Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Scene Graph Retrieval Using Contrastive Learning in Graph Neural Networks

Diploma Thesis

by

Boufalis Odyssefs Dimitrios

Επιβλέπων: Αθανάσιος Βουλόδημος
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

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

Scene Graph Retrieval Using Contrastive Learning in Graph Neural Networks

Diploma Thesis

by

Boufalis Odyssefs Dimitrios

Επιβλέπων: Αθανάσιος Βουλόδημος
Επ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Μαρτίου, 2024.

.....	.....	.....
Αθανάσιος Βουλόδημος	Γεώργιος Στάμου	Ανδρέας-Γεώργιος Σταφυλόπατης
Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π

Αθήνα, Μάρτιος 2024

.....

Μπουφαλης Οδυσσευς Δημητριος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα Νευρωνικά Δίκτυα Γράφων (ΝΔΓ) έχουν αναδειχθεί ως ένα μετασχηματιστικό παράδειγμα σε διάφορους τομείς λόγω της αξιοσημείωτης ικανότητάς τους να μοντελοποιούν πολύπλοκες σχέσεις που ενυπάρχουν σε δεδομένα με δομή γραφημάτων. Η δύναμη αναπαράστασης των ΝΔΓ επεκτείνεται σε διάφορους τομείς όπως η ανάλυση κοινωνικών δικτύων, η βιοπληροφορική, τα συστήματα συστάσεων και οι μοριακές επιστήμες μεταξύ άλλων. Παραδοσιακά, προκειμένου να αντιμετωπιστεί το γνωστό πρόβλημα της ομοιότητας γράφων, έχουν χρησιμοποιηθεί ευρέως αλγόριθμοι που προσεγγίζουν την απόσταση επεξεργασίας γράφων (GED) καθώς και πυρήνες γράφων. Πρόσφατα, η πρόοδος των τεχνικών βαθιάς μάθησης για δομημένα δεδομένα στη μορφή γράφων έδωσε την ευκαιρία να αναπτυχθούν νευρωνικές προσεγγίσεις βασισμένες σε ΝΔΓ για το πρόβλημα της ομοιότητας γράφων. Σε αυτό το πλαίσιο, τα ΝΔΓ έχουν αποδειχθεί ιδιαίτερα ισχυρά, αποδεικνύοντας την ικανότητα τους να συλλαμβάνουν περίπλοκα δομικά μοτίβα και σημασιολογικές σχέσεις μέσα σε γράφους.

Η παρούσα διπλωματική εργασία διερευνά τη δύναμη αναπαράστασης των νευρωνικών δικτύων γράφων (ΝΔΓ) που εκπαιδεύονται στο πλαίσιο της αντιθετικής μάθησης για την ανάκτηση γράφων σκηνής, μια εργασία ζωτικής σημασίας για την ολοκληρωμένη κατανόηση σκηνών που προέρχονται από εικόνες. Αξιοποιώντας τις δυνατότητες των ΝΔΓ στην αποτύπωση πολύπλοκων σχέσεων, η μελέτη χρησιμοποιεί καθιερωμένες τεχνικές μη επιβλεπόμενης αντιθετικής μάθησης για την παραγωγή υψηλής ποιότητας αναπαραστάσεων που διατηρούν τις αποστάσεις μεταξύ των γράφων. Επιπλέον, εισάγεται μια ασθενώς επιβλεπόμενη απώλεια αντιθετικής μάθησης για την περαιτέρω βελτίωση των μετρικών ανάκτησης αυτών των μοντέλων. Η βασική αλήθεια για την αξιολόγηση των μοντέλων καθορίζεται με τη χρήση προσεγγιστικών αλγορίθμων Graph Edit Distance (GED), με έμφαση στον αλγόριθμο διμερούς ταιριάσματος. Τα πειραματικά αποτελέσματα αναδεικνύουν την ανώτερη επίδοση των προτεινόμενων μοντέλων αντιθετικής μάθησης στην προσέγγιση της βασικής αλήθειας GED σε σύγκριση με τους γνωστούς πυρήνες γραφημάτων, επικυρώνοντας την αποτελεσματικότητα των αντιθετικών ΝΔΓ στην καταγραφή τόσο των λεπτών σχέσεων όσο και του σημασιολογικού περιεχομένου των γραφημάτων σκηνής. Δεδομένης της υπεροχής τους στην παραγωγή υψηλής ποιότητας αναπαραστάσεων, τα ΝΔΓ μπορούν στη συνέχεια να χρησιμοποιηθούν για την παροχή εξηγήσεων με αντιπαράδειγμα αξιοποιώντας την ικανότητά τους στην εργασία ανάκτησης γραφημάτων. Αυτά τα μοντέλα επιτρέπουν την εξαγωγή του πιο παρόμοιου γράφου σκηνής που ανήκει σε διαφορετική κλάση σε απάντηση σε ένα γράφο σκηνής ερωτήματος. Αυτή η ικανότητα τους χρησιμεύει ως ισχυρό εργαλείο για την επεξήγηση της διαφορετικής ταξινόμησης του σχετικού ζεύγους εικόνων από το οποίο έχουν παραχθεί οι γράφοι σκηνής. Αποκαλύπτοντας και αναδεικνύοντας τις λεπτές δομικές διαφορές εντός των γράφων σκηνής που οδηγούν στις διαφορετικές ταξινομήσεις, οι εξηγήσεις με αντιπαράδειγμα που βασίζονται σε ΝΔΓ προσφέρουν πολύτιμες πληροφορίες για τις διαδικασίες λήψης αποφάσεων ενός μοντέλου ταξινομητή , προωθώντας τη βαθύτερη κατανόηση των σημασιολογικών διαφορών μεταξύ των εικόνων και ενισχύοντας την ερμηνευσιμότητα των συστημάτων μηχανικής μάθησης.

Λέξεις Κλειδιά — Νευρωνικά Δίκτυα Γράφων, Αντιθετική Μάθηση, Ομοιότητα Γραφημάτων, Ανάκτηση Γραφημάτων, Γράφοι σκηνών, Εξηγήσεις με Αντιπαράδειγμα

Abstract

Graph Neural Networks (GNNs) have emerged as a transformative paradigm in various domains due to their remarkable ability to model complex relationships inherent in graph-structured data. The representation power of GNNs extends across diverse fields such as social network analysis, bioinformatics, recommendation systems and molecular sciences among others. Traditionally, in order to tackle the well known graph similarity problem, algorithms approximating Graph Edit Distance (GED) as well as Graph Kernels have been widely used. Recently, the advancement of deep learning techniques for graph-structured data has given rise to graph based neural approaches for the graph similarity problem. In this context, GNNs have been proven to be particularly potent, demonstrating the capability to capture intricate structural patterns and semantic relationships within graphs.

This diploma thesis delves into the representation power of Graph Neural Networks (GNNs) trained within the Contrastive Learning Framework for scene graph retrieval, a task pivotal for comprehensive scene understanding. Leveraging the capabilities of GNNs in capturing complex relationships, the study employs well-established unsupervised contrastive learning techniques to produce high quality and distance preserving graph embeddings. Additionally, a rank aware weak supervised contrastive learning loss is introduced to further enhance the retrieval metrics of these models. Ground truth for evaluation is established using approximate Graph Edit Distance (GED) algorithms, with a focus on the bipartite matching algorithm. The experimental results showcase the superior performance of the proposed contrastive learning models in approximating the GED ground truth compared to well known Graph Kernels, validating the effectiveness of Contrastive GNNs in capturing both subtle relationships and the semantic contents of scene graphs. Given their superiority in producing high-quality embeddings, GNNs can be then used to provide Counterfactual Explanations by leveraging their adeptness in graph retrieval tasks. These models enable the extraction of the most similar scene graph from another class in response to a query scene graph. This capability serves as a powerful tool for semantically explaining the differential classification of the underlying pair of images from which the scene graphs have been generated. By uncovering and highlighting the subtle structural nuances within the graphs that contribute to dissimilar classifications, GNN-based counterfactual explanations offer valuable insights into the decision-making processes of the model, promoting a deeper understanding of the semantic disparities between images and enhancing interpretability in machine learning systems.

Keywords — Graph Neural Networks, Contrastive Learning, Graph Similarity, Graph Retrieval, Scene Graphs, Counterfactual Explanations

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον κ. Βουλόδημο και τον κ.Στάμου για την ευκαιρία που μου δώσανε να εκπονήσω την διπλωματική μου εργασία στο AILS Lab καθώς και για τη σημαντική βοήθεια που μου παρείχαν κατά τη διάρκεια των αιτήσεων. Η ολοκλήρωση αυτής της διπλωματικής δεν θα ήταν εφικτή χωρίς την πολύτιμη βοήθεια της Αγγελικής, της Μαρίας και του Νίκου τους οποίους ευχαριστώ από καρδιάς. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για όλη την στήριξη που μου παρείχε όλα αυτά τα χρόνια για να εκπληρώσω τα όνειρα μου καθώς και τους φίλους μου για τις ευχάριστες στιγμές που περάσαμε μαζί.

Μπούφαλης Οδυσσέας, Μάρτιος 2024

Contents

Contents xi

List of Figures xvi

1 Εκτεταμένη Περίληψη στα Ελληνικά 1
- 1.1 Θεωρητικό υπόβαθρο . . . 2
  - 1.1.1 Θεωρία Γράφων . . . 2
  - 1.1.2 Απόσταση Επεξεργασίας Γράφων . . . 3
  - 1.1.3 Πυρήνες Γράφων . . . 4
  - 1.1.4 Γράφοι Σκηνής . . . 5
  - 1.1.5 Νευρωνικά Δίκτυα Γράφων . . . 6
  - 1.1.6 Αντιθετική Μάθηση για Νευρωνικά Δίκτυα Γράφων . . . 9
  - 1.1.7 Εξηγήσεις με Αντιπαράδειγμα . . . 10
- 1.2 Προτεινόμενες Προσεγγίσεις . . . 10
  - 1.2.1 Συνεισφορά . . . 10
  - 1.2.2 Προτεινόμενα Μοντέλα . . . 11
- 1.3 Πειράματικό Μέρος . . . 15
  - 1.3.1 Σύνολο Δεδομένων . . . 15
  - 1.3.2 Μετρικές αξιολόγησης . . . 19
  - 1.3.3 Βασική Αληθεία . . . 20
  - 1.3.4 Πυρήνες Γράφων . . . 23
  - 1.3.5 Νευρωνικά Δίκτυα Γράφων . . . 24
  - 1.3.6 Ποσοτικά Αποτελέσματα . . . 25
  - 1.3.7 Ποιοτικά Αποτέλεσματα . . . 30
- 1.4 Συμπεράσματα . . . 33
  - 1.4.1 Συζήτηση . . . 33
  - 1.4.2 Μελλοντικές Κατευθύνσεις . . . 34

2 Introduction 35

3 Background 37
- 3.1 Machine Learning . . . 38
  - 3.1.1 Input Data Types . . . 38
  - 3.1.2 Learning Categories . . . 39
- 3.2 Deep Learning . . . 41
  - 3.2.1 Basic Concepts . . . 42
  - 3.2.2 Deep Learning Models . . . 47

4 Contrastive Learning 53
- 4.1 Elements of Information Theory . . . 54
  - 4.1.1 Entropy . . . 54
  - 4.1.2 Kullback-Leibler (KL) Divergence . . . 54
  - 4.1.3 Jensen-Shannon Divergence (JSD) . . . 54

4.1.4 Mutual Information . . . 54
4.2 Training Objectives in Contrastive Learning . . . 55
4.2.1 Contrastive Loss . . . 55
4.2.2 Triplet Margin Loss . . . 55
4.2.3 Mutual Information Maximization Losses . . . 56

5 Graphs **61**
5.1 Graph Theory . . . 62
5.2 Graph Similarity . . . 65
5.2.1 Graph Edit Distance . . . 65
5.2.2 Graph Kernels . . . 66
5.3 Scene Graphs . . . 72
5.4 Related Work . . . 74

6 Graph Neural Networks (GNN) **75**
6.1 Machine Learning on Graphs . . . 76
6.1.1 Motivation . . . 76
6.1.2 Permutation Invariance and Equivariance . . . 76
6.2 Spectral Approaches . . . 78
6.2.1 Elements of Graph Spectral Theory . . . 78
6.2.2 Spectral Variants . . . 79
6.3 Spatial Approaches . . . 81
6.3.1 General Framework . . . 81
6.3.2 Spatial Variants . . . 82
6.4 How to Use Graph Neural Networks . . . 87
6.4.1 Exploring Task Types in Graph Neural Networks . . . 87
6.4.2 Diverse Training Approaches for Graph Neural Networks . . . 88
6.5 Contrastive Learning on Graphs . . . 88
6.5.1 Contrasting Modes . . . 88
6.5.2 Graph Contrastive Learning with Augmentations (GraphCL) . . . 89
6.5.3 InfoGraph . . . 91
6.5.4 Deep Graph InfoMax (DGI) . . . 92
6.5.5 Deep Graph Contrastive Representation learning (Grace) . . . 93

7 Counterfactual Explanations **95**
7.1 Introduction . . . 96
7.2 Conceptual Edits and Algorithmic Framework . . . 96
7.3 GNNs for Semantic Counterfactuals . . . 99
7.4 Related Work . . . 100

8 Proposal **101**
8.1 Contributions . . . 101
8.2 Proposed Models . . . 102

9 Experiments **105**
9.1 Preliminaries . . . 106
9.1.1 Dataset . . . 106
9.1.2 Evaluation Metrics . . . 110
9.1.3 Ground Truth . . . 112
9.2 Training Details . . . 115
9.2.1 Graph Kernels . . . 115
9.2.2 GNNs . . . 116
9.3 Results . . . 117
9.3.1 Quantitative Results . . . 117
9.3.2 Qualitative Results . . . 122

10 Conclusion **127**
10.1 Discussion 127
10.2 Future Work 128

11 Bibliography **129**

List of Figures

1.1.1 Παράδειγμα Επεξεργασίας ενός γραφήματος για την μετατροπή του σε ένα άλλο. [78] 4
1.1.2 Απεικόνιση της λειτουργίας των μεθόδων πυρήνα, ... 5
1.1.3 Ένα παράδειγμα περιοχών από μία εικόνα μαζί με τους αντίστιχους γράφους σκηνής από το Visual Genome Dataset [46]. 6
1.1.4 Το πλαίσιο μάθησης GraphCL [90]. 9
1.2.1 Διαδικασία εκπαίδευσης των αντιθετικών ΝΔΓ [96]. 11
1.2.2 Σχεδιασμός των παραλλαγών κωδικοποιητή GAT/GATv2 13
1.2.3 Σχεδιασμός των παραλλαγών κωδικοποιητών GCN/GIN 14
1.3.1 Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής όπου οι τύποι σχέσεων έχουν αφαιρεθεί 15
1.3.2 Στατιστικά στοιχεία για τα γραφήματα σκηνών στο τυχαίο σύνολο 17
1.3.3 Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής από το τυχαίο σύνολο. . . . 17
1.3.4 Στατιστικά στοιχεία για τα γραφήματα σκηνών στο πυκνό σύνολο 18
1.3.5 Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής από το πυκνό σύνολο. . . . 18
1.3.6 Μια απλή αναπαράσταση του τμήματος της ιεραρχίας ουσιαστικών του WordNet κάτω από τον κόμβο ρίζας της έννοιας "entity". 21
1.3.7 Παρουσιάζονται δύο εικόνες από το πυκνό σύνολο, καθεμία από τις οποίες παρουσιάζει ένα πανομοιότυπο γράφημα σκηνής. Ο προσεγγιστικός αλγόριθμος GED απέδωσε με ακρίβεια βαθμολογία απόστασης 0, αναγνωρίζοντας ότι μοιράζονται το ίδιο σημασιολογικό περιεχόμενο . . . 22
1.3.8 Εικόνα δύο δομικά όμοιων εικόνων που βαθμολογήθηκαν με σημαντική απόσταση από τον αλγόριθμο Bipartite Matching GED λόγω του διαφορετικού σημασιολογικού τους περιεχομένου. . 23
1.3.9 Εικόνα αναζήτησης 30
1.3.10Top 3 αποτελέσματα - Μοντέλο GNN 30
1.3.11Top 3 αποτελέσματα - Καλύτερος πυρήνας 30
1.3.12Εικόνα αναζήτησης 31
1.3.13Top 3 αποτελέσματα - Μοντέλο GNN 31
1.3.14Top 3 αποτελέσματα - Καλύτερος πυρήνας 31
1.3.15Εικόνα αναζήτησης 32
1.3.16Top 3 αποτελέσματα - Μοντέλο GNN 32
1.3.17Top 3 αποτελέσματα - Καλύτερος πυρήνας 32
1.3.18Εικόνα αναζήτησης 33
1.3.19Top 3 αποτελέσματα - Μοντέλο GNN 33
1.3.20Top 3 αποτελέσματα - Καλύτερος πυρήνας 33

3.1.1 An illustration of the relationship of the fields of AI, ML and DL. Source: StackExchange . . 38
3.1.2 An illustration of the three most important categories of learning algorithms [64]. 40
3.1.3 Visualization of Embedding points from the 3D space to 2D space using PCA 41
3.2.1 An illustration of how the chain rule is used during backpropagation to compute the gradients. Source: Mayank Agarwal 45
3.2.2 An illustration of the behavior of the error function according to the model's complexity including the locations where underfitting and overfitting are identified [7]. 46
3.2.3 Single Layer Perceptron with one hidden neuron [54]. 47
3.2.4 Multi-layer Perceptron with three hidden layers. Source: Towards Data Science 47

3.2.5 An Illustration of the diagrams of the Activation functions mentioned before 49
3.2.6 An illustration of the way a kernel (filter) convoles an image 50
3.2.7 An illustration of the architecture of LeNet [92]. 51
3.2.8 An illustration of the architecture of the Trasnformer model as presented in [82]. 52

4.2.1 An illustration of how triplet margin loss works [71] 55
4.2.2 An illustration of the CPC architecture [62] 57
4.2.3 An illustration of how SimCLR pulls closely the embeddings z_i and z_j of a positive pair[20] . 58
4.2.4 Maximizing mutual information between local features and global features.[39] 59

5.1.1 A Visual Representation of a General Graph 62
5.1.2 A Visual Representation of an Undirected and a Directed Graph 62
5.1.3 A Visual Representation of two graphs along with their adjacency matrices 63
5.1.4 An Undirected Graph along with its adjacency list 64
5.2.1 Graph Edit Distance Between Two Graphs. [78] 66
5.2.2 Illustration of Kernel Trick 68
5.2.3 Illustration of the label refining process [2] 69
5.2.4 Illustration of the construction of the product graph [61] 71
5.2.5 All graphlets of size 4 [61] 72
5.3.1 An example of regions of an image along with their corresponding scene graphs from the Visual Genome Dataset [46] 73
5.3.2 The Complete Visual Genome Scene Graph of the image depicted in Figure 5.3.1. It is evident that this scene graph contains different objects, attributes and relationships, defining effectively that semantic content of the corresponding image [46]. 73

6.1.1 View of the convolutional receptive field for images and graphs [11] 76
6.1.2 A typical GNN may contain permutation equivariant layers computing node-wise features and a permutation invariant global pooling layer [16] 78
6.2.1 Multi-layer Graph Convolutional Network (GCN) with first-order filters 80
6.3.1 **Left**: The self-attention mechanism employed by GAT. **Right**: An illustration of multihead attention (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote different attention heads. The aggregated features from each head are concatenated or averaged [83]. 83
6.3.2 In a complete bipartite graph we can observe that the standard GAT computes static attention – the ranking of attention coefficients is global for all nodes in the graph, and is unconditioned on the query node. Specifically in this case, all query nodes attend mostly to the 8th key (k8). In contrast, GATv2 actually computes dynamic attention, where every query has a different ranking of attention coefficients of the keys and every node attends the most with itself [14]. . 84
6.3.3 Visual illustration of the GraphSAGE sample and aggregate approach [36]. 85
6.3.4 Visual illustration of how GAE works [88]. 86
6.5.1 A general GCL framework which, after producing two augmented graph views, it encodes them using GNN layers. The three different contrastive modes including node-node, graph-graph and patch(node)-graph interactions, are illustrated using arrows of different colours [96]. . . . 89
6.5.2 The proposed framework for Graph Contrastive Learning where $q_i(|G)$ and $q_j(|G)$ are augmentations sampled from an augmentation pool T and applied to input graph G. A shared GNN-based encoder f(·) and a projection head g(·) are trained to maximize the agreement between representations z_i and z_j via a contrastive loss [90]. 91
6.5.3 Two graphs are encoded into their corresponding feature maps by graph convolutions and jumping concatenation. The discriminator takes a (global representation, patch representation) pair as input and decides whether they are from the same graph. InfoGraph uses a batch-wise fashion to generate all possible positive and negative samples in the batch. [77]. 92
6.5.4 An illustration of the most important steps of DGI [84]. 93
6.5.5 An illustration of the local-local CL conducted by GRACE [95]. 94

7.2.1 An illustration of the Conceptual Edits as Counterfactual Explanations Framework [31]. . . . 97

8.2.1 Training pipeline of Contrastive GNNs [96]. 102
8.2.2 Design of GAT/GATv2 encoder variants 103
8.2.3 Design of GCN/GIN encoder variants 104

9.1.1 An example of an image alongside with its corresponding scene graph where the types of relationships have been removed 106
9.1.2 Statistics for scene graphs in the Random Set 108
9.1.3 An example of an image alongside with its corresponding scene graph from the Random Set. 108
9.1.4 Statistics for scene graphs in the Dense Set 109
9.1.5 An example of an image alongside with its corresponding scene graph from the Dense Set. . . 109
9.1.6 A simple representation of the portion of the WordNet noun hierarchy below the concept root node “entity”. 113
9.1.7 Here are two images from the Dense Set, each featuring an identical scene graph. The approximate GED algorithm accurately assigned a distance score of 0, recognizing that they share the same semantical content 114
9.1.8 An illustration of two structurally similar images scored with a significant distance by the Bipartite Matching GED algorithm due to their distinct semantic content. 115
9.3.1 Query Image 122
9.3.2 Top 3 Results - GNN Model 122
9.3.3 Top 3 Results - Best Kernel 122
9.3.4 Query Image 123
9.3.5 Top 3 Results - GNN Model 123
9.3.6 Top 3 Results - Best Kernel 123
9.3.7 Query Image 124
9.3.8 Top 3 Results - GNN Model 124
9.3.9 Top 3 Results - Best Kernel 124
9.3.10 Query Image 125
9.3.11 Top 3 Results - GNN Model 125
9.3.12 Top 3 Results - Best Kernel 125

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό υπόβαθρο

Οι πρόσφατες εξελίξεις στην τεχνητή νοημοσύνη (ΤΝ) έχουν μεταμορφώσει διάφορους τομείς, καθιστώντας αναγκαία την εφαρμογή προηγμένων τεχνικών για την επεξεργασία πολύπλοκων δομών δεδομένων όπως οι γράφοι. Τα Νευρωνικά Δίκτυα Γράφων (ΝΔΓ) έχουν αναδειχθεί ως βασικά εργαλεία για τη μοντελοποίηση σχέσεων σε δεδομένα γραφημάτων, με εφαρμογές που καλύπτουν τα κοινωνικά δίκτυα, τα συστήματα συστάσεων και τη βιοπληροφορική. Τα παραδοσιακά νευρωνικά δίκτυα είναι ακατάλληλα για δεδομένα γράφων, γεγονός που αναδεικνύει την ανάγκη για εξειδικευμένα μοντέλα όπως τα ΝΔΓ, ικανά να αποτυπώνουν τις ιεραρχικές σχέσεις που ενυπάρχουν σε τέτοιου είδους δεδομένα.

Το επίκεντρο αυτής της διατριβής έγκειται στην αντιμετώπιση του προβλήματος της ομοιότητας γράφων, ειδικά για γράφους σκηνής, όπου το ζητούμενο είναι η κατάταξη των γράφων-απαντήσεων με βάση την ομοιότητά τους με έναν γράφο ερωτήματος. Ενώ οι πυρήνες γράφων είναι δημοφιλείς για το έργο αυτό λόγω της ευκολίας υλοποίησης και της υπολογιστικής τους απόδοσης, προτείνουμε τη χρήση ΝΔΓ, ιδιαίτερα Αντιθετικών ΝΔΓ, για την αντιμετώπιση αυτού του προβλήματος. Αξιοποιώντας τις δυνατότητες ποιοτικής αναπαράστασης των Αντιθετικών ΝΔΓ, στοχεύουμε στη βελτίωση της ακρίβειας και της αποδοτικότητας των υπολογισμών για το πρόβλημα της ομοιότητας γράφων.

Επιπλέον, η αυξανόμενη ζήτηση για Ερμηνεύσιμη Τεχνητή Νοημοσύνη (ΧΑΙ) έχει γίνει κρίσιμη, ιδίως σε τομείς όπως η υγειονομική περίθαλψη και τα αυτόνομα οχήματα, όπου τα συστήματα Τεχνητής Νοημοσύνης χρησιμοποιούνται εκτενώς. Οι Εξηγήσεις με Αντιπαραδείγματα έχουν αναδειχθεί ως μια πολλά υποσχόμενη μέθοδος για τη διαλεύκανση των συστημάτων ΤΝ μαύρου κουτιού. Με την ανάδειξη των ελάχιστων αλλαγών που απαιτούνται για τη μετάβαση σε ένα εναλλακτικό αποτέλεσμα, οι Εξήγησεις με Αντιπαραδείγματα με βάση τα ΝΔΓ ενισχύουν τη διαφάνεια και την κατανόηση των διαδικασιών λήψης αποφάσεων συστημάτων ΤΝ.

Στην παρούσα διατριβή, εμβαθύνουμε στη χρησιμότητα των αντιθετικών ΝΔΓ για παροχή Εξηγήσεων με Αντιπάραδειγματα. Διερευνούμε διεξοδικά τρία πλαίσια μάθησης αντιθετικών ΝΔΓ, όπως τα GraphCL, InfoGraph και Grace, παράλληλα με διάφορες παραλλαγές ΝΔΓ. Στη συνέχεια, αξιολογούμε τις εκφραστικές τους δυνατότητες χρησιμοποιώντας δεδομένα γράφων σκηνής. Η εφαρμογή των Αντιθετικών ΝΔΓ για την ομοιότητα και την ανάκτηση γράφων σκηνής συνδέεται στενά με το πεδίο της Ερμηνεύσιμης Τεχνητής Νοημοσύνης. Συγκεκριμένα, η ικανότητα των αντιθετικών ΝΔΓ να εντοπίζουν τους πιο παρόμοιους γράφους σκηνής και, κατά συνέπεια, τις λιγότερο σημασιολογικά απομακρυσμένες περιπτώσεις μέσα σε ένα σύνολο δεδομένων εικόνων που αναπαρίστανται ως γράφοι σκηνής, παίζει καθοριστικό ρόλο στη δημιουργία Εξηγήσεων με Αντιπάραδειγματα για ταξινομητές εικόνας.

1.1.1 Θεωρία Γράφων

Ένα από τα κύρια αντικείμενα μελέτης των διακριτών μαθηματικών είναι τα γραφήματα. Οι γράφοι είναι μαθηματικές δομές που χρησιμοποιούνται για τη μοντελοποίηση σχέσεων ανά ζεύγη μεταξύ αντικειμένων. Ένας γράφος, συμβολίζεται ως $G = (V, E)$, όπου V είναι ένα σύνολο αντικειμένων που ονομάζονται κορυφές (ή κόμβοι) και E είναι ένα σύνολο διακριτών μη ταξινομημένων ζευγών διακριτών κορυφών που ονομάζονται ακμές (επίσης ονομάζονται σύνδεσμοι). Στην πράξη, αυτό σημαίνει ότι κάθε ακμή "συνδέει" δύο διαφορετικές κορυφές. Στην ακμή {x,y}, οι κορυφές x και y ονομάζονται τα δύο άκρα της ακμής. Η ακμή λέγεται ότι ενώνει τις κορυφές x και y και ότι προσπίπτει στις κορυφές x και y. Μια κορυφή μπορεί να υπάρχει σε ένα γράφημα και να μην ανήκει σε ακμή. Σε αυτή την περίπτωση, αναφερόμαστε σε αυτή την κορυφή ως απομονωμένο κόμβο.

Ένα γράφημα μπορεί να ταξινομηθεί σε διάφορους τύπους με βάση τις ιδιότητές του. Οι γράφοι μπορούν να ταξινομηθούν με βάση την κατεύθυνση των ακμών τους. Αναλυτικότερα, μια ακμή που συνδέει τους κόμβους x και y χαρακτηρίζεται ως μη κατευθυνόμενη εάν και τα δύο διατεταγμένα ζεύγη (x, y) και (y, x) αποτελούν μέρος του συνόλου των ακμών, υποδεικνύοντας μια αμφίδρομη σύνδεση μεταξύ x και y. Από την άλλη πλευρά, η ακμή θεωρείται κατευθυνόμενη εάν υπάρχει μόνο ένα από αυτά τα ζεύγη, υποδεικνύοντας μια κατευθυνόμενη σύνδεση. Κατά συνέπεια, ένας γράφος χαρακτηρίζεται ως μη κατευθυνόμενος όταν όλες οι ακμές του είναι μη κατευθυνόμενες, ενώ χαρακτηρίζεται ως κατευθυνόμενος αν τουλάχιστον μία ακμή είναι κατευθυνόμενη. Μια εγγενής και πολύ σημαντική ιδιότητα των μη κατευθυνόμενων γραφημάτων είναι η συμμετρία που παρουσιάζει ο πίνακας γειτνίασής τους.

Τα γραφήματα μπορούν να αναπαρασταθούν με διάφορους τρόπους για να διευκολύνουν την ανάλυση και την ανάπτυξη αλγορίθμων. Η πιο συνηθισμένη αναπαράσταση είναι ο πίνακας γειτνίασης.

Πίνακας Γειτνίασης: Ένας γράφος τάξης N μπορεί να χαρακτηριστεί πλήρως από τον πίνακα γειτνίασης A, ο οποίος είναι ένας τετραγωνικός πίνακας διαστάσεων $N \times N$. Σε αυτόν τον πίνακα, τα μη μηδενικά στοιχεία υποδηλώνουν την ύπαρξη συνδέσμου μεταξύ κορυφών. Στην περίπτωση ενός απλού γραφήματος, ο A_{ij} παίρνει μόνο δύο τιμές, 0 και 1, όπου η τιμή 0 δηλώνει αποσύνδεση και η 1 δηλώνει σύνδεση μεταξύ των κορυφών i και j. Ειδικότερα, τα μη κατευθυνόμενα γραφήματα παρουσιάζουν συμμετρικό πίνακα γειτνίασης, κάτι που σημαίνει ότι ο A_{ij} είναι ίσος με τον A_{ji}.

Εκτός από τον πίνακα γειτνίασης, ένας γράφος μπορεί επίσης να διαθέτει χαρακτηριστικά που σχετίζονται με τους κόμβους ή/και τις ακμές του. Σε τέτοιες περιπτώσεις, κάθε κόμβος (ή ακμή) προσδιορίζεται από ένα διάνυσμα χαρακτηριστικών με διάσταση D, οδηγώντας σε έναν πίνακα χαρακτηριστικών κόμβων (ή ακμών) που συμβολίζεται ως X με διαστάσεις $N \times D$.

1.1.2 Απόσταση Επεξεργασίας Γράφων

Η Απόσταση Επεξεργασίας Γράφων (GED) είναι μια σημαντική μετρική η οποία μετρά την ομοιότητα μεταξύ δύο γραφημάτων μετρώντας τις πράξεις που απαιτούνται για τη μετατροπή ενός γραφήματος σε ένα άλλο. Οι Alberto Sanfeliu και King-Sun Fu εισήγαγαν τον πρώτο μαθηματικό φορμαλισμό για την απόσταση επεξεργασίας γράφων το 1983 [70].

Τυπικά, η GED μεταξύ των γραφημάτων G_1 και G_2 συμβολίζεται ως $GED(G_1, G_2)$. Η βασική έννοια της GED, όπως εισήχθη στο [70], περιλαμβάνει τον ορισμό ενός συνόλου από Λειτουργίες Επεξεργασίας Γράφων. Οι Λειτουργίες Επεξεργασίας Γράφων και το κόστος τους συμβολίζονται ως e_i και $c(e_i) \geq 0$ αντίστοιχα, όπου $P(G_1, G_2)$ συμβολίζει το σύνολο όλων των μονοπατιών επεξεργασίας που μετατρέπουν το G_1 σε G_2. Σε αυτό το πλαίσιο, η $GED(G_1, G_2)$ μπορεί να οριστεί ως εξής:

$$GED(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in P(G_1,G_2)} \sum_{i=1}^{k} c(e_i)$$

Το σύνολο των στοιχειωδών πράξεων επεξεργασίας γραφημάτων τυπικά περιλαμβάνει:

- **Εισαγωγή κορυφής**: Εισαγωγή μίας μόνο νέας επισημασμένης κορυφής σε ένα γράφημα
- **Διαγραφή κορυφής**: Αφαίρεση μιας μεμονωμένης (συχνά αποσυνδεδεμένης) κορυφής από ένα γράφημα
- **Αντικατάσταση κορυφής**: Αλλαγή της ετικέτας (ή του χρώματος) μιας δεδομένης κορυφής
- **Edge insertion**: Εισαγωγή νέας έγχρωμης ακμής μεταξύ ενός ζεύγους κορυφών
- **Διαγραφή ακμής**: Αφαίρεση μίας ακμής μεταξύ ενός ζεύγους κορυφών
- **Αντικατάσταση ακμής**: Αλλαγή της ετικέτας (ή του χρώματος) μιας δεδομένης ακμής

Οι ακριβείς μέθοδοι για τον υπολογισμό της Απόστασης Επεξεργασίας Γράφων μεταξύ δύο γραφημάτων περιλαμβάνουν γενικά τον εντοπισμό του μονοπατιού επεξεργασίας με το ελάχιστο κόστος. Οι προσεγγίσεις που χρησιμοποιούνται για τον υπολογισμό αυτό συνήθως περιλαμβάνουν είτε αναζητήσεις εύρεσης μονοπατιών είτε προσδιορισμό των συντομότερων μονοπατιών, κάνοντας χρήση του αλγορίθμου αναζήτησης A*. Γενικά, το πρόβλημα του υπολογισμού της Απόστασης Επεξεργασίας Γραφήματος είναι NP-Hard και είναι ακόμη και δύσκολο να προσεγγιστεί ανήκοντας στην κατηγορία πολυπλοκότητας APX-Hard.

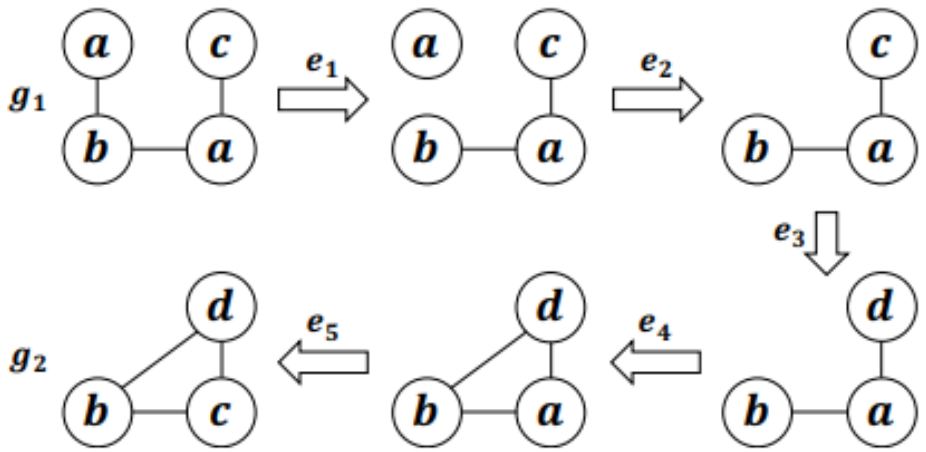


Figure 1.1.1: Παράδειγμα Επεξεργασίας ενός γραφήματος για την μετατροπή του σε ένα άλλο. [78]

Επειδή ο ακριβής υπολογισμός της GED είναι NP-Hard, έχουν αναπτυχθεί πολλές προσεγγιστικές προσεγγίσεις της απόστασης επεξεργασίας γραφήματος. Η συγκεκριμένη παραλλαγή που θα χρησιμοποιηθεί είναι ο αλγόριθμος Bipartite Matching. Πρόκειται για έναν προσεγγιστικό αλγόριθμο που εισήχθη από τον [28] ο οποίος οδηγεί σε μια υποβέλτιστη λύση στο πρόβλημα Graph Edit Distance (GED). Συγκεκριμένα, υπολογίζει αρχικά την ακριβή απόσταση επεξεργασίας λαμβάνοντας υπόψη μόνο τις λειτουργίες επεξεργασίας κόμβων, όπως εισαγωγές, διαγραφές και αντικαταστάσεις, και στη συνέχεια συμπεραίνει τις λειτουργίες επεξεργασίας ακμών. Αυτό οδηγεί σε μεγάλη επιτάχυνση, αλλά είναι επίσης ο λόγος για την υποβέλτιστη λύση.

1.1.3 Πυρήνες Γράφων

Στη μηχανική μάθηση, οι πυρήνες παίζουν θεμελιώδη ρόλο σε διάφορους αλγορίθμους στην αναγνώριση προτύπων και την εξόρυξη δεδομένων, ιδίως στο πλαίσιο των Support Vector Machines (SVM). Ένας πυρήνας είναι μια συνάρτηση που υπολογίζει την ομοιότητα μεταξύ ζευγών σημείων δεδομένων σε έναν χώρο υψηλών διαστάσεων, χωρίς να μετασχηματίζει ρητά τα δεδομένα στον εν λόγω χώρο. Αυτός ο σιωπηρός μετασχηματισμός των δεδομένων, επιτρέπει στους γραμμικούς αλγορίθμους (όπως οι γραμμικοί ταξινομητές) να αντιμετωπίζουν επιδέξια μη γραμμικά προβλήματα, αποτυπώνοντας αποτελεσματικά την ομοιότητα ή τη διαφορετικότητα μεταξύ σημείων δεδομένων και αποφεύγοντας τον υπολογιστικό φόρτο που απαιτείται για τη χαρτογράφηση των δεδομένων σε υψηλότερες διαστάσεις.

Η σημαντικότερη έννοια στη θεωρία πυρήνων ονομάζεται Kernel Trick. Πρόκειται για μια θεμελιώδη έννοια που επιτρέπει τη λειτουργία σε ένα χώρο χαρακτηριστικών υψηλών διαστάσεων, χωρίς να υπολογίζονται ποτέ οι συντεταγμένες των δεδομένων στον εν λόγω χώρο. Εάν ϕ είναι η συνάρτηση μετασχηματισμού, το τέχνασμα πυρήνα βασίζεται στον υπολογισμό της ποσότητας $K(\mathbf{x}_i, \mathbf{x}_j)$ Η συνάρτηση ϕ ικανοποιεί:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

όπου $\langle \cdot, \cdot \rangle$ είναι ένα κατάλληλο εσωτερικό γινόμενο. Η συνάρτηση $\phi : X \to V$ είναι η συνάρτηση που απεικονίζει τα αρχικά διανύσματα x_i και x_j στον χώρο υψηλότερων διαστάσεων όπου V είναι ένας χώρος Hilbert.

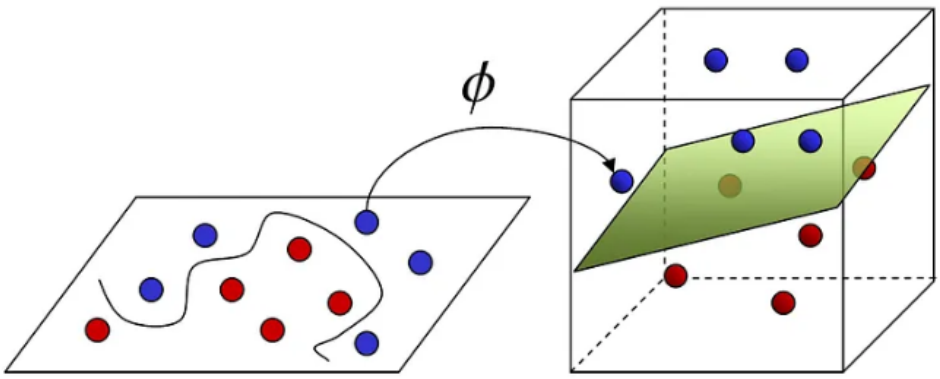


Figure 1.1.2: Απεικόνιση της λειτουργίας των μεθόδων πυρήνα,

Οι πυρήνες γραφημάτων είναι ένα ισχυρό εργαλείο στον τομέα της μηχανικής μάθησης, ειδικά σχεδιασμένο για την ανάλυση και τη σύγκριση δεδομένων που αναπαρίστανται ως γραφήματα. Αυτοί οι πυρήνες ποσοτικοποιούν την ομοιότητα μεταξύ γραφημάτων με βάση την ομοιότητα τους σε δομικό επίπεδο αλλά και με βάση τεχνικές διάδοσης μηνυμάτων (Message Passing). Όλοι αυτοί οι πυρήνες μπορούν να περιγραφούν στο πλαίσιο του γενικού ορισμού του πυρήνα ως εξής:

$$K(\mathbf{G}, \mathbf{G}^{'}) = \left\langle \phi(\mathbf{G}), \phi(\mathbf{G}^{'}) \right\rangle$$

όπου G και $G^{'}$ είναι δύο γραφήματα και $K : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{R}$. Στα πειράματά μας οι πυρήνες που θα χρησιμοποιηθούν είναι οι εξής:

- **Weisfeiler-Lehman Kernel (WL Kernel):** Ο πυρήνας WL Kernel είναι ένας πυρήνας γράφων που βασίζεται στο τεστ ισομορφισμού γράφων Weisfeiler-Lehman (WL). Λειτουργεί επαναληπτικά επανασημασιοδοτώντας κάθε κόμβο του γράφου με ένα πολυσύνολο ετικετών από τη γειτονιά του, ακολουθούμενο από μια πράξη κατακερματισμού για τη συμπίεση των νέων ετικετών. Η διαδικασία επαναλαμβάνεται για σταθερό αριθμό επαναλήψεων. Οι τελικές ετικέτες για κάθε κόμβο συγκρίνονται στη συνέχεια για τον υπολογισμό μιας βαθμολογίας ομοιότητας μεταξύ των γράφων.
- **Shortest Path Kernel:** Αυτός ο πυρήνας μετρά την ομοιότητα μεταξύ γραφημάτων με βάση τα συντομότερα μονοπάτια μεταξύ των κόμβων. Κατασκευάζει ένα διάνυσμα χαρακτηριστικών για κάθε γράφο, όπου κάθε χαρακτηριστικό αντιπροσωπεύει τη συχνότητα διαφορετικών μηκών συντομότερων μονοπατιών μεταξύ ζευγών κόμβων.
- **Random Walk Kernel:** Ο πυρήνας τυχαίου περιπάτου καταγράφει την ομοιότητα γραφημάτων προσομοιώνοντας τυχαίους περιπάτους σε γραφήματα και συγκρίνοντας την κατανομή αυτών των περιπάτων. Δημιουργούνται τυχαίοι περίπατοι σταθερού μήκους που ξεκινούν από κάθε κόμβο του γράφου.
- **Graphlet Sampling Kernel:** Ο πυρήνας Graphlet Sampling Kernel βασίζεται στη δειγματοληψία μικρών υπογραφημάτων (Graphlets) από τους αρχικούς γράφους εισόδου. Υπολογίζει μια διανυσματική αναπαράσταση κάθε γράφου μετρώντας τις εμφανίσεις διαφορετικών τύπων Graphlets.
- **Neighborhood Hash Kernel:** Ο πυρήνας Neighborhood Hash Kernel δουλεύει με αρκετά παρόμοιο τρόπο με τον Weisfeiler-Lehman Kernel, αλλά αντί για ετικέτες χρησιμοποιεί έναν διάνυσμα από δυαδικούς αριθμούς για κάθε κόμβο, και για την σύμπτυξη των ετικετών χρησιμοποιεί τις δυαδικές συναρτήσεις XOR και ROT.

1.1.4 Γράφοι Σκηνής

Στον τομέα της όρασης υπολογιστών και της κατανόησης οπτικών δεδομένων, οι γράφοι σκηνής αναδύθηκαν ως θεμελιώδεις δομές. Οι γράφοι σκηνής προτάθηκαν για πρώτη φορά στο [40] με σκοπό τη βελτίωση της ανάκτησης

εικόνων και έκτοτε έχουν προσελκύσει την προσοχή μεγάλου αριθμού ερευνητών. Στο πλαίσιο της παρούσας διατριβής, η δομή γράφου θα χρησιμοποιηθεί προκειμένου να αναπαραστήσει τη σκηνή που απεικονίζεται σε μια εικόνα. Αυτός ο τύπος αναπαράστασης, που ονομάζεται γράφος σκηνής, περιλαμβάνει πολύτιμες πληροφορίες σχετικά με τα αντικείμενα που υπάρχουν σε μια εικόνα καθώς και τις σχέσεις μεταξύ τους. Οι γράφοι σκηνής διαδραματίζουν ζωτικό ρόλο στην παροχή ενός δομημένου πλαισίου για την κατανόηση των σχέσεων μεταξύ των αντικειμένων και του περιβάλλοντός τους. Στην πράξη, αναπαρίστανται ως κατευθυνόμενοι γράφοι, όπου οι κόμβοι αναπαριστούν οντότητες (αντικείμενα), όπως αυτοκίνητα, ανθρώπους, κτίρια, μεταξύ άλλων, και οι ακμές αναπαριστούν τις σχέσεις τους χρησιμοποιώντας την ακόλουθη τριπλή δομή <υποκείμενο, σχέση, αντικείμενο>.

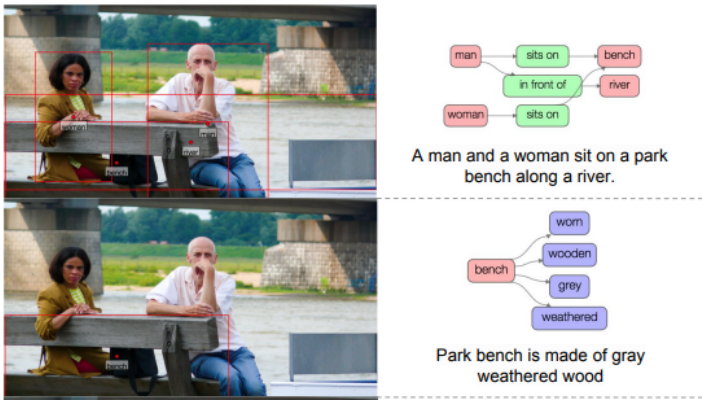


Figure 1.1.3: Ένα παράδειγμα περιοχών από μία εικόνα μαζί με τους αντίστιχους γράφους σκηνής από το Visual Genome Dataset [46].

Η αξιοποίηση των γραφημάτων σκηνής ενισχύει την κατανόηση εικόνων, καθώς αποτυπώνει το σημασιολογικό περιεχόμενο μιας εικόνας με πιο λεπτομερή τρόπο, καθώς και τις σχέσεις μεταξύ των οντοτήτων, κάτι που υπερβαίνει τις παραδοσιακές προσεγγίσεις ανίχνευσης και αναγνώρισης αντικειμένων. Το πιο σημαντικό σύνολο δεδομένων Γράφων Σκηνής που θα χρησιμοποιήσουμε στο πλαίσιο αυτής της διατριβής, είναι το Visual Genome Dataset που παρουσιάστηκε στην [46]. Αποτελείται από περισσότερες από 100.000 εικόνες και αποτελεί ένα από τα μεγαλύτερα σύνολα δεδομένων για εργασίες οπτικής κατανόησης. Κάθε εικόνα στο σύνολο δεδομένων είναι πυκνά σχολιασμένη, παρέχοντας πληθώρα πληροφοριών, συμπεριλαμβανομένων ετικετών αντικειμένων, σχέσεων αντικειμένων, περιγραφών περιοχών και σκηνών. Πηγαίνοντας πέρα από την απλή αναγνώριση αντικειμένων, το Visual Genome ενσωματώνει σχολιασμούς για τις σχέσεις αντικειμένων, περιγράφοντας λεπτομερώς πώς τα αντικείμενα μέσα σε μια σκηνή σχετίζονται μεταξύ τους, διευκολύνοντας τη σύλληψη ιεραρχικών δομών μέσα σε οπτικές σκηνές.

1.1.5 Νευρωνικά Δίκτυα Γράφων

Η έμπνευση για τα νευρωνικά δίκτυα γραφημάτων (ΝΔΓ) μπορεί να αποδοθεί στην αξιοσημείωτη επιτυχία των συνελικτικών νευρωνικών δικτύων (CNN) και στην καινοτόμο ιδέα των φίλτρων συνέλιξης που εφαρμόζουν σε εικόνες. Οι εικόνες μπορούν να θεωρηθούν ως μια ειδική περίπτωση δεδομένων με δομή γράφου, όπου οι κόμβοι είναι τα pixels και οι ακμές αντιπροσωπεύουν τη γειτνίαση μεταξύ τους. Τα νευρωνικά δίκτυα γράφων προέκυψαν ως φυσική εξέλιξη, προσαρμόζοντας το πλαίσιο συνελικτικής ανάλυσης σε γραφήματα, επιτρέποντας την εξαγωγή ουσιαστικών χαρακτηριστικών και σχέσεων από πολύπλοκα δίκτυα. Συγκεκριμένα, τα ΝΔΓ ορίζουν ένα διάνυσμα χαρακτηριστικών για κάθε έναν από τους κόμβους, συνήθως αρχικοποιημένο με εγγενείς ιδιότητες των κόμβων, οι οποίες στη συνέχεια μετασχηματίζονται από μια ακολουθία πράξεων. Η επέκταση του πλαισίου της συνέλιξης σε γράφους βασίζεται στην παρατήρηση ότι μπορούμε να θεωρήσουμε τη συνέλιξη σε γράφους για έναν συγκεκριμένο κόμβο i, ως συλλογή πληροφοριών από τους γειτονικούς κόμβους, περνώντας μηνύματα από τους γείτονες στον κόμβο i.

Ο λαπλασιανός πίνακας L ενός γράφου μπορεί να γραφτεί ως $L = U\Lambda U^T$ όπου τα ιδιοδιάνυσματα U σχηματίζουν έναν ορθοκανονικό χώρο. Έτσι, ο μετασχηματισμός Fourier γράφου ενός σήματος x και ο αντίστροφος μετασχηματισμός Fourier γράφου, μπορούν να οριστούν [12], [63], [26] ως:

$$\hat{x} = U^T x, x = U\hat{x}$$

όπου $\hat{x}$ αντιπροσωπεύει τον μετασχηματισμό Fourier γράφου του σήματος x. Είναι προφανές ότι ο μετασχηματισμός Fourier γράφου προβάλλει το σήμα γράφου εισόδου στον ορθοκανονικό χώρο που ορίζεται από τα ιδιοδιανύσματα της κανονικοποιημένης λαπλασιανής. Το θεώρημα της συνέλιξης δηλώνει ότι ο μετασχηματισμός Fourier μιας συνέλιξης μεταξύ δύο σημάτων είναι ισοδύναμος με τον σημειακό πολλαπλασιασμό των μετασχηματισμών Fourier τους. Δεδομένου αυτού, του μετασχηματισμού Fourier γράφου, ενός σήματος γράφου x και ενός φίλτρου $g \in R^n$ η πράξη συνέλιξης γράφου ορίζεται ως εξής:

$$x * g = U((U^T g) \odot (U^T x))$$

Συμβολίζοντας $g_\theta(\Lambda) = diag(U^T g)$, έχουμε:

$$x * g_\theta = U g_\theta U^T x$$

όπου $g_\theta = diag(\theta)$ είναι ο διαγώνιος πίνακας που αντιστοιχεί στους συντελεστές του φασματικού φίλτρου. Αυτό το φασματικό φίλτρο έχει μεγάλη σημασία, καθώς οι διάφορες εκδοχές Συνελικτικών Δικτύων Γράφων διαφέρουν με βάση την επιλογή αυτού του φίλτρου.

Οι βασικές παραλλαγές Συνελικτικών Νευρωνικών Δικτύων Γράφων που θα χρησιμοποιήσουμε σε αυτή τη διπλωματική είναι οι εξής:

- Το GCN [44] ήταν από τα πρώτα ΝΔΓ που γεφύρωσαν τις φασματικές και χωρικές προσεγγίσεις. Το προτεινόμενο φίλτρο συμπυκνώνει ουσιαστικά τη γειτονιά 1-βήματος ενός κόμβου και η διαδοχική εφαρμογή φίλτρων αυτής της μορφής συμπυκνώνει τη γειτονιά k-οστής τάξης ενός κόμβου. Ο προτεινόμενος τελεστής συνέλιξης έχει την εξής μορφή:

  $$X' = x * g_\theta \approx \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

  όπου $\tilde{A} = A + I_N$ και $\tilde{D} = \sum_j \tilde{A}_{ij}$ είναι μια κανονικοποίηση έτσι ώστε οι τιμές της κλίσης να μην γίνονται πολύ μεγάλες ούτε πολύ μικρές με τη στοίβαξη πολλών επιπέδων. Επίσης, $X \in R^{N \times C}$, $\Theta \in R^{C \times F}$, $Z \in R^{N \times F}$, C είναι η διάσταση των διανυσμάτων των κόμβων εισόδου και F είναι η διάσταση των διανυσμάτων των κόμβων στην έξοδο. Μετά την εφαρμογή του συνελικτικού τελεστή, μια μη γραμμική συνάρτηση ενεργοποίησης (π.χ. ReLU) εφαρμόζεται συνήθως στον πίνακα εξόδου X. Συνοψίζοντας, ο κανόνας διάδοσης που διέπει ένα βαθύ νευρωνικό δίκτυο με στρώματα GCN είναι:

  $$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

  όπου $W^{(l)}$ είναι οι εκπαιδεύσιμες παράμετροι του l-οστού στρώματος, $H^{(l)}$ είναι ο πίνακας που αναπαριστά τα χαρακτηριστικά των κόμβων όπως υπολογίζονται μέχρι το στρώμα l και σ είναι μια μη γραμμική συνάρτηση ενεργοποίησης.

  Η εξίσωση που διέπει το στρώμα GCN από την χωρική σκοπία είναι η εξης:

  $$x_i' = \Theta^T \sum_{j \in N(i) \cup i} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} x_j$$

  Αυτή η εξίσωση αντιπροσωπεύει ότι το ενημερωμένο διάνυσμα για τον κόμβο i υπολογίζεται αν αθροίσουμε τα κανονικοποιημένα διανύσματα κόμβων όλων των γειτονικών του κόμβων (συμπεριλαμβανομένου και του ίδιου του κόμβου) και πολλαπλασιάσουμε το αποτέλεσμα με έναν πίνακα εκπαιδεύσιμων παραμέτρων. Το μοντέλο αυτό αποδείχθηκε πολύ αποδοτικό σε μια μεγάλη ποικιλία εργασιών και αποτελεί ίσως μια από τις σημαντικότερες ανακαλύψεις στον τομέα των GNNs εμπνέοντας πολλές άλλες σημαντικές παραλλαγές.

- Το Graph Isomorphism Network (GIN) [89] προσπάθησε να απαντήσει στο ακόλουθο ερώτημα: πότε τα νευρωνικά δίκτυα γράφων έχουν την ίδια διακριτική ικανότητα με το τεστ Weisfeiler-Lehman (WL);

Συγκεκριμένα, θέλουμε ένα ΝΔΓ να αντιστοιχίζει διαφορετικούς γράφους σε διαφορετικές αναπαραστάσεις, εάν το τεστ WL αποφασίσει ότι δεν είναι ισομορφικοί. Για το λόγο αυτό, το GIN χρησιμοποιεί τον ακόλουθο κανόνα ενημέρωσης για τις ενθέσεις κόμβων:

$$h_v^l = MLP^l((1+\epsilon^l)h_v^{l-1} + \sum_{u \in N(v)} h_u^{l-1})$$

Προκειμένου να προκύψουν αποτελέσματα σε επίπεδο γραφήματος, οι συγγραφείς προτείνουν να χρησιμοποιηθούν οι ενσωματώσεις κόμβων από όλα τα βάθη του νευρωνικού δικτύου με τον ακόλουθο τρόπο:

$$h_G = \|_{k=0}^{K} (READOUT(\{h_u^k | u \in G\} | k = 0, 1, ..., K)$$

όπου η συνάρτηση $READOUT$ επιλέγεται να είναι το άθροισμα των ενσωματώσεων των κόμβων κάθε επιπέδου του δικτύου. Με απλά λόγια, προκειμένου να αποκτήσουμε μια αρκετά εκφραστική αναπαράσταση γράφου, αθροίζουμε τις ενσωματώσεις κόμβων όλων των επιπέδων ανεξάρτητα και τέλος τις συνενώνουμε για να λάβουμε το τελικό διάνυσμα ενσωμάτωσης γράφου.

- Το Graph Attention Network (GAT) [83] πρότεινε την υιοθέτηση της ιδέας του μηχανισμού της προσοχής που παρουσιάζεται στο [82]. Συγκεκριμένα, το [83] προτείνει τον συνδυασμό μηνυμάτων από γειτονικούς κόμβους με σταθμισμένο τρόπο. Τα βάρη ορίζονται ως εκπαιδεύσιμες παράμετροι του νευρωνικού δικτύου. Η διαίσθηση πίσω από το GAT είναι ότι με τη στάθμιση των εισερχόμενων μηνυμάτων από τους γείτονες, μπορεί να αντιληφθεί ότι τα μηνύματα από ορισμένους γείτονες μπορεί να είναι πιο σημαντικά από τα μηνύματα άλλων γειτόνων.

