (via dot product) to all key vectors (one for every token), yielding a "relevance score" for each token-to-token pair. Dividing by $\sqrt{d_k}$ stabilizes gradients, and applying softmax turns those scores into attention weights that sum to 1 across the sequence. Finally, those weights multiply the corresponding value vectors to produce a weighted combination, highlighting the tokens most relevant to the current token. After doing this in all h heads, we concatenate their outputs and project back to dimension d_{model}. At a high level, self-attention lets each encoder token gather information from every other token in one shot.

2. **Add & Normalize**
   We add a residual (skip) connection from the input of the self-attention sublayer to its output and then apply layer normalization. This "Add & Norm" step stabilizes training and allows very deep stacks of layers.

3. **Feedforward Network**
   Next, each token's attended output passes through a small, position-wise MLP (a fully connected network applied independently to each position). In practice, this MLP has two linear layers with a ReLU (or GELU) in between. This feedforward block further transforms each token's representation. Intuitively, the feedforward layer adds a nonlinear "reshaping" of each token's vector after attention has collected context.

4. **Add & Normalize**
   Finally, we add another residual connection (from the input of the feedforward sublayer to its output) and apply layer normalization again. That completes one encoder layer. Repeating these four substeps N times yields the encoder's final outputs—one vector per source token, each richly informed by the entire input.

Decoder

The decoder also consists of N identical layers, each containing three main substeps, with residual connections and layer normalization after each:

1. **Masked Self-Attention:**
   The decoder receives the target sequence shifted one position to the right (so that, when predicting token i, it only "sees" tokens 1 through $i-1$). Positional encodings are added just as in the encoder. In this masked self-attention sub-layer, each token's embedding attends to all earlier tokens in the decoder—but cannot attend to any future positions. Concretely, we compute the same scaled dot-product attention

   $$\text{Attention}(Q, K, V) = \text{softmax}\Big(\tfrac{Q\,K^{\intercal}}{\sqrt{d_k}}\Big)\,V,$$

   but apply a triangular "look-ahead" mask so that position i has zero attention weight for positions $> i$. This prevents the decoder from "cheating" by looking at tokens it should not yet know.

2. **Encoder–Decoder Multi-Head Attention:**
   Next, each decoder position re-focuses on the source sentence by treating its masked-self-attention output as the "queries," while the encoder's final output vectors serve as "keys" and "values." In other words, each token in the decoder asks, "Given what I have produced so far, which tokens in the encoder's output are most relevant right now?" This cross-attention again uses the familiar scaled dot-product formula, but now all source positions are "known," so no masking is needed. The result is a new representation for each decoder token that blends its own prefix information with encoded source context.

3. **Feedforward Network:**
   Finally, each token's vector from the cross-attention step passes through a small position-wise MLP (two linear layers with a ReLU or GELU in between). This feedforward block further transforms each

vector independently, injecting nonlinearity after the model has already combined decoder prefix and encoder context.

Repeating these three sub-layers N times (always before and after each one there is a residual connection and layer normalization respectively) yields the decoder's final hidden vectors—one per target position. A final linear projection and softmax over the vocabulary produce a probability distribution for each next token:

$$\text{Softmax}(\mathbf{o})_j = \frac{\exp(o_j)}{\sum_{k=1}^{V} \exp(o_k)}, \quad j = 1, \ldots, V.$$

During training, these probabilities are compared to the true next tokens via cross-entropy. During inference, tokens are generated one by one: starting from a special ⟨BOS⟩ token, the decoder predicts token 1; that token is appended, and the decoder runs again (with the new prefix) to predict token 2, and so on until ⟨EOS⟩ is produced or a length limit is reached.

Some of the reasons for the complete domination of Transformers for many tasks are their ability to process multiple sequences in parallel—speeding up training and improving generalization—and their support for model parallelism when a single GPU/TPU is insufficient. This high degree of parallelism has driven state-of-the-art performance in NLP and enabled extensions into vision, speech, and beyond.

Vision Transformers

In 2020, a Transformer-based architecture for computer vision—called the Vision Transformer (ViT)—was proposed [20]. ViT's overall design closely follows the original Transformer for language, but replaces "words" with fixed-size image patches.

First, each input image is divided into a grid of non-overlapping square patches of size $P \times P$. Each patch is then flattened into a $P^2 \cdot C$–dimensional vector (where C is the number of color channels) and linearly projected into a d_{model}–dimensional embedding. In addition to these "patch embeddings," ViT prepends a learnable [CLS] token to the sequence, whose final embedding serves as the image representation. As in the original Transformer, we also add a learned positional encoding to each patch (and to [CLS]) so that the model retains information about each patch's location within the image.

These patch +[CLS] embeddings form a sequence of length $N+1$ (where $N = \frac{H \times W}{P^2}$ with H the height of the image and W its width) that is fed into a standard Transformer encoder (multiple layers of multi-head self-attention and feedforward blocks with residual connections and layer normalization). Because self-attention can directly link any two positions in the sequence, ViT naturally captures relationships between distant patches—e.g. it can attend to a patch in the top-left corner and one in the bottom-right simultaneously.

After passing through all Transformer layers, the final hidden state corresponding to the [CLS] token is typically used as a global image embedding for downstream tasks (classification, detection, etc.). Alternatively, one can average-pool or max-pool over all patch embeddings.

ViTs often require more pretraining data than convolutional networks—because they lack the built-in locality and translation equivariance of convolutions—but they also have much greater model capacity and have demonstrated sufficient robustness in many adversarial settings. Well-known ViT variants include ViT-B/16, DeiT (Data-efficient Image Transformers) [82], and Swin Transformer [56] , among others.

comparison between vits and cnns

Although past research suggested that ViTs might be more robust under adversarial attacks, more thorough studies have shown that under the same training parameters—with identical regularization, data augmentations, and adversarial-training samples—CNNs can actually perform better than ViTs, so in this regard the question remains open [5][75]. In terms of robustness to noise, ViTs exhibit lower misclassification errors on a variety of noise types than comparable CNNs, which can be attributed to their attention mechanism allowing them to capture global content in images [65]. Moreover, a crucial metric for vision models is out-of-distribution generalization, where ViTs often generalize better. However, recent studies have highlighted that some of the advantages of ViTs might be due to more recent and better training protocols rather than the architecture itself.

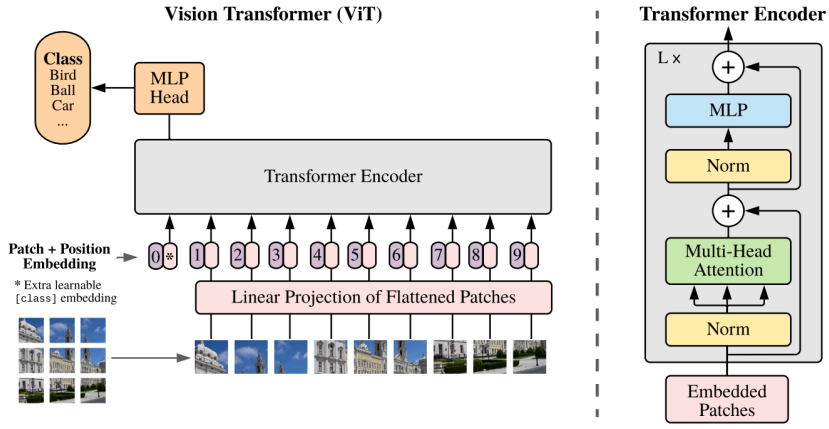


Figure 3.4.4: Model overview of the original ViT architecture: an image is split into fixed-size patches, each linearly embedded with position embeddings, and fed to a Transformer encoder with a learnable classification token .[20]

In terms of computational speed and complexity, ViTs often perform better with fine-grained, high-resolution inputs because their patch-based tokenization does not rely on pooling to down-sample, which can cause CNNs to lose subtle details and struggle to adapt to very large images—although hybrid solutions like HIRI-ViT have been proposed to address this [97] . On the other hand, CNNs' pooling and convolutional layers leverage weight sharing to greatly reduce parameter counts and FLOPs, leading to faster training and often better generalization on smaller datasets

3.4.5 Autoencoders

An autoencoder [33] is a type of neural network most often used in an unsupervised manner to learn lower-dimensional representations of data (in the form of embeddings). It consists of an encoder and a decoder, which are neural networks. If we define the input data as $x \in \mathbb{R}^n$ (an n-dimensional vector), then the encoder learns a function

$$\phi(x) \colon \mathbb{R}^n \to \mathbb{R}^m$$

with $m < n$. The decoder is another network that learns a function

$$g\big(\phi(x)\big) \colon \mathbb{R}^m \to \mathbb{R}^n,$$

mapping the latent representation back to the original dimensionality. Typically, the decoder is trained so that

$$g\big(\phi(x)\big) \approx x$$

through a loss function during training. In this way, the latent representation $\phi(x)$ retains as much information as possible from the original vector x, allowing the decoder's output to closely match the input. Thus, the encoder learns a compressed, lower-dimensional representation of the data that can be used by subsequent machine learning algorithms.

The most common choice for a loss function is the mean squared error (MSE), since reconstruction is a regression task:

$$L_{\text{MSE}} = \frac{1}{M} \sum_{i=1}^{M} \big(x_i - g\big(\phi(\tilde{x})\big)_i\big)^2,$$

which the model minimizes—reaching zero when $x_i = g(\phi(\tilde{x}))_i$. One obvious application of autoencoders is data compression: they achieve nonlinear dimensionality reduction at lower computational cost than PCA. They also excel at denoising (for example, removing noise from images or audio) and underpin many

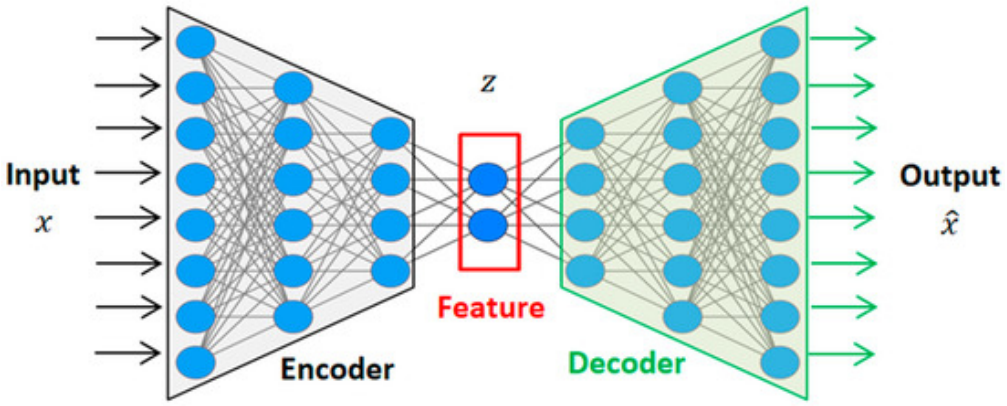


Figure 3.4.5: The architecture of a basic autoencoder [2].

computer-vision pipelines. Moreover, the simple autoencoder architecture forms the backbone of several variants—variational autoencoders [39]-that we will discuss subsequently, denoising autoencoders [86], contrastive autoencoders, and others. The basic architecture of an autoencoder can be seen in figure 3.4.5.

Variational Autoencoders

Although variational autoencoders [39] share many architectural features with simple autoencoders, their objectives and mathematical formulations differ substantially. VAEs are generative models that learn to produce new samples resembling the input distribution, yet they can also be applied to many of the same tasks as simple autoencoders.

Unlike deterministic autoencoders, VAEs encode each input into a probability distribution $q_\phi(z \mid x)$ over latent variables rather than a single point. Specifically, they assume this distribution is Gaussian, so the encoder outputs only the mean $\mu(x)$ and standard deviation $\sigma(x)$, which fully parameterize

$$\mathcal{N}\big(\mu(x),\, \mathrm{diag}(\sigma(x)^2)\big).$$

During training (and at inference), the encoder maps an input x to a Gaussian distribution

$$q_\phi(z \mid x) = \mathcal{N}\big(\mu(x),\, \mathrm{diag}(\sigma(x)^2)\big),$$

outputting only the vectors $\mu(x)$ and $\sigma(x)$. A latent sample $z \sim q_\phi(z \mid x)$ is then drawn and fed into the decoder, which generates a reconstruction x' that should assign high likelihood to the original x.

The VAE loss is usually written as

$$L_{\mathrm{VAE}} = \frac{1}{N}\sum_{i=1}^{N}\Big[-\mathbb{E}_{z\sim q_\phi(z|x_i)}\big[\log p_\theta(x_i \mid z)\big] \;+\; D_{\mathrm{KL}}\big(q_\phi(z \mid x_i)\,\big\|\,p(z)\big)\Big],$$

where the second term is the Kullback–Leibler divergence, a regularization that pushes the encoder's distribution toward the standard normal prior $p(z) = \mathcal{N}(0, I)$.

3.4.6 CLIP (Contrastive Language–Image Pre-training)

CLIP (Contrastive Language–Image Pre-training) is a vision–language model developed by OpenAI [67], which aims at closing the gap between images and text descriptions. For this task, it is trained on 400 million (or larger for more recent variants) image–to–text pairs available online in a contrastive manner, enabling it to learn a wide range of visual concepts and associate them with their textual descriptions. It does this by learning to project the embeddings produced for both text and images into a common latent space.

Moreover, CLIP has demonstrated an exceptional leap in zero-shot classification, which is not present in most other vision models. That means its embeddings for images and text can be used reliably on datasets not seen during training. This capability arises from the multimodal training scheme—matching images with their captions—and from using internet-sourced images for each training sample. The zero-shot capability is crucial for tasks like image retrieval, where concepts such as relations between objects are hard to classify, and where different datasets may contain different object classes.

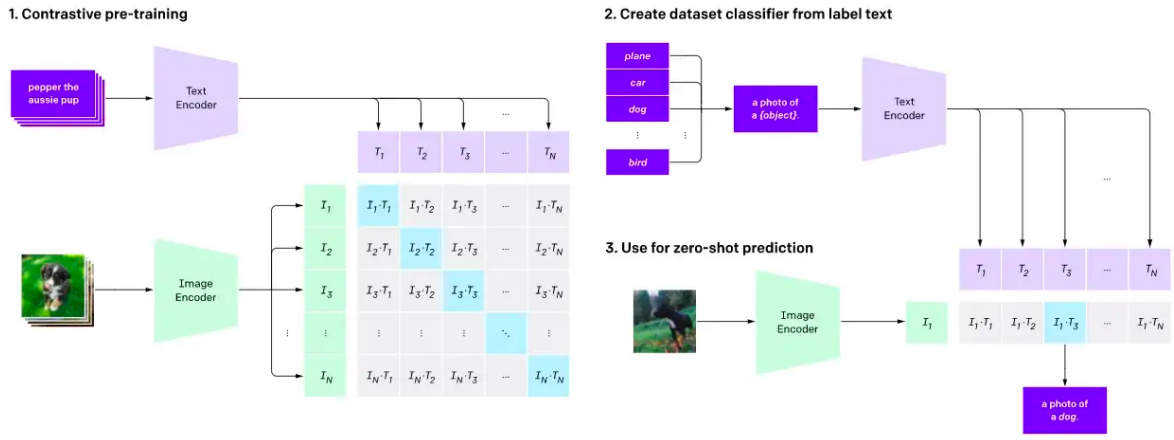


Figure 3.4.6: The training process of clip on the left and inference on the right .[67]

CLIP utilizes transformers to encode both images and text, learning to match textual descriptions with images. The contrastive style of training pushes matched-pair embeddings closer together and unmatched pairs further apart—thereby mapping both text and visual embeddings into a shared space. For example, a picture of "a man walking with a dog" will be mapped closely (by the vision encoder) to the output of that same description (by the text encoder).

For example, **openclip_ViT_H_14_laion2b_s32b_b79k** [7] used later in this thesis uses a ViT-H/14 vision transformer to encode images (i.e., it splits each image into 14×14 patches) and is trained on the LAION-2B dataset [74](≈2 billion image–text pairs scraped from the web). For the text encoder, it uses the same basic CLIP text Transformer–a 12-layer Transformer with 512-dimensional hidden states, 8 attention heads, a 32 000-token BPE vocabulary, and a maximum sequence length of 77 tokens. Both the image and text encoders project into a shared 1 024-dimensional embedding space. Finally, it is trained with the standard contrastive (InfoNCE) loss [64], which pulls matching image–text embeddings together and pushes non-matching pairs apart.

Applications of CLIP's architecture span image retrieval and ranking, visual question answering, zero-shot classification, content moderation, and automatic captioning (text generation). Of these, image retrieval and ranking and visual question answering are especially important for this thesis.

3.5 Training Process

In this section, we will discuss the basic process followed in most machine learning tasks: from the data types used by common models and tasks to the training of the actual networks, as well as inference and evaluation. We present these topics in general, but we will examine in greater detail those aspects most relevant to our work.

3.5.1 Data Types

As discussed above, machine learning aims to find patterns in data. Because data can take many forms, artificial intelligence models have expanded into diverse areas of computer science, robotics, and more. Here, we briefly discuss some of the most relevant data types and the models commonly used to handle each.

- **Text Data:** Natural language processing (NLP) was one of the first fields where deep learning achieved exceptional success. Deep architectures—most prominently Transformers—are now capable of processing raw text, learning contextual representations, and understanding linguistic concepts. Large language models (LLMs) leverage vast amounts of text available on the internet to train effectively.
- **Image Data:** Computer vision is now completely dominated by deep learning models. Convolutional Neural Networks (CNNs) made a breakthrough in image classification and feature extraction, and more recently Vision Transformers (ViTs) [20] have achieved state-of-the-art performance on many tasks. These architectures can classify objects, segment scenes, and extract high-level features with unprecedented precision.
- **Tabular Data:** Tabular (or structured) data appear in spreadsheets and relational databases, where each row represents an instance and each column corresponds to a feature. Traditional models such as decision trees, random forests and Multilayer Perceptrons (MLPs) are widely used here. More recent work has also adapted Transformer-based architectures and specialized neural networks to handle tabular inputs, often outperforming classical methods [36].
- **Time Series Data:** Time series data consist of sequences of values indexed by time (e.g., sensor readings in robotics, financial market prices, or weather measurements). Common models include Recurrent Neural Networks (RNNs) [22], Long Short-Term Memory networks (LSTMs) [34], Gated Recurrent Units (GRUs)[15]. These architectures capture temporal dependencies and trends, making them well suited for forecasting, anomaly detection, and control tasks.
- **Graph Data:** Graph-structured data encode relationships between entities, such as social networks, molecular structures, or knowledge graphs. Graph Neural Networks (GNNs) are specifically designed to operate on graphs by aggregating information from each node's neighbors to learn expressive representations. Graph data and GNNs are most relevant to our work but also find applications in recommendation systems, biology, chemistry, transportation, and many other fields.

3.5.2 Training

Neural networks consist of interconnected layers of perceptrons. Each perceptron's output depends on learnable weights assigned to its incoming connections. During training, we adjust these weights so that the network's outputs approximate our desired values. However, when a network contains many weights and its nodes influence each other, a key question arises: how do we find weight values that yield sensible predictions for our task (i.e., so that the model's outputs match the true target values)? Firstly, we need to discuss loss functions. Loss functions are used to quantify the difference (often through an arithmetic measure) between our model's predictions and the expected ground-truth values of our dataset. Our model is trained to minimize this loss function. Of course, depending on the task at hand, different loss functions may be better suited for a problem. Some of the most common choices are:

1. **Mean Squared Error (MSE)**
   Arguably the most popular loss function for regression tasks is the Mean Squared Error (MSE). This is defined as the average of the squared differences between the target and the prediction of a model:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

   where N is the number of samples, y_i is the true value, and $\hat{y}_i$ is the predicted value. MSE is usually used in regression tasks where we want to predict scalar values. The squaring in the terms makes this function sensitive to outliers, which can be either a benefit or a drawback depending on the task.

2. **Mean Absolute Error (MAE)**
   The Mean Absolute Error calculates the average absolute difference between the predicted values and the actual target values:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|.$$

Unlike MSE, we do not square the terms, so this function does not heavily penalize outliers. As a result, MAE is more robust to outliers and is suitable for tasks where large differences are expected but should not dominate the gradient updates.

3. **Huber Loss (Smooth Mean Absolute Error)**
Huber loss combines the benefits of Mean Squared Error (MSE) and Mean Absolute Error (MAE). It uses a quadratic term when $|f(x) - y| \leq \delta$ (like MSE), making it sensitive to small errors, and a linear term when $|f(x) - y| > \delta$ (like MAE), making it robust to outliers.

$$L(\delta, y, f(x)) = \begin{cases} \frac{1}{2}\,(f(x) - y)^2, & \text{if } |f(x) - y| \leq \delta, \\ \delta\,|f(x) - y| - \frac{1}{2}\,\delta^2, & \text{if } |f(x) - y| > \delta, \end{cases}$$

where δ is the threshold between quadratic and linear behavior, y the true value, and $f(x)$ the prediction.

4. **Binary Cross-Entropy Loss**
Used in binary classification, Binary Cross-Entropy (BCE) measures the difference between the true labels $y_i \in \{0, 1\}$ and predicted probabilities p_i. It's defined as

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^{N} \big[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \big],$$

where N is the number of samples. The logarithm strongly penalizes wrong predictions with high probability in the other class.

5. **Hinge Loss**
Most frequently used in support vector machines, hinge loss is a loss function designed to maximize the margin between two classes. Specifically for a classifier with true label $t \in \{-1, +1\}$ and predicted score y, the loss is

$$\ell(y, t) = \max\big(0,\ 1 - t \cdot y\big),$$

6. **InfoNCE**
InfoNCE (Info Noise Contrastive Estimation) [64] is a contrastive loss used for self-supervised or unsupervised representation learning. For each anchor x, we sample one positive example x^+ (highly similar under the task) and $N - 1$ negatives $\{x_i^-\}$ at random from the dataset. The loss is then

$$L_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{\exp\big(s\big(f(x), f(x^+)\big)\big)}{\exp\big(s\big(f(x), f(x^+)\big)\big) + \sum_{i=1}^{N-1} \exp\big(s\big(f(x), f(x_i^-)\big)\big)} \right].$$

Here $s(f(x), f(x^+))$ measures similarity between the anchor and its positive, and the denominator sums over similarity scores for the positive and all negatives. The logarithm encourages the positive pair's similarity to dominate those of the negatives.