  Η ενημερωμένη κατάσταση του κόμβου i προκύπτει τώρα από την ακόλουθη εξίσωση:

  $$h_i^{l+1} = f(\sum_{j \in N(i)} \alpha_{ij} W h_j^l)$$

  όπου f είναι μια μη γραμμικότητα που εφαρμόζεται στη σταθμισμένη άθροιση των γειτονικών κόμβων.

  Τα εκπαιδευόμενα βάρη προσοχής υπολογίζονται ως εξής:

  $$\alpha_{ij} = softmax(LeakyReLU(\alpha^T [Wh_i^l || Wh_j^l]))$$

  όπου η συνάρτηση softmax χρησιμοποιείται για να διασφαλίσει ότι τα βάρη προσοχής αθροίζουν στη μονάδα και || είναι η πράξη συνένωσης. Αυτός ο μηχανισμός αυτοπροσοχής (self-attention) που περιγράφεται παραπάνω, μπορεί να επεκταθεί με την εισαγωγή πολλαπλών κεφαλών προσοχής, όπου K ανεξάρτητοι μηχανισμοί αυτοπροσοχής εφαρμόζονται για τον υπολογισμό των κρυφών καταστάσεων των κόμβων. Οι τελικές ενθέσεις των κόμβων μπορούν στη συνέχεια να υπολογιστούν είτε με τη συνένωση των K διαφορετικών διανυσμάτων που παράγονται από τους διαφορετικούς μηχανισμούς προσοχής είτε με τον υπολογισμό του μέσου όρου τους.

- Το GATv2 που προτάθηκε στο [14] απέδειξε ότι το αρχικό μοντέλο GAT [83] υπολογίζει static attention μεταξύ των κόμβων του γράφου. Το πρόβλημα που παρουσιάζει το τυπικό επίπεδο GAT, είναι ότι η συνάρτηση προσοχής ορίζει μια σταθερή κατάταξη των κόμβων, χωρίς αυτή να εξαρτάται από τον κόμβο i του ερωτήματος κάθε φορά. Στην πράξη, αυτό σημαίνει ότι υπάρχει ένας κόμβος v στο γράφημα, στον οποίο όλοι οι υπόλοιποι κόμβοι αποδίδουν το υψηλότερο σκορ προσοχής και αυτό αποδεικνύεται αναλυτικά στο [14]. Η τροποποιήμενη εκδοχή για τον υπολογισμό των βαρών προσοχής που λύνει το πρόβλημα αυτό είναι η εξής:

  $$e_{ij} = \alpha^T (LeakyReLU([Wh_i^l || Wh_j^l]))$$

  τα οποία στη συνέχεια περνούν από ένα στρώμα softmax για να παραχθούν τα κανονικοποιημένα βάρη προσοχής α_{ij}.

1.1.6 Αντιθετική Μάθηση για Νευρωνικά Δίκτυα Γράφων

Η αντιθετική μάθηση έχει αναδειχθεί ως μια πολλά υποσχόμενη μέθοδος για την εκμάθηση αναπαραστάσεων σε εργασίες μηχανικής μάθησης. Αρχικά εφαρμόστηκε σε δεδομένα εικόνας και κειμένου με στόχο την εκμάθηση εύρωστων αναπαραστάσεων με την αντιπαραβολή θετικών με αρνητικά δείγματα. Αυτή η μεθοδολογία έχει επιδείξει αξιοσημείωτη επιτυχία στη σύλληψη περίπλοκων μοτίβων και σημασιολογίας στα δεδομένα. Η επέκταση των αρχών της αντιθετικής μάθησης σε δεδομένα με δομή γράφου έχει γίνει ενεργός τομέας έρευνας τα τελευταία χρόνια. Σε αυτή την ενότητα παρουσιάζουμε τις σημαντικότερες προσεγγίσεις που χρησιμοποιήθηκαν στο πλαίσιο της παρούσας διατριβής.

Ένα βασικό στοιχείο για την αντιθετική μάθηση είναι ο καθορισμός της κλίμακας [96] στην οποία θα χρησιμοποιηθεί. Τρεις ευρέως χρησιμοποιούμενοι τρόποι αντιπαραβολής θετικών και αρνητικών δειγμάτων είναι οι local-local και global-global CL, οι οποίοι αντιπαραβάλλουν τις αναπαραστάσεις από ίδιες κλίμακες (κόμβοι-κόμβοι, γράφοι-γράφοι), και οι global-local CL, οι οποίοι αντιπαραβάλλουν τις αναπαραστάσεις από διαφορετικές κλίμακες (κόμβοι-γράφοι) [96]. Έτσι, κόμβοι (ή γράφοι) που αποτελούν θετικά δείγματα μεταξύ τους άρα "μοιάζουν" περισσότερο αποκτούν κοντινότερες αναπαραστάσεις ενώ οι αναπαραστάσεις αρνητικών δειγμάτων αντιστοιχούν σε μακρινότερα σημεία στον πολυδιάστατο χώρο.

- Το GraphCL [90] ήταν μία από τις πρώτες προσεγγίσεις αντιθετικής μάθησης με επαυξήσεις για την προεκπαίδευση ΝΔΓ. Ένα βασικό στοιχείο της αντιθετικής μάθησης είναι οι επαυξήσεις δεδομένων, οι οποίες δεν είχαν διερευνηθεί επαρκώς για δεδομένα γράφων μέχρι τότε. Βασικές μέθοδοι επαύξησης γράφων περιλαμβάνουν την αφαίρεση κόμβων, ακμών και χαρακτηριστικών των κόμβων. Διαισθητικά, η υπόθεση που γίνεται είναι ότι η έλλειψη μερικών χαρακτηριστικών του γράφου θα πρέπει να έχει ελάχιστο αντίκτυπο στις προβλέψεις του μοντέλου. Αυτές οι τεχνικές επαύξησης δεδομένων χρησιμοποιούνται για την παραγωγή συσχετισμένων δειγμάτων (θετικά δείγματα), τα οποία αναγκάζονται να είναι πιο "κοντά" στο χώρο στον οποίο αναπαριστούμε τους γράφους ή τους κόμβους. Ο στόχος αυτός επιτυγχάνεται με τη χρήση του InfoNCE (ή παρομοίως NT-Xent) loss. Συγκεκριμένα, ένα ΝΔΓ με ένα MLP χρησιμοποιούνται για τον υπολογισμό των αναπαραστάσεων που χρησιμοποιούνται στον υπολογισμό της συνάρτησης σφάλματος.

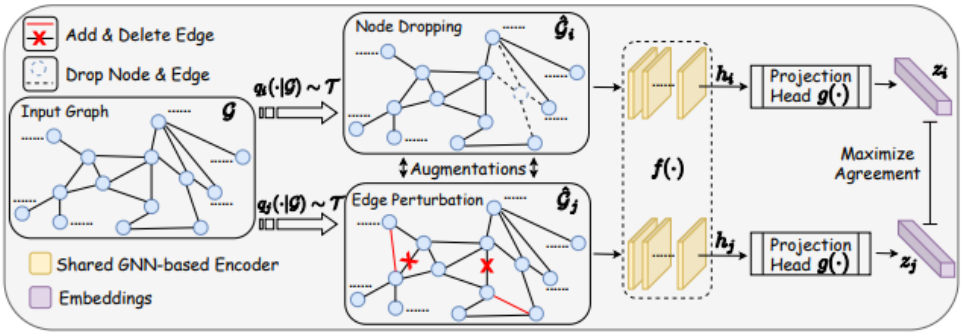


Figure 1.1.4: Το πλαίσιο μάθησης GraphCL [90].

- Το InfoGraph προτάθηκε στο [77] και σε αντίθεση με το GraphCL που χρησιμοποιεί μια global-global αντιπαραβολή θετικών και αρνητικών δειγμάτων, χρησιμοποιεί local-global αλληλεπιδράσεις. Προκειμένου να προκύψουν αναπαραστάσεις σε επίπεδο γραφήματος, οι συγγραφείς προτείνουν τη μεγιστοποίηση της αμοιβαίας πληροφορίας μεταξύ αναπαραστάσεων σε επίπεδο γραφήματος και αναπαραστάσεων σε επίπεδο κόμβων, ακολουθώντας το παράδειγμα του Deep InfoMax [39]. Έτσι, το ΝΔΓ που χρησιμοποιείται, μαθαίνει να παράγει και να ξεχωρίζει ποιές αναπαραστάσεις κόμβων ανήκουν σε κάθε γράφο με την χρήση του Jensen Shannon Mutual Information Estimator. Συνεπώς, αυτή η προσέγγιση δεν περιλαμβάνει την χρήση επαυξήσεων για την παραγωγή συσχετισμένων δειγμάτων.
- Η προσέγγιση Grace, όπως παρουσιάστηκε στο [95], περιλαμβάνει τη χρησιμοποίηση ενός αντιθετικού στόχου σε επίπεδο κόμβων, όπου δημιουργούνται δύο συσχετισμένα δείγματα γράφων με την εκτέλεση διαφόρων επαυξήσεων στα αρχικά δεδομένα όπως στο GraphCL. Η διαδικασία μάθησης επικεντρώνεται στη μεγιστοποίηση της συμφωνίας των αναπαραστάσεων των κόμβων εντός των συσχετισμένων θετικών

δειγμάτων. Σε αντίθεση με τη σύγκριση των ενσωματώσεων σε επίπεδο κόμβων με τις ενσωματώσεις σε επίπεδο γράφου, όπως στο InfoGraph, και σε αντίθεση με το GraphCL που συγκρίνει ενσωματώσεις σε επίπεδο γραφήματος, η έμφαση στο GRACE δίνεται στην αντιπαραβολή ενσωματώσεων ειδικά σε επίπεδο κόμβων. Έτσι, ο τρόπος αντιπαραβολής που χρησιμοποιείται, ανήκει στην κατηγορία των τοπικών-τοπικών CL [96]. Η μεθοδολογία περιλαμβάνει τη δημιουργία δύο επαυξημένων γραφημάτων μέσω της τυχαίας αλλοίωσης της τοπολογίας και των χαρακτηριστικών των κόμβων του αρχικού γραφήματος. Στη συνέχεια, το μοντέλο εκπαιδεύεται χρησιμοποιώντας το InfoNCE / NT-Xent loss για την ενίσχυση της συμφωνίας μεταξύ των αντίστοιχων ενσωματώσεων κόμβων σε αυτές τις όψεις. Οι τεχνικές επαύξησης που χρησιμοποιούνται περιλαμβάνουν την αφαίρεση ακμών και την απόκρυψη χαρακτηριστικών, αλλά δεν πραγματοποιείται αφαίρεση κόμβων προκειμένου να διατηρηθεί η αντιστοίχιση μεταξύ των ίδιων κόμβων στα θετικά συσχετισμένα δείγματα.

1.1.7 Εξηγήσεις με Αντιπαράδειγμα

Οι Εξηγήσεις με Αντιπαράδειγμα είναι μια έννοια που χρησιμοποιείται συνήθως στον τομέα της ερμηνεύσιμης τεχνητής νοημοσύνης (XAI) για να βοηθήσει τους χρήστες να κατανοήσουν τη λογική πίσω από τις προβλέψεις ενός ταξινομητή. Η βασική ιδέα είναι η παροχή εξήγησης με την εξέταση εναλλακτικών σεναρίων. Συγκεκριμένα, αυτά τα εναλλακτικά σενάρια σχετίζονται με ερωτήσεις όπως "Τι πρέπει να αλλάξει για να ταξινομηθεί ένα δείγμα ως X αντί για Υ;" ή "Τι θα έπρεπε να αλλάξει για να λάβει το μοντέλο μια διαφορετική απόφαση". Με αυτόν τον τρόπο, μπορούν να παράσχουν πολύτιμες πληροφορίες για τη διαδικασία λήψης αποφάσεων των ταξινομητών. Ο στόχος είναι να εξηγηθεί γιατί έγινε μια συγκεκριμένη πρόβλεψη διερευνώντας ποιες αλλαγές στα χαρακτηριστικά εισόδου θα οδηγούσαν σε διαφορετικό αποτέλεσμα ταξινόμησης.

Συγκεκριμένα, το πλαίσιο που παρουσιάζεται προτάθηκε στο [31] και βασίζεται στην παραγωγή Εξηγήσεων με Αντιπαραδειγμά με την βοήθεια Εννοιολογικών Επεξεργασιών. Για να λειτουργήσει ο προτεινόμενος αλγόριθμος πρέπει κάθε δείγμα να συνοδεύεται από ένα σύνολο Εννοιών. Έννοιες ονομάζουμε τις γενικές αναπαραστάσεις των αντικειμένων που υπάρχουν στα δεδομένα εισόδου και συνδέονται μεταξύ τους στα πλαίσια της ιεραρχίας WordNet [59]. Χρησιμοποιώντας την βάση γνώσης WordNet, υπολογίζεται στον μη κατευθυνόμενο γράφο TBox η απόσταση εννοιών μεταξύ όλων των ζευγών των παρόντων εννοιών με την χρήση του αλγόριθμου του Dijkstra. Η απόσταση εννοιών γενικεύεται στη συνέχεια σε απόσταση μεταξύ συνόλων εννοιών με την χρήση του αλγόριθμου του Karp. Επειδή κάθε σύνολο εννοιών αντιπροσωπεύει πρακτικά ένα δείγμα, ο αλγόριθμος τώρα μπορεί να υπολογίσει την εννοιολογική απόσταση μεταξύ κάθε ζεύγους δειγμάτων και στη συνέχεια να κατασκευάσει ένα γράφο με κόστος στις ακμές αυτές τις αποστάσεις. Στη συνέχεια, με την χρήση πάλι του αλγόριθμου του Dijkstra, μπορούν να υπολογιστούν τα ελάχιστα μονοπάτια στον νέο γράφο και έτσι πλέον για κάθε δείγμα x_i που ταξινομήθηκε στην κλάση C_i , το Αντιπαράδειγμα θα είναι το δείγμα που έχει την πιο κοντινή εννοιολογική απόσταση στο x_i αλλά δεν ταξινομήθηκε στην κλάση C_i.

Εύκολα μπορεί να δει κανείς ότι σε αυτά τα πλαίσια μπορούμε να αξιοποιήσουμε τα ΝΔΓ για να κατατάξουμε γράφους σκηνής που αντιστοιχούν σε εικόνες και να βρούμε τους πιο παρόμοιους για κάθε έναν. Στη συνέχεια, μπορούμε να βρούμε από τους πιο παρόμοιους εκείνον που ταξινομείται από έναν ταξινομητή εικόνων σε διαφορετική κλάση και να υπολογίσουμε, μόνο μεταξύ αυτού του ζεύγους γράφων σκηνών που προέκυψε, το Graph Edit Distance, παρέχοντας έτσι την αντίστοιχη εξήγηση με αντιπαράδειγμα που θα οδηγούσε σε διαφορετική ταξινόμηση.

1.2 Προτεινόμενες Προσεγγίσεις

1.2.1 Συνεισφορά

Παρακάτω, συνοψίζονται οι βασικές συνεισφορές της παρούσας διατριβής:

- Χρησιμοποιούμε διάφορες προσεγγίσεις Αντιθετικών ΝΔΓ για να διερευνήσουμε την Ομοιότητα Γράφων Σκηνής. Εξ όσων γνωρίζουμε, έχει υπάρξει περιορισμένο ακαδημαϊκό ενδιαφέρον στη χρήση Αντιθετικών ΝΔΓ για την αντιμετώπιση αυτού του ζητήματος. Ως εκ τούτου, στόχος μας είναι να παράσχουμε μια ολοκληρωμένη επισκόπηση του προβλήματος, των μεθοδολογιών που χρησιμοποιούμε για την επίλυσή του και να προσφέρουμε μια πρόσθετη οπτική μέσω της χρήσης των αντιθετικών μεθόδων, συμπληρώνοντας τις υπάρχουσες προσεγγίσεις στα [24] και [25].

- Με τη χρήση αντιθετικών προσεγγίσεων, μπορούμε να εκπαιδεύσουμε τα μοντέλα με μη επιβλεπόμενο τρόπο, εξαλείφοντας την ανάγκη υπολογισμού της απόστασης Graph Edit Distance μεταξύ όλων των γράφων. Έτσι, μειώνεται το πλήθος δειγμάτων από $O(n^2)$ που είναι απαραίτητο στα μοντέλα με επίβλεψη σε O(n).

- Μελετάμε διεξοδικά τις επιδόσεις διαφόρων παραλλαγών συνελικτικών ΝΔΓ ως δομικά στοιχεία στις επιλεγμένες αντιθετικές προσεγγίσεις για την ανάκτηση γραφημάτων σκηνών. Παρατηρούμε ότι η αντιθετική προεκπαίδευση, είτε σε πυκνούς είτε σε αραιούς τυχαίους γράφους, αντιπροσωπεύουν εξίσου ανταγωνιστικές προσεγγίσεις. Επιπλέον, προτείνουμε ένα Rank Aware Fine-tuning για μια επιλεγμένη ομάδα γραφημάτων σκηνής ως περαιτέρω βελτιστοποίηση των αρχικών μοντέλων. Αυτό, σε συνδυασμό με τη χρήση ενός μικρού ποσοστού επίβλεψης, ενισχύει τις δυνατότητες των μοντέλων. Επιπλέον, αξιολογούμε τον αντίκτυπο αυτών των διαφορετικών αρχιτεκτονικών αξιολογώντας τόσο ποσοτικά όσο και ποιοτικά τα αποτελέσματα μας.

- Τα μοντέλα ΝΔΓ που προτείνουμε παράγουν βαθμολογίες ομοιότητας για όλους τους Γράφους Σκηνής, επιτρέποντας την ενσωμάτωσή τους σε ένα πλαίσιο παροχής εξηγήσεων με χρήση αντιπαραδειγμάτων, παρόμοιο με αυτό που περιγράφεται στο κεφάλαιο 7.

1.2.2 Προτεινόμενα Μοντέλα

Τα μοντέλα ΝΔΓ με τα οποία πειραματιζόμαστε βασίζονται σε προσεγγίσεις αντιθετικής μάθησης ΝΔΓ που εφαρμόζονται σε διάφορα επίπεδα κλίμακας εντός του γράφου. Συγκεκριμένα, χρησιμοποιούμε τα GraphCL, InfoGraph και Grace, καθένα από τα οποία υπολογίζει την αντιθετική απώλεια στα επίπεδα Graph-Graph, Graph-Node και Node-Node, αντίστοιχα, όπως παρουσιάζεται στο κεφάλαιο 6. Αυτό καθορίζει ρητά τον τρόπο εκπαίδευσης και αξιολόγησης των μοντέλων.

- **Εκπαίδευση:** Τα Αντιθετικά ΝΔΓ λαμβάνουν παρτίδες γράφων ως είσοδο, με πρωταρχικό στόχο τη δημιουργία ουσιαστικών αναπαραστάσεων που αποτυπώνουν την υποκείμενη δομή και τις σχέσεις εντός του γράφου. Συγκεκριμένα, αυτό επιτυγχάνεται αξιοποιώντας την ιδέα ότι παρόμοιοι κόμβοι/γράφοι πρέπει να έχουν αντίστοιχα παρόμοιες αναπαραστάσεις. Κατά συνέπεια, σε αυτό το πλαίσιο εκπαίδευσης, κάθε σύνολο δεδομένων από N γράφους παρέχει N δείγματα για εκπαίδευση. Αντίθετα, οι προσεγγίσεις με επίβλεψη για την ομοιότητα γραφημάτων χρησιμοποιούν ζεύγη γραφημάτων από το αρχικό σύνολο δεδομένων. Αυτή η απλή διάκριση μειώνει σημαντικά το χρόνο εκπαίδευσης του ΝΔΓ από $O(n^2)$ σε O(n), όπου n είναι ο αριθμός των δειγμάτων.

- **Αξιολόγηση:** Η διαδικασία αξιολόγησης των προτεινόμενων μοντέλων περιλαμβάνει την απλή διαβίβαση κάθε γράφου ως είσοδο στον εκπαιδευμένο κωδικοποιητή ΝΔΓ για να ληφθούν οι τελικές αναπαραστάσεις Κόμβων. Στη συνέχεια, χρησιμοποιείται μια μέθοδος Global Pooling για την εξαγωγή μιας ενιαίας Ενσωμάτωσης σε επίπεδο γραφήματος. Αυτή η διαδικασία είναι αρκετά γρήγορη, απαιτώντας ελάχιστη υπολογιστική ισχύ (λιγότερο από 1 δευτερόλεπτο για την εξαγωγή συμπερασμάτων για 1000 γραφήματα).

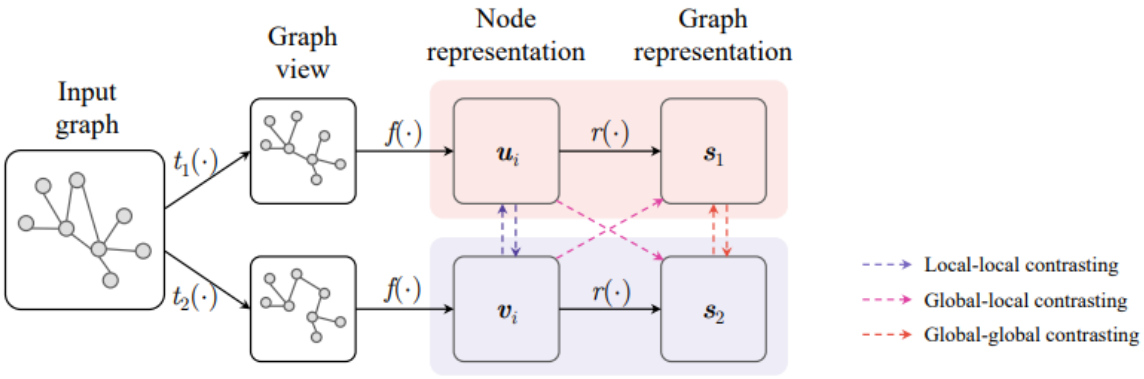


Figure 1.2.1: Διαδικασία εκπαίδευσης των αντιθετικών ΝΔΓ [96].

Η διαδικασία εκπαίδευσης ακολουθεί το περίγραμμα που παρουσιάζεται στο Σχήμα 8.2.1. Η είσοδος αποτελείται από ομάδες γράφων σκηνής, τις οποίες στη συνέχεια επεξεργάζεται ένας κωδικοποιητής ΝΔΓ. Ο κωδικοποιητής αποτελείται από στοιβαγμένα στρώματα μιας παραλλαγής ΝΔΓ σε συνδυασμό με άλλους τύπους τυποποιημένων στρωμάτων όπως ενεργοποίηση, κανονικοποίηση εκτός άλλων. Το ΝΔΓ παράγει αναπαραστάσεις σε επίπεδο κόμβων, οι οποίες στη συνέχεια αθροίζονται για να δημιουργήσουν αναπαραστάσεις γράφων.

Κατά τη διάρκεια της εκπαίδευσης, ένα σύνολο γραφημάτων σκηνής περνάει από το μοντέλο, υπολογίζοντας το αντιθετικό σφάλμα με τη χρήση ενός από τους τρόπους: Local-Local, Global-Local και Global-Global, και στη συνέχεια οι απαραίτητες αναπροσαρμογές στις παραμέτρους του δικτύου γίνονται με οπισθοδιάδοση του σφάλματος. Συγκεκριμένα, για τις τρεις διαφορετικές προσεγγίσεις που χρησιμοποιήσαμε, αξιοποιώντας την αντιθετική απώλεια σε τρεις διαφορετικές διαβαθμίσεις του γράφου, χρησιμοποιούμε τους εκτιμητές αμοιβαίας πληροφορίας που προτείνονται από τους συγγραφείς. Συγκεκριμένα, στο GraphCL [90], το οποίο λειτουργεί υπό τον τρόπο Global-Global, και στο Grace [95], το οποίο λειτουργεί υπό τον τρόπο Local-Local, χρησιμοποιούμε ως απώλεια τον εκτιμητή αμοιβαίας πληροφορίας InfoNCE. Για το InfoGraph [77], που λειτουργεί υπό τον τρόπο λειτουργίας Local-Global, χρησιμοποιούμε τον εκτιμητή αμοιβαίας πληροφορίας Jensen-Shannon. Ακολουθώντας το παράδειγμα του [20], ενσωματώνουμε μια κεφαλή προβολής (π.χ. MLP) μετά τον βασικό κωδικοποιητή ΝΔΓ και υπολογίζουμε την αντιθετική απώλεια στον πολυδιάστασο χώρο που προκύπτει από αυτή την κεφαλή προβολής. Μετά την εκπαίδευση, οι συγγραφείς του SimCLR [20] συνιστούν την μη περαιτέρω χρήση της κεφαλής προβολής. Στην περίπτωσή μας, κατά τη διάρκεια της αξιολόγησης των μοντέλων, πειραματιστήκαμε τόσο με τη χρήση όσο και με τη μη χρήση της κεφαλής προβολής, επιλέγοντας κάθε φορά το μοντέλο που παρουσίαζε την καλύτερη απόδοση.

Η υλοποίηση του κωδικοποιητή ΝΔΓ κατέχει κομβικό ρόλο στη συνολική διαδικασία. Τα σχήματα 8.2.2 και 8.2.3 παρέχουν μια λεπτομερή εικόνα του σχεδιασμού των **GAT**/**GATv2** [83, 14] και **GCN**/**GIN** [44, 89] αντίστοιχα.

Στα στρώματα που απεικονίζονται στο 8.2.2, οι συνελίξεις GAT ή GATv2 ακολουθούνται από τη συνάρτηση ενεργοποίησης ReLU. Η ενσωμάτωση του dropout επιτρέπει στους χρήστες να απενεργοποιήσουν τους νευρώνες με πιθανότητα p για να αποτρέψουν την υπερπροσαρμογή στα δεδομένα εκπαίδευσης. Οι παραλλαγές ΝΔΓ προσοχής απαιτούν προσεκτική εξέταση στον ορισμό των διαστάσεων λόγω της εφαρμογής του μηχανισμού προσοχής πολλαπλών κεφαλών, όπου οι διαστάσεις εισόδου πολλαπλασιάζονται με τον αριθμό των κεφαλών σε κάθε επίπεδο.

Όσον αφορά το GIN, ο σχεδιασμός ενός μοντέλου MLP καθίσταται απαραίτητος για την εκπαίδευση του μοντέλου. Επιλέξαμε ένα γραμμικό πλήρως συνδεδεμένο στρώμα, το οποίο διαδέχεται μια συνάρτηση ενεργοποίησης ReLU, και ένα άλλο πλήρως συνδεδεμένο γραμμικό στρώμα, όπως προτείνεται στην αρχική δημοσίευση του GIN [89].

Αξίζει να σημειωθεί ότι οι αναπαραστάσεις κόμβων που παράγονται από κάθε στρώμα των ΝΔΓ συνενώνονται πριν αθροιστούν για να δώσουν ένα μοναδικό διάνυσμα γράφου. Αυτή η πρακτική αποσκοπεί στη διατήρηση περισσότερων πληροφοριών που συλλέγονται κατά τη διάρκεια της συνέλιξης, δημιουργώντας πιο εκφραστικές αναπαραστάσεις γράφων. Μια τέτοια προσέγγιση χρησιμοποιείται και στη δημοσίευση που εισήγαγε το μοντέλο GIN [89].

Πρέπει επίσης να ορίσουμε μια συνάρτηση Global Pooling για τη συγκέντρωση των αναπαραστάσεων των κόμβων σε ένα μοναδικό διάνυσμα αναπαράστασης για κάθε γράφο. Μετά από δοκιμές με τις ευρέως χρησιμοποιούμενες συναρτήσεις mean/max/min/sum, κατέστη προφανές ότι η συνάρτηση Sum παρείχε τα καλύτερα αποτελέσματα. Ως αποτέλεσμα, όλα τα τελικά μοντέλα ενσωματώνουν τη συγκεκριμένη συνάρτηση για την παραγωγή αναπαραστάσεων γράφων από τις αναπαραστάσεις των κόμβων τους. Η απόφαση αυτή ευθυγραμμίζεται επίσης και με την προσέγγιση των δημιουργών του GIN [89], οι οποίοι επίσης επέλεξαν να αθροίσουν τις αναπαραστάσεις κόμβων για να ενισχύσουν την εκφραστικότητα.

Για την αξιολόγηση των προτεινόμενων μοντέλων στο πλαίσιο της εργασίας ανάκτησης γράφων σκηνής, οι τελικές αναπαραστάσεις γράφων συγκρίνονται μεταξύ τους με την χρήση της ομοιότητας συνημίτονου. Πειραματιστήκαμε επίσης με την απόσταση L_2, αλλά υπήρξε σημαντική πτώση της απόδοσης σε αυτή την περίπτωση.

Σε όλες τις περιπτώσεις, στα πλαίσια των GraphCL, InfoGraph και Grace, με προσεκτική ρύθμιση των υπερπαραμέτρων και επιλογή του καλύτερου κωδικοποιητή ΝΔΓ σε κάθε περίπτωση, τα μοντέλα αυτά έδωσαν ανταγωνιστικά αποτελέσματα σε inductive setting σε σύγκριση με τους Πυρήνες Γράφων. Για την περαιτέρω ενίσχυση

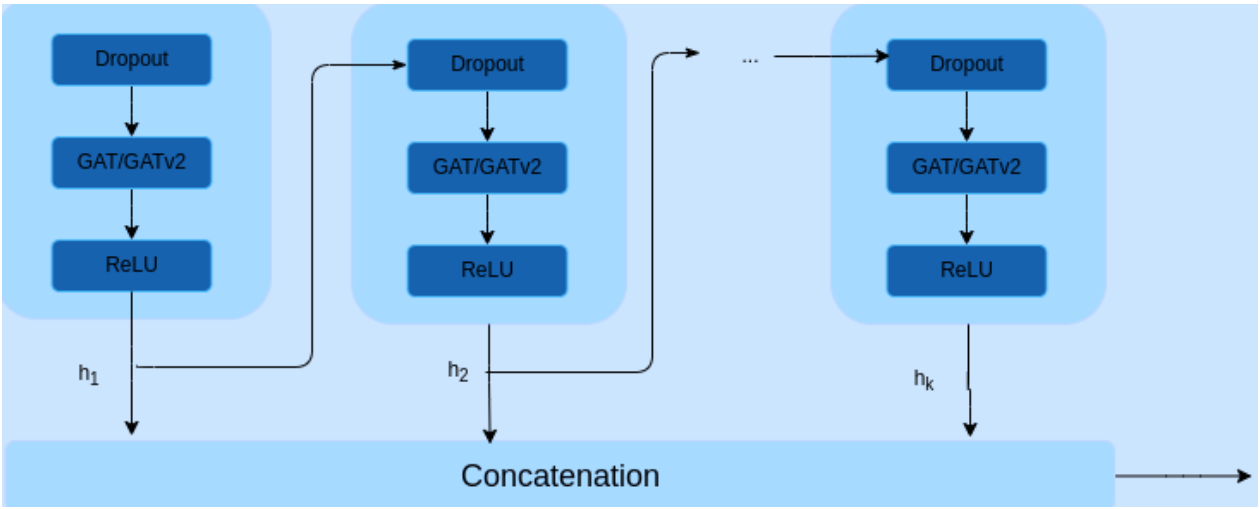


Figure 1.2.2: Σχεδιασμός των παραλλαγών κωδικοποιητή GAT/GATv2

των επιθυμητών μετρικών, αντλήσαμε έμπνευση κυρίως από το [56] (για μια πιο εμπεριστατωμένη ανάλυση μπορεί κανείς να κοιτάξει στο [19]), όπου μια μορφή ασθενούς επίβλεψης από τις λεζάντες των εικόνων προσφέρεται στα μοντέλα έτσι ώστε γραφήματα σκηνής με πιο παρόμοιες λεζάντες να αναπαρίστανται πιο κοντά από τα ΝΔΓ. Για την επίτευξη των δικών μας στόχων, τροποποιήσαμε την συνάρτηση σφάλματος έτσι ώστε να παρέχουμε μια ασθενή επίβλεψη στα ΝΔΓ με βάση το Graph Edit Distance και όχι την ομοιότητα των λεζάντων από τις αρχικές εικόνες. Μετά την αρχική προεκπαίδευση ενός μοντέλου σε ένα από τα πλαίσια GraphCL, InfoGraph ή Grace, προχωράμε σε fine tuning με βάση αυτό το σφάλμα. Κατά τη διάρκεια αυτής της φάσης, το μοντέλο εκτίθεται σε ένα περιορισμένο ποσοστό ground truth. Επιπλέον, το προτεινόμενο σφάλμα, το οποίο ορίζεται στην παρακάτω εξίσωση, λειτουργεί με αντιθετικό τρόπο. Συγκεκριμένα, επεξεργάζεται τριπλέτες γραφημάτων σκηνής. Για ένα δεδομένο γράφο A, λαμβάνουμε ομοιόμορφα και τυχαία δείγμα δύο άλλων γραφημάτων σκηνής. Το ένα με το μικρότερο GED από τον γράφο A θεωρείται ως θετικό δείγμα P, ενώ το άλλο αντιμετωπίζεται ως αρνητικό, N.

$$L = -PlogP\hat{} - (1-P)log(1-\hat{P})$$

Στην παραπάνω εξίσωση, οι αναπαραστάσεις γράφων των A, P, N συμβολίζονται ως f_A, f_P, f_N και παράγονται από ένα ΝΔΓ. Η πιθανότητα $\hat{P}$ να προβλέπονται ομοιότητες (ή αποστάσεις) με τη σωστή σειρά υπολογίζεται ως $\hat{P} = \sigma\left(\frac{f_A^T f_P - f_A^T f_N}{\tau}\right)$. Επιπλέον, η πιθανότητα $P(d_{AN} > d_{AP}) = P\left(\frac{1}{d_{AN}} < \frac{1}{d_{AP}}\right) = P(s_{AN} < s_{AP}) = \frac{d_{AN}}{d_{AP}+d_{AN}}$, αντιπροσωπεύει την επιθυμητή τιμή-στόχο.

Με τη χρήση αυτού του σφάλματος, το οποίο "τιμωρεί" το μοντέλο κάθε φορά που το αρνητικό δείγμα κωδικοποιείται πιο κοντά στον γράφο A από ό,τι το θετικό, βελτιώνονται περαιτέρω οι αρχικές αναπαραστάσεις που παράγονται μέσω της αντιθετικής προεκπαίδευσης. Κατά συνέπεια, το νέο fine tuned ΝΔΓ μπορεί να χρησιμοποιηθεί για την ανάκτηση γραφημάτων σκηνής δίνοντας ακόμη πιο ανταγωνιστικά αποτελέσματα.

Η επόμενη ενότητα περιλαμβάνει λεπτομέρειες σχετικά με τις συγκεκριμένες υπερπαραμέτρους και άλλες επιλογές (όπως στρώματα, μέγεθος αναπαράστασεις γράφου, εποχές, μέγεθος παρτίδας, βελτιστοποιητές, επαυξήσεις κ.λπ.) για κάθε μοντέλο.

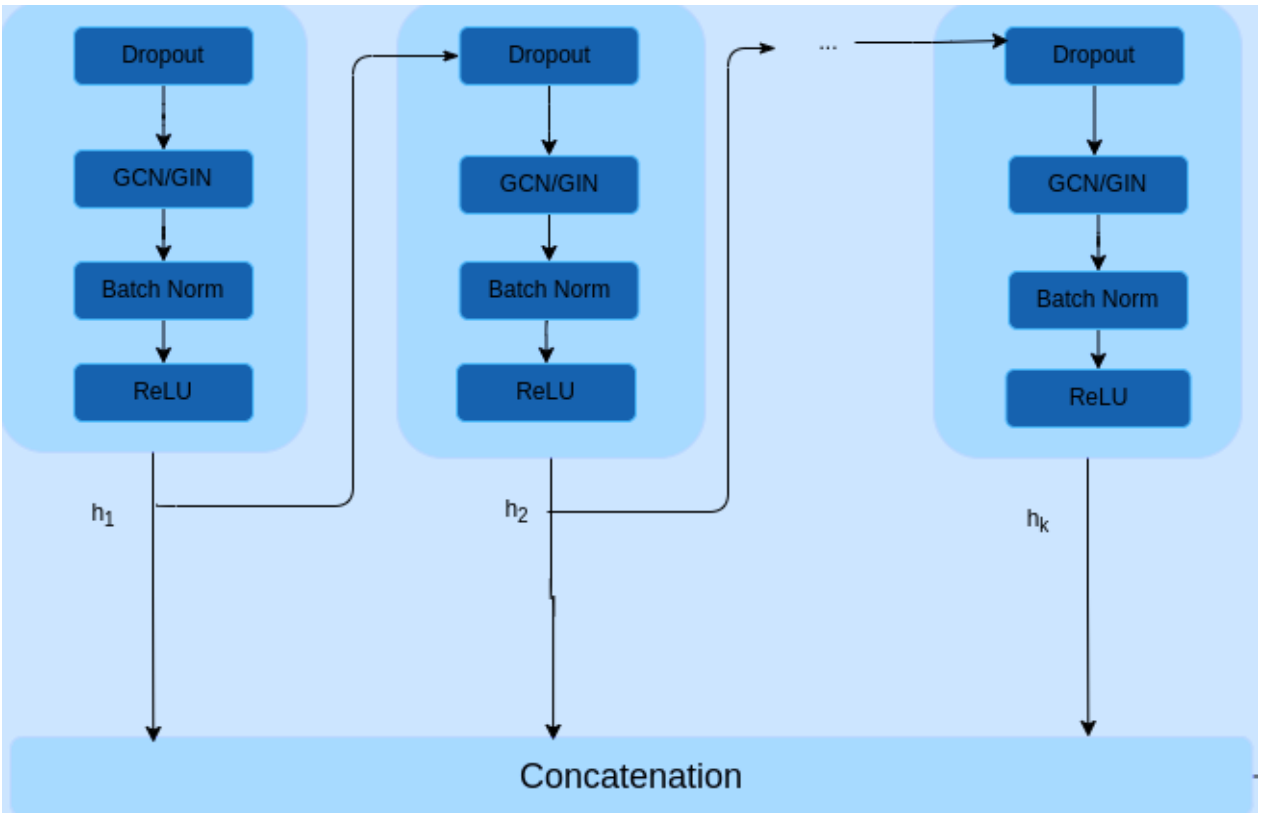


Figure 1.2.3: Σχεδιασμός των παραλλαγών κωδικοποιητών GCN/GIN

1.3 Πειράματικό Μέρος

1.3.1 Σύνολο Δεδομένων

Το σύνολο δεδομένων που θα χρησιμοποιήσουμε είναι το Visual Genome [47], ένα σύνολο δεδομένων μεγάλης κλίμακας που επικεντρώνεται στην οπτική κατανόηση με τη χρήση πληροφοριών δομημένων σε γράφους, ιδίως στον τομέα της όρασης υπολογιστών. Περιέχει πάνω από 108k εικόνες οι οποίες είναι λεπτομερώς σχολιασμένες, συμπεριλαμβανομένων των αντικειμένων, των σχέσεων και των χαρακτηριστικών εντός των εικόνων, καθώς και περιγραφές περιοχών και εικόνων. Όλες αυτές οι οντότητες αντιστοιχίζονται επίσης με το WordNet. Τα γραφήματα σκηνών που παρέχονται στο σύνολο δεδομένων προέρχονται από εικόνες κυρίως από το σύνολο δεδομένων MS-COCO [52].

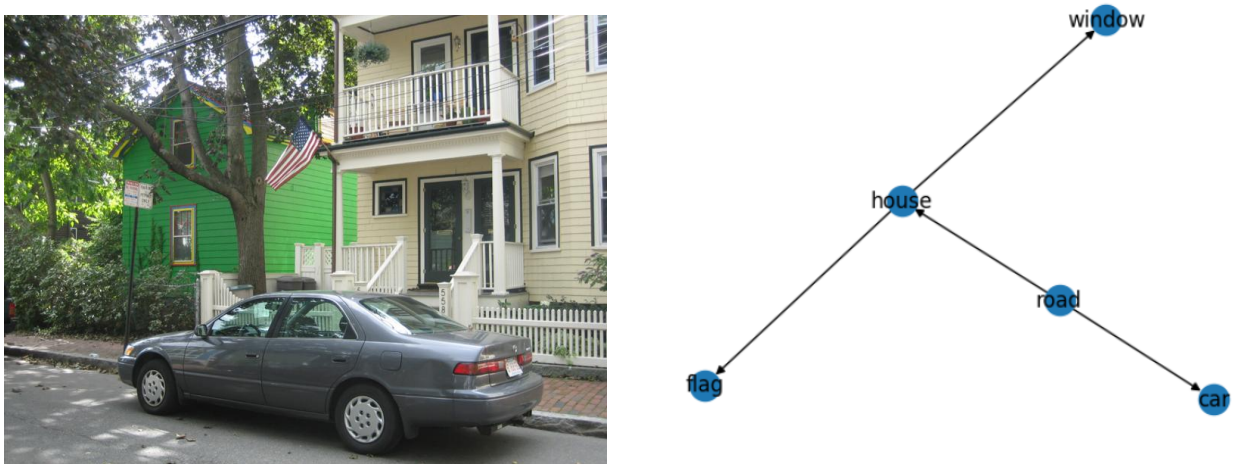


Figure 1.3.1: Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής όπου οι τύποι σχέσεων έχουν αφαιρεθεί

Όπως είναι προφανές από το παραπάνω σχήμα, οι γράφοι σκηνής αποτυπώνουν αποτελεσματικά το σημασιολογικό περιεχόμενο μιας εικόνας αναπαριστώντας τα αντικείμενα ως κόμβους και τις αλληλεπιδράσεις τους ως ακμές στο γράφο.

Προεπεξεργασία δεδομένων

- **Χαρακτηριστικά Κόμβων:** Οι αρχικοί γράφοι σκηνής από το Visual Genome δεν διαθέτουν χαρακτηριστικά κόμβων. Ωστόσο, τα ΝΔΓ απαιτούν έναν πίνακα χαρακτηριστικών με αριθμητικές τιμές ως είσοδο. Για να ξεπεράσουμε αυτόν τον περιορισμό, χρησιμοποιήσαμε το πεδίο "objects" από το αρχικό σύνολο δεδομένων. Εντός αυτού του πεδίου, για κάθε γράφο σκηνής, εξήγαμε τα synsets και τα ονόματα που σχετίζονται με κάθε αντικείμενο που υπάρχει στην εικόνα. Για τη μετατροπή των περιγραφών των αντικειμένων από συμβολοσειρές σε αριθμητικά διανύσματα, χρησιμοποιήσαμε το GloVe Word Embeddings [65], μια καθιερωμένη μέθοδο στον τομέα της επεξεργασίας φυσικής γλώσσας (NLP). Συγκεκριμένα, για κάθε όνομα αντικειμένου, πραγματοποιήσαμε μια απλή αναζήτηση στον πίνακα GloVe για να λάβουμε το αντίστοιχο αριθμητικό διάνυσμα. Επιλέξαμε την έκδοση 300 διαστάσεων του GloVe, καθώς βελτίωναν σταθερά την απόδοση παρέχοντας πιο πλούσιες πληροφορίες για τα αντικείμενα. Ταυτόχρονα, διατηρήσαμε τα synsets των κόμβων προκειμένου να τα τροφοδοτήσουμε αργότερα στον αλγόριθμο GED που υπολογίζει το ground truth χρησιμοποιώντας την ιεραρχία "is-a" που παρέχεται από το WordNet [59]. Επιπλέον, αφαιρέσαμε το πεδίο "attributes" από κάθε αντικείμενο για να αποκλείσουμε τις περιττές πληροφορίες από τους γράφους μας. Για παράδειγμα, στην περίπτωση μιας εικόνας που περιέχει ένα αυτοκίνητο, τα πιθανά χαρακτηριστικά μπορεί να περιλαμβάνουν το χρώμα ή τη μάρκα του. Για να διασφαλίσουμε ότι τα μοντέλα μας επικεντρώνονται στην παρουσία του αυτοκινήτου και όχι σε περίπλοκες λεπτομέρειες, αποφασίσαμε να αγνοήσουμε αυτές τις πρόσθετες πληροφορίες.
- **Edges:** Για να αξιοποιήσουμε το πεδίο σχέσεων που παρέχεται στο αρχικό σύνολο δεδομένων, επιλέξαμε να αναπαραστήσουμε όλες τις σχέσεις ως κατευθυνόμενες ακμές που συνδέουν τα δύο αντικείμενα. Παρόμοια με τα χαρακτηριστικά των κόμβων, οι σχέσεις παρουσιάζουν διάφορους τύπους. Ωστόσο, επιλέξαμε

να μην ενσωματώσουμε αυτή την πρόσθετη πληροφορία, δεδομένου ότι η πλειονότητα των πυρήνων γράφων δεν λαμβάνει υπόψη τα χαρακτηριστικά ακμών. Επιπλέον, ο αλγόριθμος [29] που χρησιμοποιείται για τον υπολογισμό του ground truth λαμβάνει υπόψη κυρίως πληροφορίες κόμβων. Εκτός αυτού, η βασική σημασιολογική πληροφορία των εικόνων, για την οποία ενδιαφερόμαστε κυρίως, βρίσκεται στα αντικείμενα που αναπαρίστανται ως κόμβοι και για λόγους απλότητας αποφασίσαμε να αγνοήσουμε αυτή την πρόσθετη πληροφορία.

Θα περιγράψουμε τώρα τη διαδικασία που χρησιμοποιήσαμε για να ορίσουμε και να επιλέξουμε γράφους σκηνών από δύο διαφορετικές κατηγορίες, τους πυκνούς και τους τυχαίους γράφους σκηνών. Ο λόγος πίσω από αυτόν τον διαχωρισμό είναι διττός.

Αφενός, παρατηρήσαμε ότι η συντριπτική πλειονότητα των γράφων σκηνών στο σύνολο δεδομένων ήταν πολύ αραιοί. Αυτοί οι γράφοι διαθέτουν πολυάριθμους απομονωμένους κόμβους, παρεμποδίζοντας την αποτελεσματικότητα του σχήματος μεταβίβασης μηνυμάτων που εκτελείται από τα ΝΔΓ, καθώς και τη λειτουργία διαφόρων Graph Kernels που βασίζονται στις συνδέσεις κόμβων. Ένας αραιός (τυχαία επιλεγμένος γράφος σκηνής) φαίνεται στο Σχήμα 9.1.3 όπου μπορεί κανείς να δει την αραιότητα του γράφου.

Από την άλλη πλευρά, η επιλογή του τυχαίου υποσυνόλου καθοδηγήθηκε από το χαρακτηριστικό του ότι είναι ένα υποσύνολο χωρίς περιορισμούς, που δεν απαιτεί εξειδικευμένο χειρισμό. Επιπλέον, το ενδιαφέρον μας προσανατολίστηκε στην εξέταση της προσαρμοστικότητας ενός προεκπαιδευμένου Contrastive GNN σε ένα τυχαίο υποσύνολο γράφων και στην αξιολόγηση μετέπειτα της απόδοσής του σε ένα σύνολο δεδομένων με περισσότερους περιορισμούς και ειδικά χαρακτηριστικά, όπως είναι το πυκνό.

Για τη δημιουργία του πυκνού συνόλου, εισαγάγαμε συγκεκριμένα κριτήρια για την επιλογή των γραφημάτων σκηνών. Συγκεκριμένα, επιλέξαμε γραφήματα με τουλάχιστον 3 κόμβους και 3 ακμές και οι τιμές πυκνότητάς τους ήταν στο εύρος [0.1, 1.0]. Τα στατιστικά στοιχεία αυτού του πυκνού συνόλου απεικονίζονται στο Σχήμα 9.1.4.

Για το τυχαίο σύνολο, απλά επιλέξαμε 1000 τυχαίους γράφους, επιβάλλοντας δύο περιορισμούς. Πρώτον, ο αριθμός των κόμβων έπρεπε να είναι ίσος ή μεγαλύτερος από 6 και ο αριθμός των ακμών έπρεπε να είναι ίσος ή μεγαλύτερος από 3. Αυτοί οι περιορισμοί εφαρμόστηκαν προκειμένου να μην εξεταστούν γράφοι με περιορισμένο αριθμό αντικειμένων και σχέσεων (ακμών). Τα βασικά στατιστικά στοιχεία για το τυχαίο σύνολο απεικονίζονται στο Σχήμα 9.1.2.

Είναι σημαντικό να σημειωθεί ότι σε επόμενα πειράματα, εάν είναι απαραίτητο (π.χ. σε περιπτώσεις inductive inference), θα χρησιμοποιήσουμε ένα ξεχωριστό τυχαία επιλεγμένο σύνολο, το οποίο θα υπόκειται στους ίδιους περιορισμούς, αποκλειστικά για εκπαίδευση. Η φάση του inference θα πραγματοποιείται πάντα είτε στο αρχικό πυκνό είτε στο αρχικό τυχαίο σύνολο.

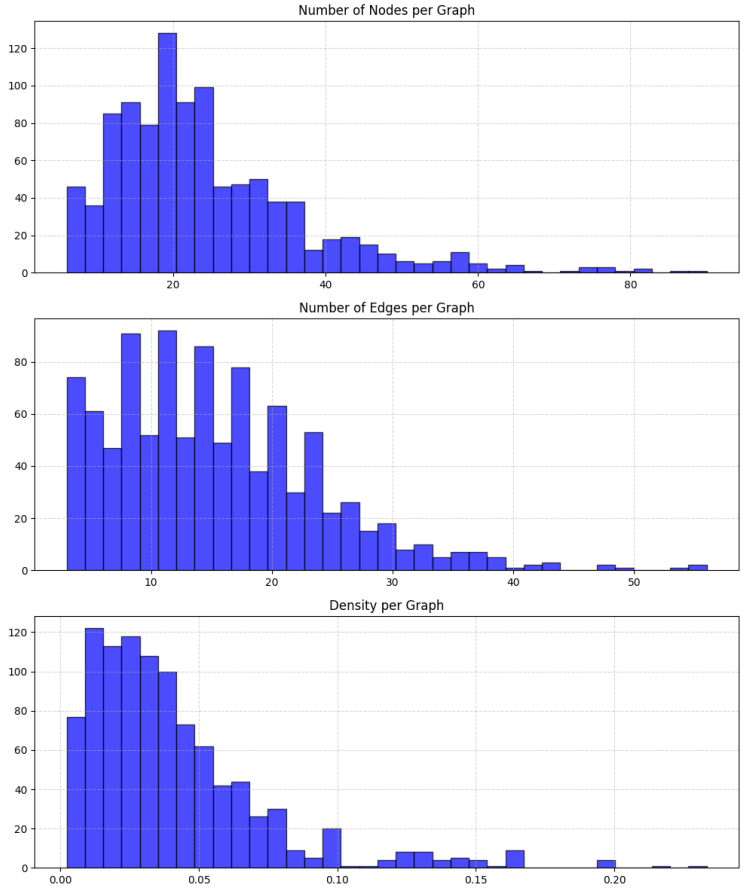


Figure 1.3.2: Στατιστικά στοιχεία για τα γραφήματα σκηνών στο τυχαίο σύνολο

Figure 1.3.3: Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής από το τυχαίο σύνολο.

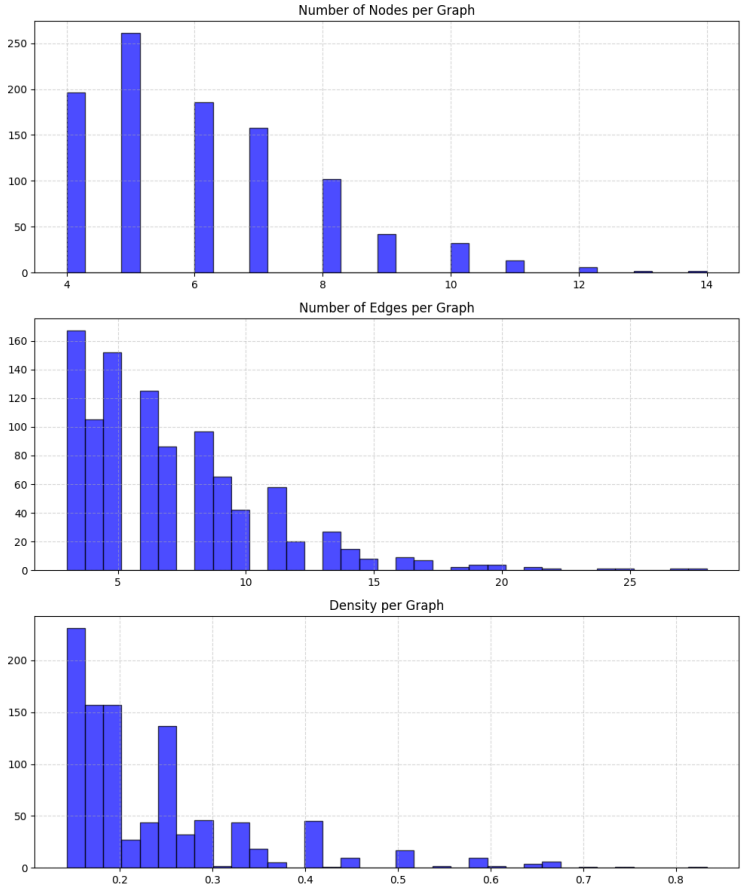


Figure 1.3.4: Στατιστικά στοιχεία για τα γραφήματα σκηνών στο πυκνό σύνολο

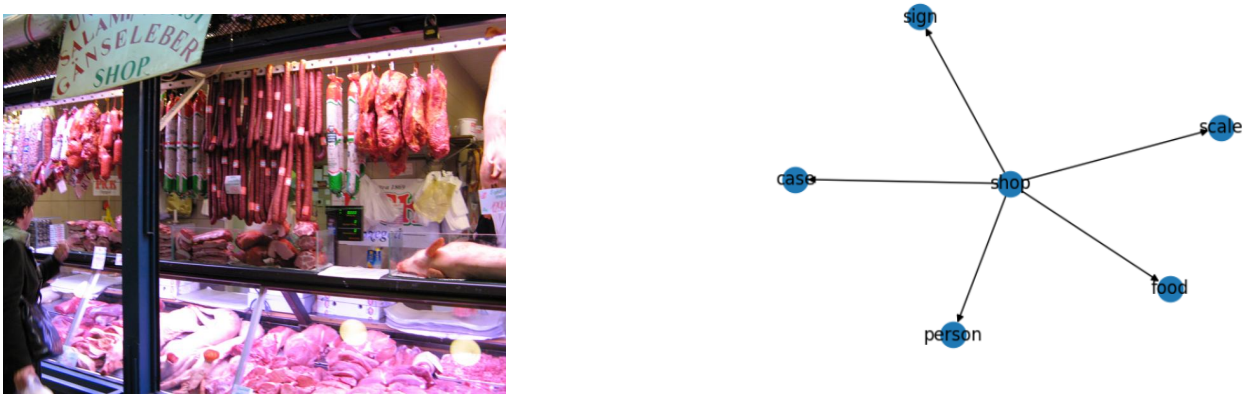


Figure 1.3.5: Ένα παράδειγμα μιας εικόνας μαζί με τον αντίστοιχο γράφο σκηνής από το πυκνό σύνολο.

1.3.2 Μετρικές αξιολόγησης

Mean Reciprocal Rank (MRR@K)

Η MRR είναι μια μετρική που αξιολογεί την αποτελεσματικότητα ενός συστήματος συστάσεων λαμβάνοντας υπόψη την θέση του πρώτου σχετικού στοιχείου που επιστρέφεται από το μοντέλο μας. Η ιδέα πίσω από την MRR είναι να επιβραβεύει τα συστήματα που τοποθετούν τα σχετικά στοιχεία υψηλότερα στη λίστα συστάσεων.

Το MRR@K για ένα σύνολο ερωτημάτων ορίζεται ως εξής:

$$\text{MRR@K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

όπου Q είναι το σύνολο των ερωτημάτων, rank_i είναι η θέση του πρώτου σχετικού στοιχείου που επιστρέφεται για το ερώτημα i, και $|Q|$ είναι ο συνολικός αριθμός των ερωτημάτων.

Mean Average Precision (MAP@K)

Η MAP είναι μια μετρική που αξιολογεί την ποιότητα ενός συστήματος συστάσεων. Τα κύρια συστατικά στοιχεία που είναι απαραίτητα για τον υπολογισμό του MAP@K παρουσιάζονται παρακάτω.

1. **Precision:**
$$\text{Precision@}k = \frac{\text{Αριθμός σχετικών στοιχείων μεταξύ των κορυφαίων } k}{k}$$

2. **Average Precision (AP):**

$$AP@K = \frac{\sum_{k=1}^{K} Precision@k \times \text{rel}(k)}{\text{Συνολικός αριθμός σχετικών περιπτώσεων}}$$

   όπου K είναι ο συνολικός αριθμός των στοιχείων στη λίστα "Predicted" (π.χ. στα πειράματά μας K = 10), και $\text{rel}(k)$ είναι μια συνάρτηση δείκτη ίση με 1 εάν το στοιχείο στην κατάταξη k είναι σχετικό και 0 διαφορετικά.