Because, as we have discussed, neural networks can be seen as nonlinear function estimators, the problem of adjusting the weights is equivalent to a nonlinear optimization problem for which gradient descent is a common method. As the name suggests, gradient descent uses the gradient of the loss function with respect to each weight to update those weights. The logic behind this is fairly simple, since the model's output depends on each weight, we should move the weights in a direction that reduces the loss. That direction is given by the negative gradient. Specifically, if w_t is the current weight and L is the loss function, the update rule is

$$w_{t+1} = w_t - \eta\, \frac{\partial L}{\partial w_t},$$

where $\eta > 0$ is the learning rate.

Of course, depending on how frequently the weights are updated, we can have different types of gradient descent(eg if we update the model parameters after evaluating each training point or a group of them):

1. **Stochastic Gradient Descent (SGD)** updates parameters very frequently—after every single training example. Its noisy gradients can sometimes harm convergence, and it can be computationally expensive on large datasets.

2. **Batch Gradient Descent** computes the gradient over the entire training dataset and updates the parameters once per epoch, after all training points have been evaluated.
3. **Mini-Batch Gradient Descent** splits the training dataset into small batches. For each batch, we compute the gradient over that subset and update the parameters. This approach leads to more stable convergence and requires less memory than full batch gradient descent.

We can also use different update functions, which give rise to various optimizers. Here we will descuss briefly some of the most common :

1. **Adagrad (Adaptive Gradient Algorithm):**
   Adagrad [21] is especially useful when we have limited useful data for a problem. It adjusts the learning rate dynamically for each parameter based on its gradient history. By assigning larger learning rates to parameters with infrequent updates, it becomes particularly beneficial on sparse data.

   We firstly accumulate squared gradients:

   $$r_t = r_{t-1} + \left(\frac{\partial L}{\partial w_t}\right)^2.$$

   Then the update rule is

   $$w_{t+1} = w_t - \frac{\eta}{\sqrt{r_t + \epsilon}} \frac{\partial L}{\partial w_t}.$$

   where η is a global learning-rate hyperparameter, ε a small constant to avoid division by zero always and L the value of the loss function .

2. **Adam (Adaptive Moment Estimation):**
   The adaptive moment estimation optimizer [40] is one of the most often used optimizers, due to its ability to work well with large datasets and big models. In Adam, each parameter has its own learning rate, which depends on past gradients of that weight and adapts as more gradients are calculated. A key feature of Adam is momentum, meaning the update term m_t is affected by past gradients—this helps each weight smooth its update "trajectory" by controlling the influence of gradient outliers. At the same time, like Adagrad, Adam adjusts the learning rate based on the square of past gradients—here via a weighted mean.

   $$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2.$$

   Bias-corrected estimates:

   $$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

   Then the update rule is

   $$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.$$

   where η is a global learning-rate hyperparameter, ε a small constant to avoid division by zero always and L the value of the loss function .

Introducing a learning-rate scheduler—which adjusts η over time—can further improve training and generalization by, for example, reducing η as training progresses.

3.5.3 Overfitting

One of the most common problems with large-scale neural network models that have many parameters is overfitting: the model fits the training dataset very well but cannot generalize to unseen data. Overfitting usually occurs when the training set is too small for the task or contains many irrelevant examples. Additionally, the number of training epochs and the choice of optimizer can influence overfitting.

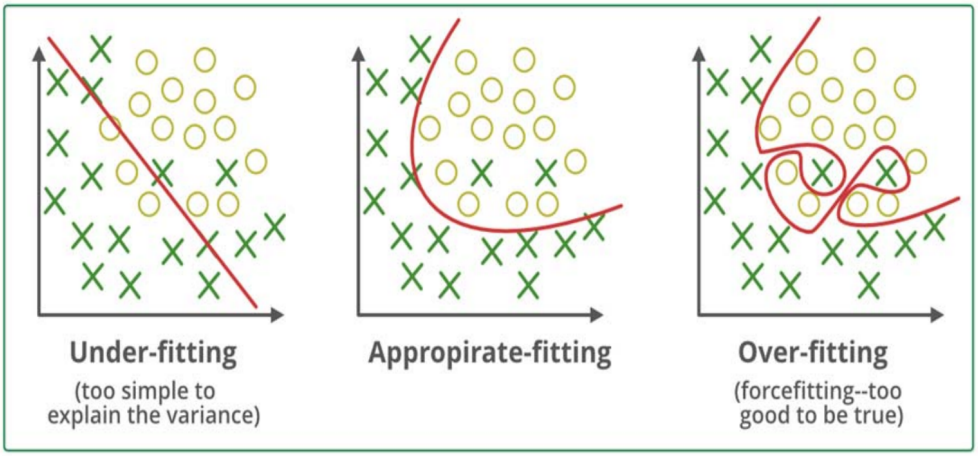


Figure 3.5.1: A simple illustration of underfitting , and overfitting on data.[43]

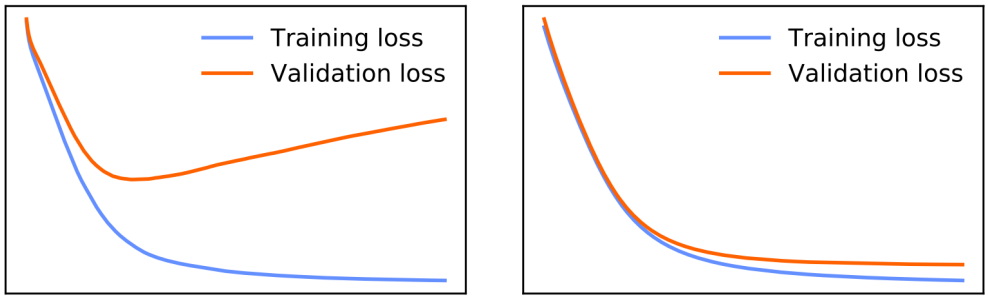


Figure 3.5.2: Overfitted and normal training histories .[53]

3.5.4 Underfitting

Underfitting occurs when the model cannot fit the training data and cannot generalize to new data. This happens when the model's capacity is insufficient to learn the underlying patterns of the task, and we must increase its complexity. Because underfitting is easier to detect and typically straightforward to remedy, we will not discuss it further.

3.5.5 Preventing Overfitting

Several techniques have been proposed to prevent or mitigate overfitting in large neural networks. Here are some of the most commonly used methods:

1. **Decrease network complexity.**
   By reducing the size of the model, we limit its capacity to memorize the training data, forcing it to learn only the most relevant features and thereby generalize better.

2. **Data augmentation.**
   Commonly used in computer vision, data augmentation applies transformations—such as rotation, horizontal or vertical flipping, brightness adjustment, and noise addition—to existing images. This effectively increases the size of the training dataset, encouraging the model to generalize.

3. **L2 regularization.**
   A common symptom of overfitting is excessively large weight values in certain connections. L2 regularization penalizes the sum of squared weights, discouraging large weights and reducing overfitting.

4. **Dropout.**
   During training, dropout randomly ignores a subset of neurons and their connections. This forces the model to produce accurate outputs using different subsets of neurons, which improves generalization [79].

5. **Early stopping.**
   By monitoring training and validation loss curves, we can detect overfitting when the validation loss, after initially decreasing, begins to increase while the training loss continues to decrease. At the point when validation loss starts to rise (i.e., the epoch when overfitting begins), we save the model parameters. These parameters typically generalize better to unseen data.

3.5.6 Model Evaluation

After training, we use appropriate metrics—depending on the task—to analyze our model's performance on unseen data. Below are some of the most common evaluation metrics, including those we use in our work.

Classification metrics

1) **Accuracy** is the ratio of correct predictions to the total number of predictions. Formally,

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

where TP = true positives, TN = true negatives, FP = false positives, and FN = false negatives.

2) **Precision** is the ratio of true positives to the sum of true positives and false positives. It is especially appropriate when false positives are costly. Formally,

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

3) **Recall** is the ratio of true positives to the sum of true positives and false negatives. It is important when false negatives are costly. Formally,

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

4) **F1 Score** is the harmonic mean of precision and recall. It balances the two when we need a single measure. Formally,

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Ranking metrics

1) **NDCG Score** (Normalized Discounted Cumulative Gain) [89] measures ranking quality by comparing the predicted ranking to an ideal ranking. Let $\text{rel}(i)$ be the relevance score of the item ranked at position i and $\text{rel}^*(i)$ be the relevance of the item at position i in the ideal ranking. Then

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{\text{rel}(i)}{\log_2(i+1)}, \qquad \text{IDCG@}k = \sum_{i=1}^{k} \frac{\text{rel}^*(i)}{\log_2(i+1)}, \qquad \text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}.$$

where DCG the Discounted Cumulative Gain , a measure of the ranking quality of a model , and IDCG the ideal discounted cumulative Grain (by the ground truth values of a dataset) .

2) **Mean Reciprocal Rank (MRR)** is one of the most used metrics for evaluating the quality of recommendation and retrieval systems. It measures the position of the first relevant item in the ranked list produced by the prediction algorithm:

$$\text{MRR} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{\text{rank}_n},$$

where N is the number of queries and rank_n is the position of the first relevant item in the results list. MRR values range from 0 to 1, with 1 indicating that the first item retrieved is always relevant.

3) **Mean Average Precision (MAP@k)** MAP considers the number of relevant items retrieved as well as their position on the list .

$$\text{MAP@}K \;=\; \frac{1}{U}\sum_{u=1}^{U}\text{AP@}K_u$$

where, AP@K is computed by summing the precision at each rank i (with $1 \le i \le K$) where a relevant item appears, and then dividing by the number of relevant items in the top-K.

$$\text{AP@}K \;=\; \frac{1}{N}\sum_{k=1}^{K}\text{Precision}(k) \;\times\; \text{rel}(k)$$

Where rel(k) is 1 if the object at position k is relevant and 0 otherwise, and N the number of relevant items retrieved withing the top K predictions . As with MRR it takes values from 0 to 1 , whith 1 indicating the ideal ranking where all relevant items are placed at the top of the retrieved list .

4) **Rank accuracy** Rank accuracy measures how well a predicted ranking matches a ground-truth ranking. Specifically, for two lists of n elements, let p_i be the item at position i in the predicted list and t_i the item at position i in the true list. Then

$$\text{RankAcc} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\big(p_i = t_i\big),$$

where $\mathbf{1}(\cdot)$ is the indicator function (1 if the items match, 0 otherwise).

Regression metrics

1) **Mean Absolute Error (MAE)** computes the average absolute difference between predicted values and true values:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\big|y_i - \hat{y}_i\big|.$$

2) **Mean Squared Error (MSE)** computes the average squared difference between predicted values and true values:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - \hat{y}_i\big)^2.$$

Correlation metrics

1) **Spearman correlation** is used on ranked data to determine how well the relationship between two variables can be described by a monotonic function. Specifically, for two ranked lists, let R_i be the rank of element i in the first list and S_i its rank in the second. Define the rank-difference

$$d_i = R_i - S_i.$$

Spearman's ρ is then

$$\rho_s = 1 - \frac{6\sum_i d_i^2}{n(n^2-1)}.$$

For example, given a list of objects $[a_1, a_2, \dots]$ and two sets of scalar scores $[0.1, 0.2, \dots]$ and $[0.3, 0.4, \dots]$, you rank each list by importance and compute ρ to see how well the second list preserves the ordering of the first.

2) **Distance Correlation** In contrast to Pearson's correlation, distance correlation measures *both* linear and nonlinear associations between two random variables (or vectors). Sometimes two variables have a strong nonlinear relationship—even though Pearson's r is near zero, they are in fact highly dependent. Distance correlation detects *any* kind of dependence [81].

1. Given samples $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$, form the pairwise distance matrices
$$a_{ij} = |x_i - x_j|, \quad b_{ij} = |y_i - y_j|.$$
Intuitively, a_{ij} and b_{ij} capture how far apart each pair of observations is on each variable.

2. Double-center each matrix by subtracting row- and column-means and adding back the grand mean:
$$A_{ij} = a_{ij} - \frac{1}{n}\sum_{k} a_{ik} - \frac{1}{n}\sum_{k} a_{kj} + \frac{1}{n^2}\sum_{k,\ell} a_{k\ell},$$
and similarly for B_{ij}.

3. Compute the *distance covariance* and *distance variances*:
$$\mathrm{dCov}^2(X,Y) = \frac{1}{n^2}\sum_{i,j} A_{ij}\,B_{ij}, \quad \mathrm{dVar}^2(X) = \mathrm{dCov}^2(X,X), \quad \mathrm{dVar}^2(Y) = \mathrm{dCov}^2(Y,Y).$$

4. Finally, the *distance correlation* is
$$\mathrm{dCorr}(X,Y) = \frac{\mathrm{dCov}(X,Y)}{\sqrt{\mathrm{dVar}(X)\,\mathrm{dVar}(Y)}},$$
with the convention that $\mathrm{dCorr} = 0$ if either marginal distance variance is zero.

Chapter 4

Graphs

Graphs are an important data structure in computer science and in many areas of physics, biology, chemistry, etc. With graphs, we can represent objects and structures such as maps (e.g., Google Maps), molecules (e.g., molecular structures like proteins), social networks (e.g., Facebook friend connections), citation networks, the Internet (e.g., Wikipedia, where terms and words link to other terms or URLs), etc. The ability of graphs to represent a wide range of objects whose structure is not a regular grid (i.e., unlike the pixels in an image, which connect only to their immediate neighbors) has made graph theory a heavily researched area by mathematicians, computer scientists, chemists, etc.

4.1 Graph Theory Basics

There are a handful of ways to define graphs in graph theory, so here we define the most common ones that are most relevant to our work.

A common definition of a graph is an ordered pair $G = (V, E)$. (The term "unordered" means that the order of the lists does not matter, since graphs have no inherent order. In other words ,you cannot say "this is the first node" of a graph.This is the key distinction between graphs and other data structures like trees or binary trees.) Here:

- V is the set of vertices or nodes.
- $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$ is the set of edges (links between nodes). Because sets cannot contain the same element multiple times, multiple edges between the same two nodes are not allowed.

Edges can be ordered pairs or unordered pairs. Specifically, edges are pairs like (u, v), where u and v are vertices of the graph ($u, v \in V$). Ordered pairs mean that the order of the pair matters, so (u, v) is a different edge than (v, u). Unordered pairs mean that $\{u, v\}$ and $\{v, u\}$ are considered the same. This distinction corresponds to directed and undirected graphs: directed graphs have ordered pairs as edges, and undirected graphs have unordered pairs.

These definitions apply formally to simple directed graphs and simple undirected graphs. One can extend them to structures where multiple edges between the same nodes are allowed—these are called directed or undirected multigraphs—which we will not discuss further in this thesis.

Extending the above definitions, a graph where each edge is assigned a numerical value (weight) is called a *weighted graph*; otherwise, it is called an *unweighted graph*. Moreover, a graph in which nodes and edges carry arbitrary feature vectors is called an *attributed graph*.

Below are definitions of some fundamental graph-theoretic concepts:

1. **Adjacency.** A node u is called *adjacent* to a node v if there exists an edge between u and v.
2. **Degree.**

- In a directed graph, the *in-degree* of a node is the number of edges arriving at that node; the *out-degree* is the number of edges departing from it.
- In an undirected graph, the *degree* of a node is simply the number of edges incident to that node.

3. **Self-loop.** A *self-loop* is an edge that connects a vertex to itself. (Simple graphs, as defined above, do not include self-loops.)
4. **Path.** A *simple path* in a graph is a sequence of vertices

$$v_1, v_2, \ldots, v_n$$

such that each consecutive pair (v_i, v_{i+1}) is an edge, and no vertex appears more than once. If $v_1 = v_n$, the sequence forms a *cycle* (or *closed path*).

5. **Cycle.** A *cycle* is a path in which the first and last vertices are the same.
6. **Isolated Node.** An *isolated node* is a node with degree 0.
7. **Adjacency Matrix.** The *adjacency matrix* of a graph is a square matrix that represents the set of edges. The rows and columns of this matrix correspond to the vertices, and the entry at row i, column j indicates whether vertices i and j are adjacent. In the simple graphs we discussed, the adjacency matrix contains 0 or 1 to indicate the absence or presence of an edge, respectively.
8. **Spectral Representation.** The adjacency matrix can be diagonalized, and its eigenvalues and eigenvectors give rise to the *spectral representation* of the graph, which we will discuss briefly later.

4.2 Graph Types

1. **Bipartite graphs** are graphs whose vertex set can be partitioned into two subsets, U and V, such that every edge connects a vertex in U with a vertex in V.
2. **Heterogeneous graphs** are graphs whose nodes and/or edges may be of different types (each type can have its own identifier space). A typical example is a social network, where users, posts, and hashtags form distinct node types, and edges encode various interactions.
3. **Scene graphs** represent a visual scene by explicitly encoding objects, their attributes, and the relationships between objects. In many computer-vision tasks we care not only about which objects are present but also about how they interact—for example, whether a person is *holding* or *throwing* a ball. Each object (node) is linked to others through labeled edges such as "holding" or "throwing." This structured representation is useful for visual question answering (VQA), image captioning, image retrieval, and other tasks that bridge vision and language modalities.

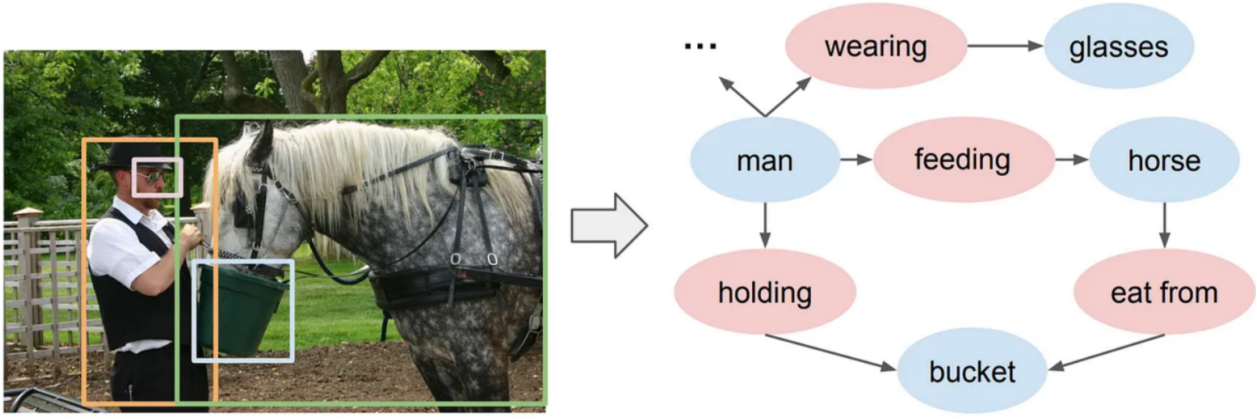


Figure 4.2.1: An example of the scene graph of an image [1].

4.3 Graph Similarity

The ability of graphs to represent many important real-world data structures makes graph comparison crucial for numerous tasks. For example, in social networks, similar neighbor graph structures (or features) for different people may indicate shared interests. Furthermore, similar molecular or protein structures often exhibit similar properties. For these applications, it is important to have consistent metrics to assess graph similarity from multiple perspectives (node features, topology, etc.). Here, we discuss some of the most common graph-similarity techniques in the literature.

2) Algorithmic Comparison / Graph-Edit Distance Since Graph Isomorphism sits in NP but isn't known to be NP-hard (and is widely conjectured to be NP-intermediate), practical systems typically fall back on approximate metrics such as graph-edit distance (GED). GED is the minimum total cost of an edit sequence that transforms one graph into another, where allowed operations are node or edge insertion, deletion, or substitution. Formally,

$$\mathrm{GED}(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\mathcal{P}(g_1, g_2)$ is the set of all edit paths turning g_1 into a graph isomorphic to g_2, and $c(e_i) \geq 0$ is the cost of each edit operation e_i.

Because computing exact GED is NP-hard, most applications rely on polynomial-time approximations. A common approach builds a cost matrix whose rows and columns correspond to nodes; entry (u, v) encodes the cost of matching node v in one graph to node u in the other (including node substitution, plus insertion or deletion of incident edges). An approximate GED is then obtained by solving the resulting assignment problem. The Hungarian algorithm [47] solves this in $O(n^3)$ time, but in practice the Volgenant–Jonker method [38]—with graph preprocessing to rule out unlikely matches and efficient data structures—runs much faster while still $O(n^3)$ in the worst case. Other popular approximations include beam search [63] and A*-based search heuristics.

3) 1-WL test The 1-WL test is an approximate algorithm which tests graphs for graph isomorphism. It is an iterative algorithm which aims at capturing the structure of the graph by refining node labels (or colors). Specifically, at the start each node is given an initial color (label), which could be the same for every node or depend on a characteristic of the nodes (commonly their degree). Then, in each iteration the node labels are updated by hashing the previous label of the node together with the labels of all of its adjacent nodes into a single new node. This process is repeated until no new labels appear or after a fixed number of iterations.

3) Graph kernels Graph kernels are functions that produce outputs which contain information about a wide range of properties of graphs . This graph representations can be used to compare the graph structure , content or more specific properties of them . Formally a kernel maps a graph x via a function φ to a Hilbert space (a space where we can define an inner product <φ(χ), φ(χ')>) then a kernel function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ computes the similarity of the graphs in this new space . One important property of kernel methods is that the comparison between different data points (graphs) does not need the explicit calculation of φ(χ) which can be computationally expensive, but we can do the comparison by calculating only k(φ(χ),φ(χ')) in the input space which can in a lot of cases be more efficient. This is known as the kernel trick . Well–known kernels include:

- **Weisfeiler–Lehman (WL) subtree kernel:**
  Based on the 1-WL test, Shervashidze et al. [77] defined a kernel function that runs the WL-test algorithm and, at each iteration t, records the number of occurrences of each node label in a histogram $\phi_t(G)$. After h iterations, these histograms are concatenated into one feature vector for the graph,

  $$\big[\phi_0(G),\, \phi_1(G),\, \ldots,\, \phi_h(G)\big],$$

  and the kernel between two graphs G and H is computed as the inner product of their concatenated histograms.

- **Random–walk kernel**

  One of the most famous graph kernels is the random–walk kernel [26], which counts the number of matching walks between two graphs G and G'. By a "walk" we mean a sequence of nodes $(u_0, u_1, \ldots, u_k)$ where each (u_{i-1}, u_i) is an edge. Directly counting matching walks is expensive, so we first build the product graph $G_\times = G \times G'$. Its nodes are all pairs (u, u') with $u \in V(G)$, $u' \in V(G')$, and we connect (u, u') to (v, v') if and only if $(u, v) \in E(G)$ and $(u', v') \in E(G')$. It can then be shown that the total number of length-k matching walks equals the number of length-k walks in $G_\times$.More formally, for a graph with adjacency matrix $A_\times$, the (i, j)-entry of $A_\times^k$ equals the number of walks of length k from node i to node j. Hence, if $A_\times$ is the adjacency matrix of the product graph and λ is a decay factor, one common definition is:

  $$K(G, G') \;=\; \sum_{k=0}^{\infty} \lambda^k \, \mathbf{1}^\top A_\times^k \, \mathbf{1},$$

  where $\mathbf{1}$ is the all-ones vector.

- **Subgraph–matching kernel:**
  The subgraph–matching kernel [44] counts how many subgraph structures of up to k size are common between two graphs. More concisely, k is usually set small for computational-complexity reasons. For every i between 2 and k, we count, for each possible graph with i nodes, how many occurrences of this graph exist in both G and G'. Then we concatenate these results for each i, and by multiplying the concatenated histograms (the kernel function) we get a pretty expressive measure of structural similarity between the graphs.

  To compute this kernel, one constructs the (weighted) product graph of the two inputs and then performs clique enumeration on it, summing the weights of all cliques to obtain the final kernel value. Specifically, given two graphs G and G', as with the random-walk kernel we first build the product graph $G \times G'$ and assign to each node (u, u') a weight $\kappa_V(u, u')$ (vertex-similarity) and to each edge $((u, u'), (v, v'))$ a weight $\kappa_E((u, v), (u', v'))$. It can then be shown that each clique in this product graph corresponds to a subgraph isomorphism in the originals (since an edge in the product graph means $(u, v) \in E(G)$ and $(u', v') \in E(G')$; a clique thus represents a matched subgraph). The weight of each clique implements the weighted version of the kernel. For each clique (matching) ϕ we compute

  $$K(G, G') = \sum_{\phi \in B(G, G')} \lambda(\phi) \prod_{v \in S} \kappa_V\big(v, \phi(v)\big) \prod_{e \in S \times S} \kappa_E\big(e, \psi(e)\big),$$

  which gives us the weighted formulation of the kernel.

- **Shortest–path kernel:**
  The shortest–path kernel [9] calculates how many shortest paths of each length exist in two graphs and then concatenates those counts for comparison. For the case of labeled graphs, we can simplify by computing the shortest–path graph of G, denoted $S(G) = (V, E_s)$, where V is the same set of nodes as in G and

  $$E_s = \{\, (u, v) \mid \text{there is a path between } u \text{ and } v \text{ in } G \}.$$

  On each edge $(u, v) \in E_s$ we assign a weight $d_G(u, v)$ (for example, the length of the shortest path in G between u and v). Then the base kernel

  $$k_{\text{walk}}^{(1)}\big((u, v),\, (u', v')\big) = \begin{cases} d_G(u, v) + d_{G'}(u', v') & \text{if } (u, v) \in E_s(G) \text{ and } (u', v') \in E_s(G'), \\ 0 & \text{otherwise.} \end{cases}$$

  Finally, the full shortest–path kernel is

  $$K\big(S(G), S(G')\big) = \sum_{e \in E_s(G)} \sum_{e' \in E_s(G')} k_{\text{walk}}^{(1)}(e, e').$$

- **Graphlet kernel:**
  The graphlet kernel [76] is similar to the subgraph–matching kernel, but the subgraphs we count are drawn from a **predefined** set of small "graphlets". Specifically, in the subgraph–matching kernel we

count, for each possible graph with k nodes, its occurrences in the two target graphs. With the graphlet kernel, we fix a pool of simple graphlets (e.g., all connected induced subgraphs up to size k) and count only those occurrences, building feature histograms from that pool.

4) GNN-based comparison Many recent developments in graph comparison utilize deep learning techniques such as GNNs to create both node-level and graph-level representations. These embeddings can be used to compare graphs locally or globally, capturing information about both topology and the individual elements. These techniques can then be trained in a task-specific manner, making them suitable for various applications. For example, in this thesis we will utilize GNNs to produce embeddings describing images for the task of image retrieval.

Chapter 5

Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of artificial neural networks specialized for tasks where the input data is in graph form.

In recent years, the use of these networks has increased, because a wide range of problems and datasets can be represented as graphs. Examples include social networks, which describe the interactions of people and their interests; scene graphs, which describe the relationships among objects in images; and molecular graphs, where nodes represent atoms and edges represent bonds. Tools like `PyTorch Geometric` [24] have helped standardize the representation and processing of these graph structures.

5.1 Applications

GNNs can be used for different types of predictions:

- **Node-level prediction**, where the goal is to predict labels or properties for individual nodes within a graph. Common examples are recommendation systems [98] and social networks, where we learn node representations that capture interests, connections, etc.
- **Edge-level prediction**, where the goal is to predict whether a connection exists between two nodes, or to classify the type of connection. An application is molecular structure modeling, where predicting links between atoms can determine different properties of the molecule or protein [27].
- **Graph-level (global) prediction**, where the entire graph is assigned a single label or property. Used in image retrieval with scene graphs (as described in this thesis) [99], and in any task where a global summary of the graph is needed.

5.2 Permutation Invariance

The need for neural network models specific to graph data arises from the fact that usual neural architectures—such as convolutional neural networks—cannot be applied directly to graphs, which are neither grid-like nor sequential (as in RNNs or transformers).

One might represent a graph by its adjacency matrix, but any operation on that matrix can yield different results depending on the ordering of its nodes. This is problematic because, as defined in the previous chapter, a graph is an unordered pair $G = (V, E)$. In other words, a function that processes the nodes of the graph should not depend on the order of nodes or edges in the adjacency list or node list—instead, the resulting representation for each node must be identical for any permutation of the nodes and adjacency. This property is called permutation invariance.Furthermore, our GNN function should ideally be permutation equivariant, meaning that permuting the input graph's nodes (and its adjacency list) results in the same permutation of the output node representations. More formally, suppose K denotes either the adjacency matrix or an

adjacency list derived from the node set $V = \{v_1, v_2, \dots\}$. Let P be a permutation matrix that reorders the nodes. Then a graph-processing function f should satisfy:

- **Permutation invariance**:
$$f(P\,K\,P^\top) \;=\; f(K).$$

- **Permutation equivariance**:
$$f(P\,K\,P^\top) \;=\; P\,f(K).$$

Here, P is a permutation matrix. Permutation invariance means that f does not depend on the arbitrary ordering of rows and columns in the adjacency matrix, while permutation equivariance means that if we permute the adjacency matrix by P, then the output of f is permuted in the same way. Functions such as summing or averaging node features over all vertices are permutation invariant, while typical GNN message-passing layers (e.g., GraphSAGE [28] or GCN [42]) are permutation equivariant, since they aggregate neighbor information in a way that does not depend on any fixed node ordering.GNNs are designed most often to collect each node's 1-hop neighbors' features, use a permutation-invariant aggregation (e.g., sum), and then compute node updates in a way that remains permutation-equivariant.

5.3 Graph Isomorphism

Although the graphs we will visit in our work contain rich node embeddings and most GNNs will have sufficient capacity to distinguish them and produce rich representations, it is important to mention graph isomorphism as a measure of the distinguishable power of our models and a fundamental task in graph theory. Graph isomorphism determines whether two graphs share exactly the same structure, differing only by a relabeling of nodes. Formally, let $G_1 = (V, E)$ and $G_2 = (V', E')$ be two graphs with adjacency matrices A_1 and A_2 and, if node features are present, feature matrices X_1 and X_2. Graphs G_1 and G_2 are isomorphic if there exists a permutation matrix P such that

$$P\,A_1\,P^{\mathsf{T}} = A_2 \quad \text{and} \quad P\,X_1 = X_2.$$

In other words, by permuting the rows and columns of A_1 (and accordingly reordering the rows of X_1), one recovers exactly A_2 (and X_2).

Deciding whether two arbitrary graphs are isomorphic is believed to belong in a class of problems known as NP–intermediate. While graph isomorphism is not known to be NP–complete, no polynomial-time algorithm is guaranteed to solve all cases. Nevertheless, powerful approximate tests—most notably the Weisfeiler–Lehman (WL) algorithm—effectively distinguish nonisomorphic graphs in many practical situations [91].

5.4 GNN variants

GNNs generally fall into two categories:

- **Spectral-based GNNs**, where the filters for aggregating information are defined in terms of the graph Laplacian's eigenbasis.

- **Spatial-based GNNs**, where aggregation is performed directly on each node's neighborhood in the graph. Spatial-based GNNs are currently more commonly used and are more computationally efficient, since they do not require computing eigenvalues or eigenvectors of large matrices.

In this section, we will discuss some of the most notable GNN variants and how they handle and update graph data differently. We begin with the most general framework—Message Passing Neural Networks (MPNNs)—into which many other GNN architectures can be cast.

5.4.1 MPNNs

Message-Passing Neural Networks (MPNNs) provide a general framework for learning on graph-structured data by iteratively updating node representations through message-passing operations. Originally introduced by Gilmer et al [27] , MPNNs are designed to be permutation-equivariant—an essential property when processing graphs, since nodes have no inherent ordering. In essence, MPNNs perform spatial graph convolutions: just as a CNN kernel aggregates information from neighboring pixels, an MPNN node aggregates information from its neighboring nodes.

At each layer k, every node u has a hidden state $h_u^{(k)}$. The message-passing update from layer k to layer $k+1$ can be written as follows:

$$h_u^{(k+1)} = \mathsf{UPDATE}^{(k)}\Big(h_u^{(k)}, m_{N(u)}^{(k)}\Big), \quad m_{N(u)}^{(k)} = \mathsf{AGGREGATE}^{(k)}\big(\{\, h_v^{(k)} \mid v \in N(u)\}\big),$$

where:

- $N(u)$ is the set of neighbors of node u.
- $\mathsf{AGGREGATE}^{(k)}$ is a permutation-invariant function (for example, a sum or mean followed by a neural network) that combines the hidden states of all neighbors of u into a single message $m_{N(u)}^{(k)}$.
- $\mathsf{UPDATE}^{(k)}$ is another differentiable function (e.g., a neural network) that takes the current hidden state $h_u^{(k)}$ and the aggregated message $m_{N(u)}^{(k)}$, producing the next hidden state $h_u^{(k+1)}$.

Multiple message-passing layers can be stacked so that information propagates across farther reaches of the graph. After K layers, each node's representation $h_u^{(K)}$ encodes not only its own features but also structural information from nodes up to K hops away. These learned node embeddings can be used directly for node-level tasks—such as predicting node labels or link existence—or further aggregated (e.g., summed or averaged) across all nodes to form a global graph representation that summarizes the entire graph's content (commonly used in graph-level classification).

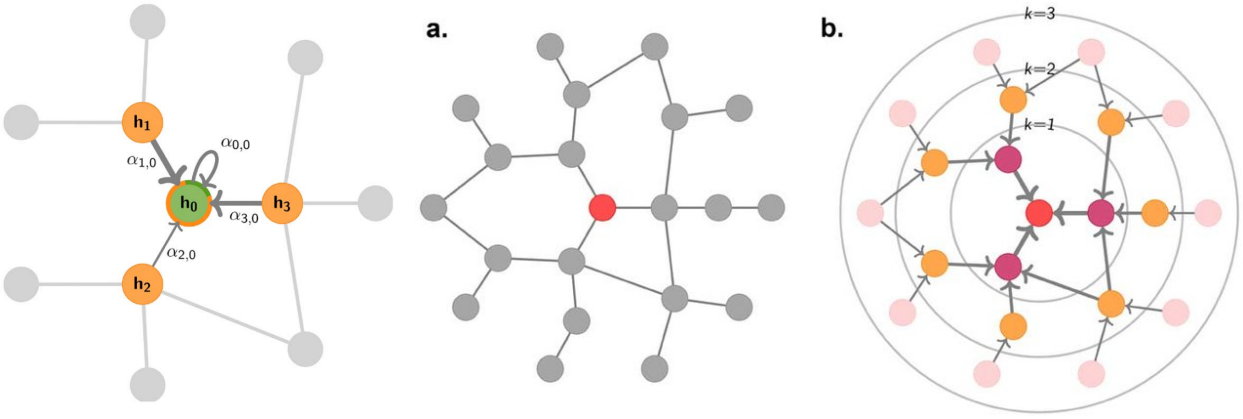


Figure 5.4.1: Aggregation of information from neighbor nodes and an illustration of k-hop messages between nodes.

5.4.2 Graph Convolutional network

A GCN [42] can be viewed as applying *learnable frequency–domain filters* to signals that live on the graph's nodes. The original Graph convolutional network proposed by Kipf & Welling bridges the gap between spectral based and spatial based gnns, showing that GCN which is a first order approximation of localized spectral filters on graphs is equivalent to a message passing formula which is associated with MPNNs . Here we explain the steps of this results which highlight the operations of spectral GCNs as well as the baseline GCN proposed by Kipf and Welling.

- **Graph Fourier basis.** The eigenvectors U of the (normalized) Laplacian $L = I_N - D^{-1/2}AD^{-1/2}$ play the role of "sines and cosines" on the graph. Transforming a signal x into this basis, $U^\top x$ (which define the graph Fourier transform) , tells us *how much* the signal contains every "graph frequency" which is defined by the eigenvalues of the Laplacian λ_i.
- **Spectral filter.** A filter is any diagonal function $g_\theta(\Lambda) = \mathrm{diag}\big(g_\theta(\lambda_1), \dots, g_\theta(\lambda_N)\big)$. Multiplying each Fourier coefficient by $g_\theta(\lambda_i)$ emphasises or suppresses that frequency, giving

$$g_\theta * x \;=\; U\, g_\theta(\Lambda)\, U^\top x.$$

  Different choices of $g_\theta(\Lambda)$ therefore control *how strongly a node mixes information with its neighbours* (low–pass filters perform neighbourhood smoothing, high–pass filters accentuate sharp differences, and so on).
- **Chebyshev trick.(ChebNet [17])** Computing U is impractical as it is computationally expensive to perform this matrices multiplications , so Hammond et al.(2011) [29] approximate $g_\theta(\Lambda)$ by a length-K Chebyshev expansion $\sum_{k=0}^{K} \theta'_k\, T_k(\tilde{\Lambda})$, where $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$. Which gives as $g_\theta * x \approx \sum_{k=0}^{K} \theta'_k\, T_k(\tilde{L})\, x$. Because $T_k(\tilde{L})$ is a k-hop polynomial in L, the resulting filter is K-localized as the Kth-order polynomial in the Laplacian depends only on nodes at maximum K steps away from the target node . Also this order K Chebyshev expansion can be evaluated in $\mathcal{O}(|E|)$ time.
- **First-order and renormalisation.** Setting $K = 1$, λmax to 2 and $\vartheta 0$ and $\vartheta 1$ of the Chebysev polynomials to 1 , further decrease the number of learnable parameters and it is expected that the other parameters of the neural network will learn to adjust to the scale changes caused by this factors gives the simple operator $\big(I_N + D^{-1/2}AD^{-1/2}\big)x$. Kipf & Welling [42] re-normalize it to $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ with $\tilde{A} = A + I_N$ to avoid numerical instabilities, leading to the layer rule

$$H^{(\ell+1)} = \sigma\big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(\ell)}W^{(\ell)}\big).$$

In short, a GCN learns *how much of each graph frequency to keep* and implements that choice with fast, localized polynomial filters, turning the graph's connectivity into a powerful convolutional operator.Furthermore GCN uses a layer-wise propagation rule as other GNN variants of the spatial type ,and it is generally known that in a wide range of applications is capable of encoding both the graph structure and the node features of it .

5.4.3 GIN (Graph Isomorphism Network).

The goal of GIN [95] is to create a GNN with the same distinguishing capability for graphs as the Weisfeiler–Lehman test (1-WL). In GIN, each node's embedding is updated by aggregating its own previous embedding together with the sum of its neighbors' embeddings, and then passing the result through a multi-layer perceptron (MLP).

The update rule for GIN at layer k is then :

$$h_v^{(k)} \;=\; \mathrm{MLP}^{(k)}\Big((1+\varepsilon^{(k)})\, h_v^{(k-1)} \;+\; \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \Big),$$

where $\varepsilon^{(k)}$ is either a learnable scalar or a fixed constant (e.g., zero). By choosing an MLP with sufficiently large hidden dimensions, one can ensure that the mapping from the summed neighborhood embeddings to the new embedding is also injective (unique inputs lead to unique outputs) .

Xu *et al.* [95] showed that, under these conditions, there exist parameters for the MLP and enough hidden units such that this GIN network is provably as powerful as the 1-WL test in terms of discriminative power (i.e., the ability to distinguish non-isomorphic graphs based on their structure).However, in many practical settings—especially when nodes already carry rich, highly discriminative features (e.g., scene-graph embeddings, visual or language features)—that extra "1-WL power" often doesn't translate into better task performance; models like GAT or GCN, with their attention mechanisms or simpler inductive biases, can actually generalize more effectively on real-world data.

5.4.4 Graph Attention Networks (GAT)

Introduced by [84], Graph Attention Networks (GATs), inspired by the self-attention mechanism in transformer models, augment the message-passing process with a learnable attention mechanism that allows each node to weight its neighbours differently. Earlier graph-convolutional models aggregate neighbour features uniformly. GATs instead learn *attention coefficients* that emphasize the most relevant neighbours.

For each node i and neighbour $j \in \mathcal{N}(i)$, GAT computes

$$e_{ij} = \mathrm{LeakyReLU}\big(\mathbf{a}^\top [\mathbf{W}\,\mathbf{h}_i \,\|\, \mathbf{W}\,\mathbf{h}_j]\big), \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)} \exp(e_{ik})}.$$

where e_{ij} is the unnormalized attention score between nodes i and j, computed via a learnable vector $\mathbf{a}$ and weight matrix $\mathbf{W}$. The normalized coefficient α_{ij} determines how much node j contributes when updating node i.

The updated feature for node i is then

$$\widetilde{\mathbf{h}}_i = \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\mathbf{W}\,\mathbf{h}_j\Big),$$

where $\sigma(\cdot)$ is a nonlinear activation (e.g., ELU or ReLU).

A *multi-head* variant improves expressiveness by computing K independent attention heads and then averaging (or concatenating) their outputs. With averaging, for example:

$$\mathbf{h}'_i = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)}\,\mathbf{W}^{(k)}\,\mathbf{h}_j\Big),$$

where each head k has its own parameters $\mathbf{W}^{(k)}$ and $\mathbf{a}^{(k)}$, producing coefficients $\alpha_{ij}^{(k)}$ via the same Equations.

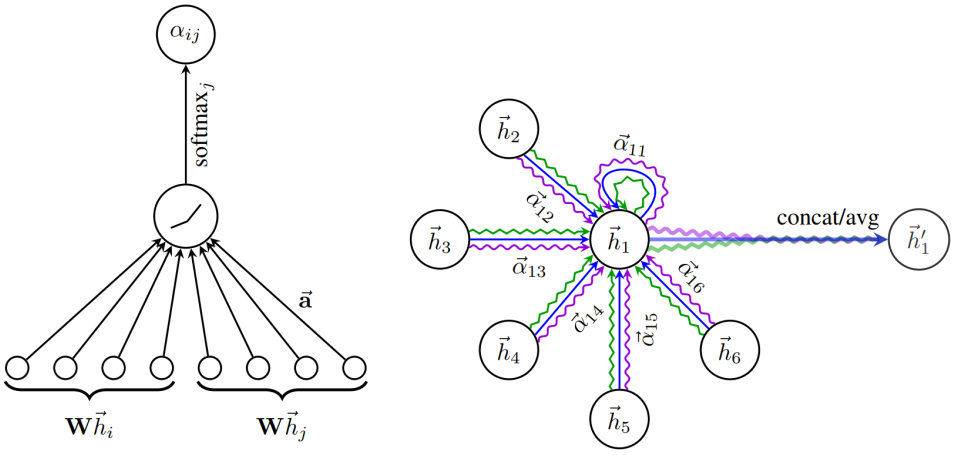


Figure 5.4.2: The single-head attention mechanism and the multi-head extension in the original GAT architecture [84].

5.4.5 GATv2

In a follow-up paper, the GAT authors showed that original GAT layers are limited in expressivity because their attention coefficients do not allow each node to attend differently to its neighbors based on its own features. Concretely, in GAT the unnormalized score can be written as

$$e(h_i, h_j) \;=\; \mathrm{LeakyReLU}\big(\mathbf{a}_1^\top\,\mathbf{W}\,\mathbf{h}_i \;+\; \mathbf{a}_2^\top\,\mathbf{W}\,\mathbf{h}_j\big),$$

where $\mathbf{a} = [\,\mathbf{a}_1 \,\|\, \mathbf{a}_2\,]$. Because the term $\mathbf{a}_2^\top \mathbf{W}\,\mathbf{h}_j$ depends only on j, there is always some node $j_{\max}$ whose $\mathbf{a}_2^\top \mathbf{W}\,\mathbf{h}_{j_{\max}}$ is largest among all $j \in V$. By monotonicity of LeakyReLU and the softmax, $j_{\max}$ attains the highest attention weight for *every* query node i. In other words, GAT's attention ranking is identical (static) for all i.

To fix this, GATv2 [10] reorders the operations so that the ReLU nonlinearity applies *before* the dot product with $\mathbf{a}$. Now the unnormalized attention becomes

$$e(h_i, h_j) \;=\; \mathbf{a}^\top \,\mathrm{LeakyReLU}\big(\mathbf{W}\,[\,\mathbf{h}_i \,\|\, \mathbf{h}_j\,]\big).$$

Because $\mathrm{LeakyReLU}\big(\mathbf{W}[\,\mathbf{h}_i \| \mathbf{h}_j\,]\big)$ jointly depends on both $\mathbf{h}_i$ and $\mathbf{h}_j$ before multiplying by $\mathbf{a}$, this attention is no longer reducible to a single affine map, and thus it can be "dynamic" (different for each query node i).