3. **Mean Average Precision (MAP):** MAP είναι ο μέσος όρος των τιμών AP που υπολογίστηκαν στο προηγούμενο βήμα για πολλαπλά ερωτήματα. Ορίζεται ως εξής:

$$MAP@k = \frac{\sum_{i=1}^{Q} AP@K}{Q_i}$$

   όπου Q είναι ο συνολικός αριθμός των ερωτημάτων.

Normalized Discounted Cumulative Gain (NDCG@K)

1. **Cumulative Gain (CG):** Το αθροιστικό κέρδος είναι ένα μέτρο που αθροίζει τις βαθμολογίες συνάφειας των κορυφαίων K (στα πειράματά μας K=10) στοιχείων στην "προβλεπόμενη" λίστα κατάταξης που επιστρέφει ένα μοντέλο συστάσεων. Η βαθμολογία συνάφειας είναι ένα εύρος βαθμών συνάφειας όπου το 0 είναι το λιγότερο συναφές και κάποια υψηλότερη τιμή αναπαριστά το πιο συναφές. Προκειμένου να καθορίσουμε τις βαθμολογίες συνάφειας στα πειράματά μας, χρησιμοποιούμε το αντίστροφο του Graph Edit Distance Ground Truth, καθώς η έννοια της συνάφειας είναι αντίστροφη αυτής του Edit Distance. Στη συνέχεια, χρησιμοποιώντας ένα Min-Max Scaler μετατρέπουμε αυτές τις τιμές σε μια κλίμακα από το 1 έως το 10 προκειμένου να αποφύγουμε ακραίες τιμές. Ο τύπος για το αθροιστικό κέρδος στη θέση K έχει ως εξής:

$$CG@K = \sum_{i=1}^{K} rel_i$$

2. **Discounted Cumulative Gain (DCG):** Το DCG είναι μια επέκταση του CG που εισάγει έναν παράγοντα για να δώσει λιγότερη σημασία στα στοιχεία που εμφανίζονται χαμηλότερα στον κατάλογο συστάσεων. Ο τύπος

για το DCG@K είναι:

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{\log_2(i+1)}$$

Με αυτόν τον τρόπο, υψηλότερες βαθμολογίες αποδίδονται σε στοιχεία που είναι σχετικά και εμφανίζονται επίσης υψηλότερα στον κατάλογο συστάσεων.

3. **Ideal Discounted Cumulative Gain (IDCG):** Το IDCG αντιπροσωπεύει το μέγιστο εφικτό DCG για ένα δεδομένο σύνολο στοιχείων. Ο τύπος για το IDCG@K είναι παρόμοιος με το DCG@K, αλλά λαμβάνει υπόψη την ιδανική ταξινόμηση:

$$IDCG@K = \sum_{i=1}^{K} \frac{rel_{\text{ideal},i}}{\log_2(i+1)}$$

όπου $rel_{\text{ideal},i}$ είναι η συνάφεια του ιδανικού στοιχείου στη θέση i του ταξινομημένου καταλόγου.

4. **Normalized Discounted Cumulative Gain (NDCG):** Το NDCG είναι η κανονικοποιημένη εκδοχή του DCG, που υπολογίζεται διαιρώντας το DCG@K με το IDCG@K. Αυτή η κανονικοποίηση εξασφαλίζει ότι η μετρική εμπίπτει στο εύρος [0, 1], με το 1 να υποδηλώνει τέλεια κατάταξη. Ο τύπος για το NDCG@K είναι:

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

Η NDCG είναι ιδιαίτερα χρήσιμη κατά τη σύγκριση αποτελεσμάτων μεταξύ διαφορετικών συνόλων δεδομένων ή συστημάτων, καθώς παρέχει ένα τυποποιημένο μέτρο για ποσοτική σύγκριση. Συνοπτικά, η NDCG@K είναι μια ολοκληρωμένη μετρική που λαμβάνει υπόψη τόσο τη συνάφεια όσο και τη θέση των στοιχείων σε μία λίστα συστάσεων. Στην περίπτωσή μας, όπου έχουμε πολλαπλά ερωτήματα, υπολογίζουμε απλώς τη μέση τιμή του NDCG για όλα τα ερωτήματα.

1.3.3 Βασική Αλήθεια

Είναι προφανές από την υποενότητα "Μετρικές αξιολόγησης" ότι για να αξιολογηθεί η αποτελεσματικότητα των μοντέλων, είναι απαραίτητο να προσδιοριστούν τα πραγματικά αποτελέσματα για κάθε ερώτημα. Όπως έχουμε ήδη αναφέρει, η βασική αλήθεια μεταξύ δύο γράφων υπολογίζεται από τον αλγόριθμο Graph Edit Distance. Στην περίπτωσή μας, πρέπει να υπολογίσουμε έναν πίνακα βασικής αλήθειας όπου θα αποθηκεύονται οι αποστάσεις επεξεργασίας γράφων μεταξύ οποιουδήποτε ζεύγους γράφων. Ωστόσο, το υπολογιστικό κόστος του ακριβούς Graph Edit Distance είναι απαγορευτικό, καθώς δεν υπάρχει πολυωνυμικός αλγόριθμος για αυτόν τον σκοπό.

Για να δημιουργηθεί ο πίνακας βασικής αλήθειας για τα τυχαία και πυκνά σύνολα, καθένα από τα οποία αποτελείται από 1000 γραφήματα, ο αλγόριθμος GED πρέπει να εκτελεστεί περίπου 500.000 φορές για κάθε σύνολο. Για να μειώσουμε τον υπολογιστικό φόρτο που συνδέεται με τον υπολογισμό του GED, χρησιμοποιήσαμε τον προσεγγιστικό αλγόριθμο Bipartite Matching. Προκειμένου να τον υλοποιήσουμε, χρησιμοποιήσαμε τη βιβλιοθήκη Deep Graph Learning σε Python, γνωστή ως DGL.

Ωστόσο, θα πρέπει πρώτα να δώσουμε στον αλγόριθμο το κόστος της εισαγωγής, της διαγραφής και της αντικατάστασης των κόμβων. Για το σκοπό αυτό αξιοποιούμε το WordNet [59] Tbox graph προκειμένου να υπολογίσουμε την απόσταση μεταξύ κάθε δύο αντικειμένων που χρειαζόμαστε όπως προτείνεται στην [31].

Το WordNet είναι μια λεξιλογική βάση δεδομένων της αγγλικής γλώσσας που οργανώνει τις λέξεις σε σύνολα συνωνύμων (synsets), τα οποία αντιπροσωπεύουν έννοιες ή σημασίες. Μια σημαντική πτυχή του WordNet είναι η ιεραρχική του δομή, ιδιαίτερα η ιεραρχία *is-a*, η οποία αποτυπώνει τις σχέσεις υπερώνυμου-υποωνύμου. Στην ιεραρχία αυτή, ένα *hypernym* είναι ένας γενικότερος όρος ή έννοια, ενώ ένα *hyponym* είναι ένας ειδικότερος όρος ή έννοια. Η σχέση *is-a* δηλώνει ότι ένα υποώνυμο είναι υποτύπος ή περίπτωση του υπερωνύμου του. Για παράδειγμα:

- **Hypernym:** Ζώο
  - **Hyponym:** Θηλαστικό, Πουλί, Ψάρι

Σε αυτό το παράδειγμα, τα "Θηλαστικό", "Πουλί" και "Ψάρι" είναι hyponyms του hypernym "Ζώο".

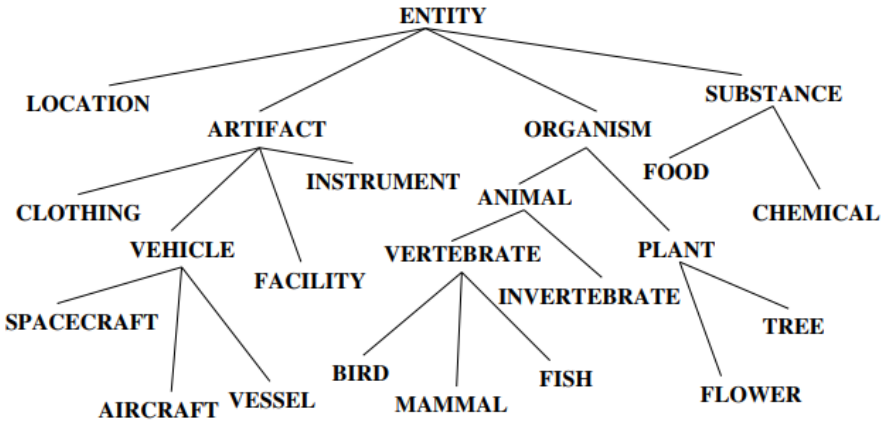


Figure 1.3.6: Μια απλή αναπαράσταση του τμήματος της ιεραρχίας ουσιαστικών του WordNet κάτω από τον κόμβο ρίζας της έννοιας "entity".

Καθώς το σύνολο δεδομένων Visual Genome παρέχει ένα αντίστοιχο Synset για κάθε αντικείμενο σε ένα γράφημα σκηνής, και εφόσον έχουμε αναπαραστήσει κάθε αντικείμενο ως κόμβο σε ένα γράφημα, η χρήση του WordNet για τη μέτρηση της απόστασης εννοιών γίνεται απλή. Συγκεκριμένα, χρησιμοποιούμε το πακέτο NLTK Python [53] το οποίο μας παρέχει πρόσβαση στη βάση δεδομένων WordNet και στην ιεραρχία ουσιαστικών.

Προκειμένου να υπολογίσουμε την απόσταση μεταξύ δύο εννοιών στην "is-a" ιεραρχία χρησιμοποιούμε τη συνάρτηση `path_similarity` που περιλαμβάνει το πακέτο NLTK. Η συνάρτηση αυτή υπολογίζει ένα σκορ που κυμαίνεται από 0 έως 1 και δηλώνει την ομοιότητα μεταξύ δύο λεκτικών εννοιών με βάση το συντομότερο μονοπάτι στην ταξινομία is-a.

Έτσι, προκειμένου να υπολογίσουμε το κόστος των αντικαταστάσεων κόμβων χρησιμοποιούμε την `path_similarity` για να εξάγουμε το σκορ ομοιότητας για όλα τα πιθανά ζεύγη κόμβων μεταξύ του γράφου G_i και του γράφου G_j. Στη συνέχεια, η εννοιολογική απόσταση μεταξύ κάθε ζεύγους κόμβων υπολογίζεται ως $1 - \text{similarity\_score}$. Για το κόστος εισαγωγής και διαγραφής κόμβων ακολουθείται η ίδια διαδικασία, αλλά αντί να χρησιμοποιούμε τη συνάρτηση `path_similarity` για ζεύγη εννοιών, τη χρησιμοποιούμε για τον υπολογισμό της ομοιότητας μεταξύ κάθε κόμβου και του κόμβου entity που είναι ο κόμβος-ρίζα της ιεραρχίας is-a.

Αυτή η διαδικασία επαναλαμβάνεται για κάθε πιθανό ζεύγος γραφημάτων σκηνής τόσο για τα πυκνά όσο και για τα τυχαία σύνολα και προκύπτουν οι τελικοί πίνακες βασικής αλήθειας. Ο χρόνος εκτέλεσης για το "Τυχαίο" σύνολο δεδομένων ήταν περίπου ~ 8 ώρες και για το "Πυκνό" σύνολο δεδομένων ήταν περίπου ~ 1 ώρα.

Εδώ, παρουσιάζουμε ένα σύνολο αντιπροσωπευτικών εικόνων και αναλύουμε τις βαθμολογίες ομοιότητάς τους χρησιμοποιώντας τη μετρική Graph Edit Distance που ορίσαμε.

Στην εικόνα 9.1.7, παρουσιάζονται δύο διαφορετικές εικόνες, η καθεμία με πανομοιότυπα γραφήματα σκηνών. Η αποτελεσματικότητα του αλγορίθμου Bipartite Matching είναι εμφανής, καθώς προσδιορίζει με ακρίβεια μια απόσταση επεξεργασίας γραφήματος 0, υποδεικνύοντας την ομοιότητα μεταξύ των εικόνων με βάση τις αντίστοιχες αναπαραστάσεις γραφήματος σκηνής.

Στην τελευταία εικόνα, 9.1.8, παρουσιάζονται δύο εικόνες -η μία από ένα μπάνιο και η άλλη από έναν αγώνα τένις- με δομικά πολύ όμοιους γράφους σκηνής. Παρά το γεγονός αυτό, ο αλγόριθμος GED τους αποδίδει υψηλή τιμή απόστασης και η εικόνα του γηπέδου τένις δεν εμφανίζεται καν στη λίστα με τις 20 πρώτες συστάσεις. Αυτή η παρατήρηση υπογραμμίζει τον κρίσιμο ρόλο της σημασιολογίας των αντικειμένων, καθώς ο προσεγγιστικός αλγόριθμος GED αποδίδει σημαντική βάρος σε αυτή τη σημασιολογία. Για παράδειγμα, η έννοια του "πλακιδίου" απέχει σημαντικά στην ιεραρχία του WordNet από την έννοια του "παίκτη". Όλες αυτές οι παρατηρήσεις τεκμηριώνουν την αποτελεσματικότητα του αλγορίθμου Bipartite Matching ως εξαιρετικού μέτρου ομοιότητας, επιβεβαιώνοντας την απόφασή μας να τον χρησιμοποιήσουμε ως βασική αλήθεια.

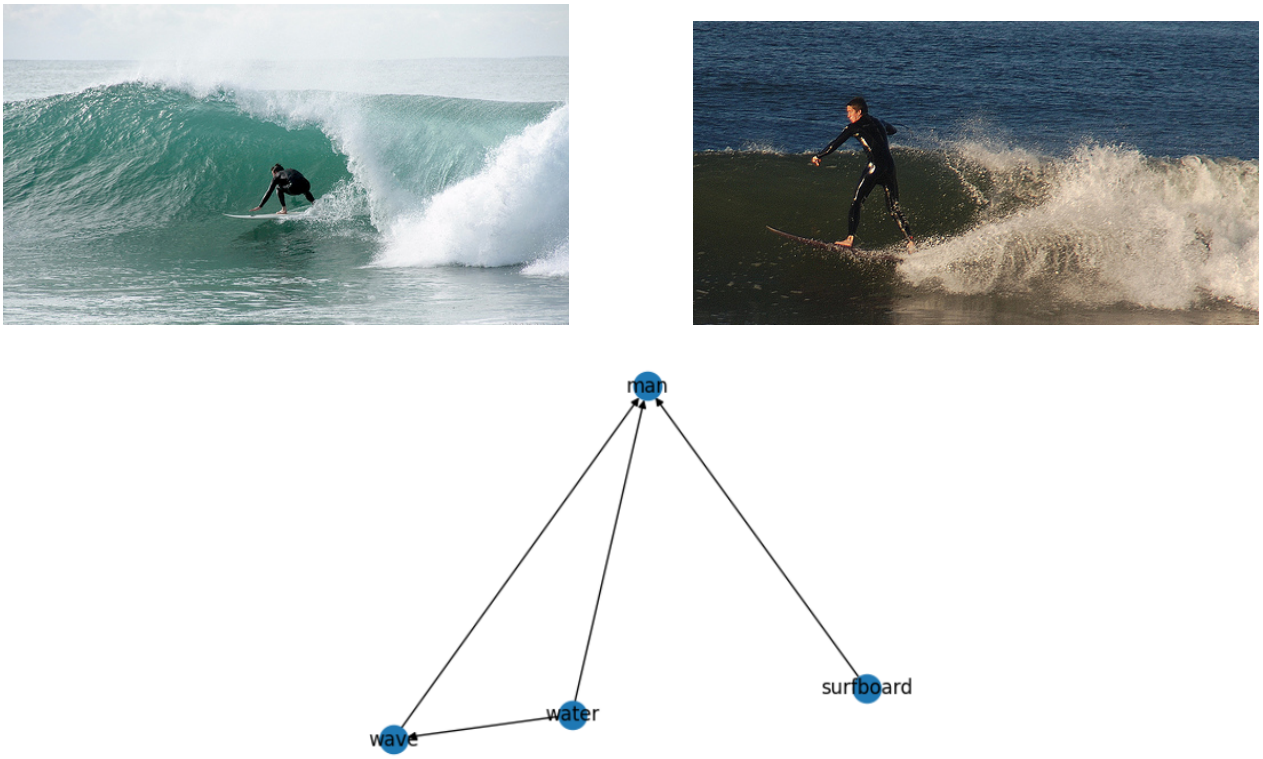


Figure 1.3.7: Παρουσιάζονται δύο εικόνες από το πυκνό σύνολο, καθεμία από τις οποίες παρουσιάζει ένα πανομοιότυπο γράφημα σκηνής. Ο προσεγγιστικός αλγόριθμος GED απέδωσε με ακρίβεια βαθμολογία απόστασης 0, αναγνωρίζοντας ότι μοιράζονται το ίδιο σημασιολογικό περιεχόμενο

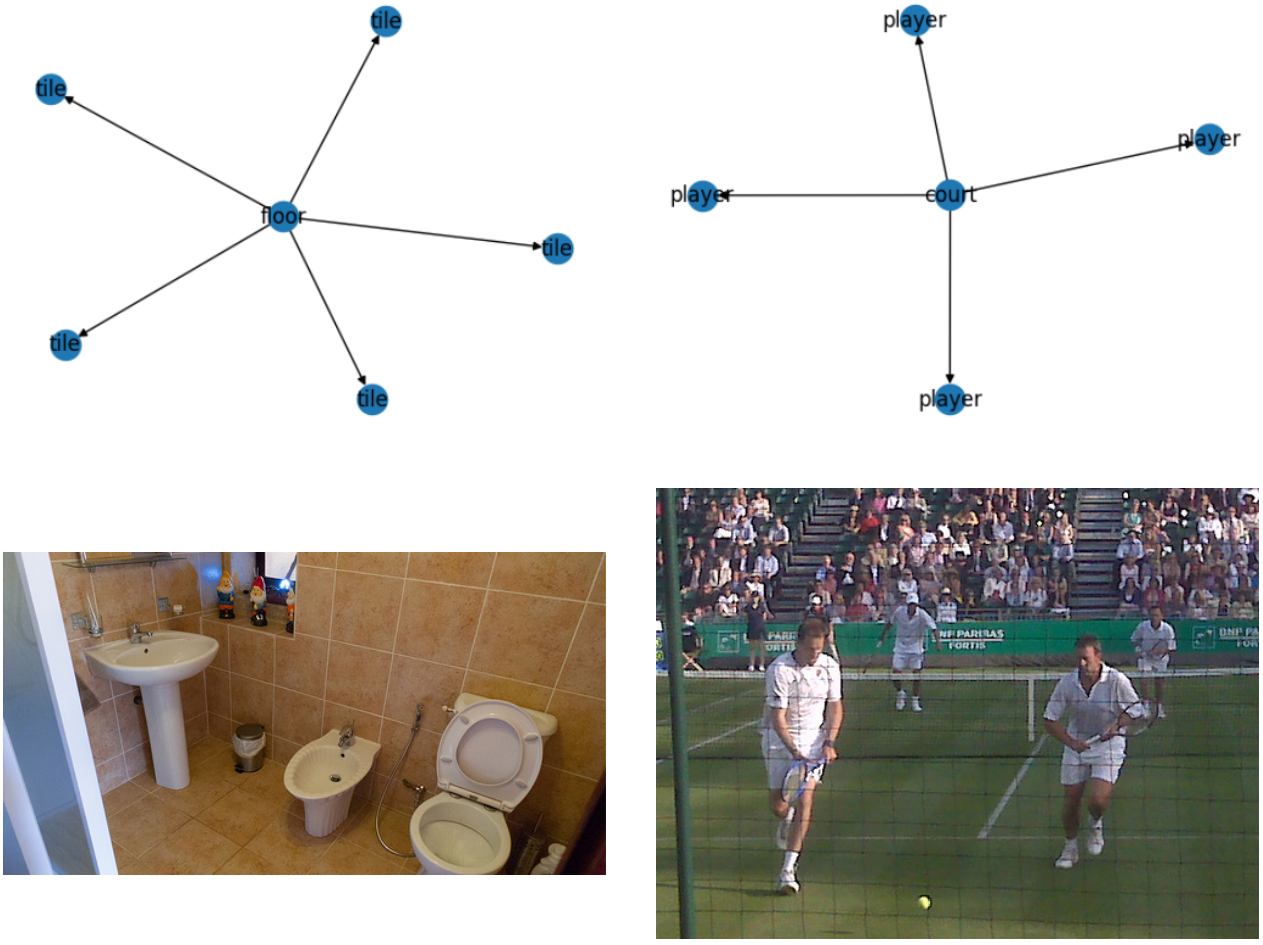


Figure 1.3.8: Εικόνα δύο δομικά όμοιων εικόνων που βαθμολογήθηκαν με σημαντική απόσταση από τον αλγόριθμο Bipartite Matching GED λόγω του διαφορετικού σημασιολογικού τους περιεχομένου.

1.3.4 Πυρήνες Γράφων

Η κυρίαρχη προσέγγιση για την αξιολόγηση της ομοιότητας γράφων περιλαμβάνει τη χρήση πυρήνων γράφων. Κατά συνέπεια, διάφοροι πυρήνες, όπως οι Weisfeiler-Lehman (WL), Shortest Path, Neighborhood Hash, Random Walk και Graphlet Sampling Kernel, αξιολογήθηκαν στο υποσύνολο των γράφων μας χρησιμοποιώντας τη βιβλιοθήκη GraKeL Python [75].

Οι λειτουργίες των προαναφερθέντων πυρήνων περιγράφονται λεπτομερώς στο υποκεφάλαιο 5.2.2. Όπως και η βασική αλήθεια, αυτοί οι πυρήνες δημιουργούν πίνακες ομοιότητας για κάθε ζεύγος γραφημάτων. Αξίζει να σημειωθεί ότι, σε αντίθεση με τους πυρήνες γραφημάτων, ο GED χρησιμεύει ως μέτρο ανομοιότητας, προσφέροντας βαθμολογίες που υποδηλώνουν τον βαθμό ανομοιότητας μεταξύ ζευγών γραφημάτων.

Η βιβλιοθήκη GraKeL [75] προσφέρει μια φιλική προς το χρήστη διεπαφή για τον προσδιορισμό των τιμών των διαφόρων παραμέτρων που χρησιμοποιούνται σε διάφορες παραλλαγές των πυρήνων γράφων. Οι παράμετροι που χρησιμοποιήσαμε για κάθε έναν από τους πέντε πυρήνες γράφων είναι οι εξής:

Weisfeiler-Lehman Kernel: Ο πυρήνας Weisfeiler-Lehman λαμβάνει δύο κύριες παραμέτρους. Η πρώτη είναι ο βασικός πυρήνας k που χρησιμοποιείται μεταξύ δύο γράφων και η δεύτερη είναι ο αριθμός των επαναλήψεων h που χρησιμοποιούνται. Ορίζουμε τον αριθμό των επαναλήψεων σε 20 και χρησιμοποιούμε τον προεπιλεγμένο βασικό πυρήνα γραφήματος, ο οποίος είναι ο πυρήνας ιστογράμματος κορυφών.

Random Walk Kernel: Για τον πυρήνα τυχαίου περιπάτου, χρησιμοποιήσαμε την παραλλαγή πυρήνα RandomWalkLabeled, η οποία λαμβάνει επίσης υπόψη τις ετικέτες των κόμβων. Χρησιμοποιήθηκαν ο προεπιλεγ-

μένος τύπος πυρήνα και οι παράμετροι lambda, δηλαδή γεωμετρικός με τιμή lambda (λ) 0.1. Ο γεωμετρικός προσδιορίζει απλώς τον τρόπο με τον οποίο θα πραγματοποιηθεί η εσωτερική άθροιση.

Shorest Path Kernel: Για τον πυρήνα συντομότερων μονοπατιών, πρέπει να καθοριστούν δύο κύριες παράμετροι. Πρώτον, η πιο κρίσιμη παράμετρος ονομάζεται `algorithm_type`, η οποία καθορίζει τον υποκείμενο αλγόριθμο που χρησιμοποιείται για τον υπολογισμό των συντομότερων μονοπατιών. Μπορεί κανείς να επιλέξει μεταξύ των αλγορίθμων Dijkstra ή Floyd-Warshall ή να επιλέξει την αυτόματη επιλογή, η οποία επιλέγει αυτόματα την ταχύτερη επιλογή με βάση τους γράφους εισόδου. Η άλλη παράμετρος συμβολίζεται ως `with_labels`, καθορίζοντας αν οι κόμβοι έχουν ετικέτες ή όχι. Και για τις δύο παραμέτρους, χρησιμοποιήσαμε την προεπιλεγμένη ρύθμιση του αλγορίθμου Auto και `True` για το `with_labels`.

Neighborhood Hash Kernel: Για τον πυρήνα Neighborhood Hash, πρέπει να καθοριστούν δύο σημαντικές παράμετροι, οι οποίες ονομάζονται R και `bytes` και αντιπροσωπεύουν τον μέγιστο αριθμό επαναλήψεων και το μέγεθος byte των hash των κόμβων, αντίστοιχα. Επιλέξαμε τις προεπιλεγμένες τιμές και για τις δύο παραμέτρους, θέτοντάς τες στις τιμές 3 και 2.

Graphlet Sampling Kernel: Η κύρια παράμετρος για αυτόν τον πυρήνα είναι το μέγιστο μέγεθος των graphlets που θα χρησιμοποιηθούν, και το θέσαμε στην προεπιλεγμένη τιμή, η οποία είναι 5.

1.3.5 Νευρωνικά Δίκτυα Γράφων

Προκειμένου να υλοποιήσουμε και να αξιολογήσουμε τα προτεινόμενα μοντέλα, χρησιμοποιήσαμε τη βιβλιοθήκη εκμάθησης γράφων Pytorch Geometric [30] και χρησιμοποιήσαμε επίσης ορισμένα εργαλεία από τη βιβλιοθήκη PyGCL [96].

Inductive Learning: Το inductive learning είναι ένας τύπος μάθησης όπου το σύστημα προσπαθεί να μάθει τα υποκείμενα μοτίβα ή κανόνες από τα δεδομένα εκπαίδευσης προκειμένου να κάνει προβλέψεις για μελλοντικά δεδομένα που δεν έχει δει κατά την εκπαίδευση. Στόχος σε αυτή την περίπτωση είναι η γενίκευση από συγκεκριμένες περιπτώσεις σε γενικούς κανόνες ή πρότυπα. Μόλις το μοντέλο εκπαιδευτεί, μπορεί να χρησιμοποιηθεί για να κάνει προβλέψεις σε νέα δεδομένα.

Transductive Learning: Το transductive learning είναι ένας τύπος μάθησης όπου το σύστημα προσπαθεί να προβλέψει τις ετικέτες ή τις ιδιότητες συγκεκριμένων, μη επισημασμένων περιπτώσεων με βάση τα διαθέσιμα επισημασμένα δεδομένα. Σε αντίθεση με το inductive learning, το transductive learning δεν στοχεύει στην εκμάθηση ενός γενικού μοντέλου της υποκείμενης κατανομής δεδομένων. Αντ' αυτού, επικεντρώνεται στην πραγματοποίηση προβλέψεων για τις συγκεκριμένες μη επισημειωμένες περιπτώσεις που παρέχονται.

Στα πειράματα που πραγματοποιήθηκαν έγιναν τόσο σε inductive όσο και σε transductive settings. Συγκεκριμένα, τα αρχικά πειράματα που παρουσιάζονται εμπίπτουν στην περίπτωση του inductive learning, όπου εκπαιδεύουμε τα μοντέλα μας σε ένα τυχαία επιλεγμένο σύνολο γραφημάτων σκηνών. Στη συνέχεια, η αξιολόγηση πραγματοποιείται τόσο στο αρχικό τυχαίο σύνολο όσο και στο αρχικό πυκνό σύνολο για την αξιολόγηση της γενίκευσης των μοντέλων σε γραφήματα σκηνών με διαφορετικά χαρακτηριστικά.

Επιπλέον, το προτεινόμενο fine tuning επικεντρώνεται στην ενίσχυση της απόδοσης των μοντέλων για ένα συγκεκριμένο μη επισημασμένο σύνολο γραφημάτων που παρέχεται (π.χ. το υποσύνολο Dense) χρησιμοποιώντας μόνο ένα μικρό ποσοστό ετικετών, εμπίπτοντας έτσι στην κατηγορία του transductive learning. Αυτή η προσέγγιση επιτρέπει την προσαρμογή των αρχικών προ-εκπαιδευμένων μοντέλων σε συγκεκριμένα σύνολα γραφημάτων σκηνής με ιδιαίτερα χαρακτηριστικά.

Για τα αρχικά inductive πειράματα, δοκιμάσαμε ξεχωριστά και τις τέσσερις παραλλαγές που παρουσιάσαμε: GCN, GAT, GATv2 και GIN, με κάθε μία από τις τρεις contrastive προσεγγίσεις: GraphCL, InfoGraph και Grace. Στη συνέχεια, για τη διαδικασία του fine tuning, πειραματιστήκαμε με τους πιο επιτυχημένους συνδυασμούς.

Για τις υπερπαραμέτρους των μοντέλων GNN, θα απαριθμήσουμε αρχικά εκείνες που ήταν κοινές σε όλες τις παραλλαγές. Χρησιμοποιήσαμε τον βελτιστοποιητή Adam για την εκπαίδευση όλων των μοντέλων και πραγματοποιήσαμε επίσης πειράματα με τις παραμέτρους του βελτιστοποιητή, ιδίως με τον ρυθμό μάθησης και την αποσύνθεση των βαρών. Μετά από προσεκτικό πειραματισμό, ορίσαμε τον ρυθμό μάθησης εξίσου στο $1e-4$ για όλα τα μοντέλα.

Πειραματιστήκαμε επίσης με τον αριθμό των χρησιμοποιούμενων στρωμάτων και καταλήξαμε στο συμπέρασμα

ότι τα μοντέλα με 2 στρώματα και συνένωση των αποτελεσμάτων του πρώτου και του δεύτερου στρώματος υπερτερούν έναντι άλλων συνδυασμών. Μια άλλη σημαντική παράμετρος είναι η διαστατικότητα των τελικών αναπαραστάσεων των γράφων. Διαπιστώσαμε ότι η βέλτιστη λύση ήταν να είναι 512.

Επιπλέον, οι συνελικτικές παραλλαγές που χρησιμοποιούν την προσοχή πολλαπλών κεφαλών απαιτούν ιδιαίτερη μέριμνα. Πραγματοποιήθηκαν πειράματα με 1, 2 και 4 κεφαλές, μεταξύ των οποίων η διαμόρφωση με 4 κεφαλές έδωσε σταθερά τα καλύτερα αποτελέσματα.

Τέλος, οι υπερπαράμετροι των εποχών, του μεγέθους της παρτίδας και των επαυξήσεων είναι ιδιαίτερα σημαντικές για τα Αντιθετικά Νευρωνικά Δίκτυα Γράφων. Γενικά, η αντιθετική μάθηση βασίζεται στο να φέρνει κοντά θετικά δειγμάτα (παρόμοια ή σημασιολογικά συναφή δείγματα) στο χώρο αναπαράστασης, ενώ τα αρνητικά δείγματα (ανόμοια δείγματα) απομακρύνονται περισσότερο. Αυτά τα θετικά και αρνητικά δείγματα παίζουν καθοριστικό ρόλο στην ενθάρρυνση του μοντέλου να μάθει ουσιαστικές αναπαραστάσεις, που συνήθως λαμβάνονται από το εσωτερικό κάθε παρτίδας. Ως εκ τούτου, το μέγεθος της παρτίδας είναι μια ζωτικής σημασίας υπερπαράμετρος σε αυτό το πλαίσιο. Σε γενικές γραμμές, η αντιθετική μάθηση τείνει να επωφελείται από μεγαλύτερα μεγέθη παρτίδων, αν και είναι σημαντικό να επιτευχθεί μια ισορροπία. Ένα μέγεθος παρτίδας που δεν είναι ούτε πολύ μικρό ούτε πολύ μεγάλο είναι το βέλτιστο για την επίτευξη ικανοποιητικών αποτελεσμάτων μάθησης.

Όσον αφορά τις τεχνικές επαύξησης δεδομένων, μέσω πειραματισμού είδαμε ότι η αφαίρεση κόμβων και ακμών, μαζί με τη συγκάλυψη χαρακτηριστικών, έδωσαν τα καλύτερα αποτελέσματα. Για την παραλλαγή του GraphCL, επιλέξαμε τελικά την αφαίρεση κόμβων και ακμών. Ωστόσο, για την παραλλαγή Grace, αποφασίσαμε την αφαίρεση ακμών και τη συγκάλυψη χαρακτηριστικών για να διατηρήσουμε την απαραίτητη αντιστοίχιση των κόμβων που είναι απαραίτητη για το Local-Local αντιθετικό σφάλμα. Σε γενικές γραμμές, παρατηρήσαμε ότι η χρήση μικρών ποσοστών για την επαύξηση χαρακτηριστικών ήταν πιο ωφέλιμη για την κατασκευή θετικών δειγμάτων, σε αντίθεση με τα μεγάλα ποσοστά που έτειναν να οδηγούν σε σημαντική αλλοίωση του αρχικού γράφου.

Παρακάτω συνοψίζουμε στον παρεχόμενο πίνακα τους καλύτερους συνδυασμούς που χρησιμοποιήθηκαν για κάθε ένα από τα μοντέλα:

Table 1.1: Βέλτιστες Τιμές Υπερπαραμέτρων

Model	**GNN**	**Epochs**	**Batch Size**	**Augmentations**	**Attention Heads**
GraphCL	GCN	50	64	Node/Edge Dropout $p = 0.05$	-
	GIN	40	128	Node/Edge Dropout $p = 0.05$	-
	GAT	20	128	Node/Edge Dropout $p = 0.05$	4
	GATv2	20	128	Node/Edge Dropout $p = 0.05$	4
InfoGraph	GIN	40	128	-	-
Grace	GCN	50	128	Node Dropout/Feature Masking $p = 0.15$	-
	GIN	80	128	Node Dropout/Feature Masking $p = 0.15$	-
	GAT	20	16	Node Dropout/Feature Masking $p = 0.15$	4
	GATv2	40	16	Node Dropout/Feature Masking $p = 0.15$	4

1.3.6 Ποσοτικά Αποτελέσματα

Αρχικά, θα παρουσιάσουμε τις τελικές τιμές MAP, MRR και NDCG που πέτυχαν τόσο οι Graph Kernels όσο και τα μοντέλα ΝΔΓ στο αρχικό τυχαίο σύνολο. Κάθε συγκεκριμένο μοντέλο ΝΔΓ αντιστοιχεί στο βέλτιστο μοντέλο με υπερπαραμέτρους που περιγράφονται αναλυτικά στον πίνακα 9.1. Όλα τα ΝΔΓ εκπαιδεύτηκαν σε ένα ξεχωριστό τυχαίο υποσύνολο 1000 γραφημάτων σκηνής. Οι Μετρικές Αξιολόγησης εμφανίζονται στους Πίνακες 9.2 και 9.3. Στον πρώτο, παρατηρούμε τις επιδόσεις των πυρήνων γράφων και των μη επιβλεπόμενων ΝΔΓ που εκπαιδεύονται αντιθετικά, ενώ στον δεύτερο, παρατηρούμε τις επιδόσεις των καλύτερων μοντέλων ΝΔΓ μετά το rank aware fine tuning που εκθέτει τα μοντέλα σε ένα μικρό ποσοστό της βασικής αλήθειας.

Η πρώτη παρατήρηση που κάνουμε για τους πυρήνες γράφων είναι η υπεροχή του πυρήνα συντομότερων μονοπατιών στις μετρικές MAP@10 και MRR@10, μαζί με τον πυρήνα Neighborhood Hash, ο οποίος έχει ανταγωνιστικές επιδόσεις σε όλες τις μετρικές και υπερτερεί όλων των πυρήνων στην NDCG@10. Οι άλλοι τρεις

Table 1.2: Τελικές τιμές μετρικών για τους πυρήνες γράφων και τα μοντέλα ΝΔΓ χωρίς fine tuning, στο αρχικό τυχαίο υποσύνολο σε inductive setting. Η εκπαίδευση έχει πραγματοποιηθεί σε ένα άλλο τυχαίο υποσύνολο. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε αρχιτεκτονική.

	MAP@10	**MRR@10**	**NDCG@10**
Shortest Path Kernel	**0.1148**	**0.4862**	0.6851
Random Walk Kernel	0.0508	0.2509	0.5971
Weisfeiler-Lehman Kernel	0.0995	0.4315	0.6560
Graphlet Sampling Kernel	0.1030	0.3810	0.6296
Neighborhood Hash Kernel	0.1035	0.4733	**0.7112**
GraphCL			
GCN	0.1123	0.4213	0.6737
GIN	**0.1157**	**0.4614**	**0.6882**
GAT	0.0919	0.4157	0.6788
GATv2	0.0933	0.4068	0.6805
InfoGraph			
GIN	0.1229	0.4649	0.6982
Grace			
GCN	0.1016	0.4291	0.6819
GIN	0.1122	0.4550	0.6872
GAT	**0.1243**	0.4769	0.7479
GATv2	0.1223	**0.4892**	**0.7504**

Table 1.3: Τελικές τιμές μετρικών για τα καλύτερα μοντέλα ΝΔΓ με fine tuning, στο αρχικό τυχαίο υποσύνολο με εποπτεία από ένα μικρό υποσύνολο της βασικής αλήθειας. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε μετρική.

	MAP@10		**MRR@10**		**NDCG@10**	
	10 epochs	**30 epochs**	**10 epochs**	**30 epochs**	**10 epochs**	**30 epochs**
GraphCL						
GIN	0.1431	0.1881	0.5271	0.5624	0.7436	0.8009
InfoGraph						
GIN	0.1738	0.2164	0.5311	0.5717	0.7747	0.8120
Grace						
GAT	**0.1957**	0.2177	**0.5864**	0.5840	**0.8117**	0.8176
GATv2	0.1935	**0.2350**	0.5616	**0.6005**	0.8075	**0.8227**

πυρήνες ξεπερνίονται από αυτούς τους δύο και στις τρεις μετρικές. Η κακή απόδοση του πυρήνα Random Walk μπορεί να εξηγηθεί αν τον συσχετίσουμε με την υποκείμενη λειτουργικότητά του. Συγκεκριμένα, αυτός ο πυρήνας βασίζεται σε θεμελιώδεις δομικές ιδιότητες ενός γραφήματος, ιδίως σε τυχαίους περιπάτους. Ωστόσο, οι γράφοι σκηνής είναι συνήθως μικροί γράφοι και επιπλέον, το τυχαίο υποσύνολο είναι αρκετά αραιό. Κατά συνέπεια, οι τυχαίοι περίπατοι δεν είναι τόσο κατατοπιστικοί για τους γράφους σκηνών. Ομοίως, ο πυρήνας graphlet sampling βασίζεται και αυτός σε δομικές ιδιότητες του γράφου, συγκεκριμένα στην εμφάνιση πρωτοτύπων (graphlets), και για το λόγο αυτό δεν αποδίδει ικανοποιητικά. Από την άλλη πλευρά, οι πυρήνες Weisfeiler-Lehman και Neighborhood Hash βασίζονται στις ετικέτες των γραφημάτων και υλοποιούν μια διαδικασία επαναετικετοποίησης των κόμβων και ενημέρωσης των ετικετών τους με βάση τις ετικέτες των γειτόνων. Αυτή η διαδικασία μοιάζει πολύ με το σχήμα μεταβίβασης μηνυμάτων που χρησιμοποιείται από την πλειονότητα των ΝΔΓ. Αυτή η προσέγγιση φαίνεται να είναι αρκετά επιτυχής. Ο πυρήνας Neighborhood Hash έχει πράγματι τις καλύτερες επιδόσεις όσον αφορά το NDCG@10 με διαφορά 3% σε σύγκριση με τον πυρήνα Shortest Path Kernel και έχει τις δεύτερες καλύτερες βαθμολογίες MRR@10 και MAP@10. Ωστόσο, ο πυρήνας Weisfeiler-Lehman είναι τρίτος όσον αφορά τα MRR@10 και NDCG@10 και μόνο τέταρτος στον MAP@10, αλλά βρίσκεται κοντά στις τρεις πρώτες θέσεις. Περιμέναμε να έχει καλύτερες επιδόσεις, αλλά είναι γενικά επαρκώς ανταγωνιστικός σε σχέση με τους άλλους πυρήνες.

Όσον αφορά τα μοντέλα ΝΔΓ, η αρχική παρατήρηση είναι ότι όλα τους είναι επαρκώς ανταγωνιστικά έναντι αρκετών πυρήνων γράφων. Ωστόσο, δυσκολεύονται να ξεπεράσουν τις καλύτερες βαθμολογίες των βασικών πυρήνων γράφων, ιδίως όσον αφορά τη μετρική NDCG, καθώς και τις άλλες δύο μετρικές. Ωστόσο, στο πλαίσιο αντιθετικής μάθησης Grace, παρατηρούμε ότι η χρήση των παραλλαγών προσοχής GAT και GATv2 μπορεί να ξεπεράσει τους καλύτερους πυρήνες σε όλες τις μετρικές. Συγκεκριμένα, όταν συνδυάζεται με την παραλλαγή GATv2, υπερτερεί του Neighborhood Hash Kernel με περιθώριο 4% στη μετρική NDCG@10.

Προχωρώντας σε πιο σύνθετες αρχιτεκτονικές, μπορούμε να δούμε ότι το Rank Aware Fine Tuning, χρησιμοποιώντας ένα μικρό ποσοστό της βασικής αλήθειας του αρχικού Random Set, παρέχει σημαντική βελτίωση και στις τρεις μετρικές, όπως φαίνεται από τον Πίνακα 9.3. Ας προσδιορίσουμε πρώτα το ποσοστό της βασικής αλήθειας που αντιστοιχεί σε 10 και 30 εποχές, στο χειρότερο σενάριο. Σε κάθε εποχή, για κάθε ένα από τα 1000 γραφήματα, λαμβάνονται 2 γραφήματα ως θετικά και αρνητικά δείγματα. Συνεπώς, μία εποχή εκθέτει το μοντέλο σε 2000 τιμές βασικής αλήθειας. Ο συνολικός αριθμός ζευγών βασικής αλήθειας για 1000 γραφήματα είναι 500.000. Ως αποτέλεσμα, 10 εποχές αντιστοιχούν στη χρήση του 4% της βασικής αλήθειας και 30 εποχές αντιστοιχούν στη χρήση του 12% της βασικής αλήθειας. Έτσι, χρησιμοποιώντας ένα μικρό ποσοστό μόνο της βασικής αλήθειας, είμαστε σε θέση να επιτύχουμε τιμές MAP@10, MRR@10 και NDCG@10 που είναι πάνω από 10% υψηλότερες σε σύγκριση με τα καλύτερα αποτελέσματα από τους πυρήνες γράφων. Αυτά τα αποτελέσματα επιτεύχθηκαν με τη χρήση των πιο υποσχόμενων συνδυασμών, ιδίως του πλαισίου Grace με τις παραλλαγές ΝΔΓ GAT/GATv2 και fine tuning για 30 εποχές. Επιπλέον, 10 εποχές είναι επίσης αρκετές προκειμένου να δει κανείς μια σημαντική βελτίωση στις μετρικές.

Προκειμένου να δούμε πώς τα μοντέλα μας αποδίδουν σε ένα άλλο σύνολο γράφων σκηνών με συγκεκριμένα και πιο περιορισμένα χαρακτηριστικά, αποφασίσαμε να δοκιμάσουμε τα καλύτερα μοντέλα ΝΔΓ εκπαιδευμένα χωρίς επίβλεψη στο αρχικό Πυκνό Σύνολο. Οι τελικές βαθμολογίες MAP, MRR και NDCG που πέτυχαν τόσο οι πυρήνες γράφων όσο και τα μοντέλα ΝΔΓ στο αρχικό πυκνό σύνολο εμφανίζονται στους πίνακες 9.2. Στον πίνακα 9.3 παρατηρούμε τις επιδόσεις των καλύτερων μοντέλων ΝΔΓ μετά τη διαδικασία του fine tuning, η οποία τα εκθέτει σε ένα μικρό ποσοστό της βασικής αλήθειας του Πυκνού Συνόλου αυτή τη φορά.

Στο πυκνό σύνολο, ο πυρήνας Weisfeiler-Lehman επιτυγχάνει τις υψηλότερες τιμές στις μετρικές MAP@10 και NDCG@10, ακολουθούμενος από τους πυρήνες Shortest Path και Neighborhood Hash, οι οποίοι επίσης έχουν ανταγωνιστικές επιδόσεις. Για άλλη μια φορά, οι πυρήνες Graphlet Sampling και Random Walk Kernel έχουν αρκετά χαμηλές επιδόσεις. Είναι προφανές για άλλη μια φορά ότι τα Αντιθετικά ΝΔΓ, που εκπαιδεύονται σε ένα τυχαίο σύνολο, μπορούν να γενικεύσουν αρκετά καλά στο πυκνό σύνολο και έχουν αρκετά ανταγωνιστικές επιδόσεις, ξεπερνώντας ακόμη και τους καλύτερους πυρήνες με μια μικρή διαφορά 1% στο NDCG@10, 3% στο MRR@10 και 1% στο MAP@10.

Επιπλέον, μπορούμε να δούμε ότι το Rank Aware Fine Tuning παρέχει πάλι σημαντική ώθηση και στις τρεις μετρικές, όπως φαίνεται από τον πίνακα 9.7. Χρησιμοποιώντας ένα μικρό ποσοστό της βασικής αλήθειας, είμαστε σε θέση να επιτύχουμε τιμές των MAP@10, MRR@10 και NDCG@10 που είναι περίπου 9-10% υψηλότερες σε σύγκριση με τα καλύτερα αποτελέσματα από τους πυρήνες γράφων.

Table 1.4: Τελικές τιμές μετρικών για τους πυρήνες γράφων και τα καλύτερα προ-εκπαιδευμένα μοντέλα ΝΔΓ χωρίς fine tuning, στο αρχικό πυκνό υποσύνολο σε inductive setting. Η εκπαίδευση πραγματοποιήθηκε σε ένα τυχαίο υποσύνολο γράφων. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε αρχιτεκτονική.

	MAP@10	**MRR@10**	**NDCG@10**
Shortest Path Kernel	0.1237	**0.4618**	0.6171
Random Walk Kernel	0.0928	0.3913	0.5964
Weisfeiler-Lehman Kernel	**0.1253**	0.4570	**0.6279**
Graphlet Sampling Kernel	0.0389	0.0965	0.5317
Neighborhood Hash Kernel	0.1034	0.3868	0.6132
GraphCL			
GCN	0.1281	0.4724	0.6230
GIN	**0.1339**	**0.4914**	**0.6302**
InfoGraph			
GIN	0.1354	0.4882	0.6282
Grace			
GAT	0.1152	0.4715	0.6285
GATv2	**0.1280**	**0.4902**	**0.6394**

Table 1.5: Τελικές τιμές μετρικών για τα καλύτερα μοντέλα ΝΔΓ μετά από fine tuning, στο αρχικό πυκνό υποσύνολο με εποπτεία από ένα μικρό υποσύνολο της βασικής αλήθειας. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε μετρική.

	MAP@10		**MRR@10**		**NDCG@10**	
	10 epochs	**30 epochs**	**10 epochs**	**30 epochs**	**10 epochs**	**30 epochs**
GraphCL						
GIN	0.1670	0.1990	0.5292	0.5784	0.6713	0.7006
InfoGraph						
GIN	0.1605	0.2003	0.5322	0.5657	0.6735	0.7001
Grace						
GAT	0.1723	0.2111	0.5247	0.5688	0.6834	0.7073
GATv2	**0.1732**	**0.2207**	**0.5537**	**0.5915**	**0.6876**	**0.7140**

Προκειμένου να δικαιολογήσουμε την επιλογή μας να προ-εκπαιδεύσουμε πρώτα τα ΝΔΓ αντιθετικά σε ένα τυχαίο σύνολο και στη συνέχεια να προχωρήσουμε στο fine tuning στο επιλεγμένο πυκνό σύνολο, παρέχουμε τον Πίνακα 9.6, όπου μπορεί κανείς να παρατηρήσει την απόδοση των ΝΔΓ μόνο με τη χρήση fine tuning και χωρίς προεκπαίδευση. Αυτά τα ΝΔΓ εκπαιδεύτηκαν μόνο για 30 εποχές χρησιμοποιώντας το προτεινόμενο Rank Aware σφάλμα, χωρίς αντιθετική προεκπαίδευση. Βλέπουμε ότι αυτά τα μοντέλα υπερτερούν επιτυχώς των πυρήνων γράφων. Ωστόσο, υπολείπονται με σημαντική διαφορά των αντίστοιχων μοντέλων που έχουν προ-εκπαιδευτεί αντιθετικά και στη συνέχεια έχουν κάνει fine-tuning για τον ίδιο αριθμό εποχών (π.χ. 30 εποχές), αναδεικνύοντας έτσι τα πλεονεκτήματα της προσέγγισής μας.

Table 1.6: Τελικές τιμές μετρικών για τα καλύτερα μοντέλα ΝΔΓ που εκπαιδεύτηκαν για 30 εποχές μόνο με το loss του fine tuning. Η εξαγωγή συμπερασμάτων πραγματοποιείται στο αρχικό πυκνό υποσύνολο.

	MAP@10	**MRR@10**	**NDCG@10**
	30 epochs	**30 epochs**	**30 epochs**
GIN			
GIN	0.1704	0.5318	0.6773
GAT			
GAT	0.1624	0.5208	0.6739
GATv2			
GATv2	0.1651	0.5293	0.6732

Συμπερασματικά, τα αντιθετικά ΝΔΓ αποδεικνύονται μια εξαιρετικά αποτελεσματική λύση για το πρόβλημα της ομοιότητας γράφων σκηνής. Επιπλέον, είναι σημαντικό να σημειωθεί ότι το μεγαλύτερο πλεονέκτημα αυτών των ΝΔΓ έναντι των Πυρήνων Γράφων είναι ότι παρέχουν αναπαραστάσεις γενικής χρήσης που μπορούν να χρησιμοποιηθούν για διάφορες εργασίες όπως ταξινόμηση κόμβων και γράφων.

Τέλος, παρουσιάζουμε τους χρόνους εκπαίδευσης για την αντιθετική προεκπαίδευση στο τυχαίο σύνολο και την αντιθετική προεκπαίδευση ακολουθούμενη από fine tuning στο πυκνό σύνολο.

Table 1.7: Τελικοί χρόνοι εκπαίδευσης για τα καλύτερα μοντέλα ΝΔΓ με fine tuning, στο αρχικό πυκνό υποσύνολο με επίβλεψη από ένα μικρό υποσύνολο της βασικής αλήθειας.

	Χρόνος Αντιθετικής Προεκπαίδευσης	**Χρόνος Προεκπαίδευσης + Fine Tuning**	
		10 epochs	**30 epochs**
GraphCL			
GIN	32sec	44sec	1min 05sec
InfoGraph			
GIN	50sec	1min 04sec	1min 46sec
Grace			
GAT	2min 12sec	2min 50sec	4min 02sec
GATv2	4min 28sec	5min 36sec	6min 39sec

Οι υψηλότεροι χρόνοι εκπαίδευσης για τις παραλλαγές GAT είναι κάτι που περιμέναμε, καθώς η εκπαίδευση πολλαπλών κεφαλών προσοχής απαιτεί επιπλέον χρόνο. Επιπλέον, το πλαίσιο Grace είναι υπολογιστικά λίγο πιο βαρύ σε σχέση με τα υπόλοιπα καθώς εκτελεί την αντιθετική απώλεια για κάθε κόμβο μέσα στο batch (και όχι για κάθε γράφο στο batch). Επιπλέον, λαμβάνοντας υπόψη ότι χρησιμοποιούμε 4 % ή 12 % της βασικής αλήθειας, το υπολογιστικό trade-off είναι σίγουρα υπέρ μας, καθώς χρειαζόμαστε μόνο περίπου 4 % ή 12 % του χρόνου που απαιτείται για τον συνολικό υπολογισμό της βασικής αλήθειας για να ενισχύσουμε σημαντικά τις μετρικές μας εάν επιλέξουμε να τελειοποιήσουμε τα μοντέλα με χρήση fine tuning.

1.3.7 Ποιοτικά Αποτέλεσματα

Για την ποιοτική αξιολόγηση των προτεινόμενων μοντέλων, παρέχουμε τις εικόνες που αντιστοιχούν στους γράφους σκηνών που συστήνουν ως πιο όμοιους τόσο τα ΝΔΓ όσο και οι Πυρήνες Γράφων ως απόκριση σε συγκεκριμένα ερωτήματα. Συγκεκριμένα, παρουσιάζουμε μερικές εικόνες ερωτημάτων μαζί με τις πρώτες εικόνες που ανακτήθηκαν στο πυκνό σύνολο από το καλύτερο μοντέλο μας, το Grace με κωδικοποιητή GATv2 που έχει προεκπαιδευτεί σε ένα τυχαίο σύνολο 1000 γράφων και στη συνέχεια έχει γίνει fine tuned στο πυκνό σύνολο για 30 εποχές.

Figure 1.3.9: Εικόνα αναζήτησης

Figure 1.3.10: Τοπ 3 αποτελέσματα - Μοντέλο GNN

Figure 1.3.11: Τοπ 3 αποτελέσματα - Καλύτερος πυρήνας

Figure 1.3.12: Εικόνα αναζήτησης

Figure 1.3.13: Τοπ 3 αποτελέσματα - Μοντέλο GNN

Figure 1.3.14: Τοπ 3 αποτελέσματα - Καλύτερος πυρήνας

Figure 1.3.15: Εικόνα αναζήτησης

Figure 1.3.16: Τοπ 3 αποτελέσματα - Μοντέλο GNN

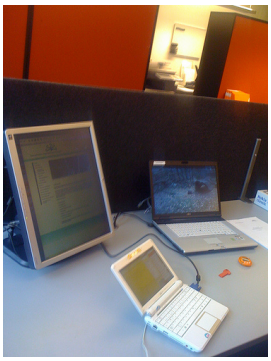


Figure 1.3.17: Τοπ 3 αποτελέσματα - Καλύτερος πυρήνας

Figure 1.3.18: Εικόνα αναζήτησης

Figure 1.3.19: Τοπ 3 αποτελέσματα - Μοντέλο GNN



Figure 1.3.20: Τοπ 3 αποτελέσματα - Καλύτερος πυρήνας



Αυτά τα ποιοτικά αποτελέσματα επιβεβαιώνουν την υπεροχή του προτεινόμενου μοντέλου μας, η οποία έχει αποδειχθεί και ποσοτικά. Φαίνεται ότι το μοντέλο μας μπορεί να ανακτήσει εικόνες με πιο κοντινό σημασιολογικό περιεχόμενο από τους καλύτερους πυρήνες γράφων σε πολλές και διαφορετικές περιπτώσεις.

1.4 Συμπεράσματα

1.4.1 Συζήτηση

Σε αυτή τη διατριβή, αντιμετωπίσαμε το πρόβλημα της ομοιότητας γραφημάτων μέσα από το πρίσμα της αντιθετικής μάθησης. Τα πειράματά μας αποσκοπούσαν στην αξιολόγηση της αποτελεσματικότητας των προτεινόμενων μοντέλων Αντιθετικών ΝΔΓ σε σύγκριση με βασικούς Πυρήνες Γράφων, στο σύνολο δεδομένων Visual Genome. Μέσω μιας συστηματικής διερεύνησης διαφορετικών αρχιτεκτονικών ΝΔΓ και πλαισίων αντιθετικής μάθησης, συνοδευόμενης από ενδελεχή αξιολόγηση με τη χρήση μετρικών όπως η Mean Average Precision - MAP, η Mean Reciprocal Rank - MRR και η Normalized Discounted Cumulative Gain - NDCG, σημαντικά συμπεράσματα εξήχθησαν.

Αρχικά, ήταν εμφανής η υπεροχή ορισμένων πυρήνων γράφων, όπως οι Shortest Path, Neighborhood Hash και ο Weisfeiler-Lehman Kernel. Ωστόσο, παρατηρήθηκαν περιορισμοί σε πυρήνες που βασίζονται σε μεγάλο βαθμό σε δομικές ιδιότητες, όπως ο Random Walk Kernel και ο Graphlet Sampling Kernel, γεγονός που υποδηλώνει την ανεπάρκειά τους στην αποτύπωση των περιπλοκών των γραφημάτων σκηνής.

Τα αντιθετικά ΝΔΓ επέδειξαν ιδαίτερη ανταγωνιστικότητα έναντι διαφόρων πυρήνων γράφων, ιδίως οι παραλλαγές προσοχής, όπως το GAT και το GATv2, ιδίως όταν χρησιμοποιήθηκαν στα πλαίσια της αντιθετικής

μάθησης Grace. Αντίθετα, η παραλλαγή Generalized-Isomorphism Network (GIN) επέδειξε αξιοσημείωτη ανταγωνιστικότητα όταν στην αντιθετική απώλεια εμπλέκονταν αναπαραστάσεις γράφων (πλαίσιο GraphCL και InfoGraph). Ο πειραματισμός μας περιλάμβανε τέσσερις διαφορετικές παραλλαγές ΝΔΓ για καθεμία από τις προσεγγίσεις αντιθετικής μάθησης GraphCL, InfoGraph και Grace, αξιολογώντας την αποτελεσματικότητα, την πολυπλοκότητα και τις εκφραστικές τους ικανότητες. Συγκεκριμένα, χρησιμοποιήσαμε τα GCN, GIN, GAT και GATv2. Αυτές οι παραλλαγές ΝΔΓ διευκόλυναν την αναπαράσταση γράφων σε ένα χώρο ικανό να αποτυπώνει τόσο τις δομικές όσο και τις σημασιολογικές ομοιότητες μεταξύ των γράφων. Στη συνέχεια, αυτές οι αναπαραστάσεις χρησιμοποιήθηκαν για τη σύγκριση γραφημάτων, διευκολύνοντας τον εντοπισμό του πιο όμοιου γράφου για κάθε γράφο-ερώτημα. Για τη δημιουργία μιας βασικής αλήθειας, χρησιμοποιήσαμε έναν προσεγγιστικό αλγόριθμο Graph Edit Distance που έχει τις ρίζες του στην αντιστοίχιση διμερών γραφημάτων.

Επιπλέον, η εισαγωγή του Rank Aware Fine Tuning, που χρησιμοποιεί ένα μικρό ποσοστό δεδομένων βασικής αλήθειας, ενίσχυσε σημαντικά τις επιδόσεις σε όλες τις μετρικές. Οι συγκρίσεις μεταξύ των αντιθετικά προεκπαιδευμένων μοντέλων ΝΔΓ και εκείνων χωρίς αντιθετική προεκπαίδευση υπογράμμισαν την αποτελεσματικότητα της προτεινόμενης προσέγγισης, με την αντιθετική προεκπαίδευση ακολουθούμενη από το fine tuning να αποδίδει καλύτερα αποτελέσματα.

Συνοψίζοντας, τα πειράματα τεκμηριώνουν σταθερά την αποτελεσματικότητα των τεχνικών αντιθετικής μάθησης στην εργασία ομοιότητας γράφων, ξεπερνώντας σταθερά τις παραδοσιακές μεθόδους που βασίζονται σε πυρήνες.

Τέλος, αυτά τα εκπαιδευμένα μοντέλα ΝΔΓ μπορούν να χρησιμοποιηθουν για την παροχή Εξηγήσεων με Αντιπαράδειγμα. Συγκεκριμένα, ο εντοπισμός του πιο όμοιου ζεύγους εικόνων μας επιτρέπει να υπολογισούμε ορισμένες μόνο αποστάσεις επεξεργασίας, αντί του υπολογισμού αποστάσεων επεξεργασίας για κάθε ζεύγος του συνόλου δεδομένων. Ως αποτέλεσμα, η προτεινόμενη μέθοδος, η οποία χρησιμοποιεί τον αντίστοιχο γράφο σκηνής μιας εικόνας για την ανάκτηση του σημασιολογικά πιο κοντινού γράφου σκηνής, εισάγει σημαντική επιτάχυνση σε αυτή τη διαδικασία.

1.4.2 Μελλοντικές Κατευθύνσεις

Για μελλοντικές εργασίες, θα μπορούσαν να εξεταστούν διάφορες προσεγγίσεις για την περαιτέρω ενίσχυση της κατανόησης και της εφαρμογής των αντιθετικών νευρωνικών δικτύων γράφων ΝΔΓ στον τομέα της ομοιότητας γράφων σκηνής αλλά και πέραν αυτού. Συγκεκριμένα.

1. **Πειραματισμός με άλλες προσεγγίσεις αντιθετικής μάθησης**: Αυτή η διατριβή επικεντρώνεται στα αντιθετικά ΝΔΓ για εργασίες ομοιότητας γραφημάτων σκηνής. Σε αυτό το πλαίσιο, η διερεύνηση εναλλακτικών τεχνικών αντιθετικής μάθησης θα μπορούσε να προσφέρει πολύτιμες γνώσεις. Παραλλαγές όπως αυτές που βασίζονται στην απώλεια Barlow Twins Loss [10] ή Bootstrapped Graph Latents [79], μεταξύ άλλων, έχουν επιδείξει πολλά υποσχόμενα αποτελέσματα σε δεδομένα γράφων, γεγονός που δικαιολογεί την περαιτέρω διερεύνηση της απόδοσής τους για το έργο της ανάκτησης γραφημάτων σκηνής.

2. **Ενσωμάτωση των χαρακτηριστικών ακμών στα ΝΔΓ**: Εκτός από τις διερευνηθείσες προσεγγίσεις, η ενσωμάτωση χαρακτηριστικών ακμών στα δεδομένα γράφων θα μπορούσε να εμπλουτίσει περαιτέρω την εκφραστική δύναμη των Αντιθετικών ΝΔΓ. Τα χαρακτηριστικά ακμών περιέχουν πολύτιμες πληροφορίες σχετικά με τις σχέσεις μεταξύ των κόμβων σε έναν γράφο, οι οποίες μπορούν να ενισχύσουν σημαντικά την ικανότητα του μοντέλου να καταγράφει σημασιολογικές και δομικές ομοιότητες.

3. **Διαφορετικές Στρατηγικές Fine Tuning**: Η περαιτέρω διερεύνηση άλλων στρατηγικών fine tuning θα μπορούσε να αποφέρει σημαντικά οφέλη. Αυτό περιλαμβάνει τον πειραματισμό με διάφορα ποσοστά βασικής αλήθειας ή τη διερεύνηση εναλλακτικών σφαλμάτων που θα προσέφεραν στα μοντέλα ΝΔΓ επίγνωση της σωστής κατάταξης ομοιότητας των γράφων. Επιπλέον, θα πρέπει να διεξαχθούν πειράματα με τη χρήση είτε των ίδιων είτε διαφορετικών rank-aware σφαλμάτων με στόχο να καταστούν τα μοντέλα ικανά να αποδίδουν εξίσου καλά ή ακόμη καλύτερα και σε inductive settings.

Chapter 2

Introduction

Recent developments in artificial intelligence (AI) have transformed various domains, necessitating advanced techniques for complex data structures like graphs. Graph Neural Networks (GNNs) have emerged as pivotal tools for modeling relationships within graph data, with applications spanning social networks, recommendation systems, and bioinformatics. Traditional neural networks are ill-suited for graph data, emphasizing the need for specialized models like GNNs capable of capturing hierarchical relationships inherent in interconnected data points.

The focus of this thesis lies in addressing the Graph Similarity problem, especially for scene graphs, wherein the task is to rank answer graphs based on their similarity to a query graph. While Graph Kernels have been popular for this task due to their ease of implementation and computational efficiency, we propose employing GNNs, particularly Contrastive Graph Neural Networks, to tackle this problem. By leveraging the representation power of Contrastive GNNs, we aim to enhance the accuracy and efficiency of graph similarity computations.

Moreover, the increasing demand for Explainable AI (XAI) has become critical, especially in sectors like healthcare and autonomous vehicles where AI systems are extensively utilized. Counterfactual Explanations have emerged as a promising method for elucidating black-box AI systems. By highlighting the minimal alterations needed to shift to an alternate outcome, GNN-based counterfactual explanations enhance transparency and comprehension of AI decision-making processes.

In this thesis, we delve into the utility of Contrastive GNNs in providing Counterfactual Explanations. We thoroughly investigate Contrastive Graph Neural Network frameworks, such as GraphCL, InfoGraph, and Grace, alongside various GNN variants. We then evaluate their expressive capabilities using scene graph data. The application of Contrastive GNNs for scene graph similarity and retrieval is closely linked to the realm of XAI. Specifically, the ability of Contrastive GNNs to pinpoint the most similar instances, and consequently, the least semantically distant ones within a dataset of images represented as scene graphs, plays a pivotal role in generating counterfactual explanations for visual classifiers.

The outline of this thesis is as follows:

- We will begin by furnishing all the necessary foundational knowledge in Machine Learning algorithms and concepts, encompassing Graph Theory, Deep Learning theory, various Graph Kernel variants, multiple Graph Neural Network architectures, and approximations of Graph Edit Distance.
- Next, we will provide a more comprehensive and precise explanation of Counterfactual Explanations, focusing particularly on Conceptual Edits that offer Black-Box Explainability of deep learning models.
- Lastly, we will present various combinations of Contrastive Graph Neural Network (GNN) models for scene graph similarity and highlight their performance. Additionally, we will experiment with a rank aware fine-tuning process in order to further boost desired metrics. We will compare these results with conventional graph kernel methods and draw conclusions based on their expressiveness, as well as their evaluation using both quantitative and qualitative measures.

Chapter 3

Background

This chapter introduces a brief overview of theoretical concepts that are essential to understand the building blocks that will be used in the present work. Artificial Intelligence (AI) is a broad field of computer science that aims to create machines capable of intelligent behavior. Today, as a result of the rapid development of the field through the last years, the terms Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) are usually confused.

Machine Learning (ML) is a subset of AI that focuses on developing statistical algorithms that enable computers to learn from data and improve their performance over time without explicit programming. Machine Learning and Statistics are closely related fields. In particular, machine learning techniques find generalizable predictive patterns from samples by leveraging huge amounts of data.

The evolution of Deep Learning (DL), which is a subset of machine learning, stands as a crucial turning point in the trajectory of artificial intelligence, marking a transformative and indispensable moment. Deep Learning is a specialized form of ML that involves neural networks with multiple layers, enabling the model to automatically extract intricate features and patterns from data. Neural Networks are computational models which were originally inspired by mechanisms of learning and information processing on the human brain.

Machine Learning and Deep Learning models are currently used, successfully, in various domains handling different data modalities. For example, RNNs are more suitable for sequential data, CNNs excel in processing grid-like data, such as images and video, Transformers excel in handling sequential data with long range dependencies and more recently GNNs were developed to handle graph structured data.

Deep Learning models have achieved remarkable results in multiple domains such as Computer Vision, Natural Language Processing, Speech Recognition, Recommendation Systems, Geometric Deep Learning, with applications spanning from medicine and finance to autonomous vehicles and language translation. The remarkable advancements in deep learning owe their existence to the rapid development of Graphics Processing Units (GPUs), essential for the complext computations required by deep learning models, and the availability of vast datasets.

Contents

3.1 Machine Learning 38
3.1.1 Input Data Types 38
3.1.2 Learning Categories 39
3.2 Deep Learning 41
3.2.1 Basic Concepts 42
3.2.2 Deep Learning Models 47

3.1 Machine Learning

Machine learning encompasses various data modalities, each tailored to handle different types of information. The most common data modalities fed as input to ML models are presented below.

3.1.1 Input Data Types

Structured Data

- **Definition:** Structured data is organized and formatted in a way that is easily recognizable, typically in tabular form. It includes data stored in databases, spreadsheets, or CSV files, where each record consists of rows and columns.
- **Examples:** Customer information, financial transactions and demographic data.
- **Machine Learning Techniques:** Commonly used techniques for structured data include linear regression, decision trees, support vector machines, and ensemble methods.

Unstructured Data

- **Definition:** Unstructured data lacks a predefined data model and is more challenging to analyze due to its non-tabular and unpredictable nature. This type includes text, images, audio, and video data.
- **Examples:** Text documents, images, audio recordings, video clips and MRIs.
- **Machine Learning Techniques:** For unstructured data, techniques such as natural language processing (NLP), computer vision, and audio signal processing are employed. Deep learning methods like convolutional neural networks (CNNs), recurrent neural networks (RNNs) and Transformers are often used for these data types.

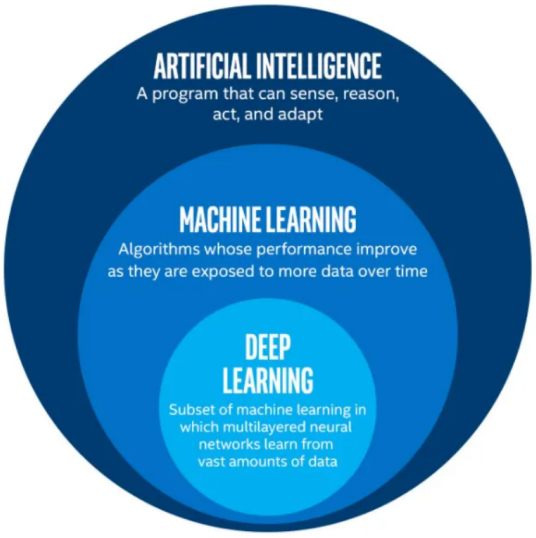


Figure 3.1.1: An illustration of the relationship of the fields of AI, ML and DL. Source: StackExchange

Sequential Data

- **Definition:** Sequential data represents information that occurs in a specific order or sequence, where the order of the elements is meaningful. Time series data is a common subtype of sequential data.
- **Examples:** Stock prices, weather patterns, speech signals and DNA/RNA sequences.
- **Machine Learning Techniques:** Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) and Transformers networks are popular for handling sequential data. These architectures are designed to capture dependencies and patterns over time.

Spatial Data

- **Definition:** Spatial data involves information related to the physical location or spatial distribution of objects. It is common in applications such as geographic information systems (GIS) and satellite imagery.
- **Examples:** Maps, satellite images and sensor data with spatial coordinates.
- **Machine Learning Techniques:** Convolutional Neural Networks (CNNs) are well-suited for spatial data as they can effectively capture spatial hierarchies and patterns.

Graph Data

- **Definition:** Graph data represents relationships between entities in a network. Nodes and edges in the graph correspond to entities and connections, respectively.
- **Examples:** Social networks, citation networks, knowledge graphs, scene graphs and molecules.
- **Machine Learning Techniques:** Graph neural networks (GNNs) are specifically designed for analyzing graph-structured data, allowing models to learn from the relationships and connections within the graph.

3.1.2 Learning Categories

Machine Learning algorithms are traditionally categorized according to the nature of the signal available to the learning system during training. Generally, there are three main categories of machine learning algorithms, namely supervised, unsupervised and reinforcement learning. More specialized techniques also include semi-supervised and self-supervised learning. In this thesis, we will mainly focus on self supervised and semi supervised algorithms for graph structured data.

Supervised Learning

In supervised learning, the algorithm is trained on a labeled dataset, where the input data, consisted of the feature vector x, is paired with the corresponding output label y. The goal is to learn a mapping $y = f(x)$ from the input to output so that the algorithm can make accurate predictions or classifications on new, unseen data. Examples include regression (predicting a continuous value), such as predicting stock prices, and classification (predicting a categorical label), such as classifying a tumor as benign or malicious. In practice, the algorithm tries to approximate the probability density function $p(y|x)$ in order to make accurate predictions. Learning Algorithms such as Linear Regression, Naive Bayes, Support Vector Machines and Decision Trees lie into this category.

Unsupervised Learning

Unsupervised learning deals with unlabeled data. Instead of responding to the supervision signal, these algorithms are designed to find patterns, structures, or relationships within the data. In this case, there are no specific output labels to guide the learning process. Instead, these algorithms identify inherent structures or groups in the data. The most important families of this type of learning include clustering and dimensionality reduction. Clustering is the process of dividing data points into clusters (groups) based on their similarities or differences. Algorithms in this subcategory include k-means and hierarchical clustering. Dimensionality reduction is the transformation of the initial data points into another space, where the number of samples remains the same, but the dimensionality of each sample is reduced. The most well known algorithm in this category is called Principal Component Analysis (PCA).

Semi-supervised Learning

Semi-supervised learning combines elements of both supervised and unsupervised learning. The model is trained on a dataset that contains both labeled and unlabeled data. This approach leverages the available labeled data along with the structure present in the unlabeled data to improve the model's performance. Examples include using a small labeled dataset along with a larger unlabeled dataset for training.

Self-supervised Learning

Self-supervised learning is a type of unsupervised learning where the algorithm generates its own labels from the input data. The model is trained to predict a part of the input from other parts of the same input, effectively creating a supervisory signal without external annotations. Examples include predicting missing parts of an image or predicting the next word in a sentence.

Reinforcement Learning

Reinforcement learning involves training a virtual agent to make sequential decisions in an environment to maximize a cumulative reward signal. The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions. Examples include training a robot to navigate a room, a game-playing AI, or optimizing resource allocation in a complex system.

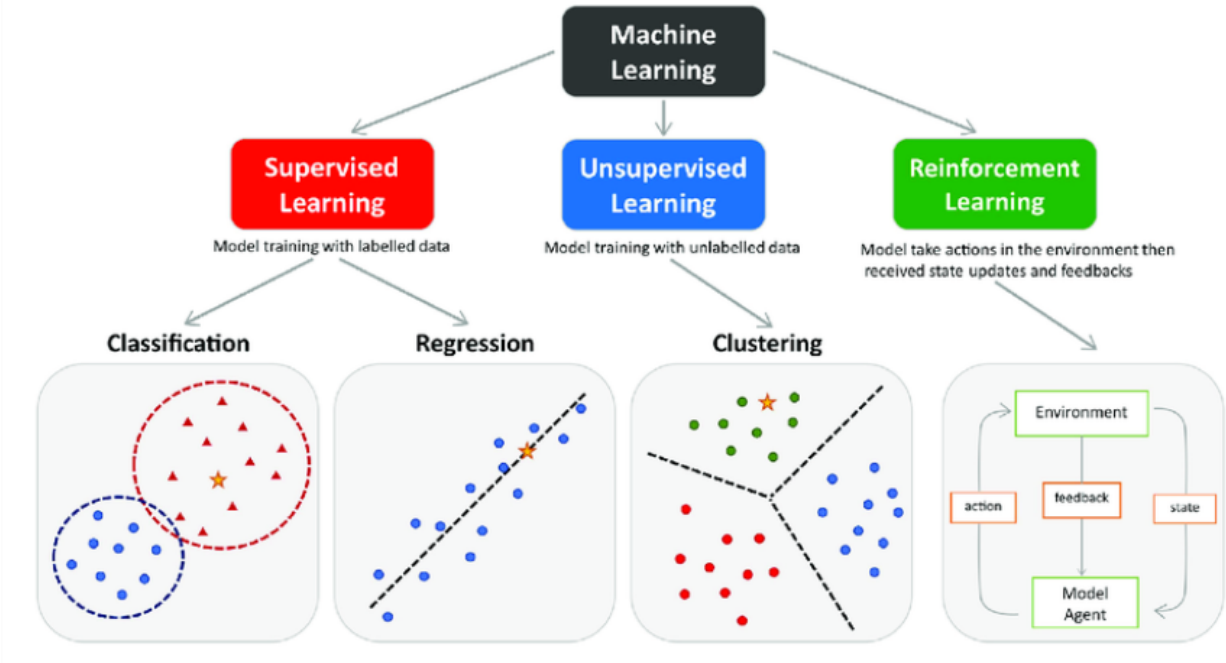


Figure 3.1.2: An illustration of the three most important categories of learning algorithms [64].

Embeddings

A key concept that we will use a lot in the context of this thesis is embeddings. Embeddings play a crucial role in machine learning, particularly in the realm of natural language processing (NLP) and other domains. In general, embeddings are representations of data in a lower-dimensional space. In the context of machine learning, they are often used to represent high-dimensional data, such as words, sentences, or even entire documents, in a more compact and meaningful form.

Specifically, in this thesis we will make use of Word Embeddings. Word embeddings are widely used in NLP. They represent words as real valued vectors in a continuous vector space, capturing semantic relationships between words. Popular word embedding techniques include Word2Vec [58] and GloVe [65] which we will later use. The goal of good word embeddings is to map words with similar meaning closer together.

Except of word embeddings there also other ways to generate embeddings for various types of data. These methods include Neural Networks as well as Dimensionality Reduction approaches such as Principal Component Analysis (PCA).

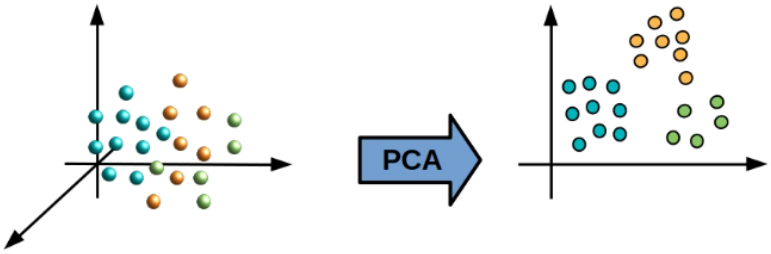


Figure 3.1.3: Visualization of Embedding points from the 3D space to 2D space using PCA

Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in statistics and machine learning. Its primary goal is to transform a high-dimensional dataset into a lower-dimensional space while retaining as much of the original variability as possible. PCA identifies the directions, called principal components, along which the data varies the most. The first principal component captures the maximum variance, and each subsequent component captures the maximum remaining variance, with the constraint that the components are orthogonal to each other. Principal components are orthogonal to each other, meaning they are uncorrelated. This is a crucial property as it ensures that the information captured by each component is distinct. PCA involves the computation of the eigenvalues and eigenvectors of the covariance matrix of the original data. Eigenvectors represent the directions of maximum variance, and eigenvalues indicate the magnitude of variance along those directions. After identifying the principal components, one can project the original data onto a subspace spanned by a selected number of these components, effectively reducing the dimensionality of the data.

However, PCA assumes that the data is linear and that the principal components capture the most significant directions of variance. This assumption may be problematic in the case where the data exhibits non-linearities.

3.2 Deep Learning

The main motivation behind the development of Deep Learning can be partially attributed to the failure of traditional Machine Learning Algorithms to generalize well to new examples. The failure of traditional ML on tasks such as speech and object recognition happened mainly due to the fact that generalizing to new examples becomes exponentially more difficult when working with high-dimensional data. Deep Learning was invented in order to overcome this and other important limitations.

3.2.1 Basic Concepts

Data serves as the foundation upon which deep learning models are built. The quality, quantity, and diversity of the data directly influence a model's ability to generalize and make accurate predictions. The process of preparing and organizing data for training is known as data preprocessing, which may involve tasks such as normalization, augmentation, and balancing to enhance model performance. In a supervised setting, a dataset consisted of n samples is usually denoted as $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

Additionally, the choice of a suitable dataset is crucial. Datasets should be representative of the problem domain and diverse enough to encompass various scenarios the model might encounter in real-world applications. Imbalances in the dataset, where certain classes or patterns are underrepresented, can lead to biased models. Addressing these imbalances is essential for achieving fair and accurate predictions.

Features are the distinctive characteristics or attributes of the input data that the model uses to make predictions. In traditional machine learning, feature engineering involves selecting and crafting relevant features manually. However, one of the strengths of deep learning lies in its ability to automatically learn hierarchical and abstract features from raw data. In the above dataset D, every element x_i is a multi-dimensional vector consisted of different attributes.

Loss Function

The goal of any Deep Learning algorithm is to learn a parameterized function $f(\cdot)$ that either accurately matches input samples to their corresponding labels in the case of supervised learning or accurately learns the distribution of data $p(x)$ in a generative setting. Therefore, it is necessary to define a way to measure the error (loss) of such a model. For instance, in the case of supervised learning, the loss function can measure the distance between the true label y and the estimated label $\hat{y}$ by the model. This could be done using a loss function $L(\hat{y}, y)$ whose output is a real number. The goal of deep learning algorithms during the training phase is to try to minimize the loss function used, as it is evident that the lower the error value, the better the prediction.

Given the aforementioned dataset D in a supervised setting, a cost function L calculated per sample and a parameterized model $f(x; \theta)$ (a classifier) the total loss is defined as the average loss across all training samples:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(f(x_i; \theta), y_i)$$

During the training process as it was already mentioned, the goal is to minimize the loss function with respect to the model's parameters. Introducing some mathematical formalism the goal is written as:

$$\hat{\theta} = argmin_{\theta} L(\theta)$$

Depending on the task, a different loss function should be selected, since classification models (binary or multi-label) and regression ones should use a different method to calculate the loss. The most popular and widely used Loss Functions,are the following:

- **Mean Squared Error Loss (MSE)** is one of the simplest and most widely used loss functions for regression tasks where the goal of the model is to predict a simple numerical value and not a class. The MSE is defined as followed:

$$L = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

- **Mean Absolute Error Loss** or L1 loss, is also used when the task to be performed is regression. However, MAE is preferred to Mean Squared Error Loss when there are many outliers in the training data. In particular, MAE is defined as the average of the absolute differences between the predicted and

actual values. Since MAE uses the absolute differences, large errors (outliers) contribute proportionally to the overall error, but they don't get squared as they do in MSE. Thus, this loss function is more robust to outliers and is defined as followed:

$$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - f(x_i)|$$

- **Huber Loss** is a robust loss also used for regression purposes. In particular, Huber Loss is a combination of Mean Squared Error and Mean Absolute Error providing a controllable threshold paremeter δ. It is defined, per sample, as followed:

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

- **Cross Entropy Loss** or **Negative Log Likelihood** is a loss function widely used in classification tasks. In particular, given a classification model whose output is a probability value between 0 and 1 for each class, Cross Entropy Loss penalizes the model more when it makes predictions that are confidently wrong and less when the predictions are close to the actual labels. The generalized Cross Entropy Loss for M different classes is defined, per sample, as followed:

$$L = -\sum_{k=1}^{M} p_k log(q_k)$$

where, p and q are the true and the predicted probability distributions for each class, respectively. A special case of Cross Entropy Loss, is the **Binary Cross Entropy Loss** where only two classes exist for classification. In this case, the Binary Cross Entropy Loss is defined, across a dataset of n samples as followed:

$$L = -\frac{1}{n}\sum_{i=1}^{n}(y_i \cdot log(f(x_i)) + (1 - y_i) \cdot log(1 - f(x_i)))$$

- **Hinge Loss** is usually used in the context of support vector machines (SVMs) and other models for classification tasks. It is designed to quantify the accuracy of a classification model by penalizing misclassifications. The primary goal of hinge loss is to encourage correct classification by penalizing the model for predictions that fall within the "margin" between the decision boundary and the true class. In a binary classification setting, where y (the true class label) takes only the values $+1$ or -1 the Hinge Loss is defined, per sample, as followed:

$$L(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

- **Kullback-Leibler (KL) Divergence Loss** is used in probabilistic models, when dealing with probability distributions. It is a measure of how dissimilar two probability distributions are and is also used in Variational Autoencoders. It is defined as followed:

$$L = \sum_i p(i) \log\left(\frac{p(i)}{q(i)}\right)$$

where p and q are probability distributions.

Gradient Descent

After the most important Loss Functions have been introduced, the next step is to determine an algorithm that could lead to the minimization of the chosen cost function. Gradient descent is an optimization algorithm commonly used in machine learning and deep learning for minimizing the cost function during the training of a model. This family of algorithms, make use of the gradients' property to point to the direction of the steepest increase of the given cost function.

In particular, starting with the initial values for the model parameters, by repeatedly computing an estimate of the cost function, the gradients of this function are calculated with respect to each parameter. Then, the parameteres are adjusted in the opposite direction of the gradient in order to reduce the cost. The mathematical equation representing this updating rule can be found below:

$$\theta' = \theta - \epsilon \nabla_\theta L(\theta)$$

where ϑ represents the model's parameters and ϵ corresponds to the learning rate. The learning rate is a hyperparameter that controls how much the parameters are updated in each iteration of the algorithm. Usually, it is set equal to a small positive constant, often by trying several values during the stage of hyperparameter tuning and keeping the value that yields the best results. Its choice can prove crucial to the model's performance.

In general, there are many variations of the gradient descent algorithm, where each offers trade-offs in terms of convergence speed, stability, and sensitivity to hyperparameters.

One of the most popular variations is called Stochastic Gradient Descent (SGD). The difference with SGD is that the parameters are updated using the gradient computed from only a single training example (x_i, y_i) at each iteration instead of using the whole training dataset to calculate the gradients. In this way, SGD is much faster than simple Gradient Descent since the latter performs many more computations of the gradients, even for similar samples of the training dataset, in order to update the parameters of the model. Instead, SGD performs only one update at a time. However, SGD may lead to oscillations around the minimum.

Another variations, include Mini-Batch Gradient Descent which strikes a balance between SGD and Gradient Descent by updating the parameters using a small random subset (mini-batch) of the training data at each iteration. In this case, tuning of the mini-batch size is important. Other important gradient descent algorithms include AdaGrad, RMSProp, and Adam. In particular, AdaGrad adapts the learning rate for each parameter based on the historical gradient information, RMSprop addresses the diminishing learning rate issue of Adagrad by using a moving average of squared gradients and Adam, which is maybe the most widely used optimizer, combines the ideas of momentum and RMSprop, incorporating both moving averages of gradients and squared gradients.

Backpropagation

Backpropagation has played a pivotal role in the success of artificial neural networks and deep learning. As we have already seen, in order to minimize the cost function, the computation of gradients with respect to the model parameters is essential. However, especially in deep neural networks with many layers and even millions and billions of parameteres, the numerical evaluation of the gradients is quite expensive. For this purpose, [69] suggested computing the gradients through the back-propagation algorithm. The algorithm works backward through the layers of the network to compute the gradient of the loss with respect to the parameters. Specifically, the partial derivatives of the loss with respect to the parameters are calculated using the chain rule of calculus. The overall process is highly efficient, making use of computational graphs and storing already computed values. In this way, it is easier to fine tune the parameters and to further minimize the model's loss and improve the overall performance.

Hyperparameter Tuning and Evaluation

Hyperparameter tuning is a crucial aspect of training deep learning models as it significantly influences the performance and generalization of the network. Hyperparameters are external configurations set before the training process begins, unlike model parameters that are learned during training. Finding the optimal combination of hyperparameters is essential for achieving the best possible model performance. Examples of hyperparameters include learning rate, batch size, number of layers, number of neurons in each layer, dropout rate, activation functions, the number of attention heads and many others. There are many Hyperparameter Search Strategies. The most important ones use techniques such as Grid Search, Random Search and Bayesian Optimization in order to explore different combinations of hyperparameters in a more systematical and effective way. In this way, the optimal set of hyperparameters that balance model complexity, training efficiency, and generalization can be identified.

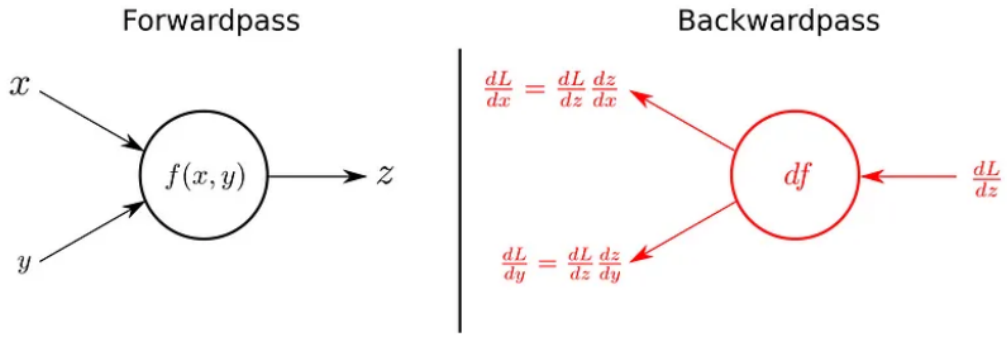


Figure 3.2.1: An illustration of how the chain rule is used during backpropagation to compute the gradients. Source: Mayank Agarwal

Evaluating and testing a deep learning model is crucial to ensure its performance and reliability in making predictions or classifications. Several metrics play a key role in assessing the model's effectiveness, and these metrics help in understanding different aspects of its performance. After the model has completed the training phase, its performance should be evaluated on a separate dataset, typically called the validation set or test set. For instance, in classification tasks the most important metrics are the following:

- Accuracy calculates the ratio of correctly predicted instances to the total number of instances and is defined as followed:
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
- Precision measures the ability of a model to correctly identify positive instances out of all instances predicted as positive and is defined as followed:
$$Precision = \frac{TP}{TP + FP}$$
- Recall evaluates the ability of a model to capture all the positive instances and is defined as followed:
$$Recall = \frac{TP}{TP + FN}$$
- F1 score is the harmonic mean of precision and recall. It provides a balanced measure that considers both false positives and false negatives. F1 score is useful when there is an uneven class distribution and is defined as followed:
$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

where TP represents True Positives, TN represents True Negatives, FP represents False Positives, and FN represents False Negatives. These metrics collectively provide a comprehensive view of the model's performance, helping you make informed decisions about its suitability for specific tasks.

Generalization and Overfitting

A deep learning model should not only perform well on data seen during the training process but it should be able to perform well on unseen or new data, beyond the examples it was trained on. Generalization is a key concept in the context of deep learning models and it is referred to the model's ability to perform well on new, unseen data drawn from the same distribution with the training dataset. This concept is very important due to the fact that the training dataset is only a sample of all the possible inputs, probably containing noisy and incomplete inputs. The ultimate goal in machine and deep learning is to build models that generalize well to new, unseen data, ensuring robust performance in real-world scenarios. Consequently, this goal demands

that both the training and the test error should be small enough. The gap between these two types of errors leads us to define two very important concepts, namely Underfitting and Overfitting.

Underfitting happens when a model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both the training and new data. High train error is usually a common sign that a model underfits the data and the main causes for this is that the model may have too few parameters or inadequate complexity.

On the other hand, Overfitting occurs when a model learns the training data too well, capturing even the existing noise and details that are specific to the training dataset but do not generalize well to new data. Common signs of overfitting include high performance on the training set and poor performance on the validation or test set. The main causes of overfitting are usually the use of overly complex models or insufficient regularization.

In order to tackle overfitting regularization techniques are used imposing constraints on the model parameters during training. Specifically, regularization forces the model to choose the smallest in order of parameters solution. This is done by implying a term $\lambda R(\theta)$ in the loss function penalizing the size of the model. The term λ is a hyperparameter and $R(\theta)$ is the regularization function of the parameters which often takes the form of the norm of the parameters.

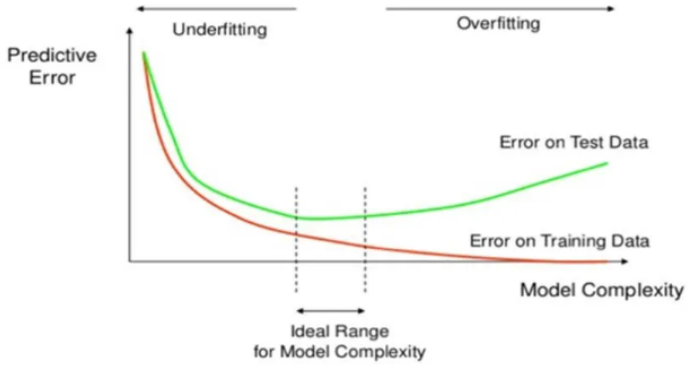


Figure 3.2.2: An illustration of the behavior of the error function according to the model's complexity including the locations where underfitting and overfitting are identified [7].

The two most common regularization techniques are the L_1 and L_2 regularization which are defined as followed:

- L_1 **Regularization (Lasso)** adds the absolute values of the parameters (weights) as a penalty term to the loss function. It encourages sparse weight matrices by pushing some weights to exactly zero and is defined as followed:

$$R_{L_1}(W) = \sum_{i=1}^{n} |w_i|$$

- L_2 **Regularization (Ridge)** adds the squared values of the parameters as a penalty term to the loss function. It discourages large weight values and helps in preventing overfitting. It is also often referred to as weight decay and is defined as followed:

$$R_{L_2}(W) = \sum_{i=1}^{n} w_i^2$$

where in the above equations w_i represent the weights (parameters) of the model.

Moreover, another way of preventing overfitting is by using dropout. Dropout during training randomly "drops out" (sets to zero) a proportion of neurons in the network, preventing reliance on specific neurons and promoting a more robust network.

3.2.2 Deep Learning Models

In order to dive into the details of deep learning models, something essential in order to after proceed in presenting Graph Neural Networks (GNNs), we should first introduce the cornerstones of modern deep neural networks.

Firstly, we will introduce the basic functioning of shallow neural networks. An essential building block of shallow neural networks is the perceptron. The perceptron is a shallow neural network consisted of only one layer of neurons where the output is determined as shown in Figure 3.2.3. In particular, each neuron in perceptron takes inputs, weighs them separately, sums them up and then passes this sum through a so called "activation function" to produce the output. Shallow neural networks refer to neural networks with only a small number of hidden layers between the input and output layers. These networks are considered "shallow" because they lack the depth of more complex architectures like deep neural networks and are more suitable for tasks where the relationships between input and output can be effectively captured with fewer layers.

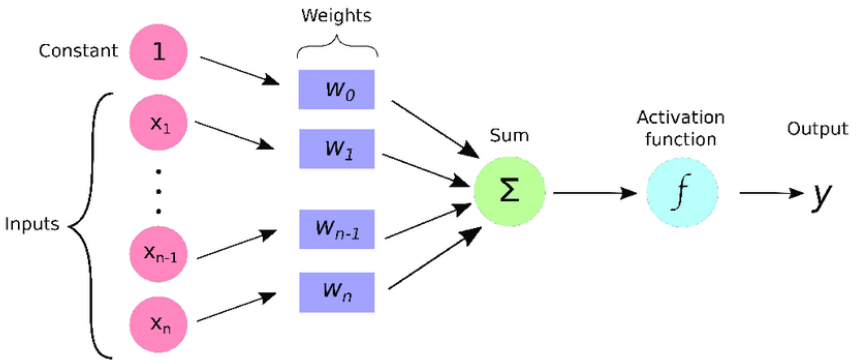


Figure 3.2.3: Single Layer Perceptron with one hidden neuron [54].

In order to overcome the limitations of shallow neural networks, Multi-Layer Perceptrons (MLPs) or differently Feed Forward Neural Networks (FFNNs) were introduced. MLPs are a type of artificial neural network that consists of multiple layers of nodes (neurons) arranged in a feedforward fashion which typically means that the information is only processed in one direction from the input nodes, through the hidden nodes to the output nodes. The layers typically include an input layer, one or more hidden layers, and an output layer. Each node in the network is connected to every node in the adjacent layers. The connections have associated weights that are adjusted during training to learn the mapping between inputs and outputs.

The output of each neuron of a MLP is calculated as it was presented in the case of perceptron in Figure 3.2.3, followed by a non-linear activation function. By stacking multiple layers together and using the non-linearity produced by the activation functions, MLPs have the ability to distinguish data that is not linearly separable.

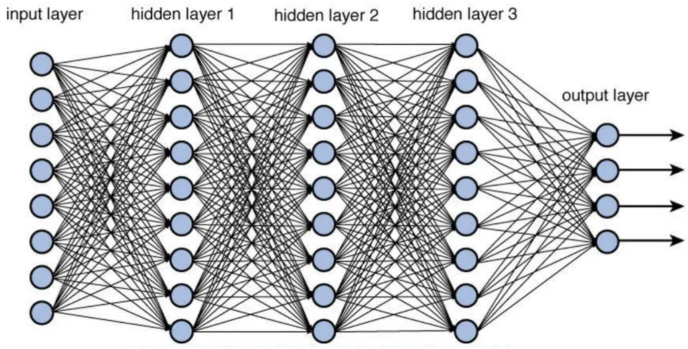


Figure 3.2.4: Multi-layer Perceptron with three hidden layers. Source: Towards Data Science

It is evident that Activation Functions play a crucial role in deep neural networks by introducing non-linearity to the model. They enable neural networks to learn complex patterns and relationships in data. Without non-linear activation functions, a neural network would essentially be a linear model, and stacking linear layers would not increase the model's capacity to learn intricate patterns. Here we will introduce the most imporant non-linear activation functions oftenly used in deep neural networks including also for completeness the Linear Activation Function.

- **Linear Activation Function** is a function where the output is proportional to the input. However, it suffers from several limitations with the most important being that its derivative is constant making backpropagation impossible. This is the reason why non-linear activation functions are usually used. It is defined as followed:
$$f(x) = ax + c$$

- **Sigmoid Activation Function** is a function that squashes the input values between 0 and 1. The derivative of the sigmoid can be expressed in terms of the function itself as $\sigma^{'}(x) = \sigma(x) \cdot (1 - \sigma(x))$. The Sigmoid function is prone to vanishing gradients which can hinder learning in deep neural networks. It is often used in the output layer for binary classification problems when the output has to be a probability and is defined as followed:
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic Tangent (tanh) Activation Function** is a function similar to the sigmoid, but it squashes the input values between -1 and 1. It is zero-centered, which helps mitigate issues with vanishing gradients. However, it still suffers from vanishing gradients like the sigmoid. Moreover, it is mostly used in hidden layers rather than in the output layer. Similar to the sigmoid, the derivative of tanh can be expressed in terms of the function itself as $tanh^{'}(x) = 1 - tanh^{2}(x)$ and is defined as followed:
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Rectified Linear Unit (ReLU) Activation Function** sets negative values to zero and leaves positive values unchanged. It is one of the most widely used activation functions in hidden layers due to its simplicity and efficiency in training. However, it suffers from the "dying ReLU" problem in which neurons using this function die during training and stop learning entirely. The issue arises when the input to a ReLU neuron is consistently negative, causing the neuron to output zero and thus the gradient to be also zero leading to no updates to the corresponding weights during training. It is defined as followed:
$$f(x) = \max(0, x)$$

- **Leaky ReLU Activation Function** introduces a small, non-zero slope for negative inputs. In this way, it mitigates the "dying ReLU" problem where neurons could become inactive during training. Leaky ReLU sets negative inputs to a small value, proportional to the input and as a result the gradient of these units will be equal to a and not zero. It is defined as followed:
$$f(x) = \max(a \cdot x, x), a > 0$$

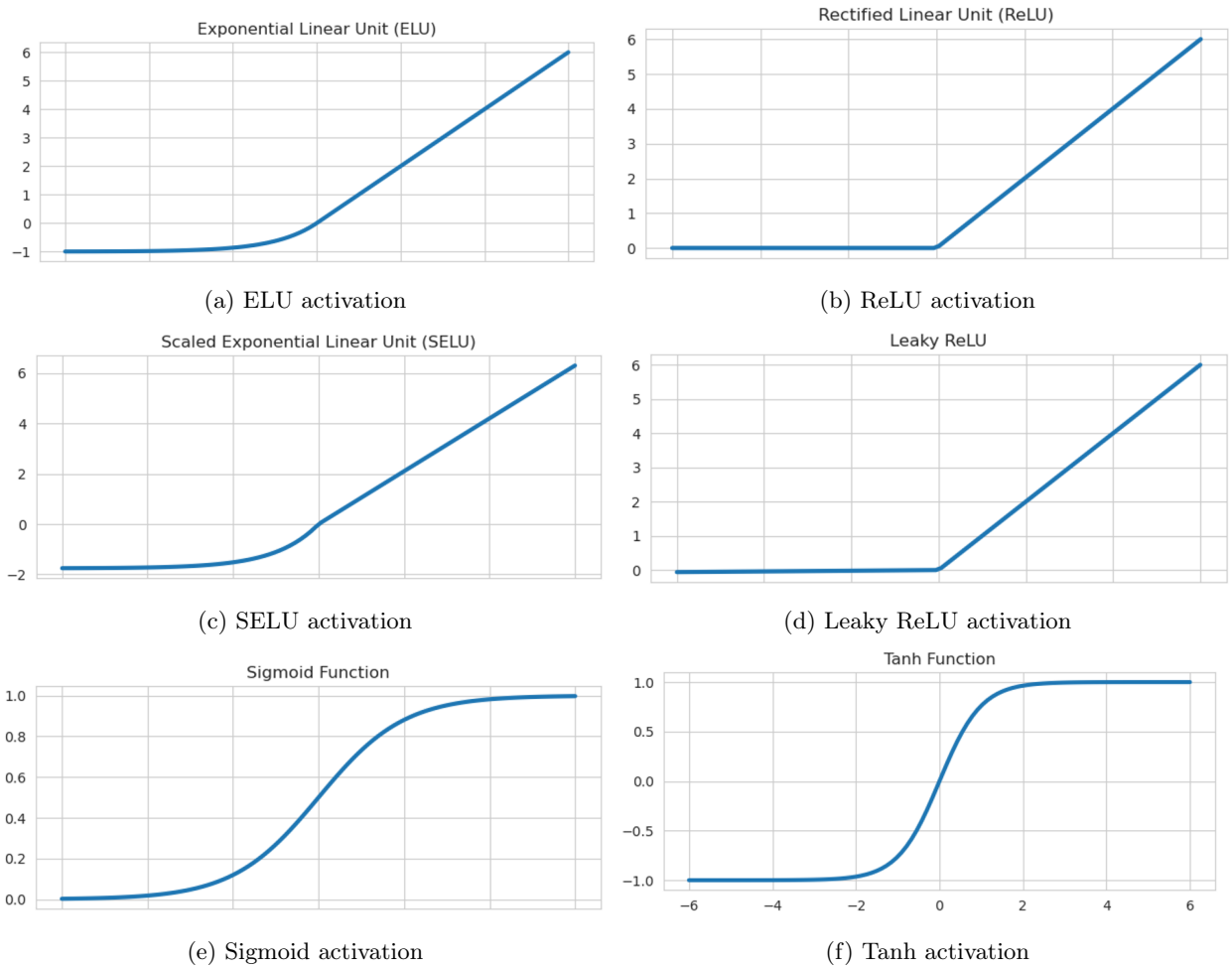
- **Exponential Linear Unit (ELU) Activation Function** is similar to ReLU for positive values, but smooth for negative ones, effectively combining the advantages of ReLU and Leaky ReLU. It addresses the "dying ReLU" problem and improves model robustness by producing smooth gradients. It is defined as followed:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot (e^x - 1) & \text{if } x \leq 0 \end{cases}$$

- **Scaled Exponential Linear Unit (SELU) Activation Function** is well known for its self-normalizing property. In a network using SELU activations, the activations tend to converge towards a mean of 0 and a standard deviation of 1 during training. This helps in maintaining stable gradients and facilitates better convergence. Moreover, SELU is designed to address the vanishing and exploding gradient problems commonly encountered in deep neural networks. The self-normalizing property allows for the gradients to neither vanish nor explode, promoting stable and efficient learning. It is defined as followed:

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha \cdot (e^x - 1) & \text{if } x \leq 0 \end{cases}$$

In summary, activation functions are critical components in deep neural networks, influencing the model's capacity to learn and generalize from data. The choice of activation function should be made based on the specific characteristics of the problem at hand and the architecture of the neural network. An illustration of the diagrams of the aforementioned activation functions can be seen in Figure 3.2.5.



(a) ELU activation

(b) ReLU activation

(c) SELU activation

(d) Leaky ReLU activation

(e) Sigmoid activation

(f) Tanh activation

Figure 3.2.5: An Illustration of the diagrams of the Activation functions mentioned before

Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a specialized type of deep neural network architecture inspired by the visual processing in the human brain introduced in [49]. The model introduced in [49] is well known as LeNet and its components can be seen in Figure 3.2.7 . CNNs are particularly well-suited for tasks involving grid-like data, such as images. Common tasks include image recognition, object detection, and image segmentation. CNNs were developed primarily for handwritten digit recognition, however the advances in hardware and the development of large datasets led to their widespread use during recent years. By extending the fundamental notion of convolution to graphs, the field of GNNs also emerged.

In general, the input data of CNNs is usually a tensor of shape (number of inputs) × (input height) × (input width) × (input channels). Each element of this tensor typically represents the value of the corresponding pixel.

The basic architecture of a CNN comprises three main types of layers:

- **Convolutional Layers:** These layers apply convolutional operations to the input data, highlighting spatial hierarchies of patterns. The convolution operation is a fundamental building block of CNNs. It involves sliding, usually small filters (kernels) over the input data, capturing local patterns and learning hierarchical representations, exploiting in the same time the locality of pixels. A convolutional layer typically consists of several filters, each with its own set of parameters. These filters are learnable during training and are initialized usually with random values. The application of convolutional kernel to an input image can be seen in Figure 3.2.6 . After passing through a convolutional layer, the image becomes abstracted to a feature map with dimensions (number of inputs) × (feature map height) × (feature map width) × (feature map channels). After the application of the convolution operator, non-linear activation functions, such as ReLU, are used to introduce non-linearity to the network, aiding in capturing complex relationships within the data.

  Moreover, some important parameters used along with the convolutional operator are called Stride and Padding.

  Stride refers to the step size that the convolutional filter takes when sliding over the input data. A larger stride reduces the spatial dimensions of the output feature map, while a smaller stride preserves more spatial information.

  Padding involves adding extra border pixels around the input data before applying convolution. These additional pixels are usually set to zero. Padding helps maintain the spatial dimensions of the input, preventing the reduction in size that would occur without padding.

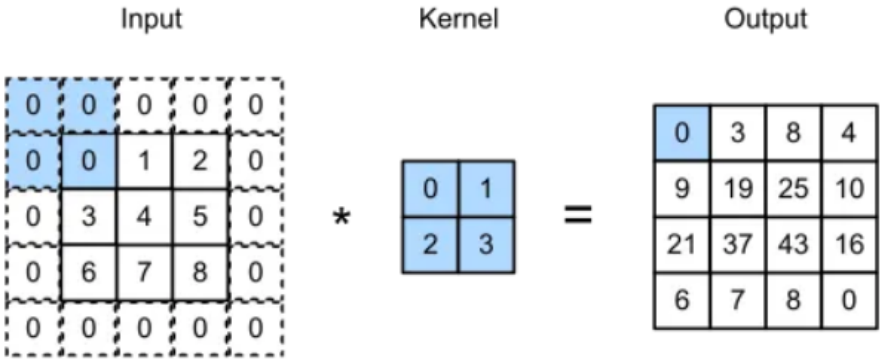


Figure 3.2.6: An illustration of the way a kernel (filter) convoles an image

- **Pooling Layers:** Pooling layers downsample the spatial dimensions of the input volume, retaining important features. There are two main types of pooling layers: max pooling and average pooling. In max pooling, the output of each region is the maximum value within that region. It helps retain the most significant features in a given region while discarding less important information. In average

pooling, the output of each region is the average (mean) value within that region. It computes the average of the values in each region, which can be less prone to outliers than max pooling.

- **Fully Connected Layers:** These layers connect every neuron in one layer to every neuron in the next, enabling high-level feature combination. Fully connected layers are typically found towards the end of the network. Typically, the flattened feature matrix goes through a fully connected layer to classify the images.

CNNs are trained using backpropagation and gradient descent. Weights of the filters are adjusted during training to minimize the difference between predicted and actual output. A crucial aspect of CNNs is weight sharing. In weight sharing, the same set of weights (parameters) of the filters is applied to different regions of the input data. As the filter slides or convolves across the input, the same weights are used at every position.

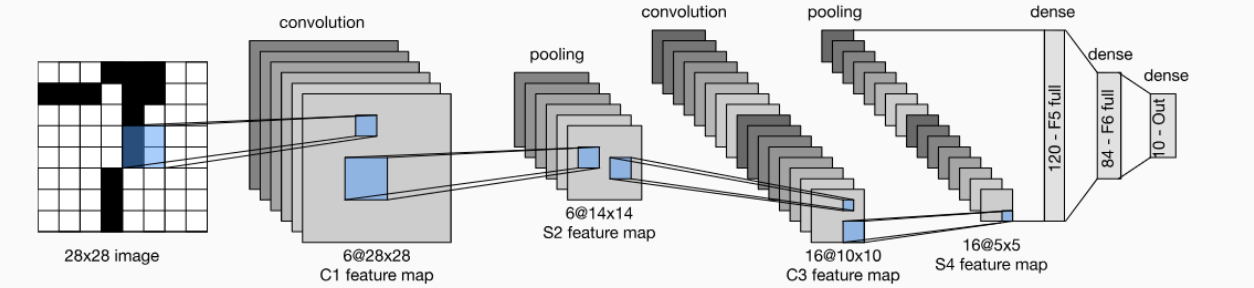


Figure 3.2.7: An illustration of the architecture of LeNet [92].

Transformers

The Transformers model, introduced by Vaswani et al. in 2017 [82], has revolutionized natural language processing (NLP) and various other fields. Its self-attention mechanism enables capturing complex dependencies in sequential data, making it a versatile choice for tasks like language translation, summarization, and image processing. Moreover as we will see later on, the attention mechanism inspired the Graph Attention Network introduced in [83].

Architecture Overview

The Transformers model architecture is built on self-attention mechanisms, enabling parallelization and capturing long-range dependencies efficiently. It consists of an encoder-decoder structure, with each layer containing self-attention and feedforward sub-layers.

- **Self-Attention Mechanism:** The self-attention mechanism allows the model to weigh different parts of the input sequence differently. Given an input sequence X, the self-attention mechanism computes a set of attention scores, which are used to create a weighted sum of the input embeddings. This process is governed by the following equation also known as scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Here, Q, K, and V denote query, key, and value matrices, respectively, and d_k is the dimension of the key vectors. In order to obtain the query, key and value matrices usually a set of trainable weight matrices are used as followed:

$$Q = X \cdot W_Q, \quad K = X \cdot W_K, \quad V = X \cdot W_V$$

The softmax function is applied to obtain the attention weights and ensure that they sum up to 1.

- **Multi-Head Attention:** Multi-head attention involves computing the attention mechanism for a sequence, not only one time, but multiple times, using multiple attention heads. Each attention head is processed separately, and then the results of all attention heads are concatenated. This allows the model to attend to different parts of the input simultaneously, enhancing its capacity to capture diverse patterns.
- **Positional Encoding:** Since Transformers lack inherent sequential information, positional encoding is added to the input embeddings to convey the position of tokens. Various positional encoding methods exist, such as sine and cosine functions, providing the model with crucial information about token order.
- **Feedforward Neural Networks:** Each transformer layer contains a feedforward neural network, typically composed of two linear transformations with a ReLU activation function in between. This allows the model to capture complex non-linear relationships in the data.
- **Layer Normalization and Residual Connections:** Layer normalization and residual connections are employed to stabilize training. Layer normalization normalizes the inputs to each layer, preventing internal covariate shift, while residual connections facilitate the flow of gradients through the network.

Training Transformers involves optimizing model parameters using gradient-based optimization algorithms. The Transformers model has been successfully applied to various NLP tasks, achieving state-of-the-art results. Its flexibility has led to adoption in diverse domains, including computer vision with the model of Vision Transformer. Despite its success, Transformers face challenges such as scalability for very long sequences and interpretability of attention mechanisms.

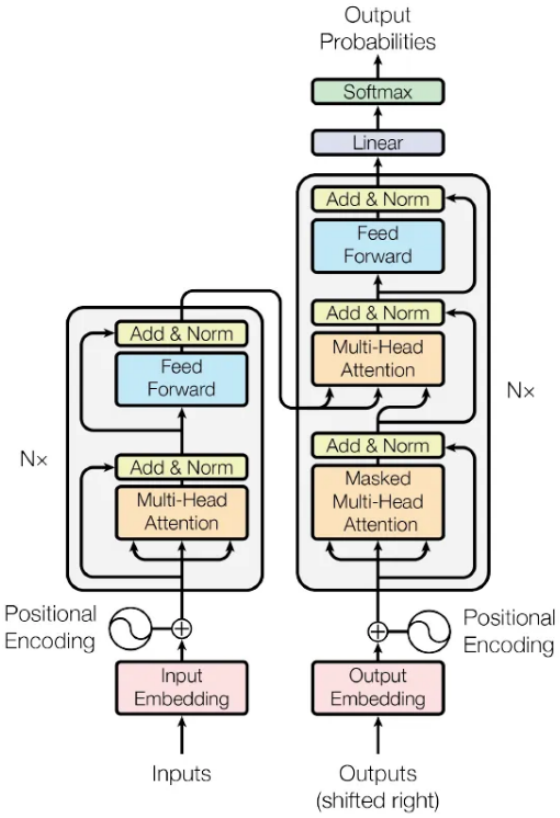


Figure 3.2.8: An illustration of the architecture of the Trasnformer model as presented in [82].

Chapter 4

Contrastive Learning

Contrastive learning aims at learning representations of data by contrasting between similar and dissimilar samples. In this way, similar entities stay close to each other while dissimilar ones are pushed farther apart in the embedding space. Similarity of samples can be defined using different metrics and the most commonly employed ones are cosine similarity which gauges the angle between two vectors in R^d and the Euclidean distance that calculates the straight line distance between points in R^d. Contrastive learning can be applied to both supervised and unsupervised settings. When working with unsupervised data, contrastive learning is one of the most powerful approaches in self-supervised learning and several contrastive learning strategies have achieved comparable or superior performance to fully-supervised state-of-the-art models on various tasks and different data modalities such as images, text and graph data among others. The ability of these approaches in capturing and learning meaningful representations requiring less labeled data compared to supervised learning, makes them versatile and effective across a wide range of machine learning tasks. In this chapter we will review the main ingredients of these methods.

Contents

4.1 Elements of Information Theory 54
4.1.1 Entropy 54
4.1.2 Kullback-Leibler (KL) Divergence 54
4.1.3 Jensen-Shannon Divergence (JSD) 54
4.1.4 Mutual Information 54
4.2 Training Objectives in Contrastive Learning 55
4.2.1 Contrastive Loss 55
4.2.2 Triplet Margin Loss 55
4.2.3 Mutual Information Maximization Losses 56
InfoNCE Loss 56
Normalized Temperature-Scaled Cross Entropy Loss (NT-Xent) 57
Deep InfoMax 58

4.1 Elements of Information Theory

Before we continue to the presentation of the most important approaches in contrastive learning it is essential that we first introduce some basic concepts of Information Theory which are strongly related with the optimization goals of different variants of contrastive losses. Information theory is a branch of applied mathematics and electrical engineering involving the quantification of information. Developed by Claude Shannon in the 1940s, it was first developed to provide a framework for understanding the fundamental limits and capabilities of communication systems. The principles of information theory have broad implications, spanning fields such as telecommunications, data science, and artificial intelligence.