It should also be mentioned that although the GAT-style attention calculation and the GATv2-style attention is known to work well in many scenarios with graph data. In principle we can use any type of attention like the ones from other deep learning models [4].

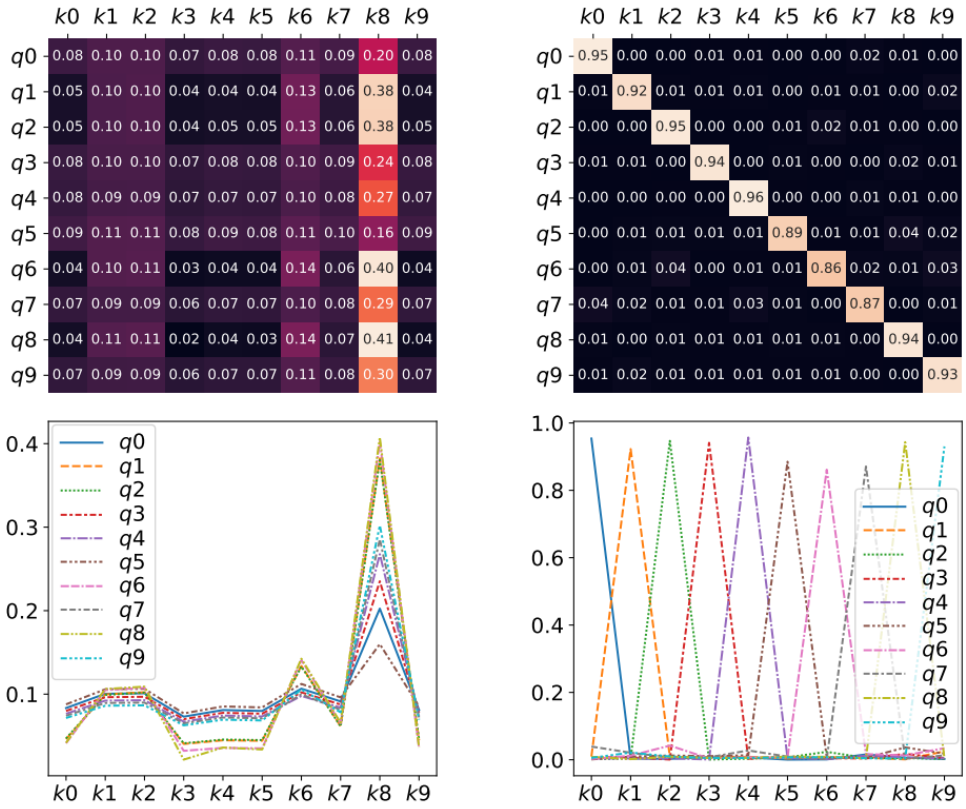


Figure 5.4.3: The original gat produces static attention in contrast to GATv2 which produces dynamic . For example here in the simple gat every nodes attends the most on node 8 , while in GATv2 every node attends to different nodes.[10]

5.4.6 Graph Autoencoders

Extending the idea of autoencoders to graphs, a graph autoencoder not only projects each node into a low-dimensional embedding but also preserves—and can reconstruct—the graph's original structure. Specifically, given a graph $G = (V, E)$ with node set V and adjacency matrix A, a graph autoencoder uses graph neural networks (most often graph convolutional networks) to encode the feature matrix $X \in \mathbb{R}^{N \times d}$ and the adjacency A into a latent representation $Z \in \mathbb{R}^{N \times d'}$ with $d' < d$. The decoder then reconstructs the adjacency via a sigmoid of the inner product of latent vectors [87]:

$$\hat{A}_{ij} = \sigma\big(z_i^\top z_j\big),$$

where σ is the logistic sigmoid. Intuitively, connected nodes yield a high dot product and disconnected nodes yield a low one. In the simplest case, the reconstruction loss is the binary cross-entropy over all pairs:

$$L_{\text{recon}} = -\sum_{i,j}\Big[A_{ij}\log\hat{A}_{ij} + (1-A_{ij})\log\big(1-\hat{A}_{ij}\big)\Big].$$

A **Variational Graph Autoencoder (VGAE)** extends this framework by making the encoder output a probability distribution for each node's embedding—usually a Gaussian parameterized by μ_i and σ_i. During training, one samples

$$z_i \sim \mathcal{N}\big(\mu_i,\, \mathrm{diag}(\sigma_i^2)\big)$$

and applies a KL-divergence penalty to encourage each node's distribution to stay close to a standard normal prior. This probabilistic extension yields a smoother, generative latent space for graph-based data [41].

5.5 Over-smoothing

Over-smoothing is a common problem in message-passing GNNs: after several rounds of aggregation, node representations tend to become nearly identical, making them less discriminative for downstream tasks. This issue grows worse as we stack more layers, since repeated averaging of neighboring features forces distinct embeddings to converge.

While over-smoothing can ruin node representations for many graph tasks, its effects are most intuitively obvious in node classification. As we discussed above, GNN message-passing lets neighboring nodes share information so that nodes at small graph distances end up with similar embeddings. But there is no built-in mechanism to prevent distant nodes from also collapsing together. The receptive field of a GNN is controlled by its depth, and by analogy with CNNs on images—where hundreds of layers deepen the receptive field—one might expect that deeper GNNs should be more expressive and better at capturing global structure. In practice, however, most GNNs are kept very shallow (e.g. 3–5 layers). Graphs differ radically from images in both topology and scale, and beyond a task-dependent "sweet spot" in depth, oversmoothing predominates: node embeddings become so uniform that, for example, a node classifier can no longer distinguish classes and performance degrades.

A great deal of research now focuses on mitigating GNN oversmoothing, but it is important to remember that—under common assumptions about graph structure—oversmoothing is mathematically inevitable as depth grows. Xu et al. (2018) [95] define the *influence* $I_K(u,v)$ of an initial node feature $h_u^{(0)}$ on a final representation $h_v^{(K)}$ by summing all entries of the Jacobian

$$\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}}.$$

In other words,

$$I_K(u,v) \;=\; \mathbf{1}^\top \left|\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}}\right| \mathbf{1},$$

where $\mathbf{1}$ is the all-ones vector. This quantity measures how much the initial embedding of node u contributes to the final embedding of node v. Xu et al. [95] show that for standard GNNs using (self-loop) mean aggregation which can be extended to weighted aggregation,

$$I_K(u,v) \;\propto\; p_{G,K}(v\,|\,u),$$

is the probability of reaching v after K steps of a random walk starting from u. As $K\to\infty$, these random-walk probabilities converge to the stationary distribution, so every node influences every other node roughly equally (the probability of ending in v in the random walk becomes independant of u so they no longer depend on the starting point of the node) . Consequently, node features become indistinguishable—i.e., over-smoothed.

Because deeper GNNs amplify this convergence, stacking many layers can hurt performance by washing out local differences in node features.

There have been several metrics proposed to measure oversmoothing in GNNs, although a unified definition of what such a metric should capture does not yet exist. Rusch et al. [73] propose that a good measure should:

1. detect when node features converge to a constant node vector,
2. capture this convergence in a strict, exponential form, and
3. avoid degenerate choices for similarity functions.

Some of the most common metrics are:

- **Dirichlet energy** , [11],[78] which quantifies the smoothness of node representations over edges:

$$E(X^{(n)}) \;=\; \frac{1}{|V|}\sum_{i\in V}\sum_{j\in\mathcal{N}(i)} \big\|X_i^{(n)} - X_j^{(n)}\big\|_2^2.$$

- **Mean average distance (MAD)** , [73] which measures the average pairwise distance (or similarity) of neighboring embeddings but fails to satisfy all of Rusch et al.'s conditions:

$$\mu(X^{(n)}) \;=\; \frac{1}{|V|}\sum_{i\in V}\sum_{j\in\mathcal{N}(i)} \Big(1 - \frac{(X_i^{(n)})^\top X_j^{(n)}}{\|X_i^{(n)}\|\,\|X_j^{(n)}\|}\Big).$$

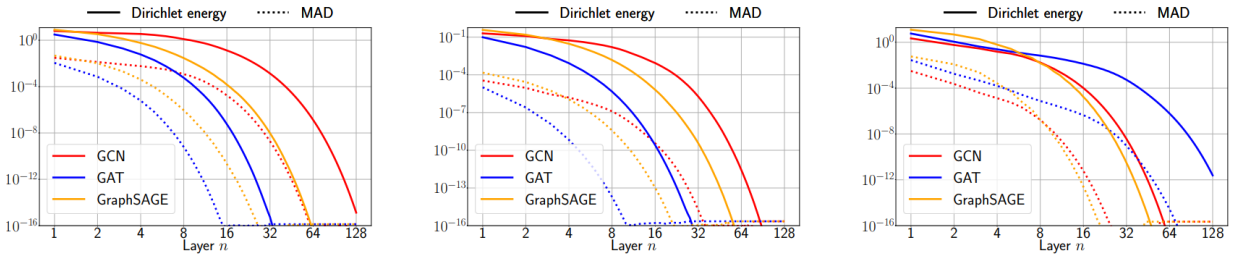


Figure 5.5.1: Dirichlet energy and Mean Average Distance (MAD) of layer-wise node features Xn propagated through a GAT, GCN and GraphSAGE for three different graph datasets, (left) small-scale Texas graph, (middle) medium-scale Cora citation network, (right) large-scale Facebook network (Cornell5).[73]

Several simple techniques help mitigate over-smoothing:

1. **Concatenation.** Instead of replacing the old embedding entirely, GraphSAGE [28] concatenates the previous node representation h_u with the updated message:

$$\mathrm{UPDATE}_{\mathrm{concat}}(h_u, m_{\mathcal{N}(u)}) = \big[\,\mathrm{UPDATE}_{\mathrm{base}}(h_u, m_{\mathcal{N}(u)}) \,\big\|\, h_u\big].$$

   This preserves the original information so that later layers do not fully overwrite earlier signals.

2. **Residual (skip) connections.** One can add a weighted shortcut from the previous embedding:

$$\mathrm{UPDATE}_{\mathrm{res}}(h_u, m_{\mathcal{N}(u)}) = \alpha_1 \circ \mathrm{UPDATE}_{\mathrm{base}}(h_u, m_{\mathcal{N}(u)}) + \alpha_2\, h_u,$$

   where α_1, α_2 are learnable or fixed scalars. This approach, akin to residual blocks in CNNs [31], allows gradients and features to flow more directly from earlier layers. Architectures using this idea have shown significant improvements [52, 14].

3. **Jumping Knowledge connections.** Instead of using only the last layer's embedding, Jumping Knowledge [94] builds the final node representation by combining features from every message-passing layer. For node u with layer-wise embeddings $h_u^{(0)}, h_u^{(1)}, \ldots, h_u^{(K)}$, one sets

$$z_u = f_{\mathrm{JK}}\big(h_u^{(0)} \,\|\, h_u^{(1)} \,\|\, \cdots \,\|\, h_u^{(K)}\big),$$

   where $\|$ denotes concatenation and f_{JK} can be a max-pool, an LSTM aggregator, or a linear projection. By pulling information from all layers, Jumping Knowledge mitigates over-smoothing and preserves both shallow and deep signals.

4. **Regularization and normalization.** A simple yet effective strategy is to penalize oversmoothing metrics (e.g. Dirichlet energy [101]), normalize embeddings (e.g. PairNorm [100]), or inject noise by randomly dropping edges or connections (e.g. DropEdge [70], Graph DropConnect [30]).

In practice, these concatenation and residual strategies often make training deeper GNNs more stable and reduce over-smoothing. They are especially effective on node classification tasks in networks exhibiting homophily—that is, when a node's label is closely tied to its immediate neighborhood.

Chapter 6

Image Retrieval

Image retrieval is the problem of ranking images from a database based on their similarity to a query. Methods can be grouped into three categories—text-based, vision-based (content-based), and scene-graph-based—each relying on different types of queries or features. In this thesis, we focus on scene-graph-based image-to-image retrieval but begin with brief introductions to text-based and vision-based approaches to highlight shared ideas.

6.1 Text-Based Retrieval

Text-based image retrieval involves matching textual descriptions with visual content, aiming to find the most appropriate images in response to a user's request. Various methods, ranging from early keyword-based retrieval techniques to modern deep learning–based models, have been proposed to enhance the accuracy and efficiency of this process. Some of the most recent approaches involve training large vision–language models on pairs of images and textual descriptions, such as CLIP [7] and BLIP [54], which can match images with captions. Furthermore, scene-graph–based retrieval can be viewed as a subcategory of this type because scene graphs encode textual representations of images and their relations, providing a more structured representation than pure captions.

Some notable examples of image-to-text retrieval relevant to our work utilize and analyze scene graphs. Wang et al. [88] construct scene graphs for images and textual graphs for captions, capturing rich semantic information in both modalities, then fuse this information using a hierarchical attention network to compute similarity between textual descriptions and images. Liu et al. [92] construct graphs from captions and images, then use a multi-view fusion network to explore cross-modal relationships between the scene and textual graphs.

6.2 Vision-Based Retrieval (Content-Based Image Retrieval, CBIR)

Content-Based Image Retrieval (CBIR) [66],[85] uses a query image to retrieve similar images by comparing automatically extracted visual features—such as shapes, color, texture, and spatial layout or nowadays visual embeddings. Early CBIR systems relied on hand-crafted descriptors such as color histograms, SIFT or SURF local features [55],[6], and edge/texture filters [50]. These traditional methods were often sensitive to changes in illumination, viewpoint, or background clutter and lacked the ability to capture high-level semantics.

With the advent of deep learning, particularly convolutional neural networks (CNNs), CBIR systems now can easily detect objects from a wide range of categories within images and they can typically extract features from a pre-trained CNN's final layers to produce highly rich global representations for images . These learned features encode object presence, context, and other semantic attributes rather than just low-level pixel patterns or geometric features of the scene (blobs , gradients etc) . Some architectures also leverage raw pixel data alongside CNN features to achieve finer detail. Each image is converted into a compact feature

vector, and similarity between two images (or parts of them) is measured via a distance or similarity function (e.g., cosine similarity or Euclidean distance) between their vectors.

A key objective in modern CBIR is to design models as well as comparison functions that achieve both high inter-class separation (ensuring that images from different categories lie far apart in feature space) and strong intra-class compactness (ensuring that images of the same category cluster closely). This ensures that relevant images are retrieved accurately while irrelevant ones remain distant , a process which we also discussed generally for embeddings. Deep learning techniques—such as triplet loss or contrastive loss—are commonly used to train CNN embeddings that satisfy these constraints. Many content-based methods do not rely on textual annotations (in inference time) ; they use visual features alone, which is advantageous when annotations are missing or unreliable. However, because CBIR focuses on visual similarity, it may fail to distinguish semantically different images that although depict similar semantically things, the low-level appearance characteristics of them are not easily distinguished by computer vision models like CNNs and ViTs (for example a black and white image might have a completely different embedding representation from a colored one even if they show the same things in the scene) .

6.3 Scene-Graph-Based Retrieval

Scene-graph-based retrieval addresses the semantic limitations of CBIR by explicitly modeling objects and their relationships. A scene graph represents an image as a structured graph in which nodes correspond to detected objects (including labels and attributes), and edges encode pairwise relations such as "on top of," "next to," or "holding." Many CBIR systems can detect objects in an image, but in many cases this is not enough for semantic retrieval. Which objects interact and the relations describing these interactions can contain crucial information for improved retrieval. For example, two images might both contain a "person" and a "dog," but the first could depict "a person looking at a dog" while the second shows "a person walking with a dog." This distinction is obviously important for the retrieval process. Moreover, as described above, the advantages of CBIR systems in capturing visual information should not be downplayed; many systems therefore aim to bridge the gap between visual and semantic image retrieval, as we discuss below.

There exist a handful of datasets with human-annotated scene graphs, such as Visual Genome [45] (which features diverse object and relationship categories, although some annotations suffer from poor quality); cleaner subsets like the PSG (Panoptic Scene Graph) dataset [96]; the Open Images dataset [48] (which provides rich object annotations, including segmentation masks, but covers fewer object classes and relationships than Visual Genome); and CLEVR [60] (containing synthetic 3D objects and their relationships—placements and attributes—typically used for relational question answering).

Many automated scene-graph generation and comparison models have been proposed [93],[102]. Models for automatic scene-graph generation typically begin with an object detector (e.g., Mask R-CNN or Faster R-CNN) to identify object instances and their bounding boxes. Once detection is complete, a scene graph is constructed: each node represents an object class (often with attributes), and each edge denotes a relationship between two nodes. After obtaining scene graphs for our images, we must compare them. Approaches range from algorithmic methods such as graph-edit distance (GED) to deep-learning techniques based on graph neural networks (GNNs).

For example, Chaidos et al. [12] utilized an unsupervised retrieval framework using graph autoencoders with GED as a similarity measure between two image graphs for semantically enhanced image-to-image retrieval. Moreover, Dimitriou et al.[19] utilized GNNs to represent images as their graph embeddings, training them so that embedding similarity approximated GED, and used these embeddings to produce counterfactual explanations. Especially important for this thesis is the work of Yoon et al. [99], who used scene graphs processed by GNNs to produce embeddings that match the textual representation of images, giving the model a semantic description of their contents and connections. Also Maheshwari et al.[59] employed graph convolutional networks with a contrastive ranking loss to perform image-to-image retrieval.

Some systems also enrich node features with CNN-derived visual descriptors before the GNN stage. By fusing structural cues from the scene graph with visual cues from the raw image, they preserve both high-level semantics and fine-grained visual features. For example, *Wang et al.* (2023) [90] insert CNN features at every node, then run a GNN to obtain a global embedding and compute local comparisons between nodes,

substantially boosting retrieval accuracy by combining structural relations with visual context.

Chapter 7

Proposal

In this section, we first propose a model trained to predict the important objects and triplets (object 1, relation, object 2) in an image's scene graph. We then introduce a modified GATv2 layer that processes these graphs for the image-to-image retrieval task. We will analytically justify all our design decisions, both empirically and intuitively. We begin by listing the main contributions of this thesis and then thoroughly explain the entire pipeline of our model.

7.1 Contributions

The contributions of this thesis can be summarized as follows:

- We utilize both the scene graphs of images—obtained from the Visual Genome dataset—and visual information for the objects and the image as a whole to perform image retrieval based on query images. Scene graphs provide semantic details about objects and their relations. Since this has been explored before, we introduce new methods to process these graphs and combine textual and visual information for more expressive graph data.
- We use a transformer encoder with an additional multi-head attention module with learned queries to build a model that, given textual and visual object features plus global image information, predicts the importance of objects and relations in a scene graph. We then propose a pruning algorithm that removes the many unimportant nodes present in the annotations of the Panoptic Scene Graph Generation dataset, which, although a cleaner version of the Visual Genome dataset, still contains abundant purely and inconsistently annotated scene graphs.
- We explore various graph neural networks and introduce a GATv2-like layer that incorporates edge embeddings into the attention computation and message passing. Our aim is to generate rich graph-level embeddings for the task of image-to-image retrieval using as much available information for our images-graphs as possible.
- We propose several optimizations to our pipeline and architecture, and we explain ,empirically and intuitively ,their motivations and their actual effectiveness for image-to-image retrieval.

7.2 Proposed Model

Here, we describe the pipeline we propose for image-to-image retrieval and analyze each of its components in detail.

7.2.1 Graph Dataset

Dataset We use images from the intersection of the PSG (panoptic scene graph generation) and MS COCO datasets. From PSG, we obtain the scene graph and bounding-box coordinates for each object in an image.

From MS COCO, we use five captions per image, the similarity between the captions of two images serves as their ground-truth semantic similarity.

Graph construction For the construction of the graphs on which we apply GNNs to produce final embeddings, we use the sentence and vision transformers of the OpenCLIP_ViT_H_14_laion2b_s32b_b79k model to obtain, for each object in the intersected dataset described above, a 1024-dimensional sentence embedding and a 1024-dimensional visual embedding. To the visual embedding, we append five normalized values—the bounding-box coordinates and the object's area in the image—yielding a 1029-dimensional visual vector. We then concatenate this with the sentence embedding to form a 2053-dimensional node embedding. We also use the sentence transformer to obtain 1024-dimensional embeddings for relations—since relations lack bounding boxes, they have only textual embeddings. Furthermore, we obtain a global visual embedding for the whole image using the same visual transformer; we pad this vector with zeros to 2053 dimensions and connect this global node to every other node (we do not have embeddings for this edges so we use embeddings with only zeros). Finally, we split the dataset into train, validation, and test sets (referred to hereafter as the train, val, and test splits), which remain fixed during training, validation, and testing.An illustration of the graph construction process can be seen in figure 7.2.1

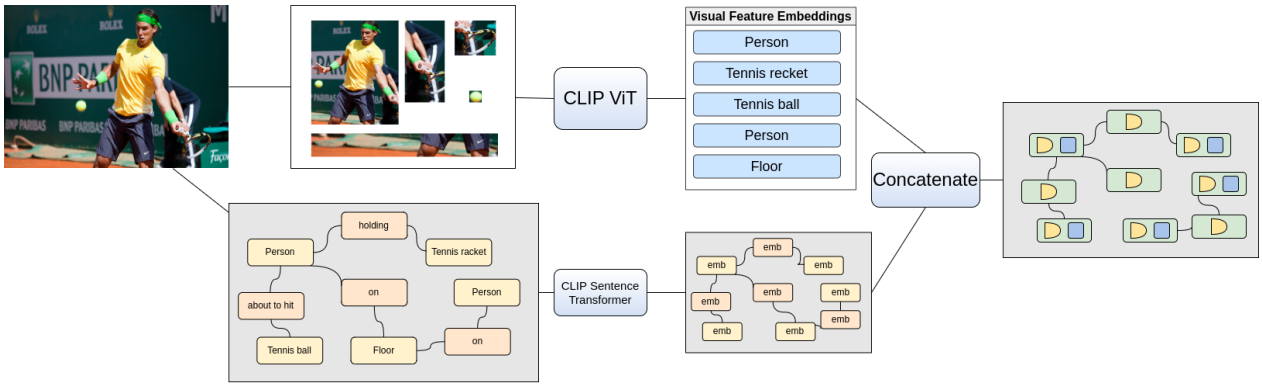


Figure 7.2.1: Object and relation names, together with the graph structure , are taken from the PSG dataset. Visual information for each object is extracted by applying the CLIP visual transformer to its bounding box. Each graph node is the concatenation of its textual and visual embeddings, whereas each edge stores only the textual embedding of its relation.