4.1.1 Entropy

Entropy is a key concept in information theory introduced in [72], representing the average amount of uncertainty associated with a random variable. It is often measured in bits and higher entropy implies greater unpredictability. Entropy for a random variable X is defined as:

$$H(X) = \mathbb{E}[-\log p(X)]$$

4.1.2 Kullback-Leibler (KL) Divergence

KL Divergence denoted as $D_{KL}(P \parallel Q)$ [48] is a type of statistical distance which measures how one probability distribution P diverges from a second probability distribution Q. It is often used to quantify the difference between two probability distributions, providing a measure of information lost when one is used to approximate the other. The $D_{KL}(P \parallel Q)$ is always non-negative and is defined as:

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x \sim p(x)} \left[\log \frac{p(X)}{q(X)} \right]$$

4.1.3 Jensen-Shannon Divergence (JSD)

Jensen-Shannon Divergence [51] is a symmetric and smoothed version of KL Divergence. It measures the similarity between two probability distributions. JSD is commonly used to compare the dissimilarity between two probability distributions, and it is particularly useful when dealing with sparse data. It is defined as:

$$\text{JSD}(P \parallel Q) = \frac{1}{2} D_{\text{KL}}(P \parallel M) + \frac{1}{2} D_{\text{KL}}(Q \parallel M)$$

where $M = \frac{1}{2}(P + Q)$ is a mixture distribution of P and Q.

4.1.4 Mutual Information

Mutual information (MI) measures the amount of information obtained about one random variable by observing the other random variable. It quantifies the degree of dependence between the variables and is expressed as the reduction in uncertainty about one variable due to knowledge of the other. Mutual information determines how different the joint distribution of two random variables X,Y is from the product of the marginal distributions of X and Y. The difference of the joint distribution from the marginals is calculated using Kullback-Leibler Divergence and thus mutual information is defined as:

$$I(X;Y) = D_{KL}(P(x,y) \parallel P(x)P(y)) = \mathbb{E}_{(x,y) \sim P(x,y)} \left[\log \frac{P(x,y)}{P(x)P(y)} \right]$$

4.2 Training Objectives in Contrastive Learning

During the years, various loss functions have been developed to facilitate effective contrastive learning, each catering to specific nuances of the learning task. These losses play a crucial role in shaping the embedding space, where similar instances are drawn closer, while dissimilar ones are pushed apart. In this section we present the most foundational as well as the most commonly used ones which we used in this thesis.

4.2.1 Contrastive Loss

One of the earliest training objectives used to predict relative distances between inputs (Metric Learning) in a contrastive manner is the Contrastive Loss [21]. Given a set of input samples $\{x_i\}$ along with their corresponding label $y_i \in \{1, ..., K\}$ among K classes our goal is to learn a parameterized function e.g a neural network, $f_\theta : X \to R^d$ that maps $\{x_i\}$ into an embedding vector such that samples from the same class have similar embeddings and samples from different classes have sufficient different ones. Thus, contrastive loss takes a pair of inputs (x_i, x_j) and minimizes their distance in the embedding space when they are from the same class but pushes their distance over than some margin value m otherwise. The lower bound distance between samples of different classes is controlled by the hyperparameter m in which we usually refer to as the margin.

$$L(x_i, x_j, \theta) = \mathbb{1}(y_i = y_j)d(f_\theta(x_i), f_\theta(x_j)) + \mathbb{1}(y_i \neq y_j)max(0, m - d(f_\theta(x_i), f_\theta(x_j)))$$

where d can be any distance metric but usually the Euclidean distance is employed.

4.2.2 Triplet Margin Loss

Triplet Margin Loss was first introduced in FaceNet [71] in 2015 and since then it has been one of the most popular loss functions for supervised metric learning. Here, instead of pairs, triplets are used. Given an anchor sample x, these triplets are formed by sampling a positive sample x^+ that belongs to the same class with the anchor and then sampling a negative sample x^- that belongs to a different class.

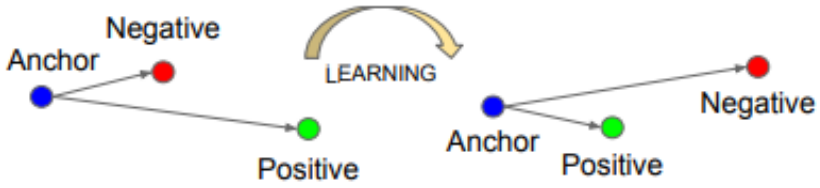


Figure 4.2.1: An illustration of how triplet margin loss works [71]

Triplet marging loss forces dissimilar pairs to be distant from any similar pairs by at least a certain margin value m.

$$L(x, x^+, x^-, \theta) = \sum_{x \in X} \max(0, d(f_\theta(x), f_\theta(x^+)) - d(f_\theta(x), f_\theta(x^-)) + m)$$

A nice property of the Triplet Margin Loss is that it can tolerate some intra-class variance in such a way as to include outliers while still ensuring a margin between samples from different clusters, e.g., negative pairs, unlike Contrastive Loss which forces the distance between an anchor and any positive sample essentially to zero even if there is no interference from negative samples.

4.2.3 Mutual Information Maximization Losses

This family of approaches seeks to maximize the Mutual Information between samples drawn from the joint distribution $p(x, y)$ (positive-related pairs) and those drawn from the product of marginals $p(x)p(y)$ (negative-unrelated pairs). This technique forces the model to encode more similar samples closer in the embedding space and push farther apart dissimilar ones.. MI is a fundamental quantity for measuring the amount of information obtained from a random variable X by observing some other random variable Y. As Mutual Information is notoriously difficult to compute, especially in continuous and high dimensional spaces, lower bounds of Mutual Information called Mutual Information Estimators, such as InfoNCE and Jensen Shannon, are used. Particularly, we seek to maximize these lower bounds using deep neural networks, supposing that these bounds are tight enough. Bounding Mutual Information in high dimensions remains a challenging topic.

The following approaches that will be presented, can be unified under a shared mathematical formalism. In particular, these approaches follow MINE [8] that uses a lower-bound to the Mutual Information (MI) based on the Donsker-Varadhan (DV) representation [27] of the Kullback-Leibler divergence given by:

$$I(X;Y) = D_{\mathrm{KL}}(J \parallel M) \geq \hat{I}_{\omega}^{(DV)}(X;Y) = \mathbb{E}_{p(x,y)}[T_{\omega}(x,y)] - \log \mathbb{E}_{p(x')p(y')}\left[e^{T_{\omega}(x',y')}\right]$$

where $T_{\omega} : X \times Y \to \mathbb{R}$ is a discriminator function modeled by a neural network with parameters ω, J is the joint distribution, M is the product of marginals and $\hat{I}$ is a mutual information estimator. As we will see later on, $T_{\omega}(x, y)$ usually takes as inputs the embeddings of x and y produced by an encoder and then calculates a similarity measure (usually cosine similariy) between these embeddings. In practice, given an encoder $E_{\psi} : X \to Y$ with parameters ψ (e.g., a neural network), where X and Y are the domain and range of a continuous and (almost everywhere) differentiable parametric function, these methods optimize E_{ψ} by simultaneously estimating and maximizing $I(X, E_{\psi}(X))$ using a selected MI estimator $\hat{I}$.

The InfoNCE mutual information estimator can be written as:

$$I^{InfoNCE}(X;Y) = \mathbb{E}_{p(x,y)}\left[T(x,y) - \log \sum_{x' \sim p(x)} e^{T(x',y)}\right] = \mathbb{E}_{p(x,y)}\left[\log \frac{e^{T(x,y)}}{\sum_{x'} e^{T(x',y)}}\right] = \mathbb{E}_{p(x,y)}\left[\log \frac{f(x,y)}{\sum_{x'} f(x',y)}\right]$$

where, due to distinct motivations, InfoNCE adopts a different resampling process than the DV approach. Specifically, DV resamples all x' and y' from their marginal distributions in the negative log term while InfoNCE only resamples x'.

As we are primarily interested in maximizing mutual information, not its precise value, [39] suggest to replace the KL divergence with the Jensen-Shannon Divergence(JSD), resulting in the JSD mutual information estimator:

$$I^{JSD}(X;Y) = D_{\mathrm{JS}}(J \parallel M) \geq 2\log 2 + \mathbb{E}_{p(x,y)}\left[-sp(-T(x,y))\right] - \mathbb{E}_{p(x')p(y)}\left[sp(T(x',y))\right]$$

where $sp(x) = log(1 + e^x)$ is the softplus function.

InfoNCE Loss

The InfoNCE loss was first introduced in Contrastive Predictive Coding [62] where given a context vector c (e.g an anchor sample) it learns to optimize the negative log probability of identifying the positive sample drawn from the conditional distribution $p(x|c)$ amongst a set of negative unrelated samples drawn from the marginal distribution $p(x)$. It belongs to a family of approaches which seek to estimate and maximize the mutual information between 2 variables using neural networks [8], [39]. A simple neural network $f_{\theta}(x, c)$ is used to estimate the density $\frac{p(x|c)}{p(x)}$ because the probability of identifying the positive sample among a set $X = \{x_i\}_{i=1}^{N}$ of N samples is proved to be equal to:

$$p = \frac{\frac{p(x_{pos}|c)}{p(x_{pos})}}{\sum_{j=1}^{N} \frac{p(x_j|c)}{p(x_j)}} = \frac{f_{\theta}(x_{pos}, c)}{\sum_{j=1}^{N} f_{\theta}(x_j, c)}$$

Specifically, this method is based on the fact that the anchor and the positive sample should be encoded in a way that maximally preserve mutual information of the original c (anchor) and x (positive sample) signals. The mutual information between positive sample x and context vector c is equal to:

$$I(x;c) = \sum_{x,c} p(x,c) log \frac{p(x|c)}{p(x)}$$

where the term $\frac{p(x|c)}{p(x)}$, is estimated by the neural network. Thus, by maximizing $f_\theta(x_{pos}, c)$ for a given anchor sample c, the mutual information between the positive sample and the anchor is proved to be also maximized. The InfoNCE loss that optimizes the negative log probability of identifying the positive sample correctly given an anchor sample is given by the following equation:

$$L_{InfoNCE} = -\mathbb{E}\left[\log \frac{f_\theta(x_{pos}, c)}{\sum_{x \in X} f_\theta(x, c)}\right]$$

It is proved that minimizing $L_{InfoNCE}$, maximizes a lower bound of the true MI between positive pairs [66], [62]. The Contrastive Predictive Coding specific setup is related to sequence prediction tasks. It employs in first place an encoder that maps the input x to a latent space z and then, by using an autoregressive model the context vector c is calculated up to a timestep t. Then, a simple log bilinear model $f_\kappa(x_{t+k}, c_t) = exp(z_{t+k}^T W_k c_t)$ is used to estimate the density $\frac{p(x_{t+k}|c_t)}{p(x_{t+k})}$ where z_{t+k} is the encoded input and W_k is a trainable matrix. By optimizing the InfoNCE loss, the encoder, the autoregressive model and the log bilinear model are all together trained in order to produce embeddings c_t and z_{t+k} that maximally preserve the mutual information between x_{t+k} and c_t.

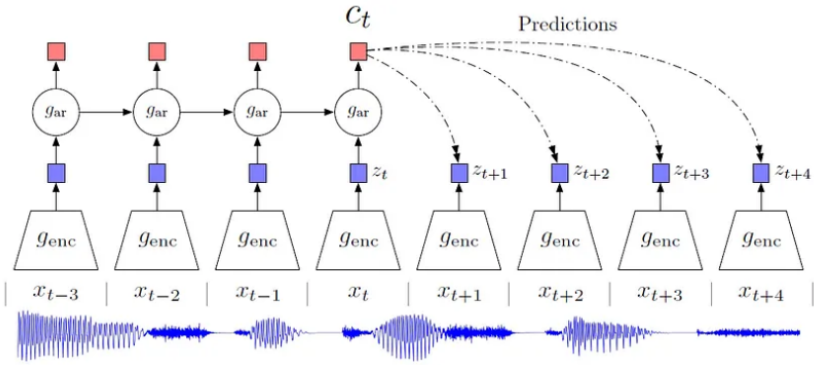


Figure 4.2.2: An illustration of the CPC architecture [62]

Normalized Temperature-Scaled Cross Entropy Loss (NT-Xent)

The NT-Xent and the InfoNCE losses are essentially the same. The name NT-Xent Loss was introduced in the SimCLR paper [20] and it is possibly the most widely used contrastive loss.

The SimCLR framework consists of four major components. Firstly, a Data Augmentation Module is used, which transforms a given sample randomly in two ways, yielding two correlated views of the same example. Secondly, a Base Encoder followed by an MLP is employed to extract representation vectors from augmented data examples. Finaly, a Contrastive Loss Function is at the core of the model. Specifically, in this approach positive samples are obtained by employing augmentation techniques to each sample in a batch of size N, resulting to $2N$ data points. Furthermore, given a specific sample from the batch and its augmented view,

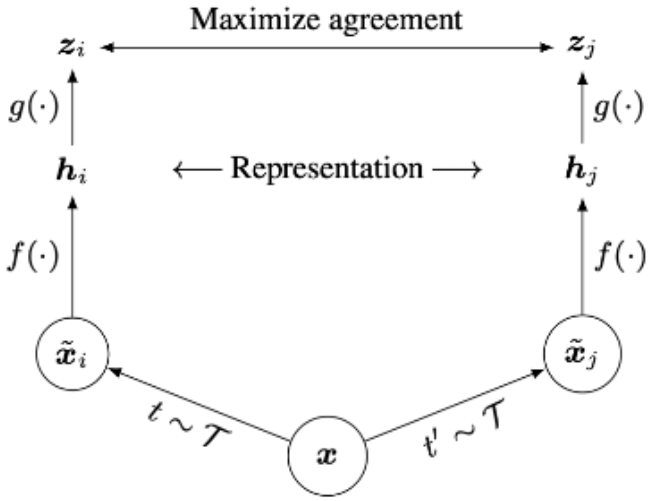


Figure 4.2.3: An illustration of how SimCLR pulls closely the embeddings z_i and z_j of a positive pair[20]

which is considered as a positive pair, all the other $2N-2$ samples are treated as negative samples for the contrastive loss. Thus, the loss function for a specific positive pair of samples (i, j) is defined as:

$$l_{i,j} = -\log \frac{exp(sim(f_\theta(x_i), f_\theta(x_j))/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} exp(sim(f_\theta(x_i), f_\theta(x_k))/\tau)}$$

where f is a neural network, τ is a hyperparameter called temperature and sim is a similarity metric, usually the cosine similarity. It is evident that this expression resembles a softmax classifier that classifies positive and negative samples correctly. This should encourage the score function to assign large values to positive examples and small values to negative examples, where the score function is $exp(sim(f_\theta(x_i), f_\theta(x_j))/\tau)$. This is exactly the same formula used in the InfoNCE loss. The total loss for each batch is then computed by taking the average of the terms $l_{i,j}$ for all positive pairs in the batch expressed by the following equation:

$$L = \frac{1}{2N} \sum_{k=1}^{N} [l(2k-1, 2k) + l(2k, 2k-1)]$$

Minimizing this loss with respect to the neural network parameters, forces the model to encode more similar examples closer in the embedding space and push farther apart dissimilar ones. The augmentations used to obtain the positive samples can vary a lot and they depend heavily on the modality of the data. For images, these augmentations could include transformations like rotations, flips, cropping, changes in brightness, and more while in graphs these transformations include node dropping, edge dropping, and feature masking among others.

Deep InfoMax

Deep InfoMax [39] originally developed for images, defines as the contrastive task the problem of whether a pair of global features and local features are from the same image. Specifically, Local Deep InfoMax maximizes mutual information between a global flat summary feature vector of an image, and a collection of local feature vectors (a $M \times M$ feature map) of the same image, pulled from an intermediate layer of the convolutional encoder. Deep InfoMax can work with various Mutual Information Estimators such as Jensen Shannon and InfoNCE. According to [39] the InfoNCE MI estimator outperforms JSD on downstream tasks. Given $f(x)$ as the global summary representation, $f(x^+)$ as the local feature map from the same image (positive samples), and $f(x^-)$ as the local feature maps from different images (negative samples), the loss function using the InfoNCE estimator, is exactly the aforementioned InfoNCE loss. In a batch training

setting, for every positive example, negative examples are considered all combinations of local patches from all images across the batch with the relevant global summary vector.

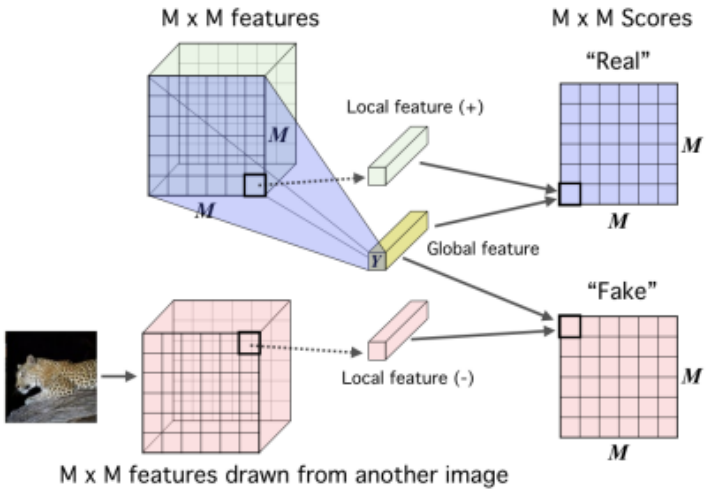


Figure 4.2.4: Maximizing mutual information between local features and global features.[39]

Chapter 5

Graphs

Contents

5.1 Graph Theory **62**
5.2 Graph Similarity **65**
5.2.1 Graph Edit Distance 65
5.2.2 Graph Kernels 66
5.3 Scene Graphs **72**
5.4 Related Work **74**

5.1 Graph Theory

One of the principal study objects of discrete mathematics are graphs. Graphs are mathematical structures used to model pairwise relations between objects. A graph, is denoted as $G = (V, E)$, where V is a set of objects called vertices (also called nodes or points) and E is a set of distinct unordered pairs of distinct vertices called edges (also called links). In practice, this means that every edge "connects" two distinct vertices. In the edge {x,y}, the vertices x and y are called the endpoints of the edge. The edge is said to join x and y and to be incident on x and on y. A vertex may exist in a graph and not belong to an edge. In this case, we refer to this vertex as an isolated node.

In any simple graph there is at most one edge joining a given pair of vertices. However, many results that hold for simple graphs can be extended to more general objects in which two vertices may have several edges joining them. In addition, we may remove the restriction that an edge joins two distinct vertices, and allow self-loops which are edges joining a vertex to itself. The resulting object, in which loops and multiple edges are allowed, is called a general graph or, simply, a graph which serves as a generalization. Thus every simple graph is a graph, but not every graph is a simple graph. In literature, the term "graph" is usually used as a synonym for a simple graph, i.e. a graph without any self-loops and no more than one edge connecting any pair of vertices.

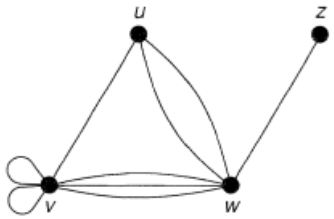


Figure 5.1.1: A Visual Representation of a General Graph

A graph can be classified into different types based on its properties. Graphs can be classified based on the directionality of the edges between their nodes. To elaborate, an edge connecting nodes x and y is termed as undirected if both ordered pairs (x, y) and (y, x) are part of the set of edges, indicating a bidirectional connection between x and y. On the other hand, the edge is considered directed if only one of these pairs is present, indicating a directional connection. Consequently, a graph is categorized as undirected when all its edges are undirected, and it is termed directed (or digraph) if at least one edge is directed. An inherent and very important property of undirected graphs is the symmetry exhibited by their adjacency matrix.

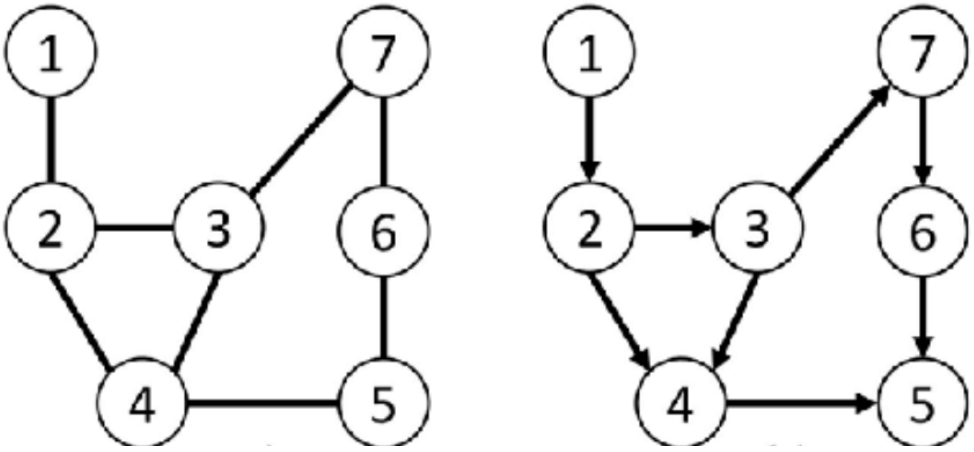


Figure 5.1.2: A Visual Representation of an Undirected and a Directed Graph

In order to proceed in the definition of the adjacency matrix of a graph we will first introduce some important definitions:

Order: The order of a graph is defined by the number of its vertices, denoted as |V|

Size: The size of a graph is defined by the number of its edges, denoted as |E|

Adjacency: Two vertices u and v are *adjacent* in G if there exists an edge $\{u, v\}$ connecting them.

Neighborhood: The neighborhood $N(u)$ of a vertex u in a graph is defined as the set of nodes adjacent to it

Degree: The *degree* of a vertex v in an undirected graph is the number of edges incident to v. In a directed graph, it is divided into *in-degree* (number of edges coming into the vertex) and *out-degree* (number of edges going out from the vertex). In a simple graph of order n, the maximum degree of any vertex is upper bounded by the quantity $n - 1$. Moreover, the number of edges are upper bounded by the quantity $\frac{n(n-1)}{2}$.

Graphs can be represented in various ways to facilitate analysis and algorithmic development. The most common representations are the Adjacency Matrix and the Adjacency List.

Adjacency Matrix: A graph of order N may be fully characterized by its adjacency matrix A which is a square matrix of dimensions $N \times N$. In this matrix, non-zero elements indicate the existence of a link between vertices. In the case of a simple graph, A_{ij} takes only two values, 0 and 1, where the value 0 denotes disconnection and 1 indicates connection between vertices i and j. Notably, undirected graphs demonstrate a symmetric adjacency matrix, indicating that A_{ij} is equal to A_{ji}.

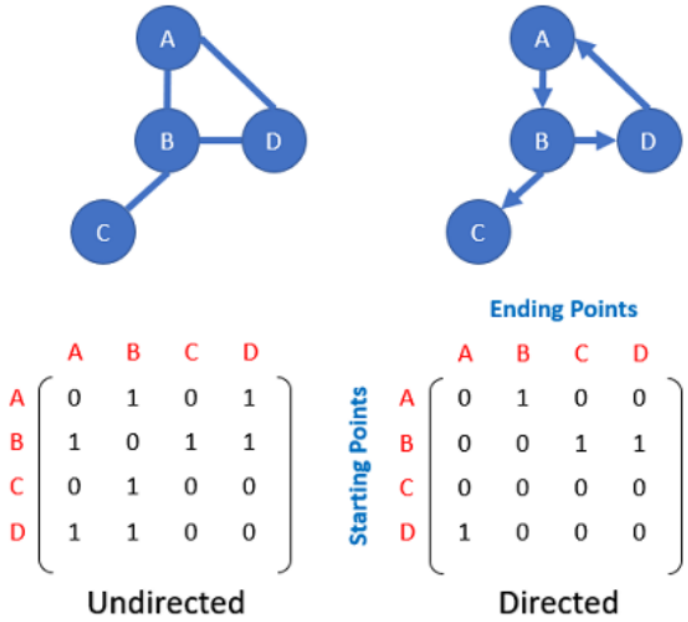


Figure 5.1.3: A Visual Representation of two graphs along with their adjacency matrices

Besides the adjacency matrix, a graph can also possess attributes associated with its nodes and/or edges. In such instances, each node (or edge) is identified by a feature vector with a dimension of D, leading to a node (or edge) feature matrix denoted as X with dimensions $N \times D$.

Furthermore, in some cases A_{ij} take on positive integer values bigger than 1. This lead us to another important distinction which is the classification of a graph as a weighted or unweighted graph. The distinction arises from whether the edges connecting nodes have assigned values. In a weighted graph, each edge is linked to a numerical weight or cost, typically representing factors like distance, time, or cost, quantifying the connection between nodes. These weights introduce more information and intricacy, allowing for a more

accurate representation of real-world scenarios. In this case, the weight value for the corresponding edge would be present for each node pair in the adjacency matrix. Conversely, in an unweighted graph, all edges are treated equally, lacking assigned numerical values. This makes it suitable for representing relationships where only connectivity matters.

Adjacency List: In this representation, each vertex of the graph is associated with a list that contains its neighboring vertices, forming an efficient and compact way to express the graph's connectivity.

For an undirected graph, the adjacency list captures edges in a symmetrical manner. Each vertex's list includes the vertices to which it is directly connected. In the case of a directed graph, the adjacency list distinguishes between incoming and outgoing edges for each vertex.

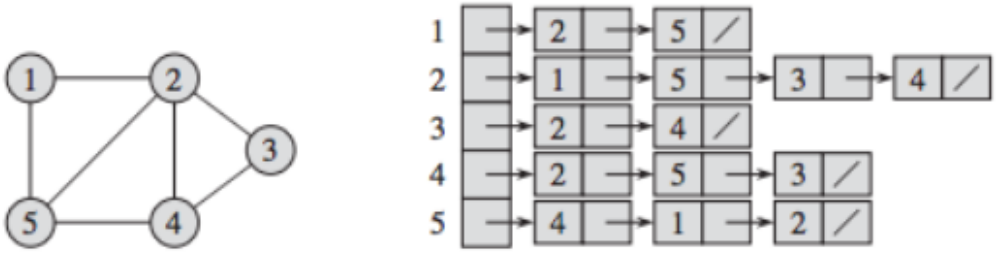


Figure 5.1.4: An Undirected Graph along with its adjacency list

The advantages of using an adjacency list include its space efficiency, especially for sparse graphs where the number of edges is significantly less than the maximum possible. Traversing the neighbors of a vertex is straightforward, and adding or removing edges can be done efficiently. However, determining the presence of an edge between two vertices may require scanning through the adjacency list, potentially resulting in a higher time complexity.

Moreover, some fundamental concepts of graph theory include concepts such as paths, walks, cycles, connected graphs, and subgraphs. All of them can be used to derive valuable graph metrics, which will prove particularly beneficial at defining several important Graph Kernels later on.

Path: A path in a graph G is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that each adjacent pair of vertices v_i and v_{i+1} is connected by an edge. The length of a path is the number of edges it contains. A path with only one vertex and no edges is considered to have a length of 0. A simple path is a path in which no vertex is repeated (revisited).

Walk: A walk is a generalization of a path. Unlike a path, a walk can revisit vertices and edges. It is a more general concept and includes repeated vertices and edges.

Cycle: A cycle is a path in which the first and last vertices are the same.

Subgraph: A graph $H = (V', E')$ is called a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Connected Graph: A connected graph is a graph in which there is a path (a sequence of edges) between every pair of vertices. In other words, for any two vertices in a connected graph, there exists at least one path that connects them. If a graph is not connected, it may consist of multiple isolated components, each of which is a connected subgraph. In the context of directed graphs, connectedness can be categorized into two distinct forms: weak and strong. Weak connectivity in a directed graph implies that for any pair of vertices u and v, there is a path either from u to v or from v to u. Conversely, strong connectivity in a directed graph requires the existence of paths in both directions for any pair of vertices.

5.2 Graph Similarity

The problem of Graph Similarity revolves around determining the degree of similarity between two graphs, essentially establishing a mapping $m : G \times G \to \mathbb{R}$ that characterizes their likeness or dissimilarity. The Graph Similarity problem, essential in applications like biological network analysis and social network comparison, draws on a rich history rooted in graph theory and computational mathematics. Over time, various algorithms, including graph kernels and graph edit distance algorithms, have been developed to measure the similarity of a pair of graphs.

Graph Edit Distance (GED) is a pivotal metric within this context, which gauges the similarity between two graphs by counting operations needed to transform one graph into another, whereas Graph Kernels is a polynomial alternative to GED. This thesis explores GED and Graph Kernels, comparing them with the concept of utilizing Contrastive Graph Neural Networks to map graphs to feature vectors for similarity assessment. In the following section we will introduce the basic concepts of the aforementioned methods.

5.2.1 Graph Edit Distance

As we have already mentioned, Graph Edit Distance (GED) is a measure of similarity (or dissimilarity) between two graphs G_1 and G_2. Alberto Sanfeliu and King-Sun Fu introduced the first mathematical formalization of graph edit distance in 1983 [70]. GED can be regarded as a broader form of alternative distances, like string edit distance, tree edit distance [94], or Hamming distance [37], provided that graphs are constructed with appropriate constraints.

Formally, the Graph Edit Distance between the graphs G_1 and G_2 is denoted as $GED(G_1, G_2)$. The core concept of GED, as it was introduced in [70], involves defining a set of Graph Edit Operations. The Graph Edit Operations and their costs are denoted as e_i and $c(e_i) \geq 0$ correspondingly where $P(G_1, G_2)$ denotes the set of all edit paths transforming G_1 into G_2. In this context, $GED(G_1, G_2)$ can be defined as:

$$GED(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in P(G_1,G_2)} \sum_{i=1}^{k} c(e_i)$$

The set of elementary Graph Edit Operations typically includes:

- **Vertex insertion**: Introduce a single new labeled vertex to a graph
- **Vertex deletion**: Remove a single (often disconnected) vertex from a graph
- **Vertex substitution**: Change the label (or color) of a given vertex
- **Edge insertion**: Introduce a new colored edge between a pair of vertices
- **Edge deletion**: Remove a single edge between a pair of vertices
- **Edge substitution**: Change the label (or color) of a given edge

Precise methods for calculating the Graph Edit Distance between two graphs generally involve converting the task into one focused on identifying the edit path with the minimum cost. The approaches employed for this computation typically involve either pathfinding searches or determining the shortest paths, making use of the A* search algorithm. In general, the problem of computing Graph Edit Distance is NP-hard and is even hard to approximate belonging in the APX-hard complexity class.

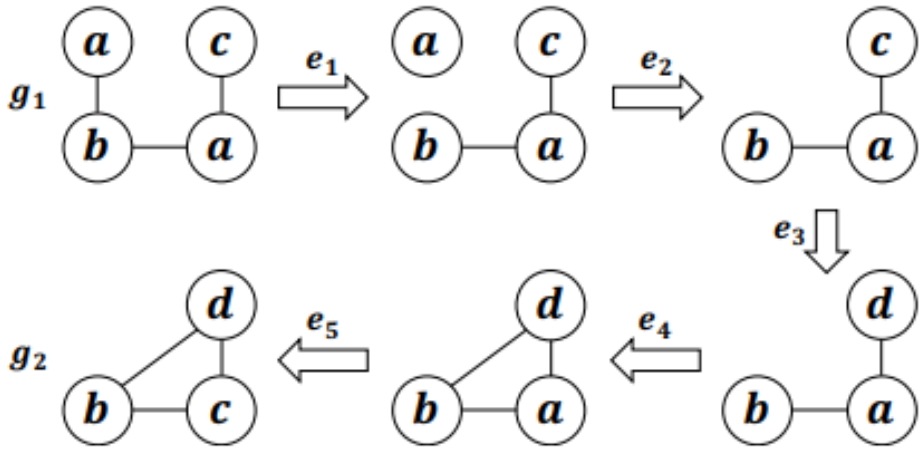


Figure 5.2.1: Graph Edit Distance Between Two Graphs. [78]

Because the exact computation of GED is NP-Hard, many graph edit distance approximation approaches have been developed.

Bipartite Heuristic, introduced by [68], enhances the A* algorithm's search for the target path by introducing a new heuristic function. While this method accelerates the search, it still ensures an optimal solution. However, it cannot deterministically solve the Graph Similarity problem in polynomial time. This heuristic is seldom employed unless the exactness of the similarity measure between two graphs is crucial.

A* Beamsearch proposed in [60], is one of the first approximate Graph Edit Distance method that speeds up the process by modifying the standard A* algorithm. It limits the number of expanded nodes in the search tree, exploring only the most promising partial matches. While this optimization reduces computational complexity, making it faster, it may not find the optimal solution, providing a suboptimal solution.

Bipartite Matching, is an approximate algorithm introduced by [28] which leads to a suboptimal solution to the Graph Edit Distance (GED) problem. In particular, it firstly computes the exact edit distance considering only node edit operations such as insertions, deletions and substitutions and then infers the edge edit operations. This results in a great speed-up, but is also the reason for the suboptimality of the solution. The node edit distance problem is treated as a Linear Sum Assignment Problem (LSAP), which can be solved in polynomial time using the Volgenant-Jonker (VJ) algorithm [41]. The complexity of the VJ algorithm is $O(n^3)$, but in practice it is much faster. In particular, for the LSAP problem, we want to assign the nodes of the two graphs represented by the two vertices sets, where each assignment has a precomputed cost c_{ij}. The VJ algorithm computes the minimum cost node assignment and the implied edit operations of the edges are then inferred. In this thesis, we will use this method to obtain the ground truth for the Graph Edit Distance problem as it is widely used due to its efficient balance between the accuracy of the approximate solution and computational cost.

Hausdorff Matching, introduced in [32], operates similarly to Bipartite Matching but employs a quadratic-time approximation algorithm based on Hausdorff Matching as a heuristic function to approximate the graph edit distance.

5.2.2 Graph Kernels

Despite the ability of Graph Edit Distance algorithms to accurately calculate graph similarity, they suffer from a lack of computational efficiency. The GED problem is known to be NP-hard, and the task of selecting suitable costs for the edit operations is complex and quite challenging. In contrast, Graph Kernels present a polynomial time approach for gauging graph similarity, offering an efficient, expressive, and widely applicable alternative to the complexities associated with Graph Edit Distance (GED) algorithms. Before delving into various graph kernels that will be used in this thesis, we will initially introduce the more general and fundamental concept of kernels in the field of machine learning.

Kernels

In machine learning, kernels play a fundamental role in various algorithms for pattern recognition and data mining, particularly in the context of support vector machines (SVMs) and kernelized methods. A kernel is a function that computes the similarity between pairs of data points in a high-dimensional space, without explicitly transforming the data into that space. This implicit transformation of data, enables linear algorithms (such as linear classifiers) to adeptly address nonlinear problems by effectively capturing the similarity or dissimilarity between data points and avoiding the computational burden required to map the data to higher dimensions. As a result, kernel methods have proved to be essential for capturing complex relationships and enabling algorithms to operate efficiently in non-linear domains.

The most important concept in Kernel Theory is called the Kernel Trick. The Kernel Trick is a fundamental concept enabling to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the projections of all pairs of data in the feature space. Breaking down this concept into details we have:

- **Linear Separation in Higher Dimensions:** In many machine learning real-world scenarios, the data may not be linearly separable in the original feature space. However, it might be separable in a higher-dimensional space. This is very important as linear classifiers such as SVMs excel when data is linearly separable, but they struggle otherwise.

- **Kernel Functions:** A Kernel Function is a function that computes the similarity or in other words the inner product between pairs of data points in a feature space without explicitly calculating the coordinates of that space. Common kernel functions for two vectors (data points) x_i and x_j include:

  1. Linear Kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$

  2. Polynomial Kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$

  3. Radial Basis Function (RBF) Kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$ where the quantity $\frac{1}{2\sigma^2}$ is a parameter often denoted as γ

- **Feature Mapping:** The kernel trick is based on the fact that the dot product of the transformed feature vectors can be expressed as a function of the dot product of the original feature vectors without explicitly calculating the transformation. In other words, if ϕ is the transformation function, the kernel trick relies on computing the quantity $K(\mathbf{x}_i, \mathbf{x}_j)$ without computing $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ separately. The function ϕ satisfies:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

  where $\langle \cdot, \cdot \rangle$ is a proper inner product. The function $\phi : X \to V$ is the function that maps the original vectors x_i and x_j into the higher dimensional space where V is a Hilbert Space.

- **SVM and Decision Boundary:** Within SVMs, the kernel trick emerges as a potent tool. By applying a kernel function, the decision boundary undergoes a transformation into a more intricate shape in a higher-dimensional space. This transformation facilitates SVMs in effectively distinguishing data that lacks linear separability in its original space.

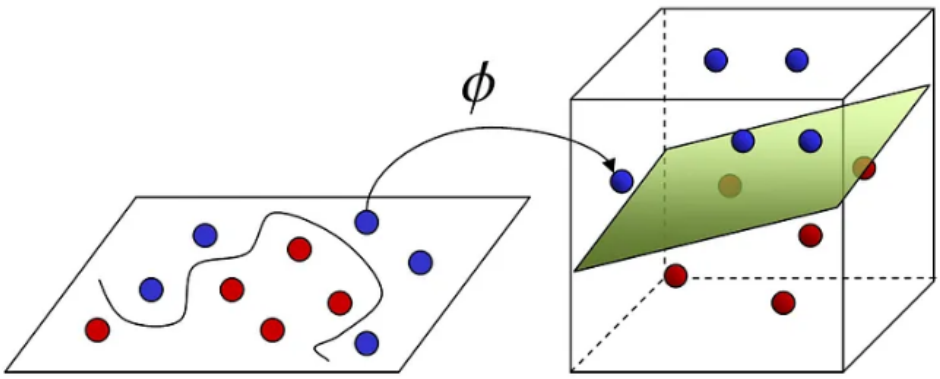


Figure 5.2.2: Illustration of Kernel Trick

Graph kernels are a powerful tool in the field of machine learning, specifically designed to analyze and compare structured data represented as graphs. These kernels quantify the similarity between graphs by measuring the similarity of their respective substructures, capturing both local and global patterns. These substructures include counting of graphlets, random walks and shortest paths as well as message propagation techniques. All of them can be described in the context of the general kernel definition as:

$$K(\mathbf{G}, \mathbf{G}^{'}) = \left\langle \phi(\mathbf{G}), \phi(\mathbf{G}^{'}) \right\rangle$$

where G and $G^{'}$ are two graphs and $K : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{R}$. They play a crucial role in tasks such as graph classification, clustering, and regression. Graph kernels enable the application of traditional machine learning techniques to graph-structured data, making it possible to leverage the rich relationships and dependencies present in complex systems.

In the context of this thesis, we will explore five graph kernel methods which will be compared to Contrastive Graph Neural Network models. The subsequent sections will provide insights into the functioning of these graph kernels.

Weisfeiler-Lehman Kernel

The Weisfeiler-Lehman (WL) kernel is a powerful graph kernel based on the Weisfeiler-Lehman Isomorphism test proposed in [87], which is a technique for distinguishing non-isomorphic graphs by iteratively refining their vertex labels. This framework was first introduced in [74] and consists of the following steps:

1. In the first step, the algorithm assigns an initial label to each vertex in the graph

2. In each iteration, for each vertex, the algorithm concatenates its current label with the sorted list of labels of its neighbors forming a multiset of labels. This process captures the structural information around each vertex

3. For each vertex now, the aggregated multiset of labels is compressed via a hash function to form a new label

4. Then the steps 2 and 3 are repeated for a fixed number of iterations or until convergence

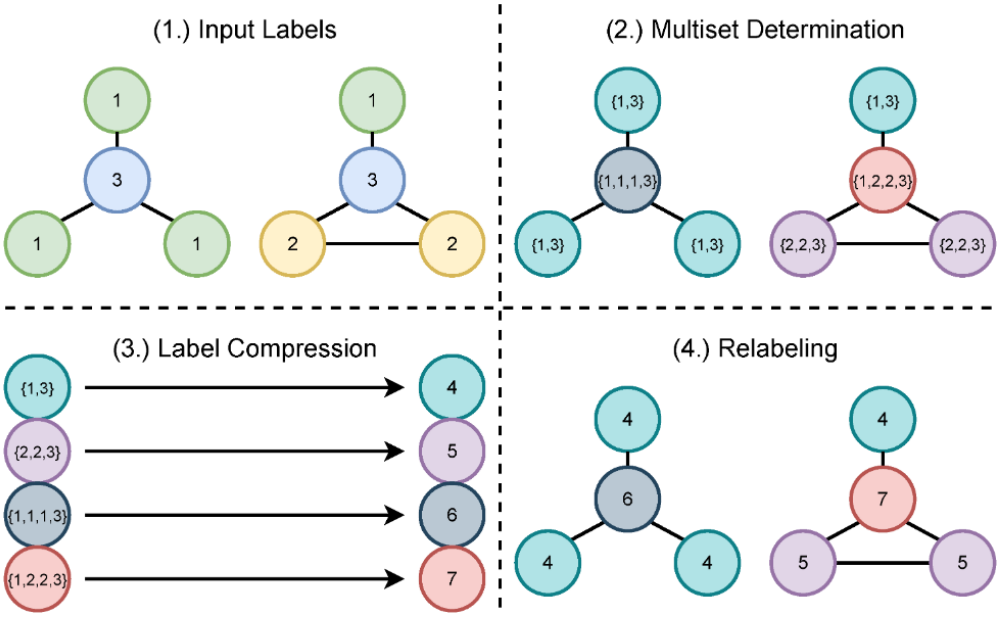


Figure 5.2.3: Illustration of the label refining process [2]

The number of iterations used are usually denoted as h, which means that for two graphs G and $G^{'}$ the Weisfeiler-Lehman is performed at heights from 0 to h. We define as $G_0 = G$ the graph with the original labels and as G_i the graph after applying i iterations of label refining. As a result, the algorithm computes the graphs $\{G_0, G_1, ..., G_h\}$ until height h.

In this context, the WL kernel for h iterations, which operates on top of a base kernel denoted as k, is defined as:

$$k_{WL}(G, G') = k(G_0, G_0{}') + k(G_1, G_1{}') + ... + k(G_h, G_h{}')$$

A common selection for the kernel k is the Vertex Histogram kernel and it is based on the vertex histogram of a graph G. The vertex histogram is defined as a vector $f = (f_1, f_2, \ldots, f_d)$, such that $f_i = |v \in V : \mathrm{l}(v) = i|$ for each $i \in L$ where L is the set of all labels. The node labels involved in this computation are the final hashed labels calculated by the Weisfeiler-Lehman iterative process. Practically, the vector f at point i is equal to the number of vertices possessing the label i. Then, the vertex histogram kernel is defined as:

$$k(G, G^{'}) = \left\langle f, f^{'} \right\rangle$$

In this way, the WL kernel computes the similarity of two graphs.

Shortest Path Kernel

One of the very first, and most influential, graph kernels is the shortest-path (SP) kernel introduced in [13]. The shortest-path kernel breaks down graphs into shortest paths and evaluates pairs of such paths based on their lengths and the labels of their endpoints. The initial step involves transforming the input graphs into shortest-paths graphs. For a given input graph $G = (V, E)$, a new graph $S = (V, E_s)$ is created, referred to as the shortest-path graph. In order to calculate S, some all-pairs shortest path algorithm is employed, such as Floyd-Warshall. This graph shares the same set of vertices as the original graph G, and its edge set is a superset of G's edges, connecting all vertices reachable by walks in G. To complete the transformation, labels are assigned to edges in S, with each label representing the shortest distance between its endpoints in G.

Let G_i, G_j be two graphs, and S_i, S_j their corresponding shortest-path graphs. The shortest-path kernel is defined on $S_i = (V_i, E_i)$ and $S_j = (V_j, E_j)$ as:

$$k(S_i, S_j) = \sum_{e_i \in E_i} \sum_{e_j \in E_j} k^{(1)}_{\text{walk}}(e_i, e_j)$$

where $k^{(1)}_{\text{walk}}(e_i, e_j)$ is a positive semidefinite kernel on edges of length 1.

In labeled graphs, the $k^{(1)}_{walk}(e_i, e_j)$ kernel is designed to compare both the lengths of the shortest paths corresponding to edges e_i and e_j, and the labels of their endpoint vertices.

The kernel $k^{(1)}_{\text{walk}}(e_i, e_j)$ is usually the dirac kernel:

- For unlabeled graphs: $k_{\text{edge}}(e_1, e_2) = \delta(\ell(e_1), l(e_2)) = \begin{cases} 1, & \text{if } \ell(e_1) = \ell(e_2), \\ 0, & \text{otherwise} \end{cases}$
- For labeled graphs: $k_{\text{edge}}(e_1, e_2) = \delta(\ell(e_1), \ell(e_2)) = \begin{cases} 1, & \text{if } \ell(e_1) = \ell(e_2) \wedge (\ell(e_1^1) = \ell(e_2^1)) \wedge (\ell(e_1^2) = \ell(e_2^2)) \\ 0, & \text{otherwise} \end{cases}$

  where we denote as ℓ the label of a vertex and e^1, e^2 are the two endpoints of the corresponding edge.

In order to compute the shortest path kernel, the process involves calculating the shortest paths for every pair of vertices in both graphs which takes $O(n^3)$ time and comparing all pairs of shortest paths between the two graphs something that needs $O(n^4)$ time. As a result, the overall runtime complexity of the Shortest Path Kernel is $O(n^4)$ which becomes impractical for large graphs.

Random Walk Kernel

The Random Walk Kernel is a well-established and extensively investigated family of kernels. Kernels within this category primarily focus on enumerating matching walks in the two input graphs. There are several variations of random walk kernels. The k-step random walk kernel compares random walks up to length k in the two graphs. The most widely used member of this family of kernels is the Geometric Random Walk Kernel [33], which compares walks when $k \to \infty$.

In order to proceed to the formal definition of the Geometric Random Walk Kernel we will first introduce some basic concepts. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their product-graph denoted as $G_\times$ is computed as:

$$V_\times = \{(v_1, v_2) : v_1 \in V_1 \wedge v_2 \in V_2\} \text{ where for labeled graphs we also need } \ell(v_1) = \ell(v_2)$$
$$E_\times = \{\{(v_1, v_2), (u_1, u_2)\} : \{v_1, u_1\} \in E_1 \wedge \{v_2, u_2\} \in E_2\}$$

A visualization of the construction of the product graph can be seen in Figure 5.2.4. In general, it is true that the k-th power of the adjacency matrix A of graph G computes walks of length k, which means that A^k_{ij}, represents the number of walks of length k from vertex i to vertex j. It is now evident that performing a random walk on $G_\times$ is equivalent to conducting a simultaneous random walk on G_1 and G_2. This mean that common walks of length k can be computed using $A^k_\times$. Now we can define the geometric random walk kernel for all pair of vertices belonging to $|V_\times|$ and for paths up to infinity as:

$$K^\infty_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \sum_{r=0}^{\infty} [\lambda^r A^r_\times]_{ij} = e^T (I - \lambda A_\times)^{-1} e$$

where e is the all-ones vector, and λ are positive weights used to ensure convergence of the corresponding geometric series. The geometric random walk kernel converges only if $\lambda < \frac{1}{\lambda_\times}$, where $\lambda_\times$ is the largest eigenvalue of $A_\times$. Direct computation of the geometric random walk kernel requires $O(n^6)$ time severely limiting its applicability to real-world applications. However, various optimizations [85] have succeeded in reducing it to $O(n^3)$.

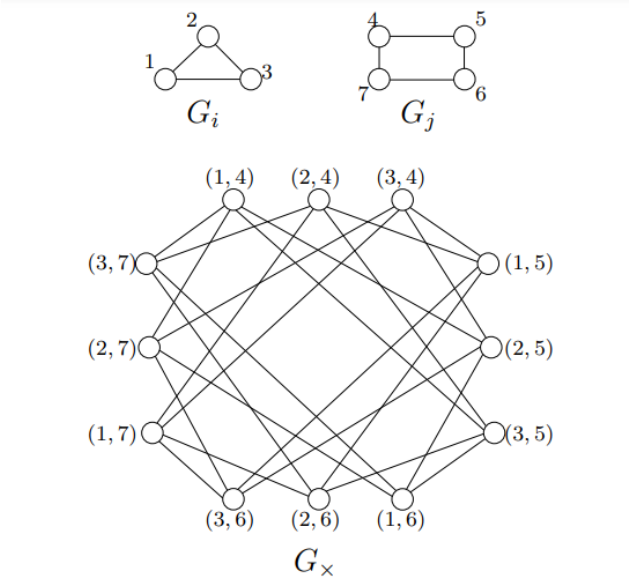


Figure 5.2.4: Illustration of the construction of the product graph [61]

Neighborhood Hash Kernel

The Neighborhood Hash Kernel proposed in [38] works very similarly to the Weisfeiler-Lehman Kernel. In particular, it is designed for graphs with labeled nodes and its approach to comparing graphs involves changing node labels and keeping track of the shared labels. To achieve this, the kernel replaces discrete node labels with fixed-length binary arrays. These arrays are then updated using logical operations to incorporate information about the neighborhood structure of each vertex into the labels. As we have already mentioned, this kernel transforms each discrete node label to a fixed-length binary array consisting of d bits as:

$$s = \{b_1, b_2, \ldots, b_d\}$$

where the constant d is sufficiently large to cover all possible discrete node labels. The most important step of the algorithm as in the WL kernel, is the update of the nodel labels using the neighbourhood information. This steps involves the usage of the binary operations XOR and the left cycle-rotation ROT. Using this update rule, the new label (hash) of node v, denoted as $NH(v)$, is computed as:

$$NH(v) = \text{ROT}(\ell(v)) \oplus \left(\bigoplus_{i=1}^{d} \ell(u_i) \right)$$

where $N(v) = \{u_1, \ldots, u_d\}$ is the set of neighbors of v. The resulting hash $NH(v)$ is still a bit array of length d. After the neighbourhood hash update rule we presented above is applied, the kernel to compare two graphs G and $G^{'}$ is defined as:

$$k(G, G') = \frac{c}{|V| + |V'| - c}$$

where c is the number of labels the two graphs have in common. By updating the bit labels several times, the new labels can capture high-order relationships between vertices across their h-hop neighbourhood. Hence, by updating the node labels of the two graphs $\{G\}$ and $\{G'\}$ for h times the similarity of this pair of graphs can be calculated as:

$$k(G, G') = \frac{1}{h} \sum_{i=1}^{h} k(G_i, G'_i)$$

The computational complexity of the neighborhood hash kernel is $O(dhn\bar{D})$, where $n = |V|$ is the number of vertices of the graphs, and $\bar{D}$ is the average degree of their vertices.

Graphlet Kernel

The Graphlet Kernel, firstly introduced in [67] and optimized in [73] using sampling, defines a similarity measure between two graphs by considering the presence and frequency of specific graphlets in each graph. The more similar the graphlets' distribution between two graphs, the higher their graphlet sampling kernel similarity score. Graphlets are small subgraphs within a larger graph. Graphlet sampling involves extracting these small subgraphs from the original graph. The size and structure of graphlets can vary, and they are often used to capture local patterns or motifs within the graph. Typically, graphlets contain k nodes, where k is usually a small number like 3, 4, or 5.

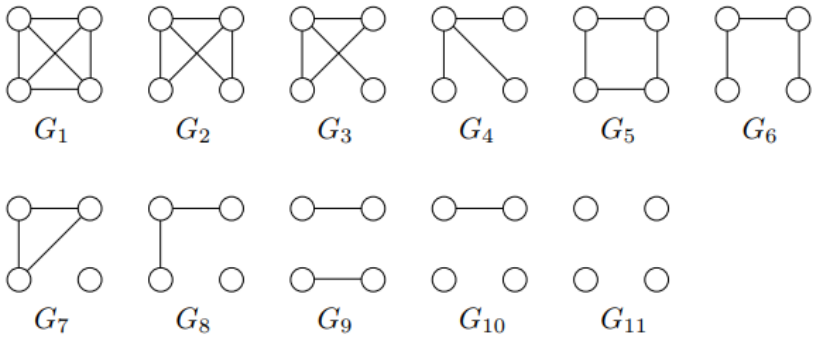


Figure 5.2.5: All graphlets of size 4 [61]

The set of size-k graphlets is usually denoted as $G = \{\text{graphlet}_1, \text{graphlet}_2, \ldots, \text{graphlet}_d\}$. Moreover, the d-dimensional vector $f_G \in N^d$ has in its i-th entry the frequency of occurrence of graphlet_i in the graph G, typically denoted as $\#(\text{graphlet}_i \subseteq G)$. Given two graphs G and $G^{'}$ to be compared, the vectors f_G and $f_{G'}$ are calculated. The graph kernel is easily calculated now as:

$$k(G, G^{'}) = \langle f_G, f_{G'} \rangle$$

However, there are $\binom{n}{k}$ size-k subgraphs in a graph and as a result, the exhaustive calculation of graphlets, requires exponential time $O(n^k)$, which is computationally prohibitive. For this purpose, the authors of [73], showed that by sampling a fixed number of graphlets the empirical distribution of graphlets will be sufficiently close to their actual distribution in the graph, thus mitigating the high computational cost of the initial approaches.

5.3 Scene Graphs

In this chapter, we have already explored some basic concepts of graphs as well as different approaches of computing the similarity between pairs of them. Graph-structured data is omnipresent in various applications due to its ability to represent and model complex relationships and interactions among different entities. From Social Networks, where users and their connections are represented as nodes and edges, Biological Networks, where nodes represent biological entities (e.g., proteins, genes) and edges represent interactions between them, to Knowledge Graphs and Transportation Networks, graph representation is an essential tool to analyze the properties of such structures.

In the dynamic intersection of computer vision and visual data understanding, scene graphs emerged as foundational structures. Scene Graphs were first proposed in [40] in order to enhance image retrieval and since then have attracted the attention of a large number of researchers. In the context of this thesis, the graph structure will be used in order to represent the scene depicted in an image. This type of representation, called scene graph, includes valuable information about objects present in an image as well as the relationships

among them. Scene graphs play a vital role in providing a structured framework to understand relationships between objects and their surroundings. In practice, they are represented as directed graphs where nodes represent entities (objects) such as cars, people, buildings, among others, and the edges represent their relationships using the following triplet structure <subject, relation, object>.

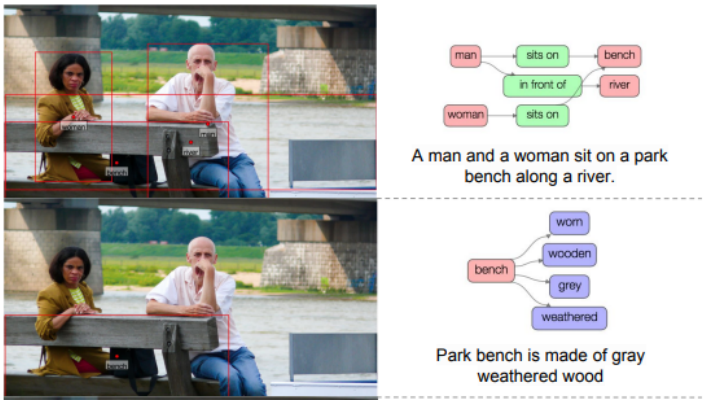


Figure 5.3.1: An example of regions of an image along with their corresponding scene graphs from the Visual Genome Dataset [46]

Leveraging scene graphs enhances scene understanding by capturing the semantic content of an image in a more detailed manner as well as the relationships among entities something that goes beyond traditional object detection and recognition approaches. They form a robust foundation for image captioning systems, facilitating the generation of contextually rich and accurate captions. For Visual Question Answering (VQA) tasks, scene graphs offer a structured understanding of visual elements where the image and the semantics of the corresponding scene graph are more effectively understood due to the fact that, unlike viewing it merely as a collection of pixels, the scene graph is perceived as an assembly of interconnected entities, facilitating a more effective interpretation. Using Scene Graphs in VQA models enhances significantly their performance [93].

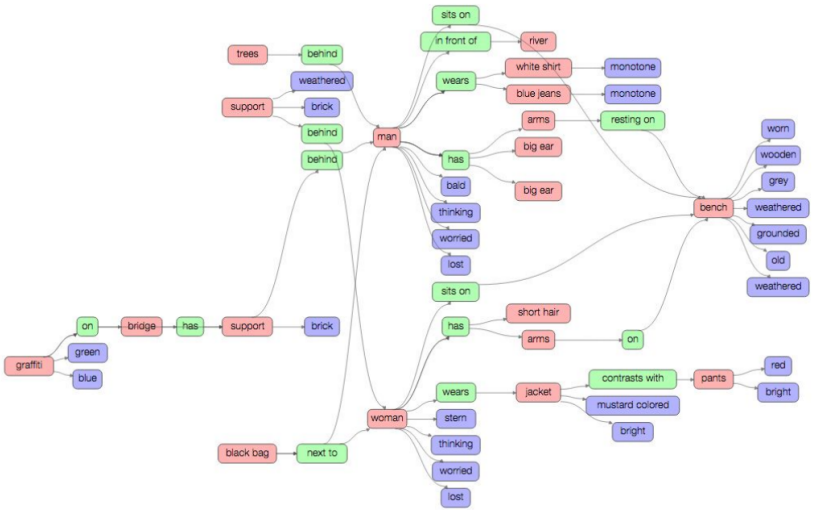


Figure 5.3.2: The Complete Visual Genome Scene Graph of the image depicted in Figure 5.3.1. It is evident that this scene graph contains different objects, attributes and relationships, defining effectively that semantic content of the corresponding image [46].

The most important Scene Graph dataset that we will use in the context of this thesis, is the Visual Genome Dataset introduced in [46]. Comprising over 100,000 images, it stands as one of the largest datasets for visual understanding tasks. Each image in the dataset is densely annotated, providing a wealth of information,

including object labels, object relationships, region and scene descriptions. Notably, the annotations extend to multiple instances of the same object within an image, contributing to a nuanced comprehension of object variations and placements. Going beyond mere object recognition, Visual Genome incorporates annotations for object relationships, detailing how objects within a scene relate to each other facilitating the capture of hierarchical structures within visual scenes.

Another important application of scene graphs pertains to computer vision, where the creation of scene graphs from images as well as the reverse process are significant. Numerous studies concentrate on Scene Graph Generation, employing conventional as well as neural methods. In particular, when generating an image, employing a graph instead of plain text offers more comprehensive information about object relationships that sentences are unable to convey.

5.4 Related Work

In this thesis, our focus will be on the exploration of graph similarity for scene graphs. As we have already seen, graph similarity has been extensively studied over the years. Starting with the introduction of Graph Edit Distance [70] as a graph similarity metric, to other similarity measures such as Maximum Common Subgraph [18] and Graph Isomorphism [9], which are methods with NP-hard complexities as they rely on the problem of graph isomorphism, a significant scientific endeavor has been dedicated to addressing this problem.

Apart from these traditional approaches, many optimizations have been proposed in order to speed up the execution of graph similarity algorithms. These methods use various heuristics, such as A* and Beam Search, to finally compute GED approximations and we have discussed them earlier in this chapter. In the context of this thesis, in order to obtain a ground truth measure for evaluation, we will use the bipartite matching algorithm which formulates the problem as a Linear Sum Assignment Problem (LSAP) which is solved using the Jonker-Volgenant assignment algorithm [41]. The selection of this approach is due to the fact that it provides an efficient balance between the accuracy of the approximate solution and computational cost.

However, the central focus of this thesis lies on neural approaches, particularly Graph Neural Networks (GNNs). GNNs inherently incorporate graph structures at the node level producing meaningful embeddings with respect to the complex structure of graphs. Consequently, one approach to address the graph similarity problem, is to use Graph Convolutional Networks or other GNN layers that are able to produce permutation invariant representations. Unlike this family of approaches, which treats each graph individually during training, other approaches work on pairs of graphs comparing them during the training stage. For instance, SimGNN [4] combines graph with node level information, by proposing an attention mechanism to emphasize crucial nodes, based on a selected similarity metric. Similarly, Graph Matching Networks [50] also compute a joint similarity score of a pair of graphs by using cross-graph attention. UGraphEmb [5] employs a siamese GNN to train on graph pairs. In practice, during the training process, regression is performed on a precomputed graph edit distance measure for each pair, placing this method within the realm of supervised learning. However, the computation step needed to calculate the graph edit distance measure for each pair of graphs is quite expensive.

It should be noted that the approaches mentioned earlier are applied to graph datasets that represent proteins, citation networks and program dependency graphs among others. None of these approaches are designed for scene graphs. Moreover, Contrastive Learning GNNs for unsupervised graph similarity have not been studied for scene graphs, at least to the best of our knowledge. In general, the attention on scene graph data in the context of graph similarity applications has been limited. The most relevant publication focused on producing scene graph embeddings for similarity is [56] which proposes using GCN and a weak supervision signal derived from the caption similarity of the corresponding scene graphs and do not relates to Graph Edit Distance.

Chapter 6

Graph Neural Networks (GNN)

As it has been established from the previous chapter, graph-structured data is prevalent in various domains such as social networks, biological systems, molecules, materials, knowledge graphs and recommendation engines and many other research areas. Graphs, ubiquitous in various applications, encode invaluable relationships often challenging for traditional neural networks to capture effectively. For this reason, Graph Neural Networks (GNN), a different type of neural networks operating specifically on graph data, were invented. Graph Neural Networks (GNNs) are a subset of models within the broader field of Geometric Deep Learning (GDL) [15], [16] which focuses on developing deep learning techniques for non-Euclidean structured data, including graphs, point clouds, meshes, and manifolds. The connection between GNNs and geometric deep learning lies in their shared goal of effectively handling data with inherent geometric or relational structures. Adapting deep learning techniques to respect and exploit this inherent geometric structure, beyond the grid-like structures traditionally processed by standard neural networks, is the key difference of these models from traditional neural networks. In this chapter we will cover the most significant aspects of GNNs, present the most important GNN variants, explain how we use them and how contrastive learning can work in this context.

Contents

6.1 Machine Learning on Graphs . . . 76
6.1.1 Motivation . . . 76
6.1.2 Permutation Invariance and Equivariance . . . 76
6.2 Spectral Approaches . . . 78
6.2.1 Elements of Graph Spectral Theory . . . 78
6.2.2 Spectral Variants . . . 79
6.3 Spatial Approaches . . . 81
6.3.1 General Framework . . . 81
6.3.2 Spatial Variants . . . 82
6.4 How to Use Graph Neural Networks . . . 87
6.4.1 Exploring Task Types in Graph Neural Networks . . . 87
6.4.2 Diverse Training Approaches for Graph Neural Networks . . . 88
6.5 Contrastive Learning on Graphs . . . 88
6.5.1 Contrasting Modes . . . 88
6.5.2 Graph Contrastive Learning with Augmentations (GraphCL) . . . 89
6.5.3 InfoGraph . . . 91
6.5.4 Deep Graph InfoMax (DGI) . . . 92
6.5.5 Deep Graph Contrastive Representation learning (Grace) . . . 93

6.1 Machine Learning on Graphs

6.1.1 Motivation

The inspiration for graph neural networks (GNNs) can be traced back to the remarkable success of convolutional neural networks (CNNs) and their innovative concept of convolution filters applied in images. Images can be viewed as a special instance of graph structured data in which the nodes are the pixels and the edges represent adjacency between pixels. Convolution filters are based on the spatial locality of pixels, which as we said can be seem as nodes on the grid, by sliding rectangular kernels with a small receptive field over the image to produce various feature maps. CNNs have demonstrated unparalleled efficiency in capturing spatial hierarchies and local patterns in grid-like data. Graph neural networks emerged as a natural evolution, adapting the convolutional framework to graphs, enabling the extraction of meaningful features and relationships from complex networks. Specifically, GNNs define an embedding vector for each of the nodes, usually initialized with inherent node properties, which are then transformed by a sequence of learnable layers. The extension of the convolutional framework to graphs is based on the observation that we can view convolution on graphs for a specific node i, as gathering information from the neighbouring nodes by passing messages from the neighbours to node i. However, graphs do not have an inherent ordering of their nodes and as the number of possible permutations increases factorially with the number of nodes, GNNs should incorporate notions such as invariance and equivariance in their architecture.

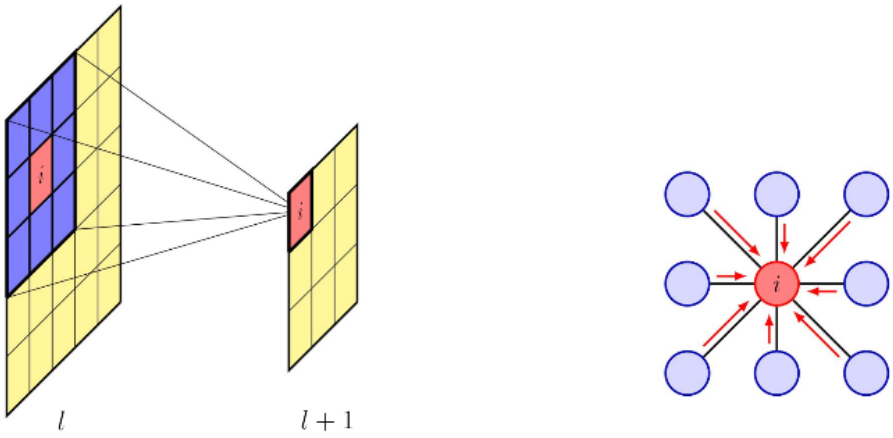


Figure 6.1.1: View of the convolutional receptive field for images and graphs [11]

6.1.2 Permutation Invariance and Equivariance

A graph is usually represented with an adjacency matrix denoted by A, a convenient way to specify the edges between nodes in the graph. However, in order to define the adjacency matrix, first a specific ordering of the nodes must be defined. If there are N nodes in the graph, we index them using $n = 1, ..., N$ and as a result the adjacency matrix A has dimensions $N \times N$ where $A_{i,j} = 1$ if there is an edge connecting nodes i and j (for weighted graphs the value of $A_{i,j} = w_{i,j}$) and $A_{i,j} = 0$ otherwise. For undirected graphs, the adjacency matrix will by symmetric. Moreover, for each node n of the graph we have a D-dimensional vector x_n representing the features of this node. We can stack all these vectors into a feature matrix X of dimensionality $N \times D$ where every row n is given by the vector x_n. In order to use this representation an ordering of the nodes should be predefined. However, the graph and its properties remain the same regardless of the decided ordering of the nodes. In order to address this challenge the neural network should learn a function which respects the various symmetries of the graph.

A *permutation matrix* $\mathbf{P}$ specifies a particular permutation of the node ordering and it has the same dimensionality $N \times N$ with the adjacency matrix. In particular, $\mathbf{P}$ contains a single 1 in each row and a single 1 in

each column, with 0 in all other positions. For example, if we have a graph with 4 nodes we can define $\mathbf{P}$ as:

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

If we name our nodes as (A, B, C, D) this permutation matrix corresponds to the transformation $(A, B, C, D) \rightarrow (B, C, D, A)$ of the node ordering. When we perform such a node re-ordering, this results in the permutation of the rows of the feature matrix $\mathbf{X}$. This can be achieved with matrix multiplication as following:

$$\tilde{X} = PX$$

For the adjacency matrix however, both the rows and the columns must be permuted. The rows can be permuted as before by left-multiplication of $\mathbf{P}$ with the adjacency matrix and the columns can be permuted by right-multiplication with $\mathbf{P}^T$. This gives rise to the permuted adjacency matrix:

$$\tilde{A} = PAP^T$$

As our goal is to learn a function that will not depend on a specific node ordering, we should design a permutation invariant function to any permutation of the nodes.

A *permutation invariant* function f is defined as:

$$f(PX) = f(X)$$

where X for the case of graph neural networks is the feature matrix, and $\mathbf{P}$ is a permutation matrix. This equation should hold for every permutation matrix $\mathbf{P}$. However, although such a permutation invariant function is suitable to ensure that any global property of the graph does not depend on the node ordering, it destroys information for each individual node. This happens, because it practically treat nodes as sets and not each one individually [57].

For this reason, if we want to make predictions in the node level, a node re-ordering should be reflected to our predictions by permuting correspondingly the node features in the output. This lead us to the notion of *permutation equivariance*. A *permutation equivariant* function f is defined as:

$$f(PX) = Pf(X)$$

which should be true for every permutation matrix $\mathbf{P}$. In practice, node-level outputs are very valuable for various downstream tasks, such as node classification.

Summarizing and extending the above definitions to include both the feature and the adjacency matrix, where when we permute the nodes, we expect the edges to accordingly act and be permuted identically, we have the following definitions for invariance and equivariance:

- Invariance

$$f(PX, PAP^T) = f(X)$$

- Equivariance

$$f(PX, PAP^T) = Pf(X)$$

As we will see later on, graph neural networks are usually constituted of a number of equivariant layers and if we need predictions on the level of the entire graph, a global pooling layer which is invariant to permutations is finally used. After we have presented the core ideas of invariance and equivariance we will proceed with the idea of Graph Convolution. Graph Convolution is at the core of processing graph data and the research community has well studied both spectral and spatial approaches, which we will present in the two following sections.

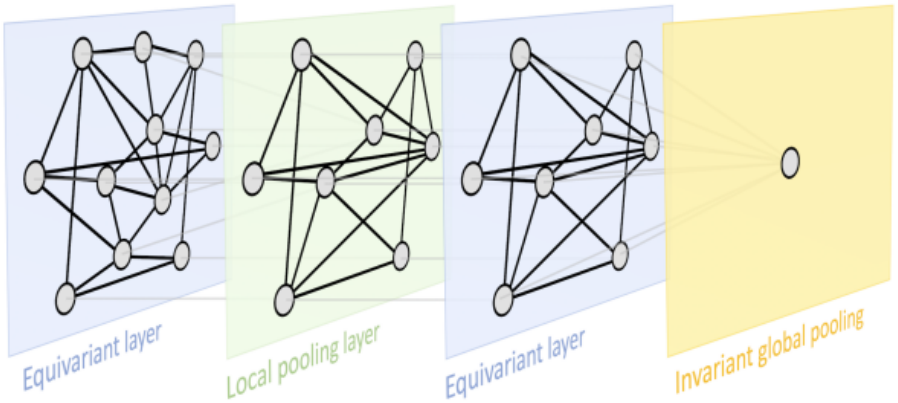


Figure 6.1.2: A typical GNN may contain permutation equivariant layers computing node-wise features and a permutation invariant global pooling layer [16]

6.2 Spectral Approaches

It is well established that GNNs are usually divided intro spectral and spatial methods. In this section, we will summarize the ideas of graph spectral theory and their relation to graph convolution. Graph Convolution, from a graph signal processing perspective (GSP) [63], [26], can be viewed as a function acting on a graph signal. Usually, a graph signal is defined on the set of vertices V of a graph, giving a real number as value to each vertex of the graph.

6.2.1 Elements of Graph Spectral Theory

In graph signal processing it is common to assume undirected graphs. In this context, a core element of a graph, which is very useful in the spectral domain, is the Graph Laplacian. The Graph Laplacian is, like the adjacency matrix, a mathematical representation of an undirected graph and is defined, for a graph with N nodes, as:

$$L = D - A$$

where D is a diagonal degree matrix with $D_{i,i} = \sum_j A_{i,j}$ and A is the adjacency matrix. However, usually the normalized Laplacian is used. Normalizing the Laplacian makes the spectral properties of the graph less dependent on the scale of the graph. It allows for better comparison between graphs of different sizes and structures and they often lead to more interpretable spectral properties. The normalized Laplacian is defined as:

$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

The normalized graph Laplacian matrix is real, symmetric and positive semidefinite. With this property, the normalized Laplacian matrix can be diagonalized as:

$$L = U \Lambda U^T$$

where $U = [u_0, u_1, ..., u_{n-1}] \in R^{N \times N}$ is the matrix of eigenvectors and Λ is the diagonal matrix containing the eigenvalues (spectrum) of the graph. The eigenvectors of the normalized Laplacian matrix form an orthonormal space because the laplacian L is normalized and thus $U^T U = \mathrm{I}$. As we mentioned before, in the field of graph signal processing, a graph signal denoted as $x \in R^n$ is a feature vector of all the nodes in the graph where x_i is the value of the ith node. Because $\mathrm{L} = U \Lambda U^T$ and the eigenvector form an orthonormal

space, the Graph Fourier transform of a signal x and the inverse graph Fourier transform, can be defined [12], [63], [26] as:

$$\hat{x} = U^T x, x = U\hat{x}$$

where $\hat{x}$ represents the Graph Fourier Transform of signal x. It is obvious that the Graph Fourier Transform projects the input graph signal to the orthonormal space defined by the eigenvectors of the normalized laplacian. The convolution theorem states that the Fourier transform of a convolution between two signals is equivalent to the pointwise multiplication of their Fourier transforms. Given this, the Graph Fourier Transform, a graph signal x, and a filter $g \in R^n$ the graph convolution operation is defined as:

$$x * g = U((U^T g) \odot (U^T x))$$

where $\odot$ is the element-wise Hadamard product. By denoting $g_\theta(\Lambda) = diag(U^T g)$, we have:

$$x * g_\theta = U g_\theta U^T x$$

where $g_\theta = diag(\theta)$ is the diagonal matrix that corresponds to the spectral filter coefficients. This spectral filter is of great significance as the various versions of spectral-based Graph Convolutional Networks (GCN) differ based on the selection of this filter.

6.2.2 Spectral Variants

Spectral Convolutional Neural Network

Spectral CNN [17] treats the filter $g_\theta = diag(\theta)$ with parameters $\theta \in R^n$ as the parameters of the neural network and optimizes them by gradient descent. However, this method suffers from high computational complexity because eigendecomposition is computationally expensive, it depends on the input graph's structure, and it has non-spatial locality.

Chebyshev Spectral CNN

In ChebNet [22] the basic idea is to approximate the parameterized filter $g_\theta(\Lambda)$ with polynomials such as:

$$g_\theta(\Lambda) = \sum_{k=0}^{K} \theta_k \Lambda^k$$

where $\theta \in R^K$ is a vector of polynomial coefficients. As a result of the presence of Λ^k in the polynomial filter and because $d_G(i,j) > K$ implies that $L^K(i,j) = 0$, these spectral filters are exactly K-localized. However, the computation cost is still high and for this reason Chebyshev polynomials $T_k(x)$ of order k which can be computed recursively from the graph laplacian are proposed. Specifically, the following recursive relation holds for Chebyshev polynomials of order k:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

with $T_0 = 1$ and $T_1 = x$. The convolution is now defined as:

$$x * g_\theta = \sum_{k=0}^{K} \theta_k^{'} T_k(\tilde{L}) x$$

where $\tilde{L} = \frac{2L}{\lambda_{max}} - I_n$ where λ_{max} is the largest eigenvalue of the graph laplacian L. By the diagonalization of the graph laplacian it is easy to see that:

$$T_k(\tilde{L}) = U T_k(\tilde{\Lambda}) U^T$$

Thus, this kind of filter mitigates the non-locality of Spectral CNN and its high computational complexity.

Graph Convolutional Network

In GCN [44] it was first proposed to conduct a first order approximation of the ChebNet, which results in limiting the layer-wise convolution operation to $K = 1$, mitigating in this way effectively the problem of overfitting. By setting $K = 1$ and approximating $\lambda_{max} \approx 2$ the equation $x * g_\theta = \sum_{k=0}^{K} \theta_k^{'} T_k(\tilde{L})x$ transforms to a linear approximation defined as:

$$x * g_\theta \approx \theta_0^{'} x + \theta_1^{'}(L - I_N)x \approx \theta_0^{'} x - \theta_1^{'} D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

By setting $\theta = \theta_0^{'} = -\theta_1^{'}$ we constrain the number of parameters and minimize the number of operations, such as matrix multiplications per layer, which leads to the simpler expression:

$$x * g_\theta \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x$$

A filter of this form, as it constitutes a first order approximation of ChebNet, convolves effectively the 1-hop neigbourhood of a node. Furthemore, successive application of filters of this form convolve the k th-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model. However, $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigemvalues in the range $[0, 2]$, and thus stacking multiple layers of this type can lead to exploding/vanishing gradients. To address this problem, the authors propose the following renormalization trick $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I_N$ and $\tilde{D} = \sum_j \tilde{A}_{ij}$. Now, the convolution operator takes the following compact form:

$$X^{'} = x * g_\theta \approx \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X\Theta$$

where $X \in R^{N \times C}$, $\Theta \in R^{C \times F}$, $Z \in R^{N \times F}$, C is the dimensionality of the input node vectors and F is the dimensionality of the output node vectors. After the convolutional operator is applied, a non linear activation function (e.g. ReLU) is usually applied to the output matrix X.

Summarizing, the propagation rule which governs a deep neural network with GCN layers is:

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

where $W^{(l)}$ are the trainable parameters of the l-th layer, $H^{(l)}$ is the matrix representing the node features as calculated until layer l and σ is a non-linear activation function.

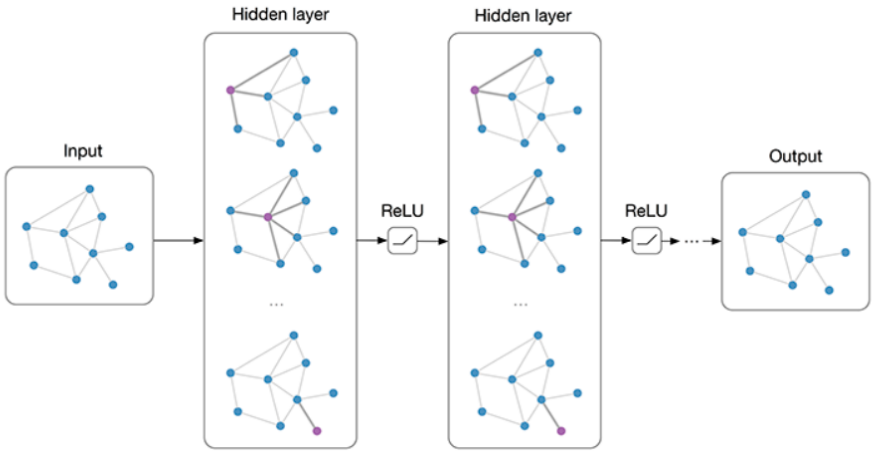


Figure 6.2.1: Multi-layer Graph Convolutional Network (GCN) with first-order filters

The equation governing the GCN layer can be also perceived as a spatial approach, thus bridging the gap between spectral and spatial approaches. In particular, we can rewrite the equation $X^{'} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$ in a node-wise manner, underlying the spatial nature of the GCN layer:

$$x_i^{'} = \Theta^T \sum_{j \in N(i) \cup i} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} x_j \tag{6.2.1}$$

This equation represents that the updated vector for node i is calculated if we sum the normalized node vectors of all its neighbouring nodes (including also itself) and multiply the result with a matrix of trainable parameters. This model proved to be very efficient on a wide variety of tasks and is probably one of the most significant breakthroughs in the field of GNNs inspiring many other important variants.

6.3 Spatial Approaches

As we have seen in the previous section, GCN [44] played a pivotal role in bridging the gap between spectral and spatial approaches. Spatial approaches use the spatial locality of nodes in order to define the graph convolution. In this section we will describe the basic framework for most spatial GNNs as well as some of the most important spatial architectures.

6.3.1 General Framework

The proliferation of diverse GNN variants has accentuated the pressing need for a unified and comprehensive framework. In this section, we will describe the most prevalent framework for spatial Convolutional GNNs.

Message Passing Neural Network (MPNN) [34] is the most important framework describing the family of Convolutional GNNs. The procedure is divided into two phases, namely the message passing and an optional readout phase for graph-level tasks. The goal is to define a nonlinear transformation of node embeddings that is differentiable with respect to a set of weight parameters and which maps the node embeddings of layer l into corresponding node embeddings in layer $l+1$. For each node n in the graph and for each layer l of the network we define as h_n^l the D-dimensional vector of the embedding of node n at layer l.

The first phase, can be divided into two successive stages, usually defined by Aggregate and Update functions. In the aggregation stage, for each node n, messages are passed to that node from its neighbours and are combined to form a new vector z_n^l. The aggregation function is defined as:

$$z_n^l = Aggregate(\{h_m^l : m \in N(n)\})$$

The way z_n^l is formed, is permutation invariant as it does not depend on the ordering of the neighbours. The Aggregate function is quite flexible and it can also contain learnable parameters. The second stage, called update, which is defined as:

$$h_n^{l+1} = Update(h_n^l, z_n^l)$$

updates the embedding vector of node n. Usually there is a non linear function as part of the Update operation. An Aggregate function followed by an Update one, constitutes one layer of the network. Multiple layers of this type can be stacked in order to extend the message passing beyond the 1-hop neighbourhood of each node.

The readout phase as we said is optional and it is usually used when graph level predictions are required. In this case, the graph representation is obtained by employing the readout function R to all the embedding vectors of the nodes produced by the final message passing layer, which we name K. Then, the readout function is defined as:

$$h_G = Readout(h_n^K | n \in G)$$

where h_G is the graph embedding. Generally, the aforementioned framework is easily extended to also include edge features. By denoting e_{nm}^l as the edge embedding vector of the edge connencting nodes n and m as computed from the l-th layer of the network we can extend the message passing framework to also include edge features. The message passing equations are extended with:

$$e_{nm}^{l+1} = Update_{edge}(e_{nm}^l, h_n^l, h_m^l)$$
$$z_n^{l+1} = Aggregate_{node}(\{e_{nm}^{l+1} : m \in N(n)\})$$
$$h_n^{l+1} = Update_{node}(h_n^l, z_n^{l+1})$$

where the edge embeddings are updated by combining their previous state with the state of the nodes that are connected to this edge.

There are many different possible forms of the Aggregate function. The simplest ones, that do not depend on the ordering of the input, include summation $Aggregate(\{h_m^l : m \in N(n)\}) = \sum_{m \in N(n)} h_m^l$ and average $Aggregate(\{h_m^l : m \in N(n)\}) = \frac{1}{|N(n)|} \sum_{m \in N(n)} h_m^l$ where [44] is a variation of this approach.

Another approach includes learnable parameters inside the aggregate function in the form of:

$$Aggregate(\{h_m^l : m \in N(n)\}) = MLP_\theta \left(\sum_{m \in N(n)} MLP_\phi(h_m^l) \right)$$

As a result of the flexibility of MLPs, the above equation can approximate any permutation invariant function that maps a set of embeddings to a single embedding [91].

As with the Aggregate function, the Update function, given the aggregated vector z_n^l and the hidden representation h_n^l of node n, is usually generally defined as:

$$Update(h_n^l, z_n^l) = f(W_a h_n^l + W_b z_n^l + b)$$

where f is a nonlinear activation function such as ReLU. If we set $W_a = W_b = W$ the updated state of node n is then defined as:

$$h_n^{l+1} = Update(h_n^l, z_n^l) = f(W \sum_{m \in N(n), n} h_m^l + b)$$

As we have seen, the GCN operation from [44] can be described with the equation 6.2.1. This equation can be easily derived from the MPNN framework we defined above. Using the MPNN framework we can describe a variety of different spatial convolutional GNNs. Their difference lies mainly in the way they define the aggregation, update and readout functions.

6.3.2 Spatial Variants

In this section we will present some of the most important spatial GNN variants, the majority of which can be derived as a variant of the MPNN [34].

Graph Attention Network (GAT) [83] proposes to adopt the idea of attention presented in [82] which is very powerful in the context of the transformer architecture, for graphs. Specifically, [83] proposes a variant of the Aggregate function of the MPNN framework that combines the messages from neighbouring nodes in a weighted manner. The weights are defined as learnable parameters of the neural network. The intuition behind GAT is that by weighting the incoming messages from the neighbours, it can capture an inductive bias that says that the messages from some neighbours may be more important from the messages of other neighbours.

The updated state of node i is now derived from the following equation:

$$h_i^{l+1} = f(\sum_{j \in N(i)} \alpha_{ij} W h_j^l)$$

where f is a non-linearity applied to the weighted aggregation of neighbooring nodes.

The learnable attention weights are calculated as:

$$\alpha_{ij} = softmax(LeakyReLU(\alpha^T [Wh_i^l || Wh_j^l]))$$

where the variable α^T denotes a set of learnable parameters usually a MLP, the softmax function is used to ensure that the attention weights sum to one and $||$ is the concatenation operation.

This *self-attention* mechanism described above, can be extended by introducing multiple attention heads, where K independent self-attention mechanisms are applied to compute the hidden states of the nodes. The final embeddings of the nodes can then be calculated by either concatenating the K different embedding vectors produced by the different attention mechanisms or by taking the average of them. The equations producing the new embedding of node i using the concatenation or the average operation are the following:

$$h_i^{l+1} = \|_{k=1}^K f(\sum_{j \in N(i)} \alpha_{ij}^k W^k h_j^l)$$

$$h_i^{l+1} = f(\frac{1}{K} \sum_{k=1}^K \sum_{j \in N(i)} \alpha_{ij}^k W^k h_j^l)$$

where α_{ij}^k are the normalized attention coefficients computed by the k-th attention mechanism.

The aforementioned mechanism is called *Multi-head attention*. Usually, when the GAT encoder is defined, the number of attention heads should be also determined.

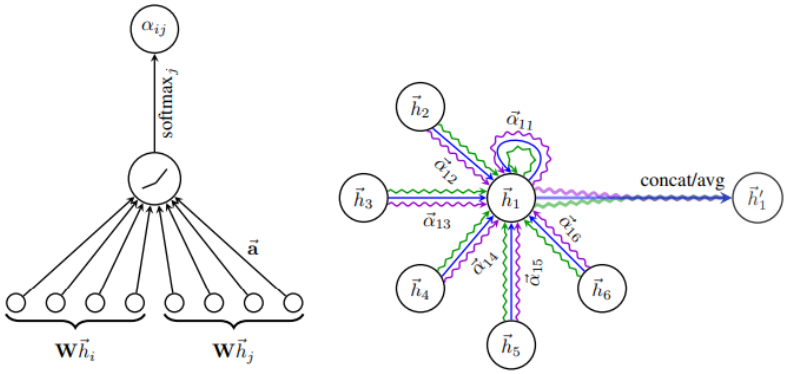


Figure 6.3.1: **Left**: The self-attention mechanism employed by GAT. **Right**: An illustration of multihead attention (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote different attention heads. The aggregated features from each head are concatenated or averaged [83].

GATv2 [14] proved that the original GAT model [83] computes a *static attention* among the nodes of the graph. In particular, GAT uses the attention mechanism to focus on the most relevant nodes in a graph when updating the features of a specific node. However, the *static attention problem* that the standard GAT layer exhibits, means that the ranking of the attended nodes is unconditioned on the query node. Thus, the attention function defines a constant ranking of the nodes, unconditioned on the query node i, which means that the GAT network doesn't rank nodes' attention score depending on the specific node pair, but

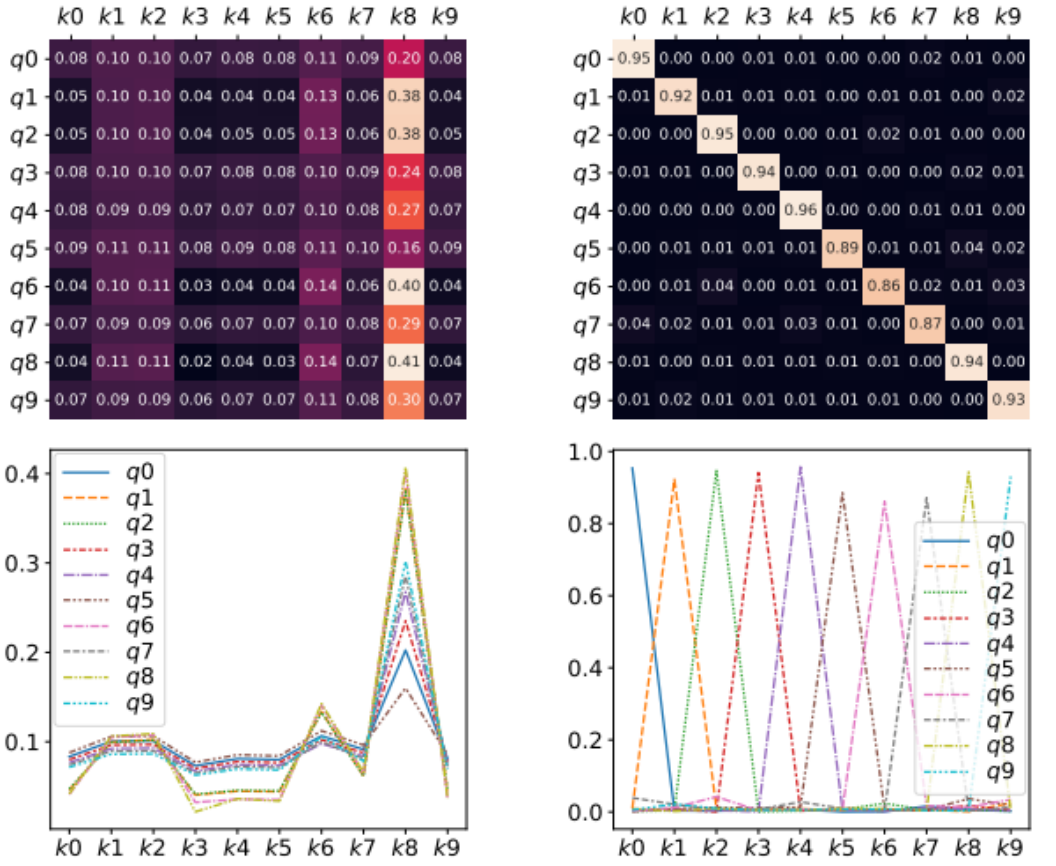
it produces global attention score rankings. In practice, this means that there is a node v in the graph, that all the other nodes assign it the highest attention score and this is analytically proved in [14]. The original GAT layer defines the scores e_{ij} between two nodes i and j as:

$$e_{ij} = LeakyReLU(\alpha^T[Wh_i^l||Wh_j^l])$$

which are then passed through a softmax layer in order to produce the normalized attention weights. The main problem with this formulation is that the learned layers W and α^T (which is a simple MLP) are applied consecutively, and thus can be collapsed into a single linear layer, contstraining the expressive power of the layer. To fix this limitation, in GATv2 [14] they propose to simply apply the α^T layer after the nonlinearity (LeakyReLU), and the W layer after the concatenation, effectively applying an MLP to compute the score for each query-key pair. The new scores for the nodes i and j are now defined as:

$$e_{ij} = \alpha^T(LeakyReLU([Wh_i^l||Wh_j^l]))$$

which are then passed through a softmax layer to produce the normalized attention weights α_{ij}. The key difference here is that α^T is now outside the non-linearity effectively computing *dynamic attention*. This is much more expressive than the GAT layer with the same number of parameters.



(a) Attention in standard GAT (Veličković et al. (2018)) (b) Attention in GATv2, our fixed version of GAT

Figure 6.3.2: In a complete bipartite graph we can observe that the standard GAT computes static attention – the ranking of attention coefficients is global for all nodes in the graph, and is unconditioned on the query node. Specifically in this case, all query nodes attend mostly to the 8th key (k8). In contrast, GATv2 actually computes dynamic attention, where every query has a different ranking of attention coefficients of the keys and every node attends the most with itself [14].

GraphSAGE [36] learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. The primary innovation of GraphSAGE lies in its ability to perform inductive learning, which means that it can generalize to unseen nodes during the training process. The model operates by sampling and aggregating information from a node's neighborhood, capturing the local graph structure. The key components of GraphSAGE can be summarized as follows:

- **Neighborhood Sampling:** GraphSAGE samples uniformly a fixed-size neighborhood around each node, capturing its immediate local graph context. This sampling strategy allows the model to handle large graphs efficiently.
- **Aggregation Strategy:** After sampling, GraphSAGE employs an aggregation function (e.g., mean, LSTM) to combine information from the sampled neighbors. This step helps capture the structural and relational characteristics of the node's local environment.
- **Learnable Aggregation Functions:** The aggregation functions in GraphSAGE are parameterized and learnable. This means that during training, the model adapts and refines the aggregation strategy based on the characteristics of the graph data.
- **Scalability:** GraphSAGE is scalable to large graphs and computationally efficient due to its neighborhood sampling approach. It allows for the representation learning of nodes in graphs with millions or even billions of edges.
- **Inductive Learning:** One of the significant advantages of GraphSAGE is its inductive learning capability. The model can generalize its learned representations to nodes that were not present during training, making it applicable to a wide range of real-world scenarios.

In the context of the MPNN framework, the equations describing the way **GraphSAGE** functions are the following:

- **Aggregation:**
$$h^k_{N(i)} = Aggregate_k(\{h^{k-1}_j : \forall j \in S_{N(i)}\})$$
where $S_{N(i)}$ is a random sample of the neighboorhood of node i.
- **Update:**
$$h^k_i = f(W^k \cdot [h^{k-1}_i \parallel h^k_{N(i)}])$$
where f is a non-linear activation function.

For the aggregation function, the authors of [36] proposed several variants, such as mean, max and LSTM aggregation. However, as the aggregation function should not depend in the order of the neighboors (e.g permutation invariant) the LSTM aggregation operation which is not inherently permutation invariant, should be treated carefully in order to achieve permutation invariance. One way to achieve this according to the authors, is to apply the LSTM operator to a random permutation of the node's neighbors, assimilating in this way an unordered set.

GraphSAGE is practically an extension of the GCN to inductive unsupervised learning. It is proposed that it should be trained using an unsupervised graph-based loss, but can also be trained in a supervised manner if needed.

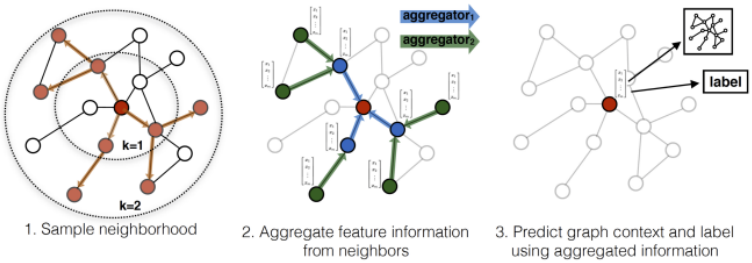


Figure 6.3.3: Visual illustration of the GraphSAGE sample and aggregate approach [36].

Graph Isomorphism Network (GIN) [89] attempted to solve the following problem: when do graph neural networks have the same discriminative capacity as the Weisfeiler-Lehman (WL) test? In particular, we want a GNN to map different graphs onto different embeddings if the WL test decides they are non-isomorphic. GIN [89] is the first spatial approach that addresses the inability of previous spatial models to discriminate between different graph structures. The authors firstly denoted that the embeddings of a set of neighboring nodes form a multiset. Then, the most powerful GNN should map two nodes to the same embedding only when their neighborhoods form the same multiset, in order for the mapping function to be injective and discriminative enough. Thus, they approached a GNN's aggregation function as a class of functions over multisets that a neural network can represent, and analyze whether they are able to construct injective multiset functions. For this reason, GIN uses the Multi-Layer Perceptron and the sum function as the aggregator. For each layer, node embeddings are summed and the result is concatenated. Thus, the expressiveness of the sum operator is combined with the memory of previous iterations by using concatenation. The authors proved that choosing the following update rule for node embeddings:

$$h_v^l = MLP^l((1+\epsilon^l)h_v^{l-1} + \sum_{u \in N(v)} h_u^{l-1})$$

results in the desired injective ability. In order to obtain graph level results, the authors propose to use node embeddings from all depths of the neural network in the following manner:

$$h_G = \|_{k=0}^{K} (READOUT(\{h_u^k | u \in G\} | k = 0, 1, ..., K)$$

where the $READOUT$ is choosed to be a simple summation of the node embeddings of each level of the network. In simple terms, in order to obtain an expressive enough graph representation, we sum the node embeddings of all levels independently, and finally we concatenate them to obtain the final graph embedding vector.

Graph AutoEncoder (GAE) and Variational Graph AutoEncoder (VGAE) are graph autoencoder architectures proposed in [45]. Graph autoencoders are used for unsupervised learning on graph data.

The simple GAE consists of an encoder and a decoder as in traditional autoencoders. The encoder consists of multiple GCN layers [44] with the goal to project the graph into a lower dimensional space. The encoder takes as input the feature and adjacency matrices and produces the graph's latent representation matrix Z by sequentially applying the logic behing the GCN layer. In general, in order to build the encoder, one can simply stack layers of any GNN encoder that produces node embeddings.

The decoder leverages the latent representation Z derived from the encoder to reconstruct the graph's adjacency matrix. This reconstruction process involves calculating the inner-product between the embeddings of two vertices and subsequently applying the sigmoid function. The sigmoid function aids in predicting the likelihood of a connection between these two vertices. The model undergoes training using the CrossEntropy loss, comparing the real adjacency matrix A with the reconstructed adjacency matrix $\hat{A}$. The equation governing this process is presented below, where z_v denotes the embedding of node v.

$$\hat{A}_{v,u} = dec(z_v, z_u) = \sigma(z_v^T z_u)$$

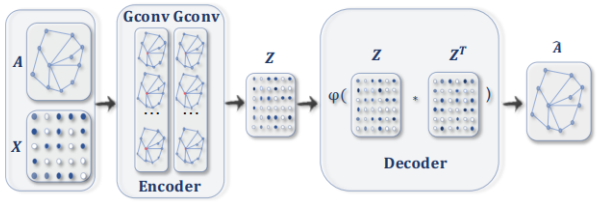


Figure 6.3.4: Visual illustration of how GAE works [88].

The VGAE, based on the variational autoencoder proposed by Kingma [43], is the variational version of GAE, which aims to learn the underlying distribution of graph data. By learning the underlying data distribution, the VGAE can be also used for generative tasks. In VGAE, an encoder transforms each node's features into a distribution in the latent space, allowing for the modeling of uncertainty. The model is trained to reconstruct the adjacency matrix of the graph while simultaneously regularizing the latent space using the Kullback-Leibler (KL) divergence. This incorporation of variational inference enables VGAE to capture the inherent uncertainty in the graph structure. The distribution of the latent variables $p(Z)$, known as the prior distribution, is almost always set to be the Standard Normal Distribution. Hence, the encoder maps the input data x to two vectors, the mean (μ) and the standard deviation (σ), which parameterize a multivariate Gaussian distribution in the latent space. Usually, two GNNs are used in a VGAE to produce these two vectors, using also an auxiliary method called the reparameterization trick. This method reparameterizes the sampling operation in a way that makes it differentiable. Instead of directly sampling from the latent distribution, the trick introduces a deterministic transformation involving a parameter-free noise term that is sampled from a fixed distribution (typically a standard normal distribution). This way, the sampling operation becomes differentiable, allowing for the backpropagation of gradients through the sampling process.

As we said, the loss of the VGAE includes the reconstruction of the adjacency matrix of the graph while simultaneously regularizing the latent space using the Kullback-Leibler (KL) divergence. The regularization is achieved through the Kullback-Leibler (KL) divergence term, which measures the difference between the learned distribution $q(Z|X, A)$ and the target Gaussian distribution $p(Z)$. The optimization objective that we seek to maximize, known as the Evidence Lower Bound (ELBO), is thus given by the following equation:

$$L = \mathbb{E}_{q(Z|X,A)}\left[log(p(A|Z))\right] - \mathrm{KL}(q(Z|X, A) \parallel p(Z))$$

where the first term represents the reconstruction loss and the second term measures the discrepancy between the prior and the approximate distribution.

6.4 How to Use Graph Neural Networks

As we have already described, the recent advancements in the field of Graph Machine Learning have given rise to a plethora of Graph Neural Network (GNN) models, each specializing in specific tasks or offering unique optimizations. Now that we have a clear picture of the most important variants, we can present the tasks for which they are adept as well as the different possible training settings.

6.4.1 Exploring Task Types in Graph Neural Networks

GNN variants are able to produce embeddings in different levels making them flexible enough to focus on various aspects of the graph structure. The different task types for which GNNs can be used, can be categorized into three main categories, namely Node-Level Tasks, Edge-Level Tasks and Graph-Level Tasks.

- **Node-Level Tasks:** Node-level tasks focus on understanding and processing information at the individual node level within a graph. In the context of GNNs, these tasks involve learning representations for each node based on its local neighborhood, something that can be achieved by utilizing the node representations computed by one of the already presented GNN models. This includes tasks such as node classification, where the goal is to predict the label or category of each node, and node regression, where the objective is to predict a continuous value associated with each node.
- **Edge-Level Tasks:** Edge-level tasks involve understanding and modeling relationships between pairs of nodes in a graph. By using GNNs, useful conclusions can be drawn concerning the edges of a given graph. Examples of edge-level tasks include link prediction, where the objective is to predict missing or potential connections between nodes, and edge classification, where the goal is to assign labels to edges based on their attributes or characteristics.
- **Graph-Level Tasks:** Graph-level tasks operate at the global level of the entire graph, considering its overall structure and properties. These tasks involve summarizing the information from all nodes and edges to make predictions or assessments about the entire graph. Graph-level tasks include graph

classification, where the goal is to assign a label to the entire graph, and graph regression, where the objective is to predict a continuous value associated with the entire graph. To accomplish this, readout operations, as we have already seen, are usually employed. The subject of this dissertation is focused in making predictions in the graph level.

6.4.2 Diverse Training Approaches for Graph Neural Networks

Moreover, as it is the case for traditional neural networks, the different GNN variants can also be categorized according to the type of signal they are trained with. The three fundamental types of training are namely Supervised Learning, Semi-Supervised Learning and Unsupervised Learning.

- **Supervised Learning:** In supervised learning scenarios, data is labeled, encompassing classification and regression tasks at the node, edge, or graph levels.
- **Semi-Supervised Learning:** Tasks in this category primarily operate on partially labeled graphs, where only some nodes have labels. Node classification, performed in a robust manner, is a common task in semi-supervised learning. Additionally, link prediction can be executed in a semi-supervised fashion.
- **Unsupervised Learning:** Tasks characterized by completely lack of labels fall into the realm of unsupervised learning. Examples include node clustering as well as learning meaningful and expressive node and graph embeddings. Node and Graph embeddings, in particular, can be learned in a purely unsupervised manner using an end-to-end encoder-decoder neural network such as VGAE [45] or through contrastive learning techniques. Notably, in this dissertation we will use different variants that belong to the contrastive learning family. Contrastive learning can be considered a form of self-supervised learning. In self-supervised learning, the model is trained on a task where the labels are generated from the input data itself, without requiring external annotations.

In summary, comprehending the versatility of GNN variants in terms of both task types and training approaches is essential for their effective utilization in diverse applications within the realm of geometric deep learning. In the next section we will present in a detailed manner the basic concepts of contrastive learning in graphs that we will use in this dissertation.

6.5 Contrastive Learning on Graphs

As we have already discussed in chapter 4 Contrastive Learning has emerged as a promising paradigm for enhancing the representation learning process in machine learning tasks. Originally applied to image and text data, contrastive learning aims to learn robust representations by contrasting positive samples with negative samples. This methodology has shown remarkable success in capturing intricate patterns and semantics in data. Extending the principles of contrastive learning to graph-structured data has become an active area of research. The unique challenges posed by graphs demand tailored approaches. In this section, we delve into the motivation behind employing contrastive learning techniques specifically for graph data, presenting the most important ones that were used in the context of this dissertation.

6.5.1 Contrasting Modes

A key element for contrastive learning is to determine the contrastive mode [96] that will be employed. The contrastive mode is used to define the sets of positive and negative samples, which are essential in order to proceed to the calculation of the contrastive objective that will be optimized. In the context of graphs, positive and negative sets are established at various levels of detail within the graph through different contrasting modes. In common practice, three widely utilized contrasting modes are local-local CL and global-global CL, which contrasts embeddings at identical scales, and global-local CL, which contrasts embeddings across different scales [96]. Specifically, in same scale contrastive modes, embeddings of the two augmented graph views corresponding to the same node or graph form the positive set, where all the others in the batch are naturally selected as negatives. In cross-scale (global-local) contrastive mode, for every global graph embedding s, the positive samples are considered all node embeddings v_i within the graph. For node

datasets, only local-local and global-local CL are suitable, while all three modes can be employed for graph datasets.

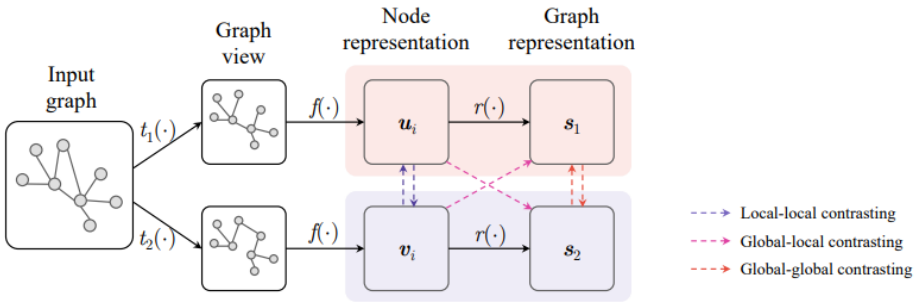


Figure 6.5.1: A general GCL framework which, after producing two augmented graph views, it encodes them using GNN layers. The three different contrastive modes including node-node, graph-graph and patch(node)-graph interactions, are illustrated using arrows of different colours [96].

6.5.2 Graph Contrastive Learning with Augmentations (GraphCL)

GraphCL [90] was one of the first contrastive learning approaches with augmentations for GNN pre-training that was developed. One key element of constrastive learning are data augmentations which were under-explored for graph-data. For this purpose, in GraphCL [90] four types of graph data augmentations were first designed. These data augmentation techniques are utilized to produce correlated views (positive samples) which are forced to be closer in the embedding space. GraphCL belongs to a class of models that employ augmentation schemes that generate congruent pairs to model the joint distribution of positive pairs. The proposed model is proved to perform mutual information maximization as we have examined it in chapter 4.

The data augmentation techniques, including both topology and feature augmentations, proposed in order to define positive and negative samples are divided into the following categories:

- **Node dropping:** In the context of a graph G, node dropping involves the random removal of a specific fraction of vertices and their associated connections. The fundamental assumption is that the absence of these vertices does not alter the semantic content of G. The probability of dropping each node adheres to a default independent and identically distributed (i.i.d.) uniform distribution, or any other specified distribution.
- **Edge perturbation:** Introduces disruption to the connections within G by randomly adding or removing a specific proportion of edges. This suggests that the semantic interpretation of G exhibits a certain robustness to variations in the pattern of edge connectivity. Additionally, an independent and identically distributed (i.i.d.) uniform distribution is used for the random addition or removal of each edge.
- **Attribute Masking:** Attribute masking forces models to recover masked vertex attributes using the remaining attributes. The underlying assumption is that missing partial vertex attributes should have a minimal impact on the model predictions.
- **Subgraph:** This method of data augmentation samples a subgraph from G using a random walk. The underlying assumption is that the semantics of G can be largely retained within its partial local structure.

After the basic data augmentation techniques for graph-data are defined, the training procedure resembles SimCLR [20]. Firstly, the provided graph G undergoes graph data augmentations to generate two correlated views $\hat{G}_i, \hat{G}_j$ as a positive pair. The strategic selection of data augmentations is crucial for different graph dataset domains. Secondly, a GNN-based encoder $f(\cdot)$, which could be one of GCN, GIN, GAT among others, extracts the graph-level embeddings h_i, h_j for the two augmented views of the original graph. Moreover, a non-linear transformation $g(\cdot)$, named projection head, maps the augmented graph-level representations to

another latent space where the contrastive loss will be calculated. From this, it is evident that GraphCL falls into the category of global-global CL contrastive mode. For the projection head, the authors propose a two-layer perceptron (MLP), which is applied to obtain z_i, z_j.

The final step involves establishing a contrastive loss function, denoted as $L(\cdot)$, to enforce maximizing the consistency between positive pairs z_i, z_j when compared to negative pairs. The authors propose the adoption of the normalized temperature-scaled cross-entropy loss (NT-Xent) [62], [20] for this purpose, which is practically the same as the InfoNCE loss and which we have presented in detail in chapter 4. For minibatch training, a random minibatch containing N graphs is sampled, generating $2N$ augmented graphs. The notation is updated to $z_{n,i}, z_{n,j}$ to represent the two augmented views of the nth graph in the minibatch. Negative pairs are implicitly derived from the other graphs within the same minibatch, following the approach in [20]. Introducing the cosine similarity function as $\text{sim}(z_{n,i}, z_{n,j}) = \frac{z_{n,i}^T z_{n,j}}{|z_{n,i}| \cdot |z_{n,j}|}$, the NT-Xent for the nth graph is formally defined as:

$$l_n = -\log \left(\frac{\exp(\text{sim}(z_{n,i}, z_{n,j})/\tau)}{\sum_{n'=1, n' \neq n}^{N} \exp(\text{sim}(z_{n,i}, z_{n',j})/\tau)} \right)$$

The total loss for each batch is then computed by taking the average of the terms l_n for all positive pairs in the batch expressed by the following equation:

$$L = \frac{1}{N} \sum_{k=1}^{N} l_n$$

The loss L can be rewritted for a batch of graphs as:

$$\begin{aligned}
L &= -\frac{1}{N} \sum_{n=1}^{N} \log \frac{\exp\left(\text{sim}(z_{n,i}, z_{n,j})/\tau\right)}{\sum_{n'=1, n' \neq n}^{N} \exp(\text{sim}(z_{n,i}, z_{n',j})/\tau))} \\
&= -\frac{1}{N} \sum_{n=1}^{N} \left[\frac{\text{sim}(z_{n,i}, z_{n,j})}{\tau} - \log \left(\sum_{n'=1, n' \neq n}^{N} \exp(\text{sim}(z_{n,i}, z_{n',j})/\tau) \right) \right] \\
&= -\frac{1}{N} \sum_{n=1}^{N} \text{sim}\left(g(f(\hat{G}_{n,i})), g(f(\hat{G}_{n,j})) \right) / \tau + \frac{1}{N} \sum_{n=1}^{N} \log \left(\sum_{n'=1, n' \neq n}^{N} \exp\left(\text{sim}\left(g(f(\hat{G}_{n,i})), g(f(\hat{G}_{n,j})) \right) / \tau \right) \right) - \log N \\
&= -\mathbb{E}_{P(\hat{G}_i, \hat{G}_j)} T(f(\hat{G}_i), f(\hat{G}_j)) + \mathbb{E}_{P(\hat{G}_i)} \log \left(\mathbb{E}_{P(\hat{G}_j)} e^{T(f(\hat{G}_i), f(\hat{G}_j))} \right) - \log N
\end{aligned}$$

where $P(\hat{G}_i, \hat{G}_j)$, $P(\hat{G}_i)$, and $P(\hat{G}_j)$ are respectively the joint and the marginal distributions of the augmented graphs. The function $T(\cdot, \cdot)$ is a learnable discriminator function parametrized by the similarity function $\text{sim}(\cdot, \cdot)$ (usually the cosine similarity), temperature factor τ, and the projection head $g(\cdot)$.

Now we can see that, as in [62], [20], this form of loss fits the formulation of the InfoNCE loss (which is the negative InfoNCE estimator that we have presented in chapter 4), such that minimizing it, is equivalent to maximizing a lower bound of the mutual information between the latent representations of two views of graphs. Consequently, minimizing this loss with respect to the neural network parameters, forces the model to encode more similar graphs closer in the embedding space and push farther apart dissimilar ones. Moreover, the authors argue that the projection head has a crucial role in this framework helping reaching a much tighter lower bound compared with dropping the projection head.

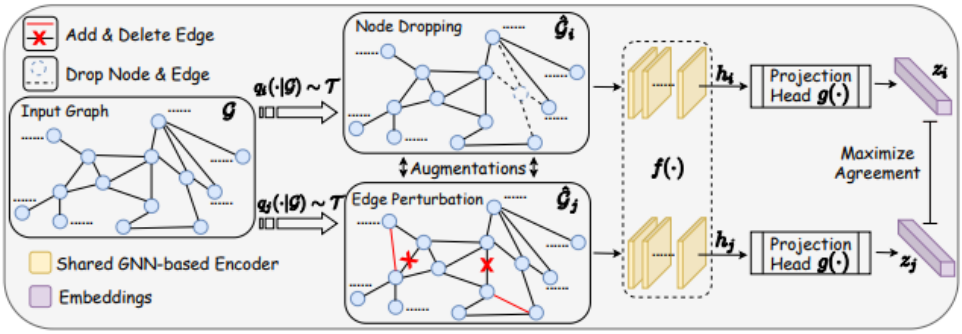


Figure 6.5.2: The proposed framework for Graph Contrastive Learning where $q_i(|G)$ and $q_j(|G)$ are augmentations sampled from an augmentation pool T and applied to input graph G. A shared GNN-based encoder f(·) and a projection head g(·) are trained to maximize the agreement between representations z_i and z_j via a contrastive loss [90].

6.5.3 InfoGraph

InfoGraph was proposed in [77] and in contrast with GraphCL which employs a global-global contrastive mode, it uses local-global interactions in order to obtain graph level embeddings. In order to derive graph level representations, the authors propose to maximize the mutual information between graph-level and patch-level representations following the paradigm of Deep InfoMax [39]. This enables the produced graph representations to capture shared aspects across all substructures (patches) which are present in the data.

In order to train the model in a mini-batch fashion, we can consider a mini-batch of training graph samples $G = \{G_j \in G\}_{j=1}^{N}$. No graph augmentation techniques are required for training. Passing the graphs through a K-layer graph neural network with parameters ϕ yields node embeddings. After the initial k layers of the graph neural network, the input graph is encoded into a set of patch representations $\{h_i^{(k)}\}_{i=1}^{N}$. Subsequently, feature vectors from all depths are aggregated into a single feature vector that encompasses patch information at various scales centered at each node as following:

$$h_i^{\phi} = \text{CONCAT}(\{h_i^{(k)}\}_{k=1}^{K})$$

$$H^{\phi}(G) = \text{READOUT}(\{h_i^{\phi}\}_{i=1}^{N})$$

where, h_i^{ϕ} denotes the summarized patch representation centered at node i produced by jumping concatenation, and $H^{\phi}(G)$ is the global graph representation vector after applying the READOUT pooling function to the concatenated node embeddings.

Moreover, instead of using the InfoNCE MI estimator, the authors propose the Jensen-Shannon MI estimator in order to maximize MI on global/local pairs. For a mini-batch of N graphs the goal is to maximize:

$$\phi, \hat{\psi} = \arg\max_{\phi,\psi} \sum_{G \in G} \frac{1}{|G|} \sum_{u \in G} I_{\phi,\psi}(h_u^{\phi}; H^{\phi}(G))$$

where $I_{\phi,\psi}$ is the mutual information estimator modeled by discriminator T_{ψ} and parameterized by a neural network with parameters ψ. The Jensen-Shannon MI estimator is defined as we have seen in chapter 4 as:

$$I_{\phi,\psi}(h_i^{\phi}(G); H^{\phi}(G)) = \mathbb{E}_P[-\text{sp}(-T_{\phi,\psi}(h_i^{\phi}(x), H^{\phi}(x)))] - \mathbb{E}_{P \times P'}[\text{sp}(T_{\phi,\psi}(h_i^{\phi}(x'), H^{\phi}(x)))]$$

By denoting as P the set of nodes belonging to a graph G (positive samples) and as Q the set of all other nodes (negative samples), using cosine similarity (d) as the discriminator, we can compute the JS estimator for each graph G easily using the formula:

$$I\left(H^{\phi}(G)\right) = -\frac{1}{P}\sum_{h_i^{\phi}\in P(G)} \text{sp}(-d(h_i^{\phi}, H^{\phi}(G))) - \frac{1}{Q}\sum_{z_i^{\phi}\in Q(G)} \text{sp}(d(z_i^{\phi}, H^{\phi}(G)))$$

Regarding the GNN encoder, the authors opt for GIN [89] since it imparts a more effective inductive bias for graph-level applications. It is also crucial to carefully design graph neural networks to ensure discriminative graph representations across various instances. For this purpose, the authors advocate for employing a summation approach over a mean-based one for the READOUT function, asserting that it can yield significant insights into the graph's size.

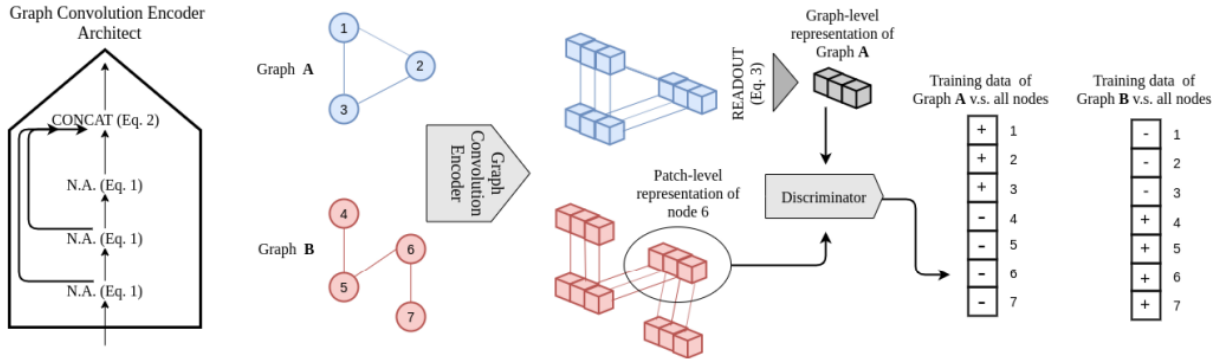


Figure 6.5.3: Two graphs are encoded into their corresponding feature maps by graph convolutions and jumping concatenation. The discriminator takes a (global representation, patch representation) pair as input and decides whether they are from the same graph. InfoGraph uses a batch-wise fashion to generate all possible positive and negative samples in the batch. [77].

6.5.4 Deep Graph InfoMax (DGI)

Deep Graph InfoMax [84] also inspired by Deep InfoMax [39] is similar to InfoGraph. However, it focuses on learning unsupervised node and not graph embeddings via MI maximization, mostly for one graph and not across a batch of different graphs. For this purpose, in order to obtain negative samples the authors, propose to corrupt the original graph. This approach also falls into the category of global-local CL [96] as it constructs global-local pairs in the following manner:

- **Corrupt** G to a new graph: $H = C(G)$
- **Encode** each node of both of these graphs in order to obtain patch (node) representations
- **Summarize** the true graph into a summary vector s by applying a readout function to the patch representations of the true graph
- **Score** the encoded embedding vectors of both G and H, using the discriminator and G's summary vector s: $D(v, s)$ for v in G and H
- **Collate** all the scores in a loss function that tries to maximize $D(v, s)$ if v is the embedding vector of a true node and minimize it if v is the embedding vector of a corrupted node.

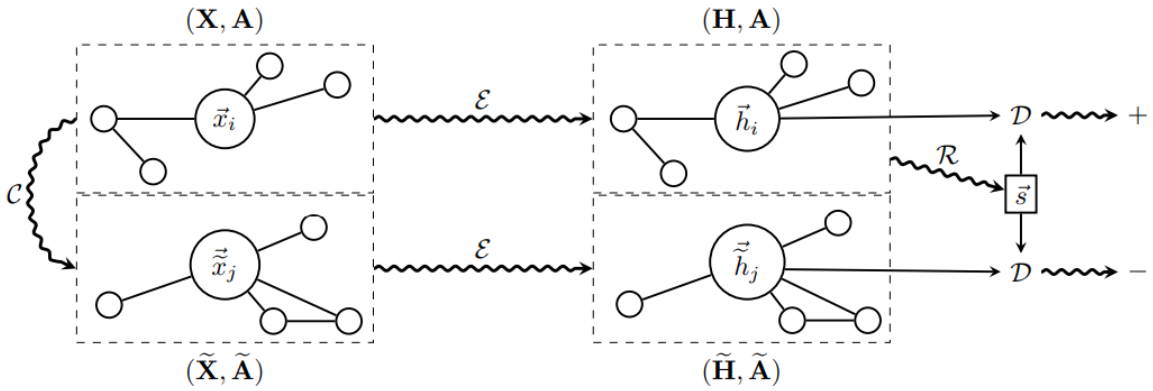


Figure 6.5.4: An illustration of the most important steps of DGI [84].

For the encoder the authors choose the GCN [44] and for the READOUT function they use the mean of all the node embeddings. For a simple corruption function, they propose to randomly shuffle the initial feature matrix X of the original graph. More sophisticated approaches include corrupting also the adjacency matrix, among others. For the loss function, they employ the Jensen-Shannon MI estimator we have already seen, forcing the discriminator to give scores close to 1 for the true nodes, and scores close to 0 for the corrupted nodes when they are compared with the summary vector s.

6.5.5 Deep Graph Contrastive Representation learning (Grace)

Grace, as introduced in [95], presented an innovative approach to unsupervised graph representation learning. This involves employing a contrastive objective at the node level, where two graph views are created by performing various augmentations to the original graph data. The learning process focuses on maximizing the agreement of node representations within these two views. In contrast to comparing node-level embeddings with global ones as in InfoGraph and DGI and in contrast to GraphCL which compares graph level embeddings, the emphasis in GRACE is on contrasting embeddings specifically at the node level. Thus, the contrasting mode employed, lies in the category of local-local CL [96]. The methodology involves generating two augmented graph views through randomly corrupting the topology and the node attributes of the original graph. Afterwards, the model is trained using the normalized temperature-scaled cross-entropy loss (InfoNCE / NT-Xent) to enhance the agreement among corresponding node embeddings in these views. The augmentation techniques used include removing edges and masking features but no node dropping is performed in order to keep the alignment between the same nodes in the two views. The loss used is exactly the same as in the case of GraphCL except that instead of passing graph embeddings inside the loss we pass node level embeddings. If we consider a mini-batch of graphs, for each node only one positive sample exists. Negative samples are not calculated explicitly, instead, given a positive pair, negative samples are considered all other nodes in the two views. This results in the following expression:

$$l_{(u_i,v_i)} = \log \frac{e^{\frac{sim(u_i,v_i)}{\tau}}}{e^{\frac{sim(u_i,v_i)}{\tau}} + \sum_{k=1,k\neq i}^{N} e^{\frac{sim(u_i,v_k)}{\tau}} + \sum_{k=1,k\neq i}^{N} e^{\frac{sim(u_i,u_k)}{\tau}}}$$

where in order to compute the loss for a positive pair of nodes, we need in the denominator to include both inter-view negative nodes and intra-view negative nodes. Thus, the two summations in the denominator represent these two quantities. In order to obtain the total loss in a batch, we simply take the average of these losses over all positive pairs.

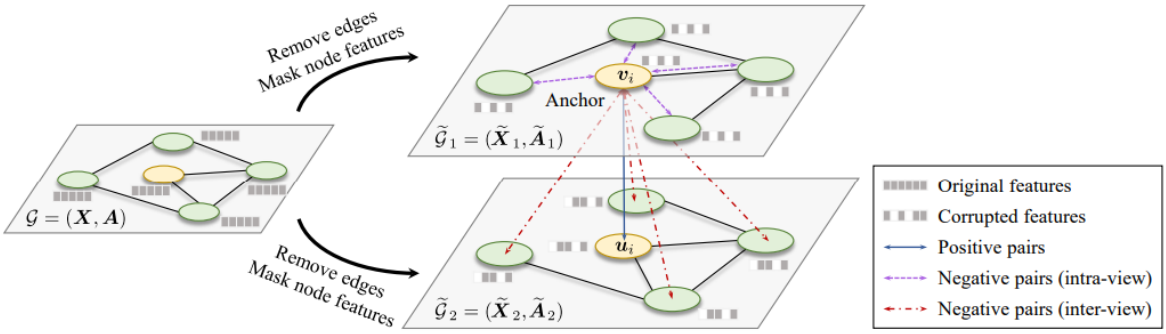


Figure 6.5.5: An illustration of the local-local CL conducted by GRACE [95].

Chapter 7

Counterfactual Explanations

As AI systems become more prevalent in various aspects of our lives, it is crucial for users to trust and understand the decisions made by these systems. Explainable AI (XAI) [3] refers to the set of techniques and methodologies designed to make the decision-making processes of artificial intelligence (AI) systems more transparent and understandable to humans . The need for XAI arises as AI models, particularly those based on deep learning and complex algorithms, often function as "black boxes," making it challenging to interpret their outputs and understand the underlying reasoning behind their decisions. XAI helps build trust and promotes social acceptance of AI systems, especially in critical applications such as healthcare, finance and autonomous vehicles. This happens because users are more likely to adopt and accept AI systems if they can comprehend and trust the decisions made by them, especially in applications where human lives or sensitive information are involved. Moreover, XAI can provide valuable insights into how models work, enabling experts to debug and improve their models more effectively, identifying and addressing biases and discrimination that may also arise from AI models.

In the context of this thesis, we will focus on a specific explainability technique known as Counterfactual Explanations. In this chapter we will introduce the basic concepts behind this type of explanations and explain their details thoroughly based on [31] and [23].

Contents

7.1 Introduction **96**
7.2 Conceptual Edits and Algorithmic Framework **96**
7.3 GNNs for Semantic Counterfactuals **99**
7.4 Related Work **100**

7.1 Introduction

Counterfactual Explanations are a concept commonly used in the field of explainable artificial intelligence (XAI) to help users understand the reasoning behind a classifier's predictions. The basic idea is to provide an explanation by considering alternative scenarios. Specifically, these alternative scenarios relate to questions such as "What has to change for this to be classified as X instead of Y?" or "What would need to change in order for the model to make a different decision". In this way, they can provide valuable insights for the decision-making process of classifiers. The goal is to explain why a particular prediction was made by exploring what changes in the input features would lead to a different outcome. For example, in a credit scoring model, a counterfactual explanation might reveal the changes in financial variables that could lead to a more favorable credit decision.

Generally, good Counterfactual Explanations could posses several attributes which we will shortly explain below:

- **Model Agnostic:** There are both model agnostic and model specific counterfactual explanation methods. The fact that counterfactual explanations can be generated for a wide range of AI models, including model agnostic techniques that only work with the model inputs and outputs and not the internal structure of specific models, boosts them with increased flexibility. This is in contrast to other explainability methods, such as decision trees which are effective for capturing non-linear relationships in data, but they might struggle with very complex, high-dimensional spaces. Counterfactual explanations can still be effective in such cases, dealing with complex black-box models like deep neural networks
- **Actionability:** Counterfactual explanations aim to explain a model's decision by presenting a scenario in which a different set of inputs would have led to a different outcome. Actionability becomes relevant when considering whether the proposed changes in the counterfactual scenario are realistic and implementable. An actionable counterfactual explanation is one that provides insights that can be translated into actionable steps or decisions to achieve a desired outcome. In other words, it helps users or stakeholders understand not only why a particular decision was made but also what changes can be made to influence the decision in the future. For example, in the context of AI decision-making, if a model denies a loan application, a counterfactual explanation may suggest changes to the applicant's features (such as income or credit score) that would have resulted in an approval. Actionability in this context would involve ensuring that the suggested changes are practical and feasible for the applicant to implement, such as improving their credit score over time.
- **User Friendly:** Counterfactual explanations are often designed to be more user-friendly and intuitive, because they are contrastive to the current instance and because they are selective, meaning they usually focus on a small number of feature changes. This makes them more suitable for applications where end-users with varying levels of technical expertise need to understand and trust the model predictions
- **Fairness:** Counterfactual explanations can be used to explore and address biases in model predictions. By generating counterfactual instances that lead to different outcomes, it becomes possible to identify and mitigate biases in the model ensuring its robustness
- **Instance Based Explanations:** Counterfactual explanations are generation for every input sample. Instance-based explanations focus on a specific instance or prediction, providing a highly tailored explanation for that particular case. This can be more precise and relevant to the user's immediate concerns

A good counterfactual explanation should be able to produce the predefined prediction as closely as possible. However, there are usually multiple different counterfactual explanations for the same instance which explain it equally well. Usually, the goal is to provide a minimal explanation which is the closest one in terms of some distance metric, such as set edit or graph edit distance which we will use in this thesis.

7.2 Conceptual Edits and Algorithmic Framework

In this section, we will introduce the Counterfactual Explanation Framework presented in [31] and we will show how it can be integrated in the context of scene graph retrieval using Contrastive GNNs. In particular,

in [31] the authors propose a theoretical framework in order to compute the Optimal counterfactual path for a specific input, in the context of black box classifiers. The method is based on the notion of conceptual edits which are more interpretable and work on the semantic space instead of working in the pixel level. In order to present the exact definition of conceptual edits we should first introduce some notation.

The approach makes use of the notation from Description Logics [80] in order to describe concepts and their relationships. Concepts are the general representations of objects present in the input data. The Black Box Classifier is formally denoted as $F : D \rightarrow [0,1]^c$ where c is the number of distinct classes and D is the classifier's input domain that we want to explain. Concerning the classifier, we only need to be able to observe its output given a specific input.

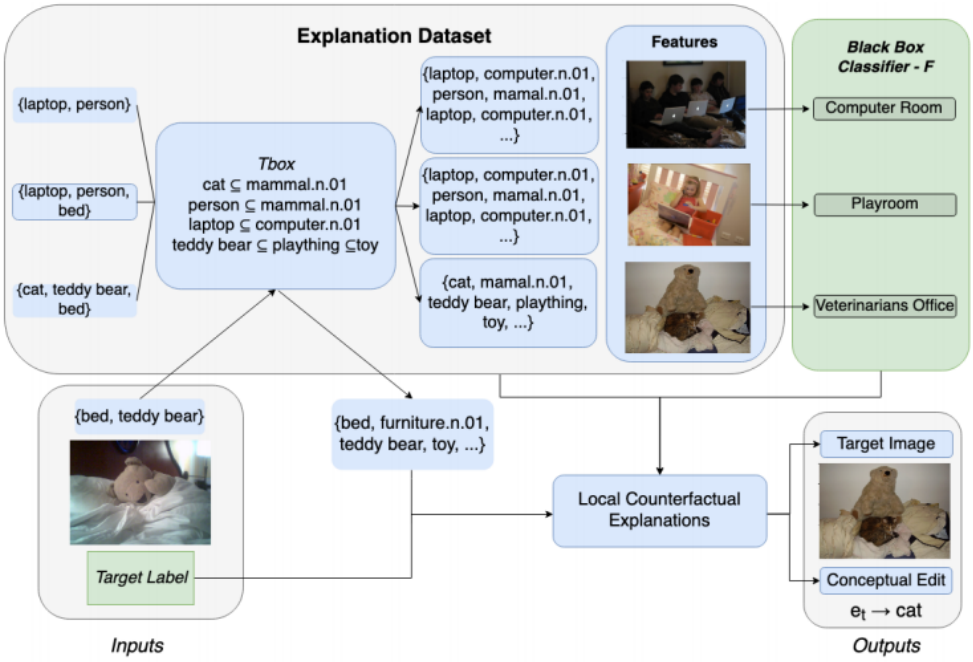


Figure 7.2.1: An illustration of the Conceptual Edits as Counterfactual Explanations Framework [31].

An important concept used in this framework is the Explanation Dataset. Given the domain D of the classifier (which can be for example images from the COCO dataset [52]) and a set of atomic concepts CN, the explanation dataset is defined as:

$$(x_i, C_i), \text{ which is a set of tuples where } x_i \in D \quad \text{and } C_i \subseteq CN$$

Another crucial detail is that, the atomic concepts CN, should belong to an external knowledge, such as WordNet [59]. In this way, it is ensured that there is a TBox for the atomic concepts providing us with a hierarchy between them.

A TBox is a set of terminological axioms of the form $A \subseteq B$, where $A, B \subseteq CN$, as they are defined in the WordNet [59] noun hierarchy. A more useful representation of the TBox is if we consider it as a knowledge graph. This graph is a directed graph $G(V, E)$ in which every node represents a different atomic concept or the universal concept $\top$ and the edges have a direction from A to B. If we ignore the direction of the edges we have the undirected TBox graph. Moreover, weights can be assigned to the edges of the graph and if any concept isn't included in any other concept, it is included in the universal concept T.

Now, we can define another important term used in the framework, called Concept Distance. The Concept Distance $d_T(A, B)$ between two concepts A and B, is calculated in the undirected TBox graph and is defined as the length of the shortest path between the two vertices corresponding to the concepts A and B. After we have determined the Concept Distance we can now introduce the notion of Conceptual Edits.

A Conceptual Edit transforms a set of concepts A via one of the following three ways:

- Replacement of concept $A \in A$ with concept $B \notin A$ denoted as $e_{A \to B}(A)$
- Deletion of concept $A \in A$ denoted as $e_{A \to T}(A)$
- Insertion of concept $B \notin A$ into A denoted as $e_{T \to B}(A)$

Each of the aforementioned Conceptual Edits $\epsilon_{x \to y}$ where $x, y \in CN$ is linked to a cost following the edit operation. This cost is equal to the concept distance $d_T(x, y)$ between the concepts x, y in the corresponding TBox graph.

By leveraging the definition of a concept set edit, we can now establish a concept set edit distance for sets of concepts. This distance is determined as the lowest cost produced by a series of concept set edits needed to transform the initial set of concepts into the second set. Intuitively, the concept set edit distance measures how conceptually similar two elements of an explanation dataset are and plays a crucial role in generating counterfactual explanations.

However, as we have already discussed earlier in the previous section, at the core of a good counterfactual explanation lies the fact that we want to find small conceptual alterations which lead to significant changes in the classifier's decision. In order to take this fact into account, we define, for two elements $a = (x_a, C_a)$ and $b = (x_b, C_b)$ of the explanation dataset, a measure called Significance of Transformation. Given the classifier F, the TBox graph and the concept set edit distance for the set of concepts C_a and C_b denoted as $D_T(C_a, C_b)$, the Significance of Transformation is defined as:

$$\sigma(a, b) = \frac{|F(x_a) - F(x_b)|}{D_T(C_a, C_b)}$$

It is evident that the quantity $\sigma(a, b)$ and as a result, the significance of the transformation, is high, when the set edit distance between the concepts is small and the variation in the classifier's output is high. Now, an intermediate weighted graph can be computed, where there is a vertex for every sample of the explanation dataset and each pair of nodes (a, b) is connected with a directed edge of weight $\frac{1}{\sigma(a,b)}$.

Finally, this intermediate weighted graph can be used in order to generate Local and Generalized Counterfactual Explanations. These two types of counterfactual explanations are defined below:

- **Local Counterfactual Explanations**: This type of counterfactual explanations, assumes an explanation dataset $D = \{x_i, C_i\}$, a classifier F and the intermediate graph G(V,E) where $V = D$ and E contains edges of weight $\frac{1}{\sigma(a,b)}$ for every pair of $a, b \in D$. A local counterfactual explanation from an element a of the explanation dataset to an element b of a different target class H, such that $F(b) = H$ and $F(a) \neq H$, is every path between nodes a and b. However, we want to find optimal counterfactual explanations and therefore we keep only shortest paths on the constructed graph that correspond to a set of conceptual edits that transform the concept a to a different concept belonging to the target class H in a minimal way
- **Generalized Counterfactual Explanations**: This type of counterfactual explanations involves the computation of counterfactuals for only a subset of instances of the explanation dataset, which are usually samples classified to a specific class by the classifier. Given also a target class H, Generalized Counterfactual Explanations is a measure of importance for every concept C. Intuitively, this importance measure, is a quantity that calculates how often a concept is deleted or inserted in order to lead to a transformation leading to the target class H, starting from the instances of the selected region of the explanation dataset that are classified to a different class. Introducing some mathematical formalism, lets denote a region of the explanation dataset D as R_Q. Given that E_{R_Q} is the multiset containing the conceptual set edit transformations, generated by the optimal local counterfactual explanations in order to transform every element of R_Q to the target class H, the importance of every concept $C \in CN$ is calculated according to the following formula:

$$\frac{|\{e_{x \to C} \in E_{R_Q}\}| - |\{e_{C \to x} \in E_{R_Q}\}|}{|R_Q|}$$

where $x \in CN$.

In order to determine the overall complexity of the proposed framework, we should carefully look at the calculations involved at every step of the general algorithm. Let's analyze the process and break it down into its constituent algorithms.

At the first stage of the algorithm, the Concept Distance between all pairs of the present concepts on the undirected TBox graph should be calculated. This is achieved using Dijkstra's algorithm on the udirected TBox graph where each vertex corresponds to one concept and every edge corresponds to one subsumption relationship between two concepts. Dijkstra's algorithm can effectively compute the shortest path between every two concepts.

Subsequently, at the next stage of the process, the Concept Set Edit Distance from one set of concepts A to another set of concepts B should be calculated. This quantity measures the minimal conceptual edits in order to turn the set of concepts A into the set of concepts B. This is achieved by constructing a bipartite graph where each concept belonging to A is connected to every concept belonging to B, after the deletion of common concepts of the two sets. Each concept is represented by a node in the bipartite graph and the edges of this graph are assigned with weights. The values of these weights are equal to the concept distance between the two concepts that the edge connects and they were calculated using Dijkstra's algorithm in the first step. Then, the Concept Set Edit Distance is computed by calculating the minimum weight full matching of the bipartite graph by using Karp's algorithm [42] which has a time complexity of $O(|A||B|log|B|)$.

Finally, in order to compute Local Counterfactual Explanations, a graph, with vertices all the elements of the explanation dataset and edges with weights $\frac{1}{\sigma(a,b)} = \frac{D_T(C_a,C_b)}{|F(x_a)-F(x_b)|}$ for all pairs of elements of the explanation dataset, is constructed. The quantity $D_T(C_a, C_b)$ is the Concept Set Distance calculated in the second step using Karp's algorithm. This algorithm also provides us with the minimal edits needed in order to compute this set distance which are then placed as labels to the edges of the constructed graph. Now, by using again Dijkstra's algorithm in order to compute shortest paths in the constructed graph between two elements of the explanation dataset, the Optimal Local Counterfactual Explanations are generated.

7.3 GNNs for Semantic Counterfactuals

However, the aforementioned framework proposed in [31], suffers from certain limitations. While the definition of concepts as objects or relations semantically enriches the proposed approach, by leveraging Set Edit Distance to measure the discrepancy between two samples (e.g images) of the dataset, crucial information regarding the way the objects are related, is sacrificed. Following the approach presented in [24] we leverage Graph Neural Networks in order to take full advantage of graph structured data. Specifically, by representing the input data as semantic graphs, something that, in the case of images, means that we work with the corresponding scene graphs, the relations between the sets of concepts (e.g the edges between nodes in scene graphs) are taken into consideration and therefore a richer representation is used in order to obtain conceptual counterfactual explanations.

In this case, instead of calculating the Set Edit Distance, it is crucial to be able to calculate the Graph Edit Distance (GED) between the semantic graphs of the dataset. By calculating GED, our model will be later capable to retrieve the most similar graphs belonging to another, target class, and therefore generate a semantic counterfactual explanation for a vision classifier's decision. The set of edits of this counterfactual explanation will be consisted of the vertex/edge insertion, deletion and substitution operations as calculated from GED. However, these graph edits cannot be exactly calculated as GED relates to the NP-hard graph similarity problem. In order to bypass solving this computational expensive problem, or using a still computational expensive GED approximation algorithm, we leverage GNNs in order to obtain scene graph representations on the embedding space. By reassuring that our GNN model embeds scene graphs with respect to their graph proximity relationship, we can retrieve, for a given query graph, the most similar data points of the explanation dataset belonging to a target class and then compute GED only between the query graph and the most similar retrieved one, thus yielding the necessary graph edits to transfer to the target class. As a result, the computational cost needed to obtain the counterfactual explanations is significantly reduced as the GED is computed only between the query graph and the most similar retrieved ones and not between all possible candidates.

The specific architectures that we will explore for the GNN encoder, belong to the family of Contrastive GNNs. In this way, the training procedure of the neural network can be completely unsupervised (or weakly supervised) and no need of expensive pre-computation of GED is necessary. Moreover, integrating Contrastive GNNs to this semantic counterfactual explanation framework, completes the line of work proposed in [24], where Graph Autoencoders and Supervised GNN encoders are extensively studied.

7.4 Related Work

Counterfactual explanations have attracted the attention of a significant proportion of researchers. As a result, many works have been published focused in this field in recent literature. The vast majority of these works focuses on visual classifiers. These approaches leverage feature level attributes in order to find, highlight and modify impactful areas of an image that lead to a different classification result. Approaches proposed from Goyal et al. [35] and Vandenhende et al. [81] are based on pixel-level edit methods, focusing on marking and altering significant image areas to influence the model's predictions. However, what stands out in [81] is the usage of an auxiliary model for semantic similarity prediction between local regions, in an attempt to enforce semantically consistent area exchanges.

Moreover, several works includes external knowledge from ontological knowledge, such as knowledge graphs, semantically enriching the model. This is illustrated in works such as [76], where the WordNet hierarchy is used, and in [1] where an ontology-based approach is used to assist a classifier for a semantic segmentation task by providing explanations for misclassification errors.

Finally, the majority of works combining Graph Neural Networks and Counterfactual Explanations, are focused on explaining GNNs themselves and not providing Counterfactual Explanations using GNNs. More details can be found in Lucic et al. [55] and Bajaj et al. [6].

Chapter 8

Proposal

In this chapter, we propose the Contrastive Graph Neural Network model, that will be used to tackle the problem of Scene Graph Similarity in the context of the Information Retrieval task on the Scene Graph dataset, Visual Genome. In particular, all the experimentation with the original contrastive models as well as the proposed rank aware fine tuning will be presented and explained thoroughly. In the following sections, we first highlight the main contributions of this thesis and after that present the proposed models in detail.

8.1 Contributions

Below, the key contributions of this thesis are summarized:

- We employ various Contrastive Graph Neural Network approaches to investigate Scene Graph Similarity as our chosen task. To the best of our knowledge, there has been limited academic focus on utilizing Graph Contrastive Learning to address this issue. Therefore, our objective is to provide a comprehensive overview of the problem, the methodologies we employ for its resolution, and to offer an additional perspective through the use of contrastive methods, complementing the existing approaches in [24] and [25].

- By employing Contrastive Approaches, we can train the models in an unsupervised manner, eliminating the necessity to calculate Graph Edit Distance between graph samples. Moreover, these architectures eliminate the need for pairs of graphs during training, thereby decreasing the sample order from O(n^2) in supervised models to O(n). These two essential factors are the main drivers behind the expedited training of the proposed models.

- We thoroughly study the performance of various Graph Convolutional variants as building blocks in the employed Contrastive approaches for the scene graph retrieval task. We observe that Graph Contrastive pretraining, whether on dense or sparse random graphs, represents equally competitive approaches. Additionally, we propose Rank Aware Fine-tuning for a selected pool of scene graphs as a further optimization of the original models. This, coupled with the use of a small percentage of ground truth, enhances the retrieval capabilities of the models. Furthermore, we evaluate the impact of these different architectures by assessing both quantitative and qualitative results in the Graph Similarity task.

- The GNN models we propose generate similarity scores for all Scene Graphs, enabling their integration into a Counterfactual Explainer Framework, similar to the one outlined in Chapter 7. The resulting comparative results serve as the basis for generating counterfactual explanations. Importantly, there is limited previous literature that combines these elements to make this task of AI interpretability easier.

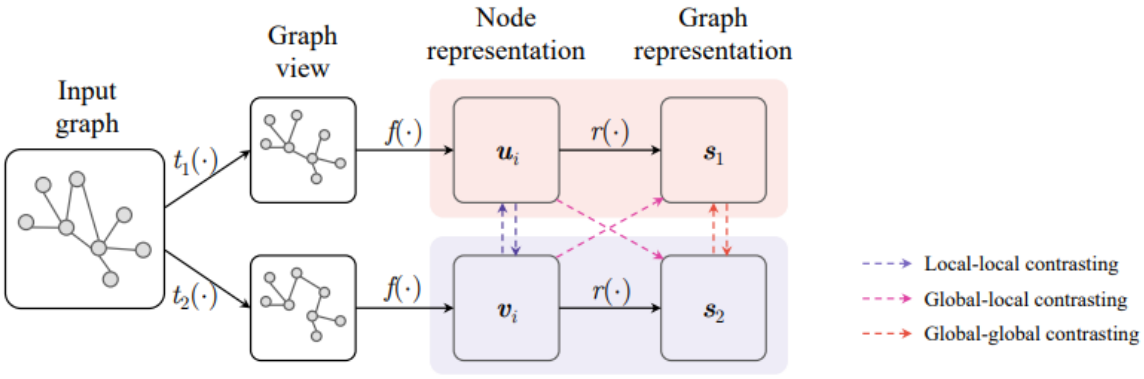


Figure 8.2.1: Training pipeline of Contrastive GNNs [96].

8.2 Proposed Models

The GNN models we experiment with are based on Graph Contrastive approaches applied at various levels of granularity within the graph. Specifically, we use GraphCL, InfoGraph, and Grace, each calculating the contrastive loss at the Graph-Graph, Graph-Node, and Node-Node levels, respectively as presented in Chapter 6. This explicitly determines the way the models are trained and evaluated.

- **Training:** Contrastive GNNs receive batches of graphs as input, with their primary goal being to generate meaningful representations that capture the underlying structure and relationships within the graph. Specifically, this is achieved by leveraging the idea that similar nodes/graphs should have corresponding representations. Consequently, in this training context, each dataset of N graphs provides N samples for training. In contrast, supervised GNN approaches for graph similarity utilize pairs of graphs from the original dataset. This straightforward distinction significantly reduces the training time of the GNN model from O(n^2) to O(n), where n is the number of samples.
- **Evaluation:** The evaluation process of the proposed Contrastive GNN models involves simply passing the graph as input to the trained GNN Encoder to obtain the final Node Embeddings. Subsequently, a Global Pooling method is employed to derive a single Graph Level Embedding. This procedure is quite fast, requiring minimal computational power (less than 1 second for the inference of 1000 graphs).

The training process follows the outline presented in Figure 8.2.1. The input consists of batched scene graphs, which are then processed by a GNN Encoder. The encoder comprises stacked layers of a GNN variant combined with other types of standard layers like activation, normalization, and dropout. The GNN model produces node-level embeddings, which are then pooled to create global graph embeddings.

During training, a set of scene graphs passes through the model, computing the contrastive loss under one of the modes: Local-Local, Global-Local, and Global-Global, and back-propagating it through the network. In particular, for the three different approaches we employed, leveraging contrastive loss at three different granularities of the graph, we use the Mutual Information estimators proposed by the authors. Specifically, in GraphCL [90], which functions under the Global-Global mode, and in Grace [95], which works under the Local-Local mode, we use the InfoNCE Mutual Information estimator as loss. For InfoGraph [77], operating under the Local-Global mode, we use the Jensen-Shannon Mutual Information estimator. Following the example of [20], we incorporate a projection head (e.g., MLP) after the base GNN encoder and calculate the contrastive loss in the embedding space spanned by this projection head. After training, the authors of SimCLR [20] recommend discarding the projection head. In our case, during inference, we experimented with both using and not using the projection head, selecting the model that exhibited the best performance each time.

The GNN encoder's implementation holds a pivotal role in the overall process. Figures 8.2.2 and 8.2.3 provide a detailed insight into the design of the **GAT/GATv2** [83, 14] and **GCN/GIN** [44, 89] variants, respectively.

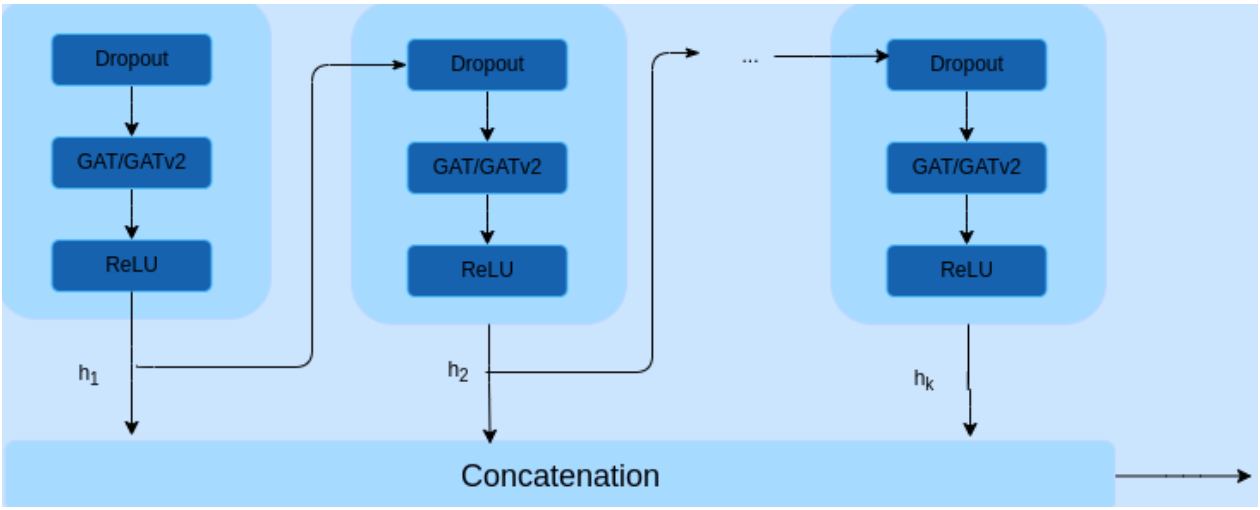


Figure 8.2.2: Design of GAT/GATv2 encoder variants

In the layers illustrated in 8.2.2, GAT or GATv2 convolutions are followed by the ReLU activation function. Incorporating dropout allows users to deactivate neurons with a probability parameter p to prevent overfitting. The attention variants demand careful consideration in dimension definition due to the application of multi-head attention, wherein input dimensions are multiplied by the number of heads in each layer.

Conversely, the architectures of GCN and GIN variants differ primarily through the utilization of batch normalization after each convolutional layer. Concerning GIN, designing an MLP model becomes necessary to train the weight parameter of the central node of the convolution. We chose a linear fully connected layer, succeeded by a ReLU activation function, and another fully connected linear layer, as proposed in the GIN paper [89]. Dropout is once again employed.

It is noteworthy that the node embeddings generated by each layer of the GNN encoders are concatenated before being pooled. This practice aims to preserve more information gathered throughout the process, creating more expressive graph embeddings, which have demonstrated superior performance in the scene graph retrieval task. Such an approach is also used in the GIN paper [89].

We must also establish a Global Pooling function for aggregating Node Embeddings into a singular Graph Embedding. Following trials with commonly used global pooling functions (mean/max/min/sum), it became apparent that the Sum pooling function delivered the most favorable results. As a result, all ultimate models incorporate this particular function. This decision aligns with the approach of GIN's [89] creators, who also chose to sum node embeddings to enhance expressivity. To optimize efficiency, these operations are preferably conducted in batches

For inference in the context of the Scene Graph Retrieval Task, the final graph embedding vectors are compared as classical vectors, and for this purpose, we decided to use cosine similarity. We also experimented with L_2 distance, but there was a significant performance drop in this case.

In all cases, within the frameworks of GraphCL, InfoGraph, and Grace, with careful hyperparameter tuning and the selection of the best GNN encoder in each instance, these models delivered competitive results in an inductive setting compared to Graph Kernels, surpassing them by a small margin. To further enhance the desired metrics, we draw inspiration mainly from [56] (for a more in depth view one can look at [19]), where weak supervision from the text modality (the image captions of the corresponding scene graphs) is leveraged to bring scene graphs with more similar captions closer together and push dissimilar ones apart. In pursuit of this, we modify the rank-aware loss to provide weak supervision from a distance perspective using Graph Edit Distance rather than using caption similarity. After initially pretraining a model within one of the frameworks GraphCL, InfoGraph, or Grace, we proceed to fine-tune it. During this fine-tuning phase, the model is exposed to a limited percentage of Graph Edit Distance (GED) ground truth. The rank-aware loss, defined in the equation below, operates in a contrastive fashion. It processes triplets of scene graphs.

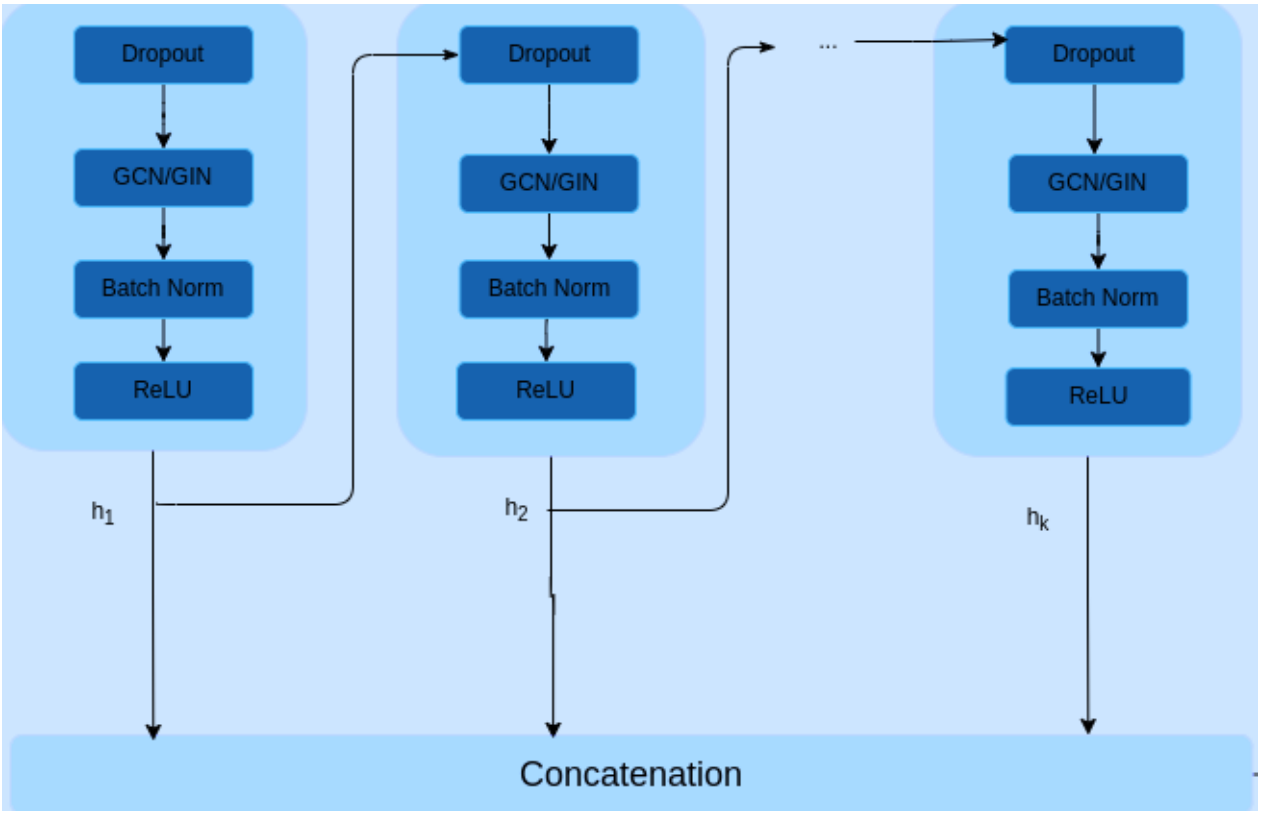


Figure 8.2.3: Design of GCN/GIN encoder variants

For a given anchor A, we uniformly and randomly sample two other scene graphs. The one with the smallest GED from the anchor is considered a positive sample P, while the other is treated as a negative one, N.

$$L = -Plog\hat{P} - (1-P)log(1-\hat{P})$$

In the equation above, the graph embeddings of A, P, N are denoted as f_A, f_P, f_N and are produced by the GNN model. The probability $\hat{P}$ of having similarities (or distances) in the correct order is calculated as $\hat{P} = \sigma\left(\frac{f_A^T f_P - f_A^T f_N}{\tau}\right)$. Additionally, $P(d_{AN} > d_{AP}) = P\left(\frac{1}{d_{AN}} < \frac{1}{d_{AP}}\right) = P(s_{AN} < s_{AP}) = \frac{d_{AN}}{d_{AP}+d_{AN}}$, represents the desired target value.

By using this loss, which 'punishes' the model whenever the negative sample is embedded closer to the anchor than the positive one, the initial meaningful embeddings produced through Mutual Information maximization are further improved. Consequently, the newly fine-tuned GNN can be employed for scene graph retrieval, resulting in even more competitive results compared to graph kernels.

The upcoming chapter will provide details on the particular hyperparameters and other selections (such as layers, latent size, epochs, batch size, optimizers, augmentations etc.) for each model.

Chapter 9

Experiments

In this chapter, we will provide a thorough explanation of the experiments conducted to assess the proposed models and compare them to our baseline, Graph Kernels. This section begins by presenting preliminary information about the Visual Genome dataset, along with the preprocessing steps applied to the scene graphs. Subsequently, we will delve into the utilization and implementation of Graph Edit Distance to derive the Ground Truth, accompanied by a detailed presentation of the Evaluation Metrics relevant to the Information Retrieval task.

Having laid the groundwork, we proceed to analyze our experimentation with the Contrastive GNN models, followed by the presentation of evaluation results. In addition to quantitative findings, we include visual representations of the most similar images to provide a qualitative and more intuitive insight into the performance of the proposed models.

Contents

9.1 Preliminaries . . . 106
9.1.1 Dataset . . . 106
9.1.2 Evaluation Metrics . . . 110
9.1.3 Ground Truth . . . 112
9.2 Training Details . . . 115
9.2.1 Graph Kernels . . . 115
9.2.2 GNNs . . . 116
9.3 Results . . . 117
9.3.1 Quantitative Results . . . 117
9.3.2 Qualitative Results . . . 122

9.1 Preliminaries

9.1.1 Dataset

The dataset that we will use is Visual Genome [47], a large-scale dataset focused on visual understanding using graph structured information, particularly in the realm of computer vision. It contains over 108k images which are detailed annotated, including objects, relationships, and attributes within the images as well as region and image descriptions. All these entities are also mapped to WordNet synsets. This dataset serves as a benchmark for evaluating the performance of algorithms and models in tasks related to image understanding, object recognition, and scene analysis. The scene graphs provided in the dataset are paired with images mainly from the MS-COCO [52] dataset.

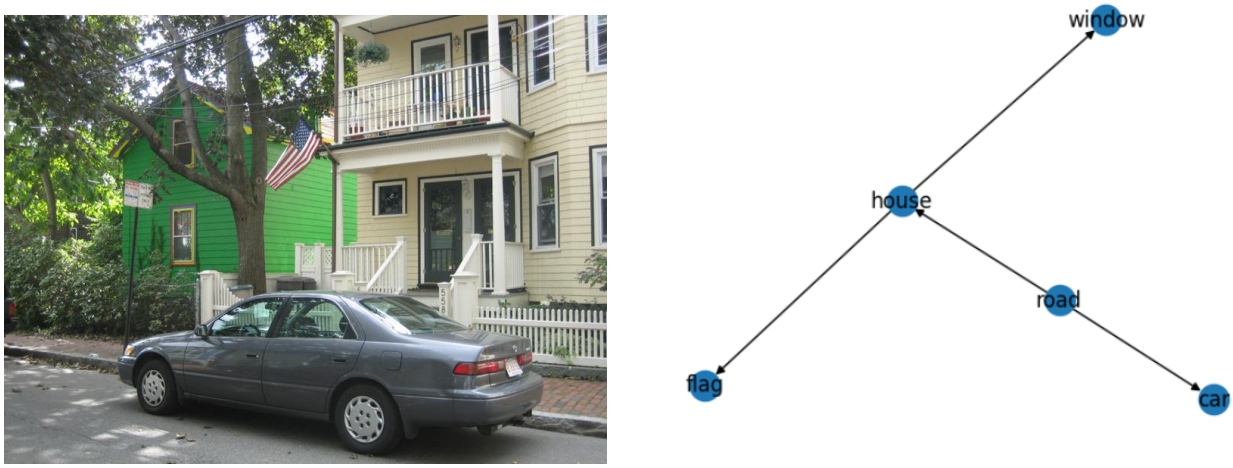


Figure 9.1.1: An example of an image alongside with its corresponding scene graph where the types of relationships have been removed

As it is evident from the figure above, scene graphs capture effectively the semantic content of an image by representing objects as nodes and their interactions as edges in the graph.

Data preprocessing

To process the scene graphs using graph neural networks, we need to preprocess them to determine the initial node attributes and establish our approach to utilizing the relationship information provided in the original dataset. Below, we provide a detailed explanation of our methodology.

- **Node Attributes:** The original Visual Genome scene graphs lack node attributes. However, Graph Neural Networks (GNNs) necessitate a feature matrix with numerical values as input. To overcome this limitation, we utilized the 'objects' field from the original dataset. Within this field, for each scene graph, we extracted the synsets and names associated with every object present in the image. To convert the string descriptions of objects into numerical vectors, we employed GloVe Word Embeddings [65], a well-established method in the field of Natural Language Processing (NLP) trained on word co-occurrence, effectively capturing the concepts conveyed by the words. Specifically, for each object name, we conducted a straightforward lookup in the GloVe table to obtain the corresponding node features. We opted for the 300-dimensional version of GloVe embeddings, as they consistently improved performance by providing more comprehensive information about the objects. In the same time, we kept the node synsets in order to later feed them to the GED algorithm that computes the ground truth using the "is-a" hierarchy provided by WordNet [59]. For this purpose, we removed all nodes without synsets assigned to them in order to make them WordNet compatible. The exact procedure will be presented in the ground truth subsection. Additionally, we removed the 'attributes' field from every object to exclude redundant information from our graphs. For instance, in the case of an image containing a car, potential attributes might include its color or brand. To ensure our models focus on the presence of the car rather than intricate details, we decided to disregard this additional information.

- **Edges:** To leverage the relationships field provided in the original dataset, we opted to represent all relationships as directed edges connecting the object to the specified subject. Similar to node attributes, relationships exhibit various types. However, we chose not to incorporate this additional information, given that the majority of graph kernels do not consider edge features. Moreover, the approximate bipartite matching Graph Edit Distance algorithm [29] used to compute the ground truth primarily accounts for node information. Besides that, the core semantic information of the images, in which we are mostly interested, resides in the objects represented as nodes and for simplicity we decided to disregard this additional information.

Following the aforementioned preprocessing, the subsequent task involves identifying the subset(s) of scene graphs that will be utilized for training and evaluation in the retrieval task. As we experiment with Contrastive Graph Neural Networks (GNNs) under various modes (Transductive and Inductive), we have initially selected specific subsets of scene graphs with defined attributes. In either case, we have set the number of scene graphs in each pool to be 1000, where each scene graph within a pool will serve as a query graph for the retrieval task, and the other 999 graphs will be used as answer graphs. In this manner, by ensuring that the Answer Set is sufficiently large to encompass a diverse range of graphs, the evaluation of the proposed retrieval models can be more accurate.

We will now describe the procedure we employed to define and select scene graphs from two different categories, namely Dense and Random Scene Graphs. The motivation behind this split into two pools of 1000 graphs is twofold.

On the one hand, we observed that the vast majority of scene graphs in the dataset were very sparse (e.g have low density). These graphs featured numerous isolated nodes, impeding the effectiveness of the message passing scheme executed by GNNs, as well as the functioning of various Graph Kernels reliant on node connections. A sparse (randomly selected scene graph) can be seen in Figure 9.1.3 where one can see the obious sparsity of the graph. The primary motivation behind this choice was to leverage the GNN message passing capabilities to their fullest extent in the dense subset, given the limitations posed by sparse scene graphs.

On the other hand, the selection of the random subset was driven by its characteristic of being an unconstrained sub-dataset, requiring no specialized handling. Furthermore, our interest extended to examining the adaptability of a pretrained Contrastive GNN in a random subset and assessing its performance when applied to a more constrained dataset like the dense one.

To create the Dense Set, we introduced specific criteria for selecting scene graphs. Specifically, we chose graphs with a minimum of 3 nodes and 3 edges, and their density values fell within the range of [0.1, 1.0]. The key element here lies in adding the density constraint to enhance the utilization of GNN Message-Passing capabilities. Graph statistics for the Dense Set are depicted in Figure 9.1.4.

For the Random Set, we simply picked 1000 random graphs, imposing two constraints. Firstly, the number of nodes had to be equal to or greater than 6, and the number of edges had to be equal to or greater than 3. These constraints were applied in order not to consider graphs with limited number of objects and relationships (edges). Basic statistics for the Random Set are depicted in Figure 9.1.2.

It is important to note that in subsequent experiments, if necessary (e.g., in inductive settings), we will utilize a distinct randomly selected set, subject to the same constraints, exclusively for training purposes. Inference will consistently be carried out within either the Original Dense or Random Sets.

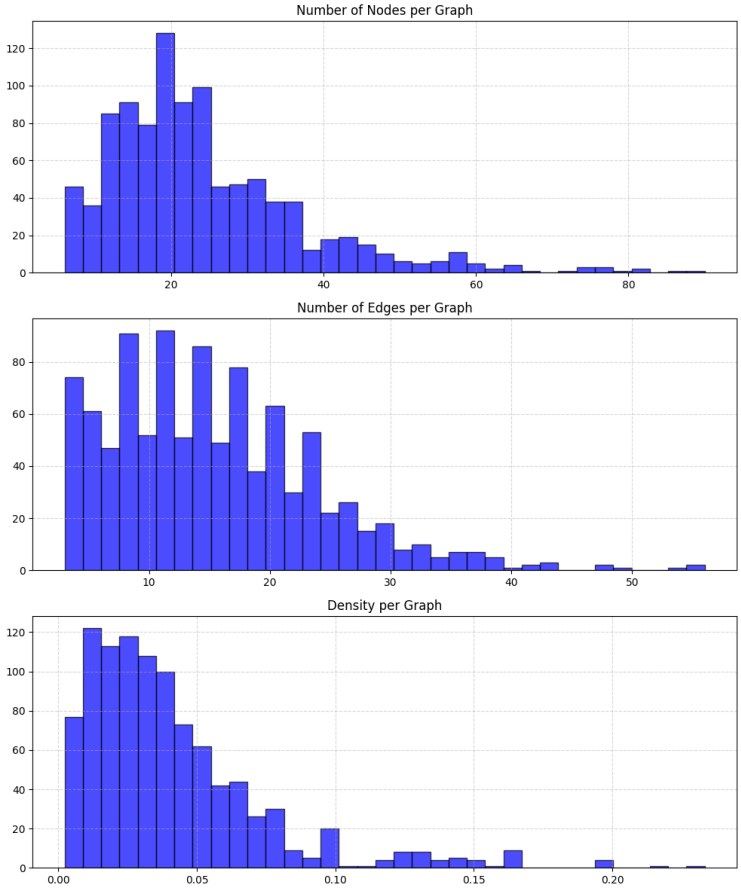


Figure 9.1.2: Statistics for scene graphs in the Random Set

Figure 9.1.3: An example of an image alongside with its corresponding scene graph from the Random Set.

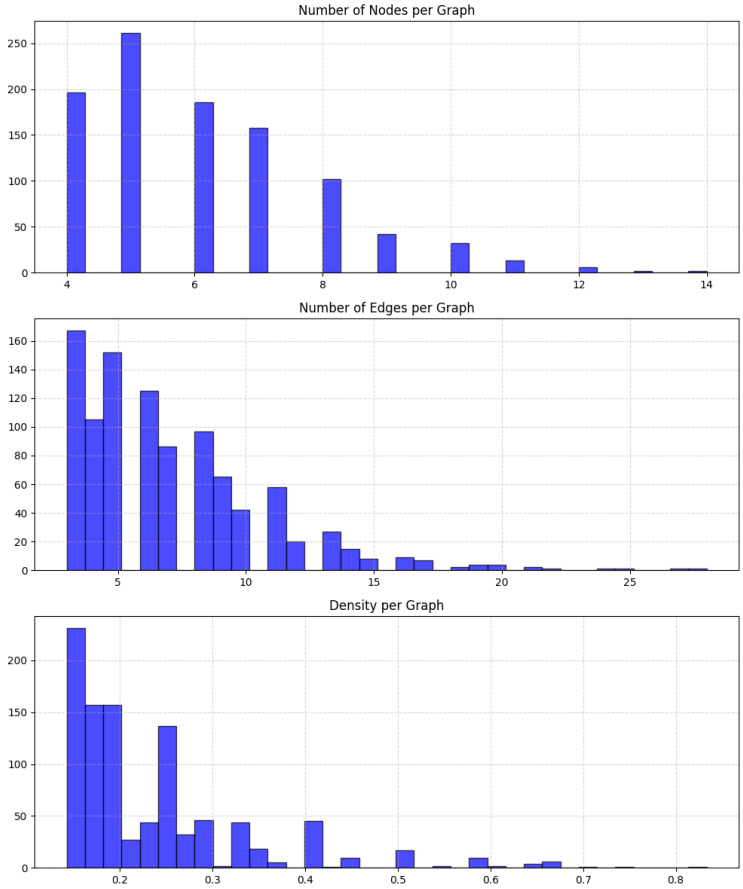


Figure 9.1.4: Statistics for scene graphs in the Dense Set

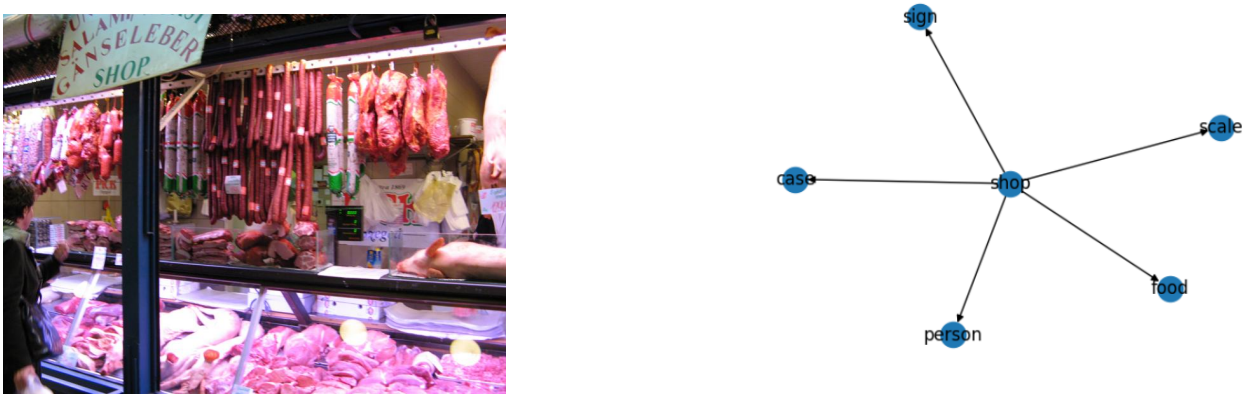


Figure 9.1.5: An example of an image alongside with its corresponding scene graph from the Dense Set.

9.1.2 Evaluation Metrics

As it has been made evident through this thesis, the task for which we employ Graph Kernels and Contrastive Graph Neural Networks is an Information Retrieval task. Consequently, the methods used ultimately output ranked lists of the most similar scene graphs for a given query. Therefore, the metrics suitable for comparing results with the ground truth, as established by the approximated Bipartite Matching GED algorithm, and for evaluating our models' performance are traditional information retrieval metrics.

In the context of information retrieval, MAP (Mean Average Precision), MRR (Mean Reciprocal Rank), and NDCG (Normalized Discounted Cumulative Gain) are important and the most prevalent evaluation metrics used to assess the performance of information retrieval systems, and these are the metrics that we will use.

These metrics are offline, order-aware, and suitable for evaluating Information Retrieval Systems by comparing the rankings of 'Actual' and 'Predicted' results for the same Query object. 'Order-Aware' indicates that returning an actual relevant result at rank one is more favorable than at rank four within the context of these metrics. An essential aspect of this process involves determining the size of relevant answers and the number of predicted elements for evaluation. To strike a balance, we have chosen to consider as 'Relevant' the first 50 elements of the 'Actual' rankings and evaluate on the first 10 of the 'Predicted' elements.

This prevents overly optimistic metrics (such as MRR) when the size of 'Relevant' elements is too long or a loss of effectiveness when it's too short due to a high number of 'incorrect' items in the 'Predicted' ranking. These considerations are crucial for ensuring a meaningful evaluation of our models. In the upcoming definitions, we will practically set k to 10 in our experiments.

Mean Reciprocal Rank (MRR@K)

The Mean Reciprocal Rank (MRR) is a metric that evaluates the effectiveness of a recommendation system by considering the reciprocal rank of the first relevant item returned in the "Predicted" list of elements. Reciprocal rank is the multiplicative inverse of the rank of the first relevant item. The idea behind MRR is to reward systems that place relevant items higher in the recommendation list.

The MRR@K for a set of queries is defined as:

$$\text{MRR@K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

where Q is the set of queries, rank_i is the rank of the first relevant item returned for query i, and $|Q|$ is the total number of queries. Some key characteristics of MRR@K include:

- MRR focuses on the rank of the first relevant item, emphasizing the importance of presenting relevant items early in the list. For this reason, it is in general easily interpretable compared to other metrics
- MRR values range from 0 to 1, where 1 indicates that the first relevant item is always ranked first across all queries
- Nevertheless, MRR has its constraints. It fails to consider the placements of subsequent pertinent items beyond the initial one and treats all relevant items with equal importance. To illustrate, suppose we want to suggest around 10 products to a user. We request the information retrieval (IR) system to fetch 10 items. It's possible that only one item in the top rank is truly relevant, with the rest being irrelevant. While having nine out of ten irrelevant items is a disappointing outcome, the Mean Reciprocal Rank (MRR) would still yield a perfect score of 1.0. Consequently, it is frequently employed in conjunction with other metrics like Average Precision or NDCG to provide a more comprehensive evaluation

Mean Average Precision (MAP@K)

Mean Average Precision (MAP) is a metric that assesses the quality of a retrieval system. It quantifies the average precision values for various retrieval queries and then computes their mean. Precision is a measure of accuracy, specifically the ratio of relevant instances retrieved to the total instances retrieved. The main components essential for the calculation of MAP@K are presented below.

1. **Precision:**
   - Precision is calculated for each position (up to K = 10 for our experiments) in the "Predicted" ranking list returned by the retrieval system. It is the ratio of relevant instances to the total number of instances retrieved up to that position.
   - Precision at rank k is given by:

$$\text{Precision@}k = \frac{\text{Number of relevant items among the top } k}{k}$$

2. **Average Precision (AP):**
   - Average Precision is the average of precision values at all positions where a relevant element is present in the "Predicted" List. It is computed by summing the precision values at each relevant position and dividing by the total number of relevant instances.
   - It is defined as following:

$$AP@K = \frac{\sum_{k=1}^{K} Precision@k \times \text{rel}(k)}{\text{Total number of relevant instances}}$$

   where K is the total number of items in the "Predicted" list (e.g in our experiments K = 10), and rel(k) is an indicator function equal to 1 if the item at rank k is relevant and 0 otherwise.

3. **Mean Average Precision (MAP):** MAP is the mean of AP values calculated in the previous step across multiple queries. It is defined as:

$$MAP@k = \frac{\sum_{i=1}^{Q} AP@K}{Q_i}$$

   where Q is the total number of queries.

An important aspect of MAP@K metric is that it places importance on the correct ordering of relevant items. A system that consistently ranks relevant items higher will have a higher MAP score. In general, MAP@K is particularly relevant in scenarios where the order of retrieval matters, such as search engine results, recommendation systems, and object detection. However, it is sensitive to the number of relevant items in the dataset as a small number of relevant items may lead to a less reliable evaluation. MAP@K provides a clear, interpretable measure of system performance, as it directly relates to the precision and ordering of relevant items.

Normalized Discounted Cumulative Gain (NDCG@K)

Normalized Discounted Cumulative Gain (NDCG@K) is a widely used metric in information retrieval and recommendation systems to evaluate their effectiveness. It measures the quality of a ranked list of items by considering both the relevance and the position of each item within the list. Let's break down the components and concepts associated with NDCG@K:

1. **Cumulative Gain (CG):** Cumulative Gain is a measure that sums up the relevance scores of the top K (in our experiments K = 10) items in the "Predicted" ranked list returned by a retrieval model. The relevance score is different this time. It is a range of relevance ranks where 0 is the least relevant, and some higher value is the most relevant. In order to define the relevance scores in our experiments, we use the inverse of the Graph Edit Distance Ground Truth as the concept of relevance is inverse to that of Edit Distance. Then, using a Min-Max Scaler we transform these values into a scale of 1 to 10 in order to avoid extreme values. The formula for Cumulative Gain at position K is as follows:

$$CG@K = \sum_{i=1}^{K} rel_i$$

2. **Discounted Cumulative Gain (DCG):** DCG is an extension of Cumulative Gain that introduces a discount factor to give less importance to items that appear lower in the ranked list. The discount factor is often logarithmic to penalize lower-ranked items more. The formula for DCG@K is:

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{\log_2(i+1)}$$

In this way, higher scores are assigned to elements that are relevant and also appear higher in the "Predicted" List.

3. **Ideal Cumulative Gain (IDCG):** IDCG represents the maximum achievable Cumulative Gain for a given set of items. It is obtained by sorting the items based on their true relevance scores in descending order and calculating the Cumulative Gain. The formula for IDCG@K is similar to DCG@K, but it considers the ideal ordering:

$$IDCG@K = \sum_{i=1}^{K} \frac{rel_{\text{ideal},i}}{\log_2(i+1)}$$

where $rel_{\text{ideal},i}$ is the relevance of the ideal item at position i in the sorted list.

4. **Normalized Discounted Cumulative Gain (NDCG):** NDCG is the normalized version of DCG, calculated by dividing the DCG@K by the IDCG@K. This normalization ensures that the metric falls within the range [0, 1], with 1 indicating a perfect ranking. The formula for NDCG@K is:

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

NDCG is particularly useful when comparing results across different datasets or systems, as it provides a standardized measure for quantitative comparison. However, it is much more complex than other metrics as a results of the complicated calculations involved. As a result, it is not easily intuitively interpretable/ In summary, NDCG@K is a comprehensive metric that considers both the relevance and position of items in a ranked list. In our case, where we have multiple queries, we just calculate the mean value of NDCG across all queries.

9.1.3 Ground Truth

It is apparent from the Evaluation Metrics subsection that to assess the effectiveness of the intended models, it is essential to identify the actual relevant outcomes for each query. To accomplish this objective, it is necessary to establish a quantitative measure, ultimately resulting in the computation of the ground truth. As we have already mentioned in subsection 5.2.1, the ground truth similarity score between two graphs is computed by the Graph Edit Distance algorithm. In our case, we need to compute a ground truth matrix where the graph edit distances between any pair of graphs will be stored. However, the computational cost of the exact Graph Edit Distance is prohibitive as it does not exist a polynomial algorithm for it.

To generate the ground truth matrix for the Random and Dense Sets, each comprising 1000 graphs, the GED algorithm must be executed approximately 500,000 times for each set. To alleviate the computational burden associated with GED computation, we employed the Bipartite Matching approximation algorithm, leveraging the Volgenant-Jonker assignment algorithm [41] as outlined in Section 5.2.1. In essence, the Bipartite Matching approximation initially calculates the exact edit distance, focusing solely on node edit operations like insertions, deletions, and substitutions. Subsequently, it deduces the edge edit operations. The node edit distance problem is treated as a Linear Sum Assignment Problem (LSAP), where the Volgenant-Jonker (VJ) algorithm is employed. This algorithm efficiently solves the LSAP in polynomial time. This approximation algorithm was selected as it offers a reliable estimation of the optimal solution in real-world scenarios while maintaining a comparatively low computational cost. In order to implement the approximate Bipartite Graph Matching Edit distance algorithm, we used the Deep Graph Learning library in Python, known as DGL, as detailed by Wang et al. in their work [86].

However, we should first provide the algorithm with the costs of insertion, deletion and substitution of nodes. For this purpose we leverage the WordNet [59] Tbox graph in order to compute the concept distance between every two objects that we need as proposed in [31].

WordNet is a lexical database of the English language that organizes words into sets of synonyms (synsets), representing concepts or meanings. Developed at Princeton University, WordNet is widely used in various natural language processing tasks.

One significant aspect of WordNet is its hierarchical structure, particularly the *is-a* hierarchy, which captures hypernym-hyponym relationships. In this hierarchy, a *hypernym* is a more general term or concept, while a *hyponym* is a more specific term or concept. The *is-a* relationship signifies that a hyponym is a subtype or instance of its hypernym. For instance:

- **Hypernym:** Animal
  - **Hyponyms:** Mammal, Bird, Fish

In this example, "Mammal," "Bird," and "Fish" are hyponyms of the hypernym "Animal." This hierarchical structure allows for a more nuanced understanding of the relationships between words, making WordNet a valuable resource for tasks such as word sense disambiguation, information retrieval, and semantic similarity analysis. The *is-a* hierarchy is crucial for capturing the hierarchical organization of concepts in the English language.

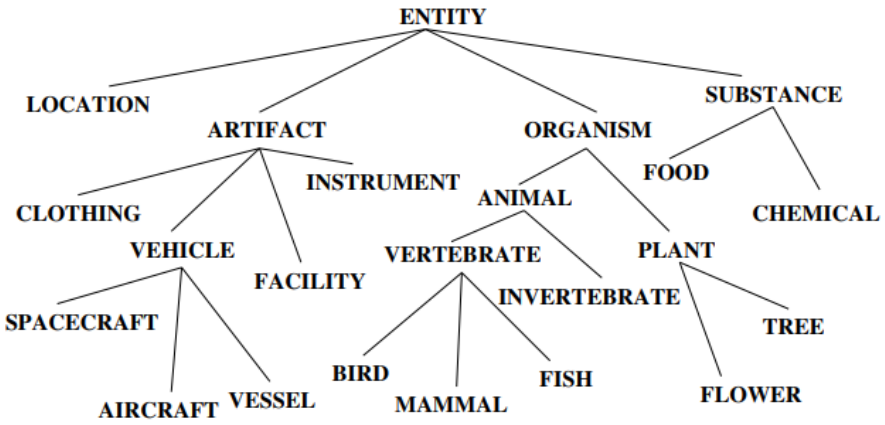


Figure 9.1.6: A simple representation of the portion of the WordNet noun hierarchy below the concept root node “entity”.

As the Visual Genome dataset provides a corresponding Synset for each object in a scene graph, and since we have represented each object as a node in a graph, the utilization of WordNet for measuring concept distance becomes straightforward. Specifically, we utilize the NLTK Python package [53] which provides us with access to the WordNet database and the Noun Hierarchy.

In order to compute the concept distance between two synsets in the is-a hierachy we use the `path_similarity` function that the NLTK package encompasses. This function calculates a score ranging from 0 to 1 denoting the similarity between two word senses based on the shortest path in the is-a (hypernym/hyponym) taxonomy.

Thus, in order to provide the Bipartite Matching algorithm with the costs of Node Substitutions we use the `path_similarity` to derive the similarity score for all possible pairs of nodes between graph G_i and graph G_j. Afterwards, the concept distance between every pair of nodes is calculated as $1 -$ similarity_score. For Node Insertion and Deletion costs, the same procedure is followed but instead of using the `path_similarity` function for pairs of concepts, we use it to calculate the similarity between each node and the Entity Node which is the root node of the is-a hierarchy.

This procedure is repeated for every possible pair of scene graphs both for the Dense and the Random Sets and the final ground truth matrices are obtained. The runtime for the "Random" dataset was approximately $\sim$ 8 hours, and for the "Dense" dataset, it was approximately $\sim$ 2 hours.

Here, we present a set of representative images and analyze their similarity scores using the Graph Edit Distance metric we defined.

In Figure 9.1.7, two distinct images are featured, each with identical scene graphs. The effectiveness of the Bipartite Matching algorithm is evident as it accurately identifies a graph edit distance of 0, indicating the similarity between the images based on their respective scene graph representations.

The last figure, 9.1.8, presents two images—one from a bathroom and the other from a tennis game—with structurally very similar scene graphs. Despite this, the ground truth approximate GED algorithm assigns them a high score, and the tennis court image doesn't even appear in the top 20 ranking list. This observation underscores the crucial role of object semantics, as the approximate GED algorithm places significant importance on these semantics. For instance, the concept of a 'tile' is notably distant in the WordNet hierarchy from the concept of a 'player'. All these observations substantiate the effectiveness of the Bipartite Matching algorithm as an excellent similarity measure, affirming our decision to employ it as our ground truth

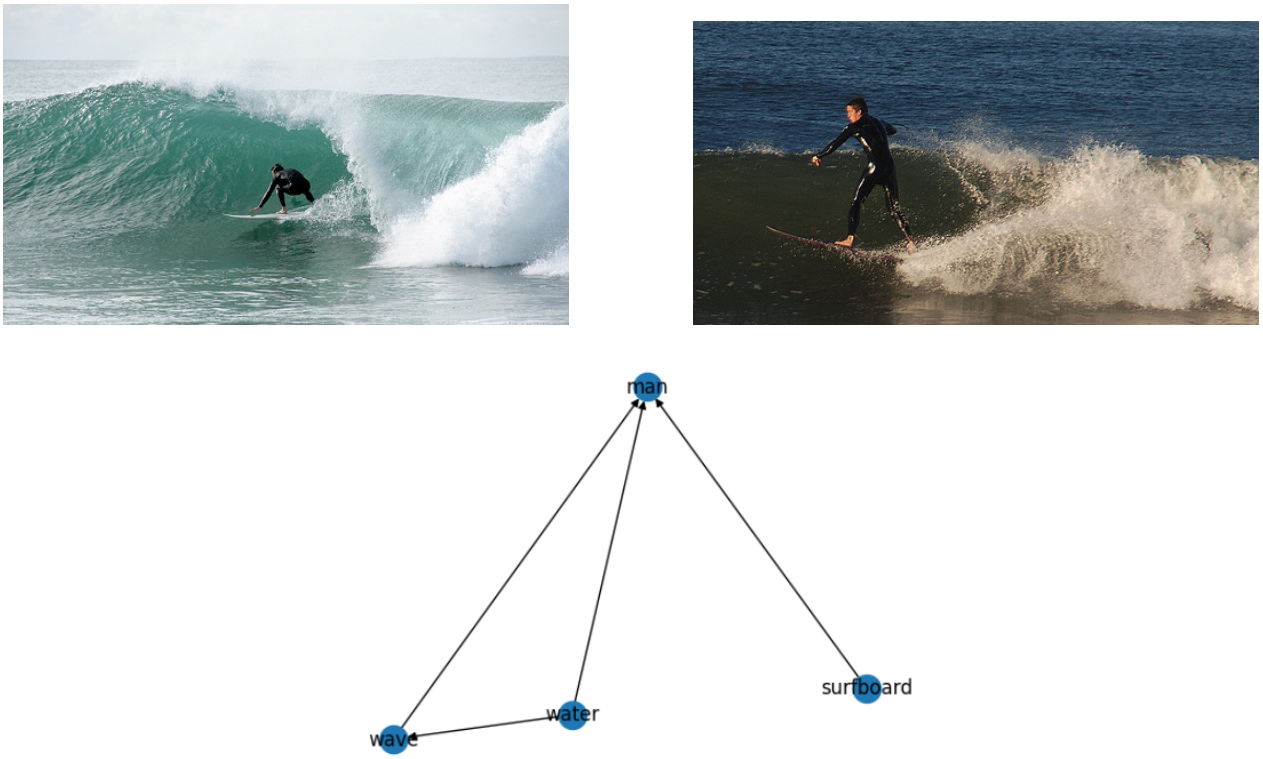


Figure 9.1.7: Here are two images from the Dense Set, each featuring an identical scene graph. The approximate GED algorithm accurately assigned a distance score of 0, recognizing that they share the same semantical content

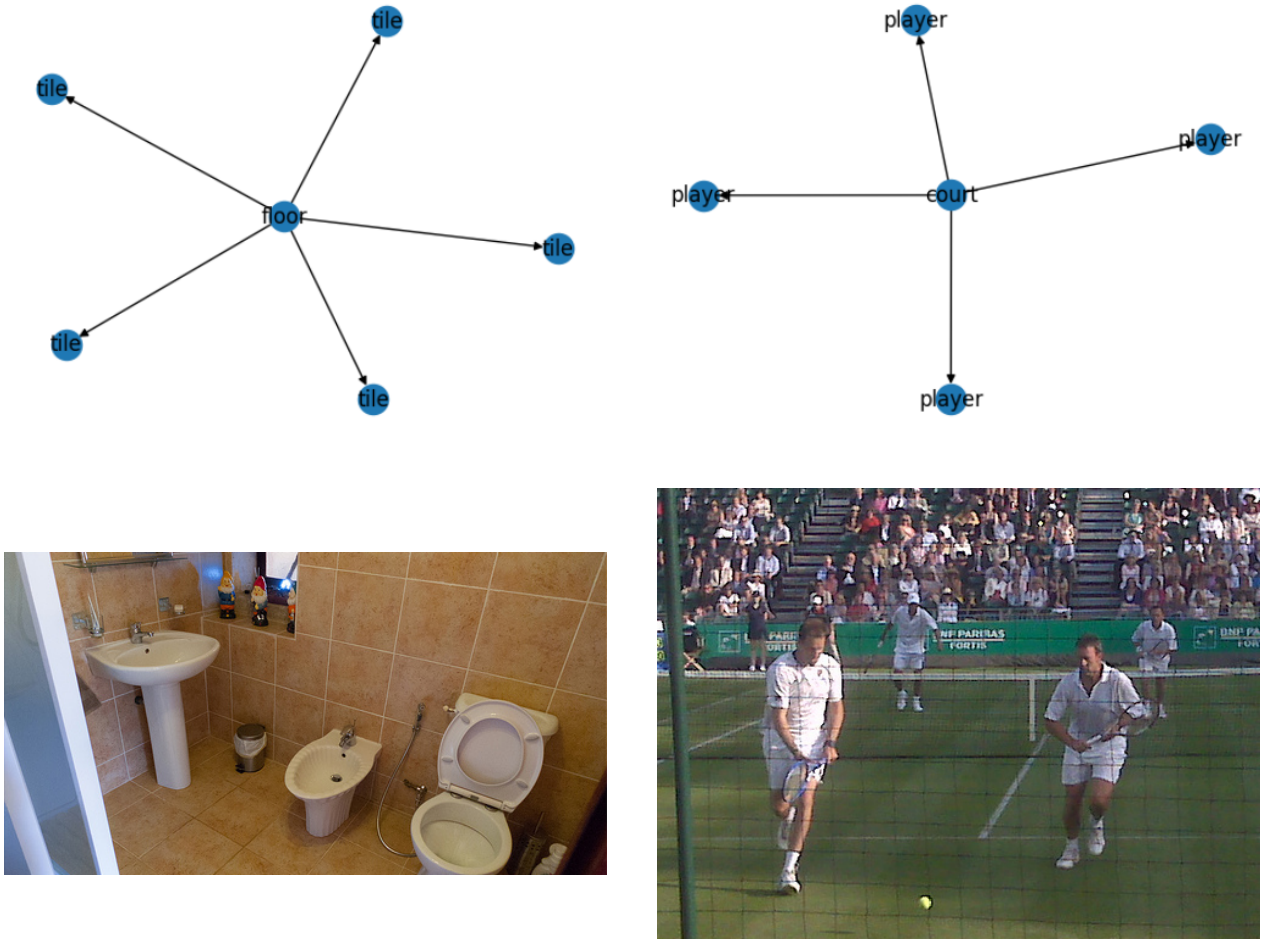


Figure 9.1.8: An illustration of two structurally similar images scored with a significant distance by the Bipartite Matching GED algorithm due to their distinct semantic content.

9.2 Training Details

9.2.1 Graph Kernels

For the conducted experiments, it is crucial to establish baseline techniques to compare them with our Contrastive GNN models in the task of scene graph similarity. As detailed in subsection 5.2.2, the predominant approach for assessing graph similarity involves the utilization of graph kernels. Consequently, various kernels, such as Weisfeiler-Lehman (WL), Shortest Path, Neighborhood Hash, Random Walk, and Graphlet Sampling Kernel, were evaluated on our subset of graphs using the GraKeL Python library [75].

The operations of the aforementioned kernels are detailed in subsection 5.2.2. Much like the ground truth, these kernels generate similarity matrices for each pair of graphs. It's noteworthy that, in contrast to graph kernels, GED serves as a dissimilarity measure, offering scores that signify the extent of dissimilarity between pairs of graphs.

The GraKeL library [75] offers a user-friendly interface for determining the values of different parameters utilized in various variants of graph kernels. The parameters that we used for each of the five graph kernels are the following:

Weisfeiler-Lehman Kernel: The Weisfeiler-Lehman Kernel takes two main parameters. The first one is the base kernel k used between two graphs, and the second one is the number of iterations h used. We set the iteration number to 20, and we use the default base graph kernel, which is the Vertex Histogram Kernel.

Random Walk Kernel: For the Random Walk Kernel, we employed the RandomWalkLabeled kernel variation, which also considers node labels. The default kernel type and lambda parameters were used, namely geometric with a lambda (λ) value of 0.1. Geometric simply determines the way the inner summation will be performed as described in subsection 5.2.2.

Shortest Path Kernel: For the Shortest Path Kernel, two main parameters need to be determined. Firstly, the most crucial parameter is called `algorithm_type`, which determines the underlying algorithm used to compute the shortest paths. One can choose among the Dijkstra or Floyd-Warshall algorithm or opt for the Auto selection, which automatically selects the faster option based on input graphs. The other parameter is denoted as `with_labels`, determining whether nodes have labels or not. For both parameters, we used the default setting of Auto algorithm and `True` for `with_labels`.

Neighborhood Hash Kernel: For the Neighborhood Hash Kernel, two important parameters should be determined, called R and `bytes`, representing the maximum number of Neighborhood Hash iterations and the byte size of node hashes, respectively. We opted for the default values for both parameters, setting them to 3 and 2.

Graphlet Sampling Kernel: The main parameter for this kernel is the maximum size of graphlets to be used, and we set this to the default value, which is 5.

9.2.2 GNNs

In order to build and evaluate the proposed models, we used the graph learning library Pytorch Geometric [30] and we also used some tools from the PyGCL library [96] for our contrastive modules.

An important observation for our experiments concerning the training and testing data lies in the concepts of inductive and transductive learning, which we will explain shortly below.

Inductive Learning: Inductive learning is a type of learning where the system tries to learn the underlying patterns or rules from training data in order to make predictions about unseen, future data. The goal of inductive learning is to generalize from specific instances to general rules or patterns. In other words, it aims to infer a general function from specific examples. Once the model is trained, it can be used to make predictions on new, unseen data.

Transductive Learning: Transductive learning, on the other hand, is a type of learning where the system attempts to predict the labels or properties of specific, unlabeled instances based on the labeled data available. Unlike inductive learning, transductive learning doesn't aim to learn a general model of the underlying data distribution. Instead, it focuses on making predictions for the specific unlabeled instances provided.

Transductive learning is particularly useful when the goal is to predict labels for a specific set of instances without necessarily generalizing to new, unseen data. It can be thought of as a form of semi-supervised learning, where the model leverages both labeled and unlabeled data to make predictions.

In graph machine learning, transductive learning is particularly relevant due to the interconnected nature of graph data. Graphs often represent complex relationships between entities, such as social networks, biological networks, or citation networks. By exploiting the local and global structural information encoded in the graph, transductive learning methods can effectively make predictions for unlabeled nodes based on their proximity and relationships with labeled nodes. Transductive learning approaches are crucial for tasks such as node classification and link prediction among others.

In the experiments conducted, we explored both Transductive and Inductive settings. Specifically, the initial experiments presented fall under the inductive setting, where we train our models on a randomly selected set of scene graphs. Subsequently, inference is performed on both the original Random Set and the original Dense Set to evaluate the models' generalization across scene graphs with different attributes compared to graph kernels.

Furthermore, the proposed fine-tuning does not aim to learn a general model of the underlying data distribution. Instead, it focuses on enhancing the models' performance in predicting for a specific unlabeled set of graphs provided (e.g., the Dense subset) using only a small percentage of labels thus falling into the

transductive category. This approach allows the initial pre-trained models on a random subset to be adapted to specific sets of scene graphs.

For the initial inductive experiments, we separately tested all four variants we examined: GCN, GAT, GATv2, and GIN, with each of the three contrastive modules: GraphCL, InfoGraph, and Grace. Then, for the fine-tuning process on the dense set, we experimented with the most successful combinations.

For the hyperparameters of the GNN models, we will first list those that were common across all model variations. We employed the Adam optimizer for training all models, and we also conducted experiments with the optimizer's parameters, particularly the learning rate and weight decay. After careful experimentation, we set the learning rate equally to $1e-4$ for all the models.

We also experimented with the number of layers used, and we concluded that models with 2 layers and concatenation of the results of the first and second layers outperformed other combinations. Another important parameter is the dimensionality of the final embeddings. We determined that the optimal solution was for them to be 512.

Moreover, the convolutional variants that utilize multi-head attention require specific attention to the number of heads. Experiments were conducted with 1, 2, and 4 heads, among which the configuration with 4 heads yielded consistently the best results.

Finally, the hyperparameters of epochs, batch size and augmentations are critically important for Contrastive Graph Neural Networks (GNNs). In general, contrastive learning hinges on the principle of bringing positive samples (similar or semantically related samples) closer together in the embedding space while pushing negative samples (dissimilar samples) farther apart. These positive and negative samples play a crucial role in encouraging the model to learn meaningful representations, typically obtained from within each batch. Therefore, the batch size is a vital hyperparameter in this context. In general, contrastive learning tends to benefit from larger batch sizes, although it's essential to strike a balance. A batch size that is neither too small nor too large is optimal for achieving effective learning outcomes.

Regarding augmentation techniques, our comprehensive experimentation revealed that node and edge dropout, along with feature masking, showed the most promise. For the GraphCL variant, we ultimately selected node and edge dropout. However, for the Grace variant, we decided on edge dropout and feature masking to preserve the necessary node alignment for the Local-Local contrastive loss. In general, we observed that relatively small percentages for dropout and feature augmentations were more beneficial for constructing positive samples, as opposed to large percentages which tended to result in significant corruption of the initial graph.

Furthermore, for all Graph Neural Networks (GNNs), we employed a projection head, such as a Multi-Layer Perceptron (MLP), after the global pooling operation. This projection head is utilized to map the graph embedding into another latent space where the contrastive loss is computed. Typically, this MLP head comprises a single fully connected linear layer followed by a ReLU activation, then another linear layer. When training Contrastive GNNs solely in an unsupervised manner, we discovered that graph embeddings generated immediately after the global pooling operation yielded superior performance for all convolutional variants, with the exception of Graph Attention Network (GAT) and GATv2, where the use of the projection head proved beneficial during inference. Concerning the fine tuning procedure on the dense set, we found that training the GNN along with the MLP head and then using these two modules together for inference was the best performing option.

Below we summarize in the provided table the best combinations that were utilized for each of the models:

9.3 Results

9.3.1 Quantitative Results

Firstly, we will present the final MAP, MRR, and NDCG scores achieved by both the Graph Kernels and GNN Models on the original Random Set. Each specific GNN model corresponds to the Optimal model with hyperparameters detailed in Table 9.1. All GNNs are trained on a separate random subset of 1000 scene graphs. The Evaluation Metrics are displayed in Tables 9.2 and 9.3. In the former, we observe the

Table 9.1: Optimal Hyperparameters

Model	GNN	Epochs	Batch Size	Augmentations	Attention Heads
GraphCL	GCN	50	64	Node/Edge Dropout $p = 0.05$	-
	GIN	40	128	Node/Edge Dropout $p = 0.05$	-
	GAT	20	128	Node/Edge Dropout $p = 0.05$	4
	GATv2	20	128	Node/Edge Dropout $p = 0.05$	4
InfoGraph	GIN	40	128	-	-
Grace	GCN	50	128	Node Dropout/Feature Masking $p = 0.15$	-
	GIN	80	128	Node Dropout/Feature Masking $p = 0.15$	-
	GAT	20	16	Node Dropout/Feature Masking $p = 0.15$	4
	GATv2	40	16	Node Dropout/Feature Masking $p = 0.15$	4

performance of graph kernels and unsupervised GNNs trained contrastively, while in the latter, we observe the performance of the best GNN models after the fine-tuning process, which exposes them to a small percentage of ground truth.

Table 9.2: Final Metric Scores for the Graph Kernels and the GNN models without fine tuning, on the Original Random Subset in an inductive setting. Training has been performed on another random subset. Bold denotes the best result for each architecture.

	MAP@10	MRR@10	NDCG@10
Shortest Path Kernel	**0.1148**	**0.4862**	0.6851
Random Walk Kernel	0.0508	0.2509	0.5971
Weisfeiler-Lehman Kernel	0.0995	0.4315	0.6560
Graphlet Sampling Kernel	0.1030	0.3810	0.6296
Neighborhood Hash Kernel	0.1035	0.4733	**0.7112**
GraphCL			
GCN	0.1123	0.4213	0.6737
GIN	**0.1157**	**0.4614**	**0.6882**
GAT	0.0919	0.4157	0.6788
GATv2	0.0933	0.4068	0.6805
InfoGraph			
GIN	0.1229	0.4649	0.6982
Grace			
GCN	0.1016	0.4291	0.6819
GIN	0.1122	0.4550	0.6872
GAT	**0.1243**	0.4769	0.7479
GATv2	0.1223	**0.4892**	**0.7504**

The first observation we make with Graph Kernels is the superiority of the Shortest Path Kernel in terms of MAP@10 and MRR@10, along with the Neighborhood Hash Kernel, which performs competitively across all metrics and outperforms all kernels in terms of NDCG@10. The other three Kernels are surpassed by these two on all three metrics. The underperformance of the Random Walk kernel can be explained if we correlate it with its underlying concept. Specifically, this kernel relies on fundamental structural properties of a graph, particularly in random walks. However, scene graphs are typically small graphs, and moreover, the random subset is quite sparse. Consequently, random walks are not as informative about scene graphs. Similarly, the graphlet sampling kernel also relies on structural properties of the graph, specifically on the appearance of prototypes (graphlets), and for this reason, it does not perform adequately. On the other hand, Weisfeiler-Lehman and Neighborhood Hash Kernels rely on the labels of graphs and implement a procedure of re-labeling the nodes and updating the information. This procedure resembles a lot the message passing

Table 9.3: Final Metric Scores for the best GNN models with fine tuning, on the Original Random Subset with supervision from a small subset of the ground truth. Bold denotes the best result for each metric.

	MAP@10		MRR@10		NDCG@10	
	10 epochs	**30 epochs**	**10 epochs**	**30 epochs**	**10 epochs**	**30 epochs**
GraphCL						
GIN	0.1431	0.1881	0.5271	0.5624	0.7436	0.8009
InfoGraph						
GIN	0.1738	0.2164	0.5311	0.5717	0.7747	0.8120
Grace						
GAT	**0.1957**	0.2177	**0.5864**	0.5840	**0.8117**	0.8176
GATv2	0.1935	**0.2350**	0.5616	**0.6005**	0.8075	**0.8227**

scheme employed by the majority of GNNs. This approach seems to be successful in considering as much graph information as possible to compare pairs of graphs and extract similarity scores. The Neighborhood Hash kernel indeed performs best in terms of NDCG@10 with a 3% improvement compared to the Shortest Path Kernel and has the second-best MRR@10 and MAP@10 scores. However, the Weisfeiler-Lehman Kernel is third in terms of MRR@10 and NDCG@10 and only fourth in MAP@10, but it is close to the top three. We expected it to perform better, but it is generally competitive with the other kernels, with the exception of the Random Walk kernel that performs poorly.

Regarding the Contrastive GNN models, the initial observation is that all of them are competitive against several graph kernels. However, they struggle to outperform the best scores of the baseline Graph Kernels, especially in terms of NDCG, as well as the other two metrics. However, in the context of the Grace framework, where the contrastive loss is performed at the level of node embeddings rather than graph embeddings, we observe that the use of attentional variants GAT and GATv2 can outperform the best kernels in all metrics. Specifically, when combined with the GATv2 variant, it outperforms the Neighborhood Hash Kernel by a margin of 4% in the NDCG@10 metric.

A general observation about the combination of the three different frameworks with the four different GNN variants is that when graph embeddings are used in the contrastive loss (e.g., GraphCL, InfoGraph), the GIN variant is the most competitive across all metrics. This fact justifies the fame of GIN as one of the most powerful GNNs in tasks at the graph level. However, when the contrastive loss is performed at the node level, the attentional variants, utilizing attention between node embeddings, appear to be quite beneficial surpassing all other combinations. It should also be noted that the sparsity of the Random Set does not appear to significantly influence the performance of contrastive learning on a large scale.

Moving on to more complex architectures, we can see that the Rank Aware Fine Tuning, using a small percentage of the ground truth of the original Random Set, provides a significant boost across all three metrics as it is evident from Table 9.3. Let's first determine the percentage of the ground truth that corresponds to 10 and 30 epochs in the worst case scenario. In each epoch, for each of the 1000 graphs, 2 graphs are sampled as positive and negative samples. Consequently, one epoch exposes the model to the relative similarity of positive and negative samples to the anchor, making use of 2000 ground truth pairs. The total number of ground truth pairs for 1000 graphs is 500,000. As a result, 10 epochs correspond to the use of 4% of the ground truth, and 30 epochs correspond to the use of 12% of the ground truth. Thus, by utilizing a small percentage of the ground truth, we are able to achieve values of MAP@10, MRR@10, and NDCG@10 that are more than 10% higher compared to the best results from the graph kernels with 30 epochs of fine-tuning. These results were attained using the most promising combinations, particularly the Grace Framework with GAT/GATv2 variants. Moreover, 10 epochs are also enough in order for one to see a significant boost to the metrics.

In order to see how our models perform to another Set of scene graphs with specific and constrained characteristics we decided to test the best unsupervised contrastively pretrained (trained on a random set of 1000 scene graphs) GNN models on the original Dense Set. The final MAP, MRR, and NDCG scores achieved by both the Graph Kernels and GNN Models on the original Dense Set are displayed in Tables 9.2. In Table

9.3 we observe the performance of the best GNN models after the fine-tuning process, which exposes them to a small percentage of the dense ground truth this time. Each specific GNN model corresponds to the best performing models from the pretraining on the random set.

In the Dense Subset, the Weisfeiler-Lehman Kernel achieves the highest scores for MAP@10 and NDCG@10, followed by the Shortest Path and Neighborhood Hash Kernels, which also perform competitively well. Once again, the Graphlet Sampling and Random Walk Kernels perform quite poorly. It is evident once again that the Contrastive GNNs, trained on a random set, can generalize quite well on the dense set and perform competitively enough, even surpassing the best kernels by a small margin of 1% in NDCG@10, 3% in MRR@10, and 1% in MAP@10.

Table 9.4: Final Metric Scores for the Graph Kernels and the best pretrained GNN models without fine tuning, on the Original Dense Subset in an inductive setting. Training has been performed on a random subset. Bold denotes the best result for each architecture.

	MAP@10	**MRR@10**	**NDCG@10**
Shortest Path Kernel	0.1237	**0.4618**	0.6171
Random Walk Kernel	0.0928	0.3913	0.5964
Weisfeiler-Lehman Kernel	**0.1253**	0.4570	**0.6279**
Graphlet Sampling Kernel	0.0389	0.0965	0.5317
Neighborhood Hash Kernel	0.1034	0.3868	0.6132
GraphCL			
GCN	0.1281	0.4724	0.6230
GIN	**0.1339**	**0.4914**	**0.6302**
InfoGraph			
GIN	0.1354	0.4882	0.6282
Grace			
GAT	0.1152	0.4715	0.6285
GATv2	**0.1280**	**0.4902**	**0.6394**

Table 9.5: Final Metric Scores for the best GNN models with fine tuning, on the Original Dense Subset with supervision from a small subset of the ground truth. Bold denotes the best result for each metric.

	MAP@10		**MRR@10**		**NDCG@10**	
	10 epochs	**30 epochs**	**10 epochs**	**30 epochs**	**10 epochs**	**30 epochs**
GraphCL						
GIN	0.1670	0.1990	0.5292	0.5784	0.6713	0.7006
InfoGraph						
GIN	0.1605	0.2003	0.5322	0.5657	0.6735	0.7001
Grace						
GAT	0.1723	0.2111	0.5247	0.5688	0.6834	0.7073
GATv2	**0.1732**	**0.2207**	**0.5537**	**0.5915**	**0.6876**	**0.7140**

Once again, we can see that the Rank Aware Fine Tuning, using a small percentage of the ground truth of the original Dense Set, provides a significant boost across all three metrics as it is evident from Table 9.7. By utilizing a small percentage of the ground truth, we are able to achieve values of MAP@10, MRR@10, and NDCG@10 that are approximately 9-10% higher compared to the best results from the graph kernels with 30 epochs of fine-tuning. Moreover, 10 epochs are also enough in order for one to see a significant boost to the metrics. For the fine tuning, we selected for each framework the most promising GNN variant.

In order to justify our choice of first pretraining the GNNs contrastively on a random set and then fine-tuning them on the selected set, we provide Table 9.6, where one can observe the performance of the GNN

encoders used for fine-tuning. These GNN encoders were trained for only 30 epochs using the proposed Rank Aware loss, without contrastive pretraining. However, these models successfully outperform the graph kernels, albeit by a significant margin from the corresponding models that have been contrastively pretrained and then fine-tuned for the same number of epochs (e.g., 30 epochs), thus highlighting the benefits of our approach.

Table 9.6: Final Metric Scores for the best GNN models trained for 30 epochs only with the ranking loss. Inference is performed on the Original Dense Subset.

	MAP@10	**MRR@10**	**NDCG@10**
	30 epochs	**30 epochs**	**30 epochs**
GIN			
GIN	0.1704	0.5318	0.6773
GAT			
GAT	0.1624	0.5208	0.6739
GATv2			
GATv2	0.1651	0.5293	0.6732

It is safe to say that the GNN methods consistently outperform the kernel-based approaches and are able to achieve significant improvements. In conclusion, GNNs prove to be a highly effective solution for the Graph Similarity task. Additionally, it's important to note that the biggest advantage of GNNs over Graph Kernels is that they provide general-purpose embeddings that can be utilized for various downstream tasks.

Finally, we present the training times for solely contrastive pretraining on the random set and contrastive pretraining followed by fine tuning on the dense set.

Table 9.7: Final Training Times for the best GNN models with fine-tuning, on the Original Dense Subset with supervision from a small subset of the ground truth.

	Contrastive Pretraining Time	**Pretraining + Fine Tuning Time**	
		10 epochs	**30 epochs**
GraphCL			
GIN	32sec	44sec	1min 05sec
InfoGraph			
GIN	50sec	1min 04sec	1min 46sec
Grace			
GAT	2min 12sec	2min 50sec	4min 02sec
GATv2	4min 28sec	5min 36sec	6min 39sec

The higher training times for GAT variants is something we expected as the training of multiple attention heads requires additional time. Moreover, the Grace framework is computationally a bit heavier compared to the others as it performs the contrastive loss for every node in the batch (and not for every graph in the batch). Additionally, taking into account that we also use 4 % or 12 % of the ground truth, the computational trade-off is certainly in our favor, as we only need approximately 4% or 12% of the time required for total ground truth computation to significantly boost our metrics if we choose to fine-tune the models.

9.3.2 Qualitative Results

To qualitatively evaluate the proposed models, we should also incorporate the images corresponding to the scene graphs recommended by both the GNNs and Graph Kernels in response to specific queries. Specifically, we will display some query images along with the first few retrieved images in the dense set by our best model, Grace with GATv2 encoder fine-tuned for 30 epochs. Since our model does not have access to the images during training and relies solely on the corresponding scene graphs, there might be cases where the images are not similar, but the similarity of the graphs leads the model to rank them highly. Below, we provide pairs of query-answer examples to illustrate these scenarios.

Figure 9.3.1: Query Image

Figure 9.3.2: Top 3 Results - GNN Model



Figure 9.3.3: Top 3 Results - Best Kernel



Figure 9.3.4: Query Image

Figure 9.3.5: Top 3 Results - GNN Model



Figure 9.3.6: Top 3 Results - Best Kernel



Figure 9.3.7: Query Image

Figure 9.3.8: Top 3 Results - GNN Model

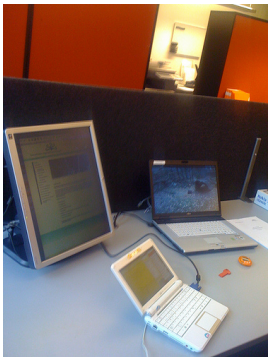


Figure 9.3.9: Top 3 Results - Best Kernel

Figure 9.3.10: Query Image

Figure 9.3.11: Top 3 Results - GNN Model

Figure 9.3.12: Top 3 Results - Best Kernel

These qualitative results affirm the superiority of our GNN model, which has also been demonstrated quantitatively. It appears that our model can retrieve images with closer semantic content than the best kernels in various cases.

Chapter 10

Conclusion

10.1 Discussion

In this thesis, we addressed the challenge of Graph Similarity through the lens of Contrastive Learning. Our experiments aimed to evaluate the effectiveness of proposed Contrastive Graph Neural Network (GNN) models compared to baseline Graph Kernels, focusing on graph similarity tasks using the Visual Genome dataset. Through a systematic exploration of different GNN architectures and contrastive learning frameworks, accompanied by thorough evaluation using metrics such as Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (NDCG), several significant insights were gained.

Initially, the superiority of certain Graph Kernels, such as the Shortest Path Kernel, Neighborhood Hash Kernel and Weisfeiler-Lehman was evident. However, limitations were observed with kernels relying heavily on structural properties, such as the Random Walk Kernel and Graphlet Sampling Kernel, suggesting their inadequacy in capturing the intricacies of scene graphs.

Contrastive GNN models demonstrated competitiveness against various Graph Kernels, particularly attentional variants such as GAT and GATv2, especially when contrastive loss was applied at the node embedding level (Grace Framework). Conversely, the Generalized-Isomorphism Network (GIN) variant exhibited remarkable competitiveness when graph embeddings were integrated into the contrastive loss (GraphCL and InfoGraph framework). Our exploration encompassed four distinct convolutional Graph Neural Network (GNN) variations for each of the GraphCL, InfoGraph, and Grace contrastive approaches, evaluating their effectiveness, complexity, and expressive capacities. Specifically, we employed GCN, GIN, GAT, and GATv2. These GNN variants facilitated graph embedding in a space capable of capturing both structural and semantic similarities among graphs. Subsequently, these embeddings were utilized for graph comparison, facilitating the identification of the most similar pair for each graph. To establish a ground truth, we utilized an approximate Graph Edit Distance algorithm rooted in bipartite graph matching, complemented by hierarchical semantic information extracted from WordNet and path similarity algorithms applied to this hierarchy. Our comprehensive analysis provided significant insights into the quality and expressive capabilities of all the aforementioned methods.

Furthermore, the introduction of Rank Aware Fine Tuning, utilizing a small percentage of ground truth data, significantly boosted performance across all metrics. This fine-tuning process showcased the ability of GNN models to generalize well, even surpassing the performance of the best Graph Kernels on both sparse and dense subsets of the dataset.

Comparisons between contrastively pretrained GNN models and those without contrastive pretraining underscored the efficacy of the proposed approach, with contrastive pretraining followed by fine-tuning yielding superior results.

In summary, the experiments firmly establish the efficacy of Contrastive GNN techniques in the graph similarity task, consistently surpassing traditional kernel-based methods. Additionally, the adaptability of GNN

embeddings creates opportunities for utilizing these models across different downstream tasks, highlighting their usefulness beyond solely graph similarity tasks. Therefore, this research offers valuable perspectives on the utilization of Contrastive GNNs in tasks related to graphs.

Ultimately, the primary objective of training these GNN models and assessing them in the context of Information Retrieval tasks is to integrate them into a Counterfactual-Explanations Framework. Specifically, identifying the most similar pair of images allows for the computation of only some specific edit distances, rather than calculating edit distances for every pair of the dataset. As a result, the proposed method, which utilizes the corresponding scene graph of an image for retrieving the optimal pair, introduces a significant acceleration in this process. This framework incorporates external knowledge to provide explanations for samples within any dataset. If this external knowledge is structured in graph formats, then GNN models, akin to those examined in this thesis, can be deployed to enhance performance and reduce computation times.

10.2 Future Work

For future work, several paths could be explored to further enhance the understanding and application of Contrastive Graph Neural Networks (GNNs) in the domain of graph similarity and beyond:

1. **Exploration of Additional Contrastive Learning Techniques**: This thesis focuses on Contrastive Graph Neural Networks for graph similarity tasks. In this context, exploring alternative contrastive learning techniques could offer valuable insights. Variants such as those based on Barlow Twins Loss [10] or Bootstrapped Graph Latents [79], among others, have demonstrated promising results in graph data, warranting further investigation into their performance for the task of scene graph retrieval.
2. **Integration of Edge Attributes in the aggregation functions**: In addition to the explored avenues, incorporating edge attributes into the graph representation could further enrich the expressive power of Contrastive Graph Neural Networks (GNNs). Edge attributes contain valuable information about relationships between nodes in a graph, which can significantly enhance the model's ability to capture semantic and structural similarities.
3. **Fine-tuning Strategies**: Further exploration of fine-tuning strategies could yield significant benefits. This includes experimenting with various percentages of ground truth data for fine-tuning or investigating alternative approaches for rank-aware fine-tuning, which could enhance model performance and generalization across varied datasets. Additionally, experiments should be conducted using either the same or different rank-aware losses with the goal of making the models capable of performing equally well or even better in an inductive setting.

By pursuing these paths for future work, researchers can continue to advance the understanding and practical applications of Contrastive GNN techniques, ultimately contributing to the broader field of graph representation learning in scene graphs and its diverse applications.

Chapter 11

Bibliography

[1] Alirezaie, M. et al. "A symbolic approach for explaining errors in image classification tasks". In: *Working Papers and Documents of the IJCAI-ECAI-2018 Workshop on*. 2018.

[2] Alvarez-Gonzalez, N., Kaltenbrunner, A., and Gómez, V. "Beyond Weisfeiler Lehman with Local Ego-Network Encodings". In: *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1234–1265. ISSN: 2504-4990. DOI: 10.3390/make5040063. URL:

[3] Arrieta, A. B. et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[4] Bai, Y. et al. "Simgnn: A neural network approach to fast graph similarity computation". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 2019, pp. 384–392.

[5] Bai, Y. et al. "Unsupervised inductive graph-level representation learning via graph-graph proximity". In: *arXiv preprint arXiv:1904.01098* (2019).

[6] Bajaj, M. et al. "Robust Counterfactual Explanations on Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 5644–5655. URL:

[7] Al-Behadili, H. N. K., Ku-Mahamud, K. R., and Sagban, R. "Rule pruning techniques in the ant-miner classification algorithm and its variants: A review". In: *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*. 2018, pp. 78–84. DOI: 10.1109/ISCAIE.2018.8405448.

[8] Belghazi, M. I. et al. *MINE: Mutual Information Neural Estimation*. 2021. arXiv: 1801.04062 [cs.LG].

[9] Berretti, S., Del Bimbo, A., and Vicario, E. "Efficient matching and indexing of graph models in content-based retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.10 (2001), pp. 1089–1105.

[10] Bielak, P., Kajdanowicz, T., and Chawla, N. V. "Graph Barlow Twins: A self-supervised representation learning framework for graphs". In: *Knowledge-Based Systems* 256 (2022). ISSN: 0950-7051. DOI: 10.1016/j.knosys.2022.109631. URL:

[11] Bishop, C. M. and Bishop, H. *Deep Learning - Foundations and Concepts*. Ed. by S. Cham. 1st ed. 2023. ISBN: 978-3-031-45468-4. DOI: https://doi.org/10.1007/978-3-031-45468-4.

[12] Bo, D. et al. *A Survey on Spectral Graph Neural Networks*. 2023. arXiv: 2302.05631 [cs.LG].

[13] Borgwardt, K. and Kriegel, H. "Shortest-path kernels on graphs". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. 2005, 8 pp.-. DOI: 10.1109/ICDM.2005.132.

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

[15] Bronstein, M. M. et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[16] Bronstein, M. M. et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. arXiv: 2104.13478 [cs.LG].

[17] Bruna, J. et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[18] Bunke, H. and Shearer, K. "A graph distance metric based on the maximal common subgraph". In: *Pattern recognition letters* 19.3-4 (1998), pp. 255–259.

[19] Burges, C. et al. "Learning to rank using gradient descent". In: *Proceedings of the 22nd international conference on Machine learning*. ICML '05. Bonn, Germany: ACM, 2005, pp. 89–96. ISBN: 1-59593-180-5. DOI: 10.1145/1102351.1102363. URL:

[20] Chen, T. et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG].

[21] Chopra, S., Hadsell, R., and LeCun, Y. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1 (2005), 539–546 vol. 1. URL:

[22] Defferrard, M., Bresson, X., and Vandergheynst, P. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in neural information processing systems* 29 (2016).

[23] Dervakos, E. et al. *Choose your Data Wisely: A Framework for Semantic Counterfactuals*. 2023. arXiv: 2305.17667 [cs.AI].

[24] Dimitriou, A. et al. *Graph Edits for Counterfactual Explanations: A comparative study*. 2024. arXiv: 2401.11609 [cs.LG].

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

[26] Dong, X. et al. "Graph Signal Processing for Machine Learning: A Review and New Perspectives". In: *IEEE Signal Processing Magazine* 37.6 (Nov. 2020), pp. 117–127. ISSN: 1558-0792. DOI: 10.1109/msp.2020.3014591. URL:

[27] Donsker, M. D. and Varadhan, S. R. S. "Asymptotic evaluation of certain markov process expectations for large time. IV". In: *Communications on Pure and Applied Mathematics* 36.2 (1983), pp. 183–212. DOI: https://doi.org/10.1002/cpa.3160360204. eprint: URL:

[28] Fankhauser, S., Riesen, K., and Bunke, H. "Speeding Up Graph Edit Distance Computation through Fast Bipartite Matching". In: *Graph-Based Representations in Pattern Recognition*. Ed. by X. Jiang, M. Ferrer, and A. Torsello. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 102–111. ISBN: 978-3-642-20844-7.

[29] Fankhauser, S., Riesen, K., and Bunke, H. "Speeding up graph edit distance computation through fast bipartite matching". In: *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer. 2011, pp. 102–111.

[30] Fey, M. and Lenssen, J. E. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[31] Filandrianos, G. et al. "Conceptual Edits as Counterfactual Explanations." In: *AAAI Spring Symposium: MAKE*. 2022.

[32] Fischer, A. et al. "A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance". In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by P. Fränti et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 83–92. ISBN: 978-3-662-44415-3.

[33] Gartner, T., Flach, P., and Wrobel, S. "On Graph Kernels: Hardness Results and Efficient Alternatives". In: *Learning Theory and Kernel Machines*. Ed. by B. Scholkopf and M. K. Warmuth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 129–143. ISBN: 978-3-540-45167-9.

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

[35] Goyal, Y. et al. "Counterfactual Visual Explanations". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 2376–2384.

[36] Hamilton, W., Ying, Z., and Leskovec, J. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[37] Hamming, R. W. "Error detecting and error correcting codes". In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.

[38] Hido, S. and Kashima, H. "A Linear-Time Graph Kernel". In: *2009 Ninth IEEE International Conference on Data Mining*. 2009, pp. 179–188. DOI: 10.1109/ICDM.2009.30.

[39] Hjelm, R. D. et al. *Learning deep representations by mutual information estimation and maximization*. 2019. arXiv: 1808.06670 [stat.ML].

[40] Johnson, J. et al. "Image retrieval using scene graphs". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3668–3678. DOI: 10.1109/CVPR.2015.7298990.

[41] Jonker, R. and Volgenant, A. "A shortest augmenting path algorithm for dense and sparse linear assignment problems". In: *Computing* 38.4 (1987), pp. 325–340.

[42] Karp, R. *An Algorithm to Solve the mxn Assignment Problem in Expected Time O (mn log n)*. Tech. rep. UCB/ERL M78/67. EECS Department, University of California, Berkeley, Sept. 1978. URL:
[43] Kingma, D. P. and Welling, M. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].
[44] Kipf, T. N. and Welling, M. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).
[45] Kipf, T. N. and Welling, M. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).
[46] Krishna, R. et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV].
[47] Krishna, R. et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *International journal of computer vision* 123.1 (2017), pp. 32–73.
[48] Kullback, S. and Leibler, R. A. "On Information and Sufficiency". In: *Ann. Math. Statist.* 22.1 (1951), pp. 79–86.
[49] LeCun, Y. et al. "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.
[50] Li, Y. et al. "Graph matching networks for learning the similarity of graph structured objects". In: *International conference on machine learning*. PMLR. 2019, pp. 3835–3845.
[51] Lin, J. "Divergence measures based on the Shannon entropy". In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 145–151. DOI: 10.1109/18.61115.
[52] Lin, T.-Y. et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].
[53] Loper, E. and Bird, S. *NLTK: The Natural Language Toolkit*. 2002. arXiv: cs/0205028 [cs.CL].
[54] Lopez-Bernal, D. et al. "Education 4.0: Teaching the Basics of KNN, LDA and Simple Perceptron Algorithms for Binary Classification Problems". In: *Future Internet* 13 (July 2021), p. 193. DOI: 10.3390/fi13080193.
[55] Lucic, A. et al. "Cf-gnnexplainer: Counterfactual explanations for graph neural networks". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 4499–4511.
[56] Maheshwari, P., Chaudhry, R., and Vinay, V. "Scene graph embeddings using relative similarity supervision". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 3. 2021, pp. 2328–2336.
[57] Maron, H. et al. "Invariant and equivariant graph networks". In: *arXiv preprint arXiv:1812.09902* (2018).
[58] Mikolov, T. et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).
[59] Miller, G. A. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.
[60] 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.
[61] Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. "Graph Kernels: A Survey". In: *Journal of Artificial Intelligence Research* 72 (Nov. 2021), pp. 943–1027. ISSN: 1076-9757. DOI: 10.1613/jair.1.13225. URL:
[62] Oord, A. van den, Li, Y., and Vinyals, O. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].
[63] Ortega, A. et al. *Graph Signal Processing: Overview, Challenges and Applications*. 2018. arXiv: 1712.00468 [eess.SP].
[64] Peng, J. et al. "Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges". In: *Frontiers in Pharmacology* 12 (Sept. 2021). DOI: 10.3389/fphar.2021.720694.
[65] Pennington, J., Socher, R., and Manning, C. D. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL:
[66] Poole, B. et al. *On Variational Bounds of Mutual Information*. 2019. arXiv: 1905.06922 [cs.LG].
[67] Pržulj, N. "Biological network comparison using graphlet degree distribution". In: *Bioinformatics* 23.2 (Jan. 2007), e177–e183. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl301. URL:
[68] Riesen, K., Fankhauser, S., and Bunke, H. "Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic." In: Jan. 2007.

[69] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.
[70] Sanfeliu, A. and Fu, K.-S. "A distance measure between attributed relational graphs for pattern recognition". In: *IEEE transactions on systems, man, and cybernetics* 3 (1983), pp. 353–362.
[71] Schroff, F., Kalenichenko, D., and Philbin, J. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298682. URL:
[72] Shannon, C. E. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.
[73] Shervashidze, N. et al. "Efficient graphlet kernels for large graph comparison". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by D. van Dyk and M. Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 488–495. URL:
[74] Shervashidze, N. et al. "Weisfeiler-lehman graph kernels." In: *Journal of Machine Learning Research* 12.9 (2011).
[75] Siglidis, G. et al. "GraKeL: A Graph Kernel Library in Python." In: *J. Mach. Learn. Res.* 21.54 (2020), pp. 1–5.
[76] Silva, V. S., Freitas, A., and Handschuh, S. "Exploring knowledge graphs in an interpretable composite approach for text entailment". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7023–7030.
[77] Sun, F.-Y. et al. *InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization*. 2020. arXiv: 1908.01000 [cs.LG].
[78] Takami, S. and Inokuchi, A. "Accurate and Fast Computation of Approximate Graph Edit Distance based on Graph Relabeling". In: *International Conference on Pattern Recognition Applications and Methods*. 2018. URL:
[79] Thakoor, S. et al. *Large-Scale Representation Learning on Graphs via Bootstrapping*. 2023. arXiv: 2102.06514 [cs.LG].
[80] *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd ed. Cambridge University Press, 2007.
[81] Vandenhende, S. et al. *Making Heads or Tails: Towards Semantically Consistent Visual Counterfactuals*. 2022. arXiv: 2203.12892 [cs.CV].
[82] Vaswani, A. et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
[83] Veličković, P. et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).
[84] Veličković, P. et al. *Deep Graph Infomax*. 2018. arXiv: 1809.10341 [stat.ML].
[85] Vishwanathan, S. V. N. et al. *Graph Kernels*. 2008. arXiv: 0807.0093 [cs.LG].
[86] Wang, M. et al. "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". In: *arXiv preprint arXiv:1909.01315* (2019).
[87] Weisfeiler, B. and Leman, A. "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI, Series* 2.9 (1968), pp. 12–16.
[88] Wu, Z. et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.
[89] Xu, K. et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).
[90] You, Y. et al. *Graph Contrastive Learning with Augmentations*. 2021. arXiv: 2010.13902 [cs.LG].
[91] Zaheer, M. et al. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG].
[92] Zhang, A. et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).
[93] Zhang, C., Chao, W.-L., and Xuan, D. *An Empirical Study on Leveraging Scene Graphs for Visual Question Answering*. 2019. arXiv: 1907.12133 [cs.CV].
[94] Zhang, K. "A constrained edit distance between unordered labeled trees". In: *Algorithmica* 15.3 (1996), pp. 205–222.
[95] Zhu, Y. et al. *Deep Graph Contrastive Representation Learning*. 2020. arXiv: 2006.04131 [cs.LG].
[96] Zhu, Y. et al. *An Empirical Study of Graph Contrastive Learning*. 2021. arXiv: 2109.01116 [cs.LG].