7.2.2 Importance Prediction Module

Ground-Truth Scoring. As a ground-truth importance score for single objects, we use the average dot product between the sentence embedding of the object label and the sentence embeddings of the five captions for that image. For triplets, we form the phrase "object$_1$ relation object$_2$" using the PSG labels, obtain its sentence embedding, and compute its average dot product with the caption embeddings. For example, if a scene graph contains the objects "person" and "cat" with the relation "looking," and one caption is "A woman looking at two cats," then the importance score for "person" is the dot product of the embedding of "person" with the caption embeddings (averaged over all captions), likewise for "cat," and for the triplet we compute the dot product of the embedding of "person looking cat" with the caption embeddings.

Task Setup. We train an importance prediction module to identify, for each graph, which node-objects and which triplets (object 1, relation, object 2) are most important. The model takes as input: the embedding for object 1 (its concatenated textual and visual embeddings), the textual embedding for the relation, the embedding for object 2, a global textual embedding for the graph (described below), and a global visual embedding for the whole image. To enable both triplet and object importance prediction, we can zero out the relation and object 2 embeddings; in that case, the model predicts only the importance of object 1.

Global Textual Embedding. For the global textual embedding, we first constructed text-only versions of our scene graphs, promoted each edge to a node (forming a bipartite graph), and then trained an unsupervised Infograph algorithm [80] on the train split to produce textual embeddings for each image. For the training

split, we use embeddings generated by Infograph during training; for validation and test, we use the Infograph model's predictions. This global embedding captures the overall structure and object labels of the graph, which intuitively helps our module determine importance.

Architecture. Before inputting them to the importance prediction module, we pass each of the five embeddings through a linear projection layer to project them into a common dimension. We then feed these tokens through a transformer encoder that, via its self-attention mechanism, updates each embedding based on the others—highlighting the most salient features. The transformer encoder thus produces five updated embeddings, one per input token. Next, following Arar et al., we apply a multi-head attention layer with learned queries [3] to derive even more task-specialized embeddings. Finally, we pool the resulting embeddings and pass them through a small MLP to obtain a scalar score for the triplet or object.The overall architecture of our model is visualized in 7.2.2.

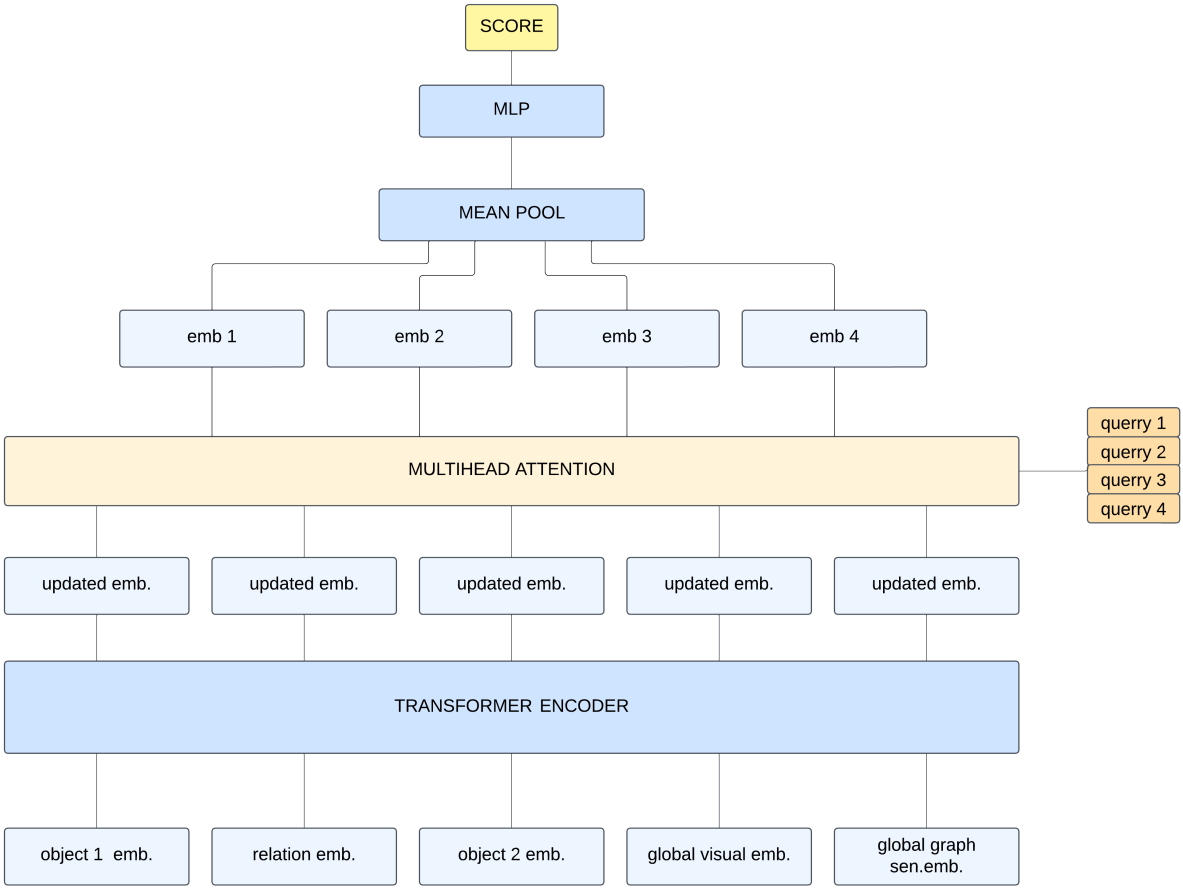


Figure 7.2.2: The architecture of our importance prediction module, the model takes as input 5 embeddings (object1,relation,object2,global visual embedding, global graph sentence embedding) and firstly passes them through an transformer encoder, then the updated embeddings are passed as keys and values through a multihead attention module with learned query embeddings. Finally we pool the resulting embeddings and pass them through a small mlp to produce a final scalar score.

Training. To train the importance prediction module, we construct a dataset from the train split. For each object in each image, we include a sample of the form (object_1 embedding, zero embedding, zero embedding, global visual embedding, global graph textual embedding) with its ground-truth object score. We also include samples for each triplet—(object_1 embedding, relation embedding, object_2 embedding, global visual embedding, global graph textual embedding)—with its ground-truth triplet score. We train the model by minimizing the mean squared error between its predictions and the ground-truth scores.

7.2.3 Graph Neural Network Training

Ground Truth The ground truth similarity between two images is taken as the average similarity between each pair of their captions (because we have 5 captions for each image we have 25 pairs) . Furthermore the ground truth importance between an object or triplet in an image and the image is taken as described in the previous section 7.2.2 .

Graph pruning Before we pass our graphs through a graph neural network to produce a global embedding for them, we perform a filtering of the nodes and edges . Specifically during training for each image in the train split , we use the ground truth importance values for the objects , to form a list of scalar values . We then firstly apply a threshold to this values , saying that every object above this value is important (labeled as 1) . Also we apply the Jenks natural breaks algorithm to this values and we label the objects which where clustered in the cluster with bigger values as important (1) . Finally the important objects are the union of this two methods and every object not included is considered unimportant. Similarly we do the same thing for every triplet in the image , finding which triplets (object 1, relation, object 2) are important.

We then for each graph in the train split mask our the nodes: We keep the nodes that are important (label them as 1 in the mask) . We also keep all the nodes that are part of an important triplet (so if a triplet object 1, relation , object 2 is important we keep object 1 and object 2) . So our final node list consists of the union of this two . Moreover the global visual embedding node is considered important . Finally we keep only the subset of the original edges which connect nodes that are important. An example of the pruning process can been seen in figure 7.2.3, the global visual node as it was labeled as important is always kept as well as its connections with the nodes after pruning (in the example we excluded the global node for optical clarity) .We also tried versions for our model where we kept only the important objects only , not the ones that are part of an important triplet, but the model performed slightly worse in this case.

The intuition behind the Jenks algorithm is that, from observing importance scores across images, we noticed high variance among clearly important objects, so a fixed threshold alone was insufficient. Moreover, because Jenks natural breaks is a distribution-based method, it finds a natural boundary in the scalar scores, although alternatives such as k-means clustering could also be suitable.

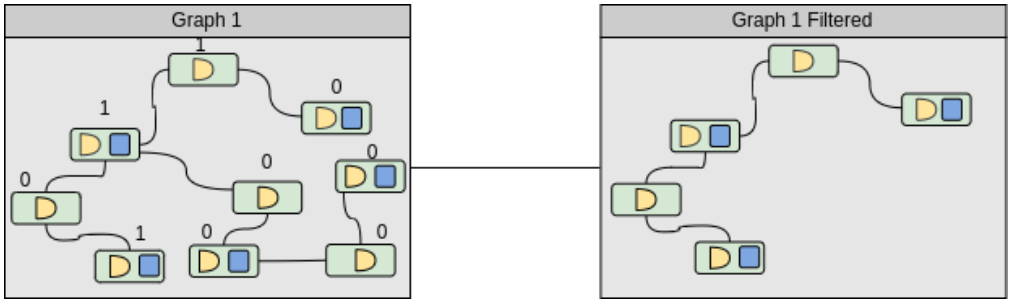


Figure 7.2.3: For pruning our graphs , we keep only the union of nodes that are important (labeled as 1) , and nodes that are part of important triplets (here denoted as 1 in the edge) .

GNN We pass each pruned graph S through our GNN to obtain an embedding $\mathbf{g}(S) \in \mathbb{R}^d$. During training we draw graph pairs (S_i, S_j), measure their similarity with the dot product

$$f(S_i, S_j) = \langle \mathbf{g}(S_i), \mathbf{g}(S_j) \rangle,$$

and ask it to match the average caption similarity

$$s(I_i, I_j) = \frac{1}{25} \sum_{k=1}^{5} \sum_{\ell=1}^{5} \langle \mathbf{c}_{i,k}, \mathbf{c}_{j,\ell} \rangle,$$

where $\mathbf{c}_{i,k}$ is the embedding of the k-th caption of image I_i. We minimize the mean-squared error

$$L_{ij} = \big(f(S_i, S_j) - s(I_i, I_j) \big)^2,$$

thereby training the graph embeddings to mirror the semantic similarity implied by the captions. The full training pipeline is shown in Figure 7.2.4.

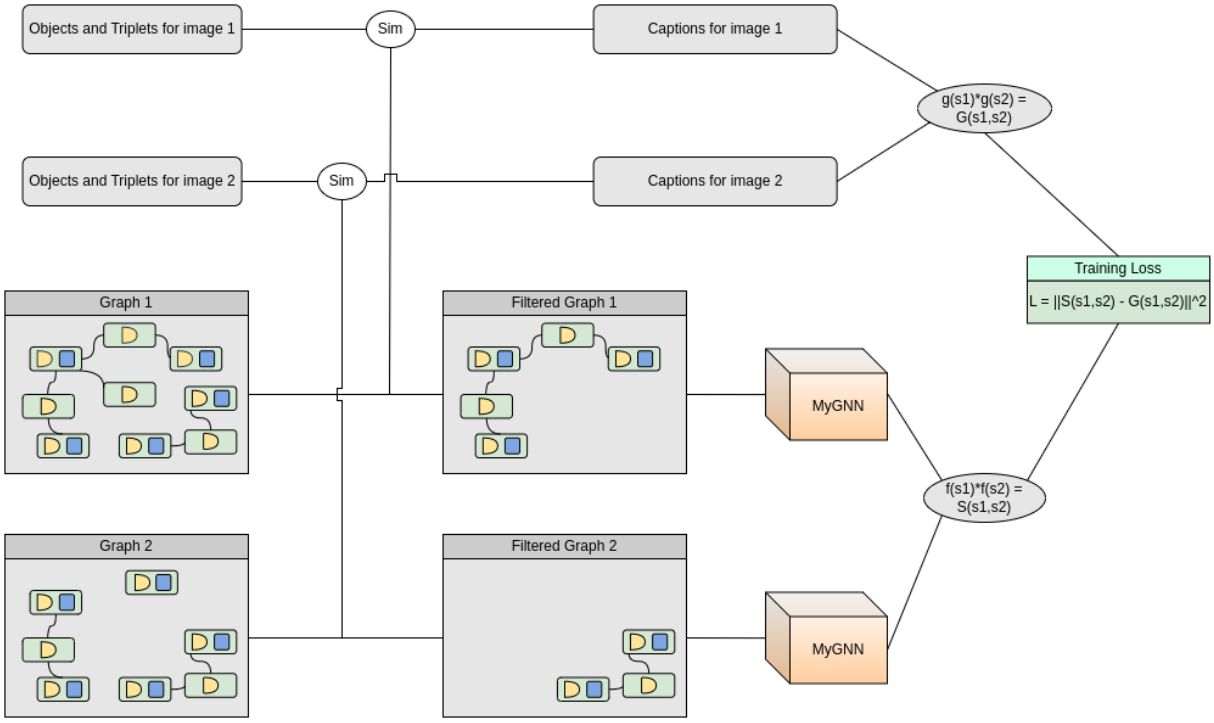


Figure 7.2.4: Our training pipeline . We firstly prune our graphs based on the important objects and triplets in them, then we pass this graphs through a gnn model to produce global embeddings for them . Finally we compare the similarity of two graphs with the ground truth similarity of them obtained by their captions and train with mean squared error loss.

7.2.4 Graph Neural Network Proposal

One main issue with common GNN architectures (e.g., GAT, GIN) is that, during message passing, edge embeddings often go unused—reducing model expressivity by ignoring useful information. In our scene graphs, each edge carries a textual relation (e.g., "person – playing with – dog"), but prior image-to-image retrieval methods either discard these relations entirely [99] or only use them at final matching (e.g., via graph-matching scores) [90], rather than enriching the messages passed between nodes.

We propose a GATv2-style layer that incorporates edge embeddings both in the attention-coefficient computation and in the message updates. Previous attempts to integrate edge information into GAT or GATv2 architectures have used edge embeddings only in the attention score e_{ij}, concatenating them with the node embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$, and then optionally re-injecting edge embeddings in the final GNN layer [13]. In contrast, we argue that the first GAT layer should already integrate edge embeddings, and that subsequent layers should learn node representations informed by both node and edge features. Moreover, we replace concatenation with summation of edge and node embeddings for both message passing and attention calculation. Since our node embeddings combine textual and visual features and our edge embeddings are purely textual, we first split each neighbor's embedding $\mathbf{z}_j$ into $\mathbf{z}_j^{\text{text}}$ and $\mathbf{z}_j^{\text{vis}}$. We then apply separate linear transforms:

$$\begin{aligned}
\mathbf{z}_j^{\text{text}\,\prime} &= W_t\,\mathbf{z}_j^{\text{text}}, \quad & W_t &\in \mathbb{R}^{d_{\text{out}} \times d_{\text{text}}},\\
\mathbf{z}_j^{\text{vis}\,\prime} &= W_v\,\mathbf{z}_j^{\text{vis}}, \quad & W_v &\in \mathbb{R}^{d_{\text{out}} \times d_{\text{vis}}},\\
\mathbf{e}'_{ij} &= W_e\,\mathbf{e}_{ij}, \quad & W_e &\in \mathbb{R}^{d_{\text{out}} \times d_{\text{edge}}}.
\end{aligned}$$

We then reassemble the neighbor's embedding by adding the projected edge to its textual part and concatenating with its visual part:

$$\mathbf{z}'_j = \left[\mathbf{z}_j^{\text{text}\,\prime} + \mathbf{e}'_{ij} \,\middle\|\, \mathbf{z}_j^{\text{vis}\,\prime}\right].$$

The to be updated node $\mathbf{z}_i$ remains $\left[\mathbf{z}_i^{\text{text}} \| \mathbf{z}_i^{\text{vis}}\right]$.

Finally, we apply standard GATv2 attention over these mixed embeddings:

$$\begin{aligned}
e_{ij} &= \mathbf{a}^{\top}\,\text{LeakyReLU}\big(W_{\text{att}}\,[\,\mathbf{z}_i \,\|\, \mathbf{z}'_j\,]\big),\\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\displaystyle\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})},\\
\mathbf{z}'_i &= \sigma\Big(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}\,W_m\,\mathbf{z}'_j\Big).
\end{aligned}$$

In this way, the node embeddings produced by the first layer already encode useful information from both nodes and edges. Because edge embeddings themselves are not updated in that layer, subsequent layers revert to standard GATv2 blocks without additional edge injection. An illustration of the message passing used in this layer can be seen in figure 7.2.5

7.2.5 Inference

During inference, for every image–graph in the test set we first compute a scalar importance score for each object and triplet using the trained importance-prediction module described above. As in training, we label objects and triplets as important by taking the union of (i) a fixed threshold and (ii) Jenks natural breaks. With these predicted importance labels we prune the graph exactly as in training. The filtered graph is passed through the GNN we trained, as discussed above—either our proposed model or alternative backbones evaluated in the experiments section—to obtain a global graph embedding. Finally the predicted similarity between two images (graphs) is the dot product of their embeddings. We illustrate the inference phase in figure 7.2.6

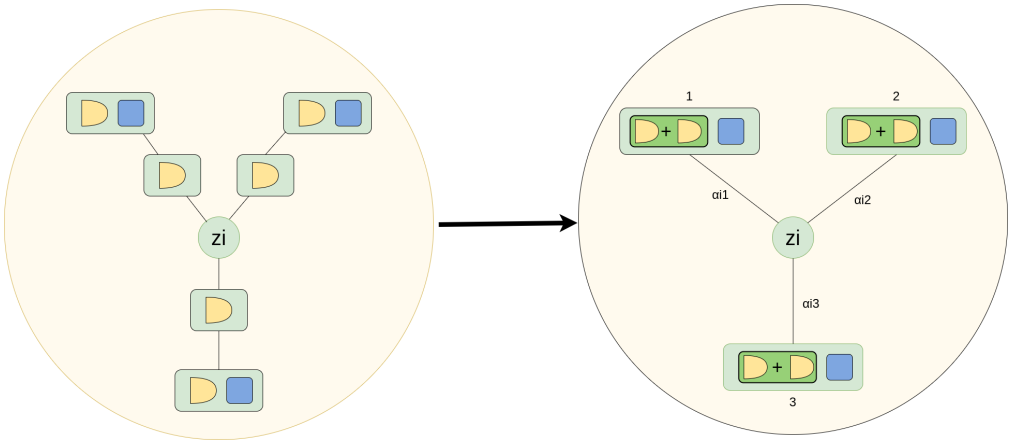


Figure 7.2.5: The message passing we perform in the first layer of our model. We add the textual parts of the node embeddings with the edge embeddings and form new nodes that contain information for both the edges and nodes , then we perform GATv2 attention with this new destination nodes.

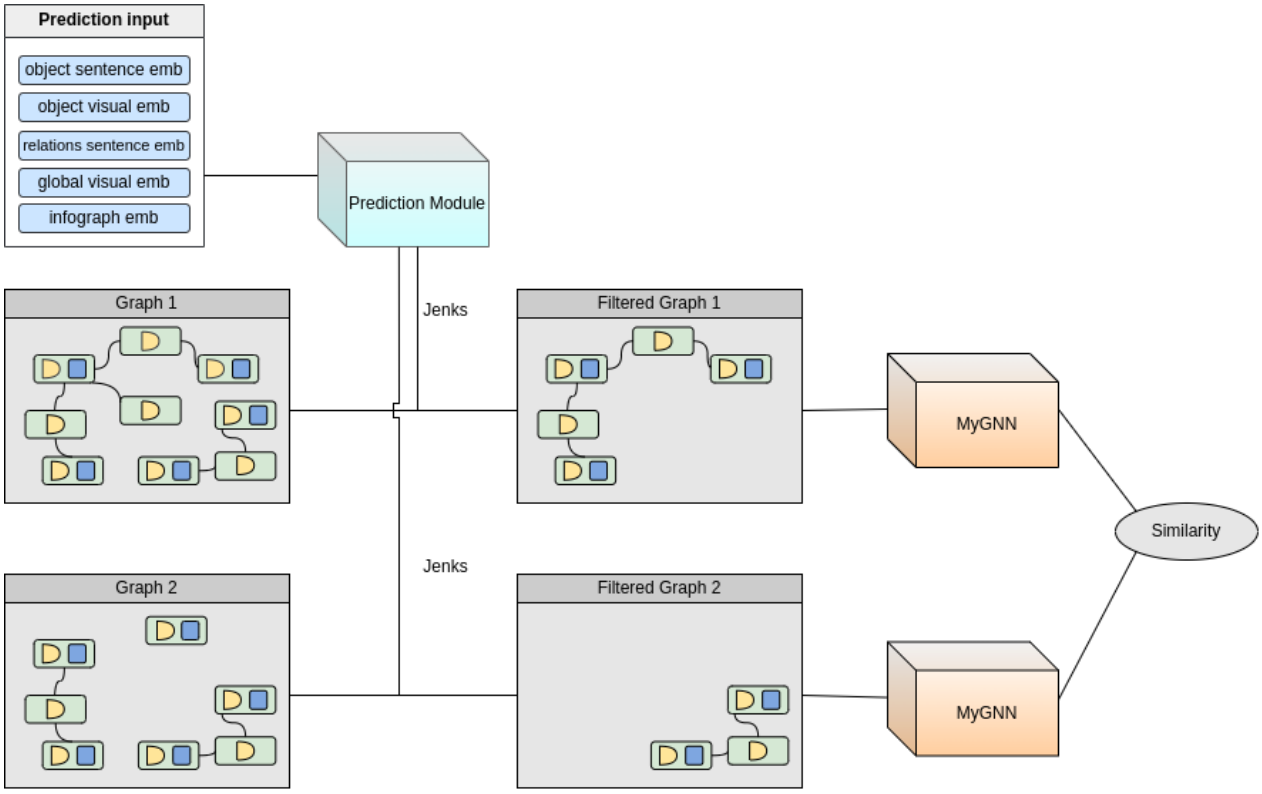


Figure 7.2.6: During inference , firstly we filter our graphs based on the importance of each object and triplet in them , with the use of the importance prediction module , and then we pass them through a GNN , trained as discussed above. Finally the similarity of two images-graphs is obtained by the dot product of their resulting embeddings.

Chapter 8

Experiments

In this chapter we will discuss the various experiments conducted to evaluate the effectiveness of each part of our pipeline and proposed models discussed above. Also information about the dataset we used and the metrics to evaluate our models will be given .We will also present the hyperparameters and implementation choices, and explain the intuition behind each.

Finally we will present the results for different GNN variants and compare the effectiveness of our model with or without different important features of it . We will also present qualitative results for the ground truth most similar images for example query images, as well as the results of our model .

Contents

8.1 Preliminaries

8.1.1 Dataset

We used images common to both the Panoptic Scene Graph Generation (PSG) dataset and the MS-COCO dataset. As discussed above, we selected these because PSG provides better in quality scene-graph annotations for the images, than Visual Genome annotations which often contain noise, irrelevant information, and an inconsistent set of object and relation classes. We further restricted our set to images that also appear in MS-COCO, since we needed their captions to construct ground-truth similarities between image pairs (see Section 7.2). We used the intersection of images from these two sources without discarding images based on graph density.

We then randomly split the resulting dataset into 80 % training, 12 % validation, and 8 % test subsets. This yielded 37 352 images for training, 5 603 for validation, and 3 736 for testing.

This scene graphs use a predetermined vocabulary of object and relationship names. To process them as described above, we first converted each name into an embedding via the sentence-transformer component of the `OpenCLIP_ViT_H_14_laion2b_s32b_b79k` model. We also used that model's visual transformer to obtain visual embeddings for every object.

For the ground-truth similarity between two images, we took five captions per image and converted them into sentence embeddings using the `all-mpnet-base-v2` model from the `sentence-transformers` library in Python. As discussed above, we then computed the average of all pairwise similarities between the five captions of each image pair to yield the final ground-truth similarity score.

8.2 Importance prediction module details

8.2.1 Architectural Decisions and Hyperparameters

The basic outline of the importance-prediction module is described in Section 7.2.2 and illustrated in Figure 7.2.2. For the graph-textual embedding, we trained an Infograph self-supervised model for 120 epochs with an output dimension of 1 024. Each input token is then projected to 1 536 dimensions via a linear layer—deviating from this size (either up or down) negatively impacts performance.

For the transformer encoder, we found that three encoder layers with 32 attention heads each, yielded the best results. After the encoder, the updated tokens pass through a multi-head attention layer with learnable queries: we use 4 queries and project the output again to 1 536 dimensions, we also again use 32 attention heads for this layer. We aggregate the four resulting embeddings using mean pooling and feed them into a simple two-layer MLP with GeLU activations. The first MLP layer reduces the 1 536-dimensional vector to 768 dimensions, and the second layer maps 768 dimensions down to a single scalar output.

8.2.2 Training

We trained a single model to predict both object importance and triplet importance. In our dataset, triplet samples take the form (object_1, relation, object_2), while singleton samples replace the relation and second object with zeros (object_1, 0, 0), prompting the model to predict only the importance of object_1. We adopted this unified approach because it yielded slightly better performance than training separate models for object and triplet importance.

8.2.3 Training Details and Hyperparameters

We trained our model for 10 epochs using the Adam optimizer with a learning rate of 1×10^{-5} and mean squared error loss. The data loader used a batch size of 32, four worker threads, and random shuffling each epoch.

To mitigate overfitting, we applied a dropout rate of 0.2 to each transformer encoder layer. We also employed a learning rate scheduler that multiplies the learning rate by 0.9 after every epoch and implemented early stopping with a patience of 3 epochs.

8.2.4 Ground Truth and Metrics

Our model predicts a scalar score $s \in [0, 1]$ representing the importance of each object or triplet in an image. Binary importance labels (important-unimportant) are derived from these continuous scores in two steps:

1. **Thresholding:** We mark any element with $s \geq 0.4$ as important.
2. **Jenks Natural Breaks:** We apply Jenks clustering with two classes. If the minimum score of the "important" class and the maximum score of the "unimportant" class differ by less than 0.1, we merge all elements into the important class.

The final set of important objects (and triplets) for each image is taken as the union of the elements selected by these two methods.

For each image, we evaluate the continuous predictions against ground truth using Spearman's rank correlation coefficient ,Distance correlation, MSE and MAE (see Section 3.5.6). For the resulting binary labels (important vs. unimportant), we compute accuracy, precision, recall, and F1 score (see Section 3.5.6).
Example. For one image with seven objects, the ground-truth scalar importance scores and their binary labels are

$$[0.1266, 0.1266, 0.4529, 0.4529, 0.3555, 0.0772, 0.0389] \longrightarrow [0, 0, 1, 1, 1, 0, 0],$$

and our model's predictions are

$$[0.2829, 0.0723, 0.4650, 0.4475, 0.3467, 0.0323, 0.0413] \longrightarrow [1, 0, 1, 1, 1, 0, 0].$$

8.2.5 Quantitative results

In this section we evaluate the performance of our importance–prediction module. We also examine the effect of the multi-head attention layer with *learned queries* by comparing the full model (**Learnable Queries**) to a version without it (**No Learnable Queries**). The alternative architecture still uses three transformer–encoder layers (adding more did not help) followed by mean pooling and the same MLP described earlier. Although the model is trained on the combined dataset (objects + triplets), we report results separately for triplet prediction and object prediction.

We evaluate our models in two "levels" .With the regression metrics we evaluate the ability of our model to predict the scalar importance scores for objects and triplets (eg for the example in section 8.2.4 the scalar scores for the first object are 0.1266 in the ground truth scores and 0.2829 for our models prediction) . Similarly with the classification scores we compare the masks we would get with our algorithm (thresholding & Jenks) from the ground truth scalar scores and the scalar predictions. The results shown are the average results for the metrics individually calculated for each image.

Model	Spearman	Avg Dist. Corr.	Avg MSE	Avg MAE
Learnable Queries	**0.6655**	**0.8556**	**0.0100**	**0.0755**
No Learnable Queries	0.6596	0.8529	0.0102	0.0763

Model	Acc.	Prec.	Rec.	F1
Learnable Queries	**0.8267**	**0.8445**	0.8356	**0.8400**
No Learnable Queries	0.8214	0.8340	**0.8390**	0.8365

Table 8.1: Triplet-importance results on the test set, split into regression (top) and classification (bottom) metrics.

Model	Spearman	Avg Dist. Corr.	Avg MSE	Avg MAE
Learnable Queries	**0.5829**	**0.7690**	**0.0093**	**0.0694**
No Learnable Queries	0.5813	0.7671	0.0094	0.0699

Model	Acc.	Prec.	Rec.	F1
Learnable Queries	**0.7891**	**0.6866**	0.6818	**0.6842**
No Learnable Queries	0.7814	0.6707	**0.6826**	0.6766

Table 8.2: Object-importance results on the test set, split into regression (top) and classification (bottom) metrics.

Union Algorithm Because object-level accuracy is still modest, we also mark an object as important if it belongs to a triplet predicted important. This union improves object metrics:

Model	Acc.	Prec.	Rec.	F1
Learnable Queries	**0.8127**	**0.8095**	0.8109	**0.8102**
No Learnable Queries	0.8051	0.7964	**0.8123**	0.8042

Table 8.3: Object-importance metrics after taking the union of predicted-important objects and objects inside predicted-important triplets.

Although an 81 % accuracy may appear low for graph filtering, we emphasize that the ground-truth labels are noisy.For example when two instances share the same class (e.g. *person*) but only one is visually salient, both are marked important. This noise lowers the maximum achievable accuracy, so we assess the module mainly through its downstream impact on the full system.

8.2.6 Qualitative Results

To illustrate the behavior of our importance-prediction module, the following figures show example images from the PSG dataset alongside their corresponding scene graphs. In each graph, nodes predicted as important by our model are colored green, while those deemed unimportant appear in red. Edges that belong to triplets classified as important are likewise drawn in green, with all remaining edges in red. As the examples demonstrate, our module effectively suppresses many irrelevant objects and relations—shown in red—and consistently highlights the truly salient elements of the scene in green.

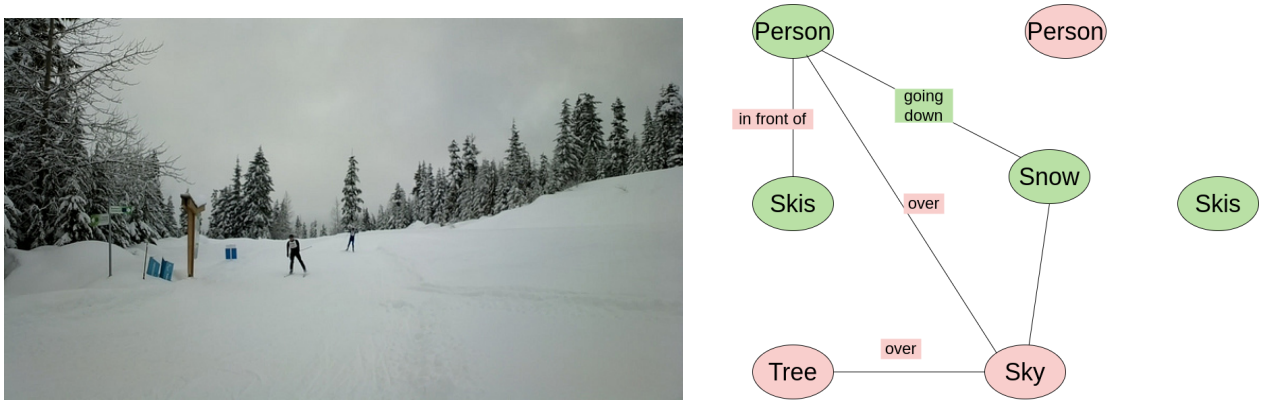


Figure 8.2.1: An example of an image with its graph, illustrating our model's predictions.

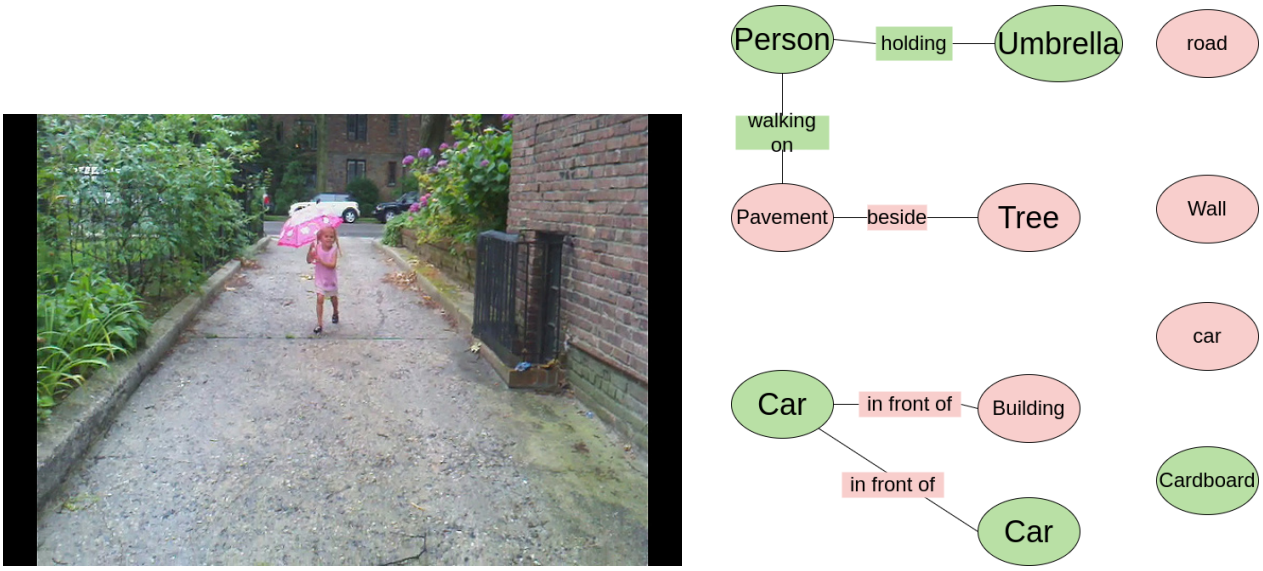


Figure 8.2.2: We observe that although most of our model's predictions are intuitively correct, there are errors. For example, here "cardboard" was predicted as important even though it does not appear in the image.

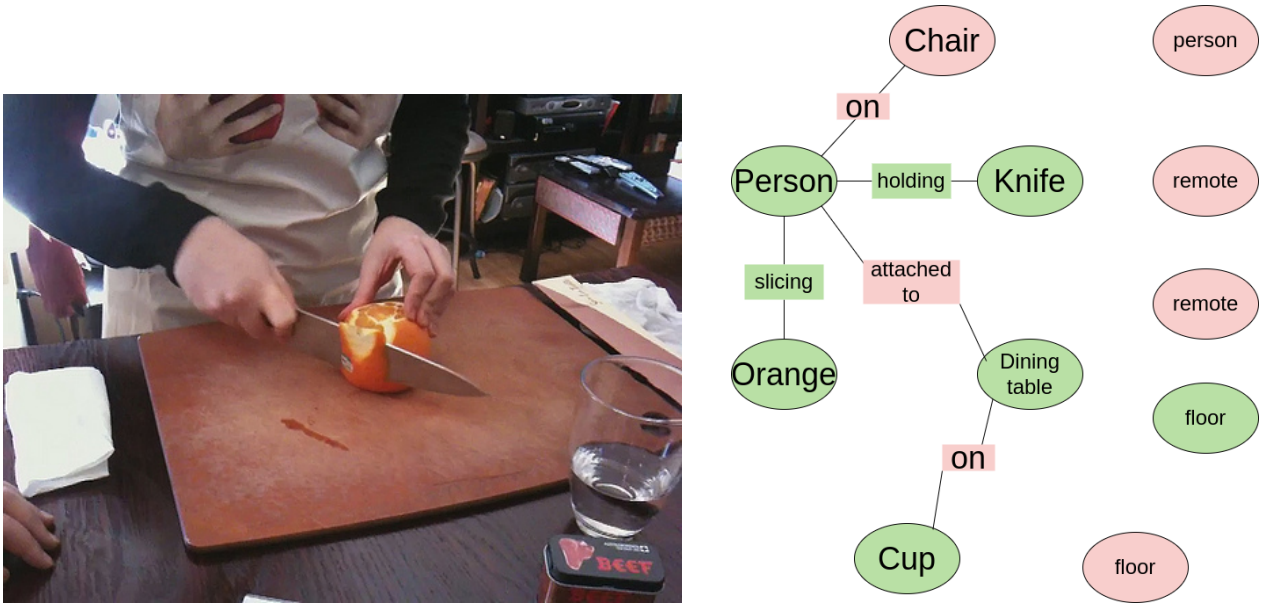


Figure 8.2.3: Our model correctly identifies in the image the objects that are intuitively important, as well as the key actions (slicing, holding).

8.3 GNN Details

In this section, we discuss the details of our GNN model, as well as standard GNNs and the parameters used for their architectures and training. All of our models were implemented using the PyTorch Geometric library, which provides the GNN layers we employ (GATv2 convolutional layers, GAT convolutional layers, GCN convolutional layers, and GIN convolutional layers).

8.3.1 Model Architecture

As described in Section 7.2.4, our model—Edge Aware GATv2—uses our proposed modified GATv2 layer in the first convolutional layer. In subsequent layers, it employs the standard GATv2 layer. This design is intentional: in the first layer, our model integrates information about neighbor-node embeddings and edge embeddings into the node representations. While the node embeddings update, the edge embeddings remain fixed, as simple techniques to update them proved troublesome and did not yield performance gains.

As mentioned in Section 7.2.4, our node embeddings have 2053 dimensions (1024 textual + 1029 visual) and our edge embeddings have 1024 dimensions. We first split each node embedding into its textual and visual components, then project both parts through separate linear layers to 1024 dimensions each. During message passing, we add the edge embedding to the textual part of each updated node embedding, as described above.

Our model performs best with three layers of dimensions [3072, 2048, 2048]. This decoder-like architecture—in which the first layer expands dimensionality—is intuitively sound: combining textual and edge embeddings inevitably introduces noise, so mapping to a higher-dimensional space helps to isolate their most informative features (Similar to the up-projection used in Graph U-Nets to refine noisy graph signals [25]). Furthermore, following standard practice with residual connections for GNNs as well as CNNs [31],[52], we omit a residual connection in the first layer (immediately after the node update) but include one in the second and third layers to stabilize training and prevent oversmoothing (see Section 5.5). This choice is particularly important for our custom layer, since a first-layer residual would dilute the injected edge information.

After the first layer (our custom Edge Aware GATv2 layer), we pass the node embeddings directly—without a ReLU activation—to the next layer. In contrast, after the second and third layers, we apply a ReLU activation (before their residual connections), which slightly improves performance and ensures we retain all salient information from the first layer. Finally, we project the resulting node embeddings through a linear layer to 2048 dimensions and mean-pool them to obtain the final graph-level embedding.

We also evaluated different GNN variants using the same design choices as above—except that the first layer was the same as the subsequent layers rather than custom. Specifically, we tried GAT, GCN, and GIN, as well as GATv2 without our custom first layer. Our model still achieved the best performance. For some of these variants, increasing the first-layer dimension did not yield significant gains. For GAT and GATv2, we used dimensions [3072, 2048, 2048]; for GCN and GIN, the optimal dimensions were [2048, 2048, 2048].

8.3.2 Training Details

We trained all models for 60 epochs using the Adam optimizer with a learning rate of 10^{-4} for the first 20 epochs. Thereafter, we applied a scheduler after each epoch that reduced the learning rate by 10%. We used the default Adam parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The batch size was 32, and we applied a dropout rate of 0.1 to each convolutional layer. To avoid overfitting, we used early stopping with a patience of 20 epochs.

8.3.3 Model Evaluation

Our test set consisted of 3,736 images. We used each image in turn as a query and ranked the remaining images by the dot-product similarity of their embeddings produced by our GNN models. We also compared these rankings against the ground-truth rankings and similarity scores, as described previously. For the metrics which require a ground truth, specific in number relevant items (MRR, MAP, Precision) , we used the first 50 most relevant items in the ground truth list. The evaluation metrics we employed are NDCG, MAP, MRR, and Precision (Section 3.5.6).

Our model, which uses a custom GATv2-like first layer to inject edge information, outperforms the other variants, as shown in the metrics tables. It also surpasses IRSGS [99], which lacks both visual embeddings

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.8803**	**0.8926**	**0.8994**	**0.9081**	**0.9143**	**0.9185**	**0.9216**
MyGATv2	0.8779	0.8908	0.8983	0.9071	0.9134	0.9176	0.9206
MyGAT	0.8758	0.8903	0.8969	0.9060	0.9120	0.9164	0.9197
MyGCN	0.8690	0.8792	0.8865	0.8963	0.9030	0.9080	0.9116
MyGIN	0.8432	0.8613	0.8699	0.8815	0.8897	0.8950	0.8992
IRSGS GCN	0.7401	0.7502	0.7559	0.7647	0.7734	0.7817	0.7891
IRSGS GIN	0.6979	0.7120	0.7217	0.7341	0.7433	0.7513	0.7580

Table 8.4: NDCG@k across models, as it can be seen our model with edge embeddings injection in the first layer outperforms other variants.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.9269**	**0.9070**	**0.8782**	**0.8558**	**0.8363**	**0.8184**
MyGATv2	0.9069	0.9257	0.9049	0.8760	0.8534	0.8336	0.8157
MyGAT	0.9020	0.9241	0.9031	0.8732	0.8499	0.8301	0.8125
MyGCN	0.8943	0.9119	0.8878	0.8563	0.8337	0.8138	0.7959
MyGIN	0.8506	0.8857	0.8600	0.8285	0.8061	0.7874	0.7706
IRSGS GCN	0.6603	0.7331	0.6968	0.6495	0.6202	0.5989	0.5825
IRSGS GIN	0.5803	0.6635	0.6322	0.5901	0.5640	0.5456	0.5303

Table 8.5: MAP@k across models

and edge information in its retrieval process. Moreover, attention-based architectures (GATv2 and GAT) consistently outperform traditional GCN and GIN layers.

Model	MRR
Edge Aware GATv2	**0.9469**
MyGATv2	0.9448
MyGAT	0.9425
MyGCN	0.9347
MyGIN	0.9080
IRSGS GCN	0.7703
IRSGS GIN	0.7059

Table 8.6: MRR across models

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.8802**	**0.8512**	**0.8060**	**0.7619**	**0.7144**	**0.6639**
MyGATv2	0.9069	0.8757	0.8488	0.8039	0.7597	0.7124	0.6616
MyGAT	0.9020	0.8722	0.8419	0.7989	0.7558	0.7092	0.6604
MyGCN	0.8943	0.8531	0.8236	0.7806	0.7377	0.6929	0.6458
MyGIN	0.8506	0.8199	0.7932	0.7500	0.7119	0.6687	0.6244
IRSGS GCN	0.6603	0.6111	0.5759	0.5325	0.5036	0.4803	0.4559
IRSGS GIN	0.5803	0.5402	0.5174	0.4839	0.4560	0.4316	0.4099

Table 8.7: Precision@k across models

8.4 Ablation Study

In this section, we highlight the effectiveness of various design choices and model variants. We compare our best model, Edge Aware GATv2, against several ablated versions:

- **-Pruning**: without graph filtering.
- **-Multi-head Attention**: without the multi-head attention with learned queries in the importance prediction module.
- **-Edge Injection**: without injecting edge embeddings in the first custom layer.
- **-Global**: without global visual embeddings in the graphs.
- **-Triplet Significance**: filtering based solely on object importance (ground truth for training, predicted importance for inference) instead of our union algorithm 7.2.3.
- **-Object Significance**: filtering by retaining only objects that are part of an important triplet and their corresponding edges.
- **-Visual Information**: without any visual information in the graphs.

The performance of this ablated versions compared to our best Edge Aware GATv2 model can be found in the following tables.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.8803**	**0.8926**	**0.8994**	**0.9081**	**0.9143**	**0.9185**	**0.9216**
-Edge Injection	0.8779	0.8908	0.8983	0.9071	0.9134	0.9176	0.9206
-Global	0.8426	0.8572	0.8653	0.8743	0.8808	0.8856	0.8893
-Triplet Significance	0.8721	0.8848	0.8921	0.9006	0.9074	0.9120	0.9150
-Object Significance	0.8035	0.8160	0.8234	0.8348	0.8424	0.8485	0.8529
-Pruning	0.8758	0.8909	0.8979	0.9065	0.9128	0.9170	0.9202
-Multi-head Attention	0.8791	0.8925	0.8992	0.9079	0.9141	0.9184	0.9214
-Visual Information	0.7518	0.7719	0.7841	0.8002	0.8115	0.8206	0.8271

Table 8.8: Ablation study: NDCG@k across variants.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.9269**	**0.9070**	**0.8782**	**0.8558**	**0.8363**	**0.8184**
-Edge Injection	0.9069	0.9257	0.9049	0.8760	0.8534	0.8336	0.8157
-Global	0.8555	0.8841	0.8581	0.8238	0.7979	0.7770	0.7586
-Triplet Significance	0.8953	0.9171	0.8954	0.8664	0.8426	0.8230	0.8053
-Object Significance	0.7808	0.8279	0.7969	0.7559	0.7295	0.7088	0.6915
-Pruning	0.9071	0.9240	0.9030	0.8746	0.8520	0.8324	0.8146
-Multi-head Attention	0.9109	0.9262	0.9065	0.8776	0.8553	0.8358	0.8180
-Visual Information	0.6777	0.7534	0.7234	0.6873	0.6645	0.6470	0.6326

Table 8.9: Ablation study: MAP@k across variants.

Model	MRR
Edge Aware GATv2	**0.9469**
-Edge Injection	0.9448
-Global	0.9095
-Triplet Significance	0.9371
-Object Significance	0.8587
-Pruning	0.9439
-Multi-head Attention	0.9462
-Visual Information	0.7881

Table 8.10: Ablation study: MRR across variants.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.9119**	**0.8802**	0.8512	**0.8060**	0.7619	**0.7144**	**0.6639**
-Edge Injection	0.9069	0.8757	0.8488	0.8039	0.7597	0.7124	0.6616
-Global	0.8555	0.8153	0.7857	0.7372	0.6930	0.6487	0.6039
-Triplet Significance	0.8953	0.8655	0.8374	0.7899	0.7481	0.7021	0.6520
-Object Significance	0.7808	0.7378	0.7048	0.6613	0.6210	0.5826	0.5429
-Pruning	0.9071	0.8724	0.8477	0.8012	0.7594	0.7120	0.6619
-Multi-head Attention	0.9109	0.8799	**0.8515**	0.8057	**0.7620**	0.7143	0.6637
-Visual Information	0.6777	0.6509	0.6287	0.6000	0.5714	0.5435	0.5129

Table 8.11: Ablation study: Precision@k across variants.

There are a lot of things we can deduce from the above tables. First of all, our full model outperforms other variants on almost all metrics. But we should also mention that our model's metrics are very close to those without edge injection and without pruning. The former is understandable, since the node textual embeddings, the global visual embedding, and the node visual embeddings can capture scene information without explicit edge features. We can also assume that, because our visual model is CLIP—which is trained on text-to-image pairing—it may capture the most significant relations in an image and encode them in the global visual embedding, reducing the need to fuse edge data into the node textual embeddings. Apart from that, even if the difference is small, our model still performs consistently better than the non–edge-injection variant. Also, the model without graph pruning has similar scores across all metrics, likely because our GAT attention models can identify important objects through the visual node embeddings without pre-filtering. However, errors in our importance-prediction module's predictions also hurt performance. On the other hand, our full model still outperforms it slightly, so pruning is indeed beneficial.

All other ablated versions show a significant reduction in performance. From the metrics, it is obvious that visual information in our graphs significantly improves our model's expressivity and capacity. Another vital component is the union algorithm used for graph filtering, as alternative filtering strategies (–Triplet Significance, –Object Significance) lead to substantial performance drops. Furthermore, the global node is crucial for performance gains, which makes sense since CLIP's visual embeddings are already capable of text-to-image ranking, and our graph structure further enhances this ability.

Here, we also perform experiments for the architecture of our model and graph-pruning, on graphs with only textual object information and of course no global visual nodes. Although visual features are clearly essential for peak performance, evaluating these components in a purely text-only setting lets us see their individual impact: both the pruning mechanism and the injection of edge information in the first GATv2 layer yield noticeably larger relative gains when no visual cues are available. The tables below illustrate these effects—not because they rival our best multimodal model overall, but to show that these components and tactics could still be beneficial in other contexts where visual data is absent.

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.7518**	**0.7719**	**0.7841**	**0.8002**	**0.8115**	**0.8206**	**0.8271**
-Edge Injection	0.7465	0.7604	0.7727	0.7882	0.7994	0.8084	0.8153
-Pruning	0.7475	0.7601	0.7683	0.7808	0.7893	0.7961	0.8017
-Edge Injection,Pruning	0.7414	0.7529	0.7617	0.7741	0.7826	0.7897	0.7954

Table 8.12: Ablation subset: NDCG@k

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.6777**	**0.7534**	**0.7234**	**0.6873**	**0.6645**	**0.6470**	**0.6326**
-Edge Injection	0.6724	0.7388	0.7057	0.6694	0.6463	0.6287	0.6143
-Pruning	0.6761	0.7416	0.7065	0.6659	0.6404	0.6210	0.6051
-Edge Injection,Pruning	0.6510	0.7268	0.6947	0.6538	0.6286	0.6096	0.5943

Table 8.13: Ablation subset: MAP@k

Model	MRR
Edge Aware GATv2	**0.7881**
-Edge Injection	0.7798
-Pruning	0.7815
-Edge Injection,Pruning	0.7658

Table 8.14: Ablation subset: MRR

Model	@1	@5	@10	@20	@30	@40	@50
Edge Aware GATv2	**0.6777**	**0.6509**	**0.6287**	**0.6000**	**0.5714**	**0.5435**	**0.5129**
-Edge Injection	0.6724	0.6232	0.6069	0.5770	0.5496	0.5226	0.4958
-Pruning	0.6761	0.6278	0.6015	0.5637	0.5323	0.5027	0.4737
-Edge Injection,Pruning	0.6510	0.6140	0.5875	0.5519	0.5208	0.4920	0.4641

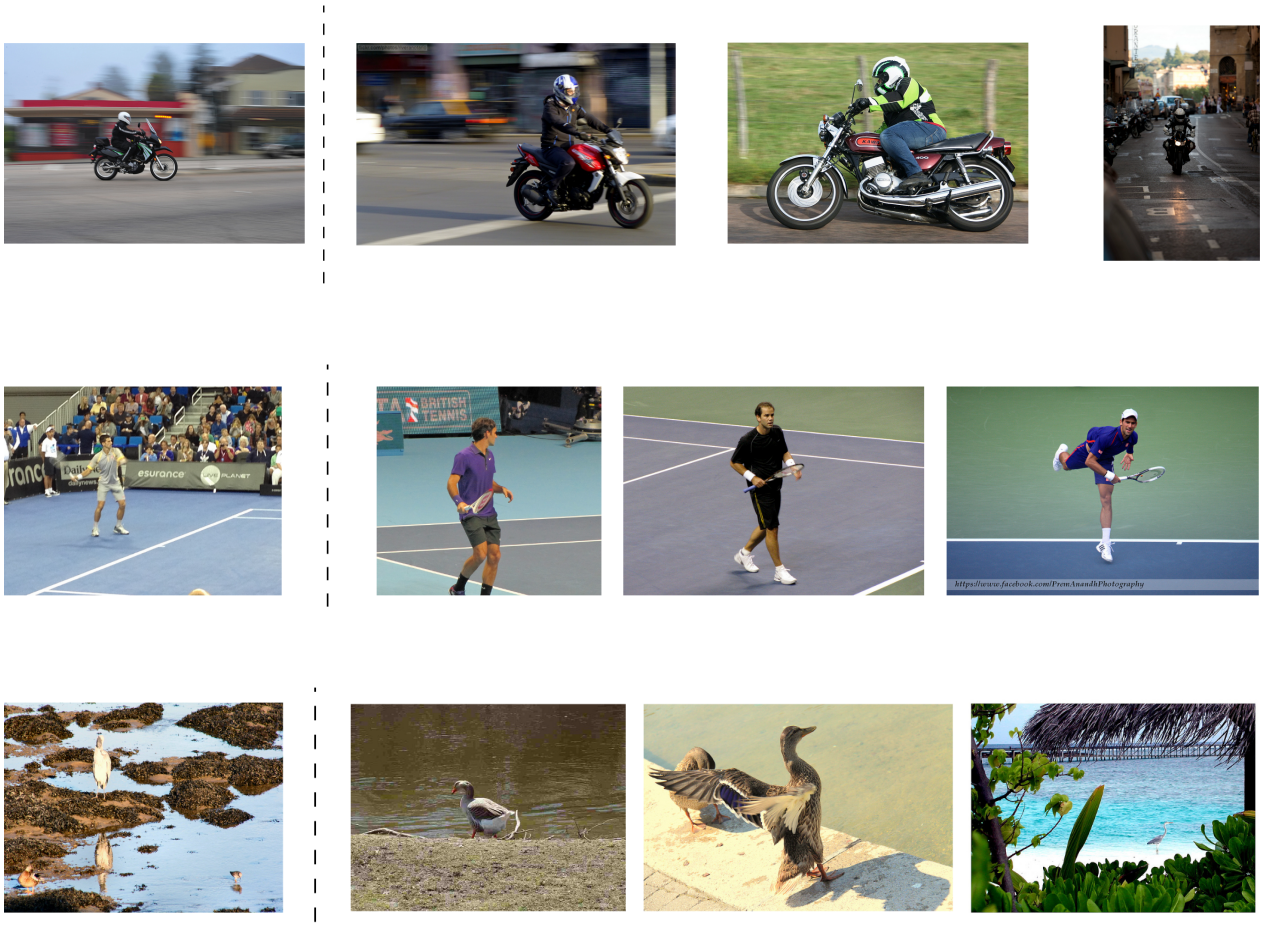
Table 8.15: Ablation subset: Precision@k

From the tables, it is evident that both graph pruning and edge injection in the first layer yield substantial performance gains—gains that are even more pronounced in a text-only setting than when visual information is available. This underscores how critically graph quality drives our retrieval results and suggests that further improvements to the importance-prediction module could deliver additional score boosts. One might also imagine applying a purely statistical pruning algorithm—based on node and edge labels or graph structure—without any visual cues; while such an approach would likely be less accurate, it points to promising directions for future work beyond the scope of this paper.

8.5 Qualitative Results

In this section, we present qualitative results for our best model, and compare its predictions with the ground-truth most similar images for a handful of query images. We also contrast results from some ablated models to highlight our model's unique strengths.

Firstly, we show example results for the top three most similar retrieved images for several query images.




Cupcakes Under Cover

From these examples, it is clear that our model retrieves relevant images under all scenarios, using both visual and semantic information. In the first example the top retrieval is visually very close to the original (both are blurred by the motorcycle's motion). In the second example ,all the tennis courts in the retrieved images are blue, even though our dataset also contains green and sand courts. The semantic information from the scene graphs is especially evident in examples 5–7: Example 5 retrieves a horse in snow exactly as in the query. Moreover in Example 6 , the first retrieved image is semantically the same as the query—a woman walking with an umbrella—and our model successfully retrieves it even though the query is in color and the retrieved image is in black and white.Furthermore example 7 returns cakes decorated with drawn animals, just like the query image.

Now we compare additional results of our best model with the ground-truth most similar images. In the following examples, on the left we show the query image; on the top right, the two ground-truth most similar images; and on the bottom right, our model's top two predictions.

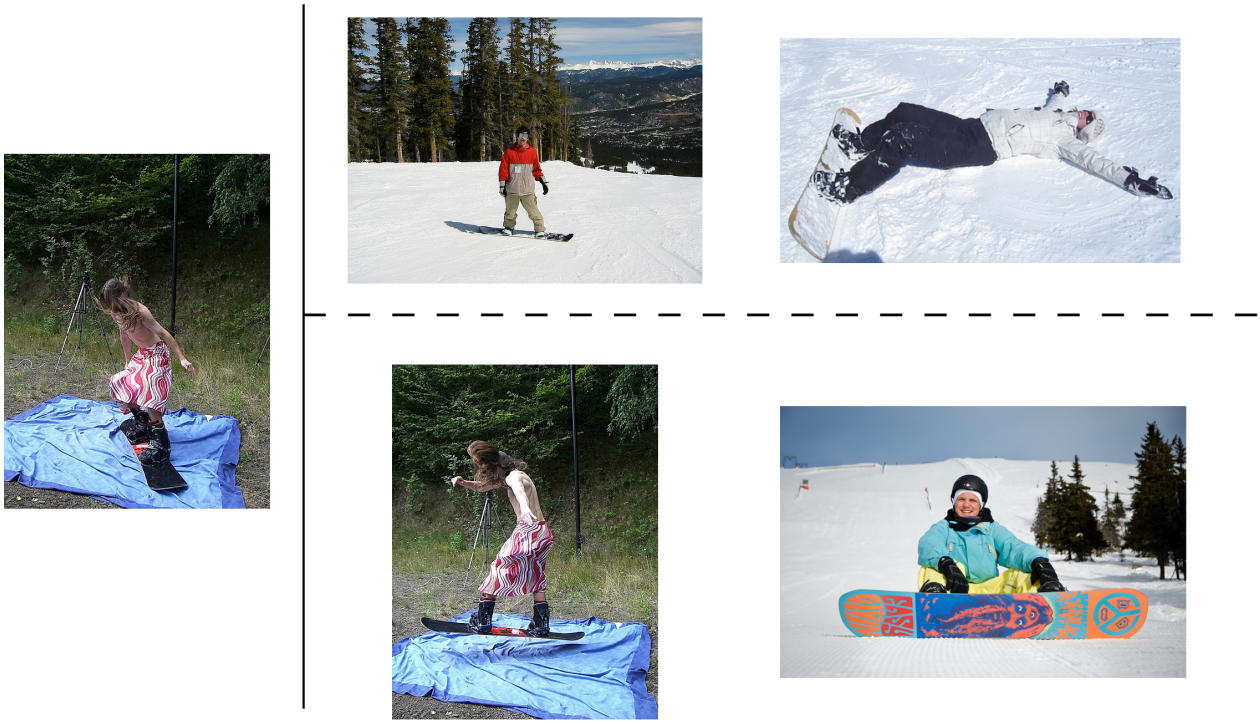


Figure 8.5.1: In this example, our model successfully retrieves the most similar image based on visual cues, even better than the ground truth. Query image on the left . Top right ground truth most similar images , and down right our best models predictions.

Figure 8.5.2: Query image on the left - top right ground truth most similar images , and down right our best models predictions.

Figure 8.5.3: Query image on the left - top right ground truth most similar images , and down right our best models predictions.

The above examples (8.5.1,8.5.2,8.5.3) serve as confirmation that, even when our model retrieves different top results than the ground-truth most similar images, those results remain highly relevant and can, in some cases, be even better than the ground truth.

We will now showcase examples where the effectiveness of our model's injection of edge embeddings—which incorporates information about object relations—allows it to retrieve more semantically relevant results than a model without this modified GATv2 layer. First, we present an example comparing the retrieved images of our Edge Aware GATv2 model with those from the simple MyGATv2 model, where the only distinction is that the former injects edge information in the first layer while the latter does not. We expect that when object relations are important and not easily deduced, our Edge Aware model will perform better.

For this reason, we include the filtered scene graphs for the query image as well as those for the images retrieved by our models, and highlight which relations should influence the retrieval process.

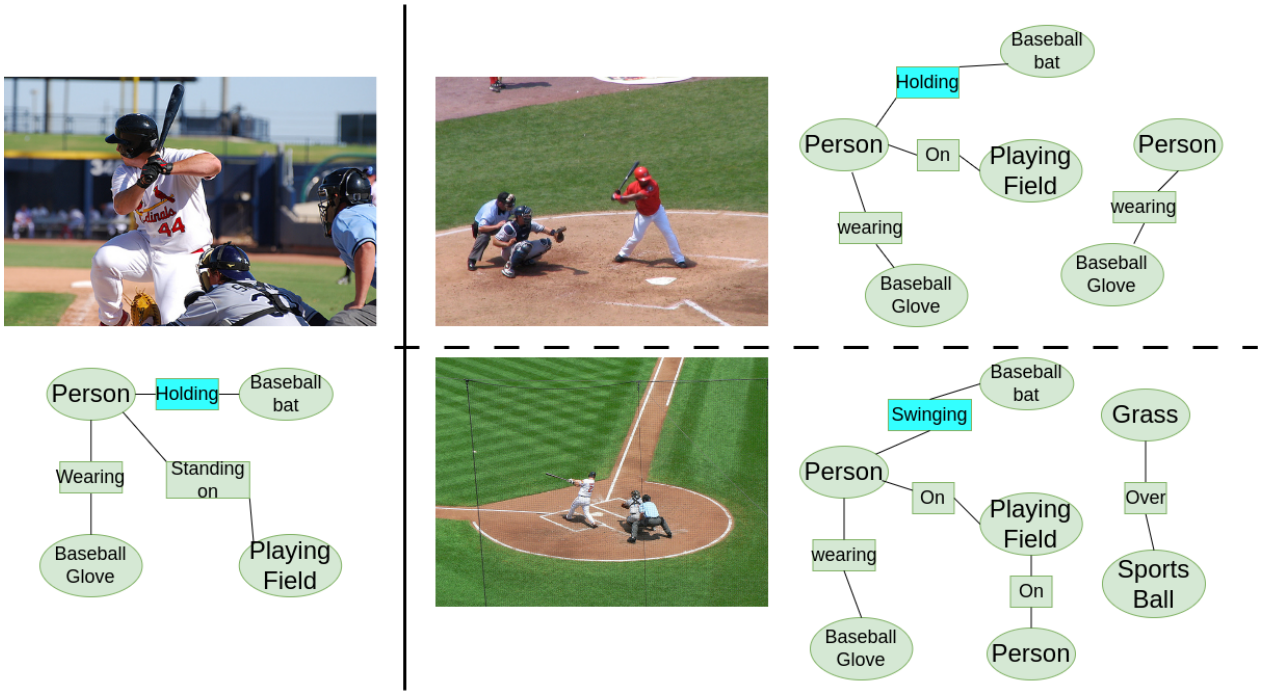


Figure 8.5.4: In this example, our best model—which incorporates edge embeddings in message passing—correctly identifies that the relation between the person and the baseball bat is "holding" rather than "swinging". In the left the query image - in the top right our best models prediction and down right the same model without edge injection (MyGATv2) .

As mentioned above, in most cases the results of our models are very close, because visual information alone is often enough for good retrieval. To further illustrate the ability of the edge-injection layer, we showcase examples of the different retrieved images from Edge Aware GATv2 and the same model without edge injection (MyGATv2) using input graphs without visual information, as we did in the ablation study section.

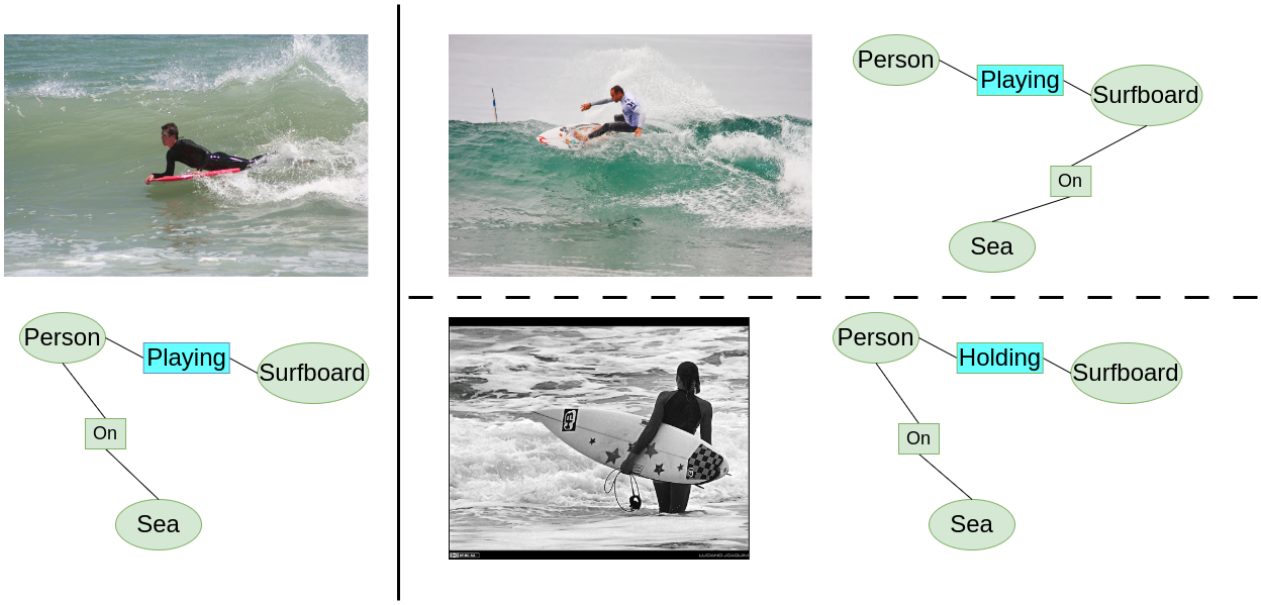


Figure 8.5.5: On the left, a query image; on the top right, our best model's predictions (graphs without visual information); on the bottom right, predictions from the same model without edge injection.

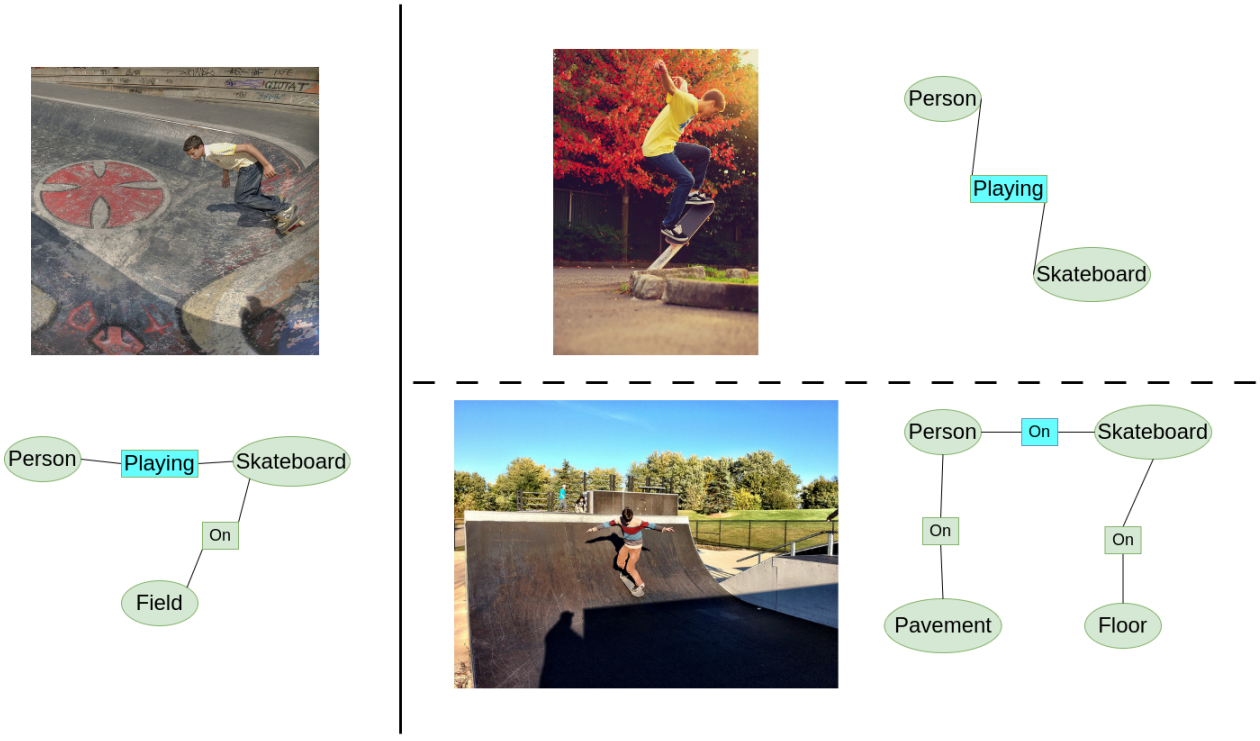


Figure 8.5.6: On the left, a query image; on the top right, our best model's predictions (graphs without visual information); on the bottom right, predictions from the same model without edge injection.

The above examples (8.5.5,8.5.6) firstly make clear that our model can in fact use the edge embeddings to retrieve better images. In the first example, our model correctly retrieves an image where the relation between the "Person" and the "Surfboard" is "Playing" and not "Holding." In the second example, interestingly, the retrieved image for the model without edge injection (bottom right) is the most similar based on ground-truth scores. Visually, that image does look more similar to the query, but the graph retrieved by our best model—with edge injection—has a structure that aligns more closely with the query graph.That can happen and shows that many of the graphs in our dataset are very noisy and not annotated consistently.

Chapter 9

Conclusion

9.1 Reflecting on the findings

In this thesis, we addressed the problem of incorporating visual features into image retrieval via scene graphs. We began by reviewing the basic machine-learning concepts used in our proposal, then proceeded to describe our models in detail and to evaluate them both quantitatively and qualitatively.

Specifically, we introduced an importance-prediction module based on a transformer encoder with a multi-head attention layer and learned queries. This module identifies the most significant objects and triplets (object 1, relation, object 2) in a scene graph by combining object names, their visual embeddings, and the visual features of the full image. Next, we proposed a pruning algorithm that takes these predictions and filters each graph, retaining only the most relevant objects and relations. For ground-truth association of an object or triplet with the image, we compare their embeddings with the embeddings of the image captions. This step was motivated by observing that many graphs in the Panoptic Scene Graph Generation dataset contain irrelevant items and inconsistent annotations. We also evaluated an ablated version of this model without the multi-head attention with learned queries layer and compared their performance.

We then fused the object-level visual information with the textual information for these objects, and we added a global node containing only the full-image visual features, thus constructing a richer, multimodal graph. To process these graphs, we designed an Edge-Aware GATv2 layer that incorporates edge embeddings directly into message passing and attention calculations for each node. Our network comprises one such Edge-Aware GATv2 layer followed by standard GATv2 layers: the former captures edge-specific information that traditional GNNs ignore, while the latter refines node representations. Our intuition was that scene graphs often encode useful semantic information in their edges—information that can enhance retrieval but is typically overlooked either because traditional GNNs do not use edge embeddings or because many edges in the Visual Genome and Panoptic Scene Graph Generation datasets are semantically weak.

We also experimented with multiple GNN architectures—GCN, GIN, GAT, and GATv2—on top of our proposed Edge-Aware layer and evaluated their performance using commonly used metrics for image retrieval and ranking: MRR, MAP, NDCG, and Precision. With these metrics, we first note that models using the GATv2 layer outperformed the others. Specifically, for NDCG@1, Edge-Aware GATv2 achieved results similar to, but slightly better than, the same model with standard GATv2 (no edge injection) or with GAT layers. These models outperformed the GCN and GIN baselines by 1 % and 2 % in MRR, 1.5 % and 3 % in NDCG@5, and showed generally higher scores across all metrics. Overall, our best model was Edge-Aware GATv2, which led in every metric, though the gains over standard GATv2 and GAT were modest.

To further validate each component of our pipeline, we evaluated ablated versions of Edge-Aware GATv2. The results showed that both pruning and edge injection indeed boost performance. We also demonstrated that incorporating object-level visual information and our global visual node significantly improves retrieval performance, and that the multi-head attention layer with learned queries for importance prediction enhances overall accuracy.

On the other hand, since our multimodal graphs are already highly expressive, we assessed the impact of pruning and the Edge-Aware GATv2 layer on graphs containing only textual information. In this context, we observed even clearer improvements across all metrics, underscoring the value of edge injection and pruning. We also presented qualitative results indicating that edge injection in fact influenced our model's decisions.

Finally, we presented qualitative results highlighting different aspects of our model and illustrating how it outperforms its ablated variants.

9.2 Future Work

Last but not least, building on our ideas and the work we presented, we would like to suggest future ideas that can maybe improve the performance of image-to-image retrieval models.

- Firstly, for the improvement of the importance prediction module, the most important aspect that decreases its effectiveness is judged to be the computation of the ground-truth similarity scores. That is why an improved algorithm for the calculation of these scores is believed to further improve the performance of this module and, as a consequence, enable better filtering of our graphs—possibly by combining information from the triplets and the objects-in-scene graph to determine which part of the image each object is referring to.
- An algorithmic translation of the relations in the Panoptic Scene Graph Generation dataset—unlike our current undirected approach, which uses identical edge embeddings for both directions of a node pair—could enrich our graphs. For example, if the relation between a person and an object is "on," the reverse relation from the object back to the person would be "under." In theory, this asymmetry could inject additional semantic information into our graphs and help any edge-aware GNN better capture and utilize edge-specific features.
- Although our Edge Aware GATv2 layer injected edge information into the nodes only in the first layer of our network, this information persists into subsequent layers. An architecture for a layer that incorporates edge information and can be stacked in multiple layers should update both the edge embeddings and the node embeddings at each layer. Furthermore, different pooling functions other than the sum we used for aggregating edge and node information should be explored. Moreover, this GNN architecture can be tested on additional datasets where edge information is more important and can influence task performance in a more drastic manner.
- Although in this thesis we experimented with the most popular GNN architectures—GCN, GAT, GIN, and GATv2—we can test additional models, some of which are already relational (incorporating edge information in their algorithms) and, for others, explore ways to adapt them to use edge information. Furthermore, different visual and textual embedding models could boost our model's performance.

Chapter 10

Bibliography

[1] Agarwal, A., Mangal, A., and Vipul. *Visual Relationship Detection using Scene Graphs: A Survey*. 2020. arXiv: 2005.08045 [cs.CV]. URL:

[2] Alaghbari, K. et al. "Deep Autoencoder-Based Integrated Model for Anomaly Detection and Efficient Feature Extraction in IoT Networks". In: *IoT* 4 (Aug. 2023), pp. 345–365. DOI: 10.3390/iot4030016.

[3] Arar, M., Shamir, A., and Bermano, A. H. *Learned Queries for Efficient Local Attention*. 2022. arXiv: 2112.11435 [cs.CV]. URL:

[4] Bahdanau, D., Cho, K., and Bengio, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL:

[5] Bai, Y. et al. *Are Transformers More Robust Than CNNs?* 2021. arXiv: 2111.05464 [cs.CV]. URL:

[6] Bay, H., Tuytelaars, T., and Van Gool, L. "SURF: Speeded up robust features". In: vol. 3951. July 2006, pp. 404–417. ISBN: 978-3-540-33832-1. DOI: 10.1007/11744023_32.

[7] Beaumont, R. *LARGE SCALE OPENCLIP: L/14, H/14 AND G/14 TRAINED ON LAION-2B*. LAION Blog, [Online; accessed 18-June-2025]. Sept. 2022.

[8] Bishop, C. "Pattern Recognition and Machine Learning". In: vol. 16. Jan. 2006, pp. 140–155. DOI: 10.1117/1.2819119.

[9] Borgwardt, K. M. and Kriegel, H.-P. "Shortest-Path Kernels on Graphs". In: *Proceedings of the Fifth IEEE International Conference on Data Mining*. ICDM '05. USA: IEEE Computer Society, 2005, pp. 74–81. ISBN: 0769522785. DOI: 10.1109/ICDM.2005.132. URL:

[10] Brody, S., Alon, U., and Yahav, E. *How Attentive are Graph Attention Networks?* 2022. arXiv: 2105.14491 [cs.LG]. URL:

[11] Butler, S. and Chung, F. "Spectral Graph Theory". In: Dec. 2013, pp. 811–824. ISBN: 9781466507289. DOI: 10.1201/b16113-54.

[12] Chaidos, N. et al. *SCENIR: Visual Semantic Clarity through Unsupervised Scene Graph Retrieval*. 2025. arXiv: 2505.15867 [cs.CV]. URL:

[13] Chen, J. and Chen, H. *Edge-Featured Graph Attention Network*. 2021. arXiv: 2101.07671 [cs.LG]. URL:

[14] Chen, M. et al. *Simple and Deep Graph Convolutional Networks*. 2020. arXiv: 2007.02133 [cs.LG]. URL:

[15] Cho, K. et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL]. URL:

[16] Cortes, C. and Vapnik, V. "Support-vector networks". In: *Chem. Biol. Drug Des.* 297 (Jan. 2009), pp. 273–297. DOI: 10.1007/%2FBF00994018.

[17] Defferrard, M., Bresson, X., and Vandergheynst, P. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG]. URL:

[18] Devlin, J. et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL:

[19] Dimitriou, A. et al. *Structure Your Data: Towards Semantic Graph Counterfactuals*. 2024. arXiv: 2403.06514 [cs.CV]. URL:

[20] Dosovitskiy, A. et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL:

[21] Duchi, J., Hazan, E., and Singer, Y. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[22] Elman, J. "Finding structure in time". In: Feb. 2020, pp. 289–312. ISBN: 9781315784779. DOI: 10.4324/9781315784779-11.

[23] Ester, M. et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[24] Fey, M. and Lenssen, J. E. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG]. URL:

[25] Gao, H. and Ji, S. *Graph U-Nets*. 2019. arXiv: 1905.05178 [cs.LG]. URL:

[26] Gärtner, T., Flach, P., and Wrobel, S. "On Graph Kernels: Hardness Results and Efficient Alternatives". In: vol. 129-143. Jan. 2003, pp. 129–143. ISBN: 978-3-540-40720-1. DOI: 10.1007/978-3-540-45167-9_11.

[27] Gilmer, J. et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG]. URL:

[28] Hamilton, W. L., Ying, R., and Leskovec, J. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI]. URL:

[29] Hammond, D. K., Vandergheynst, P., and Gribonval, R. *Wavelets on Graphs via Spectral Graph Theory*. 2009. arXiv: 0912.3848 [math.FA]. URL:

[30] Hasanzadeh, A. et al. *Bayesian Graph Neural Networks with Adaptive Connection Sampling*. June 2020. DOI: 10.48550/arXiv.2006.04064.

[31] He, K. et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL:

[32] Hendrycks, D. and Gimpel, K. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG]. URL:

[33] Hinton, G. and Salakhutdinov, R. "Reducing the Dimensionality of Data with Neural Networks". In: *Science (New York, N.Y.)* 313 (Aug. 2006), pp. 504–7. DOI: 10.1126/science.1127647.

[34] Hochreiter, S. and Schmidhuber, J. "Long Short-Term Memory". In: *Neural Computation* 9 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[35] Hotelling, H. "Analysis of a complex of statistical variables into principal components." In: *Journal of Educational Psychology* 24 (1933), pp. 498–520. URL:

[36] Huang, X. et al. *TabTransformer: Tabular Data Modeling Using Contextual Embeddings*. 2020. arXiv: 2012.06678 [cs.LG]. URL:

[37] Jenks, G. F. "The Data Model Concept in Statistical Mapping". In: 1967. URL:

[38] Jonker, R. and Volgenant, T. "A shortest augmenting path algorithm for dense and sparse linear assignment problems". In: *Computing* 38 (1987), pp. 325–340. URL:

[39] Kingma, D. P. and Welling, M. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL:

[40] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL:

[41] Kipf, T. N. and Welling, M. *Variational Graph Auto-Encoders*. 2016. arXiv: 1611.07308 [stat.ML]. URL:

[42] Kipf, T. N. and Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG]. URL:

[43] Kolluri, J. et al. "Reducing Overfitting Problem in Machine Learning Using Novel L1/4 Regularization Method". In: June 2020, pp. 934–938. DOI: 10.1109/ICOEI48184.2020.9142992.

[44] Kriege, N. and Mutzel, P. *Subgraph Matching Kernels for Attributed Graphs*. 2012. arXiv: 1206.6483 [cs.LG]. URL:

[45] Krishna, R. et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV]. URL:

[46] Krizhevsky, A., Sutskever, I., and Hinton, G. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[47] Kuhn, H. W. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109. eprint: URL:

[48] Kuznetsova, A. et al. "The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale". In: *International Journal of Computer Vision* 128.7 (Mar. 2020), pp. 1956–1981. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01316-z. URL:
[49] Lecun, Y. et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.
[50] Levesque, V. "Texture Segmentation Using Gabor Filters". In: (Jan. 2001).
[51] Li, D., Zhang, J., and Li, P. "TMSA: A Mutual Learning Model for Topic Discovery and Word Embedding". In: May 2019, pp. 684–692. ISBN: 978-1-61197-567-3. DOI: 10.1137/1.9781611975673.77.
[52] Li, G. et al. *DeepGCNs: Can GCNs Go as Deep as CNNs?* 2019. arXiv: 1904.03751 [cs.CV]. URL:
[53] Li, H. et al. "Keeping Deep Learning Models in Check: A History-Based Approach to Mitigate Overfitting". In: *IEEE Access* 12 (2024), pp. 70676–70689. ISSN: 2169-3536. DOI: 10.1109/access.2024.3402543. URL:
[54] Li, J. et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. 2022. arXiv: 2201.12086 [cs.CV]. URL:
[55] Lindeberg, T. "Scale Invariant Feature Transform". In: vol. 7. May 2012. DOI: 10.4249/scholarpedia.10491.
[56] Liu, Z. et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV]. URL:
[57] Loh, W.-Y. "Classification and Regression Trees". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1 (Jan. 2011), pp. 14–23. DOI: 10.1002/widm.8.
[58] MacQueen, J. "Some methods for classification and analysis of multivariate observations". In: 1967. URL:
[59] Maheshwari, P., Chaudhry, R., and Vinay, V. "Scene Graph Embeddings Using Relative Similarity Supervision". In: *ArXiv* abs/2104.02381 (2021). URL:
[60] Marois, V. et al. *On transfer learning using a MAC model variant*. 2018. arXiv: 1811.06529 [cs.CV]. URL:
[61] "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: (May 2016).
[62] Nair, V. and Hinton, G. "Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair". In: vol. 27. June 2010, pp. 807–814.
[63] Neuhaus, M., Riesen, K., and Bunke, H. "Fast Suboptimal Algorithms for the Computation of Graph Edit Distance". In: Aug. 2006, pp. 163–172. ISBN: 978-3-540-37236-3. DOI: 10.1007/11815921_17.
[64] Oord, A. van den, Li, Y., and Vinyals, O. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG]. URL:
[65] Paul, S. and Chen, P.-Y. *Vision Transformers are Robust Learners*. 2021. arXiv: 2105.07581 [cs.CV]. URL:
[66] Qazanfari, H., AlyanNezhadi, M. M., and Khoshdaregi, Z. N. *Advancements in Content-Based Image Retrieval: A Comprehensive Survey of Relevance Feedback Techniques*. 2023. arXiv: 2312.10089 [cs.CV]. URL:
[67] Radford, A. et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL:
[68] Ramachandran, P., Zoph, B., and Le, Q. V. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE]. URL:
[69] Reimers, N. and Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL]. URL:
[70] Rong, Y. et al. *DropEdge: Towards Deep Graph Convolutional Networks on Node Classification*. 2020. arXiv: 1907.10903 [cs.LG]. URL:
[71] Rosenblatt, F. "The perceptron: A probabilistic model for information storage and organization in the brain [J]". In: *Psychol. Review* 65 (Nov. 1958), pp. 386–408. DOI: 10.1037/h0042519.
[72] Rumelhart, D., Hinton, G., and Williams, R. "Learning Representations by Back-Propagating Errors". In: Sept. 2002, pp. 213–222. ISBN: 9780262281744. DOI: 10.7551/mitpress/1888.003.0013.
[73] Rusch, T. K., Bronstein, M. M., and Mishra, S. *A Survey on Oversmoothing in Graph Neural Networks*. 2023. arXiv: 2303.10993 [cs.LG]. URL:
[74] Schuhmann, C. et al. *LAION-5B: An open large-scale dataset for training next generation image-text models*. 2022. arXiv: 2210.08402 [cs.CV]. URL:

[75] Shao, R. et al. *On the Adversarial Robustness of Vision Transformers*. 2022. arXiv: 2103.15670 [cs.CV]. URL:
[76] Sherashidze, N. et al. "Efficient Graphlet Kernels for Large Graph Comparison". In: *12th International Conference on Artificial Intelligence and Statistics (AISTATS), Society for Artificial Intelligence and Statistics, 488-495 (2009)* 5 (Jan. 2009).
[77] Shervashidze, N. et al. "Weisfeiler–Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12 (2011), pp. 2539–2561.
[78] Shuman, D. I. et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (May 2013), pp. 83–98. ISSN: 1053-5888. DOI: 10.1109/msp.2012.2235192. URL:
[79] Srivastava, N. et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.
[80] Sun, F.-Y. et al. *InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization*. 2020. arXiv: 1908.01000 [cs.LG]. URL:
[81] Szekely, G., Rizzo, M., and Bakirov, N. "Measuring and Testing Dependence by Correlation of Distances". In: *The Annals of Statistics* 35 (Apr. 2008). DOI: 10.1214/009053607000000505.
[82] Touvron, H. et al. *Training data-efficient image transformers distillation through attention*. 2021. arXiv: 2012.12877 [cs.CV]. URL:
[83] Vaswani, A. et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL:
[84] Veličković, P. et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML]. URL:
[85] Veltkamp, R. and Tanase, M. "Content-Based Image Retrieval Systems: A Survey". In: *Technical report, Utrecht University* (Nov. 2000).
[86] Vincent, P. et al. "Extracting and composing robust features with denoising autoencoders". In: Jan. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.
[87] Wang, D., Cui, P., and Zhu, W. "Structural Deep Network Embedding". In: Aug. 2016, pp. 1225–1234. DOI: 10.1145/2939672.2939753.
[88] Wang, G., Shang, Y., and Chen, Y. *Scene Graph Based Fusion Network For Image-Text Retrieval*. 2023. arXiv: 2303.11090 [cs.CV]. URL:
[89] Wang, Y. et al. *A Theoretical Analysis of NDCG Type Ranking Measures*. 2013. arXiv: 1304.6480 [cs.LG]. URL:
[90] Wang, Y. et al. "Hi-SIGIR: Hierachical Semantic-Guided Image-to-image Retrieval via Scene Graph". In: *Proceedings of the 31st ACM International Conference on Multimedia*. MM '23. Ottawa ON, Canada: Association for Computing Machinery, 2023, pp. 6400–6409. ISBN: 9798400701085. DOI: 10.1145/3581783.3612283. URL:
[91] Wikipedia contributors. *Graph isomorphism*. [Online; accessed 18-June-2025]. 2025. URL:
[92] Wu, J. et al. "Multi-view inter-modality representation with progressive fusion for image-text matching". In: *Neurocomputing* 535 (2023), pp. 1–12. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2023.02.043. URL:
[93] Xu, D. et al. *Scene Graph Generation by Iterative Message Passing*. 2017. arXiv: 1701.02426 [cs.CV]. URL:
[94] Xu, K. et al. "Representation Learning on Graphs with Jumping Knowledge Networks". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018, pp. 5469–5479.
[95] Xu, K. et al. "How Powerful Are Graph Neural Networks?" In: *International Conference on Learning Representations (ICLR)*. 2019. URL:
[96] Yang, J. et al. *Panoptic Scene Graph Generation*. 2022. arXiv: 2207.11247 [cs.CV]. URL:
[97] Yao, T. et al. *HIRI-ViT: Scaling Vision Transformer with High Resolution Inputs*. 2024. arXiv: 2403.11999 [cs.CV]. URL:
[98] Ying, R. et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*. KDD '18. ACM, July 2018, pp. 974–983. DOI: 10.1145/3219819.3219890. URL:
[99] Yoon, S. et al. *Image-to-Image Retrieval by Learning Similarity between Scene Graphs*. 2020. arXiv: 2012.14700 [cs.CV]. URL:
[100] Zhao, L. and Akoglu, L. *PairNorm: Tackling Oversmoothing in GNNs*. 2020. arXiv: 1909.12223 [cs.LG]. URL:

[101] Zhou, K. et al. *Dirichlet Energy Constrained Learning for Deep Graph Neural Networks*. 2021. arXiv: 2107.02392 [cs.LG]. URL:

[102] Zhu, G. et al. *Scene Graph Generation: A Comprehensive Survey*. 2022. arXiv: 2201.00443 [cs.CV]. URL: