Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Unsupervised Scene Graph Retrieval using Graph Autoencoders

Diploma Thesis

by

Nikolaos Chaidos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2023

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

Unsupervised Scene Graph Retrieval using Graph Autoencoders

Diploma Thesis

by

Nikolaos Chaidos

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Οκτωβρίου, 2023.

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

Αθήνα, Οκτώβριος 2023

.....

Νικολαος Χαΐδος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα Νευρωνικά Δίκτυα Γράφων (ΝΔΓ) έχουν αναδειχθεί ως ένα βασικό μοντέλο στον τομέα της μηχανικής μάθησης, λόγω της μοναδικής τους ικανότητας να χειρίζονται δεδομένα δομημένα ως γραφήματα. Σε αντίθεση με τις παραδοσιακές αναπαραστάσεις δεδομένων, οι γράφοι αποτυπώνουν περίπλοκες σχέσεις και εξαρτήσεις μεταξύ των οντοτήτων, κάτι που εντοπίζεται συχνά σε ένα ευρύ φάσμα τομέων, όπως τα κοινωνικά δίκτυα, η μοριακή χημεία, τα συστήματα συστάσεων και άλλα. Τα ΝΔΓ υπερέχουν στη μοντελοποίηση αυτών των πολύπλοκων αλληλεπιδράσεων, επιτρέποντας αυξημένη ακρίβεια πρόβλεψης και καλύτερη κατανόηση των διαφόρων δομών. Ένα από τα πιο σημαντικά και παραδοσιακά προβλήματα στην θεωρία των γράφων, είναι η Ομοιότητα Γραφημάτων, δηλαδή η ποσοτικοποίηση της σύγκρισης δύο γράφων. Για την επίλυση αυτού του προβλήματος, χρησιμοποιούνται προσεγγιστικοί αλγόριθμοι εύρεσης Απόστασης Επεξεργασίας Γράφων, αλλά και πιο κλασσικές μέθοδοι, όπως Πυρήνες Γράφων. Επιπλέον, τα ΝΔΓ μπορούν να χρησιμοποιηθούν για την παροχή Εξηγήσεων μέσω Αντιπαραδειγμάτων, ερμηνεύοντας έτσι τις διαφορές μεταξύ των κατηγοριών του συνόλου δεδομένων. Με τους ανακτηθέντες γράφους, έχουμε πρόσβαση στο αντικείμενο που αντιπροσωπεύει το κύριο Αντιπαράδειγμα, το οποίο είναι ο πρώτος ανακτηθείς γράφος που δεν ανήκει στην ίδια κλάση με τον γράφο του ερωτήματος. Με τον γράφο Αντιπαράδειγμα, μπορούμε να εξάγουμε τις ουσιαστικές αλλαγές που πρέπει να γίνουν στο γράφημα, προκειμένου να βρούμε τις ελάχιστες διαφορές των κλάσεων των αντικειμένων.

Σε αυτήν την διατριβή, θα αντιμετωπίσουμε το πρόβλημα της Ομοιότητας Γραφημάτων, χρησιμοποιώντας Νευρωνικά Δίκτυα Γράφων. Το συνολό δεδομένων αποτελείται από Γράφους Σκηνής, δηλαδή γράφους οι οποίοι απεικονίζουν σημασιολογικά μία αντίστοιχη εικόνα. Κωδικοποιώντας τις δομές των γραφημάτων σε υψηλής διάστασης ενθετικές αναπαραστάσεις, τα ΝΔΓ ποσοτικοποιούν την ομοιότητα μεταξύ των γραφημάτων, επιτρέποντας την αποτελεσματική αντιστοίχιση γραφημάτων αλλά και γενικότερες εργασίες συσταδοποίησης. Η ικανότητά τους να προσαρμόζονται και να γενικεύονται σε διαφορετικούς τομείς γράφων καθιστά τα ΝΔΓ μια ευέλικτη και πολλά υποσχόμενη προσέγγιση για την αντιμετώπιση του προβλήματος της Ομοιότητας Γραφημάτων, προσφέροντας νέες ιδέες και λύσεις σε τομείς όπου η κατανόηση των υποκείμενων σχέσεων μέσα σε πολύπλοκα δίκτυα είναι απαραίτητη. Συγκεκριμένα, θα χρησιμοποιηθεί μία υπό-κατηγορία μη-επιβλεπόμενων ΝΔΓ, οι Αυτο-κωδικοποιητές Γράφων, οι οποίοι έχουν ως σκοπό την επιτυχής ανακατασκευή των γράφων εισόδου, ενσωματώνοντάς τους σε χώρους υψηλότερων διαστάσεων. Το κύριο πλεονέκτημα των αρχιτεκτονικών που βασίζονται σε Αυτο-κωδικοποιητές είναι η επιτάχυνση της εκπαίδευσης των μοντέλων σε σύγκριση με τις προσεγγίσεις που υλοποιούν Επιβλεπόμενη Μάθηση. Δοκιμάζονται αρκετές παραλλαγές του βασικού Αυτο-κωδικοποιητή Γράφων, και συγκρίνονται με τους πέντε επικρατέστερους Πυρήνες Γράφων. Ο σκοπός όλων των παραπάνω μοντέλων είναι να προσεγγίσουν τα αποτελέσματα που δίνει ο αλγόριθμος εύρεσης Απόστασης Επεξεργασίας Γράφων, ο οποίος είναι αρκετά δύσκολος στον υπολογισμό, και πολύ πιο χρονοβόρος. Η αξιολόγηση των αποτελεσμάτων γίνεται τόσο σε ποσοτικό επίπεδο, με χρήση τριών μετρικών για την αξιολόγηση της ανάκτησης γράφων, όσο και σε ποιοτικό επίπεδο, με την απεικόνιση των εικόνων που αντιστοιχούν στους γράφους του συνόλου δεδομένων. Εν τέλει, οι Αυτο-κωδικοποιητές Γράφων έχουν την ικανότητα να προσεγγίσουν καλύτερα τον αλγόριθμο εύρεσης Απόστασης Επεξεργασίας Γραφήματος, σε σύγκριση με τους Πυρήνες Γράφων, ενώ παράλληλα παράγουν και ενθετικές αναπαραστάσεις των γραφημάτων, οι οποίες μπορούν να χρησιμεύσουν σε αρκετά ακόμα προβλήματα.

Λέξεις-κλειδιά — Νευρωνικά Δίκτυα Γράφων, Εξηγήσεις με Αντιπαραδείγματα, Ομοιότητα Γραφημάτων, Αυτο-κωδικοποιητές Γράφων, Γράφοι Σκηνής, Πυρήνες Γράφων

Abstract

Graph Neural Networks (GNNs) have emerged as a key model in the field of machine learning because of their unique ability to handle data structured as graphs. Unlike traditional data representations, graphs capture complex relationships and dependencies between entities, which is often found in a wide range of fields such as social networks, molecular chemistry, recommender systems and others. GNNs excel at modeling these complex interactions, allowing for increased prediction accuracy and better understanding of various structures. One of the most important and traditional problems in graph theory is Graph Similarity, i.e., quantifying the comparison of two graphs. To solve this problem, approximate algorithms for finding Graph Edit Distance (GED) are used, as well as more classical methods such as Graph Kernels. Furthermore, GNNs can be used to provide Counterfactual Explanations, therefore interpreting the differences among the several classes of the dataset. With the retrieved graphs, we have access to the Counterfactual Object, which is the first retrieved graph that doesn't belong to the same class as the query graph. With the Counterfactual Object, we can extract meaningful changes needed to be made to the graph, in order to find the minimum differences of the object classes.

In this thesis, we will tackle the Graph Similarity problem using Graph Neural Networks. The dataset consists of Scene Graphs, i.e., graphs which semantically represent a corresponding image. By encoding graph structures into high-dimensional embeddings, GNNs allow quantifying the similarity between graphs, enabling efficient graph matching and clustering tasks. Their ability to adapt and generalize to different graph domains makes GNNs a flexible and promising approach to address the problem of Graph Similarity, offering new insights and solutions in areas where understanding the underlying relationships within complex networks is essential. In particular, we will use a sub-class of unsupervised GNNs, called Graph Auto-encoders, which aim to successfully reconstruct input graphs by embedding them in higher dimensional spaces. The main advantage of architectures based on Autoencoders, is the speed-up of the Training of the models, when compared to supervised approaches. Several variants of the basic Graph Auto-encoder are tested, and compared with the five predominant Graph Kernels. The purpose of all the above models is to approximate the results given by the Graph Edit Distance algorithm, which is quite difficult to compute, and much more time consuming. The results are evaluated both at quantitative level, using three metrics to evaluate graph retrieval, and at qualitative level, by plotting the images corresponding to the graphs in the dataset. Ultimately, Graph Auto-encoders have the ability to better approximate the GED algorithm, compared to Graph Kernels, while also producing graph embeddings, which can be useful in several other problems.

Keywords — Graph Neural Networks, Counterfactual Explanations, Graph Similarity, Graph Auto-encoders, Scene Graphs, Graph Kernels

Ευχαριστίες

Η περάτωση της συγκεκριμένης διπλωματικής εργασίας δεν θα ήταν δυνατή χωρίς την βοήθεια πολλών ανθρώπων. Είμαι ευγνώμων στον επιβλέποντά μου, κ. Γεώργιο Στάμου, του οποίου η καθοδήγηση και η ενθάρρυνση συνέβαλαν καθοριστικά στη διαμόρφωση αυτής της έρευνας. Ευχαριστώ όλους τους υποψήφιους Διδάκτορες του εργαστήριου AILS. Ιδιαίτερα όμως, ευχαριστώ από καρδιάς τις Αγγελική Δημητρίου και Μαρία Λυμπεραίου. Η αφοσίωσή τους στη διασφάλιση της ποιότητας της εργασίας μου, με ενέπνευσαν να εργάζομαι σκληρότερα και να βελτιώνομαι συνεχώς. Επίσης, εξίσου σημαντική ήταν και η ψυχολογική υποστήριξη που έλαβα από τους γονείς μου, τον αδελφό μου και τους φίλους μου, καθ'όλη την διάρκεια των σπουδών μου. Αναφερόμενος στους τελευταίους, θα ήθελα να ευχαριστήσω τον Παναγιώτη, τον Νίκο, τον Πέτρο και τον Κώστα, με τους οποίους μοιράστηκα τις χαρές και τις προκλήσεις, αλλά στο τέλος πετύχαμε τους στόχους μας.

Νικόλαος Χάιδος, Οκτώβρης 2023

Contents

Contents xi

List of Figures xiii

1 Εκτεταμένη Περίληψη στα Ελληνικά 1
- 1.1 Θεωρητικό Υπόβαθρο . . . 2
  - 1.1.1 Γράφοι . . . 2
  - 1.1.2 Απόσταση Επεξεργασίας Γράφων . . . 2
  - 1.1.3 Πυρήνες Γράφων . . . 3
  - 1.1.4 Γράφοι Σκηνής . . . 4
  - 1.1.5 Νευρωνικά Δίκτυα Γράφων . . . 4
  - 1.1.6 Εξηγήσεις με Αντιπαραδείγματα . . . 9
- 1.2 Προτεινόμενα Μοντέλα . . . 9
  - 1.2.1 Συσεισφορά . . . 9
  - 1.2.2 Μοντέλα Αυτο-κωδικοποιητών Γράφων . . . 10
- 1.3 Πειραματικό Μέρος . . . 14
  - 1.3.1 Σύνολο Δεδομένων . . . 14
  - 1.3.2 Βασική Αλήθεια . . . 16
  - 1.3.3 Μετρικές . . . 16
  - 1.3.4 Λεπτομέρειες Μοντέλων . . . 17
  - 1.3.5 Αποτελέσματα . . . 19
- 1.4 Συμπεράσματα . . . 26
  - 1.4.1 Μελλοντικές Κατευθύνσεις . . . 28

2 Introduction 29

3 Machine Learning 31
- 3.1 Data Modalities . . . 32
- 3.2 Machine Learning Types . . . 32
- 3.3 Definitions . . . 33
- 3.4 Deep Learning . . . 37
- 3.5 Autoencoders . . . 48

4 Graphs 53
- 4.1 Definitions . . . 54
- 4.2 Graph Similarity . . . 57
  - 4.2.1 Graph Edit Distance . . . 58
  - 4.2.2 Graph Kernels . . . 59
- 4.3 Scene Graphs . . . 64
- 4.4 Related Work . . . 66

5 Graph Neural Networks (GNN) 69
- 5.1 Introduction . . . 70

5.2 Graph Convolution . . . 71
5.2.1 Spectral-Based Graph Convolution . . . 71
5.2.2 Spatial-based Graph Convolution . . . 72
5.3 Spatial-Based GNN Modules . . . 74
5.4 Graph Auto-encoders . . . 76

6 Counterfactual Explanations **79**
6.1 Introduction . . . 80
6.2 Definitions and Framework . . . 80
6.2.1 Algorithmic Implementation . . . 82
6.2.2 Enriching the Explanation Dataset . . . 83
6.2.3 Utilization of GNN Models . . . 83
6.3 Related Work . . . 84

7 Proposal **87**
7.1 Contributions . . . 87
7.2 Proposed Models . . . 87

8 Experiments **93**
8.1 Preliminaries . . . 94
8.1.1 Dataset . . . 94
8.1.2 Ground Truth . . . 98
8.1.3 Evaluation Metrics . . . 99
8.2 Training and Inference Details . . . 103
8.2.1 Graph Kernel Parameters . . . 103
8.2.2 GNN Details and Hyperparameters . . . 103
8.3 Results . . . 107
8.3.1 Quantitative Analysis . . . 107
8.3.2 Qualitative Analysis . . . 110

9 Conclusion **113**
9.1 Reflecting on the Findings . . . 113
9.2 Future Research . . . 114

10 Bibliography **117**

List of Figures

1.1.1 Παράδειγμα ενός μη-κατευθυνόμενου (αριστερά) και ενός κατευθυνόμενου (δεξιά) γράφου[67] . 2
1.1.2 Οι Πυρήνες Γράφων, μετατρέπουν τα γραφήματα σε σημεία ενός χώρου Hilbert υψηλότερης διάστασης. Αυτό μας επιτρέπει να χρησιμοποιούμε τα παραδοσιακά μέτρα ομοιότητας, συνηθέστερα το εσωτερικό γινόμενο.[64] . . . 3
1.1.3 Ένα παράδειγμα ενός Γράφου Σκηνής, μαζί με την αρχική του εικόνα. Αυτός ο συγκεκριμένος γράφος σκηνής, περιέχει αντικείμενα, σχέσεις και χαρακτηριστικά, αλλά πρέπει να σημειωθεί ότι δεν υπάρχει κάποιος αυστηρός ορισμός που να απαιτεί όλοι οι γράφοι σκηνής να ακολουθούν ένα συγκεκριμένο μοτίβο. Μόνο ότι πρέπει έχουν τη μορφή γράφου και ότι παρέχουν σημασιολογικές πληροφορίες για τα αντικείμενα της εικόνας και τις σχέσεις μεταξύ τους.[122] . . . 5
1.1.4 Η βασική αρχιτεκτονική του ΑΚΓ, που αποτελείται από ένα Κωδικοποιητή και έναν Αποκωδικοποιητή [120] . . . 8
1.1.5 Αρχιτεκτονική του μοντέλου ARVGA, όπως παρουσιάστηκε στο [84] . . . 8
1.2.1 Χρήση Αυτο-κωδικοποιητών Γράφων για την φάση της Εκπαίδευσης. . . . 10
1.2.2 Χρήση Αυτο-κωδικοποιητών Γράφων για την φάση τους Συμπερασμού. . . . 10
1.2.3 Αρχική αρχιτεκτονική του *VGAE* . . . 11
1.2.4 Αρχιτεκτονική του *ARVGA* . . . 12
1.2.5 Αρχιτεκτονική *Feature VAGE* . . . 12
1.2.6 Αρχιτεκτονική *Combined VGAE* . . . 13
1.2.7 Αρχιτεκτονική *Combined ARVGA* . . . 13
1.2.8 Αρχιτεκτονική *Combined ARVGA MLP* . . . 14
1.3.1 Παράδειγμα ενός Γράφου Σκηνής, μαζί με την αντίστοιχη εικόνα που περιγράφει. . . . 14
1.3.2 Παραδείγματα διαφορετικών ερωτημάτων (αριστερά), με τα 3 κορυφαία αποτελέσματα (δεξιά), όπως ανακτήθηκαν από το καλύτερο μοντέλο ΝΔΓ . . . 24
1.3.3 Δύο παραδείγματα ερωτημάτων, όπου τα 2 πρώτα αντικείμενα που ανακτώνται από το καλύτερο μοντέλο ΝΔΓ (πάνω) είναι οπτικά πιο κοντά στην εικόνα του ερωτήματος (αριστερά), σε σύγκριση με τα 2 πρώτα αντικείμενα που ανακτώνται από τον καλύτερο Πυρήνα Γραφημάτων (κάτω). . . . 25
1.3.4 Παράδειγμα όπου το ερώτημα (αριστερά) και το πρώτο ανακτηθέν αντικείμενο (δεξιά) σύμφωνα με το καλύτερο μοντέλο ΝΔΓ, έχουν δομικά παρόμοιους γράφους σκηνής, αλλά αρκετά ανόμοιες εικόνες. . . . 26

3.2.1 Machine Learning Types and common use cases. . . . 33
3.3.1 Confusion Matrix for a Binary Classification Task . . . 36
3.4.1 Single-Layer Neural Network which can only learn Linear functions [11] . . . 38
3.4.2 Deep Neural Network, with multiple hidden layers, theoretically able to learn any function defined on the network's inputs [76] . . . 38
3.4.3 Most commonly used Activation Functions . . . 41
3.4.4 Original LeNet Architecture [62] . . . 42
3.4.5 Convolution Operation in a CNN [17] . . . 43
3.4.6 RNN Unfolding process, which shows the way that sequential data are processed [46] . . . 45
3.4.7 LSTM Cell Architecture [26] . . . 45
3.4.8 The original Transformer Architecture, as presented in [112] . . . 47
3.5.1 Architecture of the Autoencoder Network [24] . . . 49

3.5.2 An example where the manifold hypothesis holds true. The data, although originally in a 3-dimensional space, can be successfully embedded in a 2-dimensional latent space, without any loss of information. 50

4.1.1 Example of a Directed and an Undirected Graph [67] 54
4.1.2 Example of Construction of Adjacency Matrix from an Undirected Graph [124] 55
4.1.3 Example of a Path in a Weighted, Directed Graph [25] 56
4.1.4 Example of a Heterogeneous Graph, along with meta-paths defined on it [85] 57
4.2.1 An example where, using a kernel function, we are able to transform the data to a higher-dimensional space, and convert the classification problem into a Linearly-Separable one[74] . 61
4.2.2 Graph Kernels, transform the graphs to data points, in a higher-dimensional Hilbert space. This allows us to use the traditional similarity measures, most commonly the dot-product.[64] 62
4.3.1 Examples of the sparsity and variability of visual relationships. Being able to extract useful semantic information from these relationships, can be immensely useful for all kinds of visual tasks, that require a deeper understanding of the objects and their actions.[13] 64
4.3.2 An example of a Scene Graph, alongside its original image. This specific scene graph, contains objects, relationships and attributes, but it should be noted that there is no strict definition of a schema that all the Scene Graphs are required to follow. Only that they are in the form of graph, and that they provide semantic information of the objects in the image, and the relationships between them.[122] 65

5.1.1 The structured nature of the data points on the left, allows us to treat it as data in a Euclidean Space. On the contrary, in the case of the random graph on the right, we can't define a distance between the nodes, that allows us to obey the Euclidean Distance rules.[2] 70
5.2.1 Visualization of Spectral Graph Convolution[65] 72
5.3.1 On the left, we can see that the attention maps of *GAT* compute a global ranking of all attention weights. On the right, *GATv2* computes the correct attention weights, visualized by the characteristic diagonal in the attention matrix (because every node should attend the most with itself) [8] 75
5.4.1 The base architecture of a Graph Autoencoder, consisting of the Encoder and Decoder [120] . 78
5.4.2 Architecture of the ARVGA model, as presented in [84] 78

6.2.1 Framework Architecture for providing Counterfactual Explanations [29] 82

7.2.1 Graph Autoencoder utilization, for the Training phase. 88
7.2.2 Graph Autoencoder utilization, for the Inference phase. 88
7.2.3 Original architecture, for Variational Graph Autoencoder 89
7.2.4 *Adversarially Reguralized Graph Autoencoder* architecture 90
7.2.5 *Feature Graph Autoencoder* architecture 90
7.2.6 *Combined VGAE* architecture 91
7.2.7 *Combined ARVGA* architecture 91
7.2.8 *Combined ARVGA MLP* architecture 92

8.1.1 An example of a Scene Graph, alongside its corresponding Image 94
8.1.2 Graph Statistics for the Random Subset 96
8.1.3 A Scene Graph alongside its corresponding Image, where the graph contains numerous isolated nodes. 96
8.1.4 Graph Statistics for the Dense Subset 97
8.1.5 Example of WordNet Noun Hypernym/Hyponym Hierarchy [59] 99
8.1.6 Final Relevance Scores for NDCG, computed by the Ground-Truth rankings of GED 101
8.3.1 Example where the query (left) and the first retrieved object (right) according to the best GNN model, have structurally similar Scene Graphs, but quite dissimilar Images 110
8.3.2 Examples of different Queries (left), with the top-3 results (right), as retrieved by the best GNN model 111

8.3.3 Two example queries, where the top-2 retrieved objects by the best GNN model (top) are visually closer to the query image (left), compared to the top-2 retrieved objects by the best Graph Kernel (bottom) 112

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό Υπόβαθρο

1.1.1 Γράφοι

Ένας γράφος ή γράφημα, ορίζεται τυπικά ως μια μαθηματική δομή που συμβολίζεται με $G = (V, E)$, όπου το V αντιπροσωπεύει ένα σύνολο του οποίου τα στοιχεία αναφέρονται ως **κορυφές** ή κόμβοι, και το E υποδηλώνει ένα σύνολο από ζεύγη κορυφών, χαρακτηρίζοντάς τα ως **ακμές**, ή μερικές φορές ως συνδέσμους ή γραμμές.

Η τάξη ενός γράφου προσδιορίζεται ποσοτικά από τον αριθμό των κορυφών του, που συμβολίζεται ως $|V|$, ενώ το μέγεθος ενός γράφου καθορίζεται από τον αριθμό των ακμών του, που συμβολίζεται ως $|E|$. Ο **βαθμός** μιας κορυφής αντιστοιχεί στον αριθμό των ακμών που προσπίπτουν σε αυτήν. Ακμές, όπου και τα δύο άκρα τους προσπίπτουν στον ίδιο κόμβο, δημιουργούν έναν **βρόχο**. Σε ένα γράφημα τάξης n, ο μέγιστος βαθμός οποιασδήποτε κορυφής δεν υπερβαίνει το $n-1$ (ή το $n+1$ εάν επιτρέπονται οι βρόχοι) και το ανώτατο όριο για τον αριθμό των ακμών είναι $\frac{n(n-1)}{2}$ (ή $\frac{n(n+1)}{2}$ αν επιτρέπονται οι βρόχοι).

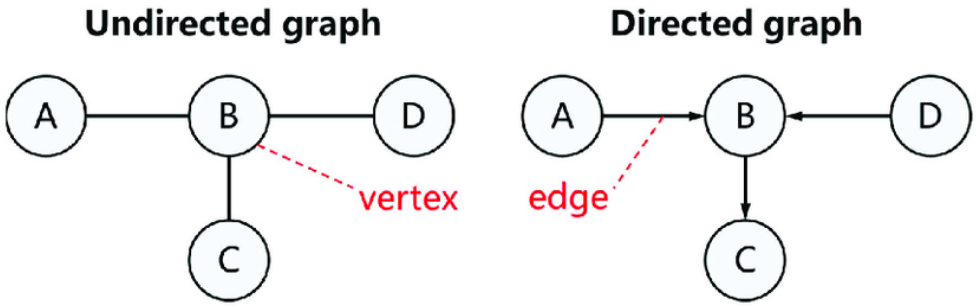


Figure 1.1.1: Παράδειγμα ενός μη-κατευθυνόμενου (αριστερά) και ενός κατευθυνόμενου (δεξιά) γράφου[67]

Οι σχέσεις μεταξύ των κόμβων σε ένα γράφημα, μπορούν να περιγραφούν πλήρως με τον *Πίνακα Γειτνίασης*. Στην περίπτωση των απλών γράφων, ο πίνακας γειτνίασης A μεγέθους $n \times n$, έχει τιμή 1 στο στοιχείο A_{ij} αν υπάρχει ακμή που να συνδέει το κόμβο i με τον j, αλλιώς έχει τιμή 0. Στους μη-κατευθυνόμενους γράφους, αυτός ο πίνακας είναι συμμετρικός, ενώ στην γενική περίπτωση των **κατευθυνόμενων γράφων**, ο πίνακας A είναι μη-συμμετρικός. Επίσης ένας γράφος μπορεί να είναι **βεβαρημένος**, όπου το στοιχείο A_{ij} μπορεί να πάρει οποιαδήποτε αριθμητική τιμή.

Επίσης, ένα είδος γράφων που είναι ιδιαίτερα σημαντικό είναι οι **Ετερογενείς Γράφοι**. Η κύρια διαφορά τους με τους απλούς γράφους, είναι ότι επιτρέπουν την υπάρξη διαφόρων τύπων/κλάσεων, τόσο για τις κορυφές όσο και για τις ακμές. Τέτοιοι γράφοι, μάς επιτρέπουν να αναπαραστήσουμε και να επεξεργαστούμε αρκετά πιο περίπλοκα και ρεαλιστικά δεδομένα.

1.1.2 Απόσταση Επεξεργασίας Γράφων

Η *Απόσταση Επεξεργασίας Γράφων* (ΑΕΓ)[94] είναι μία μετρική ομοιότητας μεταξύ δύο γράφων. Η **Ομοιότητα Γράφων** είναι ένα παραδοσιακό πρόβλημα σε αυτόν τον χώρο, το οποίο έχει να κάνει με την απάντηση στην ερώτηση *"Πόσο δομικά παρόμοιοι είναι δύο γράφοι?"*. Ο αλγόριθμος ΑΕΓ δίνει την απάντηση σε αυτό το ερώτημα, υπολογίζοντας το πλήθος των επεξεργασιών, που θα μεταμορφώσουν ένα γράφο G_i, σε κάποιον ισομορφικό γράφο του G_j. Συγκεκριμένα, για να έχει νόημα το αποτέλεσμα, υπολογίζει τις ελάχιστες επεξεργασίες που πρεέπι να γίνουν στον G_i, και το συνολικό κόστος αυτών των αλλαγών είναι το τελικό αποτέλεσμα. Οι επεξεργασίες αυτές συμπεριλαμβάνουν εισαγωγή/διαγραφή/αντικατάσταση κορυφών και εισαγωγή/διαγραφή/αντικατάσταση ακμών.

Δυστυχώς, έχει αποδειχθεί ότι ο ακριβής υπολογισμός της ΑΕΓ είναι *NP-Hard*[125]. Για αυτόν τον λόγο έχουν δημιουργηθεί αρκετές παραλλαγές του αρχικού αλγορίθμου, οι οποίες προσσεγγίζουν την βέλτιστη λύση σε πολυωνυμικό χρόνο. Η συγκεκριμένη παραλλαγή που θα χρησιμοποιηθεί στα πειράματα είναι ο αλγόριθμος *"Bipartite-Matching"*[27], ο οποίος δουλεύει παρόμοια με τον ακριβή αλγόριθμο ΑΕΓ, αλλά λαμβάνει υπόψη μόνο

τους κόμβους, υπολογίζοντας την βέλτιστη ΑΕΓ μεταξύ τους, και ύστερα προσθέτει σε αυτό το αποτέλεσμα και τις επεξεργασίες που πρέπει να γίνουν στις ακμές.

1.1.3 Πυρήνες Γράφων

Η έννοια των **Πυρήνων** είναι ζωτικής σημασίας στη μηχανική μάθηση, ιδίως για εργασίες αναγνώρισης προτύπων και ανάλυσης δεδομένων. Οι Πυρήνες είναι απαραίτητοι για το μετασχηματισμό των δεδομένων, διευκολύνοντας την αποτελεσματικότητα των αλγορίθμων μάθησης. Οι πυρήνες λειτουργούν με την έμμεση αντιστοίχιση δεδομένων σε χώρους υψηλότερων διαστάσεων, αποκαλύπτοντας πολύπλοκα μοτίβα. Αυτός ο μετασχηματισμός επιτρέπει στους γραμμικούς αλγορίθμους να αντιμετωπίζουν αποτελεσματικά μη γραμμικά προβλήματα. Το τέχνασμα *"Kernel Trick"*, ακρογωνιαίος λίθος των SVMs και των μεθόδων πυρήνων, υπολογίζει εσωτερικά γινόμενα σημείων σε χώρους υψηλότερων διαστάσεων χωρίς τον ρητό μετασχηματισμό των δεδομένων. Δηλαδή μπορούμε να εκμεταλλευτούμε τις ιδιότητες του χώρου υψηλής διαστατικότητας, χωρίς να χρειαστεί ποτέ να μεταβούμε σε αυτόν. Συγκεκριμένα, μία συνάρτηση Πυρήνα ορίζεται ως:

$$k(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >$$

όπου $\phi : \mathcal{X} \to \mathcal{V}$ είναι η συνάρτηση που απεικονίζει τα αρχικά $\mathbf{x}$ και $\mathbf{x}'$ στον χώρο υψηλότερων διαστάσεων. Με το Kernel Trick, οι Πυρήνες μπορούν να χειρίζονται μη γραμμικότητες, ενισχύοντας την αποτελεσματικότητα των γραμμικών ταξινομητών, και προσφέρουν υπολογιστική αποδοτικότητα, απαιτώντας μόνο υπολογισμούς συναρτήσεων πυρήνα ανά ζεύγη.

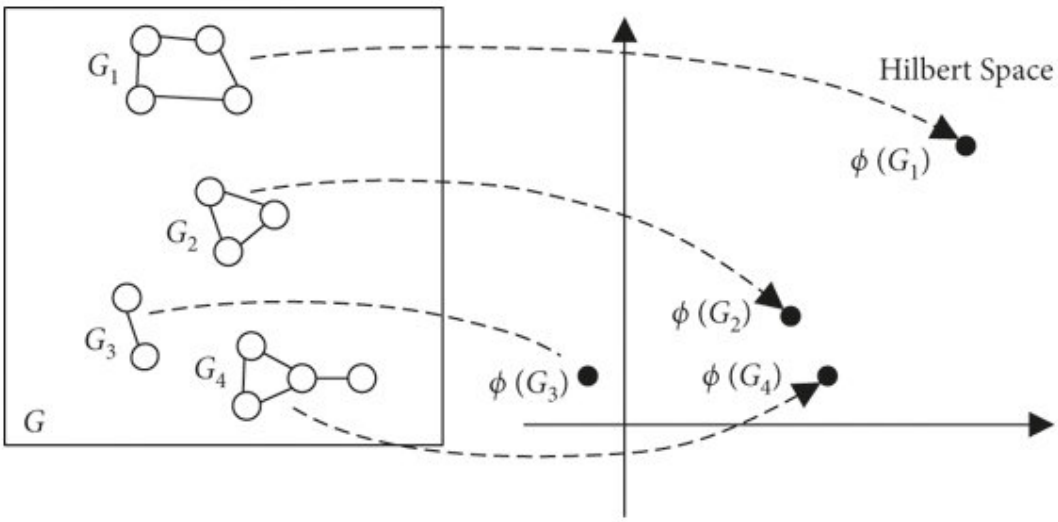


Figure 1.1.2: Οι Πυρήνες Γράφων, μετατρέπουν τα γραφήματα σε σημεία ενός χώρου Hilbert υψηλότερης διάστασης. Αυτό μας επιτρέπει να χρησιμοποιούμε τα παραδοσιακά μέτρα ομοιότητας, συνηθέστερα το εσωτερικό γινόμενο.[64]

Συγκεκριμένα, οι **Πυρήνες Γράφων**, είναι μία υπο-κατηγορία Πυρήνων, όπου τα $\mathbf{x}$ και $\mathbf{x}'$ είναι γράφοι G_i και G_j. Ο σκοπός των Πυρήνων Γράφων είναι να απεικονίσουν τους G_i και G_j ως διανύσματα σε έναν χώρο Hilbert (στον οποίο ύστερα, μπορούμε να εφαρμόσουμε εσωτερικό γινόμενο διανυσμάτων, και να πάρουμε το τελικό νούμερο που δηλώνει την ομοιότητα των δύο γράφων). Οι Πυρήνες Γράφων αποτελούν παραδοσιακά την "γρήγορη" λύση όταν θέλουμε να υπολογίσουμε Ομοιότητα Γράφων, γι'αυτό έχουν μελετηθεί αρκετά, και έχουν προταθεί αρκετοί διαφορετικοί Πυρήνες. Θα παρουσιάσουμε τους πέντε Πυρήνες Γράφων, οι οποίοι θα χρησιμοποιηθούν για στο πειραματικό μέρος, μαζί με την κεντρική ιδέα του καθενός:

- Ο **Shortest Path Kernel**[5], είναι ένας Πυρήνας ο οποίος υπολογίζει όλα τα ελάχιστα μονοπάτια, μεταξύ όλων των κόμβων, και στους δύο γράφους. Ύστερα, με βάση τα κόστη αυτών των μονοπατιών, δημιουργεί δύο καινούριους γράφους, και συγκρίνει τις ακμές τους για να υπολογίσει το τελικό αποτέλεσμα
- Ο **Weisfeiler-Lehman Kernel** [101] δουλεύει επαναληπτικά σε πολλαπλές επαναλήψεις, όπου σε κάθε επανάληψη συγκεντρώνει τις ετικέτες κάθε κόμβου και των γειτόνων του, και τις συμπιέζει σε μία καινούρια ετικέτα. Ύστερα, χρησιμοποιεί κάποιον δευτερεύον Πυρήνα, τον οποίο τον εφαρμόζει στους γράφους με τις καινούριες ετικέτες.
- Ο **NeighborHood Hash Kernel**[44] δουλεύει με αρκετά παρόμοιο τρόπο με τον Weisfeiler-Lehman Kernel, αλλά αντί για ετικέτες χρησιμοποιεί έναν δυαδικό αριθμό για κάθε κόμβο, και για την σύμπτυξη των ετικετών χρησιμοποιεί τις δυαδικές συναρτήσεις *XOR* και *ROT*.
- Ο **Random Walk Kernel**[31] χρησιμποιεί τυχαίους περίπατους πάνω στους γράφους G_i και G_j για να ορίσει μία γεωμετρική σειρά, η οποία είναι ίση με την τελική ομοιότητα των γράφων.
- Ο **Graphlet Sampling Kernel**[88] προσπαθεί να εντοπίσει την συχνότητα εύρεσης *"μικρών γράφων"* (graphlets) στους G_i και G_j, και μετά υπολογίζει την ομοιότητά τους ανάλογα με τα graphlets που βρέθηκαν.

Η κύρια διαφοροποίηση μεταξύ των παραπάνων Πυρήνων Γράφων είναι ότι οι *Graphlet-Sampling*, *Shortest-Path* και *Random-Walk Kernels*, βασίζονται σε πιο κλασσικές μετρικές και δομικά στοιχεία των γράφων, ενώ οι *NeighborHood-Hash* και *Weisfeiler-Lehman Kernels* βασίζονται στην τεχνική **Μετάβασης Μηνυμάτων** (Message Passing), όπου η πληροφορία των κόμβων *"ρέει"* επαναληπτικά προς τους γείτονές τους.

1.1.4 Γράφοι Σκηνής

Οι **Γράφοι Σκηνής** είναι δομημένες αναπαραστάσεις οπτικών σκηνών, που προσφέρουν πληροφορίες για τις σχέσεις και τα χαρακτηριστικά των αντικειμένων. Διαδραματίζουν κεντρικό ρόλο σε διάφορες εργασίες υπολογιστικής όρασης, όπως η ανίχνευση οπτικών σχέσεων, η απόδοση τίτλων εικόνας και η απάντηση οπτικών ερωτήσεων. Σε αντίθεση με τις παραδοσιακές μεθόδους ανίχνευσης αντικειμένων, οι γράφοι σκηνής παρέχουν μια πιο διαφοροποιημένη κατανόηση των οπτικών σκηνών, αξιοποιώντας το σημασιολογικό περιεχόμενο. Οι γράφοι σκηνής που προτάθηκαν για πρώτη φορά το 2015[48], αντιμετωπίζουν την πρόκληση της αποτύπωσης πολύπλοκων σχέσεων αντικειμένων σε μια σκηνή. Από την ίδρυσή τους, έχουν συγκεντρώσει σημαντική ερευνητική προσοχή, εξελισσόμενοι σε ένα ζωτικής σημασίας εργαλείο για τη βελτίωση των εργασιών υπολογιστικής όρασης.

Η παρούσα διατριβή χρησιμοποιεί συγκεκριμένα το σύνολο δεδομένων **Visual Genome**, ένα φημισμένο σύνολο δεδομένων γράφων σκηνής που παρουσιάστηκε το 2016[58]. Το Visual Genome διακρίνεται δίνοντας προτεραιότητα στις σχέσεις και τα χαρακτηριστικά μαζί με τις σημειώσεις αντικειμένων, ενισχύοντας τη συνολική κατανόηση των οπτικών σκηνών. Με περισσότερες από 100.000 εικόνες και λεπτομερείς επισημειώσεις, το σύνολο δεδομένων αποτελεί πολύτιμο πόρο για την εκπαίδευση και την αξιολόγηση μοντέλων. Η προσέγγιση *crowdsourcing*, εξασφαλίζει ένα ευρύ φάσμα ερμηνειών, και μία διαδικασία κανονικοποίησης ενισχύει περαιτέρω την ποιότητα των δεδομένων. Στην παρούσα διατριβή, οι γράφοι σκηνών από το Visual Genome χρησιμοποιούνται για την ποσοτική αξιολόγηση μοντέλων, ενώ οι αντίστοιχες εικόνες χρησιμοποιούνται για την ποιοτική αξιολόγηση. Συνολικά, η μοναδική προσέγγιση του Visual Genome στην επισημείωση συνδέει τη γλώσσα και την όραση, καθιστώντας το έναν απαραίτητο πόρο για διάφορες εργασίες υπολογιστικής όρασης και επεξεργασίας φυσικής γλώσσας.

1.1.5 Νευρωνικά Δίκτυα Γράφων

Τα δεδομένα γράφων γίνονται όλο και πιο διαδεδομένα σε διάφορους τομείς, όπως τα κοινωνικά δίκτυα, η βιοπληροφορική και τα συστήματα συστάσεων. Ωστόσο, οι παραδοσιακοί αλγόριθμοι μηχανικής μάθησης έχουν σχεδιαστεί για να χειρίζονται δεδομένα στον ευκλείδειο χώρο, ο οποίος δεν είναι κατάλληλος για δεδομένα γράφων, τα οποία έχουν πολύπλοκες σχέσεις και αλληλεξαρτήσεις μεταξύ των αντικειμένων, κάτι που δεν μπορεί να αποτυπωθεί από τους παραδοσιακούς αλγορίθμους. Ως εκ τούτου, υπάρχει ανάγκη για εξειδικευμένες

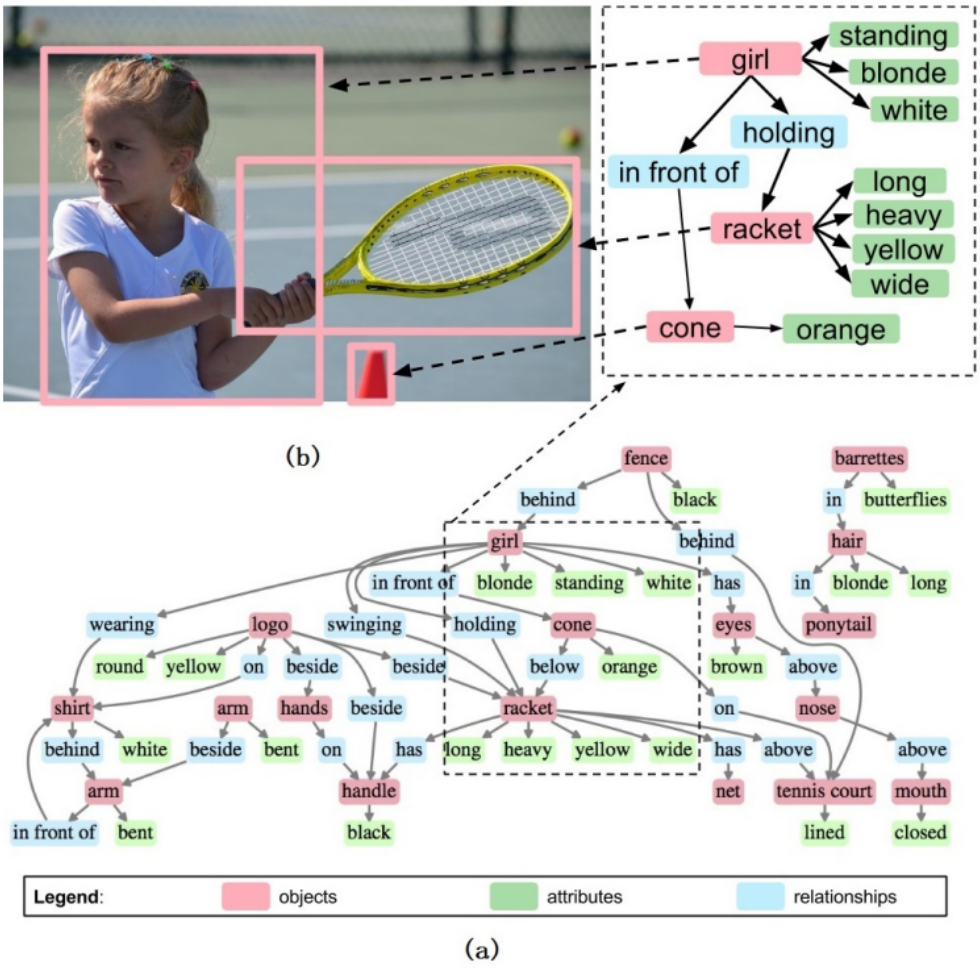


Figure 1.1.3: Ένα παράδειγμα ενός Γράφου Σκηνής, μαζί με την αρχική του εικόνα. Αυτός ο συγκεκριμένος γράφος σκηνής, περιέχει αντικείμενα, σχέσεις και χαρακτηριστικά, αλλά πρέπει να σημειωθεί ότι δεν υπάρχει κάποιος αυστηρός ορισμός που να απαιτεί όλοι οι γράφοι σκηνής να ακολουθούν ένα συγκεκριμένο μοτίβο. Μόνο ότι πρέπει έχουν τη μορφή γράφου και ότι παρέχουν σημασιολογικές πληροφορίες για τα αντικείμενα της εικόνας και τις σχέσεις μεταξύ τους.[122]

αρχιτεκτονικές που μπορούν να επεξεργάζονται αποτελεσματικά δεδομένα γράφων. Αυτές είναι οι αρχικές ιδέες που οδήγησαν στην δημιουργία των **Νευρωνικών Δικτύων Γράφων** (ΝΔΓ).

Η κύρια λειτουργία που χρησιμοποιούν τα ΝΔΓ για να επεξεργαστούν και να αναλύσουν δεδομένα σε μορφή γράφων, είναι η **Συνέλιξη Γράφων** (Graph Convolution).Όπως υποδηλώνει και το όνομά της, είναι η αντίστοιχη λειτουργία της συνέλιξης σημάτων (όπως αυτή που εφαρμόζεται στα Συνελικτικά Νευρωνικά Δίκτυα για εικόνες), αλλά με τις απαραίτητες διαφοροποιήσεις, ώστε να μπορεί να εφαρμοστεί σε γράφους.

Όπως και με την κλασσική συνέλιξη, η Συνέλιξη Γράφων μπορεί να μελετηθεί τόσο στο χωρικό πεδίο, όσο και στο φασματικό πεδίο. Στο φασματικό πεδίο, η ανάλυση ενός ΝΔΓ συμπεριλαμβάνει τον ορισμό της συνάρτησης που εφαρμόζει, αφού γίνει ο κατάλληλος μετασχηματισμός Fourier (για την συχνοτική μελέτη). Συγκεκριμένα, στην περίπτωση των συναρτήσεων στα ΝΔΓ, αυτές έχουν την εξής μορφή:

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}$$

όπου $\mathbf{g}_\theta$ είναι το *"φίλτρο"* που εφαρμόζει το ΝΔΓ, $\mathbf{x}$ είναι τα αρχικά δεδομένα σε μορφή γράφου, και $\mathbf{U}$ είναι ο πίνακας ιδιοδιανυσμάτων του Λαπλασιανού πίνακα του αρχικού γράφου. Άρα στην περίπτωση της φασματικής

ανάλυσης, η μόνη διαφορά των ΝΔΓ, είναι ο τρόπος με τον οποίο μοντελοποιούν και μαθαίνουν το φίλτρο $\mathbf{g}_\theta$.

Για την ανάλυση στο χωρικό πεδίο, τα ΝΔΓ εκμεταλλεύονται την τεχνική **Μετάβασης Μηνυμάτων**, δηλαδή διαδίδουν την πληροφορία που έχει ο κάθε κόμβος, σε όλους τους γείτονές του. Η μαθηματική έκφραση μίας τέτοιας συνάρτησης είναι:

$$\mathbf{h}_v^{(k)} = U_k\left(\mathbf{h}_v^{(k-1)}, \sum_{u\in N(v)} M_k(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_{vu}^e)\right)$$

όπου $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $U_k(\cdot)$ και $M_k(\cdot)$ είναι συναρτήσεις με μαθήσιμες παραμέτρους. Αφού υπολογιστούν οι κρυφές αναπαραστάσεις των μεμονωμένων κόμβων, που συμβολίζονται ως $\mathbf{h}_v^{(K)}$, μπορούν να προωθηθούν είτε σε **ένα επίπεδο εξόδου** για εργασίες που περιλαμβάνουν πρόβλεψη σε επίπεδο κόμβου, είτε σε **μια συνάρτηση ανάγνωσης** για εργασίες που περιλαμβάνουν πρόβλεψη σε επίπεδο γραφήματος. Η συνάρτηση ανάγνωσης είναι υπεύθυνη για τη δημιουργία μιας ενθετικής αναπαράστασης ολόκληρου του γράφου αξιοποιώντας τις κρυφές αναπαραστάσεις των κόμβων που τον αποτελούν. Τα περισσότερα χωρικά ΝΔΓ, μπορούν να γραφούν ως μια παραλλαγή της παραπάνω μαθηματικής έκφρασης.

Με την εκτενή έρευνα που έχει γίνει πάνω στα ΝΔΓ τα τελευταία χρόνια, η ερευνητική κοινότητα έχει επικεντρωθεί για αρκετούς λόγους στα χωρικά ΝΔΓ, καθώς προσφέρουν αρκετά πλεονεκτήματα, συγκριτικά με τα φασματικά ΝΔΓ. Οι τέσσερις μονάδες ΝΔΓ που θα χρησιμοποιηθούν σε αυτήν την διατριβή είναι οι εξής:

- Το **Graph Convolution Network** (GCN) προτάθηκε το 2016[55], και είναι ένα από τα πιο σημαντικά ΝΔΓ, καθώς ήταν το πρώτο που ενοποίησε τα χωρικά και τα φασματικά ΝΔΓ σε ένα μοντέλο. Συγκεκριμένα, η χωρισκή εξίσωση που το περιγράφει (δηλαδή ο συνάρτηση για το $\mathbf{h}_v^{(k)}$) είναι:

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

  όπου Θ είναι πίνακας εκπαιδεύσιμων παραμέτρων. Η αντίστοιχη φασματική εξίσωση σε μορφή πινάκων είναι:

  $$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$$

  όπου $X \in \mathbb{R}^{N\times C}$, $\Theta \in \mathbb{R}^{C\times F}$, $Z \in \mathbb{R}^{N\times F}$, και C είναι η διάσταση των διανυσμάτων των κόμβων εισόδου και F είναι η διάσταση των διανυσμάτων των κόμβων εξόδου. Εδώ, οι πίνακες $\tilde{A}$ και $\tilde{D}$ είναι παρόμοιοι με τους αρχικούς πίνακες A (πίνακας γειτνίασης) και D (πίνακας βαθμών), αφού τους κάνουμε επανακανονικοποίηση. Το *τρικ επανακανονικοποίησης*, είναι μία αντικατάσταση που πρότειναν οι δημιουργοί του GCN, με σκοπό να αποφευχθούν αστάθειες στις παραγώγους του τελικού μοντέλου. Συγκεκριμένα, χρησιμοποίησαν το $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ αντί για $I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, όπου $\tilde{A} = A + I_N$ και $\tilde{D} = \sum_j \tilde{A}_{ij}$.

  Δεδομένου ότι η παραπάνω εξίσωση είναι γραμμική, προκειμένου να επιλυθούν και μη γραμμικά προβλήματα, εφαρμόζεται μια συνάρτηση ενεργοποίησης (π.χ. ReLU) στα διανύσματα εξόδου του GCN.

- Το ΝΔΓ **Graph Attention** (GAT), προτάθηκε το 2017 [113], και η κύρια ιδέα ήταν η χρήση του μηχανισμού προσοχής, και συγκεκριμένα του *multi-head attention*[112]. Ο μηχανισμός προσοχής χρησιμοποιείται για να δημιουργήσει τα "βάρη" που αποδίδει ο κάθε κόμβος στους γείτονές του. Η τελική εξίσωση του GAT είναι:

  $$\mathbf{x}'_i = a_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} a_{i,j}\mathbf{\Theta}\mathbf{x}_j$$

  όπου τα $a_{i,j}$ είναι τα *βάρη προσοχής*, και υπολογίζονται ως εξής:

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k]\right)\right)}$$

όπου $\mathbf{a}$ και $\mathbf{\Theta}$ είναι εκπαιδεύσιμοι παράμετροι. Η αρνητική κλίση του LeakyReLU ορίζεται σε $a = 0.2$ και στην περίπτωση της Multi-Headed Attention, την εφαρμόζουν με τον ίδιο τρόπο όπως στο [112].

- Το **GATv2**[8] αποτελεί μία μετεξέλιξη του GAT, καθώς διορθώνει ένα από τα προβλήματα που είχε στον μηχανισμό προσοχής. Συγκεκριμένα, οι συγγραφείς απέδειξαν ότι το GAT υπολογίζει *στατική προσοχή*, και όχι την αναμενόμενη *δυναμική προσοχή*. Αυτό το έφτιαξαν, αλλάζοντας τον τρόπο υπολογισμού των βαρών $a_{i,j}$:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{\Theta}[\mathbf{x}_i \,\|\, \mathbf{x}_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{\Theta}[\mathbf{x}_i \,\|\, \mathbf{x}_k]\right)\right)}$$

- Τέλος το **Graph Isomorphism Network** (GIN)[121], εισάγει μια τροποποίηση όπου προσαρμόζει το βάρος του κεντρικού κόμβου χρησιμοποιώντας μια εκπαιδεύσιμη παράμετρο ϵ:

$$\mathbf{x}'_i = h_{\mathbf{\Theta}}\left((1+\epsilon)\cdot \mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \mathbf{x}_j\right)$$

όπου $h_{\mathbf{\Theta}}$ είναι μια οποιαδήποτε υλοποίηση ενός νευρωνικού δικτύου. Υποστηρίζουν επίσης ότι, θέτοντας την αθροιστική συνάρτηση **Sum** να είναι η συνάρτηση ανάγνωσης του ΝΔΓ, το *GIN* γενικεύει το τεστ Weisfeiler-Lehman και συνεπώς επιτυγχάνει τη μέγιστη διακριτική ικανότητα μεταξύ των ΝΔΓ.

Αυτο-κωδικοποιητές Γράφων

Οι **Αυτο-κωδικοποιητές Γράφων** (ΑΚΓ) είναι μια υπο-κατηγορία ΝΔΓ που χρησιμοποιούνται για μη-επιβλεπόμενη μάθηση σε δεδομένα γράφων. Ο κύριος στόχος ενός ΑΚΓ είναι η εκμάθηση μιας συμπιεσμένης αναπαράστασης του γράφου εισόδου, η οποία μπορεί να χρησιμοποιηθεί για διάφορα προβλήματα. Τα ΑΚΓ αποτελούνται από δύο κύρια συστατικά: έναν κωδικοποιητή και έναν αποκωδικοποιητή. Ο κωδικοποιητής αντιστοιχίζει τον γράφο εισόδου σε μια συμπιεσμένη αναπαράσταση, ενώ ο αποκωδικοποιητής αντιστοιχίζει τη συμπιεσμένη αναπαράσταση πίσω στον αρχικό γράφο.

Συγκεκριμένα, θα παρουσιάσουμε τρεις βασικές αρχιτεκτονικές ΑΚΓ, οι οποίες αποτελούν θεμελιώδη στοιχεία των πιο περίπλοκων μοντέλων που θα προτείνουμε πιο μετά. Αυτές οι τρεις αρχιτεκτονικές είναι οι εξής:

- Στον **Variational Graph Autoencoder** (VGAE)[54], ο *Κωδικοποιητής* μπορεί να είναι ένα οποιοδήποτε ΝΔΓ που παράγει ενθέσεις για τους κόμβους, και ο προτεινόμενος *Αποκωδικοποιητής* είναι ο **Inner-Product Decoder** (αποκωδικοποιητής εσωτερικού γινομένου). Ο τρόπος με τον οποίο λειτουργεί ο *Inner-Product Decoder* είναι ότι, με δεδομένες τις ενθέσεις κόμβων $\mathbf{Z} \in \mathbb{R}^{N\times F}$ στο λανθάνοντα χώρο (όπου F είναι η διάσταση των λανθανουσών ενθετικών αναπαραστάσεων), θα προβλέψει τον πίνακα γειτνίασης $\mathbf{A}$ υπολογίζοντας:

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^T\right)$$

Έτσι μπορούμε να δούμε, ότι "έμμεσα", το πρόβλημα στο οποίο εκπαιδεύεται ο αποκωδικοποιητής είναι η Πρόβλεψη Συνδέσμων, στις ακμές του γράφου εισόδου. Στη συνέχεια, δεδομένου του αρχικού πίνακα γειτνίασης $\mathbf{A}$, η απώλεια του απλού GAE είναι *Binary Cross-Entropy Loss* για τα θετικά και τα αρνητικά δείγματα(ακμές). Όταν ο γράφος είναι πολύ αραιός, είναι σύνηθες να υποδειγματοληπτούνται οι αρνητικές ακμές, ώστε να υπάρχει ίση ποσότητα δειγμάτων και για τις δύο κλάσεις. Για τον VGAE, προστίθεται και

η απώλεια που υπολογίζεται από την *Απόκλιση Kullback-Leibler*, μεταξύ των λανθανουσών ενθέσεων, και μία πρότερης κατανομής (συνήθως κανονικής γκαουσσιανής $\mathcal{N}(0, 1)$).

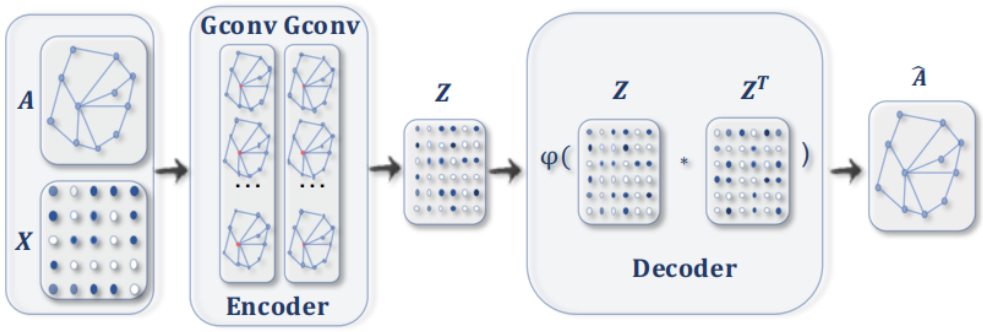


Figure 1.1.4: Η βασική αρχιτεκτονική του ΑΚΓ, που αποτελείται από ένα Κωδικοποιητή και έναν Αποκωδικοποιητή [120]

- Ο **Adverarially Reguralized Variational Graph Autoencoder** (ARVGA)[84], επεκτείνει περαιτέρω τον αρχικό *VGAE*, εφαρμόζοντας ιδέες **Ανταγωνιστικής Μάθησης**. Συγκεκριμένα, χρησιμοποιείται ένας Διευκρινιστής (που υλοποιείται ως ένα απλό νευρωνικό δίκτυο), η οποία λαμβάνει ως είσοδο τα δείγματα ενθέσεων από τον λανθάνων χώρο και δείγματα από την πρότερη κατανομή. Στόχος είναι η επιτυχής ταξινόμησή τους ως αληθή ή ψευδή δείγματα.

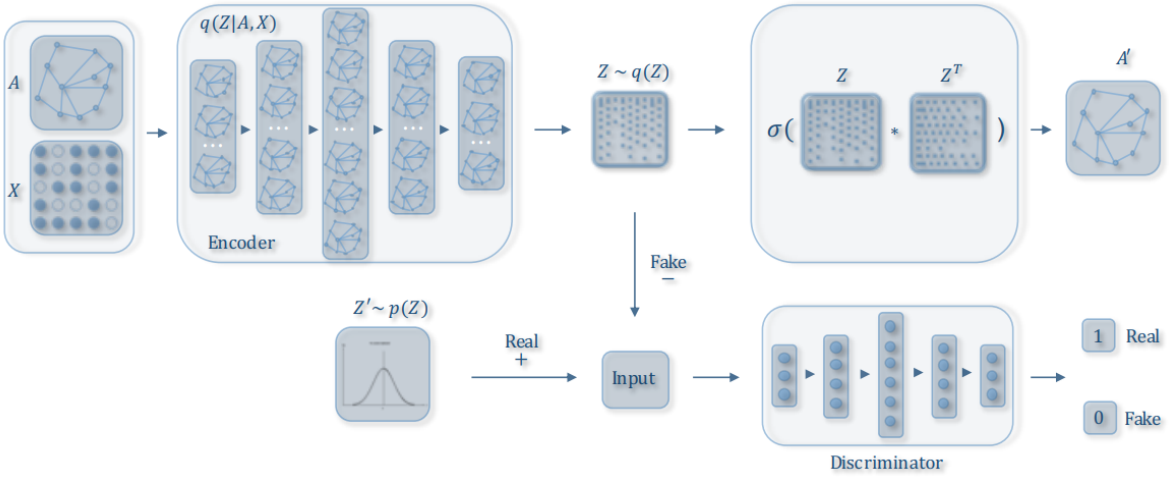


Figure 1.1.5: Αρχιτεκτονική του μοντέλου ARVGA, όπως παρουσιάστηκε στο [84]

- Ο **Graph Feature Autoencoder** [42], ήταν το πρώτο μοντέλο της οικογένειας των ΑΚΓ, που χρησιμοποίησε έναν *Αποκωδικοποιητή Χαρακτηριστικών* και όχι έναν αποκωδικοποιητή συνδέσμων. Συγκεκριμένα, χρησιμοποιούσαν τις ενθέσεις κόμβων που παρήγαγε ο κωδικοποιητής και τις πέρασαν σε ένα απλό νευρωνικό δίκτυο. Ο στόχος αυτού του δικτύου, είναι να λύσει ένα πρόβλημα παλινδρόμησης, για την ανακατασκευή του αρχικού πίνακα χαρακτηριστικών $\mathbf{X} \in \mathbb{R}^{N \times F}$ από τον λανθάνοντα πίνακα ενθέσεων $\mathbf{Z} \in \mathbb{R}^{N \times D}$. Ο τελικός όρος απώλειας, που προστίθεται από αυτόν τον Αποκωδικοποιητή Χαρακτηριστικών, είναι το *Μέσο Τετραγωνικό Σφάλμα* μεταξύ του $\mathbf{X}$ και της εξόδου $\hat{\mathbf{X}}$ του αποκωδικοποιητή.

1.1.6 Εξηγήσεις με Αντιπαραδείγματα

Οι **Εξηγήσεις με Αντιπαραδείγματα** είναι ένας τύπος εξήγησης που αποσκοπεί στην αποκάλυψη του τι θα έπρεπε να είχε γίνει διαφορετικά σε μια περίπτωση για να παρατηρηθεί ένα διαφορετικό αποτέλεσμα. Στο πλαίσιο της μηχανικής μάθησης, οι εξηγήσεις με αντιπαραδείγματα χρησιμοποιούνται για να εξηγήσουν τις αποφάσεις που λαμβάνονται από μη ερμηνεύσιμους ταξινομητές. Είναι ιδιαίτερα πολύτιμες επειδή παρέχουν αξιοποιήσιμες γνώσεις σχετικά με το πώς μπορεί να αλλάξει η είσοδος για να επιτευχθεί μια επιθυμητή έξοδος. Συγκεκριμένα, μπορούν να δώσουν μια απάντηση στην ερώτηση *"Τι θα πρέπει να αλλάξει για να ταξινομηθεί κάτι ως X αντί για Y;"*. Για παράδειγμα, ένας πελάτης τράπεζας στον οποίο έχει απορριφθεί ένα δάνειο μπορεί να λάβει μια εξήγηση με αντιπαράδειγμα που αποκαλύπτει τι θα μπορούσε να είχε κάνει διαφορετικά για να εγκριθεί η αίτησή του.

Το μεγαλύτερο πλεονέκτημα τέτοιων επεξηγήσεων, είναι ότι δεν απαιτούν κάποια εσωτερική πρόσβαση στο μοντέλο που θέλουν να ερμηνεύσουν, δηλαδή το αντιμετωπίζουν ως **μαύρο κουτί** (Black-Box). Δεδομένου ότι οι μέθοδοι παραγωγής Αντιπαραδειγμάτων δεν χρειάζονται εσωτερική πρόσβαση στα μοντέλα, δεν χρειάζεται να λαμβάνουμε υπόψη την Εξηγησιμότητα του μοντέλου μας κατά την κατασκευή/εκπαίδευση/αξιολόγησή του. Η μόνη προϋπόθεση για το μοντέλο, είναι να μπορεί να ταξινομήσει τα αντικείμενα εισόδου στις διάφορες κλάσεις.

Συγκεκριμένα, το σύστημα-επεξηγητής που θα παρουσιάσουμε, προτάθηκε από τους Φιλανδριανός και λοιποί [29], και βασίζεται στην παραγωγή Αντιπαραδειγμάτων, μέσω **Εννοιολογικών Επεξεργασιών**. Το βασικό στοιχείο αυτού του συστήματος, είναι η χρήση ενός *Συνόλου Δεδομένων Επεξηγήσεων*, δηλαδή ένα σύνολο, όπου κάθε δείγμα συνοδεύεται και από ένα σύνολο *Εννοιών*. Με την χρήση των αξιωμάτων του TBox, βρίσκει *Αποστάσεις Εννοιών*, και τις γενικεύει σε ελάχιστες *Αποστάσεις Συνόλων Εννοιών*, με τον αλγόριθμο Karp. Εφόσον το κάθε δείγμα του αρχικού συνόλου δεδομένων, αντιστοιχίζεται σε ένα σύνολο εννοιών, τότε μπορούμε να υπολογίσουμε την *εννοιολογική απόσταση* μεταξύ των δειγμάτων, να κατασκευάσουμε ένα γράφο με αυτές τις αποστάσεις, και τέλος να υπολογίσουμε ελάχιστα μονοπάτια με τον αλγόριθμο Dijkstra. Έτσι, αν έχω το δείγμα x_1 που ταξινομήθηκε στην κλάση C_i, το Αντιπαράδειγμα θα είναι το δείγμα που έχει την πιο κοντινή εννοιολογική απόσταση στο x_1 αλλά δεν ταξινομήθηκε στην C_i (Local Counterfactual Explanation). Επίσης, η παραπάνω διαδικασία μπορεί να γενικευτεί, για να παράγει εξηγήσεις για μία ολόκληρη κλάση ή ομάδα δειγμάτων, και όχι για ένα μόνο δείγμα (Global Counterfactual Explanation).

Σε μία νεότερη έκδοση του ίδιου συστήματος [21], προτείνεται η μετατροπή των Συνόλων Εννοιών, σε **Γράφους Γνώσης** (το οποίο αντιμετωπίζεται ως ABox). Δηλαδή κάθε δείγμα, θα συνοδεύται από έναν Γράφο Γνώσης, ο οποίος θα δίνει μία εννοιολογική επεξήγηση για το συγκεκριμένο δείγμα. Η κύρια διαφορά, είναι ότι τώρα έχουμε να υπολογίσουμε *Απόσταση Γράφων*, και όχι απόσταση συνόλων εννοιών. Όπως έχουμε ήδη αναφέρει, αυτό το πρόβλημα είναι NP-Hard, γι'αυτό και οι συγγραφείς περιορίζουν την πληροφορία του κάθε κόμβου μόνο μέχρι τους άμεσους γείτονές τους, βρίσκοντας έτσι μία προσεγγιστική λύση. Άρα, είναι έυκολο να δούμε ότι μπορούμε να αξιοποιήσουμε τα ΝΔΓ για να κατατάξουμε αυτούς τους γράφους και να βρούμε τους πιο παρόμοιους. Μάλιστα, με την χρήση ΑΚΓ, η όλη διαδικασία θα είναι αποτελεσματική και εύκολη στην υλοποίηση, επειδή δεν χρειάζεται να υπολογίσουμε κάποιο άλλο είδος μετρικής απόστασης/ομοιότητας, αφού η εκπαίδευση του ΑΚΓ είναι μη-επιβλεπόμενη.

1.2 Προτεινόμενα Μοντέλα

1.2.1 Συσεισφορά

Οι κύριες συνεισφορές της παρούσας διατριβής περιγράφονται παρακάτω:

- Χρησιμοποιούμε Αυτο-κωδικοποιητές Γράφων, για να αντιμετωπίσουμε το παραδοσιακό πρόβλημα της Ομοιότητας Γράφων. Εξ όσων γνωρίζουμε, η αξιοποίηση των ΑΚΓ για την επίλυση αυτού του προβλήματος, δεν έχει τύχει μεγάλης ακαδημαϊκής προσοχής μέχρι στιγμής, οπότε στοχεύουμε να δώσουμε μια ολοκληρωμένη επισκόπηση του συγκεκριμένου προβλήματος και των μεθόδων που χρησιμοποιούμε για την επίλυσή του.
- Η χρήση αρχιτεκτονικών βασισμένων σε Αυτο-κωδικοποιητές, μάς επιτρέπει να εκπαιδεύσουμε τα μοντέλα με μη-επιβλεπόμενο τρόπο, χωρίς την ανάγκη υπολογισμού της Απόστασης Επεξεργασίας Γράφων μεταξύ των δειγμάτων. Επιπλέον, αυτές οι αρχιτεκτονικές δεν απαιτούν ζεύγη γραφημάτων για την εκπαίδευση,

μειώνοντας περαιτέρω τη τάξη των δειγμάτων σε $\mathcal{O}(n)$, σε σύγκριση με $\mathcal{O}(n^2)$ για τα μοντέλα με επίβλεψη. Αυτές οι δύο θεμελιώδεις πτυχές είναι ο κύριος λόγος για την επιτάχυνση της εκπαίδευσης των προτεινόμενων μοντέλων.

- Προτείνουμε διάφορες βελτιστοποιήσεις στους βασικούς ΑΚΓ που παρουσιάζονται στην ενότητα 1.1.5, μαζί με τη κύρια ιδέα πίσω από κάθε επιλογή. Αξιολογούμε επίσης την επίδραση των βασικών στοιχείων των διαφορετικών αρχιτεκτονικών, συνδυάζοντάς τα, και παραθέτοντας τα ποσοτικά αποτελέσματα στην Ομοιότητα Γράφων.
- Τα προτεινόμενα μοντέλα ΝΔΓ, μάς παρέχουν βαθμολογίες ομοιότητας μεταξύ όλων των γράφων σκηνής. Αυτό επιτρέπει στο μοντέλο να χρησιμοποιηθεί τελικά μαζί με ένα πλαίσιο Εξήγησης με Αντιπαραδείγματα, παρόμοιο με αυτό που παρουσιάστηκε στην ενότητα 1.1.6.

1.2.2 Μοντέλα Αυτο-κωδικοποιητών Γράφων

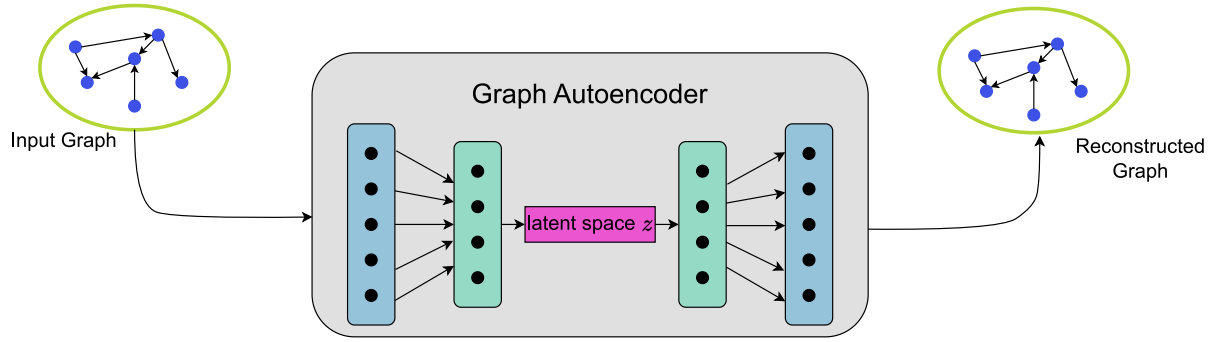


Figure 1.2.1: Χρήση Αυτο-κωδικοποιητών Γράφων για την φάση της Εκπαίδευσης.

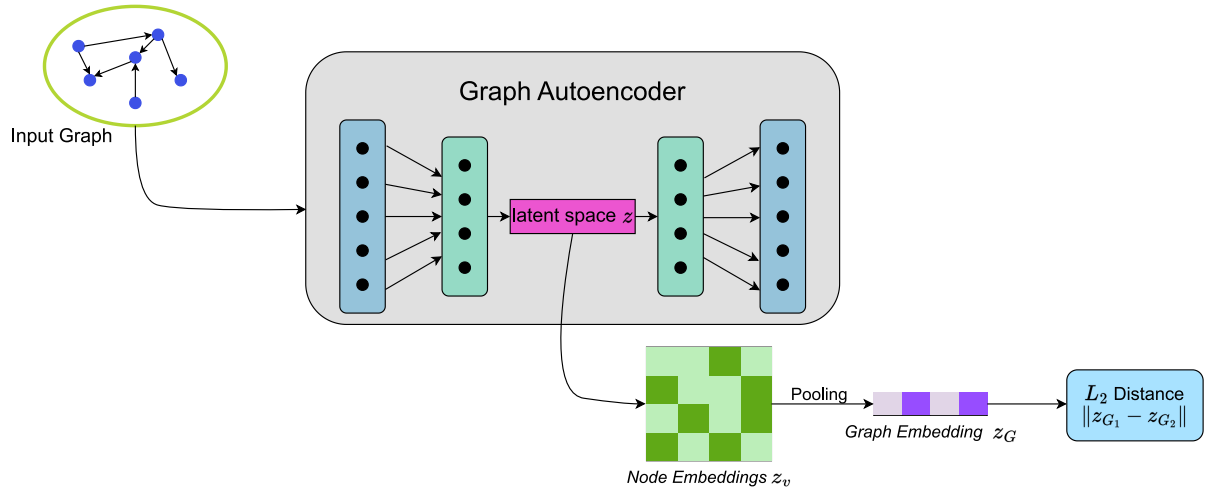


Figure 1.2.2: Χρήση Αυτο-κωδικοποιητών Γράφων για την φάση τους Συμπερασμού.

Το αρχικό, βασικό μοντέλο πάνω στο οποίο θα χτίσουμε, είναι το *VGAE*[54]. Η αρχιτεκτονική του μοντέλου παρουσιάζεται στο σχήμα 1.2.3, όπου τα κόκκινα ορθογώνια δείχνουν τους τελικούς όρους για το *Loss Function*, τα οποία το μοντέλο θα συνδυάσει και θα ελαχιστοποιήσει. Η δεύτερη θεμελιώδης αρχιτεκτονική είναι το *ARVGA*[84], το οποίο προσθέτει τον *Διευκρινιστή* σε ολόκληρο το μοντέλο, όπως φαίνεται στο Σχήμα 1.2.4.

Το πρώτο προτεινόμενο μοντέλο είναι το **Feature VGAE**, το οποίο προσθέτει έναν Αποκωδικοποιητή Χαρακτηριστικών, παράλληλα με τον αρχικό Αποκωδικοποιητή Ακμών (όπως φαίνεται στο Σχήμα 1.2.5), από την αρχική αρχιτεκτονική.

Το δεύτερο προτεινόμενο μοντέλο είναι το **Combined VGAE** (Σχήμα 1.2.6), το οποίο συνδυάζει έναν Αποκωδικοποιητή Ακμών και έναν Αποκωδικοποιητή Χαρακτηριστικών, αλλά και οι δύο χρησιμοποιούν μια μονάδα ΝΔΓ με *εκπαιδεύσιμες παραμέτρους*. Η κύρια ιδέα για αυτό το μοντέλο, είναι να προστεθεί ένα επίπεδο με εκπαιδεύσιμες παραμέτρους, πριν από τον *Inner Product Decoder*, στον Αποκωδικοποιητή Ακμών. Η διαίσθηση πίσω από αυτή την απόφαση, είναι να αναγκαστει ο Κωδικοποιητής να ενσωματώσει τη δομική γνώση του γράφου, με έναν πιο περίπλοκο τρόπο, από μια απλή πράξη Εσωτερικού Γινόμενου.

Το τρίτο προτεινόμενο μοντέλο είναι το **Combined ARVGA**, το οποίο προσθέτει τη μονάδα του Διευκρινιστή, στην αρχιτεκτονική *Combined VGAE*, όπως φαίνεται στο Σχήμα 1.2.7. Πειραματιζόμαστε επίσης με μια παραλλαγή αυτού, το **Combined ARVGA MLP**, όπου η μόνη διαφορά με το **Combined ARVGA** είναι ότι οι *εκπαιδεύσιμες παράμετροι* στον Αποκωδικοποιητή Ακμών, υλοποιούνται με ένα παραδοσιακό νευρωνικό δίκτυο (Σχήμα 1.2.8), και όχι με μια μονάδα ΝΔΓ.

Θα δώσουμε τώρα μερικές λεπτομέρειες σχετικά με τις υπομονάδες των αρχιτεκτονικών, οι οποίες εφαρμόζονται σε όλα τα μοντέλα που αναφέρθηκαν παραπάνω:

- Οι *μονάδες ΝΔΓ* που θα δοκιμαστούν, είναι αυτές που παρουσιάστηκαν στην ενότητα 1.1.5, οι οποίες είναι οι **GCN**, **GAT**, **GATv2** και **GIN**.

- Όταν υπάρχουν περισσότερες από μία *μονάδα ΝΔΓ* σε ένα μοντέλο, τότε το ίδιο είδος μονάδας χρησιμοποιείται παντού. Για παράδειγμα, δεν θα χρησιμοποιήσουμε ένα **GCN** για τον κωδικοποιητή και ένα **GAT** για τον αποκωδικοποιητή. Αυτό δοκιμάστηκε στα πρώτα πειράματα, αλλά τα αποτελέσματα έδειξαν γρήγορα ότι δεν ήταν μια βιώσιμη επιλογή για τα μοντέλα. Μια πιθανή εξήγηση γι' αυτό θα μπορούσε να είναι ότι οι διαφορετικές μονάδες ΝΔΓ, κωδικοποιούν και αποκωδικοποιούν τους γράφους με διαφορετικό τρόπο, εστιάζοντας σε διαφορετικά χαρακτηριστικά. Έτσι, η συνεργασία τους, δεν παράγει απαραίτητα αποτελέσματα ισοδύναμης ποιότητας.

- Για τον συμπερασμό των μοντέλων, όπως βλέπουμε στο Σχήμα 1.2.2, πρέπει να ορίσουμε μια συνάρτηση **Global Pooling**, η οποία συγκεντρώνει τις ενθέσεις των κόμβων σε μία εννιαία ενθετική αναπαράσταση ολόκληρου του γράφου. Αφού δοκιμάσαμε τις πιο δημοφιλείς συναρτήσεις για το Global Pooling (mean/max/min/sum), έγινε γρήγορα φανερό ότι η συνάρτηση **Sum**, είχε τα καλύτερα αποτελέσματα, οπότε όλα τα τελικά μοντέλα χρησιμοποιούν αυτή τη συνάρτηση. Το ίδιο επιχείρημα υποστηρίζεται επίσης από τους δημιουργούς της **GIN**[121].

- Για το τελικό στάδιο του συμπερασμού, χρησιμοποιούμε την απόσταση L_2 των ενθέσεων των γράφων, προκειμένου να κατατάξουμε τις βαθμολογίες ομοιότητας μεταξύ τους και να ανακτήσουμε παρόμοιους γράφους. Πειραματιστήκαμε επίσης με την *Ομοιότητα Συνημιτόνου*, αλλά η απόδοση ήταν χειρότερη συγκριτικά με την απόσταση L_2.

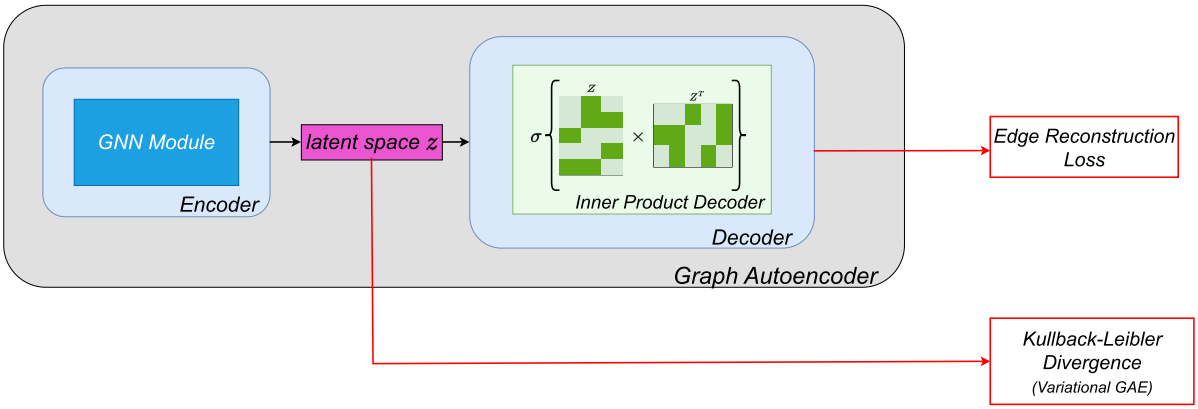


Figure 1.2.3: Αρχική αρχιτεκτονική του *VGAE*

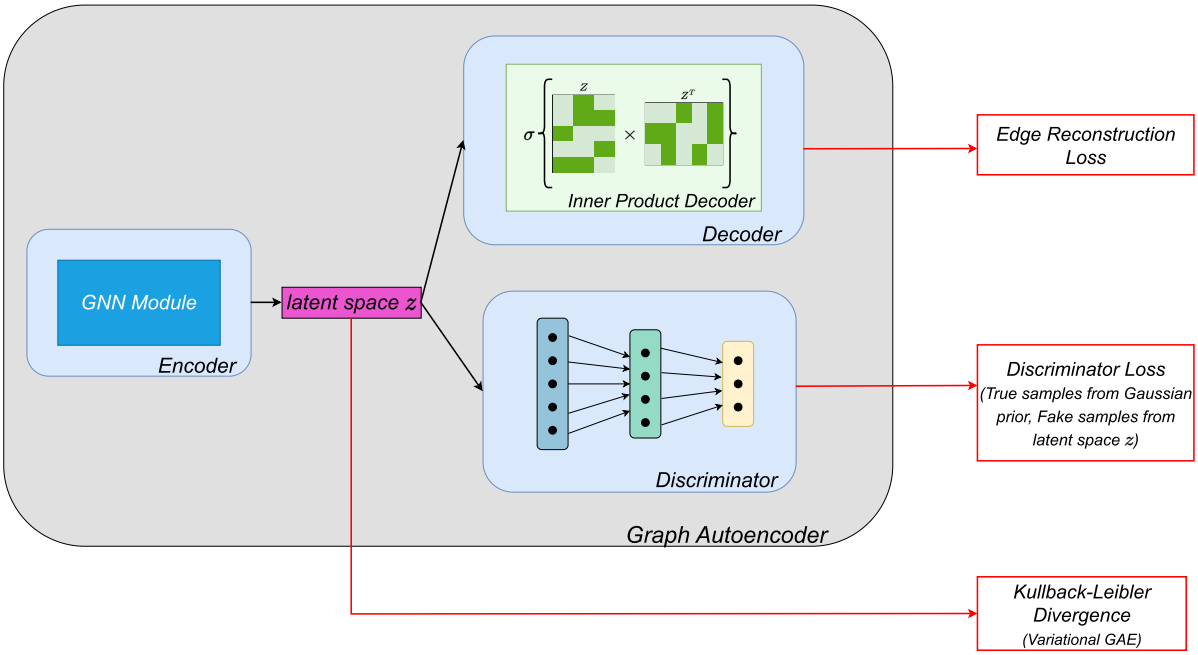


Figure 1.2.4: Αρχιτεκτονική του *ARVGA*

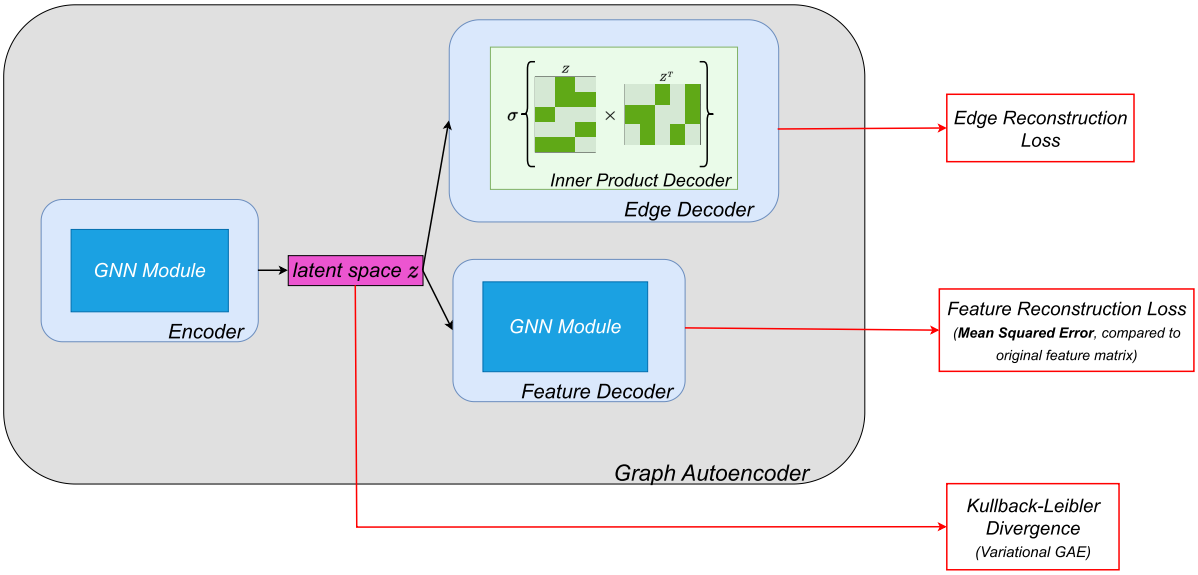


Figure 1.2.5: Αρχιτεκτονική *Feature VAGE*

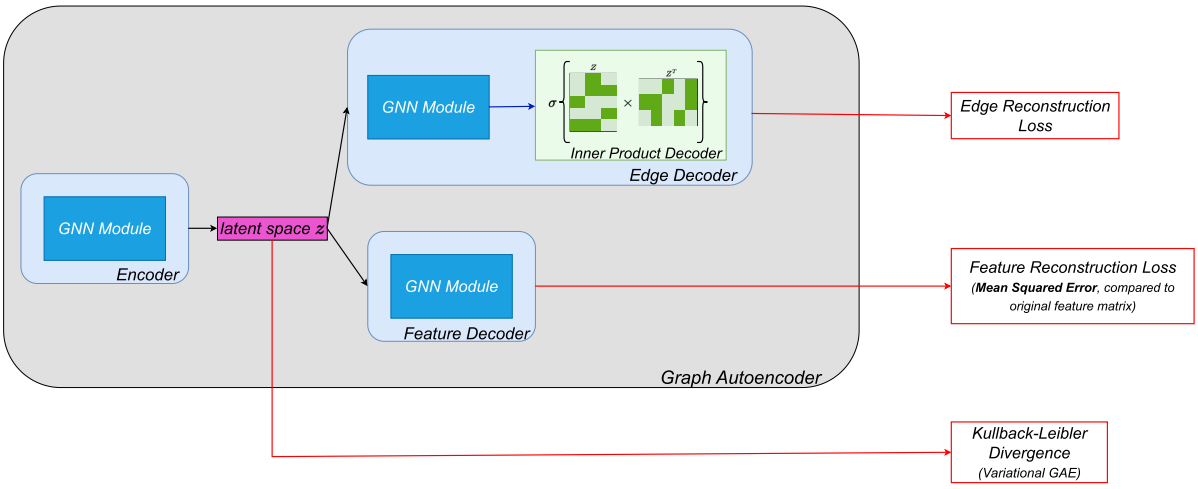


Figure 1.2.6: Αρχιτεκτονική *Combined VGAE*

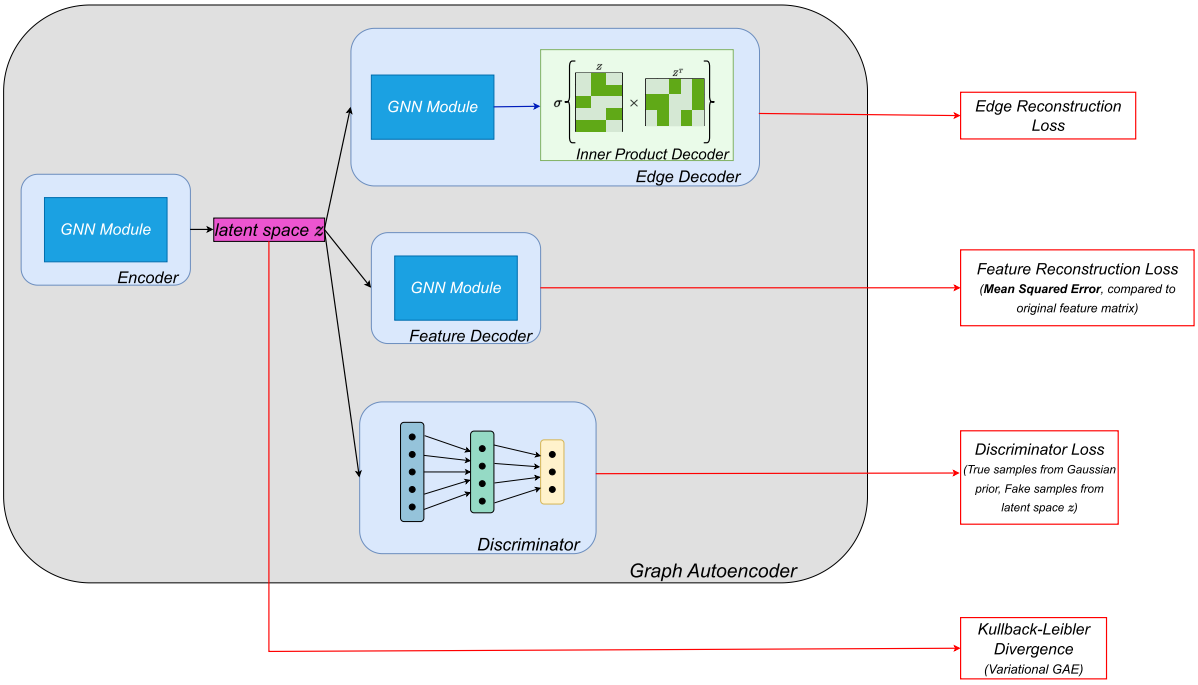


Figure 1.2.7: Αρχιτεκτονική *Combined ARVGA*

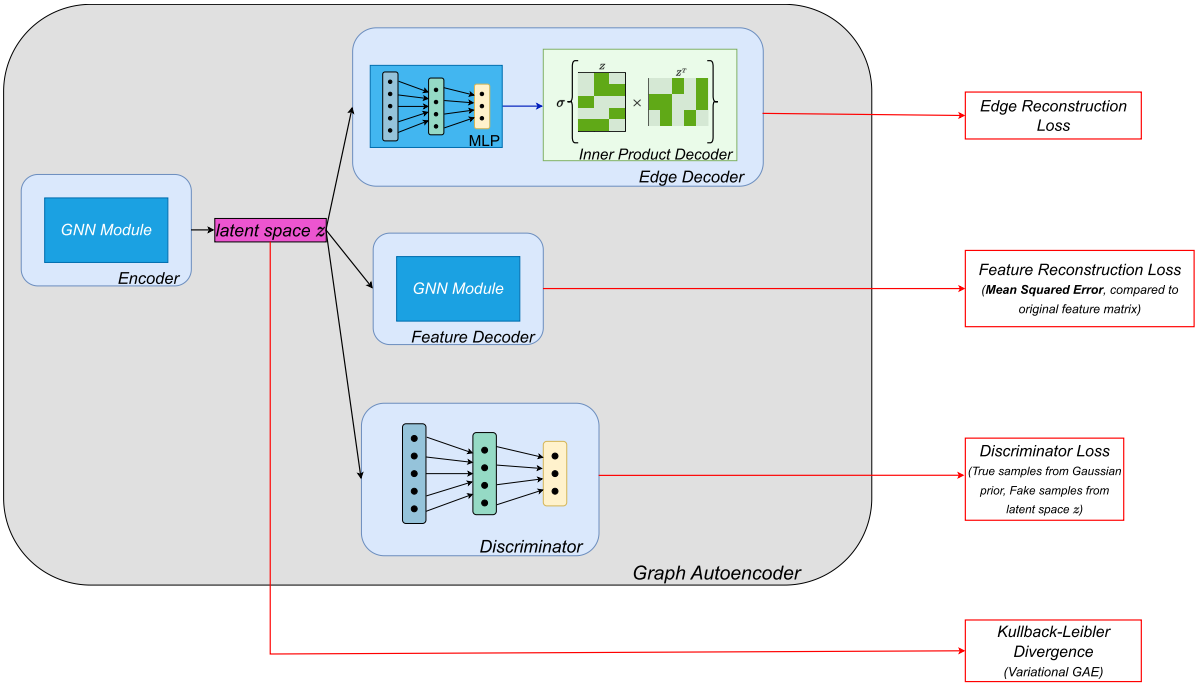


Figure 1.2.8: Αρχιτεκτονική *Combined ARVGA MLP*

1.3 Πειραματικό Μέρος

1.3.1 Σύνολο Δεδομένων

Το σύνολο δεδομένων που θα χρησιμοποιηθεί, είναι το **Visual Genome**[58], ένα ολοκληρωμένο σύνολο δεδομένων για την εκπαίδευση και τη συγκριτική αξιολόγηση της επόμενης γενιάς μοντέλων υπολογιστικής όρασης, που χρησιμοποιούν πληροφορίες δομημένες σε γράφους. Περιέχει 108.249 εικόνες από τη διασταύρωση των συνόλων δεδομένων YFCC100M[108] και MS-COCO[68]. Κάθε εικόνα συνοδεύεται από τον αντίστοιχο *Γράφο Σκηνής*, όπως εξηγείται στην ενότητα 1.1.4. Ένα παράδειγμα ενός ζεύγους Εικόνας-Γράφου_Σκηνής φαίνεται στην εικόνα 1.3.1.

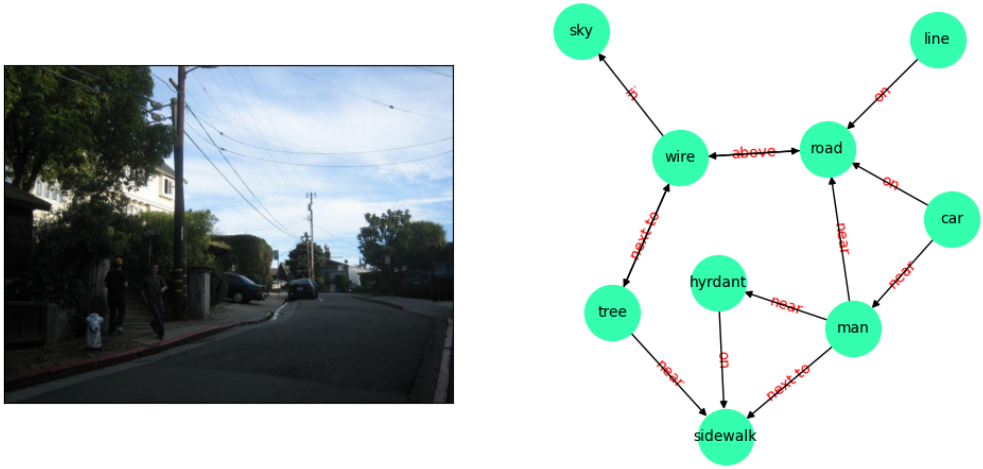


Figure 1.3.1: Παράδειγμα ενός Γράφου Σκηνής, μαζί με την αντίστοιχη εικόνα που περιγράφει.

Στο πλαίσιο του προβλήματος της Ανάκτησης Πληροφορίας σε γράφους σκηνής, ο **διαχωρισμός Ερωτήσεων-Απαντήσεων** είναι απαραίτητος. Σε αντίθεση με την παραδοσιακή επιβλεπόμενη και μη-επιβλεπόμενη μηχανική μάθηση, αυτός ο διαχωρισμός περιλαμβάνει ένα σύνολο Ερωτημάτων (*Query Set*) και ένα σύνολο Απαντήσεων (*Answer Set*) από γραφήματα σκηνών. Τα μοντέλα πρέπει να επιστρέψουν μία τελική κατάταξη των γράφων Απάντησης, με βάση την ομοιότητά τους με τους γράφους Ερωτήματος. Για τη διαχείριση του υπολογιστικού κόστους, επιλέχθηκαν 1000 γραφήματα σκηνών, με 960 στο Answer Set και 40 στο Query Set. Ο αλγόριθμος εύρεσης Απόστασης Επεξεργασίας Γράφων (ΑΕΓ), που χρησιμοποιείται για τον υπολογισμό της ομοιότητας, είναι υπολογιστικά δαπανηρός και κλιμακώνεται τετραγωνικά με τον αριθμό των γράφων (εφόσον εν τέλει, πρέπει να δημιουργήσουμε δυάδες γράφων). Η επιλογή αυτή αποτρέπει τους υπερβολικούς χρόνους υπολογισμού. Επιπλέον, ένα μικρότερο σύνολο Ερωτημάτων με μεγαλύτερο σύνολο Απαντήσεων εξασφαλίζει ουσιαστικές κατατάξεις για τις μετρικές αξιολόγησης. Διευκολύνει επίσης την ποιοτική αξιολόγηση με τη χρήση αντίστοιχων εικόνων, καθιστώντας την πιο αποτελεσματική για την αξιολόγηση της απόδοσης του μοντέλου σε σύγκριση με τα τυχαία ζεύγη ερωτήσεων-απαντήσεων.

Για την κατασκευή του συνόλου Ερωτημάτων και του συνόλου Απαντήσεων από το Visual Genome, αρχικά επιλέχθηκαν 1000 τυχαίοι γράφοι με ελάχιστους περιορισμούς στον αριθμό των κόμβων και των ακμών, ώστε να αποφευχθούν οι υπερβολικά απλοί γράφοι. Ωστόσο, αυτό οδήγησε σε **πολλούς γράφους με απομονωμένους κόμβους**, περιορίζοντας τη χρησιμότητα των μοντέλων που βασίζονται σε γράφους, όπως οι Πυρήνες Γράφων και τα μοντέλα ΝΔΓ. Για να αντιμετωπιστεί αυτό, δημιουργήθηκε ένα νέο υποσύνολο που ονομάζεται ***"Dense Subset"*** (Πυκνό Υποσύνολο). Αυτό το σύνολο περιλάμβανε γράφους με αυστηρότερους περιορισμούς στον αριθμό των κόμβων, των ακμών και της πυκνότητας, εξασφαλίζοντας καλύτερη αξιοποίηση των δυνατοτήτων *Μετάβασης Μηνυμάτων* των ΝΔΓ. Ο διαχωρισμός Query-Answer Set περιελάμβανε την επιλογή 40 τυχαίων γραφημάτων για το Query Set, ενώ τα υπόλοιπα αποτελούσαν το Answer Set, τόσο στο Random όσο και στο Dense υποσύνολο.

Η **προεπεξεργασία των Γράφων Σκηνών** γίνεται σε δύο στάδια. Στο πρώτο στάδιο, εφαρμόζονται οι παρακάτω λειτουργίες σε όλους τους γράφους:

- **Αφαίρεση Χαρακτηριστικών:** Τα χαρακτηριστικά που περιγράφουν τα αντικείμενα στο αρχικό σύνολο δεδομένων (*"attributes"*) παραλείπονται, με σκοπό να αποφευχθεί ο πλεονασμός και να αποτραπεί η εστίαση στα χαρακτηριστικά και όχι στο σημασιολογικό περιεχόμενο κατά τον υπολογισμό ομοιότητας γράφων.
- **Σχέσεις ως Ακμές:** Οι σχέσεις αναπαρίστανται ως ακμές και όχι ως ξεχωριστοί κόμβοι που συνδέονται με υποκείμενα και αντικείμενα, με σκοπό να διατηρηθεί η συνοχή, λαμβάνοντας υπόψη τις προκλήσεις κατά τη σύγκριση αντικειμένων και σχέσεων με τη χρήση της ιεραρχίας του WordNet.
- **Αφαίρεση Τύπων Σχέσεων:** Εν τέλει, δεν χρησιμοποιούμε τους τύπους/κλάσεις σχέσεων στους αρχικούς γράφους σκηνών, χρησιμοποιώντας αντ' αυτών κατευθυνόμενες ακμές (χωρίς έξτρα πληροφορία για τύπο/κλάσεις) για να μειωθεί ο πλεονασμός, δεδομένου ότι οι περισσότεροι Πυρήνες Γράφων και οι μονάδες ΝΔΓ δεν αξιοποιούν αποτελεσματικά τα χαρακτηριστικά ακμών.

Στο δεύτερο στάδιο, αντιμετωπίζεται η αναπαράσταση των χαρακτηριστικών των κόμβων, με αποτέλεσμα την δημιουργία τριών συνόλων δεδομένων:

1. **Συμβολοσειρές για Χαρακτηριστικά Κόμβων:** Τα χαρακτηριστικά κόμβων αναπαρίστανται ως απλές συμβολοσειρές (*strings*), κατάλληλες για Πυρήνες Γράφων, καθώς λειτουργούν με ετικέτες κόμβων και δεν μπορούν να αξιοποιήσουν αποτελεσματικά άλλες μορφές πληροφοριών.
2. **Ενθέσεις Λέξεων για Χαρακτηριστικά Κόμβων:** Οι μονάδες ΝΔΓ απαιτούν αριθμητικά χαρακτηριστικά κόμβων, στη μορφή ενός Πίνακα Χαρακτηριστικών (*Feature Matrix*). Οι ενθέσεις λέξεων GloVe[86] χρησιμοποιούνται για τη μετατροπή συμβολοσειρών σε ενθετικές αναπαραστάσεις, με μέγεθος 100, που χρησιμοποιούνται ως είσοδος για τα μοντέλα ΝΔΓ.
3. **Synsets για Χαρακτηριστικά Κόμβων:** Αυτό το σύνολο δεδομένων αξιοποιεί την ιεραρχία *"is-a"* που παρέχεται από το WordNet[77] για τον υπολογισμό της Βασικής Αλήθειας με τον αλγόριθμο εύρεσης ΑΕΓ. Η συγκεκριμένη ιεραρχία επιτρέπει τη μέτρηση ομοιότητας μεταξύ διαφόρων *Εννοιών* (*"Synsets"*), όπου στο δυγκεκριμένο σύνολο δεδομένων, αποτελούν τους κόμβοους των γράφων.

1.3.2 Βασική Αλήθεια

Οι βαθμολογίες ομοιότητας και οι κατατάξεις της **Βασικής Αλήθειας** υπολογίζονται με τον αλγόριθμο εύρεσης Απόστασης Επεξεργασίας Γράφων (ΑΕΓ), με προτίμηση στους προσεγγιστικούς αλγορίθμους ΑΕΓ λόγω της υπολογιστικής τους απόδοσης. Ο αλγόριθμος που επιλέχθηκε για την παρούσα διατριβή είναι η παραλλαγή *Bipartite Matching* του ΑΕΓ, η οποία εστιάζει κυρίως στις πληροφορίες κόμβων, επιτυγχάνοντας ισορροπία μεταξύ υπολογιστικού κόστους και ποιότητας προσέγγισης (όπως εξηγήθηκε και στην Ενότητα 1.1.2).

Η **Ιεραρχία *"is-a"* των Ουσιαστικών του WordNet** είναι ένα κρίσιμο στοιχείο στον υπολογισμό του ΑΕΓ. Αυτή η ιεραρχία καθιερώνει σχέσεις Υπερωνύμων/Υπωνύμων μεταξύ ουσιαστικών, προσφέροντας έναν δομημένο τρόπο σύγκρισης αντικειμένων και προσδιορισμού των ομοιοτήτων τους. Για την αξιοποίηση αυτής της ιεραρχίας χρησιμοποιείται το *πακέτο NLTK της Python* [3], το οποίο παρέχει ένα API για το WordNet, συμπεριλαμβανομένης της ιεραρχίας ουσιαστικών. Η συνάρτηση `path_similarity` από το NLTK υπολογίζει βαθμολογίες ομοιότητας με βάση τη συντομότερη διαδρομή που συνδέει τις έννοιες στην ταξινομία "is-a", επιτρέποντας τον υπολογισμό του κόστους εισαγωγής/διαγραφής/αντικατάστασης κόμβων για τον αλγόριθμο Bipartite Matching.

Η διαδικασία περιλαμβάνει τον υπολογισμό βαθμολογιών ομοιότητας για όλα τα πιθανά ζεύγη κόμβων από τους γράφους G_i και G_j" με τη χρήση της `path_similarity` και στη συνέχεια τη χρήση αυτών των βαθμολογιών για την εξαγωγή του κόστους αντικατάστασης κόμβων για τη συνάρτηση υπολογισμού του ΑΕΓ. Το κόστος εισαγωγής και διαγραφής κόμβων προσδιορίζεται με την εύρεση της απόστασης μεταξύ κάθε κόμβου και του κόμβου *"Entity"* (*"οντότητα"*), ο οποίος χρησιμεύει ως κόμβος-ρίζα της ιεραρχίας. Η διαδικασία αυτή επαναλαμβάνεται για κάθε πιθανό ζεύγος μεταξύ του Query Set και του Answer Set, τόσο για το υποσύνολο "Random" όσο και για το "Dense", με το αποτέλεσμα να είναι οι τιμές Κόστους/Απόστασης Γράφων που υπολογίζει ο αλγόριθμος ΑΕΓ. Ο χρόνος εκτέλεσης για το σύνολο δεδομένων "Random" είναι περίπου 5 ώρες, ενώ το σύνολο δεδομένων "Dense" χρειάζεται περίπου 1 ώρα για να ολοκληρωθεί.

1.3.3 Μετρικές

Τώρα θα παρουσιάσουμε τις τρεις μετρικές που θα χρησιμοποιηθούν για την ποσοτική αξιολόγηση και εκτίμηση της απόδοσης των Πυρήνων Γράφων και των μοντέλων ΝΔΓ: *NDCG*, *MAP* και *MRR*. Θα πρέπει να αναφερθεί ότι και οι τρεις μετρικές, είναι ειδικά σχεδιασμένες για να **μετρήσουν την απόδοση ενός Συστήματος Ανάκτησης Πληροφορίας**. Αυτό σημαίνει ότι οι είσοδοι περιλαμβάνουν (τουλάχιστον), μια "Αληθινή" κατάταξη και μια "Προβλεπόμενη" κατάταξη, δεδομένου του ίδιου αντικειμένου Ερωτήματος. Στην περίπτωσή μας, η "Αληθινή" κατάταξη είναι τα πρώτα 50 αντικείμενα που επιστρέφονται από τον αλγόριθμο ΑΕΓ (Βασική Αλήθεια), και η "Προβλεπόμενη" κατάταξη είναι τα αντικείμενα που επιστρέφονται από το μοντέλο που θέλουμε να ελέγξουμε. Συγκεκριμένα, αξιολογούμε όλα τα συστήματα λαμβάνοντας υπόψη τα πρώτα 10 αντικείμενα που επιστρέφονται. Παρακάτω παρουσιάζουμε πιο λεπτομερώς τις τρεις μετρικές:

- **Normalized Discounted Cumulative Gain (NDCG):** Η NDCG είναι μια ευρέως χρησιμοποιούμενη μετρική που αξιολογεί την ποιότητα των αποτελεσμάτων αναζήτησης με κατάταξη. Λαμβάνει υπόψη τόσο τη συνάφεια όσο και τη θέση κατάταξης των αντικειμένων στην τελική προβλεπόμενη κατάταξη. Η NDCG αποδίδει υψηλότερες βαθμολογίες στα αντικείμενα που είναι και πολύ συναφή και εμφανίζονται στην κορυφή της κατάταξης. Η μετρική λαμβάνει υπόψη τη φθίνουσα απόδοση της συνάφειας καθώς μετακινείστε προς τα κάτω στη κατάταξη, παρέχοντας μια πιο ρεαλιστική αξιολόγηση της ικανοποίησης του χρήστη. Οι τιμές NDCG κυμαίνονται από 0 έως 1, με υψηλότερες βαθμολογίες να υποδηλώνουν καλύτερη απόδοση. Συνολικά, η NDCG είναι η πιο αντιπροσωπευτική μετρική για συστήματα Ανάκτησης Πληροφορίας, καθώς συνυπολογίζει τις θέση και τις βαθμολογίες συνάφειας για όλα τα αντικείμενα στην προβλεπόμενη κατάταξη. Το μοναδικό αρνητικό, είναι ότι δεν είναι μία άμεσα ερμηνεύσιμη μετρική, όπως οι άλλες δύο που χρησιμοποιούμε. Η NDCG υπολογίζεται ως εξής:

$$DCG@K = \sum_{k=1}^{K} \frac{rel_k}{log_2\left(1+k\right)}$$

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

όπου rel_k είναι οι βαθμολογίες συνάφειας, και $IDCG$ είναι το ιδεατό DCG σκορ, αν αυτό υπολογιστή στην σωστή κατάταξη (αυτό υπολογίζεται, με σκοπό να είναι κανονικοποιημένο το NDCG). Όπως αναφέραμε και προηγουμένως, το K το θέτουμε ίσο με 10 στα πειράματα.

- **Mean Average Precision (MAP):** Η MAP είναι μια άλλη σημαντική μετρική για την αξιολόγηση της ανάκτησης πληροφοριών, ιδίως σε σενάρια όπου υπάρχουν πολλά σχετικά αντικείμενα για ένα ερώτημα. Υπολογίζει τη μέση ακρίβεια (*precision*) για κάθε ερώτημα και στη συνέχεια υπολογίζει τη μέση τιμή για όλα τα ερωτήματα. Η ακρίβεια μετρά το ποσοστό των σχετικών αντικειμένων στα αποτελέσματα που βρίσκονται στην κορυφή της κατάταξης και η MAP λαμβάνει υπόψη την ακρίβεια σε πολλαπλά σημεία της κατάταξης, δίνοντας μεγαλύτερη βαρύτητα στα έγγραφα που βρίσκονται σε υψηλότερες θέσεις. Οι τιμές MAP κυμαίνονται από 0 έως 1, με υψηλότερες βαθμολογίες να υποδηλώνουν καλύτερη απόδοση ανάκτησης. Σε αντίθεση με την NDCG, η MAP είναι μία ερμηνεύσιμη μετρική, αφού βασίζεται στον ορισμό του κλασικού *Precision*. Η MAP υπολογγίζεται ως εξής:

$$Precision@k = \frac{\text{Πλήθος συναφών αντικειμένων στα top } k}{k}$$

$$AP@K = \frac{\sum_{k=1}^{K} (Precision@k * rel_k)}{\text{πλήθος συναφών αποτελεσμάτων}}$$

$$MAP@K = \frac{1}{Q} \sum_{q=1}^{Q} AP@K_q$$

όπου το rel_k είναι 1 αν το αντικείμενο στην θέση k είναι συναφές, αλλιώς είναι 0, και το Q είναι το συνολικό πλήθος των ερωτημάτων.

- **Mean Reciprocal Rank (MRR):** Η MRR είναι μια μετρική που χρησιμοποιείται συνήθως για εργασίες όπως η απάντηση-ερωτήσεων και ο υπολογισμός κατάταξη, όπου υπάρχει μόνο μία σωστή απάντηση ή αντικείμενο. Αξιολογεί πόσο γρήγορα το σύστημα ανακτά το πρώτο σχετικό αποτέλεσμα. Η MRR υπολογίζει την αμοιβαία κατάταξη του πρώτου σχετικού αντικειμένου για κάθε ερώτημα και στη συνέχεια υπολογίζει τη μέση αμοιβαία κατάταξη για όλα τα ερωτήματα. Οι τιμές MRR κυμαίνονται από 0 έως 1, με υψηλότερες βαθμολογίες να υποδηλώνουν καλύτερη απόδοση. Η MRR είναι ιδιαίτερα χρήσιμη όταν η έμφαση δίνεται στην ταχύτητα εύρεσης σχετικών πληροφοριών. Το τελικό σκορ MRR είναι άμεσα ερμηνεύσιμο, και υπολογίζεται ως εξής:

$$MRR@K = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{rank_q}$$

όπου το Q είναι το πλήθος των ερωτημάτων, και το $rank_q$ είναι η θέση του πρώτου συναφούς αντικειμένου στην προβλεπόμενη κατάταξη. Εδώ, το $@K$ είναι ένας περιορισμός, ώστε να εξετάζονται μόνο τα πρώτα K αντικείμενα με την υψηλότερη κατάταξη.

1.3.4 Λεπτομέρειες Μοντέλων

Παράμετροι Πυρήνων Γράφων

Νωρίτερα, παρουσιάσαμε τους πέντε πυρήνες γράφων που θα χρησιμοποιηθούν για τα πειράματα, ως βασική τεχνική για την αντιμετώπιση της ομοιότητας γράφων. Συγκεκριμένα, η βιβλιοθήκη Python GraKel[103], περιέχει όλες τις υλοποιήσεις των Πυρήνων Γράφων που χρησιμοποιήθηκαν για την παρούσα διατριβή. Στη συνέχεια θα παραθέσουμε τις παραμέτρους για κάθε έναν από τους πέντε Πυρήνες που χρησιμοποιήθηκαν:

- **Shortest Path Kernel:** Αυτός ο πυρήνας δεν λαμβάνει συγκεκριμένες παραμέτρους που να επηρεάζουν το στάδιο του υπολογισμού, εκτός από τον υποκείμενο αλγόριθμο που βρίσκει τα συντομότερα μονοπάτια. Αυτός ο αλγόριθμος μπορεί να οριστεί σε *"Dijkstra"*, *"Floyd-Warshall"* ή *"Auto"* (αποφασίζει ποιος είναι ταχύτερος ανάλογα με τους γράφους εισόδου). Εμείς το θέτουμε σε *"Auto"*.
- **Weisfeiler-Lehman Kernel:** Εδώ, οι δύο σημαντικές παράμετροι είναι, ο αριθμός των επαναλήψεων του Weisfeiler-Lehman Test Propagation και ο υποκείμενος πυρήνας γράφων που εφαρμόζεται στους μετασχηματισμένους γράφους. Ορίζουμε τον αριθμό των επαναλήψεων σε 20, και τον υποκείμενο πυρήνα γράφων σε *Vertex Histogram*.
- **Neighborhood Hash Kernel:** Οι δύο παράμετροι σε αυτή την περίπτωση είναι, ο μέγιστος αριθμός Neighborhood Hash (δηλαδή οι επαναλήψεις), και το μέγεθος Byte των Hashes των κόμβων. Το πρώτο ορίστηκε σε 3 και το δεύτερο σε 2.
- **Random Walk Kernel:** Για αυτόν τον πυρήνα, η κύρια παράμετρος είναι η λ, για τον υπολογισμό της γεωμετρικής σειράς, η οποία ορίστηκε σε 0.1.
- **Graphlet Sampling Kernel:** Για αυτόν τον πυρήνα, η κύρια παράμετρος, είναι το μέγιστο μέγεθος των Graphlets που θα χρησιμοποιηθούν. Το ορίσαμε σε 5.

Transductive Μάθηση

Στο πλαίσιο της μηχανικής μάθησης σε γράφους, είναι σημαντικό να διακρίνουμε δύο βασικές προσεγγίσεις μάθησης/συμπερασμού: Transductive (*Μεταγωγική*) και Inductive(*Επαγωγική*). Η Inductive Μάθηση περιλαμβάνει την εκπαίδευση ενός μοντέλου σε ένα Train Set, την επικύρωσή του για να αποφευχθεί η υπερ-εκπαίδευση και, στη συνέχεια, τη δοκιμή του σε ένα αθέατο Test Set για τη δημιουργία ενός γενικευμένου μοντέλου. Από την άλλη πλευρά, η **Transductive Μάθηση** επικεντρώνεται στην πραγματοποίηση προβλέψεων για ένα συγκεκριμένο σύνολο δειγμάτων ελέγχου, χωρίς να στοχεύει στην εξαγωγή γενικευμένων κανόνων. Σε αυτή την προσέγγιση, το μοντέλο προσαρμόζει τις προβλέψεις του στα μοναδικά χαρακτηριστικά των δεδομένων ελέγχου, δίνοντας προτεραιότητα στην ακρίβεια για αυτές τις περιπτώσεις.

Η Transductive Μάθηση είναι ιδιαίτερα διαδεδομένη στη μηχανική μάθηση σε γράφους λόγω προβλημάτων όπως η ταξινόμηση κόμβων και η πρόβλεψη συνδέσμων, όπου όλα τα σχετικά δεδομένα είναι διαθέσιμα κατά την εκπαίδευση. Έννοιες όπως η υπερ-εκπαίδευση, η γενίκευση και η επικύρωση που ισχύουν για την Inductive μάθηση μπορεί να μην είναι τόσο σχετικές σε Transductive προβλήματα. Σε αυτή τη διατριβή, η εστίαση είναι στην τυποποίηση του προβλήματος της Ομοιότητας Γραφημάτων ως Transductive πρόβλημα, ευθυγραμμιζόμενη με τις κοινές πρακτικές στη Μηχανική Μάθηση σε Γραφήματα. Τα μοντέλα ΝΔΓ εκπαιδεύονται τόσο στο σύνολο ερωτήσεων (Query) όσο και στο σύνολο απαντήσεων (Answer) για την αποτελεσματική βελτιστοποίηση των αναπαραστάσεων γράφων. Η επέκταση αυτής της εργασίας σε Inductive είναι δυνατή, αλλά ξεφεύγει από το πεδίο εφαρμογής της διατριβής, αφήνοντάς την ως πιθανό πεδίο για μελλοντική εργασία.

Υπερπαράμετροι ΝΔΓ

Οι υπερπαράμετροι και οι λεπτομέρειες εκπαίδευσης/παροχής συμπερασμάτων για τα μοντέλα ΝΔΓ μπορούν να συνοψιστούν ως εξής. Όλα τα μοντέλα χρησιμοποίησαν τον βελτιστοποιητή **AdamW** με προεπιλεγμένες ρυθμίσεις ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $weight\_decay = 0.01$). Τα μοντέλα με Διευκρινιστή χρησιμοποίησαν έναν ξεχωριστό βελτιστοποιητή AdamW για την απώλεια του Διευκρινιστή. Τα αρχικά βήματα μάθησης ορίστηκαν αρχικά σε 0.001 για τα μοντέλα με Διευκρινιστή και 0.01 για τα υπόλοιπα. Ένας χρονοπρογραμματιστής `Reduce_LR_On_Plateau` χρησιμοποιήθηκε παράλληλα με τον βελτιστοποιητή, στις αρχιτεκτονικές χωρίς Διευκρινιστή. Το μέγεθος της παρτίδας (Batch Size) ορίστηκε σε 16 για όλα τα μοντέλα.

Κατά τη διάρκεια του συμπερασμού των μοντέλων, κάθε ένας από τους 1000 γράφους στα σύνολα δεδομένων περνούσε από τον Κωδικοποιητή, και η τελική ενθετική αναπαράσταση γράφου δημιουργούνταν με το άθροισμα των ενθέσεων κόμβων. Η απόσταση $\mathbf{L_2}$ επιλέχθηκε για τη σύγκριση των Ενθέσεων Γράφων, καθώς υπερείχε της Απόστασης Συνημιτόνου, ενδεχομένως λόγω της χρήσης της απώλειας *Mean Squared Error*, η οποία ευθυγραμμίζεται καλύτερα με την ευκλείδεια απόσταση των χαρακτηριστικών.

Οι υπερπαράμετροι, όπως **οι Εποχές, το Μέγεθος Ενθετικών Αναπαραστάσεων και οι Κεφαλές Προσοχής** (για το GAT και το GATv2) διέφεραν μεταξύ των μοντέλων, και αυτές ήταν οι κύριες

υπερπαράμετροι που χρειάστηκε να ρυθμίσουμε. Οι Κεφαλές Προσοχής με τιμές 2, 4 και 8 έδωσαν γενικά τα καλύτερα αποτελέσματα. Το Μέγεθος Ενθετικών Αναπαραστάσεων είχε διαφορετικές βέλτιστες τιμές για απλούστερα και πιο σύνθετα μοντέλα, με τα τελευταία να αποδίδουν καλά με μεγαλύτερες διαστάσεις. Οι εποχές κυμαίνονταν από 10 έως 200 κατά τη διάρκεια του συντονισμού. Παρακάτω, στους Πίνακες 1.1 και 1.2, παρουσιάζουμε όλες τις τελικές υπερπαραμέτρους για τα μοντέλα με την καλύτερη απόδοση. Εδώ, με τον όρο "καλύτερη απόδοση", χρησιμοποιήσαμε τη μέση τιμή των τριών μετρικών

Επίσης, θα πρέπει να σημειωθεί ότι δεν δοκιμάσαμε κάθε αρχιτεκτονική ΝΔΓ στο σύνολο δεδομένων Dense, δεδομένου ότι ο σκοπός αυτού του συνόλου δεδομένων δεν ήταν να συγκρίνουμε τις μετρικές με το σύνολο δεδομένων Random, αλλά να δούμε αν τα μοντέλα ΝΔΓ θα ξεπεράσουν τους Πυρήνες Γράφων, δεδομένου ότι οι γράφοι είναι πιο πυκνοί. Γι' αυτό δοκιμάζουμε μόνο τα δύο μοντέλα με τις καλύτερες επιδόσεις (Combined-ARVGA και Combined-ARVGA-MLP), μαζί με τα VGAE και ARVGA για να αξιολογήσουμε το κέρδος απόδοσης που έχει να προσφέρει η ανταγωνιστική εκπαίδευση, με τα πυκνότερα γραφήματα.

Model	GNN Module	Epochs	Embedding Size	Attention Heads
GAE	GCN	10	32	-
	GAT	20	32	8
	GATv2	70	32	4
	GIN	120	32	-
VGAE	GCN	10	32	-
	GAT	20	32	8
	GATv2	70	32	4
	GIN	120	32	-
Feature-VGAE	GCN	40	32	-
	GAT	170	32	4
	GATv2	50	32	4
	GIN	120	32	-
Combined	GCN	20	32	-
	GAT	170	32	4
	GATv2	160	32	8
	GIN	60	32	-
ARVGA	GCN	120	64	-
	GAT	20	32	4
	GATv2	140	32	4
	GIN	10	32	-
Combined-ARVGA	GCN	30	300	-
	GAT	190	256	4
	GATv2	150	64	8
	GIN	50	64	-
Combined-ARVGA-MLP	GCN	90	300	-
	GAT	120	300	4
	GATv2	170	300	4
	GIN	160	300	-

Table 1.1: Βέλτιστες Υπερπαράμετροι των μοντέλων ΝΔΓ, για το Random Subset

1.3.5 Αποτελέσματα

Ποσοτική Ανάλυση

Αρχικά, θα παρουσιάσουμε την τελική βαθμολογία MAP, MRR και NDCG, που επιτεύχθηκε από τους Πυρήνες Γράφων και τα μοντέλα ΝΔΓ, στο Random Subset. Κάθε συγκεκριμένο μοντέλο ΝΔΓ, αντιστοιχεί στο βέλτιστο, με τις υπερπαραμέτρους που παρουσιάζονται στον Πίνακα 1.1. Οι μετρικές αξιολόγησης παρουσιάζονται στον πίνακα 1.3.

Model	GNN Module	Epochs	Embedding Size	Attention Heads
VGAE	GCN	80	300	-
	GAT	60	300	4
	GATv2	70	300	4
	GIN	80	300	-
ARVGA	GCN	20	200	-
	GAT	140	200	4
	GATv2	40	200	4
	GIN	180	300	-
Combined-ARVGA	GCN	150	300	-
	GAT	160	300	4
	GATv2	80	300	4
	GIN	40	300	-
Combined-ARVGA-MLP	GCN	150	300	-
	GAT	130	300	4
	GATv2	160	300	4
	GIN	10	300	-

Table 1.2: Βέλτιστες Υπερπαράμετροι των μοντέλων ΝΔΓ, για το Dense Subset

Το πρώτο πράγμα που παρατηρούμε στους **Πυρήνες Γράφων**, είναι η υπεροχή του Weisfeiler-Lehman και του Neighborhood Hash Πυρήνα. Συγκεκριμένα, οι άλλοι τρεις πυρήνες πετυχαίνουν αρκετά μικρότερα σκορ, και στις τρεις μετρικές. Αυτό μπορεί να εξηγηθεί αν το συσχετίσουμε με την κύρια ιδέα του κάθε πυρήνα. Συγκεκριμένα, οι τρεις Πυρήνες που δεν είχαν τόσο καλές επιδόσεις, βασίζονται όλοι σε θεμελιώδεις δομικές ιδιότητες ενός γράφου (συντομότερα μονοπάτια, τυχαίοι περίπατοι) και στην εμφάνιση πρωτοτύπων (graphlets). Και τα τρία αυτά, αναμένεται να μην είναι επαρκώς εφαρμόσιμα στην περίπτωση των Scene Graphs, και ειδικά στην περίπτωση του Random Subset. Συγκεκριμένα στο Random Subset, η τάξη των γράφων παρουσιάζει σημαντική διακύμανση. Αυτή η διακύμανση, μαζί με τη γενικά χαμηλή πυκνότητά τους, δεν παρέχει χρήσιμες πληροφορίες στους Πυρήνες που βασίζονται σε μεγάλο βαθμό στη δομή. Από την άλλη πλευρά, οι πυρήνες Weisfeiler-Lehman και Neighborhood Hash, βασίζονται στις ετικέτες των γραφημάτων και υλοποιούν μια διαδικασία επαναεπισημείωσης των κόμβων και ενημέρωσης των πληροφοριών. Αυτό φαίνεται να αποδεικνύεται μια επιτυχημένη ιδέα, όταν πρόκειται να ληφθούν υπόψη όσο το δυνατόν περισσότερες πληροφορίες γράφου, προκειμένου να συγκριθούν ζεύγη γράφων και να εξαχθεί μια βαθμολογία ομοιότητας. Επομένως, μπορούμε να αναγνωρίσουμε την εκφραστική δύναμη της τεχνικής **Message-Passing**, επειδή οι δύο πυρήνες που πέτυχαν τα καλύτερα αποτελέσματα, ουσιαστικά υλοποιούν μια παραλλαγή αυτής της μεθόδου.

Όσον αφορά τα **Μοντέλα ΝΔΓ**, η αρχική παρατήρηση είναι ότι οι απλές αρχιτεκτονικές των **GAE** και **VGAE**, δεν επαρκούν για να ξεπεράσουν τις επιδόσεις των βασικών πυρήνων γράφων, σε καμία από τις τρεις μετρικές. Μόνο τα αποτελέσματα για το MRR είναι συγκρίσιμα, όπου το **VGAE-GAT** πέτυχε 1% χαμηλότερη βαθμολογία. Οι άλλες δύο μετρικές είναι 3% και 7% χαμηλότερες συγκριτικα με τον καλύτερο πυρήνα. Επίσης, σε αυτό το σημείο, αξίζει να αναφέρουμε, ότι δεν συμπεριλάβαμε άλλες αρχιτεκτονικές χωρίς το *Variational Loss* στο λανθάνων χώρο, επειδή έγινε γρήγορα φανερό, ότι δεν παρήγαγε καλύτερα αποτελέσματα.

Προχωρώντας σε πιο σύνθετες αρχιτεκτονικές, βλέπουμε ότι η αρχιτεκτονική **Feature-VGAE** και η αρχιτεκτονική **Combined** (οι οποίες είναι οι πρώτες που υλοποιούν και έναν αποκωδικοποιητή χαρακτηριστικών), παρέχουν ήδη μία ώθηση και στις τρεις μετρικές. Συγκεκριμένα, εξετάζοντας την NDCG (την πιο αντιπροσωπευτική μετρική μεταξύ των τριών), και οι δύο είναι υψηλότερες σε σύγκριση με τα δύο αρχικά μοντέλα. Ακόμη και σε σύγκριση με το **ARVGA**, μπορούμε να δούμε ότι τα μοντέλα με τον Αποκωδικοποιητή Χαρακτηριστικών σημειώνουν καλύτερη βαθμολογία στο NDCG.

Η μεγαλύτερη αύξηση της απόδοσης παρατηρείται στις δύο τελευταίες αρχιτεκτονικές, όπου η ανταγωνιστική εκπαίδευση από το μοντέλο ARVGA, συνδυάζεται με έναν Αποκωδικοποιητή Χαρακτηριστικών και έναν Αποκωδικοποιητή Ακμών (που χρησιμοποιεί και μία μονάδα ΝΔΓ). Οι δύο αρχιτεκτονικές **Combined-ARVGA** έχουν τις καλύτερες βαθμολογίες μεταξύ των ΝΔΓ, και στις τρεις μετρικές. Συγκεκριμένα, η έκδοση GCN της Combined-ARVGA σημειώνει την καλύτερη βαθμολογία NDCG και MAP σε όλα τα μοντέλα ΝΔΓ.

Αλλά μπορούμε ακόμα να δούμε, ότι οι Πυρήνες Γράφων εξακολουθούν να είναι καλύτεροι, όταν πρόκειται για

		MAP@10	MRR	NDCG@10
Shortest Path Kernel		0.4836	0.5377	0.2027
Random Walk Kernel		0.0147	0.0560	0.0012
Weisfeiler-Lehman Kernel		**0.6264**	0.7119	0.2790
Graphlet Sampling Kernel		0.1658	0.1741	0.0248
Neighborhood Hash Kernel		0.6257	**0.7141**	**0.2956**
GAE	GCN	0.5270	**0.6878**	**0.2132**
	GAT	0.4946	0.5747	0.1975
	GATv2	**0.5343**	0.6460	0.1907
	GIN	0.4244	0.4623	0.1631
VGAE	GCN	0.5302	0.6660	0.2177
	GAT	**0.5914**	**0.7061**	**0.2193**
	GATv2	0.5237	0.5859	0.1761
	GIN	0.4275	0.5120	0.1876
Feature-VGAE	GCN	0.5524	0.6567	0.2240
	GAT	0.5646	0.6425	0.1994
	GATv2	**0.5958**	**0.6864**	**0.2327**
	GIN	0.5007	0.5762	0.1953
Combined	GCN	**0.5676**	**0.7017**	0.2287
	GAT	0.5405	0.6428	0.2350
	GATv2	0.5598	0.6679	**0.2446**
	GIN	0.4335	0.4828	0.1215
ARVGA	GCN	**0.6104**	**0.7298**	**0.2287**
	GAT	0.5844	0.6680	0.2183
	GATv2	0.5436	0.6860	0.1998
	GIN	0.5364	0.5935	0.2238
Combined-ARVGA	GCN	**0.6353**	0.7455	**0.2796**
	GAT	0.6137	**0.7503**	0.2662
	GATv2	0.6219	0.7454	0.2416
	GIN	0.5320	0.6318	0.2448
Combined-ARVGA-MLP	GCN	**0.6277**	0.7162	0.2472
	GAT	0.6117	**0.7533**	0.2542
	GATv2	0.6107	0.7403	**0.2606**
	GIN	0.5683	0.6423	0.2585

Table 1.3: Τελικές Μετρικές για τους Πυρήνες Γράφων και τα βέλτιστα μοντέλα ΝΔΓ, στο Random Subset. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε αρχιτεκτονική.

NDCG. Συγκεκριμένα, ο πυρήνας **Neighborhood Hash** σημειώνει περίπου 2% μεγαλύτερο σκορ στο NDCG από το καλύτερο μοντέλο ΝΔΓ, αλλά τα πάει χειρότερα στο MAP και στο MRR. Αν θυμηθούμε, η ερμηνεία του MAP, είναι ότι μας λέει πόσα σχετικά αντικείμενα έχουμε στην προβλεπόμενη κατάταξη, και το MRR λαμβάνει υπόψη μόνο τη θέση του πρώτου σχετικού αντικειμένου που ανακτήθηκε. Έτσι, αν εξετάσουμε αυτές τις δύο συγκεκριμένες πτυχές της Ανάκτησης Πληροφορίας, τότε τα πιο σύνθετα μοντέλα ΝΔΓ είναι καλύτερα από τους Πυρήνες Γράφων. Αλλά από την άλλη πλευρά, το NDCG είναι η μόνη μετρική που, ουσιαστικά, λαμβάνει υπόψη τα πάντα από την προβλεπόμενη κατάταξη (τη θέση των αντικειμένων, το σκορ συνάφειας των αντικειμένων, αν ένα αντικείμενο είναι σχετικό ή όχι κλπ.) Έτσι, αν ενδιαφερόμαστε για ένα συνολικά καλύτερο σύστημα Ανάκτησης Πληροφορίας, τότε ο πυρήνας **Neighborhood Hash** εξακολουθεί να είναι η καλύτερη επιλογή.

Όσον αφορά τις συγκεκριμένες μονάδες ΝΔΓ, το πιο εκπληκτικό αποτέλεσμα είναι η υπο-απόδοση των παραλλαγών **GIN**, σχεδόν σε κάθε αρχιτεκτονική. Το GIN ήταν μακράν το μοντέλο με την καλύτερη θεωρητική θεμελίωση μεταξύ των τεσσάρων, οπότε αποτελεί έκπληξη το γεγονός ότι σημείωσε χαμηλότερη βαθμολογία από τα άλλα, σχεδόν σε όλες τις περιπτώσεις. Εκείνο που είχε γενικά τις καλύτερες επιδόσεις ήταν το **GCN**, με το **GATv2** να σημειώνει επίσης εξίσου υψηλή βαθμολογία.

Τα αποτελέσματα από αυτά τα πρώτα πειράματα είναι αυτά που μας οδήγησαν να πραγματοποιήσουμε έξτρα

		MAP@10	MRR	NDCG@10
Shortest Path Kernel		0.3086	0.3747	0.0852
Random Walk Kernel		0.0532	0.0693	0.0177
Weisfeiler-Lehman Kernel		0.3372	0.4011	0.1102
Graphlet Sampling Kernel		0.1468	0.1770	0.0374
Neighborhood Hash Kernel		**0.4449**	**0.5440**	**0.1568**
VGAE	GCN	0.4941	0.5873	0.1989
	GAT	0.4972	0.5736	0.1851
	GATv2	0.5020	0.6046	**0.2030**
	GIN	**0.5053**	**0.6067**	0.1914
ARVGA	GCN	0.4678	0.5554	0.1757
	GAT	0.4813	0.5540	0.1744
	GATv2	0.4956	0.5754	0.1911
	GIN	**0.5247**	**0.6325**	**0.2015**
Combined-ARVGA	GCN	0.4828	0.5783	0.1907
	GAT	0.4757	0.6060	0.2236
	GATv2	0.5064	0.5915	0.2156
	GIN	**0.5441**	**0.6067**	**0.2254**
Combined-ARVGA-MLP	GCN	0.5078	0.5874	0.1910
	GAT	0.5250	0.6115	0.2326
	GATv2	**0.5591**	**0.6580**	**0.2438**
	GIN	0.5099	0.5947	0.2299

Table 1.4: Τελικές Μετρικές για τους Πυρήνες Γράφων και τα βέλτιστα μοντέλα ΝΔΓ, στο Dense Subset. Τα έντονα γράμματα υποδηλώνουν το καλύτερο αποτέλεσμα για κάθε αρχιτεκτονική.

πειράματα σε πιο πυκνούς γράφους και να αξιολογήσουμε τη συμπεριφορά των ΝΔΓ και σε αυτούς. Όπως έχει ήδη αναφερθεί, δεν δοκιμάσαμε ξανά όλες τις πιθανές αρχιτεκτονικές στο Dense Subset, παρά μόνο τις πιο θεμελιώδεις και τις πιο υποσχόμενες: VGAE, ARVGA, Combined-ARVGA, Combined-ARVGA-MLP. Τα αποτελέσματα στο **Dense Subset** παρουσιάζονται στον πίνακα 1.4.

Η σημαντική διαφορά εδώ, είναι ότι τα μοντέλα ΝΔΓ υπερτερούν έναντι όλων των Πυρήνων Γράφων, και στις τρεις μετρικές. Και το επιτυγχάνουν αυτό, ακόμη και από την πρώτη αρχιτεκτονική, το απλό **VGAE**. Και το καλύτερο μοντέλο ΝΔΓ, η παραλλαγή **GATv2** του **Combined-ARVGA-MLP** επιτυγχάνει 11.5% υψηλότερο MAP, 11% υψηλότερο MRR και 9% υψηλότερο NDCG. Έτσι, μπορούμε να πούμε με ασφάλεια ότι πρόκειται για ένα πολύ καλύτερο σύστημα Ανάκτησης από τους πυρήνες γράφων για αυτή την περίπτωση. Επίσης, παρατηρούμε ότι και στις δύο περιπτώσεις (τυχαία και πυκνά γραφήματα), η μονάδα που βοηθήθηκε περισσότερο από την προσθήκη MLP στην αρχιτεκτονική Combined-ARVGA, είναι η GATv2.

Αυτό αποτελεί επίσης μια σαφή απόδειξη, της σημασίας της ποιότητας του συνόλου δεδομένων για όλα τα μοντέλα μηχανικής μάθησης. Ιδιαίτερα για τα ΝΔΓ, είναι προφανές ότι η **Πυκνότητα** των γραφημάτων εκπαίδευσης, παίζει σημαντικό ρόλο, επειδή η μέθοδος *Message-Passing* δεν μπορεί να αξιοποιηθεί αποτελεσματικά όταν υπάρχουν πολλοί απομονωμένοι κόμβοι. Επίσης, ο περιορισμός του μεγέθους του γράφου έπαιξε σημαντικό ρόλο, καθώς οι τελικές Ενθέσεις Γράφων είναι πιο επαρκώς συγκρίσιμες, όταν τα μεγέθη των γράφων είναι της ίδιας τάξης.

Μια άλλη σημαντική διαφορά είναι οι βαθμολογίες των παραλλαγών **GIN**. Σε αυτή την περίπτωση, είναι αυτές που παράγουν τα καλύτερα αποτελέσματα στις περισσότερες περιπτώσεις, ενώ στην περίπτωση του Random Set είχαν αρκετά κακές επιδόσεις. Αυτό είναι ένα ακόμη στοιχείο που υποστηρίζει τον ισχυρισμό που διατυπώσαμε παραπάνω, ότι η πυκνότητα των γραφημάτων παίζει καθοριστικό ρόλο στο να μπορούν να αξιοποιήσουν την εκφραστική τους δύναμη και να μάθουν σημαντικά χαρακτηριστικά.

Θα παραθέσουμε επίσης τους χρόνους εκτέλεσης για τους πυρήνες γραφημάτων, καθώς και τους χρόνους εκπαίδευσης (training) και συμπερασμού (inference) για τα μοντέλα ΝΔΓ, στον Πίνακα 1.5. Το μόνο σημαντικό πράγμα που πρέπει να παρατηρήσουμε εδώ, είναι ότι η Εκπαίδευση των μοντέλων ΝΔΓ, διαρκεί από μερικά δευτερόλεπτα, έως και μερικά λεπτά. Έτσι, ο συμβιβασμός, μεταξύ του χρόνου εκτέλεσης και των αποτελεσμάτων των πυρήνων, και του χρόνου Εκπαίδευσης και των αποτελεσμάτων των ΝΔΓ, αξίζει σίγουρα τον κόπο, ειδικά

		Train Time	Inference Time
Shortest Path Kernel		-	117 ms
Random Walk Kernel		-	16.5 sec
Weisfeiler-Lehman Kernel		-	338 ms
Graphlet Sampling Kernel		-	3.51 sec
Neighborhood Hash Kernel		-	263 ms
VGAE	GCN	1min 3sec	680 ms
	GAT	54.1 sec	836 ms
	GATv2	1min 13sec	899 ms
	GIN	56 sec	275 ms
ARVGA	GCN	30 sec	649 ms
	GAT	3min 41sec	855 ms
	GATv2	1min 7sec	905 ms
	GIN	4min 35sec	299 ms
Combined-ARVGA	GCN	4min 30sec	723 ms
	GAT	5min 23sec	888 ms
	GATv2	3 min	971 ms
	GIN	1min 6sec	320 ms
Combined-ARVGA-MLP	GCN	4min 29sec	697 ms
	GAT	4min 16sec	916 ms
	GATv2	5min 48sec	964 ms
	GIN	16 sec	327 ms

Table 1.5: Χρόνοι Εκπαίδευσης/Συμπερασμού των Πυρήνων Γράφων και των μοντέλων ΝΔΓ, στο Dense Subset

στην περίπτωση του Dense Subset. Δεν πρέπει επίσης να ξεχνάμε, ότι το μεγαλύτερο πλεονέκτημα των ΝΔΓ έναντι των Πυρήνων Γράφων, είναι ότι παρέχουν Ενθετικές Αναπαραστάσεις γενικής χρήσης, που μπορούν να χρησιμοποιηθούν για την επίλυση οποιουδήποτε μεταγενέστερου προβλήματος.

Συμπερασματικά, μπορούμε να δούμε ότι τα ΝΔΓ αποδεικνύονται μια πραγματικά καλή λύση, όταν πρόκειται για την εύρεση **Ομοιότητας Γράφων**, ειδικά αν έχουμε στη διάθεσή μας μια διαβεβαίωση για την ποιότητα των γραφημάτων. Όσον αφορά το χρόνο εκπαίδευσης, ανταγωνίζονται επίσης όλα τα συστήματα Επιβλεπόμενης Ομοιότητας, τα οποία θα χρειάζονταν πολύ περισσότερο χρόνο για να εκπαιδευτούν, επειδή χρησιμοποιούν ζεύγη γραφημάτων ως δείγματα και όχι μόνο γραφήματα (αυξάνοντας τετραγωνικά το πλήθος των δειγμάτων εκπαίδευσης).

Ποιοτική Ανάλυση

Για την Ποιοτική αξιολόγηση των μοντέλων, θα χρησιμοποιήσουμε τις εικόνες που αντιστοιχούν στους γράφους σκηνής, οι οποίοι ανακτώνται από τα ΝΔΓ και τους Πυρήνες. Θα παρουσιάσουμε τις Εικόνες-Ερωτήματα (Queries) και τις πρώτες εικόνες που ανακτήθηκαν, τόσο για τις επιτυχείς όσο και για τις ανεπιτυχείς περιπτώσεις. Είναι σημαντικό να θυμόμαστε εδώ, ότι **όλα τα μοντέλα που αναφέρθηκαν και δοκιμάστηκαν, δεν είχαν πρόσβαση στις Εικόνες, παρά μόνο στους αντίστοιχους Γράφους Σκηνής**. Αυτό προφανώς οδηγεί σε περιπτώσεις όπου οι Εικόνες δεν είναι όμοιες, αλλά η ομοιότητα των Γραφημάτων οδηγεί το μοντέλο να τις κατατάξει ψηλά. Παρακάτω, παρουσιάζουμε ζεύγη Ερωτήσεων-Απαντήσεων, που αποτελούν παραδείγματα των σεναρίων που εξηγήθηκαν παραπάνω.

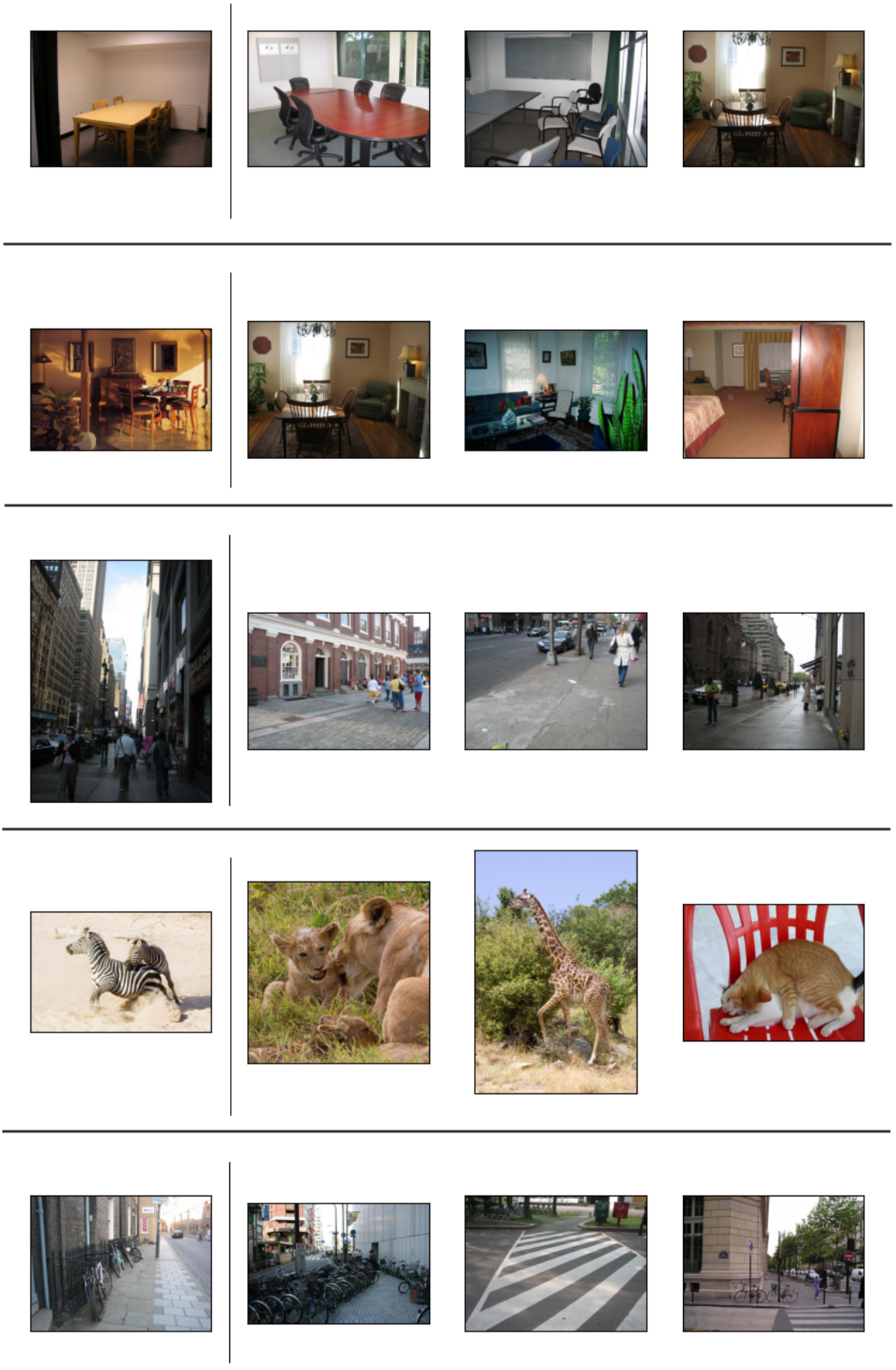


Figure 1.3.2: Παραδείγματα διαφορετικών ερωτημάτων (αριστερά), με τα 3 κορυφαία αποτελέσματα (δεξιά), όπως ανακτήθηκαν από το καλύτερο μοντέλο ΝΔΓ

Figure 1.3.3: Δύο παραδείγματα ερωτημάτων, όπου τα 2 πρώτα αντικείμενα που ανακτώνται από το καλύτερο μοντέλο ΝΔΓ (πάνω) είναι οπτικά πιο κοντά στην εικόνα του ερωτήματος (αριστερά), σε σύγκριση με τα 2 πρώτα αντικείμενα που ανακτώνται από τον καλύτερο Πυρήνα Γραφημάτων (κάτω).

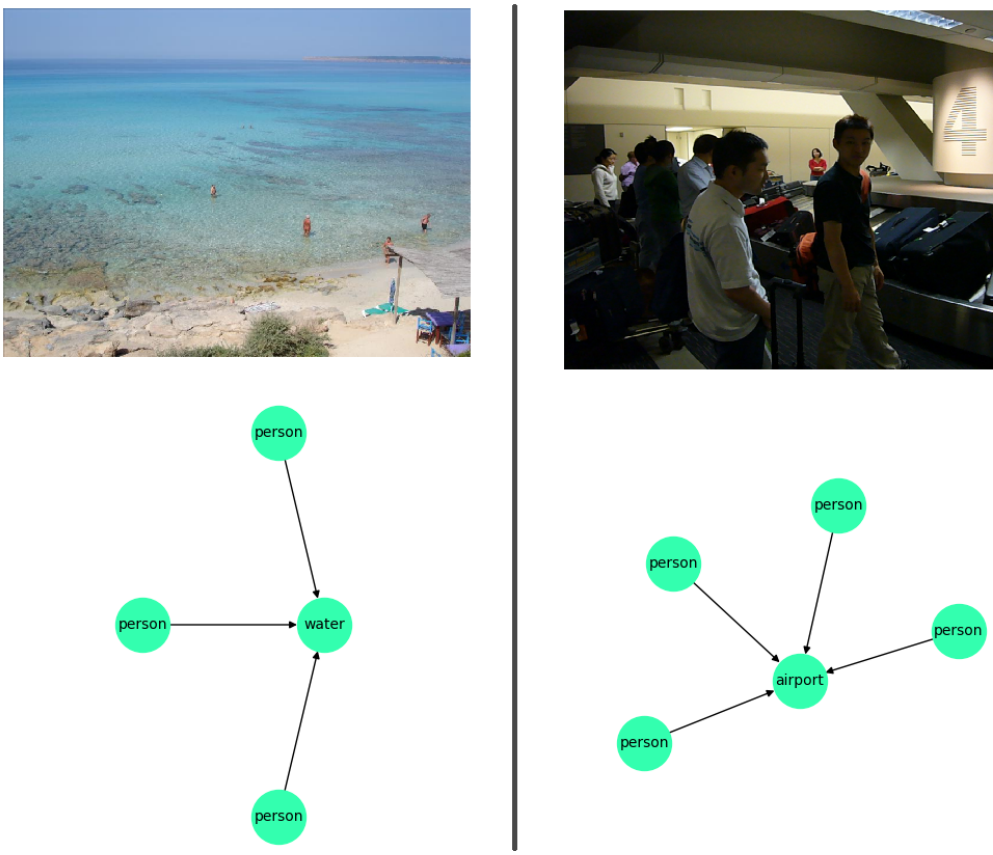


Figure 1.3.4: Παράδειγμα όπου το ερώτημα (αριστερά) και το πρώτο ανακτηθέν αντικείμενο (δεξιά) σύμφωνα με το καλύτερο μοντέλο ΝΔΓ, έχουν δομικά παρόμοιους γράφους σκηνής, αλλά αρκετά ανόμοιες εικόνες.

1.4 Συμπεράσματα

Στην παρούσα διατριβή, αντιμετωπίσαμε το πρόβλημα της Ομοιότητας Γράφων, στο πλαίσιο της κατασκευής συστημάτων **Ανάκτησης Πληροφορίας**. Συγκεκριμένα, θεωρήσαμε ως βασική αλήθεια την ομοιότητα που υπολογίζεται από τον αλγόριθμο *Απόστασης Επεξεργασίας Γράφων* (ΑΕΓ), του οποίου στόχος είναι να μετατρέψει ένα δεδομένο γράφο, σε έναν ισομορφικό κάποιου άλλου γράφου. Αυτό έχει αποδειχθεί ότι είναι NP-Hard, οπότε χρησιμοποιούνται συνήθως διάφορες προσεγγιστικές παραλλαγές του ΑΕΓ. Εδώ, χρησιμοποιούμε την παραλλαγή *Bipartite Matching*, η οποία προσεγγίζει τη βέλτιστη απόσταση επεξεργασίας γράφων, λαμβάνοντας υπόψη κυρίως τις πληροφορίες των κόμβων και τις επεξεργασίες που τους αποδίδονται. Για το κόστος εισαγωγής/διαγραφής/αντικατάστασης των γράφων σκηνής, ενσωματώνουμε την Ιεραρχία Εννοιών που παρέχεται από το *WordNet*, μαζί με αλγορίθμους "ομοιότητας μονοπατιών" που εφαρμόζονται σε αυτή την ιεραρχία. Παρόλο που οι παραλλαγές του ΑΕΓ μάς παρέχουν τα πιο ακριβή αποτελέσματα, η κλασική τεχνική για την αντιμετώπιση της ομοιότητας γράφων είναι μέσω της χρήσης των **Πυρήνων Γράφων**. Η μειωμένη υπολογιστική πολυπλοκότητά τους, η ικανότητα ερμηνείας των αποτελεσμάτων και η διαρκώς υψηλή απόδοσή τους, τους έχουν τοποθετήσει ως μια δελεαστική επιλογή εδώ και πολλά χρόνια. Επιλέγουμε να αξιολογήσουμε πέντε δημοφιλή Πυρήνες Γράφων, τρεις από τους οποίους βασίζονται σε θεμελιώδεις μετρικές γράφων, ενώ οι άλλοι δύο βασίζονται στη διάδοση πληροφορίας μέσω των ακμών.

Η κεντρική ιδέα αυτής της εργασίας, είναι να αναλύσουμε διάφορες αρχιτεκτονικές **Μη-επιβλεπόμενων Αυτο-κωδικοποιητών Γράφων**, προκειμένου να αξιολογήσουμε διεξοδικά και να συγκρίνουμε τις επιδόσεις τους με τους προαναφερθέντες Πυρήνες Γράφων. Τα ΝΔΓ που προτείνουμε και πειραματιζόμαστε, αποτελούνται από διάφορα δομικά στοιχεία, καθένα από τα οποία εστιάζει σε μια διαφορετική πτυχή της *Εκμάθησης Αναπαράστασης* δεδομένων. Συγκεκριμένα, το τελικό μοντέλο που προτείνουμε, το **Combined-ARVGA**, αποτελείται από τρία βασικά στοιχεία:

- Ο αρχικός ΑΚΓ, μαζί με τον **Inner Product Decoder**, που είναι υπεύθυνος για τη δομική ανακατασκευή του γραφήματος.
- Ο **Αποκωδικοποιητής Χαρακτηριστικών**, υπεύθυνος για την ανακατασκευή της πληροφορίας στους κόμβους του γραφήματος.
- Ο **Διευκρινιστής**, υπεύθυνος για την κανονικοποίηση των ενθετικών αναπαραστάσεων που παράγονται από τον κωδικοποιητή, μέσω της Ανταγωνιστικής Εκπαίδευσης.

Για τις πραγματικές μονάδες ΝΔΓ που χρησιμοποιούνται στους τελικούς ΑΚΓ, δοκιμάζουμε τέσσερις από τους πιο δημοφιλείς: *GCN*, *GAT*, *GATv2* και *GIN*. Όσον αφορά τους γράφους εισόδου, χρησιμοποιούμε τις ενθετικές αναπαραστάσεις λέξεων του **GloVe** και αφαιρούμε τα χαρακτηριστικά και τις πληροφορίες ακμών από τους αρχικούς γράφους σκηνών, απλοποιώντας περαιτέρω τη δομή τους. Αποκτούμε επίσης δύο υποσύνολα του Visual Genome, το **Random** υποσύνολο και το **Dense** υποσύνολο, προκειμένου να μετρήσουμε την επίδραση της ποιότητας του συνόλου δεδομένων, στην τελική απόδοση των μοντέλων. Για τη διαδικασία αξιολόγησης, χρησιμοποιήσαμε τρεις από τις πιο ευρέως χρησιμοποιούμενες μετρικές για την αξιολόγηση συστημάτων Ανάκτησης Πληροφορίας: **MAP** (μέτρηση του *precision* της τελικής κατάταξης), **MRR** (μέτρηση της θέσης του πρώτου ανακτηθέντος σχετικού αντικειμένου) και **NDCG** (η πιο γενική μετρική, η οποία λαμβάνει υπόψη τις θέσεις και τις βαθμολογίες συνάφειας).

Από τα πειράματα στο Random υποσύνολο, μπορούμε να συμπεράνουμε ότι το Combined-ARVGA ήταν η αρχιτεκτονική με τις καλύτερες επιδόσεις, με τα GCN και GATv2 να είναι οι πιο ικανές μονάδες ΝΔΓ. Πραγματοποιώντας μια μελέτη αφαίρεσης στις διάφορες αρχιτεκτονικές, υποστηρίζουμε ότι η χρήση ενός **Αποκωδικοποιητή Χαρακτηριστικών** μαζί με μια μέθοδο **Ανταγωνιστικής Κανονικοποίησης**, παίζουν πρωταρχικό ρόλο στην εκμάθηση ουσιαστικών αναπαραστάσεων για τους κόμβους του γράφου, οι οποίες μπορούν εύκολα να επεκταθούν και στην παραγωγή ενθέσεων για τις ακμές, ή ακόμη και για ολόκληρο τον γράφο. Όσον αφορά τους Πυρήνες Γράφων, η εκφραστική δύναμη της μεθόδου διάδοσης πληροφορίας είναι εμφανής στο τελικό αποτέλεσμα, καθώς οι πυρήνες γράφων Weisfeiler-Lehman και Neighborhood-Hash, ξεπέρασαν τους άλλους πυρήνες με σημαντική διαφορά. Όταν συγκρίνουμε τον καλύτερο πυρήνα με το καλύτερο μοντέλο ΝΔΓ, διαπιστώνουμε ότι το **Combined-ARVGA GCN** σημείωσε 1% υψηλότερη βαθμολογία στο MAP, 3% υψηλότερη στο MRR, αλλά $1,5\%$ χαμηλότερη στο NDCG. Το γεγονός ότι τα ΝΔΓ δεν μπορούσαν να σημειώσουν τόσο υψηλή βαθμολογία όσο οι Πυρήνες Γράφων στο NDCG, είναι αυτό που μας οδήγησε να πραγματοποιήσουμε μια πιο ενδελεχή ανάλυση του τυχαίου υποσυνόλου που χρησιμοποιήθηκε αρχικά. Διαπιστώθηκε ότι οι γράφοι σκηνής είχαν αρκετά χαμηλή πυκνότητα, σημαντικό αριθμό απομονωμένων κόμβων, καθώς και μεγάλη ποικιλία μεγεθών γραφημάτων, γεγονός που καθιστά δύσκολη τη σύγκρισή τους. Ως εκ τούτου, δημιουργήσαμε ένα δεύτερο σύνολο δεδομένων, το Dense υποσύνολο, προκειμένου να ελέγξουμε αν η ποιότητα των δεδομένων μπορεί να επηρεάσει αρνητικά την απόδοση των μοντέλων ΝΔΓ, σε σημείο που οι Πυρήνες Γράφων να τα ξεπερνούν.

Η υπόθεση αυτή επαληθεύτηκε διεξοδικά μέσω των πρόσθετων πειραμάτων που πραγματοποιήσαμε στο Dense υποσύνολο. Συγκεκριμένα, συγκρίνοντας το **Combined-ARVGA-MLP** που χρησιμοποιεί τη μονάδα GATv2 με τον Πυρήνα Γράφων με τις καλύτερες επιδόσεις, το ΝΔΓ σημείωσε υψηλότερη βαθμολογία 11% στο MAP, 11% στο MRR και 9% στο NDCG. Έτσι, μπορούμε να συμπεράνουμε με ασφάλεια ότι οι ΑΚΓ μπορούν να μάθουν σημαντικές αναπαραστάσεις γραφημάτων και να τις χρησιμοποιήσουν για να ξεπεράσουν τους Πυρήνες Γράφων στο πρόβλημα της Ομοιότητας Γράφων, δεδομένου ότι η ποιότητα του συνόλου δεδομένων έχει αναλυθεί και επικυρωθεί. Εκτός από αυτό, το σημαντικότερο πλεονέκτημα των ΑΚΓ, είναι ότι οι τελικές Ενθετικές Αναπαραστάσεις δεν έχουν εκπαιδευτεί σε κάποια μεταγενέστερη εργασία, αλλά μόνο για εκμάθηση αναπαράστασης γενικού σκοπού. Αυτό μας επιτρέπει να χρησιμοποιήσουμε αυτά τα εκπαιδευμένα μοντέλα και να εκτελέσουμε παραδοσιακές εργασίες σε επίπεδο κόμβων, ακμών ή γράφων, στους γράφους που έχουμε στη διάθεσή μας.

Τέλος, όπως αναφέρθηκε στην Ενότητα 1.1.6, ο απώτερος στόχος για την εκπαίδευση αυτών των μοντέλων ΝΔΓ και την αξιολόγησή τους στην εργασία Ανάκτησης Πληροφορίας, είναι κυρίως η υλοποίησή τους μέσα σε ένα πλαίσιο Εξηγήσεων με Αντιπαραδείγματα (όπως αυτό που προτείνεται στο [21]), το οποίο χρησιμοποιεί εξωτερική γνώση που παρέχει εξηγήσεις για τα δείγματα ενός αυθαίρετου συνόλου δεδομένων. Εάν αυτή η εξωτερική γνώση διατυπώνεται σε δομές γράφων, τότε μπορούν να αξιοποιηθούν μοντέλα ΝΔΓ, παρόμοια με αυτά που αναλύονται στην παρούσα διατριβή, ώστε να παρέχουν καλύτερη απόδοση και χαμηλότερους χρόνους υπολογισμού.

1.4.1 Μελλοντικές Κατευθύνσεις

Στην επιδίωξη της προώθησης του πεδίου του υπολογισμού της ομοιότητας γραφημάτων με τη χρήση αυτοκωδικοποιητών γράφων, η παρούσα διατριβή έθεσε ένα θεμελιώδες πλαίσιο και παρείχε πολύτιμες γνώσεις σχετικά με την αποτελεσματικότητα και τις δυνατότητες τέτοιων μοντέλων σε προβλήματα Ανάκτησης Πληροφορίας. Ωστόσο, όπως συμβαίνει με κάθε επιστημονική προσπάθεια, η εξερεύνηση αυτού του τομέα απέχει πολύ από το να είναι εξαντλητική, και πολλά αχαρτογράφητα μονοπάτια καλούν για περαιτέρω έρευνα. Σε αυτό το κεφάλαιο, εμβαθύνουμε στις πολλά υποσχόμενες κατευθύνσεις και τις ανεκμετάλλευτες δυνατότητες για μελλοντική έρευνα, με στόχο να αξιοποιήσουμε τα ευρήματα και τις μεθοδολογίες που παρουσιάστηκαν στην παρούσα εργασία. Συγκεκριμένα, οι βασικοί τομείς έρευνας που παρουσιάζουν ιδιαίτερο ενδιαφέρον είναι οι ακόλουθοι:

- Πρώτα απ' όλα, το κατά πόσο η απόδοση αυτών των μοντέλων ΝΔΓ σε ένα πρόβλημα **Inductive Ανάκτησης Πληροφορίας** είναι ικανή, είναι το πιο σημαντικό ερώτημα μετά την εργασία που παρουσιάζεται εδώ. Διατυπώσαμε το πρόβλημα ως Transductive και αποδείξαμε την υπεροχή των Αυτο-κωδικοποιητών Γράφων έναντι των Πυρήνων Γράφων, αλλά πρέπει να υπάρξει ενδελεχής πειραματισμός και αξιολόγηση για να γίνει ο ίδιος ισχυρισμός για το Inductive Inference.
- Η χρήση των **Σχεσιακών Μονάδων ΝΔΓ**, θα πρέπει επίσης να εξεταστεί, επειδή επιτρέπουν μια πολύ πιο εκφραστική αναπαράσταση για τις ακμές του συνόλου δεδομένων. Συγκεκριμένα, τα Σχεσιακά ΝΔΓ, διατυπώνουν το γράφημα εισόδου ως ένα *ετερογενές* γράφημα, όπου υπάρχουν πολυάριθμοι τύποι ακμών, ο καθένας με ένα μοναδικό πίνακα γειτνίασης. Η αξιοποίηση τέτοιων μοντέλων, θα μπορούσε να μας επιτρέψει να αλλάξουμε την προεπεξεργασία των γράφων, και να εκμεταλλευτούμε τις πληροφορίες για τις Ακμές.
- Οι Αυτο-κωδικοποιητές γενικότερα, έχουν χρησιμοποιηθεί εκτενώς σε προβλήματα **Υπο'συνθήκης Παραγωγής Δεδομένων**, και οι Αυτο-κωδικοποιητές Γράφων δεν αποτελούν εξαίρεση. Παρόλο που δεν εμβαθύναμε στις δυνατότητες Παραγωγής των ΑΚΓ σε αυτή τη διατριβή, θα μπορούσε να αποδειχθεί ένα ισχυρό εργαλείο, ειδικά στο πλαίσιο της Εξηγησιμότητας, όπου ο στόχος πολλών προτεινόμενων πλαισίων είναι η παραγωγή νέων Αντιπαραδειγμάτων με σκοπό την επεξηγησιμότητα των μοντέλων.
- Στο πλαίσιο της διατριβής, πειραματιστήκαμε μόνο με τις πιο δημοφιλείς και θεμελιώδεις **Μονάδες ΝΔΓ**, οι οποίες είναι τα GCN, GAT, GATv2 και GIN. Έχουν προταθεί αρκετές πιο εκλεπτυσμένες εκδόσεις[32, 15], οι οποίες προσπαθούν να αντιμετωπίσουν το πρόβλημα της υπερ-απόσβεσης κατά τη στοίβαξη πολλών επιπέδων. Η χρήση αυτών των μονάδων ΝΔΓ, θα μπορούσε να επιτρέψει την επεκτασιμότητα των μοντέλων που προτείνονται εδώ, προκειμένου να επιτευχθούν καλύτερες επιδόσεις σε προβλήματα Εκμάθησης Αναπαράστασης.

Chapter 2

Introduction

Recent developments in artificial intelligence (AI) have been nothing short of transformative. AI has found applications in a wide range of fields, from healthcare and finance to transportation and entertainment. One of the notable trends in AI has been the growing importance of Graph Neural Networks (GNNs). GNNs are a specialized class of neural networks designed to work with graph data, which is particularly crucial in modeling complex relationships and structures found in various domains. They have shown remarkable success in tasks like social network analysis, recommendation systems, and even drug discovery, where understanding the intricate connections between entities is essential.

The need for Graph Neural Networks arises from the limitations of traditional neural networks, which are primarily designed for grid-like data, such as images or sequences. In contrast, many real-world problems involve interconnected data points, such as social networks, transportation networks, or molecular structures. GNNs excel in capturing and leveraging the rich, hierarchical relationships within such data. Their ability to perform tasks like node classification, link prediction, and graph classification makes them invaluable tools for tackling complex, interconnected problems.

The problem that we will tackle with this thesis, is the Graph Similarity problem. Given a query graph, we want to rank all the possible "answer" graphs, based on their similarity to the query graph. Through the years, a lot of solutions have been proposed to solve Graph Similarity, the most popular ones being Graph Kernels. Graph Kernels are algorithmic methods that provide a quick way to compare graphs, using several graph metrics and graph transformations. They are usually the preferred methods, because of their ease of implementation, and their low computational cost. In this thesis, we will tackle this problem using Graph Neural Networks. Specifically, we will use un-supervised Graph Autoencoders, since supervised approaches have already been studied[129].

Another pressing concern in the AI community is the need for Explainable AI (XAI). As AI systems become more integrated into our daily lives, their decisions impact critical areas like healthcare, finance, and autonomous vehicles. However, the inherent opacity of some AI models, especially deep learning ones, raises questions about their reliability and trustworthiness. That's why the need for Black-Box Explainability methods is become more important, as the complexity of the AI models is constantly rising. One of the most reliable Black-Box Explanation methods, is through Counterfactual Explanations. This process involves analyzing the differences between the Query Object (graph), and the Counterfactual Object, which is the first retrieved graph that doesn't belong to the same class as the Query Object. The importance of the Counterfactual Object lies in the fact that it can provide us with the minimum changes needed to be made, in order to move to a different class. We will follow the procedure and algorithm defined in [29] and [21].

The outline of this thesis:

- We will firstly set the theoretical ground for defining and applying all the methods and models we will use for the experiments. This includes Graph theory, Deep Learning theory, Graph Kernel variants, several Graph Neural Network architectures, and Graph Edit Distance approximations.

- We will then give a more detailed and formalistic definition of Counterfactual Explanations, and specifically Conceptual Edits that can be used to provide Black-Box Explainability for the GNN models.
- Lastly, we will define the proposed GNN Auto-Encoder model used to tackle the Graph Similarity model, with all the details on its architecture and training methods. We will also evaluate the different GNN architectures, both in a quantitative and in a qualitative manner, and compare the results to the more conventional methods (Graph Kernels), allowing us to draw conclusions and provide meaningful insights on the advantages and disadvantages of the GNN models.

Chapter 3

Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and models that allow computers to learn and make predictions or decisions from data. Its history can be traced back to the mid-20^{th} century when researchers began exploring methods to enable computers to improve their performance through experience. Early machine learning techniques were largely rule-based and relied on expert knowledge to define decision-making processes.

One significant milestone in the history of machine learning was the development of decision trees and the ID3 algorithm by Ross Quinlan in the 1980s[90]. These methods paved the way for automated decision-making based on data, making it possible to create models that could learn from examples. Another crucial development was the introduction of support vector machines (SVMs) in the 1990s[19, 7], which enabled the classification of data points into different categories with high accuracy.

The evolution of machine learning into deep learning marked a transformative shift in the field. Deep learning is a subset of machine learning that focuses on neural networks with multiple layers (deep neural networks). Deep learning algorithms, such as convolutional neural networks (CNNs) for image recognition and recurrent neural networks (RNNs) for sequential data, have achieved remarkable success in various domains, including computer vision, natural language processing, and speech recognition. The deep learning revolution was driven by advancements in hardware, such as graphics processing units (GPUs), which accelerated the training of large neural networks, and the availability of vast datasets.

Contents

3.1 Data Modalities **32**
3.2 Machine Learning Types **32**
3.3 Definitions **33**
3.4 Deep Learning **37**
3.5 Autoencoders **48**

3.1 Data Modalities

Machine learning deals with a variety of data modalities, each with its own unique challenges and applications. Several different architectures have been developed to tackle the problems presented in each modality. The structure of the data, the way that the information is stored, and the semantic content of each modality are a just a few things that define the Machine Learning approach for modality. Below, we present the most popular modalities found in modern datasets.

Structured Data

This type of data is organized in tables or databases, typically with rows and columns, where each row represents a distinct entity (e.g., a customer, transaction) and each column represents an attribute or feature (e.g., age, income, product ID). Structured data is commonly used for tasks like regression, classification, and database management.

Unstructured Text Data

Unstructured text data lacks a specific format and can include documents, emails, social media posts, and more. Natural Language Processing (NLP) techniques are applied to analyze and extract information from text data, enabling tasks such as sentiment analysis, text summarization, and language translation.

Image Data

Image data consists of visual information represented as pixel values. Machine learning models, particularly Convolutional Neural Networks (CNNs), are used to process and analyze images. This modality is vital for image classification, object detection, facial recognition, and medical image analysis.

Time-Series Data

Time-series data comprises sequential observations recorded at regular intervals. Common examples include stock prices, temperature measurements, and sensor readings. Time-series analysis is vital for forecasting, anomaly detection, and trend analysis.

Graph Data

Graph data represents entities (nodes) and their relationships (edges) in a network structure. Graph Neural Networks (GNNs) are used for tasks such as social network analysis, recommendation systems, and knowledge graph reasoning.

3.2 Machine Learning Types

In machine learning, various learning types and paradigms are used to train models based on different approaches and objectives. The popularity of these learning types may vary depending on the problem, data, and application. The key point that differentiates these approaches is Supervision, which implicitly defines the learning goal of the machine learning model. The most popular learning types are presented below.

Supervised Learning

Supervised learning is one of the most widely used learning types. It involves training a model on labeled data, where each input is associated with a corresponding output or target. The goal is to learn a mapping from inputs to outputs, enabling the model to make predictions on unseen data. Common supervised learning algorithms include linear regression, decision trees, support vector machines, and deep neural networks.

Unsupervised Learning

Unsupervised learning deals with unlabeled data, where the model aims to discover patterns, structures, or groupings within the data. Clustering and dimensionality reduction are common tasks in unsupervised learning. Algorithms such as k-means clustering, hierarchical clustering, and principal component analysis (PCA) fall into this category.

Semi-Supervised Learning

Semi-supervised learning combines elements of both supervised and unsupervised learning. It uses a small amount of labeled data and a larger amount of unlabeled data to improve model performance. This approach is useful when obtaining labeled data is costly or time-consuming. Tasks such as speech recognition, web content classification and text document classification fall into this category.

Self-Supervised Learning

Self-supervised learning is a variation of unsupervised learning where the model generates its own labels from the data. It's often used in NLP and computer vision, where the model learns by solving pretext tasks, such as predicting masked words in a sentence or generating context from an image.

Reinforcement Learning

Reinforcement learning is employed for tasks where an agent interacts with an environment and learns to make a sequence of actions to maximize a reward signal. It is commonly used in robotics, gaming, autonomous systems, and recommendation systems. Popular algorithms in reinforcement learning include Q-learning[118] and deep reinforcement learning methods like Deep Q-Networks (DQN)[79] and Proximal Policy Optimization (PPO).

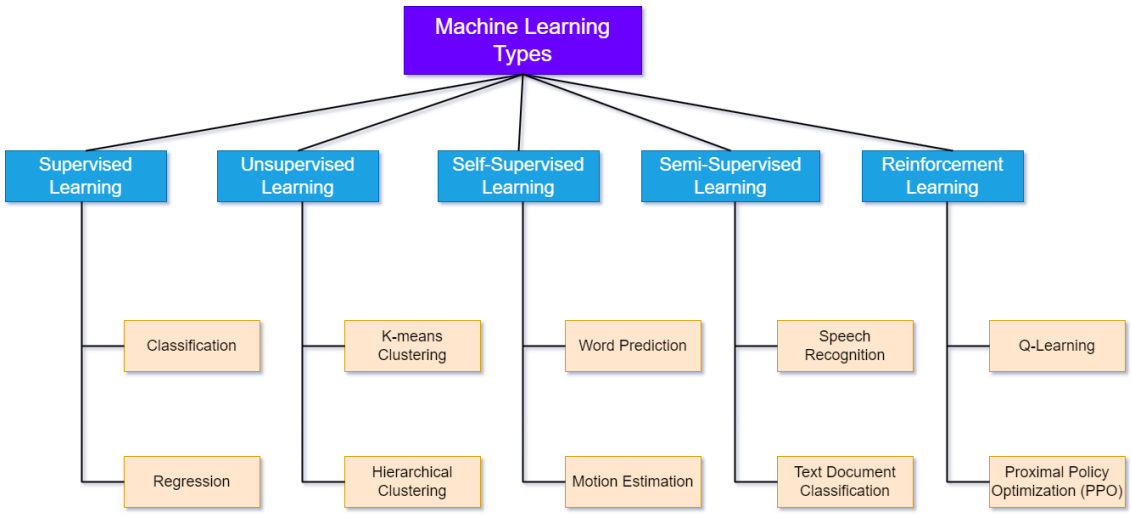


Figure 3.2.1: Machine Learning Types and common use cases.

3.3 Definitions

Machine learning is a subset of artificial intelligence (AI) that revolves around the fundamental idea of enabling computers to learn from data and make predictions or decisions without explicit programming. At its core, machine learning is built on several foundational concepts, the most important presented below.

Data is the cornerstone of machine learning. It serves as the raw material from which models learn patterns and relationships. This data can come in various forms, including text, numbers, images, and more. The quality and quantity of data significantly impact the effectiveness of machine learning models. A dataset, which is a collection of n samples, is usually represented as $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\}$, in the case of supervised learning.

Features are specific characteristics or attributes within the data that the model uses to make predictions. For example, in a spam email classification task, features could include the words used in the email, the sender's address, and the email's length. The selection and engineering of relevant features are critical to the model's performance. In the above definition of data, $\mathbf{x_n}$ represent the features of the dataset.

The process of **Training** a machine learning model involves exposing it to a dataset with known outcomes (y_n), often referred to as labeled data. During training, the model adjusts its internal parameters or weights to learn patterns and correlations within the data. This allows it to make predictions or classifications when presented with new, unseen data. The main goal of the training process is for the model to be able to predict the labels y_n given the input feature vectors $\mathbf{x_n}$. The way that this prediction affects the training process, is through the **Loss Functions**. Loss Functions take as input the true label y_n and the predicted label $\hat{y}_n$, and return the "Loss", which has to be minimized through the training process. The most popular Loss Functions are the following:

- **Binary Cross-Entropy Loss** is the most common Loss Function for Classification tasks, and especially Binary Classification. The simplified Loss Function for the binary classification can be mathematically expressed as followed:

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

- **Hinge Loss** is another Loss Function for Classification tasks, which was mostly developed for Support Vector Machine (SVM) model evaluation, and is formulated as followed:

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

- **Mean Squared Error** is the most commonly used Loss Function for Regression tasks, where the model has to predict a numerical value, not a class. The mathematical formula for this loss is:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

- **Mean Absolute Error** is the also used for Regression tasks, and it is preferred to *Mean Squared Error* when there are many outliers in the data. When dealing with outliers, *Mean Squared Error* will usually produce a much larger Loss, compared to most samples from the dataset, thus rendering it less robust to outliers than *Mean Absolute Error*. The formula of **MAE** is:

$$L = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

- **Huber Loss** is a robust regression loss function that combines aspects of MSE and MAE, with a controllable parameter for the threshold. The formula is:

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

- **Kullback-Leibler Divergence** measures the difference between two probability distributions, often used in probabilistic models:

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

Hyperparameter tuning is a critical step in the machine learning model development process, which usually takes place before, or during the model's training. It involves finding the optimal values for various hyperparameters that govern a model's behavior, such as learning rates, the number of hidden layers in a neural network, or the depth of a decision tree. Proper hyperparameter tuning can significantly improve a model's performance, making it essential whenever you aim to build a high-performing machine learning model. It helps strike the right balance between model complexity and generalization, preventing issues like overfitting or underfitting. To apply hyperparameter tuning effectively, one can use techniques like grid search, random search, or Bayesian optimization to systematically explore different combinations of hyperparameter values. These methods automate the search process and help identify the configuration that maximizes the model's predictive power, ensuring it performs well on unseen data and real-world applications.

Machine learning **Algorithms** are sets of mathematical rules and procedures that guide the learning process. These algorithms come in various types, such as decision trees, neural networks, and support vector machines, each suited to different types of data and tasks. The choice of algorithm depends on the problem at hand. Years of research have shown that there is no definitive model to outperform every other machine learning approach. So usually, the scientist has to choose the model based on the task at hand, the available resources, the allowed model complexity, and most importantly, the accumulated experience of scientists and engineers when solving that specific task.

Testing and evaluating a machine learning model is a crucial step in the development process, as it helps assess how well the model generalizes to unseen data. After the training phase, you need to validate the model's performance on a separate dataset, typically called the validation set or test set. This evaluation helps determine if the model is ready for real-world deployment. We will present the most commonly used Evaluation Metrics in Classification Tasks, with respect to *TP (True Positives)*, *FP (False Positives)*, *TN (True Negatives)* and *FN (False Negatives)*:

1. **Accuracy** measures the proportion of correctly classified instances in a classification problem. It's a simple and widely used metric but may not be suitable for imbalanced datasets:

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

2. **Precision** measures the proportion of true positive predictions out of all positive predictions. It's useful when the cost of false positives is high:

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

3. **Recall** measures the proportion of true positive predictions out of all actual positive instances. It's important when missing positive instances is costly:

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

4. **F1** is the harmonic mean of precision and recall, providing a balance between the two metrics. It's especially useful when you want to balance precision and recall, and when dealing with imbalanced data:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The samples representing *True Positives*, *False Positives*, *True Negatives* and *False Negatives* can also be visualized as a matrix, the so-called *Confusion Matrix*, which depicts the predictions of the model, and whether that prediction was correct or not. Below is the example of a textitConfusion Matrix for the case of Binary Classification:

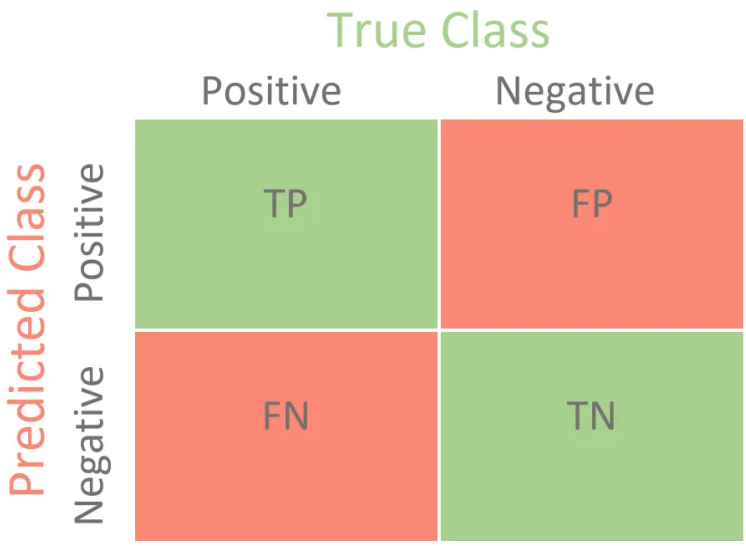


Figure 3.3.1: Confusion Matrix for a Binary Classification Task [1]

Lastly, the most common challenges faced in machine learning that affect the performance and generalization of models, are **overfitting and underfitting**. Overfitting occurs when a model learns to fit the training data too closely, capturing noise and outliers rather than the underlying patterns. This results in poor performance on unseen data as the model fails to generalize effectively. On the other hand, underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in subpar performance even on the training data itself.

The most common ways to address overfitting include:

1. **Regularization** techniques like L1 and L2 regularization add penalty terms to the model's loss function, discouraging it from assigning excessive importance to certain features or parameters, thus preventing overfitting.
2. **Cross-validation** helps in assessing a model's generalization performance. By splitting the data into multiple subsets and evaluating the model on different combinations of training and validation data, you can detect overfitting and fine-tune the model accordingly.

[1] Image from: TowardsDataScience

3. **Feature selection** also plays an important role, by removing irrelevant or redundant features, which can lead to a simpler model and reduced susceptibility to overfitting.

Additionally, in order to combat underfitting, we can:

1. **Increase model complexity**, which involves considering using more complex models with more parameters or layers, such as deep neural networks, to capture intricate patterns in the data.
2. **Feature engineering**, in order to create more informative features or transform existing ones to provide the model with a better representation of the data.
3. **Collect more data**. A larger and more diverse dataset can help the model learn the underlying patterns more effectively and mitigate underfitting.

Finding the right balance between overfitting and underfitting is often a delicate process and may require multiple iterations of model development and evaluation. Regular monitoring and fine-tuning of hyperparameters and model architecture are essential to achieve the best possible performance while avoiding these common pitfalls.

3.4 Deep Learning

The rise of **Deep Learning** in the field of artificial intelligence has been nothing short of revolutionary. Deep learning is a subset of machine learning that has gained prominence due to its ability to automatically learn and make decisions from large volumes of data. This breakthrough in AI has had a profound impact on various industries, leading to advancements in technology, healthcare, finance, and more.

One of the key reasons for the importance of deep learning is its capacity to handle complex, unstructured data like images, text, and audio. This capability has paved the way for remarkable advancements in computer vision, natural language processing, and speech recognition. Deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated superhuman performance in tasks like image classification, object detection, language translation, and speech synthesis.

In real-world applications, deep learning is now ubiquitous. In healthcare, deep learning models have shown promise in diagnosing diseases from medical images, predicting patient outcomes, and even discovering new drug candidates. In the financial sector, these models are used for fraud detection, algorithmic trading, and risk assessment. In the automotive industry, self-driving cars rely on deep learning for perception and decision-making, making them safer and more autonomous. Other examples include virtual assistants like Siri and Alexa, recommendation systems on platforms like Netflix and Amazon, and the personalization of content on social media platforms. Deep learning's adaptability and versatility make it a cornerstone technology in the modern AI landscape, with a profound impact on countless aspects of our daily lives.

Deep Neural Networks

The main idea in the field of *Deep Learning*, is to take traditional architectures, and scale them up, adding more layers and more parameters. In the case of structured/tabular data, this would mean adding numerous "hidden" layers to the pre-existing linear Neural Network ("shallow" network), along with non-linear Activation Functions in between, so that the model can learn non-linear functions as well.

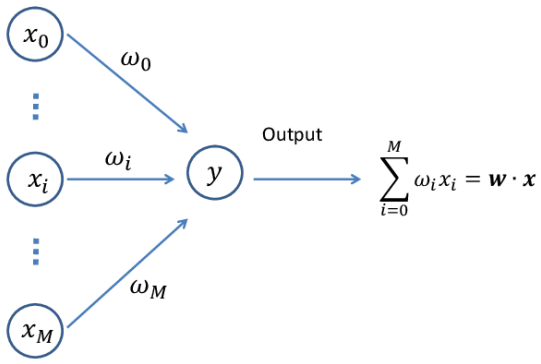


Figure 3.4.1: Single-Layer Neural Network which can only learn Linear functions [11]

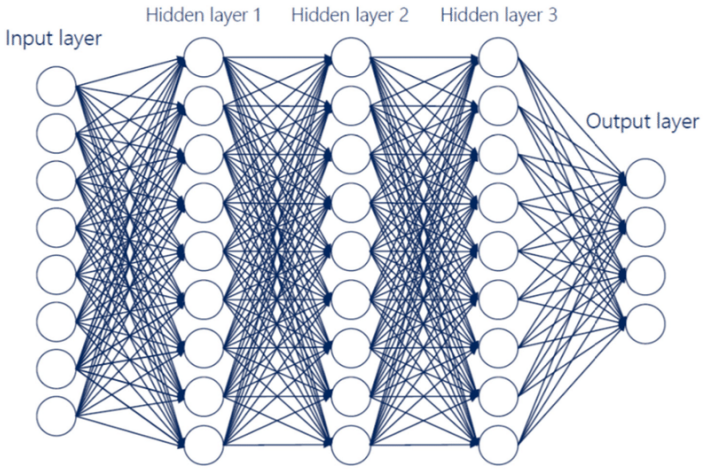


Figure 3.4.2: Deep Neural Network, with multiple hidden layers, theoretically able to learn any function defined on the network's inputs [76]

The key point that enables Deep Neural Networks to learn any non-linear function, is the use of **Activation Functions**. Activation Functions play an important role in the architecture of Neural Networks, because they are the only module that adds *non-linearity* to the whole model. As shown in Figure 3.4.1, the connections between neurons alone, only provides us with a linear mapping from input to output. That's why we add non-linear Activation Functions between each layer of the network. As far as the non-linear function that will be used, many have been shown to work well with modern architectures, and there is no definitive answer, as to which one is the best. The most commonly used ones are presented below.

- **Sigmoid Activation Function:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

Advantages: Sigmoid functions squash input values to a range between 0 and 1, making them suitable for binary classification problems. They are differentiable, making them compatible with gradient-based optimization techniques like backpropagation.

Disadvantages: Sigmoid functions can suffer from vanishing gradients, which can slow down training in deep networks. They also tend to produce outputs close to 0 or 1 for large input values, leading to saturation and slower learning.

Usage: Historically used in hidden layers of shallow networks and output layers for binary classification tasks. Less common in deep neural networks due to vanishing gradient issues.

- **Hyperbolic Tangent (Tanh) Activation Function:**

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Advantages: It has similar advantages to sigmoid but can be considered better due to being zero centred. The output of tanh lies between -1 and 1. The zero center, also makes optimization easier.

Disadvantages: It also has the problem of vanishing gradient but the derivatives are steeper than that of the sigmoid. Hence making the gradients stronger for tanh than sigmoid. Also, similarly to the sigmoid, tanh is computationally expensive to compute.

Usage: Used in hidden layers of neural networks, especially in contexts where zero-centered activations are desirable.

- **Rectified Linear Unit (ReLU) Activation Function:**

$$f(x) = \max(0, x)$$

Advantages: ReLU functions are computationally efficient and have become the default choice for many applications. They mitigate the vanishing gradient problem by providing a non-zero derivative for positive inputs, leading to faster convergence.

Disadvantages: It suffers from the dying ReLU problem. ReLU is always going to discard the negative values (i.e. de-activates it by making it 0). Because of this, the gradient of these units will also become 0 and by now we all know that 0 gradient means no weight update during backpropagation. Simply speaking, the neurons which will go to this state will stop responding to the deviation of input or the error. This, as a result, hampers the ability of the model to fit the data properly.

Usage: Widely used in hidden layers of deep neural networks, especially in convolutional neural networks (CNNs) for image processing.

- **Leaky ReLU Activation Function:**

$$f(x) = \max(\alpha * x, x) \qquad a > 0$$

Advantages: It tries to remove the dying ReLU problem. Instead of making the negative input 0, which was the case of ReLU, it makes the input value really small but proportional to the input. Because of this, the gradient doesn't saturate to 0. If the input is negative, the gradient will be α. As a result, there will be learning for these units as well.

Disadvantages: The specific choice of the leaky slope parameter can be a hyperparameter that needs tuning.

Usage: An alternative to ReLU, commonly used in scenarios where avoiding dead neurons is critical.

- **Exponential Linear Unit (ELU)** [18] **Activation Function:**

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Advantages: ELU combines the advantages of ReLU and Leaky ReLU while mitigating the dying ReLU problem and producing smooth gradients. It can lead to faster convergence and better generalization.

Disadvantages: ELU functions are computationally more expensive due to the exponential computation, and there is also the hyperparameter α which needs to be tuned.

Usage: ELU is gaining popularity in various neural network architectures as an alternative to ReLU and Leaky ReLU.

- **Scaled Exponential Linear Unit (SELU)** [56] **Activation Function:**

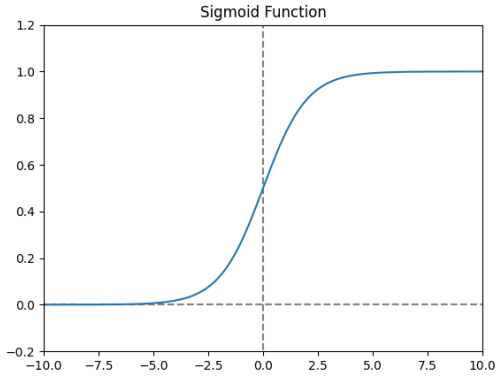
$$f(x) = \begin{cases} \lambda x & x \geq 0 \\ \lambda\alpha(e^x - 1) & x < 0 \end{cases}$$

Advantages: SELU is designed to induce self-normalizing properties in deep networks, helping to maintain consistent activations and gradients during training. It can lead to highly stable and efficient training.

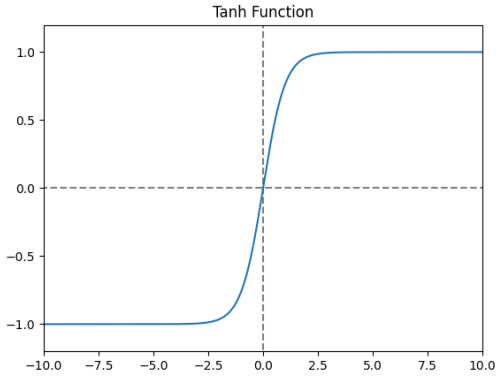
Disadvantages: SELU works best in specific scenarios, and its hyperparameters require careful tuning.

Usage: SELU is used in architectures where self-normalization and stability are crucial, but it is a relatively new activation function so it is not yet used widely in practice. ReLU stays as the preferred option.

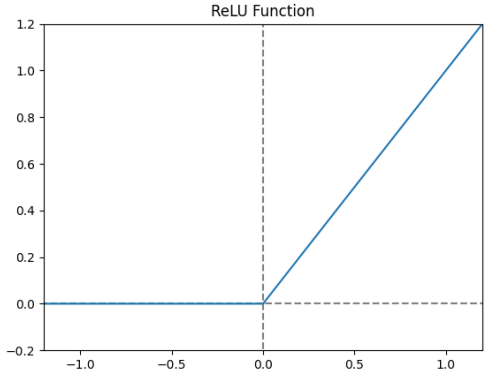
Each activation function has its strengths and weaknesses, making the choice of activation function an important consideration in neural network design, depending on the specific problem and network architecture.



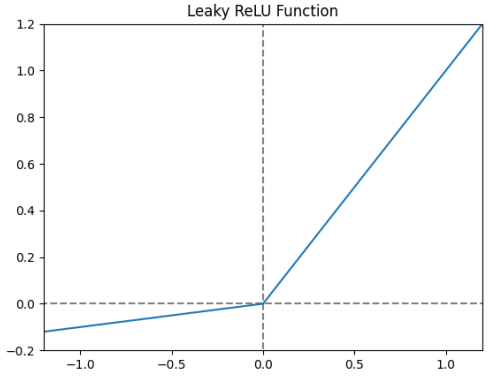
(a) Sigmoid Activation Function



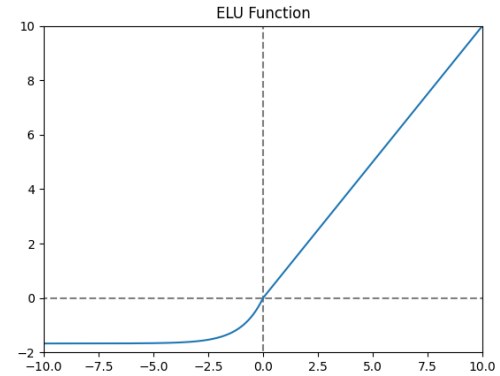
(b) Hyperbolic Tangent Activation Function



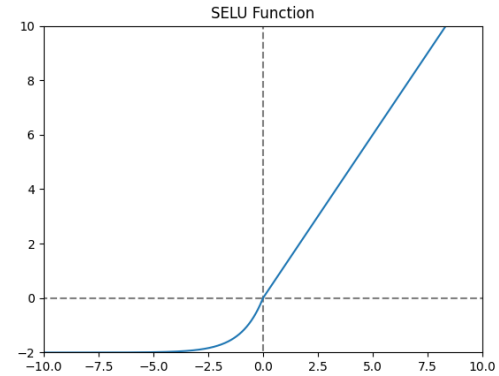
(c) ReLU Activation Function



(d) Leaky ReLU Activation Function



(e) ELU Activation Function



(f) SELU Activation Function

Figure 3.4.3: Most commonly used Activation Functions

Convolutional Neural Networks

Convolutional Neural Networks (CNNs), a class of deep learning models, have played a pivotal role in the field of computer vision and have had a profound impact on various other domains as well. The history of CNNs dates back to the late 1980s and early 1990s when Yann LeCun, along with his colleagues, developed one of the earliest CNN architectures known as LeNet[62], as shown in Figure 3.4.4. LeNet was primarily designed for handwritten digit recognition and was a breakthrough in its time. However, it wasn't until the mid-2010s that CNNs gained widespread recognition and adoption, thanks to advances in hardware, the availability of large datasets, and improvements in training algorithms.

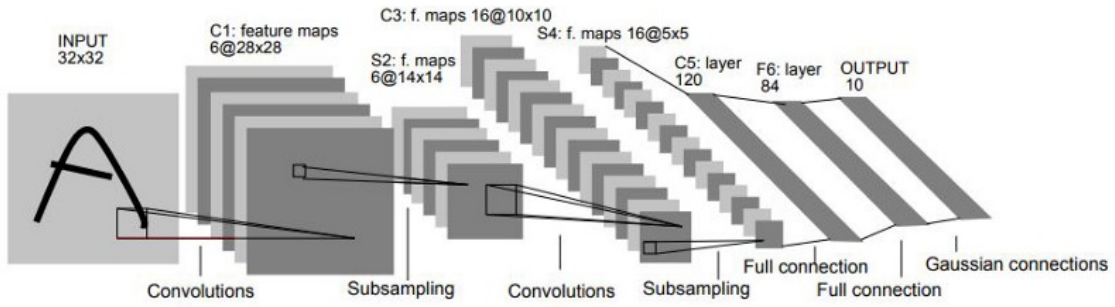


Figure 3.4.4: Original LeNet Architecture [62]

The importance of CNNs lies in their ability to automatically **learn hierarchical features** from data, making them particularly effective for tasks involving image and video analysis. CNNs employ convolutional layers that perform localized and shared-weight filtering operations, which mimic the way the human visual system processes information. This design makes CNNs adept at recognizing patterns and features in images, such as edges, textures, and shapes, in a way that is invariant to translation, rotation, and scale. This remarkable capability has revolutionized a wide range of applications, making CNNs a cornerstone of modern artificial intelligence and machine learning.

The **convolution operator** in Convolutional Neural Networks (CNNs) is a fundamental building block responsible for the network's ability to automatically learn hierarchical features from input data, typically used for image and spatial data processing. It is essential for tasks such as image recognition, object detection, and segmentation. To understand the convolution operator, let's break down its components and how it operates:

1. **Input Data and Filters:**
    - The input to a CNN is typically a multi-dimensional array, often referred to as an image or feature map. Each element in this array represents the value of a pixel or a feature at a specific location.
    - CNNs use small, learnable filters (also called kernels or weights) that have smaller spatial dimensions compared to the input. These filters are initialized with random values and are updated through training.
2. **Convolution Operation:**
    - The convolution operation involves sliding these filters over the input data, element by element, and performing a mathematical operation called convolution at each step.
    - At each position, the filter is aligned with a local patch of the input data, and an element-wise multiplication is performed between the filter and this patch, as shown in Figure 3.4.5
    - The results of these element-wise multiplications are then summed to produce a single value, which forms a single element of the output feature map.

3. **Strides and Padding:**
   - Convolutional layers can be configured with a "stride" parameter, which determines how much the filter shifts (or strides) after each convolution operation. A larger stride reduces the output size.
   - To maintain the spatial dimensions of the input in the output, padding can be added to the input data by adding zeros around its borders. This is known as "zero-padding".
4. **Feature Map Output:**
   - The result of the convolution operation forms one element of the output feature map.
   - By repeating this process for each position of the filter over the input data, a new feature map (also called a convolutional feature map) is generated. Each element in this map represents the presence of a certain feature or pattern in the input data.
5. **Multiple Filters and Depth:**
   - A convolutional layer typically consists of multiple filters, each with its own set of learnable weights. These filters capture different features or patterns.
   - The depth of the output feature map is equal to the number of filters used in the convolution layer.
6. **Non-linear Activation:**
   - After the convolution operation, a non-linear activation function (typically ReLU - Rectified Linear Unit) is applied element-wise to the entire feature map. This introduces non-linearity into the model, allowing it to learn complex patterns.

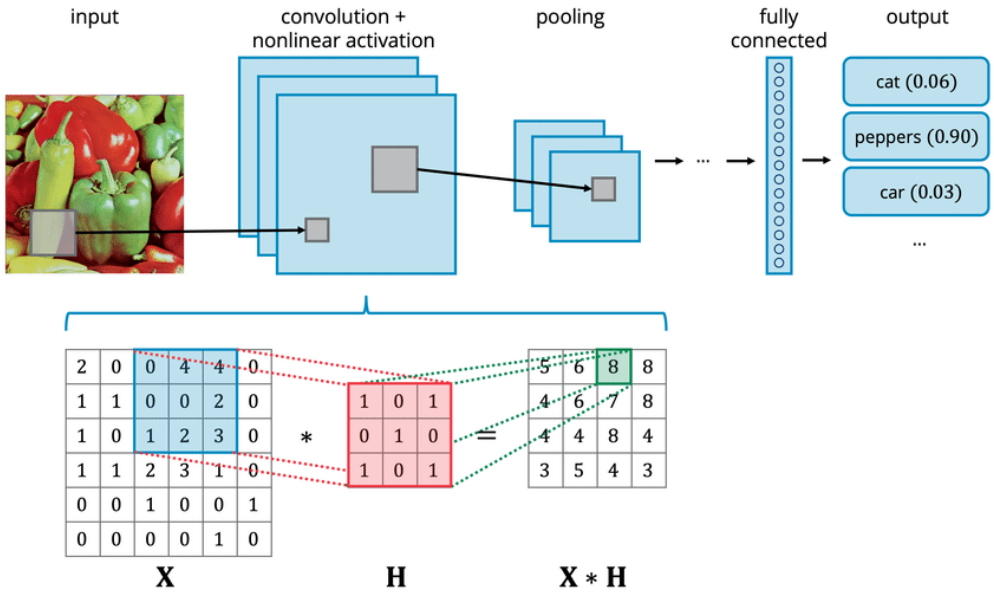


Figure 3.4.5: Convolution Operation in a CNN [17]

The key advantages of using convolution in CNNs are **weight sharing** and **local connectivity**. Weight sharing means that the same set of filter weights is applied at different spatial locations, enabling the network to recognize patterns regardless of their location in the input. Local connectivity means that each output feature depends on a localized region of the input, which helps the network focus on local patterns and reduces the number of parameters, making CNNs computationally efficient.

Through training, CNNs learn the optimal filter weights that allow them to detect relevant features or patterns in the input data, ultimately leading to superior performance in tasks like image recognition, object detection, and more.

In contemporary times, CNNs have found applications in numerous fields. In computer vision, they excel in **object recognition**, **image classification**, and **segmentation tasks**, allowing for advancements in autonomous vehicles, facial recognition, and medical image analysis. CNNs have also extended their reach to natural language processing, where they are used for tasks like sentiment analysis and text classification. In the healthcare sector, CNNs aid in diagnosing diseases from medical images, while in the finance industry, they assist in fraud detection and stock market analysis. Additionally, CNNs are employed in robotics, remote sensing, and even art generation, showcasing their versatility and adaptability across various domains. As technology continues to evolve, the role of CNNs is expected to expand further, making them an integral part of our modern digital landscape.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks specifically designed to handle sequential data and temporal dependencies. They have a rich history dating back to the 1980s when the concept was first introduced. However, it wasn't until the mid-1990s that RNNs gained significant attention, primarily due to the introduction of the **Long Short-Term Memory** (LSTM) architecture [45], which addressed the vanishing gradient problem that had plagued earlier RNN variants. LSTM allowed RNNs to effectively capture long-range dependencies in data, making them a powerful tool for a wide range of applications.

The importance of RNNs lies in their ability to model and process **sequential data**, which is prevalent in various domains, including natural language processing, speech recognition, time series analysis, and more. Unlike feed-forward neural networks, RNNs have recurrent connections that allow them to maintain a memory of previous inputs, making them well-suited for tasks where past context is crucial for understanding current data. This makes them indispensable in tasks such as language modeling, machine translation, and sentiment analysis, where the order of words in a sentence or the context of previous words greatly influences the meaning.

Specifically, RNNs operate by maintaining hidden states that capture information about past inputs. Unlike feed-forward neural networks, which process input data in a one-way flow, RNNs have feedback connections that allow them to maintain a hidden state vector h as they process each input element x_t in a sequence. This hidden state h_t serves as a kind of memory that retains information from previous time steps and influences the network's output at the current time step. The that RNNs process sequential data can also be visualized through "RNN unfolding", as shown in Figure 3.4.6

The key equations governing the operation of a simple RNN cell are as follows:

- *Hidden State Update:*
$$h_t = tanh\left(W_{hh} * h_{t-1} + W_{hx} * x_t + b_h\right)$$
- *Output Computation:*
$$y_t = W_{oy} * h_t + b_o$$

Here, h_t is the hidden state at time step t, x_t is the input at time step t, $tanh$ is the hyperbolic tangent activation function, W_{hh} and W_{hx} are weight matrices, b_h is the bias term for the hidden state, W_{oy} is the weight matrix for the output, and b_o is the bias term for the output. The $tanh$ activation function helps in controlling the information flow through the hidden state by squashing values between -1 and 1.

During training, RNNs use the **BackPropagation Through Time** (BPTT) algorithm to adjust their weights and biases to minimize a loss function, typically associated with a specific task, such as sequence prediction or classification.

However, standard RNNs have limitations, such as the **vanishing gradient problem**, which makes them ineffective at capturing long-range dependencies in sequences. To address this issue, more advanced RNN

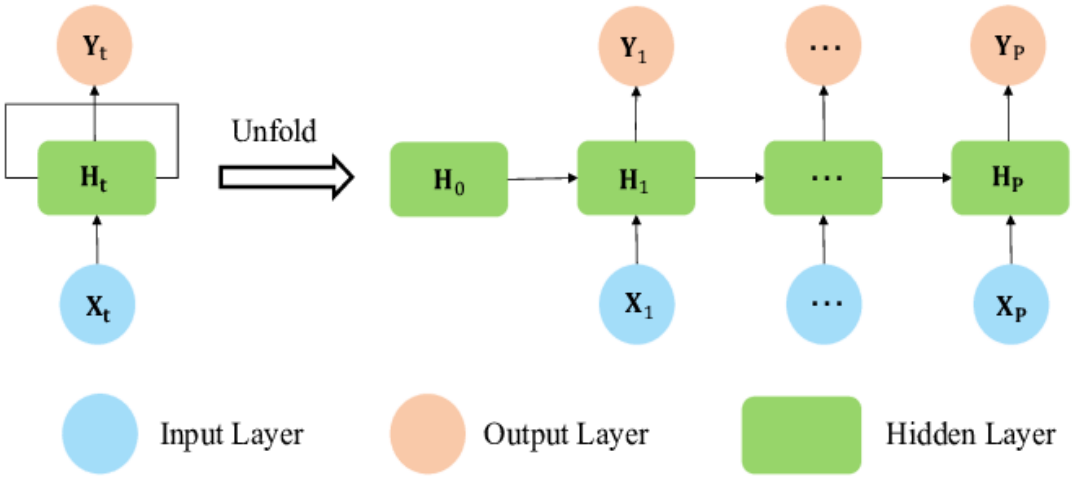


Figure 3.4.6: RNN Unfolding process, which shows the way that sequential data are processed [46]

variants like the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) were introduced. These architectures incorporate gating mechanisms that allow them to selectively retain and update information in the hidden state, making them better at handling long sequences.

LSTM, for instance, uses three gates (input, forget, and output gates) to control the flow of information and can capture long-term dependencies effectively, as shown in Figure 3.4.7. GRU, on the other hand, uses two gates (reset and update gates) and is computationally less complex than LSTM while still achieving comparable performance in many tasks.

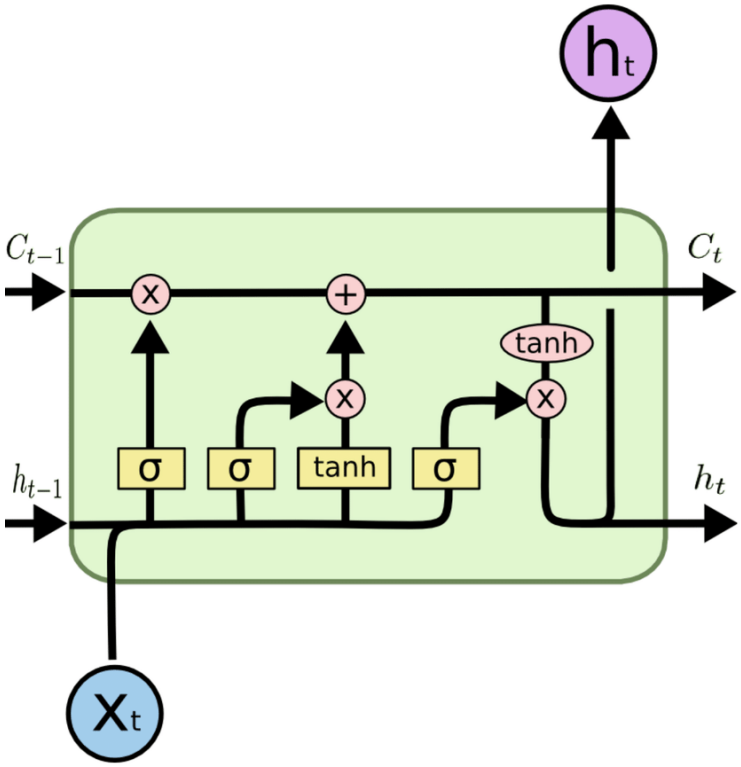


Figure 3.4.7: LSTM Cell Architecture [26]

One of the most common applications of RNNs is in natural language processing. They have revolutionized

tasks like **language generation**, where they are used in text generation models like GPT (Generative Pre-trained Transformer) and in machine translation systems like Google Translate. In addition to NLP, RNNs find extensive use in speech recognition systems, enabling voice assistants like Siri and Alexa to understand and respond to spoken language. RNNs are also widely applied in financial forecasting, where they can capture intricate temporal patterns in stock prices or economic data. Furthermore, they excel in time series prediction and analysis, which has applications in fields ranging from weather forecasting to medical diagnostics.

In summary, Recurrent Neural Networks have a long and evolving history, driven by their ability to model sequential data and temporal dependencies. Their significance lies in their versatility across various domains where sequence processing is essential, and their applications continue to grow as researchers develop more advanced variants and architectures. RNNs have become a cornerstone of modern machine learning and artificial intelligence, enabling groundbreaking advancements in fields such as natural language understanding, speech recognition, and time series forecasting. In the last few years though, with the rise of Transformers and abundance of computational resources, RNNs' popularity has decreased, mostly when it comes to building Language Models.

Transformers

The **Transformer** is a revolutionary machine learning model architecture that has had a profound impact on the field of natural language processing (NLP) and beyond. It was introduced by Vaswani et al. in 2017[112] and has since become a cornerstone in various AI applications. The Transformer architecture is known for its ability to handle sequential data efficiently, thanks to its self-attention mechanism.

The core idea behind the Transformer is **self-attention**, which allows the model to weigh the importance of different elements in a sequence when making predictions. This self-attention mechanism is often referred to as scaled dot-product attention and is computed as follows:

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

Here, Q, K, and V are matrices representing queries, keys, and values derived from the input sequence. The dot product of Q and K is scaled by $\sqrt{d_K}$ to control the magnitude of the output, and the softmax function is applied to obtain the attention weights. These weights are then used to linearly combine the values V to produce the final output.

Parallelism is a key feature of the Transformer architecture, and it plays a crucial role in its efficiency, making it well-suited for training on modern hardware. In traditional recurrent neural networks (RNNs) or convolutional neural networks (CNNs), processing sequential data involved sequential computation. Each step in the sequence depended on the previous one, limiting the potential for parallelization.

In contrast, the Transformer's architecture leverages parallelism in several ways:

1. **Multi-Head Attention:** One of the main components of the Transformer model is multi-head attention. Instead of computing attention for a sequence one element at a time, the model splits the attention mechanism into multiple heads, each of which independently computes attention for different parts of the sequence. These heads operate in parallel, and their results are concatenated and linearly transformed to produce the final output. This parallelism allows the model to capture different types of relationships and dependencies in the data simultaneously.
2. **Positional Encoding:** Transformers don't have built-in notions of position or order in the sequence. To address this, positional encodings are added to the input embeddings. These encodings are designed to convey the position information of each element in the sequence. Importantly, they are added element-wise, so the computation of positional encodings is also parallelizable. This addition ensures that the model can still capture the sequential order of the data.

3. **Batch Processing:** Transformers are highly amenable to batch processing. Multiple input sequences can be processed in parallel within a batch, which not only speeds up training but also allows for better generalization as the model sees a variety of examples in each batch. Batch processing is a fundamental technique used in deep learning to harness the computational power of GPUs and TPUs.

4. **Model Parallelism:** In cases where the model is extremely large and cannot fit on a single GPU or TPU, model parallelism can be employed. This involves splitting the model into parts that can be processed on separate devices, allowing for even greater parallelization.

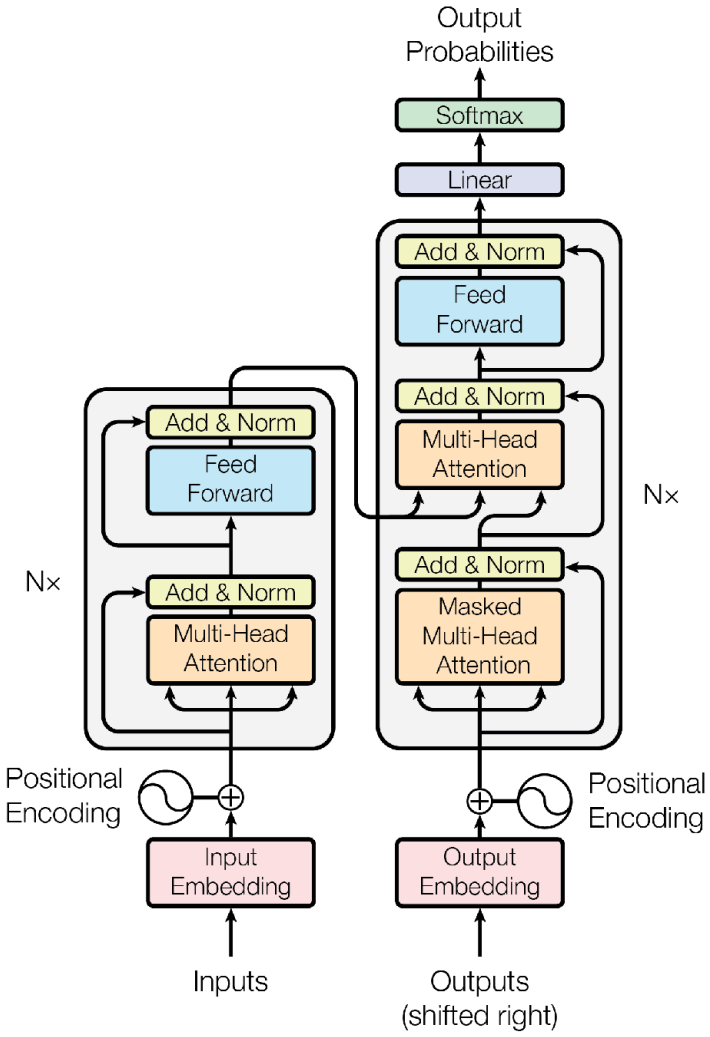


Figure 3.4.8: The original Transformer Architecture, as presented in [112]

This parallelism is a key reason why Transformers have been able to handle much larger datasets and achieve state-of-the-art results in NLP and other domains. It enables researchers and practitioners to train deep models on massive datasets efficiently, leading to breakthroughs in AI applications such as language translation, sentiment analysis, and text generation, as well as in computer vision tasks and beyond.

The Transformer architecture has found applications far beyond NLP. It has been adapted for computer vision tasks, such as image classification and object detection, under the name Vision Transformer (ViT)[23]. It has also been used in recommendation systems, speech recognition, and even in the field of reinforcement learning. The versatility and effectiveness of the Transformer architecture have made it a cornerstone of modern AI research and development, and it continues to drive innovation in various domains.

3.5 Autoencoders

Vanilla Autoencoders

Autoencoder models are a fundamental and versatile class of neural networks in the field of machine learning. They are primarily used for unsupervised learning and dimensionality reduction tasks.

Autoencoders have a rich history dating back to the 1980s. They were initially proposed as a neural network architecture for learning efficient representations of data. The idea behind autoencoders was inspired by the need to reduce the dimensionality of data while preserving its essential information. Over the years, various advancements in neural network research and the availability of large datasets have made autoencoders more popular and powerful.

An autoencoder consists of an encoder and a decoder, both typically implemented as neural networks. The encoder maps the input data into a **lower-dimensional representation** (latent space), and the decoder reconstructs the original input from this representation. The core idea is that the encoder learns to capture essential features of the data, while the decoder learns to reconstruct the input from these features. Mathematically, the encoder can be represented as:

$$z = f(x)$$

where x is the input data, z is the encoded representation, and f is the encoder function parameterized by the neural network weights.

The decoder, on the other hand, aims to reconstruct the input data from the encoded representation:

$$x' = g(z)$$

where x is the reconstructed data, z is the encoded representation, and g is the decoder function parameterized by its own set of weights.

The loss function used for training autoencoders is typically a reconstruction loss, such as **Mean Squared Error** (MSE), which measures the difference between the input data and the reconstructed data:

$$L(x, x') = ||x - x'||^2$$

Autoencoders are trained using an optimization algorithm like stochastic gradient descent (SGD) or its variants. The goal is to **minimize the reconstruction loss** by adjusting the weights of the encoder and decoder. The training process involves feeding the input data through the encoder to obtain the encoded representation and then passing this representation through the decoder to reconstruct the data. The gradients of the reconstruction loss with respect to the network weights are computed and used to update the weights through backpropagation.

The training process aims to encourage the encoder to learn meaningful and compact representations of the data, which can capture its essential features while discarding noise and non-essential information.

During inference, autoencoders can be used for various tasks:

- **Dimensionality Reduction:** The encoder can be used to project high-dimensional data into a lower-dimensional latent space, which can be useful for visualization, feature extraction, or downstream tasks.
- **Anomaly Detection:** Autoencoders can identify anomalies in data by comparing the reconstruction loss of a sample with a predefined threshold. Samples with high reconstruction loss are considered anomalies.
- **Data Generation:** By sampling from the latent space and passing the samples through the decoder, autoencoders can generate new data samples that are similar to the training data.

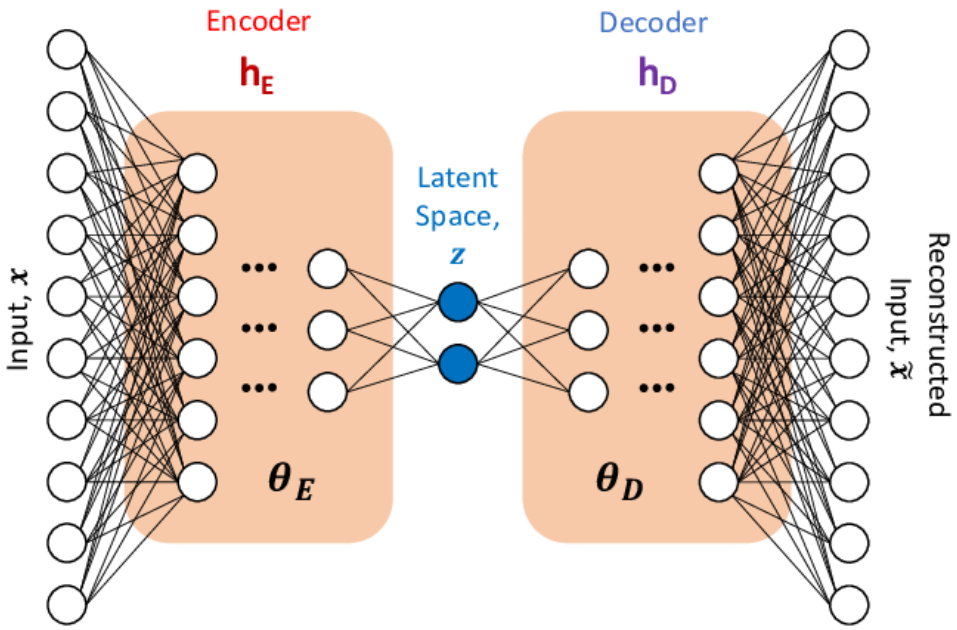


Figure 3.5.1: Architecture of the Autoencoder Network [24]

Autoencoders and the Manifold Hypothesis

The **manifold hypothesis** is a fundamental concept in machine learning that posits that real-world high-dimensional data lies on or near a lower-dimensional manifold embedded within that high-dimensional space. In simpler terms, it suggests that complex data, such as images or natural language, can be effectively represented in a lower-dimensional space that captures its essential features. This idea arises from the observation that many real-world datasets exhibit a high degree of redundancy and structure, which can be exploited for tasks like classification, clustering, and data generation.

Autoencoders have a close relationship with the manifold hypothesis. They are neural network architectures explicitly designed to uncover and learn these lower-dimensional representations, also known as the **latent space**, from high-dimensional data. By training an autoencoder, the model attempts to capture the underlying structure of the data and map it onto a lower-dimensional manifold. The encoder component of the autoencoder learns to project the input data onto this manifold, while the decoder learns to reconstruct the data from points on the manifold.

In essence, autoencoders aim to embody the manifold hypothesis by learning a compressed and meaningful representation of the data in a lower-dimensional space. This representation can then be utilized for various tasks, such as dimensionality reduction, data denoising, and even data generation. When the manifold hypothesis holds true, autoencoders can provide effective solutions for understanding and manipulating complex high-dimensional data.

Variational Autoencoders

A **Variational Autoencoder** (VAE) is a probabilistic generative model that builds upon the traditional autoencoder architecture by adding a probabilistic interpretation to the latent space. VAEs are designed to not only capture meaningful representations of data but also to generate new data samples that are similar to the training data. They are particularly useful for generative tasks, such as image generation, text generation, and data denoising.

A VAE consists of two main components, similar to a standard autoencoder: an encoder and a decoder. However, in a VAE, the encoder doesn't produce a deterministic latent representation but instead produces a probability distribution over the latent space. 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:

$$q(z|x) = \mathcal{N}(\mu(x), \sigma(x))$$

Here, z represents the latent variables sampled from the Gaussian distribution. The decoder, like in a standard autoencoder, takes samples from the latent space and reconstructs the data x'.

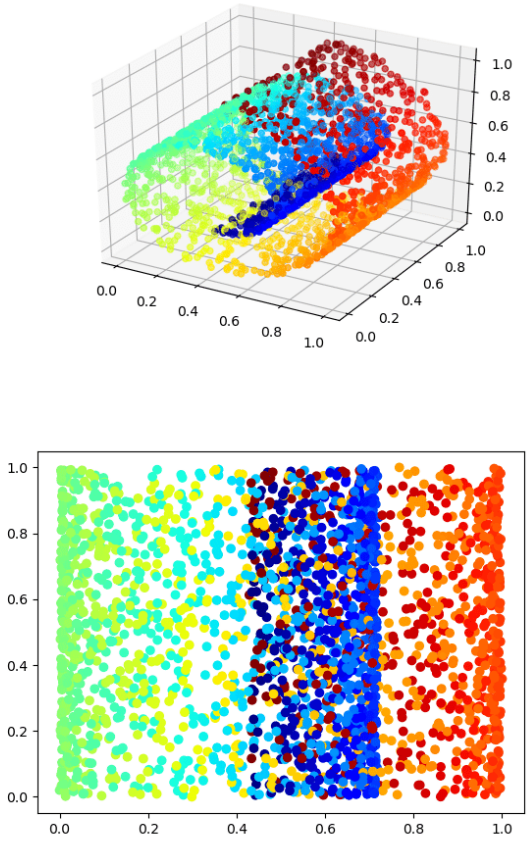


Figure 3.5.2: An example where the manifold hypothesis holds true. The data, although originally in a 3-dimensional space, can be successfully embedded in a 2-dimensional latent space, without any loss of information.

The training of a VAE involves two main objectives: reconstruction and regularization. The reconstruction objective is similar to that of a standard autoencoder, aiming to minimize the reconstruction loss (e.g., mean squared error) between the input x and the reconstructed x'.

The regularization objective encourages the latent space to follow a specific prior distribution, usually a standard Gaussian $\mathcal{N}(0, 1)$. This regularization is achieved through the **Kullback-Leibler** (KL) divergence term, which measures the difference between the learned distribution $q(z|x)$ and the target Gaussian distribution. The overall loss function for training a VAE is a combination of the reconstruction loss and the KL divergence term:

$$L(x, x', \mu, \sigma) = ReconstructionLoss\,(x, x') + KLDivergence\,(q(z|x), \mathcal{N}(0, 1))$$

The KL divergence term encourages the encoder to produce latent representations that are close to the prior distribution while still allowing for variation in the data.

During inference in a VAE, you can sample from the learned latent space distribution $q(z|x)$ to generate new data samples. To generate data samples, you sample a point z from the Gaussian distribution in the latent space and pass it through the decoder to produce x'. By controlling the properties of the sampled z(e.g., its mean and standard deviation), you can generate data samples with different characteristics.

VAEs are also used for various other tasks, including data imputation (replacing missing values in data), data denoising (removing noise from data), and semi-supervised learning (combining labeled and unlabeled data for classification tasks).

In summary, Variational Autoencoders are an extension of standard autoencoders that introduce probabilistic modeling into the latent space. They are capable of both capturing meaningful data representations and generating new data samples. Training involves minimizing a combination of a reconstruction loss and a KL divergence term to encourage a specific prior distribution in the latent space. VAEs have found wide applications in generative modeling, data generation, and various unsupervised learning tasks.

Chapter 4

Graphs

In this chapter, we will delve into the field of Graph Theory. Firstly, we will explain all the necessary fundamental definitions, in order to present one of the core concepts of this thesis, the task of Graph Similarity. Specifically, we will thoroughly analyze proposed algorithms that compute Graph Edit Distance, and then present some of the most popular Graph Kernels used to tackle this problem. Finally, we will introduce a specific type of graphs, the Scene Graphs, which is the structure implemented in the main dataset that will be used in the experiments.

Contents

4.1 Definitions **54**
4.2 Graph Similarity **57**
4.2.1 Graph Edit Distance 58
4.2.2 Graph Kernels 59
4.3 Scene Graphs **64**
4.4 Related Work **66**

4.1 Definitions

Basic Graph Definition

A graph, in the context of graph theory, is formally defined as a mathematical structure denoted by $G = (V, E)$, where V represents a set whose constituents are referred to as **vertices**, and E signifies a set of paired vertices, designating them as **edges**, or at times, as links or lines.

The vertices x and y within an edge x, y are denominated as the endpoints of the edge. This edge is deemed to connect or join the vertices x and y, and is incident upon them. It should be noted that a vertex may remain devoid of any edge association, in which case it remains unconnected to any other vertex. A **multigraph** serves as a generalization allowing for the existence of multiple edges possessing the same pair of endpoints. Some literature simplifies multigraphs by simply referring to them as graphs.

It is pertinent to mention that certain scenarios permit graphs to encompass **loops**, signifying edges that connect a vertex to itself. In order to accommodate loops, the pairs of vertices within set E must permit repetition of a vertex, resulting in the categorization of such generalized graphs as graphs with loops or, more succinctly, as graphs when it becomes evident from the context that loops are permissible.

Primarily, the set of vertices V is presupposed to be finite, which, in turn, implies that the set of edges is also finite. Although infinite graphs are occasionally contemplated, they are often construed as a specialized form of binary relation. This classification stems from the fact that many results established for finite graphs do not carry over to the infinite realm or necessitate distinct methodologies for proof.

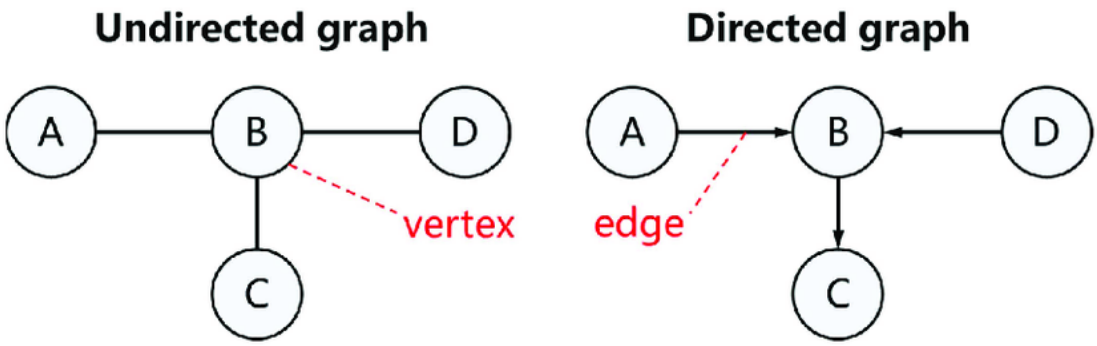


Figure 4.1.1: Example of a Directed and an Undirected Graph [67]

An empty graph is characterized by an absence of vertices, consequently leading to an empty set of edges. The order of a graph is quantified by its number of vertices, denoted as $|V|$. Meanwhile, the size of a graph is determined by its number of edges, represented as $|E|$. Nonetheless, certain contexts, such as the expression of algorithmic computational complexity, involve defining size as $|V| + |E|$; this adjustment ensures that a non-empty graph is not erroneously deemed to possess a size of zero. The **degree** or valency of a vertex corresponds to the count of edges that are incident upon it. In the presence of loops within a graph, each loop contributes twice to the degree of its associated vertex.

Within a graph of order n, the **maximum degree** of any vertex does not exceed $n - 1$ (or $n + 1$ if loops are permitted), and the upper limit for the number of edges is $\frac{n(n-1)}{2}$ (or $\frac{n(n+1)}{2}$ if loops are allowed).

The edges within a graph establish a symmetric relation among the vertices, which is termed the adjacency relation. Specifically, two vertices, x and y, are considered adjacent if and only if x, y constitutes an edge. It is worth noting that a graph may be fully characterized by its adjacency matrix A, a square matrix of dimensions n × n. In this matrix, the entry A_{ij} indicates the quantity of connections from vertex i to vertex j. In the case of a simple graph, A_{ij} assumes either the value 0, signifying disconnection, or 1, signifying connection. Furthermore, A_{ii} is consistently set to 0, as an edge in a simple graph cannot originate and terminate at the same vertex. Graphs that incorporate self-loops are identified by instances where some or all A_{ii} assume positive integer values, while multigraphs, featuring multiple edges linking vertices, are characterized by situations where some or all A_{ij} take on positive integer values. Notably, **undirected**

graphs exhibit a symmetric adjacency matrix, reflecting the equality $A_{ij} = A_{ji}$, visualized as an example in Figure 4.1.1.

Another important distinction in graphs,is the categorization between **weighted and unweighted graphs**. The difference lies in the presence or absence of assigned values to the edges connecting nodes. In a weighted graph, each edge is associated with a numerical weight or cost, often representing factors such as distance, time, or cost, which quantifies the relationship between connected nodes. These weights introduce additional information and complexity, enabling more precise modeling of real-world scenarios. Conversely, in an unweighted graph, all edges are considered equal, with no assigned numerical values, making it suitable for representing relationships where only connectivity matters without regard to specific magnitudes or distinctions among connections.

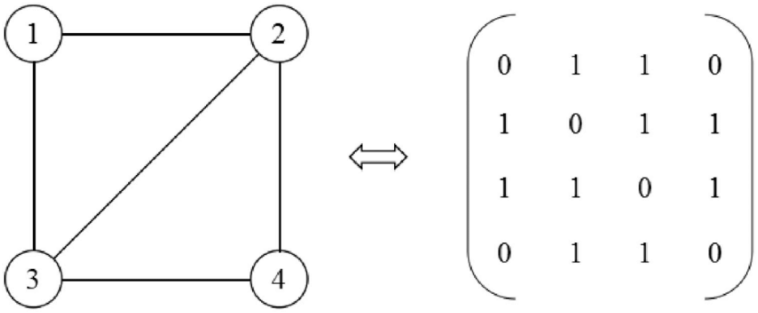


Figure 4.1.2: Example of Construction of Adjacency Matrix from an Undirected Graph [124]

Walks, Paths, Cycles and Trails

In graph theory, various terms are used to describe specific patterns or sequences of edges within a graph. Four such terms are path, walk, cycle, and trail, each with distinct definitions and characteristics.

- **Path:** A path in a graph is a sequence of vertices where each vertex is connected to the next one by an edge. In other words, it's a series of nodes in which you can travel from one node to another by following the edges, without revisiting any node. Paths can be of varying lengths, and they provide a way to explore connections and routes within a graph. A path can be as short as a single edge (a direct connection between two nodes) or extend through multiple edges.
- **Walk:** A walk is a more general concept than a path. It is also a sequence of vertices connected by edges, but unlike a path, it allows for revisiting nodes and using the same edge multiple times. A walk can represent any route through a graph, including backtracking and retracing steps. Walks are often used to analyze the traversability of a graph or study specific sequences of movements.
- **Cycle:** A cycle is a special type of walk in which the starting and ending vertices are the same, and there are no repetitions of other vertices (except for the first and last). In other words, it's a closed path where you return to the same vertex without revisiting any other vertices in between. Cycles are essential in graph theory as they help identify structures like circuits or loops within a graph.
- **Trail:** A trail is similar to a walk but imposes two extra conditions. Firstly, the starting and ending nodes must be different, and secondly, only vertices can be repeated, not the edges. Trails are often used to explore graphs while keeping track of the specific edges used. They are useful for understanding the flow and connectivity within a graph.

Paths, walks, cycles, and trails are fundamental concepts in graph theory that describe different types of sequences of vertices and edges within a graph. They can be used to extract useful Graph Metrics, that will be especially useful at defining some important Graph Kernels later on.

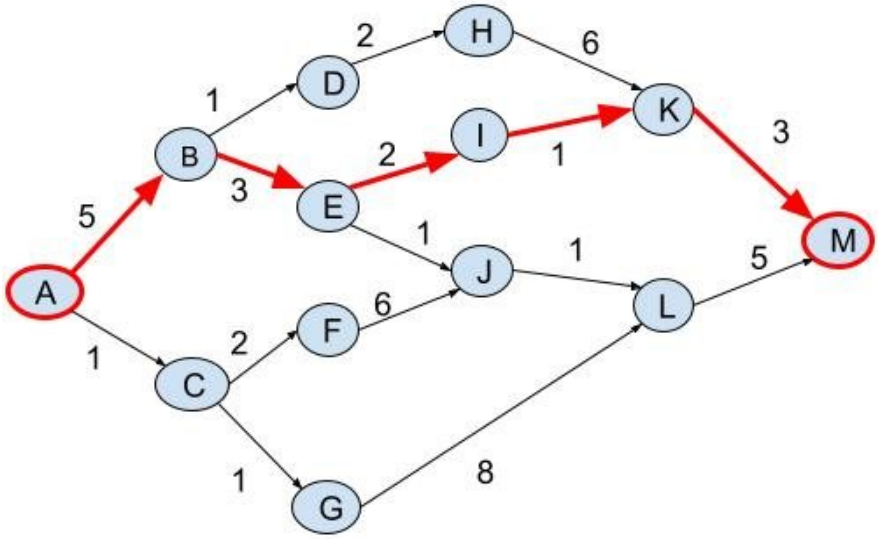


Figure 4.1.3: Example of a Path in a Weighted, Directed Graph [25]

Heterogeneous Graphs

Heterogeneous graphs are a specialized type of graph data structure that play a crucial role in representing complex, diverse relationships and entities in various domains such as social networks, recommendation systems, and knowledge graphs. Unlike homogeneous graphs, which model relationships between entities of the same type (e.g., social networks where individuals connect with other individuals), heterogeneous graphs allow connections between entities of different types. The formal definition of a heterogeneous graph involves nodes and edges of distinct types, thereby capturing rich and diverse relationships in a single graph.

What distinguishes heterogeneous graphs from other types of graphs is their ability to represent and analyze **complex, multi-modal data**. In a heterogeneous graph, nodes can represent different types of entities, such as users, products, movies, or concepts, and edges can represent various types of relationships, such as user-item interactions, actor-movie appearances, or concept-concept associations. This modeling flexibility is invaluable for solving real-world problems where entities and their relationships are multifaceted.

Heterogeneous graphs find applications in a wide range of domains. They are commonly used in recommendation systems to model user-item interactions of different types, enabling more accurate and diverse recommendations. In biology, they are employed to represent diverse biological entities (genes, proteins, diseases) and their complex interactions, aiding in the discovery of potential drug targets or understanding disease mechanisms. Knowledge graphs, which store structured information about the world, are often represented as heterogeneous graphs to capture relationships between entities of various types, enhancing semantic search and question answering.

Meta-paths are a fundamental concept in the context of heterogeneous graphs, especially when it comes to analyzing and traversing the complex relationships that exist between different types of entities within such graphs. A meta-path is essentially a high-level abstraction that defines a specific sequence of node types and edge types in a heterogeneous graph. It provides a structured way to navigate through the graph while specifying the types of nodes and relationships encountered along the path.

Here's a breakdown of key elements in a meta-path:

- **Node Types:** In a meta-path, you specify the types of nodes you want to traverse. These node types represent different entities in the graph. For example, in a movie recommendation system, node types could be "User," "Movie," and "Genre."
- **Edge Types:** You also specify the types of edges that connect the nodes in the path. These edge types represent the relationships or interactions between the entities. In the movie recommendation system,

edge types could be "User-Rated-Movie" and "Movie-Belongs-To-Genre."

By combining these node and edge types in a specific sequence, you define a meta-path that characterizes a particular type of relationship or information flow within the heterogeneous graph. For example, a meta-path in the movie recommendation system could be "User-Rated-Movie -> Movie-Belongs-To-Genre -> Movie-Belongs-To-Genre," which signifies a path where a user rates a movie, and that movie belongs to two genres.

It should be noted, that, according to the specific heterogeneous graph that we want to define a meta-path on, we don't necessarily need to acknowledge both Node Types and Edge Types. For example, we might have a graph that has distinguished types only on the nodes, not the edges. In that case, the meta-path will be a sequence consisting purely of node types, as shown in Figure 4.1.4.

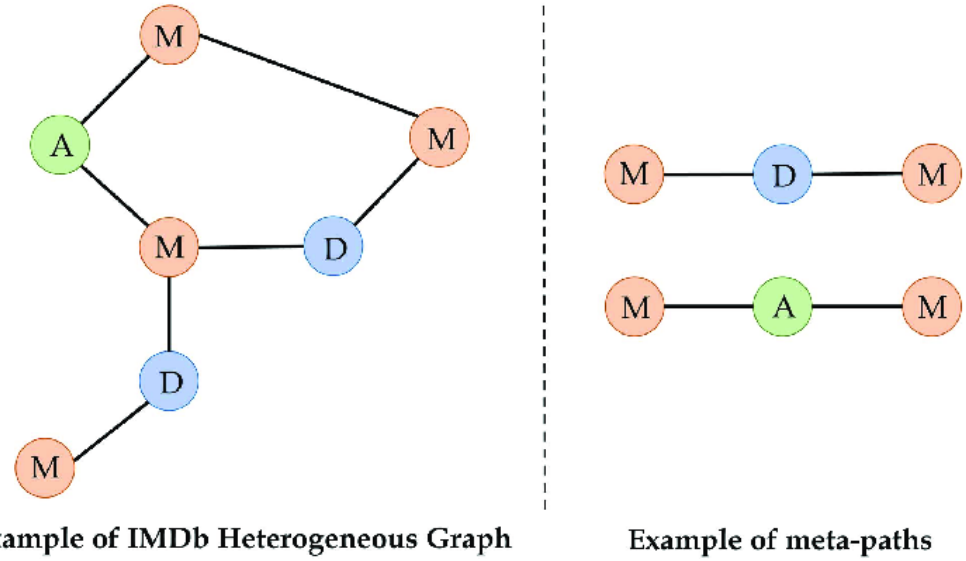


Figure 4.1.4: Example of a Heterogeneous Graph, along with meta-paths defined on it [85]

Processing heterogeneous graphs involves specialized algorithms and techniques. One key challenge is the heterogeneity of nodes and edges, which requires methods for handling different types of data and relationships. This includes meta-path-based algorithms that define meaningful paths through the graph for specific tasks, **node embedding techniques** like GraphSAGE[40] or node2vec[37] to learn representations of nodes, and attention-based models that can capture dependencies between different types of entities and relationships. Additionally, graph neural networks (GNNs) have gained popularity for their ability to propagate information across heterogeneous graphs and perform tasks like node classification, link prediction, and recommendation.

4.2 Graph Similarity

The Graph Similarity problem involves assessing the **structural resemblance** between two graphs, typically represented as networks of interconnected nodes and edges. It encompasses various applications, such as biological network analysis, social network comparison, and recommendation systems. The problem's history can be traced back to the mid-20th century, with roots in graph theory and computational mathematics. Over time, it has gained prominence due to the increasing importance of graph data in modern applications, leading to the development of diverse algorithms and techniques for measuring graph similarity, including graph kernels, graph edit distance, and spectral methods, each catering to specific use cases and data types. The Graph Similarity problem remains an active area of research with broad practical implications in various domains.

Graph Edit Distance (GED) is a fundamental concept within the realm of the Graph Similarity problem, playing a pivotal role in quantifying the dissimilarity between two graphs. GED measures the minimum number of edit operations required to transform one graph into another, where edits can involve adding, deleting, or modifying nodes and edges. This metric offers a comprehensive way to assess structural differences between complex network representations, making it an invaluable tool in fields like pattern recognition, image analysis, molecular biology, and social network comparison. The Graph Edit Distance provides a versatile framework for capturing and quantifying the extent of structural dissimilarity between graphs, enabling researchers and practitioners to gain insights into diverse data domains where graph data is prevalent.

4.2.1 Graph Edit Distance

Graph Edit Distance (GED), is a measure of similarity between two graphs, g_1 and g_2. The fundamental idea of graph edit distance, was first formalized by Alberto Sanfeliu and King-Sun Fu in 1983[94]. In order to correctly formulate GED, we need to define a set of *Graph Edit Operations*. These, typically include:

- **Vertex insertion** to introduce a single new labeled vertex to a graph
- **Vertex deletion** to remove a single (often disconnected) vertex from a graph.
- **Vertex substitution** to change the label (or color) of a given vertex.
- **Edge insertion** to introduce a new colored edge between a pair of vertices.
- **Edge deletion** to remove a single edge between a pair of vertices.
- **Edge substitution** to change the label (or color) of a given edge.

When provided with both a source and a target graph, the objective is to eliminate certain vertices and edges from the source graph, rename some of the remaining vertices and edges, and potentially add new vertices and edges. This process is aimed at ultimately achieving the desired target graph. So now, the mathematical formulation of the graph edit distance between two graphs g_1 and g_2, is:

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

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

The most common process that practical algorithms use to find the Graph Edit Distance, is to view it as an optimal-path-finding problem. That way, traditional algorithms such as A^* are implemented to solve the problem.

Graph Edit Distance Approximations

Even though Graph Edit Distance is a powerful tool in graph analysis and comparison, unfortunately it has been proven that the problem of *computing the exact Graph Edit Distance between graph g_1 and g_2 is NP-hard*[125].

In order to tackle this problem, we are going to present the four most common techniques and optimizations that modern algorithms use to compute GED:

1. **Bipartite Heuristic:** This method was presented by Riesen et al. (2007)[92], and the contribution, is a new heuristic function for the A^* algorithm, speeding up the search for the target path. This approach still provides an optimal solution, which means that this sped-up algorithm can't solve the

problem deterministically in polynomial-time. It is rarely used, when the *exactness* of the similarity measure between the two graphs is of great importance.

The proposed bipartite heuristic function works by estimating the future costs of transforming one graph into another using a fast but suboptimal bipartite graph matching algorithm. This algorithm assigns nodes and edges of the two graphs to each other in a way that minimizes the total cost of the assignments. By summing up the costs of the minimum cost node and edge assignments, the algorithm provides an approximation of the real future costs. Although this approximation does not consider any structure preserving constraints, it provides a lower bound of the future costs, which is guaranteed to return the exact graph edit distance of two given graphs. By using this heuristic function in an A^* implementation, the resulting procedure is guaranteed to return the optimal edit distance, and can significantly speed up the computation of graph edit distance compared to a brute force algorithm or a standard tree search approach.

2. **A* Beamsearch:** This optimization by Neuhaus et al. (2006)[81] is the first in the list that tries finds an *approximate* Edit Distance, therefore highly speeding up the process. The Beamsearch algorithm is a modification of the standard A^* algorithm that limits the number of nodes expanded in the search tree. Instead of expanding all successor nodes in the search tree, only a fixed number of nodes are kept in the *un-discovered* set at all times. Whenever a new partial edit path is added to the *un-discovered* set, only the s partial edit paths with the lowest costs are kept, and the remaining partial edit paths in the set are removed. This means that not the full search space is explored, but only those nodes are expanded that belong to the most promising partial matches.

   This optimization reduces the computational complexity of the algorithm, making it faster and more efficient. However, because the algorithm is not exploring the full search space, it may not find the optimal solution. Instead, it returns a suboptimal solution that is close to the optimal solution.

3. **Bipartite Matching:** This approximate algorithm by Fankhauser et al. (2011)[27] finds a suboptimal solution to the GED problem, by not taking edge information into account in the traditional way, when computing edit distance. Specifically, it computes the exact edit distance when taking into account only the nodes insertions/deletions/substitutions, and then through that, it infers the suboptimal graph edit distance of the original graphs.

   To achieve this, firstly the node-edit-distance problem is transformed to a **Linear Sum Assignment Problem** (LSAP), and then it is solved in polynomial time using the Jonker-Volgerand algorithm[49]. Overall, it is the most widely used variation of the Graph Edit Distance algorithm, because of its efficient approach, and satisfactory results.

4. **Hausdorff Matching:** This algorithm proposed by Fischer et al. (2014)[30] works in a similar way to the previous one, but it employs a quadratic-time approximation algorithm based on Hausdorff Matching as a heuristic function to compute the graph edit distance. The proposed heuristic reduces the search space and speeds up the runtime of the algorithm by one order of magnitude compared to plain A* search.

The most commonly used variations of the original GED algorithm are the *Bipartite Matching* and *Hausdorff Matching*, because they both offer a good trade-off between the accuracy of the final approximate solution and the computational cost.

4.2.2 Graph Kernels

Kernels play a crucial role in various machine learning tasks, particularly in the domain of pattern recognition and data analysis. They are essential for transforming data into a suitable format for learning algorithms

to operate effectively. Kernels are often used in the context of Support Vector Machines (SVMs) and kernel methods, but their applications extend to various other machine learning techniques, including kernel principal component analysis (PCA) and kernelized clustering.

The importance of kernels lies in their ability to implicitly map data into higher-dimensional spaces, where complex patterns and relationships become more apparent. This transformation enables linear algorithms to work effectively on nonlinear problems, as it implicitly captures the similarity or dissimilarity between data points. By using kernels, one can avoid the computational burden of explicitly **mapping data to higher dimensions** while still benefiting from the enhanced representational power. This is also the main idea behind the Kernel Trick.

The **Kernel Trick** is a fundamental concept in machine learning, especially in the context of SVMs and kernel methods. It's a clever mathematical technique that allows you to implicitly compute the dot product between data points in a higher-dimensional feature space without actually having to explicitly transform the data into that space. This is incredibly useful because it allows linear algorithms to work effectively on nonlinear problems.

Here's a more detailed explanation of the kernel trick:

1. **Motivation:** In many real-world problems, the data may not be linearly separable in its original feature space. Linear classifiers like SVMs work well when data is linearly separable, but when it's not, they struggle. The kernel trick provides a way to handle these non-linearities by projecting the data into a higher-dimensional space where it might become linearly separable.
2. **The Kernel Function:** At the core of the kernel trick is the kernel function, often denoted as $K(x, y)$. Given two data points, x and y, the kernel function computes the dot product of these points in a feature space, without explicitly calculating the coordinates of that space. Common kernel functions include the linear kernel ($K(x, y) = x \cdot y$), polynomial kernel ($K(x, y) = (x \cdot y + c)^d$), and Gaussian (Radial Basis Function) kernel ($K(x, y) = exp(-\gamma||x - y||^2)$).
3. **Implicit Mapping:** Instead of mapping the data explicitly into a higher-dimensional space, which can be computationally expensive and potentially infinite, the kernel trick computes the dot product directly in the original feature space using the kernel function. So in the end, the kernel function is formulated as followed:

   $$k(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >$$

   Where $\phi : \mathcal{X} \to \mathcal{V}$ is the function that maps the original $\mathbf{x}$ and $\mathbf{x}'$ into the higher-dimensional space. This allows linear classifiers, like SVMs, to operate in this implicit higher-dimensional space without the need for the actual transformation ϕ.
4. **Kernel SVMs:** In SVMs, the kernel trick is particularly powerful. When you apply a kernel function in an SVM, you effectively transform the decision boundary into a more complex shape in the higher-dimensional space. This enables SVMs to separate data that is not linearly separable in the original space.
5. **Computational Efficiency:** One of the significant advantages of the kernel trick is its computational efficiency. You only need to compute the kernel function for pairs of data points, which can be much faster than explicitly mapping data into a high-dimensional space, especially when dealing with large datasets.

Graph kernels, in particular, are specialized for handling structured data, such as graphs or networks. Graph kernels are designed to measure the similarity between two graphs, which is a fundamental task in applications like graph classification, clustering, and link prediction. The process of graph kernels involves defining a function that quantifies the similarity between substructures of two graphs. This function works the same way that we described above. That is, they implicitly compute the dot product of the input graphs in a higher-dimensional Hilbert space (where the dot-product operation is well-defined). But for the case of

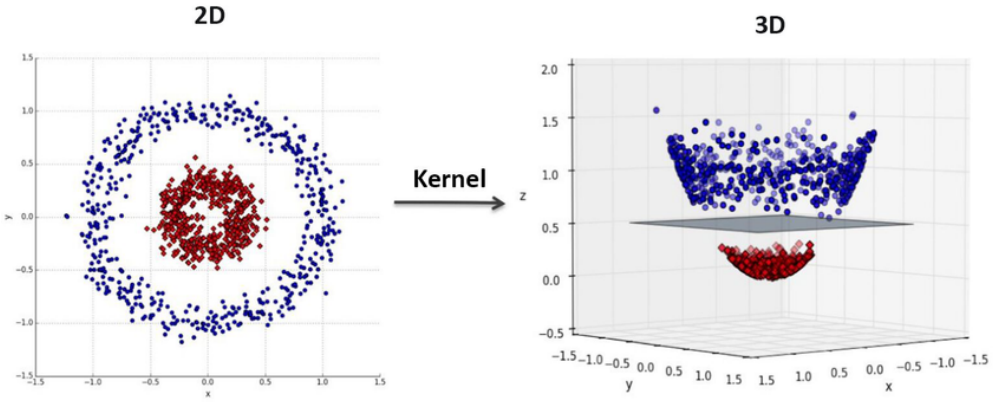


Figure 4.2.1: An example where, using a kernel function, we are able to transform the data to a higher-dimensional space, and convert the classification problem into a Linearly-Separable one[74]

Graphs, several graph metrics and other techniques are used, so that we never have to actually compute the high-dimensional function or the transformed graphs. Common techniques for this include graphlet counts, graph walks/paths, and message-propagation techniques.

Below, five common Graph Kernels are presented and explained. These are the kernels that will be used later for the experiments, and compared to the Graph Neural Network models.

- **Shortest Path Kernel:** The Shortest-Path Kernel[5] involves a process of breaking down graphs into their constituent shortest paths and subsequently comparing these paths based on their lengths and the labels of the endpoints they connect. The initial phase of the shortest-path kernel entails converting the given input graphs into graphs consisting of shortest paths. Given an input graph denoted as $G = (V, E)$, a new graph $S = (V, E_s)$ is constructed, referred to as the shortest-path graph. This newly created shortest-path graph maintains an identical set of vertices as the original graph G. However, its edge set is a larger set compared to that of G, as it contains edges connecting all vertices that are connected by a path in the original graph G. To finalize this transformation, labels are assigned to all edges within the shortest-path graph S. The label of each edge is determined by the shortest distance between its connecting endpoints in the original graph G.

  After the shortest-path graphs $S_i = (V_i, E_i), S_j = (V_j, E_j)$ have been constructed from the original G_i, G_j graphs, then the shortest-path kernel is defined as:

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

  where $k^{(1)}_{walk}(e_i, e_j)$ is a positive semidefinite kernel on edge walks of length 1. The kernel $k^{(1)}_{walk}(e_i, e_j)$ is usually defined using the dirac kernel, or more rarely the brownian bridge kernel.

  In terms of runtime complexity, the shortest-path kernel is very expensive since its computation takes $\mathcal{O}(n^4)$ time.

- **Weisfeiler-Lehman Kernel:** The Weisfeiler-Lehman Kernel[101] is based on the existing Weisfeiler-Lehman Isomorphism Test[119]. The key idea of the Weisfeiler-Lehman algorithm is to replace the label of each vertex with a multiset label consisting of the original label of the vertex and the sorted set of labels of its neighbors. The resultant multiset is then compressed into a new, short label. This relabeling procedure is then repeated for h iterations. The Weisfeiler-Lehman sequence up to height h of G consists of the Weisfeiler-Lehman graphs of G at heights from 0 to h, $\{G_0, G_1, \ldots, G_h\}$. Then the WL Kernel for graphs G and G' is defined as:

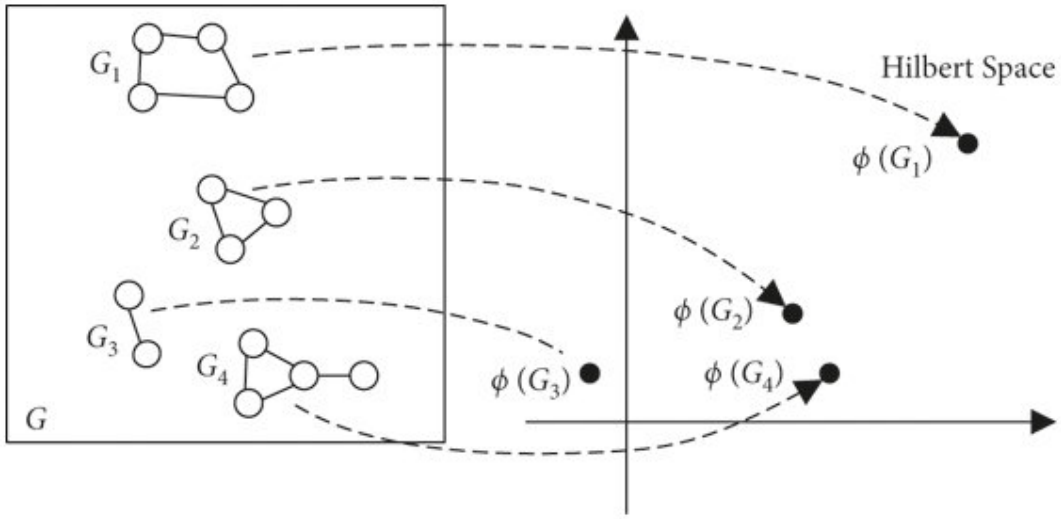


Figure 4.2.2: Graph Kernels, transform the graphs to data points, in a higher-dimensional Hilbert space. This allows us to use the traditional similarity measures, most commonly the dot-product.[64]

$$k_{WL}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \ldots + k(G_h, G'_h)$$

where k is any graph kernel, and G_i are the graphs produced by the iterative Weisfeiler-Lehman test. The kernel k is usually set to be the Vertex Histogram kernel[106], which is a linear-time kernel, and all it does is compute the histogram vector $\mathbf{f} = (f_1, f_2, \ldots, f_d)$, where $f_i = |\{v \in V : label(v) = i\}|$. The labels of the nodes, are the final compressed labels, as computed by the Weisfeiler-Lehman process.

- **Neighborhood Hash Kernel:** The Neighborhood Hash Kernel[44] works very similarly to the Weisfeiler-Lehman Kernel, in that it also has an iterative process, where the node labels are updated, according to the labels of their neighbors. The algorithm first transform each discrete node label to a bit label. A bit label is a 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 the nodes of the graphs. Then, using the binary operations XOR and left cycle-rotation ROT, the label $\ell(v)$ update process of node v is:

$$NH(v) = ROT\big(\ell(v)\big) \oplus \big(\ell(u_1) \oplus \ldots \oplus \ell(u_d)\big)$$

where $\mathcal{N}(v) = \{u_1, \ldots, u_d\}$ are the neighbor nodes of v. In order to define the final kernel formula, we will use the sub-kernel κ, defined as:

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

where c is the number of labels the two graphs have in common. So now, given the graph sequence $\{G_0, G_1, \ldots, G_h\}$ and $\{G'_0, G'_1, \ldots, G'_h\}$ by updating the node labels h times according to the Neighborhood-Hash iterative process, the final Neighborhood Hash Kernel is:

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

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

- **Random Walk Kernel:** The Random Walk Kernel, is one of the oldest and most studied family of kernels. Kernels belonging to this family have concentrated mainly on counting matching walks in the two input graphs. The most widely-used Random Walk kernel is the **geometric random walk kernel**[31] which compares walks up to infinity assigning a weight λ^k ($\lambda < 1$) to walks of length k in order to ensure convergence of the corresponding geometric series. Specifically, given two graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$, their product-graph $G_\times$ is computed as followed:

$$V_\times = \{(v_i, v_j) : v_i \in V_i \wedge v_j \in V_j \wedge \ell(v_i) = \ell(v_j)\} \tag{4.2.1}$$

$$E_\times = \{\{(v_i, v_j), (u_i, u_j)\} : \{v_i, u_i\} \in E_i \wedge \{v_j, u_j\} \in E_j\} \tag{4.2.2}$$

Performing a random walk on $G_\times$ is equivalent to performing a simultaneous random walk on G_i and G_j. Then, the geometric random walk kernel is defined as:

$$K_\times^\infty(G_i, G_j) = \sum_{p,q=1}^{|V_\times|} \Big[\sum_{l=0}^{\infty} \lambda^l A_\times^l\Big]_{pq} = e^T (I - \lambda A_\times)^{-1} e \tag{4.2.3}$$

where I is the identity matrix, e is the all-ones vector, and λ is a positive, real-valued weight. 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 kernel as defined above, requires $\mathcal{O}(n^6)$ time, but further optimizations[115] have managed to lower it to $\mathcal{O}(n^3)$.

- **Graphlet Sampling Kernel:** The main idea of this kernel, as presented in [88], is the occurrence of similar **Graphlets** in the two input graphs G_i and G_j. Specifically, Graphlets are small sub-graphs with k nodes, where k is usually a small number ($k \in \{3, 4, 5, \dots\}$).

  The process, involves counting the frequency of occurrence, of graphlets $\{graphlet_1, graphlet_2, \dots, graphlet_r\}$, and then, constructing the frequency vector for graphs G_i and G_j as:

$$f_{G_i} = \{\#(graphlet_1 \sqsubseteq G_i), \#(graphlet_2 \sqsubseteq G_i), \dots, \#(graphlet_r \sqsubseteq G_i)\}$$

$$f_{G_j} = \{\#(graphlet_1 \sqsubseteq G_j), \#(graphlet_2 \sqsubseteq G_j), \dots, \#(graphlet_r \sqsubseteq G_j)\}$$

Lastly, the final output of the kernel is:

$$k(G_i, G_j) = f_{G_i}^\top \; f_{G_j}$$

The main problem with this kernel when it was presented, was the computational cost needed to find the occurrences of all the graphlets. As we increase k to include a bigger variety of graphlets, the computational time increases exponentially. It was later proposed in [100], that we don't need to find the occurrences of all $\binom{n}{k}$ size-k graphlets, because of a sampling theorem proved by Weissman et al.[114].

This theorem affords us the ability to specify the desired level of precision in approximating the true graphlet histogram by setting an upper bound on the L1-Distance between the actual graphlet frequency distribution and the approximated counterpart. After manually setting this "level of precision", the theorem can provide us with the exact number of samples from the distribution that we need to have, in order to be close enough to the real distribution. In the case of the Graphlet Kernel, the number of samples that we need to have, are the amount of graphlets that we need to compute.

As we can see, **Graph Kernels** approach the problem of Graph Similarity from different perspectives, and they use a variety of tools. Specifically, what is important to remember here, is that the *Shortest-Path Kernel* and the *Random-Walk Kernel* are based on more conventional graph metrics, the *Graphlet-Sampling Kernel* is based on small common sub-structures found within the graphs, while the *Neighborhood-Hash Kernel* and the *Weisfeiler-Lehman Kernel* are the only ones that use the **Message-Passing** technique, where each node iteratively aggregates the information from their neighbors, and then propagates it further on. This is an important distinction, something that will become clearer later, at the Experimentation and Results.

4.3 Scene Graphs

Scene graphs are structured representations of visual scenes that can help us understand the relationships between objects. They contain information about the objects in a scene, as well as the attributes and relationships between those objects. Scene graphs have been used in a variety of computer vision tasks, including visual relationship detection, image captioning, and visual question answering. They are important because they provide a more detailed and nuanced understanding of visual scenes than traditional object detection and recognition methods, therefore allowing us to take advantage of the semantic content as well.

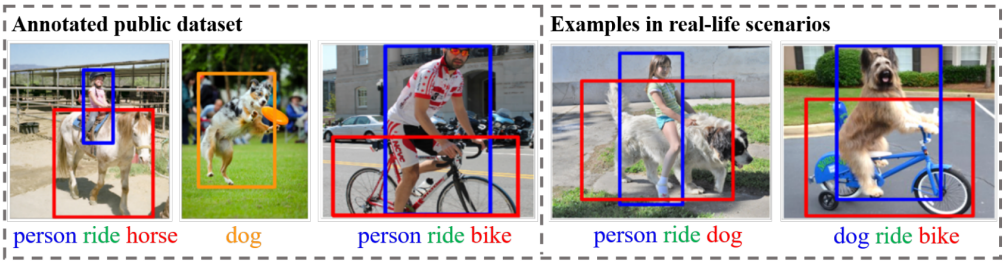


Figure 4.3.1: Examples of the sparsity and variability of visual relationships. Being able to extract useful semantic information from these relationships, can be immensely useful for all kinds of visual tasks, that require a deeper understanding of the objects and their actions.[13]

Scene graphs were first proposed in 2015[48], as a data structure that describes the object instances in a scene and the relationships between these objects. The concept of scene graphs emerged as a solution to the problem of capturing the relationships between objects in a scene, which is a higher-level visual understanding task that goes beyond simple object detection and recognition. The idea of utilizing the visual features of different objects contained in the image and the relationships between them was first introduced in 2015[48]. Since then, scene graphs have attracted the attention of a large number of researchers, and related research is often cross-modal, complex, and rapidly developing. Today, scene graphs are an important tool for understanding visual scenes and improving the performance of computer vision tasks.

Specifically in this thesis, the dataset that will be used is **Visual Genome**, one of the most popular Scene Graph datasets, that has been used in a variety of tasks. Presented in (2016)[58], Visual Genome is a large-scale dataset that aims to provide a comprehensive understanding of the visual world by connecting language and vision through crowdsourced dense image annotations. The dataset contains over 100,000 images, each with detailed annotations of objects, attributes, relationships, and region descriptions. What distinguishes Visual Genome from other datasets is its prioritization of relationships and attributes as primary components within the annotation space, in conjunction with the customary emphasis on objects. Recognition of relationships and attributes is an important part of the complete understanding of the visual scene, and

in many cases, these elements are key to the story of a scene.

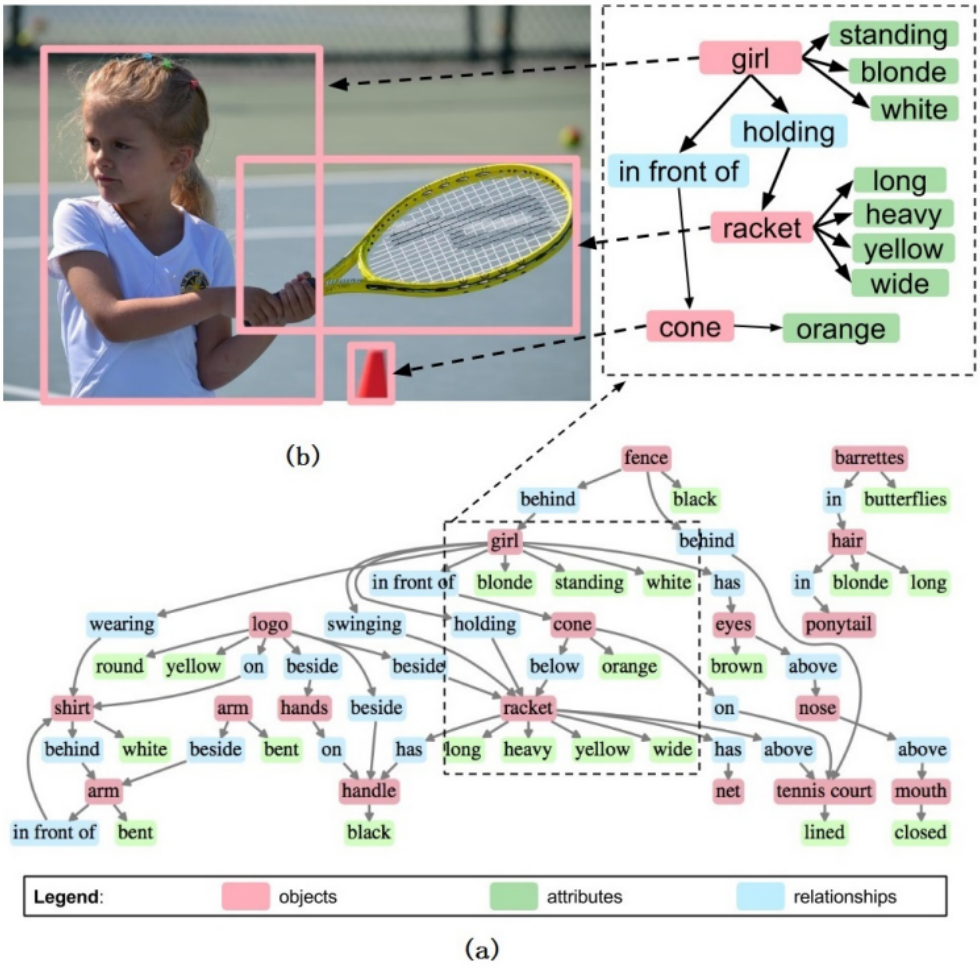


Figure 4.3.2: An example of a Scene Graph, alongside its original image. This specific scene graph, contains objects, relationships and attributes, but it should be noted that there is no strict definition of a schema that all the Scene Graphs are required to follow. Only that they are in the form of graph, and that they provide semantic information of the objects in the image, and the relationships between them.[122]

The dataset is useful for a wide range of computer vision and natural language processing tasks, including **object recognition, scene understanding, image captioning, and visual question answering**. The annotations in Visual Genome provide a rich source of information for training and evaluating models that aim to understand the visual world. For example, the region descriptions can be used to train models for image captioning, while the relationships and attributes can be used to train models for scene understanding and visual question answering.

Visual Genome is unique in its approach to annotation, which involves **crowdsourcing dense annotations** from multiple annotators for each image. This approach allows for a more comprehensive understanding of the visual world, as it captures the diversity of interpretations and perspectives that humans bring to the task of image annotation. The dataset also includes a canonicalization pipeline that resolves inconsistencies and errors in the annotations, ensuring that the dataset is of high quality and suitable for use in a wide range of applications.

Overall, Visual Genome is an important and unique dataset that provides a **comprehensive understanding of the visual world** by connecting language and vision through crowdsourced dense image annotations. The dataset is useful for a wide range of computer vision and natural language processing tasks, and its approach to annotation ensures that it captures the diversity of interpretations and perspectives that humans bring to

the task of image annotation. In this thesis, the scene graphs alone are going to be used for the Quantitative Evaluation of the models, and the corresponding images of the scene graphs are going to be used for the Qualitative Evaluation.

4.4 Related Work

Graph Kernels

In addition to the five families of kernels mentioned, there are several other popular and widely used families of kernels in the field of graph kernels. These include the Subgraph Matching Kernel, the Edge Histogram Kernel, the Pyramid Match Kernel, and the Treelet Kernel.

The **Subgraph Matching Kernel** (SMK)[57] is a family of kernels that compares the presence and frequency of subgraphs in two graphs. The SMK is useful in applications where the presence of specific subgraphs is important, such as in chemical compound classification and protein structure analysis.

The **Edge Histogram Kernel** (EHK) is a family of kernels that compares the distribution of edge labels in two graphs. The EHK is useful in applications where the labels of edges are important, such as in social network analysis and image classification.

The **Pyramid Match Kernel** (PMK)[36], compares the hierarchical structure of two graphs by computing the similarity between their corresponding pyramids. The PMK is useful in applications where the hierarchical structure of the graphs is important, such as in image classification and object recognition. The PMK has been shown to outperform other graph kernels in certain applications, and its ability to capture the hierarchical structure of graphs makes it a valuable tool for machine learning on graph data.

The **Treelet Kernel** (TK)[33] is a family of kernels that compares the frequency of small tree-like structures, called treelets, in two graphs. The TK is useful in applications where the presence of specific tree-like structures is important, such as in protein structure analysis and social network analysis.

Each of these families of kernels has its own strengths and weaknesses, and is suited to different types of applications. By using a combination of these kernels, researchers can develop powerful machine learning models for a wide range of graph-based problems.

Regarding the range of application, Graph Kernels has been successfully used in numerous fields. Traditionally, **chemistry** is one of the richest sources of graph-structured data. Graph kernels have been used extensively for predicting the mutagenicity, toxicity and anti-cancer activity of small molecules [107, 91, 72, 12, 73, 104]. **Bioinformatics** is also one of the major application domains of graph representations and therefore, of graph kernels. These include, their utilization in predicting the functional attributes of proteins characterized by both known sequences and structures [4, 96], the identification of intricate interactions implicated in the onset and progression of diseases [6], the examination of functional non-coding RNA sequences [95], and many more. Graph kernels have also served as an effective tool for many **computer vision tasks** such as for classifying images [41, 71], for detecting objects represented as point clouds [1, 82], for achieving place recognition [105]. Further work can be found in the comprehensive survey by Nikolentzos et al.[83].

Scene Graphs

Besides the task of Graph Similarity, scene graphs have been used in numerous other cases. First and foremost, scene graphs have been used to **generate images or layouts**, based on the objects and relationships that they encode [47, 78, 111, 128, 43, 110]. **Cross-modal retrieval** is also a common application in scene graph research. Image-text retrieval is a classic multi-modal retrieval task. The key to image-text cross-modal retrieval concerns learning a comprehensive and unified representation to represent multi-modal data. The scene graph is a good choice in this context, and a substantial body of research has been dedicated to this

domain[98, 89, 97, 14]. Besides retrieval, another popular multimodal task is **Visual Question Answering**. Scene Graphs can capture the essential information of images in the form of graph structures, which helps scene-graph-based VQA methods outperform traditional algorithms[126, 123, 34]. Further work can be found in the comprehensive survey by Chang et al.[13].

Chapter 5

Graph Neural Networks (GNN)

Graph Neural Networks (GNNs) have emerged as a powerful class of machine learning models designed to tackle data represented in graph structures. They have gained immense popularity due to their ability to address a wide range of real-world problems, from social network analysis to drug discovery and recommendation systems. The need for GNNs arises from the limitations of traditional deep learning methods when applied to graph data.

One key reason for the rise of GNNs is that many real-world datasets can be naturally represented as graphs, where entities (nodes) are connected by relationships (edges). Traditional deep learning methods, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are primarily designed for grid-like data, like images and sequences. They lack the capacity to capture the inherent structural information and dependencies present in graph data. GNNs, on the other hand, are specifically tailored to process graph-structured data, making them well-suited for tasks like node classification, link prediction, and graph classification.

We will further explain the possible architectures of GNNs, and why the have come to be indispensable for the machine learning community.

Contents

5.1 Introduction **70**
5.2 Graph Convolution **71**
5.2.1 Spectral-Based Graph Convolution 71
5.2.2 Spatial-based Graph Convolution 72
5.3 Spatial-Based GNN Modules **74**
5.4 Graph Auto-encoders **76**

5.1 Introduction

Graph data is becoming increasingly common in various domains, such as social networks, bioinformatics, and recommendation systems. However, traditional machine learning algorithms are designed to handle data in the Euclidean space, which is not suitable for graph data, which has complex relationships and inter-dependencies between objects, something that cannot be captured by traditional algorithms. Therefore, there is a need for specialized architectures that can process graph data effectively.

Graphs and Euclidean Space

Graphs cannot always be embedded into a Euclidean space without **distortion** or **overlap**. The primary reason for this inability to embed all graphs into Euclidean space is that some graphs possess properties or structures that make it impossible to represent them without certain distortions or overlaps.

In broad terms, non-Euclidean data is data whose underlying domain does not obey Euclidean distance as a metric between points in the domain. For visualization simplicity, we can think of $\mathbb{Z}^2$ instead, which can be seen as a "grid" of integer-valued points separated by a distance of 1. We can easily "visualize" the distance between the points of the domain.

Now let's view the case of graphs, and specifically, the underlying domain of graphs. It's a set of nodes V and a set of edges E. In the context of graph theory, it is essential to recognize that the concept of distance, as understood within the context of Euclidean geometry, does not directly apply to the nodes of a graph. When posed with the query, "What is the distance between point A and point B within this graph?" the response typically hinges upon a function that relies on the structural connectivity inherent to the graph, an integral component of its domain. Consequently, it is crucial to acknowledge that this **distance measure may vary significantly for distinct pairs of nodes** located within the graph.

Therefore, since graph metrics can't be made to obey the fundamentals of a Euclidean Distance, **graphs don't lie in a Euclidean Space**.

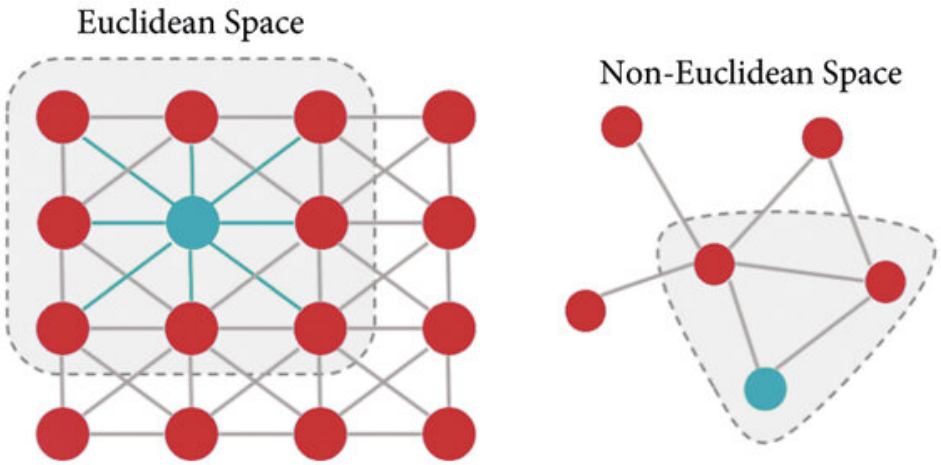


Figure 5.1.1: The structured nature of the data points on the left, allows us to treat it as data in a Euclidean Space. On the contrary, in the case of the random graph on the right, we can't define a distance between the nodes, that allows us to obey the Euclidean Distance rules.[2]

Practical challenges with Graph Data

We will now present the main problems that we encounter when processing graph data, and briefly mention the techniques that GNNs employ, in order to solve them.

One of the main challenges in processing graph data is the **lack of a fixed structure**. Unlike images or text, graphs can have varying sizes and structures. This makes it difficult to apply traditional machine learning algorithms, which require fixed-size inputs. GNNs can handle graphs of varying sizes and structures by using message passing algorithms. In message passing, each node aggregates information from its neighbors and updates its representation. This allows GNNs to handle graphs, without making any assumptions about the number of nodes, or the relationships between them.

Another challenge in processing graph data is the **lack of a natural ordering of nodes**. In Euclidean space, the ordering of dimensions is fixed, which allows traditional algorithms to process data efficiently. However, in a graph, there is no natural ordering of nodes. GNNs can handle this challenge by using graph convolutions, which are designed to be permutation invariant. This means that the order of nodes does not affect the output of the convolution, allowing GNNs to process graph data efficiently.

In summary, GNNs are needed because traditional machine learning algorithms are unable to handle graph data effectively. GNNs can handle the complex relationships and dependencies between objects in a graph by using message passing algorithms and graph convolutions. In the next section, we will explain in detail how these techniques are applied on graphs, along with the required theoretical background.

5.2 Graph Convolution

The main operation that GNNs employ in order to process and analyze graph data, is **Graph Convolution**. As the name suggests, it is the equivalent operation of Signal Convolutional (as the one applied in CNNs for images), but with the necessary differentiations, to be applicable on graph data.

Graph Convolution, and GNNs in general, can be viewed as a function that is applied on a graph signal. For the explanation of GNNs, we can simply view a graph signal, as feature vectors for the nodes of the graph. When we apply Graph Convolution on the graph signal, the result will be an updated signal.

Like all signal functions, Graph Convolution can be studied on the spatial domain, and on the spectral domain. Below, we will present the theoretical background of each domain, along with a brief history of work on this field.

5.2.1 Spectral-Based Graph Convolution

Spectral-based methods have a solid mathematical foundation in graph signal processing [102, 16, 93]. They assume graphs to be undirected. At the heart of graph analysis in the spectral domain, is the **Graph Laplacian**. The graph Laplacian matrix is a mathematical representation of an undirected graph, defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

where D is the degree matrix, and A is the adjacency matrix of the graph. A vertex with a large degree, also called a *heavy node*, results in a large diagonal entry in the Laplacian matrix dominating the matrix properties. Normalization is aimed to make the influence of such vertices more equal to that of other vertices, by dividing the entries of the Laplacian matrix by the vertex degrees. That's why the **Normalized Graph Laplacian** is more commonly used, which is defined as:

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

The normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite, which means that we can factor $\mathbf{L}$ into:

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

where $\mathbf{U} = [\mathbf{u_0}, \mathbf{u_1}, \ldots, \mathbf{u_{n-1}}] \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors ordered by eigenvalues and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues (spectrum).

In graph signal processing, a graph signal $\mathbf{x} \in \mathbb{R}^n$ is a feature vector of all nodes of a graph where x_i is the value of the i^{th} node. The graph Fourier transform to a signal x is defined as $\mathcal{F}(x) = \mathbf{U}^T x$, and the inverse **graph Fourier transform** is defined as $F^{-1}(\hat{x}) = \mathbf{U}\hat{x}$, where $\hat{x}$ represents the resulted signal from the graph Fourier transform. So now, let's also assume we have a filter $\mathbf{g} \in \mathbb{R}^n$, the convolution of graph signal $\mathbf{x}$ (node feature vectors) with the filter $\mathbf{g}$ is defined as:

$$\mathbf{x} *_G \mathbf{g} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g}))$$

Further simplifying the equation, by denoting $\mathbf{g}_\theta = diag\left(\mathbf{U}^T \mathbf{g}\right)$, we get:

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta \mathbf{U}^T \mathbf{x}$$

From this point on, the only differentiation among the different Spectral Graph Convolution variations, is the parameter $\mathbf{g}_\theta$, and what we set it to be. Specifically, *Spectral-CNN*[10] set $\mathbf{g}_\theta$ equal to a set of learnable parameters $\Theta_{i,j}^{(k)}$. This generic approach, has three main disadvantages. First, any perturbation to a graph results in a change of eigenbasis. Second, the learned filters are domain dependent, meaning they cannot be applied to a graph with a different structure. Third, eigen-decomposition requires $\mathcal{O}(n^3)$ computational complexity. In order to tackle these problems, the main idea was to **approximate $\mathbf{g}_\theta$ by using polynomials**[20, 63]. Specifically, the filter is:

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \cdots + w_d L^d = \sum_{i=0}^{d} w_i L^i$$

These polynomials can be thought of as the equivalent of 'filters' in CNNs, and the coefficients w as the weights of the 'filters'.

Although Spectral-based Graph Convolutional Networks exhibit a strong mathematical foundation, modern research has shifted in the later years to Spatial-based Graph Convolutional Networks, mostly because of the simplicity and ease of further research.

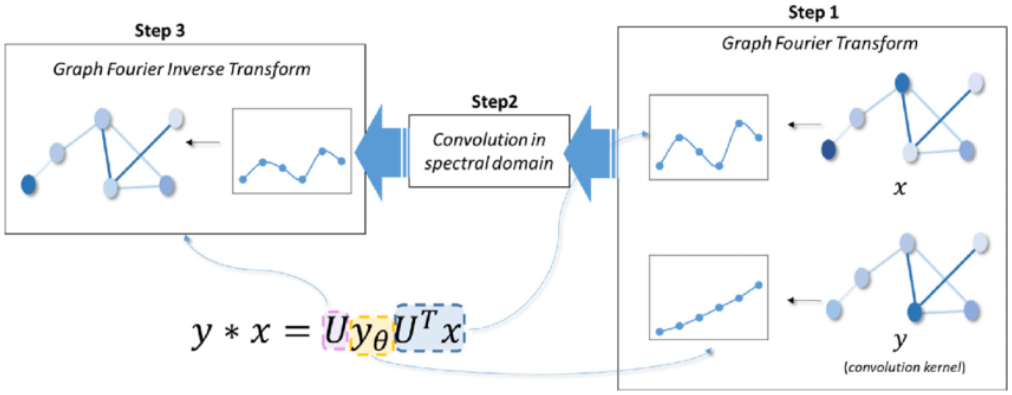


Figure 5.2.1: Visualization of Spectral Graph Convolution[65]

5.2.2 Spatial-based Graph Convolution

Similar to how a traditional Convolutional Neural Network (CNN) performs convolution operations on an image, spatial-based techniques define graph convolutions by considering the spatial relationships between nodes. In this context, images can be viewed as a special type of graph, with individual pixels serving as

nodes. Each pixel maintains direct connections to its adjacent pixels, as depicted in Figure 5.2.1. To execute a convolution, a filter is applied to a 3×3 patch, computing the weighted average of pixel values within the central node and its neighboring nodes across all channels. Analogously, spatial-based graph convolutions operate by convolving the representation of the central node with the representations of its neighboring nodes, resulting in an updated representation for the central node, as depicted in Figure 5.2.1. This propagation of node information along the graph's edges, is more commonly known as **Message Passing**.

A general framework for Spatial-based Convolutional GNNs, was introduced with the **Message Passing Neural Network** (MPNN)[35]. Within this framework, graph convolutions are viewed as a message-passing procedure, allowing the exchange of information directly between nodes along their connecting edges. MPNN executes a series of K-step message-passing iterations, enabling the gradual propagation of information across the graph. The message passing function is defined as:

$$\mathbf{h}_v^{(k)} = U_k \left(\mathbf{h}_v^{(k-1)}, \sum_{u \in N(v)} M_k(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_{vu}^e) \right)$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $U_k(\cdot)$ and $M_k(\cdot)$ are functions with learnable parameters. Once the hidden representations of individual nodes, denoted as $\mathbf{h}_v^{(K)}$, have been computed, they can be forwarded either to **an output layer** for tasks that involve predicting at the node level, or **to a readout function** for tasks that involve predicting at the graph level. The readout function is responsible for creating a representation of the entire graph by leveraging the hidden representations of its constituent nodes. Most existing Spatial-based ConvGNNs can be seen as a variation of MPNN, by setting the corresponding U_k, M_k and the readout function.

Spectral models find their theoretical basis in graph signal processing. By crafting novel graph signal filters, one can construct new Convolutional Graph Neural Networks (GNNs). However, spatial models are preferred over spectral models for the following reasons:

1. Firstly, **spectral models exhibit lower efficiency** compared to spatial models. Spectral models necessitate either eigenvector calculations or the processing of the entire graph simultaneously. In contrast, spatial models demonstrate superior scalability on large graphs by directly conducting convolutions within the graph domain through information propagation. This computation can be executed in batches of nodes rather than the entire graph.

2. Secondly, **spectral models, which rely on a graph Fourier basis, struggle to generalize** effectively to new graphs. They assume a fixed graph structure, and any alterations to the graph would result in a modification of the eigenbasis. In contrast, spatial-based models perform graph convolutions locally on each node, allowing for easy weight sharing across various locations and graph structures.

3. Thirdly, **spectral-based models are restricted to working with undirected graphs**. Spatial-based models exhibit greater flexibility in handling inputs from diverse sources, such as edge inputs, directed graphs, signed graphs, and heterogeneous graphs. This flexibility arises from their ability to seamlessly incorporate these types of graph inputs into the aggregation function.

It is common for all the Convolutional GNNs, to be stacked on multiple layers. The advantage of layer-stacking, comes from **Message Passing**, which propagates the information further than the 1-hop neighbors. Specifically, if we stack k-layers of Convolutional GNNs, the information of each node will have propagated up to the k-hop neighbors.

However, Li et al.[66] have demonstrated a significant decrease in the effectiveness of a ConvGNN as the number of graph convolutional layers increases. Since graph convolutions tend to bring the representations of neighboring nodes closer together, it can be theoretically posited that an infinite number of graph convolutional layers would ultimately cause all nodes' representations to converge to a singular point. This prompts an inquiry into whether pursuing greater depth remains a viable approach for the acquisition of knowledge from graph data.

The three factors presented above, are the main causes for the domination of Spatial-based Convolutional GNNs in research for the past few years. In the next section, we will present in detail, the most common modern Spatial-based ConvGNNs, that will also be implemented and assessed as part of this thesis.

5.3 Spatial-Based GNN Modules

Most Spatial-based Convolutional GNNs, are formulated as a variant of the MPNN[35] module. Specifically, the most common ones, that will be implemented at the Proposed Models, are the following:

- The **Graph Convolutional Network** (GCN) was proposed by Kipf et al.(2016)[55], and it is now considered the baseline for most Spatial-based GNN modules. The GCN is based on the spectral graph convolutional neural network (SGCNN) introduced by Bruna et al. (2014)[10]. It is the first GNN module, that connected Spatial-based and Spectral-based approaches. The original equation for applying a filter $\mathbf{g}_\theta$, on a graph signal $\mathbf{x}$, is defined as:

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}$$

But, as we mentioned earlier, we can approximate $\mathbf{g}_\theta(\Lambda)$ by using Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$$\mathbf{g}_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda})$$

where $\tilde{\Lambda}$ is the rescaled $\tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$. So now, the graph convolution with the signal, becomes:

$$\mathbf{x} *_G \mathbf{g}_\theta \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\mathbf{x}$$

where $\tilde{L} = \frac{2}{\lambda_{max}}L - I_N$. By using K^{th} polynomials of the Laplacian, we are essentially constricting the dependency, to nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood). Here, the authors of the paper, propose two simplifications to the above equation. They set $K = 1$, and they approximate $\lambda_{max} \approx 2$. After these simplifications, we get the linear equation, defined as:

$$\mathbf{x} *_G \mathbf{g}_\theta \approx \theta'_0 x + \theta'_1(L - I_N)x \approx \theta'_0 x + \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

In order to further minimize the number complexity, and the number of matrix multiplications, they set the parameters to be equal ($\theta = \theta'_0 = -\theta'_1$), and the expression becomes:

$$\mathbf{x} *_G \mathbf{g}_\theta \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x$$

Here, it was noticed that the eigenvalues of $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ fall in the range $[0, 2]$, so repeated applications of this operation can lead to instabilities, such as exploding/vanishing gradients. To solve that, they introduced a *renormalization trick*, by using $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ instead of $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where $\tilde{A} = A + I_N$ and $\tilde{D} = \sum_j \tilde{A}_{ij}$

So the final equation of GCN in matrix form is:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

where $X \in \mathbb{R}^{N \times C}$, $\Theta \in \mathbb{R}^{C \times F}$, $Z \in \mathbb{R}^{N \times F}$, and C is the dimension of the input node vectors, and F is the dimension of the output node vectors. Since the above equation is linear, in order to solve non-linear problems as well, an activation function (e.g. ReLU) is applied to the output vectors of the GCN.

Now, we have seen that the GCN is a simplified Spectral-based GNN, based on the SGCNN. But it can also be written as a node-wise equation:

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

So we can see, that GCN is the bridge between Spectral-Based and Spatial-based approaches. Although, since the Spatial-based formulation is much easier to understand and further extend, related work that built on top of GCN, did so exclusively on the Spatial domain.

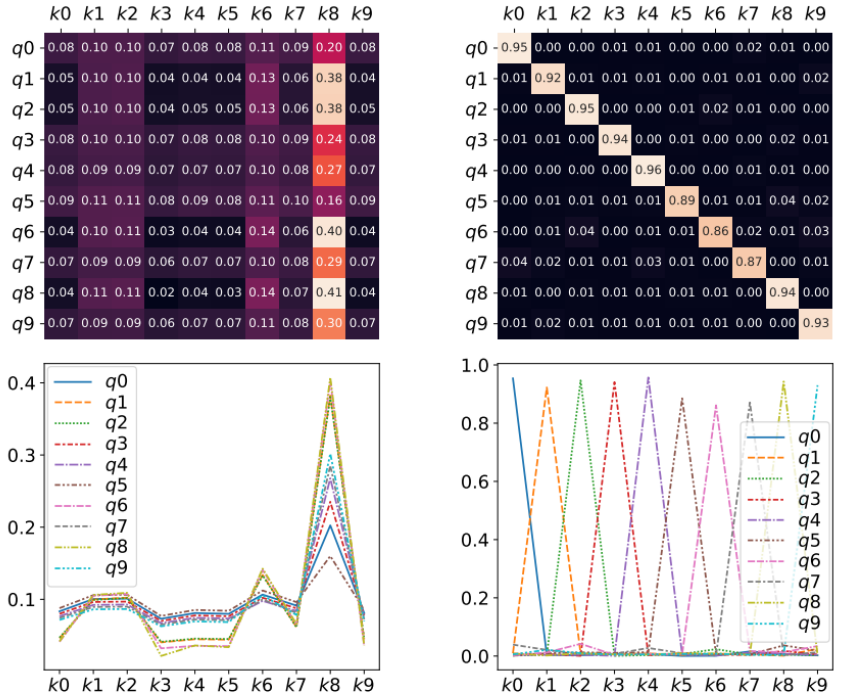
- The **Graph Attention** network (GAT), was proposed by Veličković et al.(2017) [113], and the main idea was to use attention, and especially multi-headed attention[112]. The attention technique, was used as a "weight-function", to compute the impact of the neighbor nodes, when updating the central node. Specifically, the node-wise equation is defined as:

$$\mathbf{x}'_i = a_{i,i} \mathbf{\Theta} \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} a_{i,j} \mathbf{\Theta} \mathbf{x}_j$$

where $a_{i,j}$ are the attention weights, defined as:

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k]\right)\right)}$$

where $\mathbf{a}$ and $\mathbf{\Theta}$ are learnable parameters. The negative slope of LeakyReLU is set to $a = 0.2$ and in the case of Multi-Headed Attention, they implement it the same way as in [112]



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

Figure 5.3.1: On the left, we can see that the attention maps of *GAT* compute a global ranking of all attention weights. On the right, *GATv2* computes the correct attention weights, visualized by the characteristic diagonal in the attention matrix (because every node should attend the most with itself) [8]

- **GATv2**[8] is an upgrade to the original *GAT* that was previously presented. The authors claimed that *GAT* computes a *static attention* among the nodes, and not *dynamic attention*, 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. This is intuitively a wrong implementation of the Attention Mechanism, because each node should attend the most to the themselves (highest similarity), and they shouldn't have the same ranking across all nodes' attention weights.

  This can be proven by the equation of attention weights used in the *GAT*, if we set the learned parameter a to be equal to a concatenation $a = [a_1 \| a_2]$. Then the weights $a_{i,j}$ are computed as:

$$a_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}_1^T \mathbf{\Theta} \mathbf{x}_i \; + \; \mathbf{a}_2^T \mathbf{\Theta} \mathbf{x}_j\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}_1^T \mathbf{\Theta} \mathbf{x}_i \; + \; \mathbf{a}_2^T \mathbf{\Theta} \mathbf{x}_k\right)\right)}$$

  For a specific node i, the denominator is constant (not dependent to j), and for the numerator there is a node j so that the expression $\exp\left(\text{LeakyReLU}\left(\mathbf{a}_1^T \mathbf{\Theta} \mathbf{x}_i \; + \; \mathbf{a}_2^T \mathbf{\Theta} \mathbf{x}_j\right)\right)$ is maximized (since the term $\mathbf{a}_1^T \mathbf{\Theta} \mathbf{x}_i$ is constant, for a specific i). So it is easy to see, that there is a j, that will maximize $a_{i,j}$ for every i, This is exactly the reason that *GAT* computes the static version of attention.

  We can further understand this behavior, if we visualize the attention maps of *GATv2* and *GAT*, as shown in Figure 5.3.1.

  The proposed function that computes attention weights $a_{i,j}$ in GATv2 (and leads to *dynamic attention*) is:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{\Theta}[\mathbf{x}_i \,\|\, \mathbf{x}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{\Theta}[\mathbf{x}_i \,\|\, \mathbf{x}_k]\right)\right)}$$

- **Graph Isomorphism Network** (GIN), was proposed by Xu et al.[121], where their investigation revealed that prior *MPNN*-based approaches lacked the ability to differentiate between various graph structures based on the graph embeddings they generated. To address this limitation, GIN introduces a modification wherein it adapts the central node's weight using a trainable parameter ϵ. The node-wise formula is:

$$\mathbf{x}_i' = h_{\mathbf{\Theta}}\left((1+\epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j\right)$$

  where $h_{\mathbf{\Theta}}$ denotes any implementation of an MLP. They also argue that, setting the operation of **Sum** to be the readout function of the GNN (as defined in Section 5.2.2), *GIN* generalizes the Weisfeiler-Lehman test and hence achieves maximum discriminative power among GNNs.

5.4 Graph Auto-encoders

Graph Autoencoders (*GAEs*) are a class of Graph Neural Networks that are used for unsupervised learning on graph data. The main objective of a *GAE* is to learn a compressed representation of the input graph, which can be used for various tasks, such as network embedding and graph generation. *GAEs* consist of two main components: an encoder and a decoder. The encoder maps the input graph to a compressed representation, while the decoder maps the compressed representation back to the original graph.

GAEs are important because they can learn a compressed representation of the input graph without the need for labeled data. This makes them suitable for various applications where labeled data is scarce or expensive to obtain (such as Graph Edit Distance). *GAEs* can also be used for network embedding, where the compressed representation is employed for downstream tasks, such as node classification and link prediction. In addition, *GAEs* can be used for graph generation, where the compressed representation can guide the generation of new graphs that are similar to the original graph.

Specifically, there are three main architectures that will be presented here. These are the models that will be combined and used for the experiments later on.

1. The **Variational Graph Autoencoder** (VGAE)[54], is a framework for unsupervised learning on graph-structured data. The model uses latent variables to learn compressed representations for (un)directed graphs.

   Both *GAE* and *VGAE* models, follow the basic architecture of the generic Auto-encoder, as presented in Section 3.5. Therefore, the main components of the architecture is an *Encoder* that maps the input to a latent space, and a *Decoder*, whose task is to reconstruct the input, given the latent space representation. Specifically, in the case of graph autoencoders, the *Encoder*, can be any Graph Neural Network that produces embeddings for the nodes, and the proposed *Decoder* is an **Inner-Product Decoder**. The way the *Inner-Product Decoder* works is that, given the node embeddings $\mathbf{Z} \in \mathbb{R}^{N \times F}$ in the latent space (where F is the dimension of the latent embeddings), it will predict the adjacency matrix $\mathbf{A}$ by computing:

   $$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^T\right)$$

   So we can see, that "indirectly", the task of the decoder is Link Prediction, on the edges of the input graph. Then, given the original adjacency matrix $\mathbf{A}$, the loss of the simple GAE is *Binary Cross-Entropy Loss* for the Positive and the Negative Samples(Edges). When the graph is too sparse, it is common to sub-sample the Negative Edges, to have an equal amount of samples for both classes.

   The difference of the **Variational GAE** is the extra term at the Loss function, to impose a prior distribution on the latent embeddings (as described in Section 3.5). This prior distribution is almost always set to be the Standard Normal distribution $p(\mathbf{Z}) = \prod_i p(\mathbf{z}_i) = \prod_i \mathcal{N}(\mathbf{z}_i|0, \mathbf{I})$. But, in order to compute a Mean and a Standard Deviation, the *VGAE* model needs to have two outputs in the latent space, not one (as in the case of *GAE*). Usually, these two GNNs that are used in a *VGAE* are mentioned as GNN_μ and GNN_σ, the networkσ that produce the Mean and the Standard Deviation respectively. Then, using the *reparameterization trick* [53], the final latent embedding is:

   $$\mathbf{Z} = GNN_\mu(\mathbf{A}, \mathbf{X}) + \epsilon * GNN_\sigma(\mathbf{A}, \mathbf{X})$$

   where ϵ is an arbitrary random variable, usually $\epsilon \sim \mathcal{N}(0, 1)$. And the second difference of *VGAEs*, is the loss function. As explained in Section 3.5, the *Variational GAE*, adds a second term to the loss function (besides the reconstruction loss), which is the *Kullback-Leibler Divergence* between the prior distribution (Standard Normal) and the approximated distribution (GNN_μ and GNN_σ from the encoder). Generally, **VGAEs** are preffered, when we want the final embeddings to follow a distribution, somethinhg that is of great importance in teh *Graph Similarity* task, where the encoded embeddings need to be regularized.

2. The **Adversarially Regularized Variational Graph Autoencoder** (ARVGA) proposed by Pan et al.[84], further builds upon the original *VGAE*, by implementing ideas of **Adversarial Training**. Specifically, a Discriminator Module is employed (implemented as an MLP), which receives as input, the sample embeddings from the latent space, and samples from the prior distribution. The goal is to successfully classify them as True or Fake samples.

   This addition to the original *VGAE* architecture proved to be quite important, because it helped immensely to regularize the embedding and improve its quality. As a result, the final embedding space, is much more interpretable and regularized, compared to the original architecture.

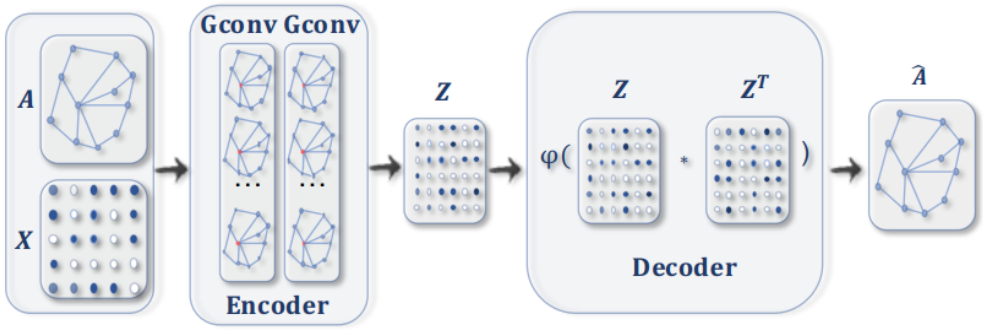


Figure 5.4.1: The base architecture of a Graph Autoencoder, consisting of the Encoder and Decoder [120]

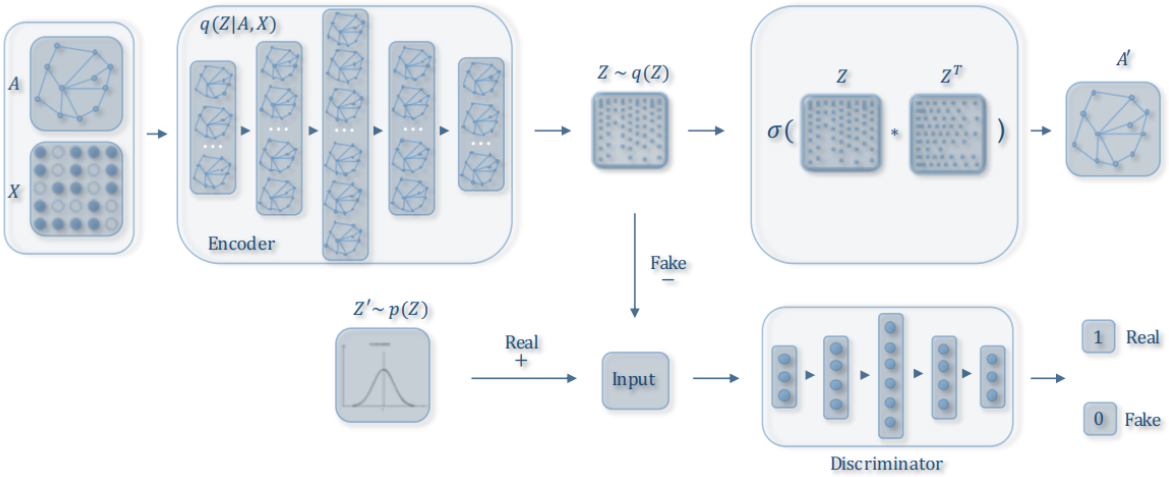


Figure 5.4.2: Architecture of the ARVGA model, as presented in [84]

3. The **Graph Feature Autoencoder** [42], was the first model in the **GAE** family, to employ a *Feature-Decoder*, not an Edge-Decoder. Specifically, they used the node embeddings produced by the Encoder, and propagated them through a MLP network. The goal of this MLP network, is a regression task, to produce the original feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ from the latent embedding matrix $\mathbf{Z} \in \mathbb{R}^{N \times D}$. The final loss term, added by this Feature Decoder, is the *Mean Squared Error* Loss between $\mathbf{X}$ and the output $\hat{\mathbf{X}}$ of the Feature Decoder.

   The main advantage of adding this Feature Decoder to the architecture, is that now, the embeddings will need to **compress feature information** as well. In the original architecture, the Decoder only predicted the Adjacency Matrix, so there was no guarantee that the final embeddings had encoded any information regarding their features. The paper argues that the addition of a Feature Decoder is especially useful in biological interaction networks, such as genes or proteins.

The concepts introduced in the three aforementioned approaches will serve as the foundational elements for devising novel **Graph Autoencoder** structures. These structures will subsequently undergo experimental evaluation, through the task of Graph Similarity.

Chapter 6

Counterfactual Explanations

Explainable Artificial Intelligence (XAI) is a critical subfield within the realm of artificial intelligence (AI) that aims to enhance the transparency and interpretability of AI systems. It addresses the fundamental challenge of making AI algorithms and models more understandable and comprehensible to both experts and non-experts. XAI seeks to bridge the gap between the black-box nature of many AI algorithms and the need for human users to trust, verify, and explain the decisions made by these systems.

The need for XAI is driven by several factors. Firstly, as AI is increasingly integrated into various aspects of our lives, including healthcare, finance, and autonomous vehicles, it becomes imperative to ensure that **AI-driven decisions can be justified and explained**. This is especially crucial in high-stakes applications where incorrect or biased decisions can have significant consequences. Additionally, regulations and ethical considerations, such as the General Data Protection Regulation (GDPR) in Europe, require organizations to provide explanations for automated decisions affecting individuals. Moreover, XAI can aid in the debugging and improvement of AI models, making them more robust and reliable.

XAI has a wide range of applications across industries. In **healthcare**, it can help doctors and medical practitioners understand the rationale behind AI-generated diagnoses, leading to more informed treatment decisions. In **finance**, XAI can be used to interpret the factors influencing algorithmic trading decisions, enhancing market transparency. Furthermore, in **autonomous vehicles**, XAI can provide insights into why a self-driving car made a specific driving maneuver, contributing to safer and more trustworthy autonomous systems. Overall, Explainable AI is a critical development in the field of AI, fostering trust, accountability, and responsible AI deployment across various domains.

In this thesis, we will focus on a specific XAI technique, the **Counterfactual Explanations**. In this chapter, the necessary definitions, formulas and algorithms will be explained in detail, primarily in accordance with the contributions [29] and [21].

Contents

6.1 Introduction **80**
6.2 Definitions and Framework **80**
6.2.1 Algorithmic Implementation 82
6.2.2 Enriching the Explanation Dataset 83
6.2.3 Utilization of GNN Models 83
6.3 Related Work **84**

6.1 Introduction

Counterfactual explanations are a type of explanation that aims to reveal what should have been different in an instance to observe a different outcome. In the context of machine learning, counterfactual explanations are used to explain the decisions made by uninterpretable classifiers. They are particularly valuable because they provide actionable insights into how to change the input to achieve a desired output. Specifically, they can provide an answer to the question *"What would have to change for something to be classified as X instead of Y?"*. For example, a bank customer who is denied a loan may receive a counterfactual explanation that reveals what they could have done differently to be approved.

Counterfactual explanations have several advantages over other explainability methods in machine learning. Here are some of the key advantages:

1. **Actionability:** Counterfactual explanations provide actionable insights into how to change the input to achieve a desired output. This is particularly valuable in high-stakes applications, such as healthcare and finance, where the decisions made by the model can have a significant impact on people's lives. By providing actionable insights, counterfactual explanations can help to ensure that the decisions made by the model are fair and unbiased.

2. **Individualized explanations:** Counterfactual explanations provide individualized explanations for each input. This is in contrast to other explainability methods, such as feature importance or partial dependence plots, which provide global explanations that apply to all inputs. Individualized explanations are important because they provide a way to understand why a particular decision was made for a specific input.

3. **Model-agnostic:** Counterfactual explanations can be generated for a wide range of machine learning models, including black-box models. This is in contrast to other explainability methods, such as decision trees or linear models, which are specific to a particular type of model. Model-agnostic explanations are important because they provide a way to understand the decisions made by any model, regardless of its complexity.

4. **Causal inference:** Counterfactual explanations are based on the concept of counterfactuals, which are statements about what would have happened if something had been different. This makes them well-suited for causal inference, where the goal is to estimate the effect of an intervention. By generating counterfactual explanations, it is possible to estimate the effect of changing the input on the output.

5. **Fairness and robustness:** Counterfactual explanations can be used to evaluate the fairness and robustness of machine learning models. By generating counterfactual explanations for a large number of inputs, it is possible to identify patterns in the model's behavior and detect biases or vulnerabilities. This can help to improve the model's performance and ensure that it is fair and robust.

Overall, counterfactual explanations have several advantages over other explainability methods in machine learning. They provide actionable insights, individualized explanations, are model-agnostic, well-suited for causal inference, and can be used to evaluate the fairness and robustness of machine learning models. The specific Counterfactual Explainer framework that we present here is proposed in [29], and a more refined version of it in [21].

6.2 Definitions and Framework

The goal of the framework presented in [29], is to provide Counterfactual Explanations for a black-box classification model, by specifying the optimal Counterfactual for a specific input sample. The Counterfactual is an object, that is classified to a different class than the input sample (according to the black-box classifier), and the modifications from the input sample are interpretable and understandable by humans.

The framework by Filalndrianos et al. [29] is based on **Knowledge Bases**, and specifically, it uses Description

Logics to formulate the objects and the relationships between them. The first prerequisite, is a **Black-Box Classifier** $F : \mathcal{D} \to [0,1]^c$, where c is the number of classes, and $\mathcal{D}$ is the domain of inputs that the classifier accepts. We only need to have a Black-Box access to this classifier, meaning that we can only provide an input and observe the output (predicted class).

The key component of the proposed framework, is the utilization of an **Explanation Dataset**. Given the domain $\mathcal{D}$ of the classifier, and a set of *atomic concepts* CN, an explanation dataset is a *set of tuples* $\{(x_i, C_i)\}$ where $x_i \in \mathcal{D}$ and $C_i \subseteq$ CN. For the *atomic concepts* CN, there also needs to be a *TBox*, that provides us with a hierarchy of these *concepts*. This *Explanation Dataset*, along with the *TBox* are able to provide us with meaningful Counterfactual Explanations, using the idea of *Conceptual Edit*.

Using the *TBox*'s axioms, written in the form $A \sqsubseteq B$, we can view it as a knowledge graph, where every axiom is replaced with an edge. Also, in order to construct a connected graph, any concept C that isn't included in any other concept, is automatically included in the $\top$ concept, as in $C \sqsubseteq \top$. Then, the *Concept Distance* $d_T(A, B)$ between two concepts A and B is defined as the shortest path on the (undirected) final graph, between these two concepts. To further expand this definition to include *Sets of Concepts*, the authors propose the **Concept Set Edit** defined on a set $\mathcal{A} \subseteq$ CN, which consists of a combination of the three following operations:

- **Replacement** of a concept $A \in \mathcal{A}$ with a concept $B \notin \mathcal{A}$, written as $e_{A \to B}(\mathcal{A})$. The cost of this operation is equal to $d_T(A, B)$.
- **Deletion** of a concept $A \in \mathcal{A}$, written as $e_{A \to \top}(\mathcal{A})$. The cost of this operation is equal to $d_T(A, \top)$.
- **Insertion** of a concept $B \notin \mathcal{A}$, written as $e_{\top \to B}(\mathcal{A})$. The cost of this operation is equal to $d_T(\top, B)$.

So now we can define the **Concept Set Edit Distance** between sets $\mathcal{A}, \mathcal{B} \subseteq$ CN, which is the *minimum cost* of a set of concept edits which transform $\mathcal{A}$ into $\mathcal{B}$. This is of great importance, because now we basically have a way of measuring the *"conceptual"* distance of any pair of samples in the *Explanation Dataset*. We can further define the **Significance of Transformation** between two samples (x_a, C_a) and (x_b, C_b) of the explanation dataset as:

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

After that, the main idea is to use these *significance* scores, to compute *Local* and *Global* counterfactual explanations. This is done by building an intermediate weighted, directed graph, whose nodes are the samples (x_i, C_i), and the edges between (x_i, C_i) and (x_j, C_j) has weight $\frac{1}{\sigma(i,j)}$. Now the *Local* and *Global* counterfactual explanations can be defined as:

- For the **Local Counterfactual Explanation**, given an input sample (x_i, C_i), and a target class H (different form the class attributed to x_i by the classifier F), the *Local Counterfactual Explanation* is equal to the path of the intermediate graph, between the node (x_i, C_i) and any sample (x_j, C_j) where $F(x_j) = H$. If this path is the shortest path possible, then this is an **Optimal Counterfactual Explanation**.

- For the **Generalized Counterfactual Explanation** (or Global Counterfactual Explanation), given any subset $R_{\mathcal{Q}} \subseteq \mathcal{D}$ of the *explanation dataset* (e.g. the subset of all samples that are classified to a specific class by the classifier), then $E_{R_{\mathcal{Q}}}$ is defined to be the multi set containing the labels of *optimal local counterfactual explanations* from each element of $R_{\mathcal{Q}}$ to the desired class H. Given a set of concepts $\mathcal{C} \subseteq$ CN, a **generalized counterfactual explanation** is an assignment of importance to every concept C $\in \mathcal{C}$, where the importance of a concept C is defined as: $\frac{|\{e_{x \to C} \in E_{R_{\mathcal{Q}}}\}| - |\{e_{C \to x} \in E_{R_{\mathcal{Q}}}\}|}{|R_{\mathcal{Q}}|}$ where $x \in$ CN.

Simply put, a *Local Counterfactual Explanation* provides with the *minimum conceptual modifications*, that need to be applied to a specific input sample, in order to change the classification outcome. On the other hand, *Generalized Counterfactual Explanation*, provides us with a significance for each concept, if we want to transform from an *original class*, to a different *target class*. The framework can be visualized in the following figure:

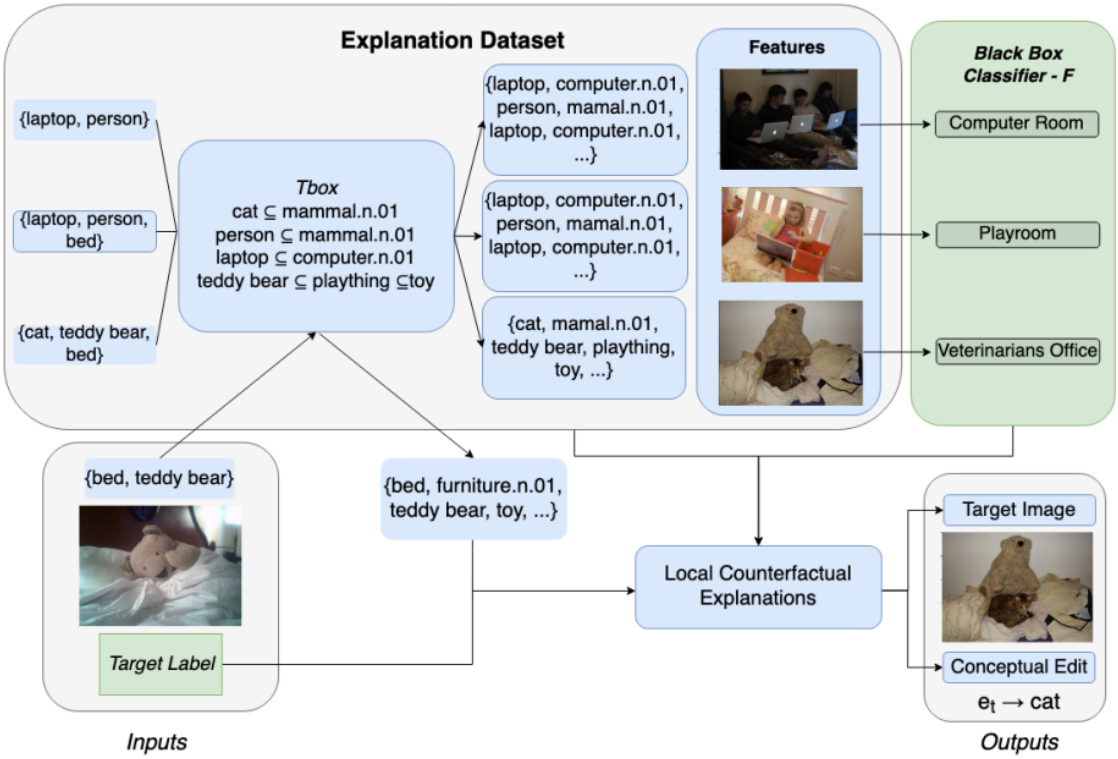


Figure 6.2.1: Framework Architecture for providing Counterfactual Explanations [29]

6.2.1 Algorithmic Implementation

Two fundamental algorithms are utilized in the framework, Dijkstra's algorithm for finding the shortest path, and Karp's Algorithm for solving the bipartite matching problem. Specifically:

- **Dijkstra's algorithm** is used in two steps of the framework: computing the shortest path between two concepts in the TBox graph, and computing the shortest path between two data points in the constructed graph.

  To compute the shortest path between two concepts in the *TBox* graph, the framework first constructs an undirected graph where each concept is a vertex and each subsumption relationship between concepts is an edge. The weight of each edge is set to 1. Dijkstra's algorithm is then used to find the shortest path between the two concepts. The time complexity of this step is $\mathcal{O}(|CN| + |T|log|CN|)$, where CN is the set of concepts and T is the set of subsumption relationships.

  Similarly, in order to compute *Local Counterfactual Explanations*, Dijkstra's algorithm is used on the constructed graph. On this graph, nodes represent data points of the *explanation dataset*, and the edges between them have weight equal to the inverse of their *Significance of Transformation*. So in this case, Djikstra's algorithm is used to provide us with the **Optimal Local Counterfactual Explanation**.

  The output of Dijkstra's algorithm in both cases is the shortest path between two concepts or data

points. This output is useful because it provides a quantitative measure of the distance between two concepts or data points, which can be used to identify the most important concepts for a given prediction and to generate meaningful counterfactual explanations.

- **Karp's algorithm**[51] is used in the *Concept Set Edit Distance* computation step of the framework. The *Concept Set Edit Distance* is a measure of the distance between two sets of concepts, and is used to identify the minimal and meaningful "conceptual edits" that need to be made to a data point to change the prediction of a black-box classifier to a desired class.

  To compute the set distance between two sets of concepts, the framework first removes any common elements from both sets. It then creates a **bipartite graph** where each element of the first set is connected to all elements of the second set with an edge, and the weight of each edge corresponds to the concept distance between the two elements. Karp's algorithm is then used to compute the minimum weight full matching of the bipartite graph.

  Karp's algorithm is a well-known algorithm for solving the assignment problem, which is a special case of the bipartite matching problem. Specifically, for the implementation used in the framework, computing the minimum weight full matching, has a time complexity of $\mathcal{O}(|\mathcal{A}||\mathcal{B}|log|\mathcal{B}|)$, where $\mathcal{A}$ and $\mathcal{B}$ are the two sets of concepts being matched. This is because the algorithm is applied to a bipartite graph with $|\mathcal{A}|$ vertices on one side and $|\mathcal{B}|$ vertices on the other side.

  The output of Karp's algorithm is the **minimum weight full matching of the bipartite graph**, which corresponds to the minimal set of concept edits needed to transform one set of concepts into the other. This means, that without any other transformations to the algorithm's output, we are provided with all the required edits (and their cost), needed for an optimal transformation of concept set $\mathcal{A}$ to concept set $\mathcal{B}$.

6.2.2 Enriching the Explanation Dataset

The authors, propose further enhancements to the above framework in Dervakos et al. (2023) [21]. The main modification is the formulation of the **Explanation Dataset**. In the framework presented above, each data point x_i, was paired with a set of concepts C_i, and this set of concepts was used for the computation of Concept Set Edit Distance.

In the newer version [21], all the data points are included in a single *ABox*. This means, that each data point, has a *1-1* correspondence with subgraph from the *ABox*, which is the connected component that includes this data point. The main advantage of using this method of representation, is that the new C_i of each data point, is essentially a knowledge graph. This is a much more powerful representation, compared to using just a *set of concepts*.

Though powerful, there is a new problem present with this modification. What was previously "Set Edit Distance", is now "Graph Edit Distance", but as we have already explained in previous chapters, this problem is NP-Complete. The way that the authors tackled this problem, was viewing the knowledge graphs, as **sets of Concepts and Roles**, and only taking into account 1-hop neighbors of each concept. So the problem, is again reduced to bipartite matching, with the addition of roles, as well as concepts. The only prerequisite, is that the *TBox* provides us with a hierarchy of both concepts, and roles. That way, we can compute edit distances between the roles as well.

6.2.3 Utilization of GNN Models

The framework that uses knowledge graphs for information of each data point, computes a sub-optimal Edit Distance, as we explained above, because it doesn't implement the Graph Edit Distance algorithm. This is

where GNN retrieval models can prove to be a useful and efficient tool. Specifically in this thesis, the main decisions for building the GNN models are the following:

- The GNN models that we construct and train, are evaluated on the **Information Retrieval** task. This means that, given a Graph Query, they can successfully rank other graphs, according to their similarity to the query.
- The specific architecture that we implement for the GNN, is the **Graph Autoencoder**. The two main reasons for that, are the Train/Inference times of the model, and the fact that they don't require any labels for the training process (unsupervised). The second reason is especially useful in the Graph Information Retrieval case, where the labels would mean computing the costly GED algorithm for every pair

Taking those into account, we can easily see how they can be implemented in the Counterfactual-Explanations Framework proposed by Dervakos et al.[21]. Specifically, since the information we have on the data points is formulated as a graph, we can use GNNs to rank the data points, and find the most similar ones. Also, by using Graph Autoencoders, the whole process will be efficient and easy to implement, because we don't have to compute any other kind of Distance/Similarity Metric.

6.3 Related Work

There have been several different taxonomies used to categorize **Counterfactual Explanations** methods, based on the way that they are retrieved from the machine learning model, and the dataset. Here, we will follow the work of Guidotti (2022)[38], as it is one of the most recent and comprehensive surveys on Counterfactual Explanations. The author proposed a taxonomy to categorize counterfactual explainers into four categories based on their approach to retrieving counterfactual explanations. These categories are:

1. **Optimization-based explainers:** These explainers define a loss function that accounts for desired properties and adopt existing optimization algorithms to minimize it. They aim to find the counterfactuals by solving an optimization problem that minimizes the distance between the original instance and the counterfactual instance while satisfying certain constraints.
2. **Heuristic search-based explainers:** These explainers aim to find counterfactuals through local and heuristic choices that, at each iteration, minimize a certain cost function. They use a search algorithm to explore the space of possible counterfactuals and find the one that best satisfies the desired properties.
3. **Instance-based explainers:** These explainers retrieve counterfactuals by selecting the most similar examples from a dataset. The idea behind IB explainers is to search into a reference population instances to be used as counterfactuals, by using a distance metric to measure the similarity between instances.
4. **Decision tree-based explainers:** These explainers are a type of counterfactual explainer that approximates the behavior of a black-box model with a decision tree and then exploits the tree structure to identify counterfactual explanations.

Among **Optimization-Based** explainers, one of the first and most important ones is *WACH*[117], a method that aims to minimize the distance between the original instance and the counterfactual, while also maximizing the similarity with a prototype. *CEM*[22], on the other hand, generates counterfactuals by minimizing the distance between the original instance and the counterfactual one, while also maximizing the probability of the counterfactual instance belonging to the opposite class. *MACE*[50] maps the problem of counterfactual search into a sequence of satisfiability (SAT) problems, while also formulating the necessary plausibility, actionability, and diversity constraints. *DICE*[80] is a method that uses a genetic algorithm to generate diverse counterfactuals, while also allowing the user to specify mutable and immutable features. *DECE*[22] is a method that uses an interactive framework, that provides counterfactual explanations through a visualization system. Finally, *CEGP*[69] is a method that uses a prototype-based approach to generate counterfactuals, while also minimizing the distance to the original instance.

In the field of counterfactual explanation methods utilizing **Heuristic Search Strategies**, several note-

worthy approaches emerge. *SEDC* by Martens and Provost (2014)[75] stands as a model-agnostic heuristic method specialized for textual data, employing a best-first search to iteratively refine explanations by selecting word removals that maximize class changes. *GIC* introduced by Lash et al. (2017)[60] addresses the Generalized Inverse Classification problem, offering three heuristic methods for tabular data, utilizing hill climbing, genetic algorithms, or their combination. *GSG* by Laugel et al. (2018)[61] employs a generative approach to create synthetic instances around the input, designed for numerical data. *POLARIS*, proposed by Zhang et al. (2018)[127], is a model-agnostic neural network explainer using heuristic search to select features to modify, aiming to ensure stability and returning symbolic explanations. These methods showcase diverse strategies for efficiently generating counterfactual explanations, encompassing text, tabular data, generative techniques, and neural networks, aiding in the interpretation of machine learning models.

In the realm of **Instance-Based** counterfactual explanation methods, four significant approaches are notable. *NNCE* (Nearest-Neighbor Counterfactual Explainer)[99] selects instances most similar to the input but with different labels to serve as counterfactuals, while *CBCE* (Case-Based Counterfactual Explainer)[52] refines this by combining feature values from similar instances to mimic differences. *FACE* (Feasible and Actionable Counterfactual Explanations)[87] focuses on generating "actionable" counterfactuals aligned with data distribution by constructing a graph and applying a shortest path algorithm. *NICE* (Nearest Instance Counterfactual Explanations)[9] offers versatile versions for categorical features, employing sparsity, diversity, or plausibility criteria to generate counterfactuals. It should be noted, that the framework presented in the previous section by Filandrianos et al. [29], belongs in this category as well.

In the domain of **Decision-Tree-Based** counterfactual explanation methods, several noteworthy approaches stand out. *TBCE* (Tree-Based Counterfactual Explainer) defines a generic strategy, which employs a surrogate decision tree trained on a reference dataset to approximate the black-box classifier's behavior, generating counterfactuals by examining leaf nodes with different predictions. *FT* (Feature Tweaking)[109] focuses on actionable feature modification in tree-based ensembles, aiming to transform instances to a different class while respecting multiple trees' predictions. *LORE* (Local Rule-based Explainer)[39] provides rule-based explanations and counterfactual rules by generating synthetic neighbors through a genetic algorithm and extracting decision rules from a trained decision tree. *FOILTREE*[116] constructs local foil trees in the neighborhood of a given instance to generate contrastive explanations, while *RF-OCSE* (Random Forest Optimal Counterfactual Set Extractor)[28] converts a Random Forest into a single decision tree and extracts counterfactual sets to highlight sub-regions where counterfactual conditions hold. These methods leverage decision trees to unravel the logic of black-box classifiers, enabling the generation of counterfactual explanations with various applications.

Chapter 7

Proposal

In this chapter, we propose the Graph Autoencoder models, that will be used for the Information Retrieval task, on the Scene Graph dataset, Visual Genome. Specifically, all the optimizations on the original models will be presented and explained thoroughly. In the following section, we will list all the contributions of this thesis, and after that, present the proposed models.

7.1 Contributions

The main contributions of this thesis are outlined below:

- We employ Graph Autoencoders, to tackle the traditional problem of Graph Similarity. To our knowledge, the utilization of Graph Autoencoders to solve this problem, hasn't seen a lot of academic attention so far, so we aim to provide a comprehensive overview of the problem at hand, and the methods we employ to solve it.
- The utilization of Autoencoder-based architectures, allows us to train the models in an un-supervised fashion, without the need to compute Graph Edit Distance between the graph samples. Additionally, these architectures don't require pairs of graphs for the training, further reducing the order of samples to $\mathcal{O}(n)$, compared to $\mathcal{O}(n^2)$ for the supervised models. These two fundamental aspects are the primary reason for the accelerated training of the proposed models.
- We propose several optimizations to the basic Graph Autencoders presented in Section 5.4, along with the intuition behind each one. We also evaluate the impact of the key components of the different architectures, by combining them and assessing the quantitative results on the Graph Similarity task.
- The proposed GNN models, provide us with similarity scores between all the Scene Graphs. This allows the model to be ultimately used alongside a Counterfactual Explainer Framework, similar to the one presented in Chapter 6.

7.2 Proposed Models

The GNNs models that we propose, are based on the Graph Autoencoder architecture. This explicitly determines the way that the models are trained and evaluated (inference):

- **Training:** Graph Autoencoders, receive as input a single graph, and the general goal is to reconstruct it, using the Node Embeddings, produced by the encoder (as shown in Figure 7.2.1). This means that if we obtain a dataset of N graphs, then we have N samples for training. In the case of Supervised-GNN approaches, we use *pairs* of graphs from the original dataset. This simple property, massively reduces the training time of the GNN model, from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, where n the number of samples (graphs).

Practically, the autoencoders need, at most, a few minutes to train on the dataset that we use, while the equivalent Supervised approaches, would require hours on the same dataset.

- **Inference:** For the inference of the proposed GNN model, we use only the *Encoder* part of the autoencoder. Specifically, we pass the graph as input to the *Encoder*, and we receive the final Node Embeddings (as shown in Figure 7.2.2). Then we use a *Global Pooling* method, to narrow down to a single Graph-Level Embedding. This process doesn't require any substantial computational power ($< 1sec$ for the inference of 1000 graphs). The most important advantage of most GNN approaches, compared to Graph Kernels, lies in the Inference of GNNs. While graph kernels only provide a similarity score between two graphs, GNN models can be used to solve most problems defined in graph-theory (node/graph classification, link prediction, clustering etc.), and that's due to the general-use Embeddings that they produce. These embeddings, obviously can help derive Edge Embeddings, Sub-graph Embeddings, as well as global Graph Embeddings, allowing for their application on several tasks.

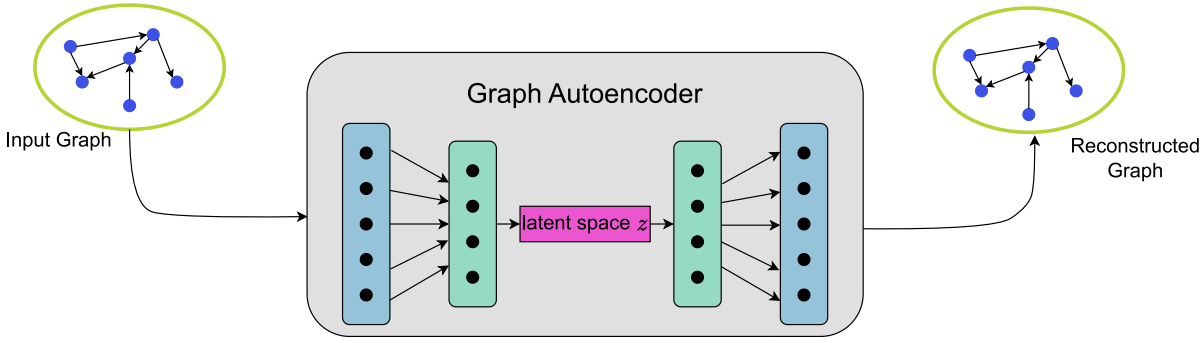


Figure 7.2.1: Graph Autoencoder utilization, for the Training phase.

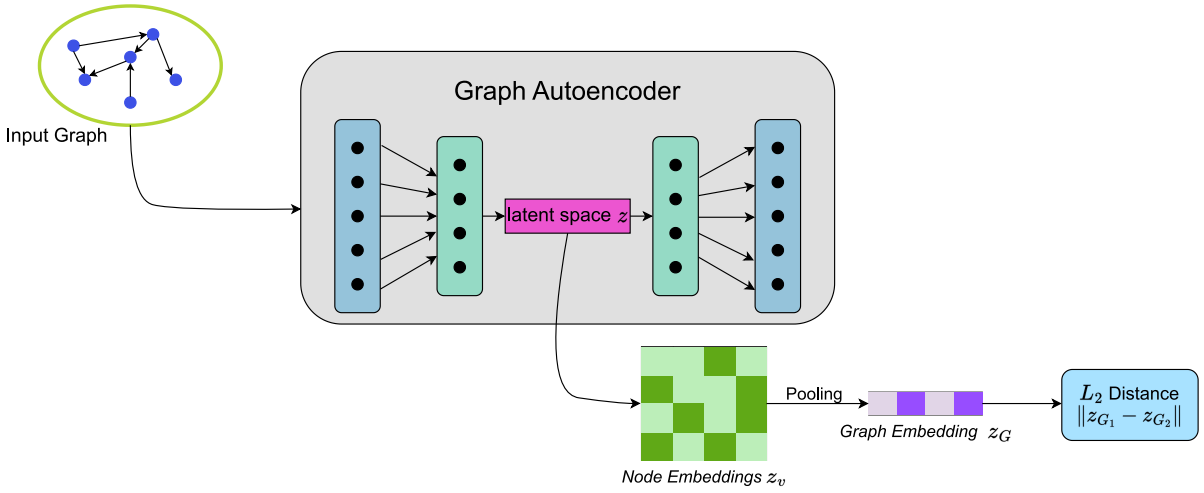


Figure 7.2.2: Graph Autoencoder utilization, for the Inference phase.

The original, base model that we will be building on top of, is the *VGAE*[54]. The model architecture, as explained in Section 5.4, is shown in Figure 7.2.3. The red rectangles, show the final *Errors*, that the model will combine and minimize.

The second fundamental architecture is the *ARVGA*[84], which adds the *Discriminator* to the whole model, in order to adversarially regulate the produced latent embeddings, as shown in Figure 7.2.4.

The first proposed model is the **Feature VGAE**, that adds a Feature Decoder, alongside the original Edge Decoder (as shown in Figure 7.2.5), from the original architecture.

The second proposed model is the **Combined VGAE** (Figure 7.2.6), which *combines* an Edge Decoder and a Feature Decoder, but they both employ a GNN module with *learnable parameters*. The main idea for this

model, is to add a layer with learnable parameters, before the *Inner Product Decoder*, in the Edge Decoder. The intuition behind this decision, is to enforce the Encoder to embed the structural knowledge of the graph, in a more complicated and intricate way, than a simple Inner Product operation.

The third proposed model is the **Combined ARVGA**, which adds the Discriminator module, to the *Combined VGAE* architecture, as shown in Figure 7.2.7. We also experiment with a variation of that, the **Combined ARVGA MLP**, where the only difference with the **Combined ARVGA** is that *learnable parameters* at the Edge Decoder, are implemented with a traditional MLP (Figure 7.2.8), not a GNN Module.

We will now provide a few details on the sub-modules of the architectures, that apply on all of the models mentioned above:

- The *GNN Modules* that will be tested, are the ones presented in Section 5.3, which are **GCN**, **GAT**, **GATv2** and **GIN**

- When there are more than one *GNN Modules* in a model, then the same module is used in each one. For example, we won't use a **GCN** for the Encoder, and a **GAT** for the Decoder. This was tested on early experiments, but the results quickly showed that it wasn't a viable option for the models. A possible explanation for that could be that different GNN Modules, encode and decode the graphs in a different way, by focusing on different features. So their collaboration, does not necessarily produce results of equivalent quality.

- For the Inference of the models, as we see in Figure 7.2.2, we need to define a **Global Pooling** function, that aggregates Node Embeddings to a single Graph Embedding. After trying the most popular functions for global pooling (mean/max/min/sum), it was quickly evident, that the **Sum** pooling function, had the best results, so all of the final models employ this function. This same argument is also supported by the creators of **GIN**[121].

- For the final stage of Inference, we use the L_2 distance of *Graph Embeddings*, in order to rank the similarity scores between them, and retrieve similar graphs. We also experimented with *Cosine Distance*, but the performance was worse than with the L_2 distance.

Specific hyperparameters and other choices (e.g. layers, latent size, epochs, batch size, optimizers etc.) of each model, will be presented in the following chapter.

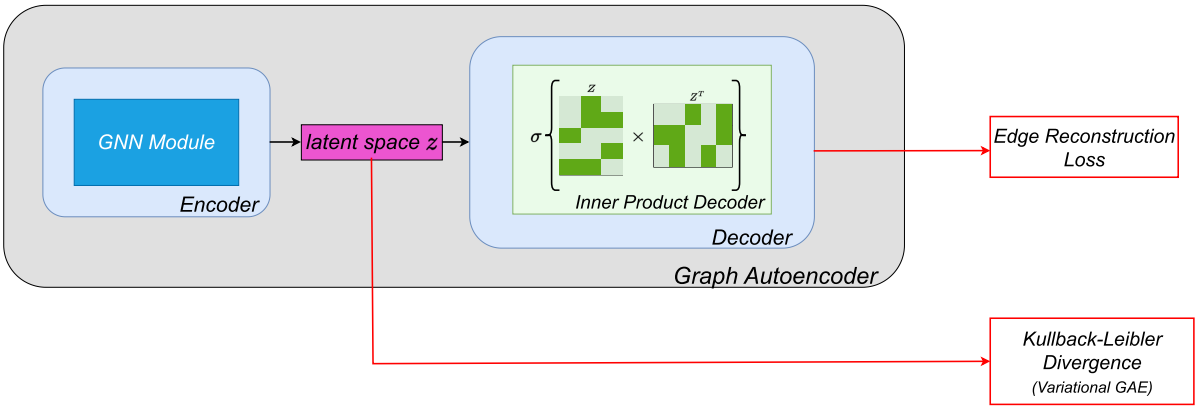


Figure 7.2.3: Original architecture, for Variational Graph Autoencoder

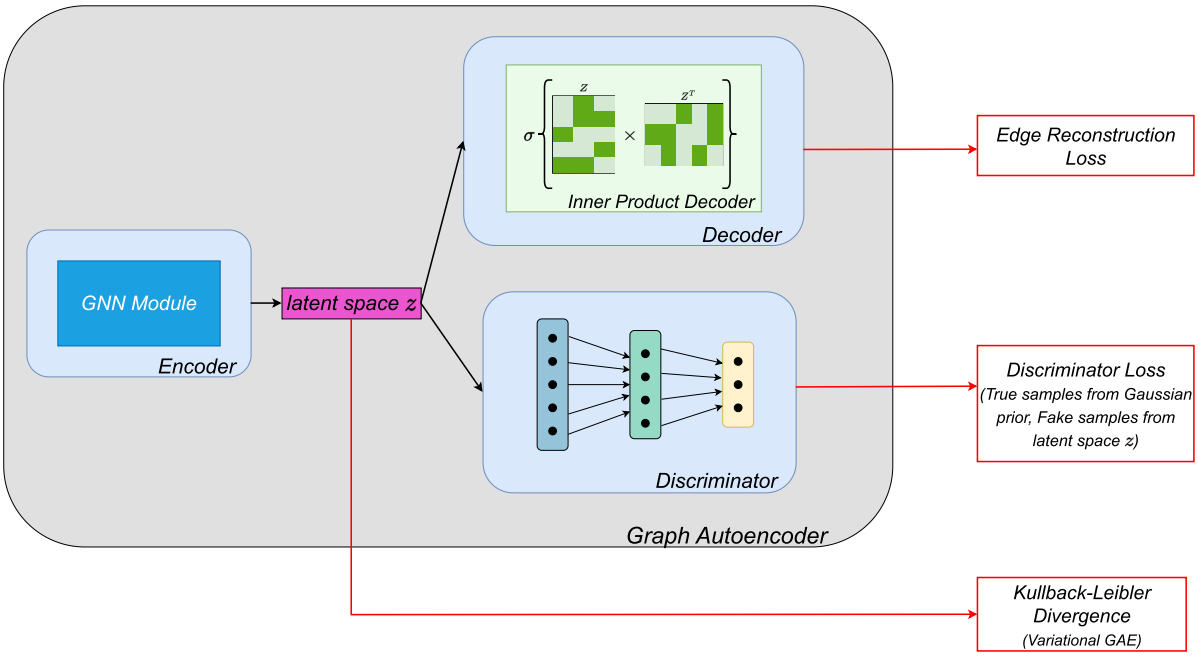


Figure 7.2.4: *Adversarially Reguralized Graph Autoencoder* architecture

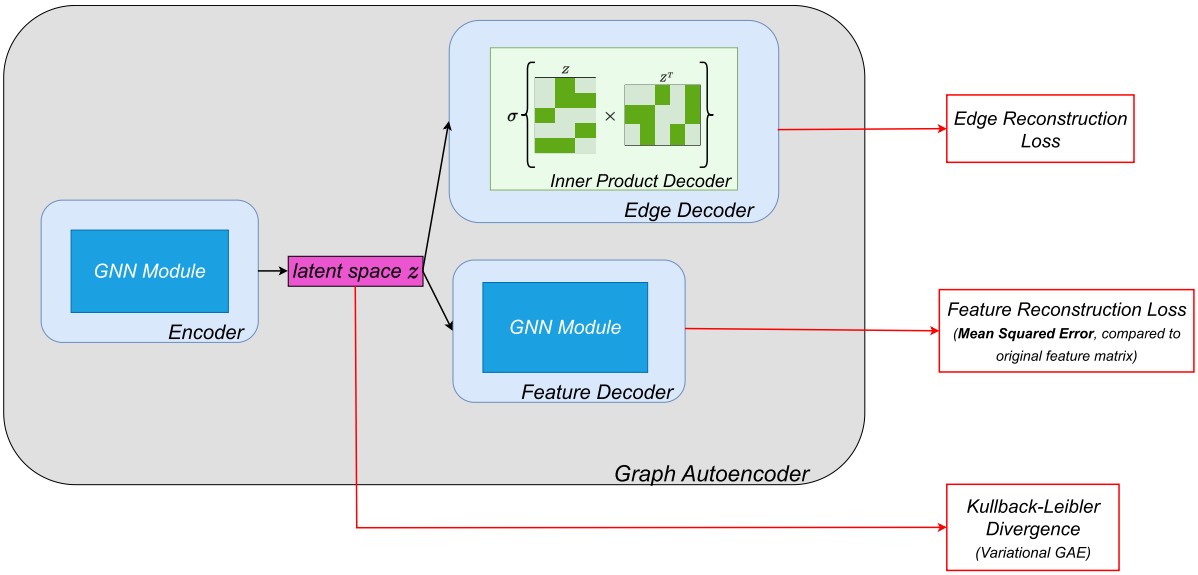


Figure 7.2.5: *Feature Graph Autoencoder* architecture

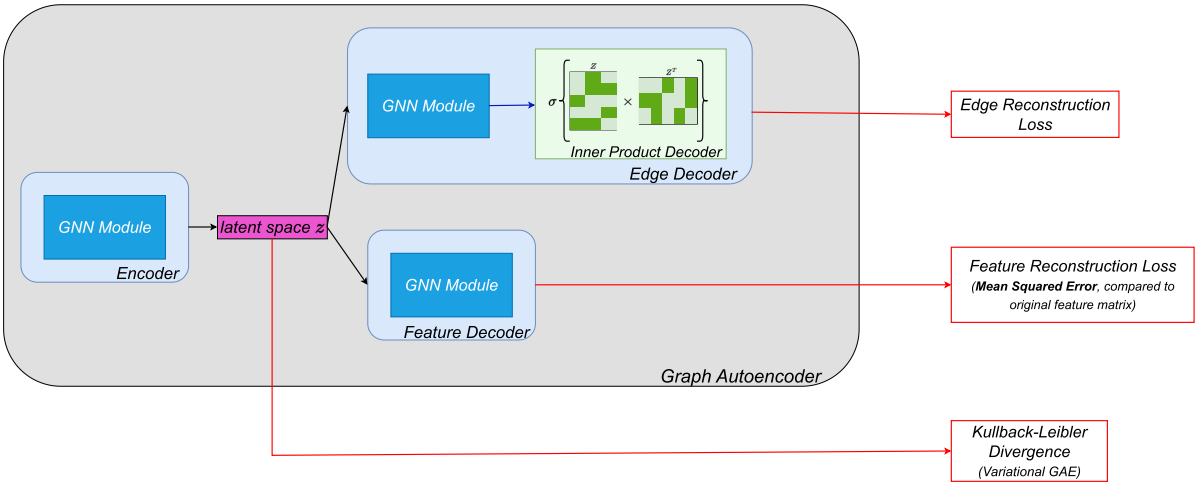


Figure 7.2.6: *Combined VGAE* architecture

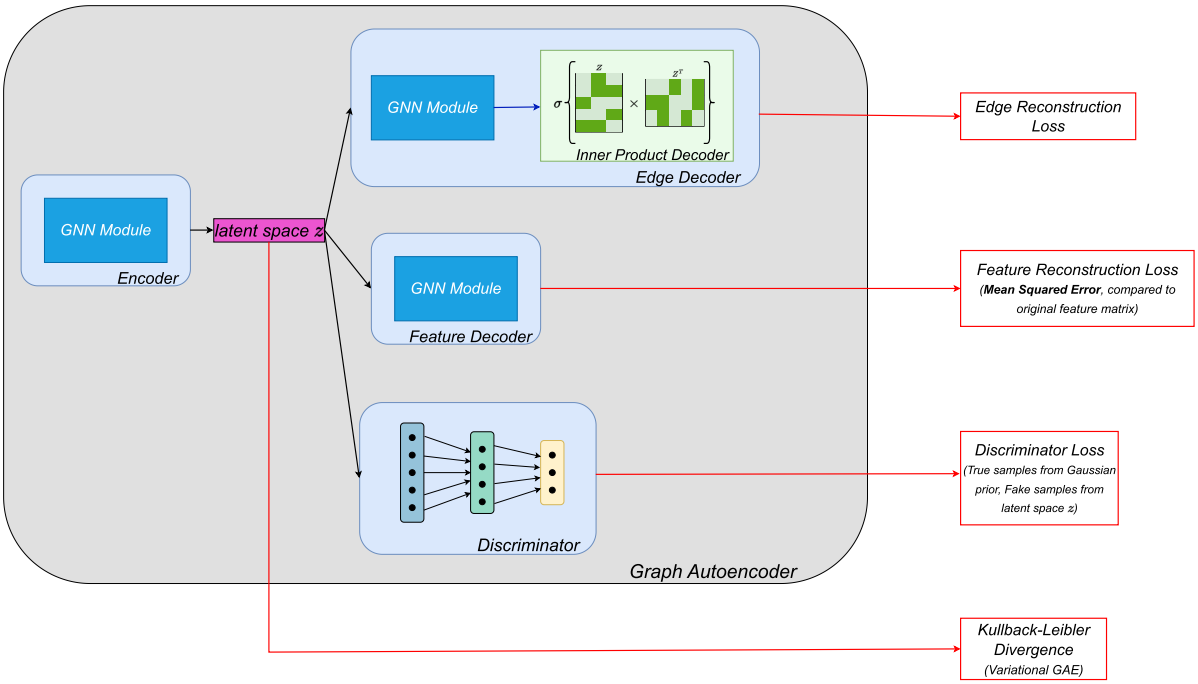


Figure 7.2.7: *Combined ARVGA* architecture

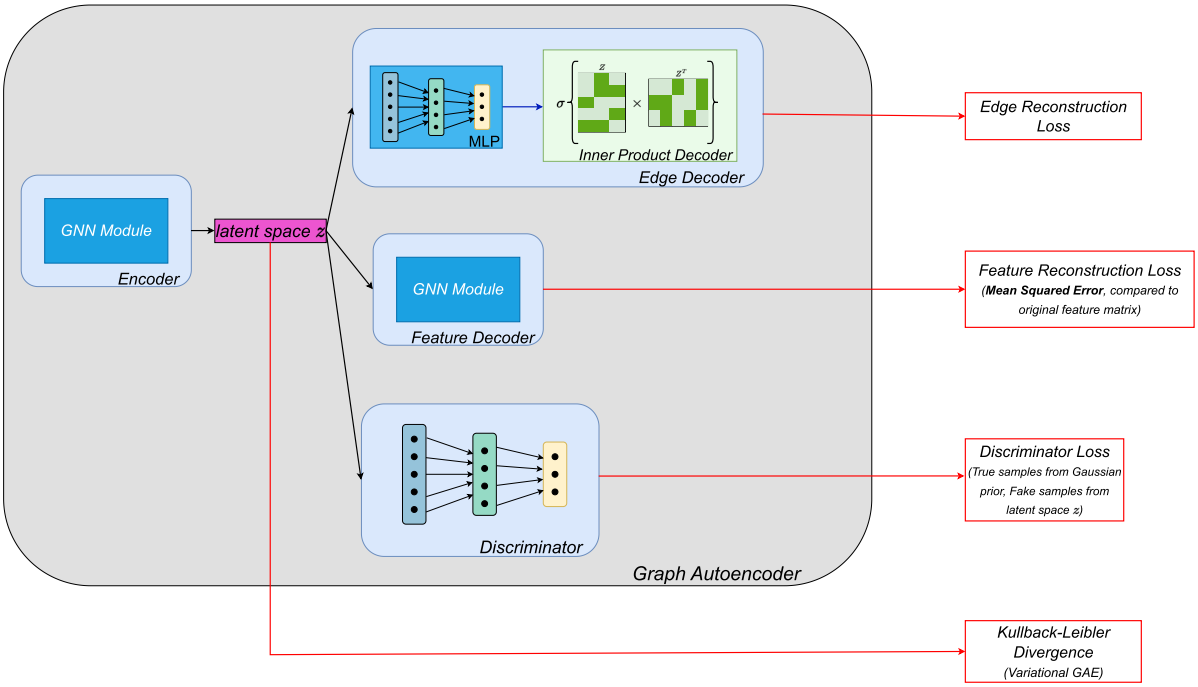


Figure 7.2.8: *Combined ARVGA MLP* architecture

Chapter 8

Experiments

In this chapter, we will explain in detail the Experiments that we carried out, in order to evaluate the proposed models, and compare them to the baseline Graph Kernels. We will start with an overview of the dataset that will be used, Visual Genome, along with the preprocessing procedure that was applied to the graphs. Then we will show how Graph Edit Distance was utilized and implemented to obtain the Ground Truth rankings, along with a comprehensive presentation of the Evaluation Metrics for the Information Retrieval task.

Following that, we will explain all the details for the Training and Inference of the GNN models, as well as the parameters for the Graph Kernels. Finally, we will present the Quantitave and Qualitative results, and compare the advantages and drawbacks of each model architecture.

Contents

8.1 Preliminaries 94
8.1.1 Dataset 94
8.1.2 Ground Truth 98
8.1.3 Evaluation Metrics 99
8.2 Training and Inference Details 103
8.2.1 Graph Kernel Parameters 103
8.2.2 GNN Details and Hyperparameters 103
8.3 Results 107
8.3.1 Quantitative Analysis 107
8.3.2 Qualitative Analysis 110

8.1 Preliminaries

8.1.1 Dataset

The dataset that will be used, is **Visual Genome**[58], a comprehensive dataset for training and benchmarking the next generation of computer vision models, that utilize graph-structured information. It contains 108,249 images from the intersection of the YFCC100M[108] and MS-COCO[68] datasets. Each image comes with its corresponding *Scene Graph*, as explained in Section 4.3. An example of an Image-Scene_Graph pair can be seen in Figure 8.1.1.

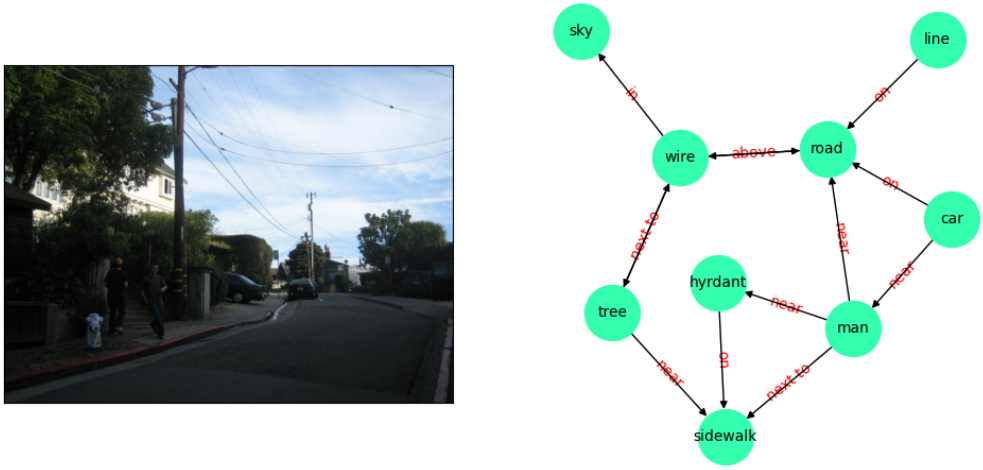


Figure 8.1.1: An example of a Scene Graph, alongside its corresponding Image

As we can see, the Scene Graphs, are capable of capturing the conceptual content of an image, which is represented by the objects and the interactions between them. Specifically, these objects are then transformed to Graph Nodes, while the interactions assume the form of Graph Edges.

Query-Answer Split

The traditional tasks of Supervised and Un-supervised Machine Learning, involve splitting the dataset into a *Train Set* and *Test Set*. For the **Information Retrieval** task on Scene Graphs, we need to have a *Query Set* of graphs, and an *Answer Set* of graphs. Specifically, the models will be given a Query Graph, and they have to rank all of the Answer Graphs, based on their similarity to the Query Graph. This will be repeated for every graph in the *Query Set*.

We chose to keep **1000** scene graphs from the original dataset. Out of these, **960** will be used for the *Answer Set* and **40** will be used for the *Query Set*. We narrowed down to these subset sizes for various reasons, including:

- **Ground Truth Computational Cost:** The Ground Truth ranking, for each of the Query Graphs, will be computed with the *Graph Edit Distance* algorithm. As it has already been mentioned (see Section 4.2.1), this is a really costly algorithm to compute. And, since the Graph Edit Distance computes the similarity between *pairs*, for every Scene Graph we add from Visual Genome, the amount of times that we have to compute GED, increases quadratically. This can quickly scale to prohibitive computation times, even for the *Approximate GED* algorithms. For example, in the *Query-Answer* split that was described above, we have to compute GED a total of $960 \times 40 = 38000$ times, and for every query graph that we add, we would have to compute GED another 960 times as well. So, it is easy to see, that the main reason for not scaling these experiments to more graphs is the computational time required for the GED algorithm.
- **Ranking for the Evaluation:** The first idea for the *Query-Answer Set* split, was to just select random pairs for the final Evaluation. This technique, poses one major problem: there is no assurance,

that there will be enough "*Answer Graphs*", in order to construct meaningful rankings, and evaluate them. What we mean by that, is that the *Information Retrieval* evaluation metrics, operate on top of a "*correct*" ranking and a "*predicted*" ranking (given a single query object). If we chose graph pairs at random, there would be no guarantee that there would be a substantial amount of graphs, that would be present in a substantial amount of graph pairs. And because there is no guarantee for that, we would end up with "*predicted*" rankings, that include a very small amount of *Answer Graphs*. Basically, we would have a very big *Query Set*, with a very small *Answer Set* for each of the query graphs. The rankings produced by that type of *Query-Answer Set* split, are much worse for evaluating the models, because all the Metrics do a much better job at representing the effectiveness of the model if the "*correct*" and "*predicted*" rankings include a big enough *Answer Set*. That is the reason that we opted for a small *Query Set*, and a much bigger *Answer Set*.

- **Qualitative Results:** For the Qualitative Evaluation of the models, we will use the corresponding images of the Scene Graphs in the *Query Set* and *Answer Set*. By having a big enough *Answer Set* (and therefore a big enough ranking for the query graphs), allows human evaluators, to better assess the performance of the proposed models. This wouldn't be as easy, if we had used random query-answer pairs for the dataset. In that case, the Qualitative Evaluation would involve presenting two pairs of random images, and comparing the two similarity scores between them that the models predicted. So in this case, a human evaluator would have to judge if the similarity of the pair $Image_1 - Image_2$ is greater than the similarity of the pair $Image_3 - Image_4$, with all four images being randomly selected from Visual Genome. We don't think that this would be a very effective way for a human evaluator to assess the results. Instead, for the proposed *Query-Answer Set* split, there would be a single query image $Image_Q$, and two rankings: $\{Image_1, Image_2, Image_3, \dots\}$ and $\{Image_1^{'}, Image_2^{'}, Image_3^{'}, \dots\}$, and a human evaluator would have to determine, which ranking is better, given the query $Image_Q$. This would basically be the question "*Which is a better Search Engine system, for the query* $Image_Q$*?*", which is a much more intuitive and easy way, for a human evaluator, to assess the performance of *Information Retrieval* Systems.

Random-Dense Subsets

Now we will explain the process that we used to sample the graphs from Visual Genome to construct the final *Query Set* and *Answer Set*.

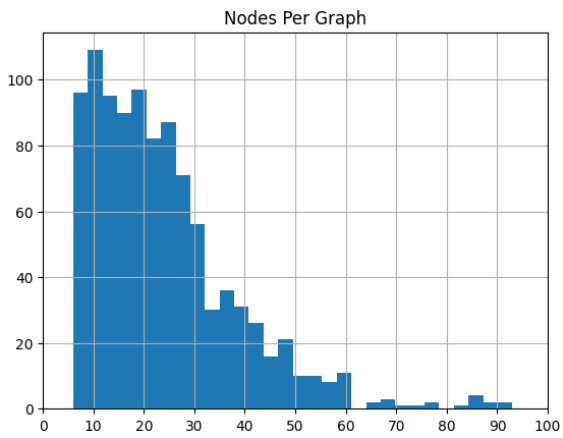
Originally, we chose 1000 random graphs, with two trivial constraints: `#Nodes` ≥ 6 and `#Edges` ≥ 3. We only set these constraints, in order to avoid graphs with minimal amount of objects and relations. We can plot some basic statistics for these graphs, such as the histograms for density, number of nodes, and number of edges, as shown in Figure 8.1.2.

The main problem here, is the Isolated Nodes Histogram. We can see that the majority of the graphs has really low density, and if we take into account the Node Histogram as well, it is easy to understand that there will be graphs, with a lot of isolated nodes. For example, a Scene Graph from the original "*Random*" subset, along with its corresponding image, is depicted in Figure 8.1.3.

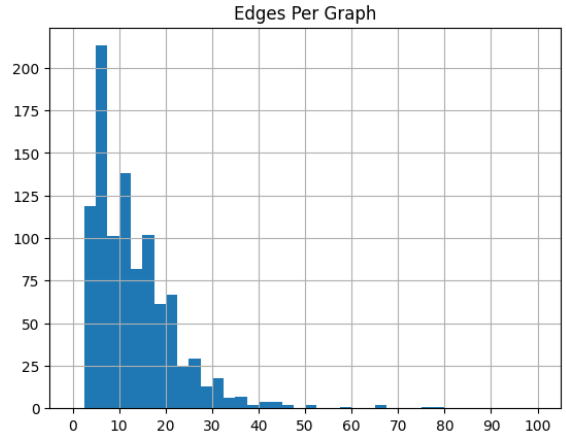
Unfortunately, most of the Scene Graphs from Visual Genome, have isolated nodes. We realized that the process of human annotation used for the Scene Graphs, ended up detecting a lot of specific objects from each image, but not necessarily any interactions with other objects.

All the Graph Kernels and the GNN Architectures, were made to operate on graph data. This means that their main tool, is to utilize the connections between the nodes, and propagate the resulting information to other nodes of the network. This powerful tool is neglected, when we decide to use graphs with as many isolated nodes as shown above. This is what drove us to conduct further experiments on a new subset, that we call *"Dense" Set*.

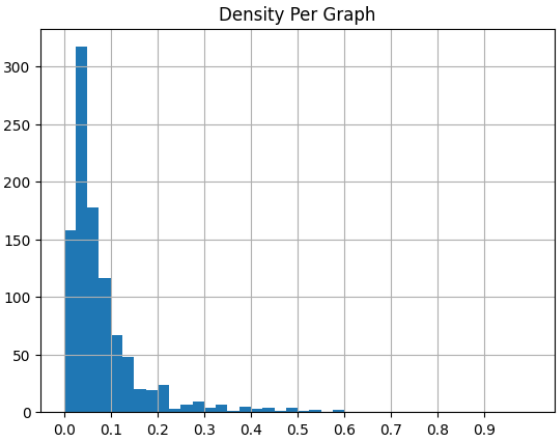
Specifically, the selection process that we implemented, to construct the *Dense Set*, is to sample Scene Graphs from Visual Genome, with the following constraints: $3 \leq$ `#Nodes` ≤ 15, `#Edges` ≥ 3 and $0.1 \leq$ `Density` ≤ 1.0. The main difference here, is the addition of the constraint placed upon the Density of the graphs. This is what will lead to greater utilization of the GNN Message-Passing ability. The upper limit on Density was set in order to exclude some outliers in the Visual Genome that has the same edges multiple times. Finally,



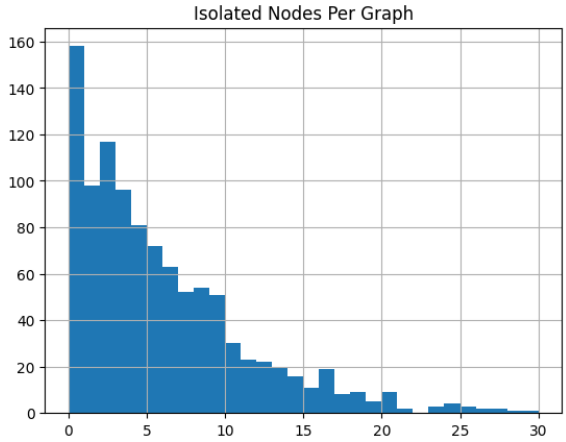
(a) Node Histogram for Random Subset



(b) Edge Histogram for Random Subset



(c) Density Histogram for Random Subset



(d) Isolated Nodes Histogram for Random Subset

Figure 8.1.2: Graph Statistics for the Random Subset

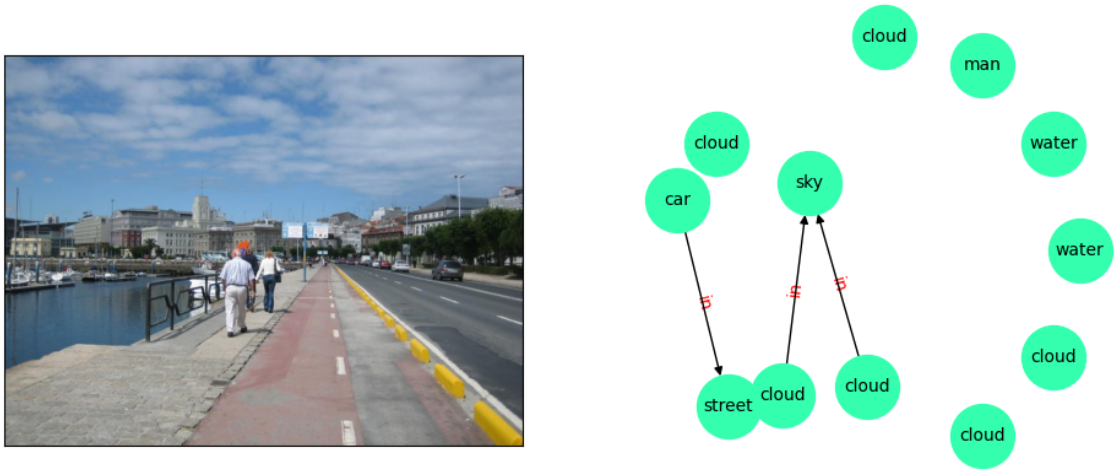
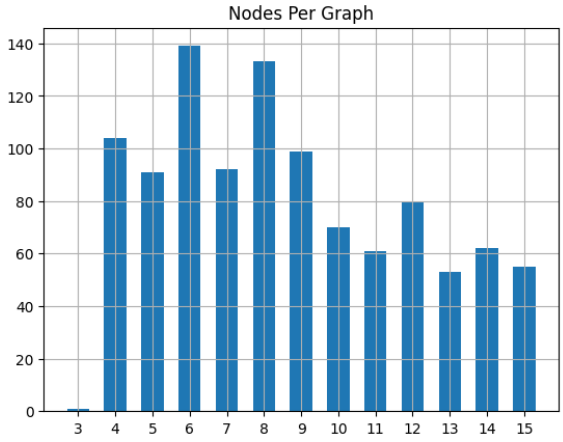


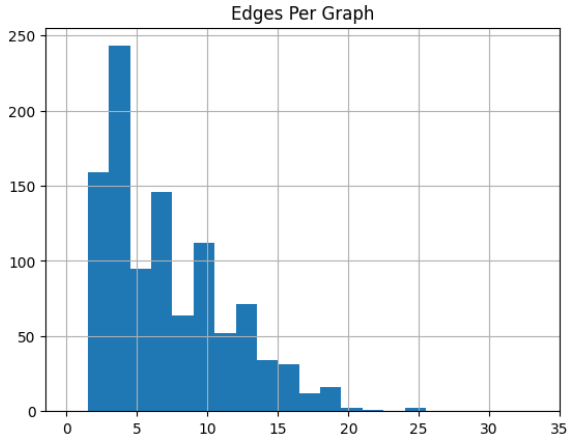
Figure 8.1.3: A Scene Graph alongside its corresponding Image, where the graph contains numerous isolated nodes.

the upper and lower limit of the amount of nodes, was set so that the order of the graphs being compared,

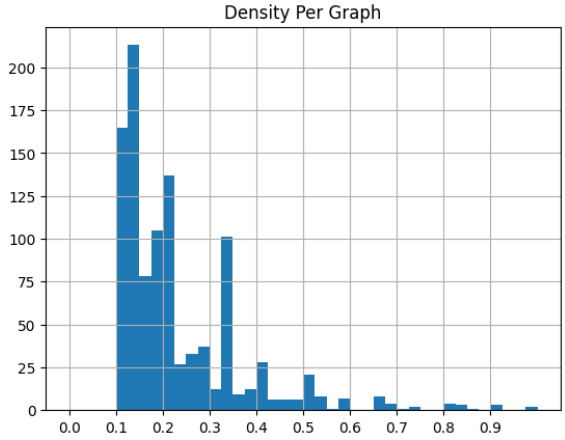
would belong to the same order. This does not hold in the Random Subset, as there are many graphs with $6-10$ nodes, as well as with ≥ 50 nodes. There are a few graphs from the original *Random* subset, that are also in the *Dense* subset, because they fulfill the aforementioned prerequisites. The graph statistics for the new *Dense Set*, are shown in Figure 8.1.4.



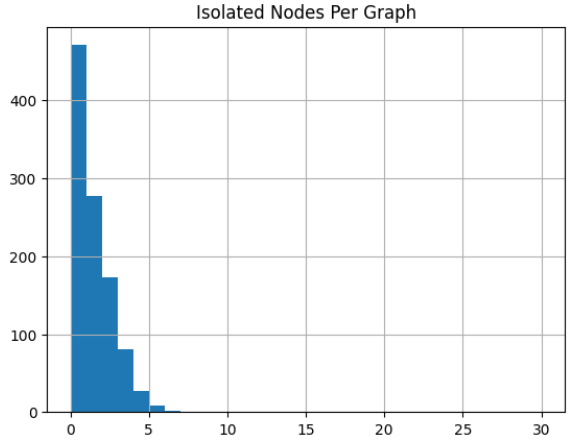
(a) Node Histogram for Dense Subset



(b) Edge Histogram for Dense Subset



(c) Density Histogram for Dense Subset



(d) Isolated Nodes Histogram for Dense Subset

Figure 8.1.4: Graph Statistics for the Dense Subset

For the *Query-Answer Set* split, in both case (*Random* and *Dense*), we just select 40 random graphs to be the *Query Set*, and the rest represent the *Answer Set*.

Graph Preprocessing

The preprocessing of the Scene Graphs, takes place in two stages. For the first stage, some specific operations are applied on all of the graphs, regardless of their later use. These operations are the following:

- **Attribute Removal:** In the original Visual Genome dataset, each object might have some attributes that describe it. In the end, we didn't keep these attributes in the graphs of the final subsets, mainly for two reasons. Firstly, we didn't want to keep redundant information in the graph, that could prove to be a hurdle in the Graph Similarity task. For example, if we had a node *"Airplane"*, that connected to its attribute node *"Green"*, we would probably have gotten a graph that was really similar to it because there is an object that is also Green. What this means, is that the GED algorithm, as well as the Graph Kernels and GNN models, might have focused more on the existence of *"Green"*, rather than the actual important semantic content of the image, which is the existence of *"Airplane"*. The

second reason that we didn't keep the attributes, has to do with the *WordNet Hierarchy*[77], which is fundamentally a hierarchy of Noun hypernyms and hyponyms. Even though it provides ways to compare the similarity of synsets that are not of the same type (e.g. noun "Airplane" and adjective "Green"), we have seen that it isn't a reliable way to compute a similarity score between them, as they don't represent things in the same class.

- **Relationships as Edges:** We chose to formalize relationships as Edges, and not as a separate node that is connected to the Subject and the Object. Again, the choice for this was mainly the *WordNet Hierarchy*, which we can't be reliably used to compare an Object (Noun), to a Relationship (Verb).
- **Relationship Type Removal:** Along with the Attributes, the original Scene Graphs from Visual Genome, have numerous types of relationships, that connect pairs of objects. We didn't keep these types, and just used un-typed, directed edges for the final subset. This decision was made mainly because most of the Graph Kernels and GNN modules used, don't take into account Edge Attributes/Features. And besides that, even the approximate algorithms of Graph Edit Distance, mostly take into account the Node Information (as explained in 4.2.1). So if we ended up keeping this information, it would just be redundant, and wouldn't be utilized as much as we would want to.

The second stage of preprocessing, regards the representation of Node Features. Specifically, three datasets were created, whose only difference was the Node Features representation:

1. **Strings for Node Features:** In this dataset, a node feature would be a simple string of the Object, such as *"Dog"*. This dataset is the one used by the Graph Kernels, because they usually can't effectively utilize any other form of information (such as an embedding), and they work with Node Labels.
2. **Word Embeddings for Node Features:** As we have already mentioned, GNN Modules, expect the Node Features, to be in a numerical form, and specifically, a Feature Matrix. To implement this matrix, we will use the GloVe Word Embeddings[86], because they have been used for many years in the field of NLP, and they have proven their benefit numerous times. For the actual transformation, we just take the Object Strings, and use the Look-Up table for GloVe, to extract the final embeddings. The specific version that we used, are the GloVe embeddings with size 100. This dataset is used as input for the GNN models.
3. **Synsets for Node Features:** For this dataset, we utilize the *"is-a"* hierarchy, provided by WordNet[77]. This dataset, is used as input to the *GED algorithm*, to compute the Ground Truth similarity scores. This hierarchy, is basically a *DAG*, where there are edges pointing from a hypernym to a hyponym, and it allows for Similarity Scores, to be defined and measured. The exact procedure will be explained in the following section.

8.1.2 Ground Truth

For the Ground Truth similarity scores, and therefore Ground Truth rankings, we will use the *Graph Edit Distance* algorithm. As mentioned in Section 4.2.1, the GED algorithm returns the edit cost, of transforming graph G_i to an isomorphic of graph G_j. The disadvantage of this, is that the exact computation of the edit cost can't be done in polynomial time, that's why almost everyone uses an approximate algorithm. The most popular approximate GED algorithms were presented in Section 4.2.1.

For this thesis, we will use the **Bipartite Matching** variation of GED, who takes into consideration mostly Node Information, when searching for possible edits. The choice for this approximate algorithm, was mostly due to the fact that many real-world applications of GED, considered this variation a good trade-off between computational cost, and approximation of the optimal solution.

WordNet played a key role in the computation of GED. *WordNet* was introduced in 1995 by George A. Miller [77], and it provides a lexical database for English, for general-purpose use. Specifically, in this thesis, we use a hierarchy constructed within the *WordNet* framework, the Noun *"is-a"* hierarchy. It is basically a *DAG*, that defines hypernym/hyponym relations between nouns. An example of that can be seen in Figure 8.1.5.

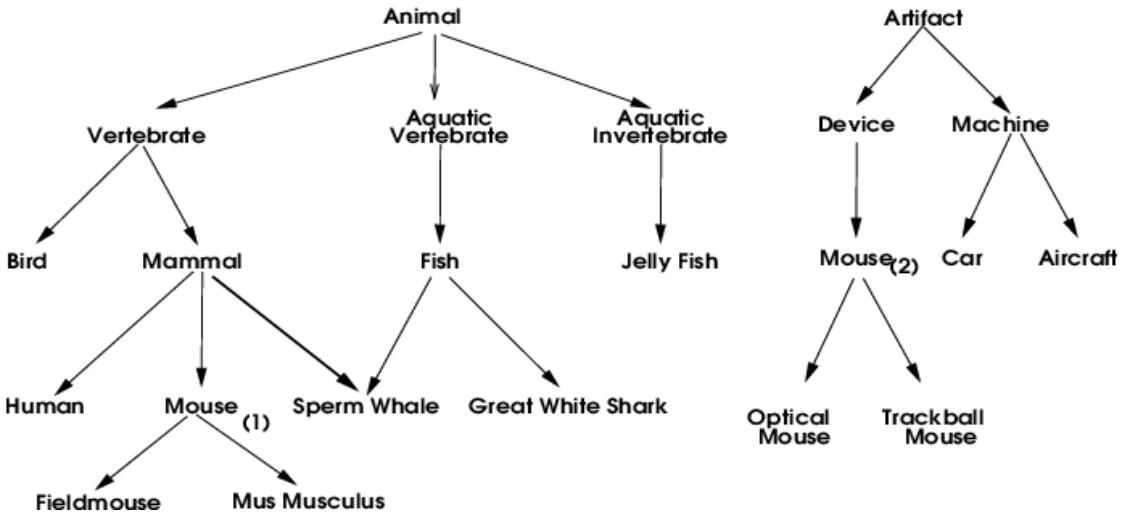


Figure 8.1.5: Example of WordNet Noun Hypernym/Hyponym Hierarchy [59]

We utilize this Hierarchy, because the Visual Genome annotations for the objects, already include the corresponding *Synset* in the Hypernym/Hyponym Hierarchy. So all objects, are already matched with a Node in the aforementioned *DAG*. This allows us to use distance metrics on this *DAG*, in order to compare objects, and find their similarities.

Specifically, we use the *NLTK python package* [3], which provides an easy-to-use API for the WordNet database, including the Noun Hierarchy. For the computation of GED, we utilize this API, to provide the *Bipartite Matching* algorithm, with the Node Insertion/Deletion/Substitution costs. The API, has implemented the function `path_similarity`, which returns a score denoting how similar two word senses are, based on the shortest path that connects the senses in the (undirected) *is-a* (hypernym/hypnoym) taxonomy. The score is in the range 0 to 1. We can use this function, to find the similarity between a Noun sense and a Verb sense, but in this case, it automatically creates a fake root node, in order to create a path between these two nodes. That's the main reason that we didn't include Attributes and Relationship Types in the Scene Graphs, because this solution didn't seem to return reliable results.

Using the `path_similarity`, we then derive the similarity score between all possible of pairs of nodes from graph G_i and graph G_j. We then parse the $1 - similarity\_score$ to the GED function, as the Node Substitution costs. For the Node Insertion and Deletion costs, we find the between of each Node, to the *Entity* Node. This is basically the root node of the *DAG*, as it has no in-edges, so there isn't any other Sense, that is more generic than this one.

We repeat the procedure explained above, for every possible pair between the *Query Set* and the *Answer Set*, for both the *"Random"* and the *"Dense"* datasets. In the end, we have the Edit Distance, between all the graph pairs, as computed by the GED algorithm. For the *"Random"* dataset, the runtime was approximately ~ 5 hours, and for the *"Dense"* dataset, the runtime was approximately ~ 1 hour.

8.1.3 Evaluation Metrics

Now we will present the three metrics that will be used for the Quantitative Evaluation and Assessment of the performance of the Graph Kernels, and the GNN Models: *NDCG*, *MAP* and *MRR*. It should be mentioned that all three metrics, are specifically designed to **measure the performance of an Information Retrieval System**. This means that the inputs include (at least), a "True" ranking and a "Predicted" ranking, given the same Query object. In our case, the "True" ranking are the first 50 objects returned by the GED algorithm, and the "Predicted" ranking are the objects returned by the model that we want to test. Particularly, we evaluate all the systems by taking into account the first 10 returned objects.

The choice of the sizes 50 and 10 is mostly intuitive, because there is no standard way in selecting the final

length of the ranking. In our case, when choosing the length of the final "True" ranking, the following two problems led us to the final decision:

- If we choose the "True" ranking to be really long, then a metric like *MRR* will almost always have a really high value, even 1.0. This is because, almost every item in the "Predicted" ranking, ends up being a correct item (in the sense that it belongs in the "True" ranking). So, as we keep increasing the length of the "True" ranking, there will be no "incorrect" items in the "Predicted" ranking.
- On the other hand, if we choose the "True" ranking to be too short, then for the same reason, we will have too many "incorrect" items in the "Predicted" ranking, therefore rendering useless some metrics, like *MRR*

Through trial and error, we found out that setting the "True" ranking length to 50, and the "Predicted" ranking to 10, will give us a sufficient trade-off between the two problems mentioned above, and enables us to gain a thorough understanding of the models' performance.

NDCG@10

NDCG, which stands for **Normalized Discounted Cumulative Gain**, is an evaluation metric commonly used in Information Retrieval tasks to measure the quality and effectiveness of search engine rankings or recommendation systems. It addresses the challenge of assessing the relevance of ranked items when there are multiple items to consider, and it takes into account both the relevance and the position of items (order-aware) in a ranked list. *NDCG* is particularly useful when dealing with scenarios where the relevance of items can vary and where users tend to focus more on the top of the ranked list.

Here's how *NDCG* is computed:

1. **Discounted Cumulative Gain (DCG):** *DCG* is a measure that assigns higher scores to items that are not only relevant but also appear higher in the ranked list. It computes the cumulative gain of items, with decreasing weights for items as their position in the list increases. The formula for *DCG* at position K is:

$$DCG@K = \sum_{k=1}^{K} \frac{rel_k}{log_2\,(1+k)}$$

   where rel_k is the is the relevance score of the item at position k, and K is the total length of ranking that is being considered (in our case it is 10). The log_2 denominator, acts as a penalty function for lower-ranked items. In order to compute relevance scores for the "True" rankings computed by GED, we take the inverse Edit Distance $\frac{1}{edit\_dist}$ of the first 50 items, and then we use a *Min-Max Scaler*, to squash them in the range $[0, 10]$. The first operation is done because the *Edit Distance* as a concept is inversely related to *Similarity/Relevance Score*. The second operation, is simply to avoid extreme values. The Histogram of the final relevance values is shown in Figure 8.1.6

2. **Ideal Discounted Cumulative Gain (IDCG):** *IDCG* is the maximum possible *DCG* that can be achieved for a given list of items, assuming they are optimally ranked according to their relevance. *IDCG* is calculated in the same way as *DCG*, but considering the best possible order of items (the "True" ranking). This is computed to solve the main problem with *DCG*, which is the fact that it doesn't have a maximum value, like *Precision* of *MRR*. We then use then use it compute the final *NDCG*.

3. **Normalized Discounted Cumulative Gain (NDCG):** *NDCG* is the normalized version of *DCG*, which takes into account the relative performance of a ranked list compared to the ideal scenario (*IDCG*). It is calculated as the ratio of the *DCG* of the "Predicted" ranking, divided by the *DCG* of the ideal ranking:

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

NDCG values range from 0 to 1, where 1 represents a perfect ranking that aligns with the ideal order of items, and lower values indicate poorer rankings.

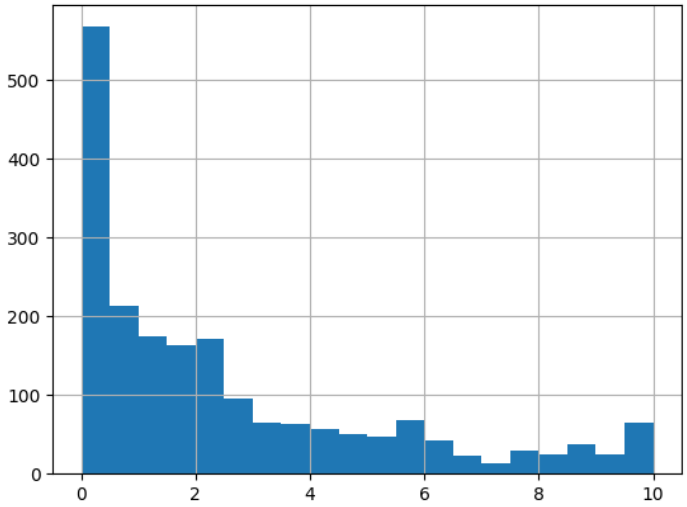


Figure 8.1.6: Final Relevance Scores for NDCG, computed by the Ground-Truth rankings of GED

NDCG is especially valuable because it captures both the relevance of items and their position in the list. This makes it suitable for evaluating search engine results, recommendation systems, and any scenario where ranking matters. It offers a more comprehensive view of system performance than simpler metrics like precision or recall, which focus solely on the presence or absence of relevant items. By considering the ordering of items and their relevances, *NDCG* provides a nuanced assessment of the quality of ranked lists in information retrieval tasks. Also, in the case where we have multiple queries and the corresponding "Predicted" rankings (40 queries in our case), we simply take the mean of the *NDCG* score of each query. Even though *NDCG* can capture and assess many aspects of the model that is being tested, the main disadvantage is that it is not easily interpretable by humans. So for example, if we know that an Information Retrieval system achieves *NDCG* score of 0.42, we don't really know anything about the quality or the performance of the model, and that is because of the log_2 terms, as well as the relevance scores. The only actual interpretation of *NDCG*, is when we assess multiple systems, where we compare their final *NDCG* values.

MAP@10

MAP, which stands for **Mean Average Precision**, is an important evaluation metric used in information retrieval tasks to assess the quality of search engine rankings, recommendation systems, and other tasks involving ranked lists of items. *MAP* takes into account both the precision and the average precision of a ranked list, providing a comprehensive measure of the system's effectiveness in retrieving relevant items.

Here's how *MAP* is computed:

1. **Precision:** Precision at a given position k in the ranked list measures the proportion of relevant items among the top k items. It is calculated as:

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

   Precision focuses on the correctness of the top k items but doesn't consider the order beyond that (order-unaware).

2. **Average Precision (AP):** Average Precision computes the average of precision values across all positions where a relevant item is present in the ranked list. It takes into account both the precision

and the position of relevant items. For each relevant item at position k, the precision at that point is calculated and then averaged over all relevant items. The formula for AP is:

$$AP@K = \frac{\sum_{k=1}^{K} (Precision@k * rel_k)}{\text{number of relevant results}}$$

Here, K is the total number of items in the list, and rel_k is the Relevance at position k, but it is not defined the same way as in *NDCG*. Here, rel_k is an indicator variable that is 1 if the item at position k is relevant and 0 otherwise.

3. **Mean Average Precision (MAP):** *MAP* is the mean of Average Precision values across different queries or datasets. It provides an overall measure of the system's performance by considering the average precision of individual ranked lists. The formula for MAP is:

$$MAP@K = \frac{1}{Q} \sum_{q=1}^{Q} AP@K_q$$

Here, Q represents the total number of queries, which is 40 in our case.

MAP is a robust evaluation metric because it rewards systems that not only retrieve relevant items but also rank them higher in the list. It considers both precision and position, providing a balanced view of system effectiveness. *MAP* is particularly useful when dealing with situations where there is a varying number of relevant items per query or dataset. It is widely used in information retrieval tasks, such as web search, recommendation systems, and document retrieval, to assess and compare the quality of different algorithms or models.

MRR@10

Mean Reciprocal Rank (*MRR*) is yet another popular evaluation metric used to evaluate the performance of information retrieval systems, that return a final ranking. *MRR* focuses on the position of the first relevant item in the ranked list and provides a simple yet informative way to evaluate the quality of systems that retrieve relevant items.

Here's how *MRR* is computed:

1. **Reciprocal Rank (RR):** Reciprocal Rank for a query is the reciprocal of the position of the first relevant item in the ranked list. In other words, if the first relevant item is at position k, then the Reciprocal Rank is $1/k$. If there are no relevant items in the list, the Reciprocal Rank is 0.
2. **Mean Reciprocal Rank (MRR):** *MRR* calculates the average Reciprocal Rank across different queries. It provides a single value that represents the average quality of the ranked lists. The formula for *MRR* is:

$$MRR@K = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{rank_q}$$

Here, the addition of $@K$, is just a constraint, to only look at the first top K ranked objects, as returned from the Information Retrieval system. So for example, if the first relevant item is at position $K + 1$, then the $MRR@K$ score will be 0, not $\frac{1}{K+1}$.

MRR has a few key characteristics:

- **Sensitivity to First Relevant Item:** *MRR* is particularly useful when you're interested in finding the best possible relevant item. It emphasizes the importance of the first relevant item by taking the reciprocal of its position. If the first relevant item is ranked higher, the *MRR* score will be higher.

- **Simplicity:** *MRR* is straightforward to compute and interpret. It condenses the evaluation process to a single value, making it easy to compare different systems or models, or even interpret the *MRR* score of a single system.
- **Ranking Quality:** *MRR* provides a measure of ranking quality without considering the positions of subsequent relevant items. This can be advantageous when the focus is on the initial interaction with the user, such as in web search or recommendation scenarios.

However, *MRR* has limitations. It does not take into account the positions of subsequent relevant items beyond the first one, and it treats all relevant items as equally important. That is why, it is commonly used along other metrics as well, such as Average Precision or *NDCG*.

8.2 Training and Inference Details

8.2.1 Graph Kernel Parameters

Earlier, in Section 4.2.2, we presented the five Graph Kernels that will be used for the Experiments, as the baseline technique for tackling graph similarity. Specifically, the Python library GraKel[103], contains all the Graph Kernel implementations that were utilized for this thesis. We will now list the parameters for each of the five Graph Kernels used:

- **Shortest Path Kernel:** This Kernel doesn't receive any specific parameters that affect the computation stage, except for the underlying algorithm that finds the shortest paths. This algorithm can be set to *"Dijkstra"*, *"Floyd-Warshall"*, or *"Auto"* (decides which one is faster according to the input graphs). We set this to *"Auto"*.
- **Weisfeiler-Lehman Kernel:** Here, the two important parameters are, the number of Iterations of Weisfeiler-Lehman Test Propagation, and the underlying Graph Kernel that is applied to the transformed graphs. We set the iteration number to 20, and the underlying Graph Kernel to *Vertex Histogram*.
- **Neighborhood Hash Kernel:** The two parameters in this case are, the maximum number of Neighborhood Hash (i.e. iterations), and the Byte size of the node Hashes. The former was set to 3 and the latter to 2.
- **Random Walk Kernel:** For this Graph Kernel, the main parameter is λ, as defined in Section 4.2.2, which was set to 0.1.
- **Graphlet Sampling Kernel:** For this Kernel, the main parameter, is the maximum size of Graphlets that will be used. We set this to 5.

8.2.2 GNN Details and Hyperparameters

Transductive Inference

Here we will explain an important aspect of the Information Retrieval task, regarding the training and testing data. To do that, we must first clarify the differences between two fundamental concepts in machine learning: **Transductive** and **Inductive** learning/inference:

- **Inductive Learning** is the more popular and well-understood method of training and evaluating a machine learning. It involves training a model on a Training Set, possibly using a Validation Set to avoid overfitting, and then testing the model's performance on the un-seen Test Set. This is the standard pipeline in order to create a machine learning model. The main goal here, is to create a model that can *induct*, i.e. inferring general rules from specific data. This is basically the same as saying *"we want the model to generalize well"*, therefore making sure that it hasn't just memorized the training data, but

instead, it has learnt a meaningful function that can be applied to the underlying distribution of the data.

- **Transductive Learning** is an approach where the primary goal is to make predictions specifically for a given set of test instances without the intention of formulating generalized rules or principles. In transductive learning, the model seeks to optimize its predictions based on the context and characteristics of the test data itself, often tailoring its decisions to those specific instances. For example, in a transductive learning scenario, a machine learning model may be trained to classify a set of medical images into two categories: benign and malignant tumors. However, instead of aiming to establish general principles of tumor classification, the model's primary focus is to make highly accurate predictions for the specific set of test images at hand, adapting its decision boundary based on the unique features of these images.

Transductive Learning is especially popular in the field of Machine Learning on Graphs, because most of the traditional problems fall into this category. For example, the *Node Classification* task of a single graph structure, is a transductive task. This is due to the fact, that the GNN model doesn't need to generalize or make predictions about unseen data, because all the data, that we want to use the model for, is present in the Training Set. Similarly, the task of *Link Prediction* is also a transductive task, because the graph that we want to predict its links, is available at the training phase.

That is why, some of the fundamental concepts of Inductive machine learning, don't apply in the case of Transductive tasks. Some of them are: *overfitting*, *generalization*, *validation* and many more.

We formalize the Graph Similarity problem, that we will tackle in this thesis, as a **Transductive Task**, in order to comply with the more popular tasks of Machine Learning on Graphs, as mentioned earlier. So in practice, the GNN models that we use, are trained on both the *Query Set* and *Answer Set*, and the goal is to learn the representations for these graphs as optimally and efficient as possible.

The extension of this task to an Inductive one, is also possible, but not covered within the scope of this thesis, and is left as Future Work.

GNN Hyperparameters

For the Hyperparameters of the GNN models, we will first list the Training/Inference decisions that were the same across all model variations. First of all, the optimizer used for the Training of the models is **AdamW**[70]. AdamW is a refined version of the well-known Adam optimization algorithm, whose main idea is *adaptive moments*. Specifically, AdamW decouples the "weight decay" process from the optimizations steps, therefore implementing a more efficient, and theoretically correct, version of L_2 regularization. Regarding the parameters, we used the default settings: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $weight\_decay = 0.01$.

It should be noted, that every GNN model that uses a Discriminator, initializes two separate optimizers, both with the AdamW algorithm, and the parameters mentioned above. The extra optimizer, is used for the loss computed by the Discriminator, and it is used to update only the parameters of the Discriminator.

As for the Learning Rates, every GNN model that uses a Discriminator, has an initial Learning Rate value of 0.001, for both optimizers. For the rest of the models, the initial Learning Rate is set to 0.01, and the **Reduce_LR_On_Plateau**[1] is also used alongside the optimizer.

For every model the **Batch Size** is set to 16. This is a setting that is applied to the data before the training of the models takes place. The Batching technique in *graph data* is quite different than with structured data (such as tabular or images), so we will briefly explain this procedure. Since there is no fixed-size data structure to store the graphs (in order to stack them, and create batches), other techniques are used to implement graph Batches. Specifically, the *PyTorch Geometric* library, combines all the graphs within a Batch, into a new unique graph. The key idea here, is that all the initial graphs, will still be isolated to the rest of the graphs in the batch, so the *Message-Passing* technique will yield the same results, but this time it will compute everything as if it is one sample.

[1] Implementation from the *PyTorch* library found here

For the final Inference of the models, as explained in Section 7.2, each graph of the 1000 present in the datasets, is passed through the Encoder, and the final Graph Embedding is the result of adding the Node Embeddings. After that, the distance that we selected for the comparison of Graph Embeddings is the $\mathbf{L_2}$ ***Distance***. The other one that we considered was *Cosine Distance*, but it ended up with inferior performance, when compared to the former one. An explanation to that could be the Feature Decoder and the use of *Mean Squared Error* loss function, which is much closer to the Euclidean distance of the Features, rather than the Cosine distance.

Finally, the Hyperparameters of **Epochs**, **Embedding Size** and **Attention Heads**(for GAT and GATv2), were the ones that were mostly different across the models. For the Attention Heads, the values 2, 4 and 8 were the ones that provided the most promising results, and the tuning was performed on these three values. Regarding the Embedding Size, initially for the *Random Set*, we noticed that the simpler models didn't perform well when increasing the value above 32 and 64. But the more complex models, that combined all the architectures, performed really well with larger latent embeddings, even surpassing the initial size of 100. This could be due to the fact, that the more complicated models learn to embed the Nodes to a latent space of size d_1, and the final Graph embedding size will also be d_1 (sum of Node embeddings). But, intuitively, the information contained within the whole graph, is more than the information contained within a node. So it is natural that if we want to learn and store more complex information of the graph, we will probably need the final latent embedding size to be larger than the initial size of the Node Features (100). Finally, for the Epochs, we noticed that this parameter was the one with the most variety in the final results, that's why a lot of values were used for the tuning, starting from 10 up to 200, with a step of 10. Below, in Tables 8.1 and 8.2, we present all the final hyperparameters for the models with the best performance. Here, by "best performance", we used the mean value of the three metrics.

Also, it should be noted, that we didn't test every GNN architecture on the Dense dataset, since the point of this dataset, was not to compare the metrics with the Random dataset, but to see if the GNN models will out-perform the Graph Kernels, given that the graphs are more dense. That's why we only test the two best performing models (*Combined-ARVGA* and *Combined-ARVGA-MLP*), along with *VGAE* and *ARVGA* to assess the performance gain that the Adversarial training has to offer, with the denser graphs.

Model	GNN Module	Epochs	Embedding Size	Attention Heads
GAE	GCN	10	32	-
	GAT	20	32	8
	GATv2	70	32	4
	GIN	120	32	-
VGAE	GCN	10	32	-
	GAT	20	32	8
	GATv2	70	32	4
	GIN	120	32	-
Feature-VGAE	GCN	40	32	-
	GAT	170	32	4
	GATv2	50	32	4
	GIN	120	32	-
Combined	GCN	20	32	-
	GAT	170	32	4
	GATv2	160	32	8
	GIN	60	32	-
ARVGA	GCN	120	64	-
	GAT	20	32	4
	GATv2	140	32	4
	GIN	10	32	-
Combined-ARVGA	GCN	30	300	-
	GAT	190	256	4
	GATv2	150	64	8
	GIN	50	64	-
Combined-ARVGA-MLP	GCN	90	300	-
	GAT	120	300	4
	GATv2	170	300	4
	GIN	160	300	-

Table 8.1: GNN Models Optimal Hyperparameters, for Random Subset

Model	GNN Module	Epochs	Embedding Size	Attention Heads
VGAE	GCN	80	300	-
	GAT	60	300	4
	GATv2	70	300	4
	GIN	80	300	-
ARVGA	GCN	20	200	-
	GAT	140	200	4
	GATv2	40	200	4
	GIN	180	300	-
Combined-ARVGA	GCN	150	300	-
	GAT	160	300	4
	GATv2	80	300	4
	GIN	40	300	-
Combined-ARVGA-MLP	GCN	150	300	-
	GAT	130	300	4
	GATv2	160	300	4
	GIN	10	300	-

Table 8.2: GNN Model Optimal Hyperparameters, for Dense Subset

8.3 Results

8.3.1 Quantitative Analysis

Firstly, we will present the final MAP, MRR and NDCG score, achieved by the Graph Kernels and the GNN Models, on the Random Subset. Each specific GNN model, corresponds to the Optimal one, with the hyperparameters presented in Table 8.1. The Evaluation Metrics are shown in Table 8.3.

		MAP@10	MRR	NDCG@10
Shortest Path Kernel		0.4836	0.5377	0.2027
Random Walk Kernel		0.0147	0.0560	0.0012
Weisfeiler-Lehman Kernel		**0.6264**	0.7119	0.2790
Graphlet Sampling Kernel		0.1658	0.1741	0.0248
Neighborhood Hash Kernel		0.6257	**0.7141**	**0.2956**
GAE	GCN	0.5270	**0.6878**	**0.2132**
	GAT	0.4946	0.5747	0.1975
	GATv2	**0.5343**	0.6460	0.1907
	GIN	0.4244	0.4623	0.1631
VGAE	GCN	0.5302	0.6660	0.2177
	GAT	**0.5914**	**0.7061**	**0.2193**
	GATv2	0.5237	0.5859	0.1761
	GIN	0.4275	0.5120	0.1876
Feature-VGAE	GCN	0.5524	0.6567	0.2240
	GAT	0.5646	0.6425	0.1994
	GATv2	**0.5958**	**0.6864**	**0.2327**
	GIN	0.5007	0.5762	0.1953
Combined	GCN	**0.5676**	**0.7017**	0.2287
	GAT	0.5405	0.6428	0.2350
	GATv2	0.5598	0.6679	**0.2446**
	GIN	0.4335	0.4828	0.1215
ARVGA	GCN	**0.6104**	**0.7298**	**0.2287**
	GAT	0.5844	0.6680	0.2183
	GATv2	0.5436	0.6860	0.1998
	GIN	0.5364	0.5935	0.2238
Combined-ARVGA	GCN	**0.6353**	0.7455	**0.2796**
	GAT	0.6137	**0.7503**	0.2662
	GATv2	0.6219	0.7454	0.2416
	GIN	0.5320	0.6318	0.2448
Combined-ARVGA-MLP	GCN	**0.6277**	0.7162	0.2472
	GAT	0.6117	**0.7533**	0.2542
	GATv2	0.6107	0.7403	**0.2606**
	GIN	0.5683	0.6423	0.2585

Table 8.3: Final Metric Scores for the Graph Kernels and the Optimal GNN models, on the Random Subset. Bold denotes the best result for each architecture.

The first thing that we observe at the **Graph Kernels**, is the superiority of the Weisfeiler-Lehman and the Neighborhood Hash Kernel. The other three Kernels, are heavily outperformed by the former ones, on all three metrics. This can be explained if we correlate it to the main idea of each of the Kernels. Specifically, the three Kernels that didn't perform so well, they are all based on fundamental structural properties of a graph (shortest paths, random walks), and on the appearance of prototypes (graphlets). All three of these, are expected to not be sufficiently applicable in the case of Scene Graphs, and especially in the case of the Random Subset. If we recall from Section 8.1.1, the order of the graphs in the Random Subset exhibits significant variation. This variation, along with their generally low density, doesn't provide useful information for the heavily structure-based Graph Kernel. 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 seems to prove a successful idea, when it comes to taking into consideration as much graph information as possible, in order to compare pairs of Graphs, and extract a similarity scores. We can therefore, acknowledge the expressive power of the **Message-Passing** technique, because the two kernels that achieved the best results, basically implement a variation of that method.

Regarding the **GNN models**, the initial observation is that the simple architectures of **GAE** and **VGAE**, are not sufficient enough to out-perform the baseline Graph Kernels, on any of the three metrics. Only the results for MRR are comparable, where **VGAE-GAT** achieved 1% lower score. The other two metrics are 3% and 7% lower than the Graph Kernels. Also at this point, it is worth mentioning, that we didn't include any other architectures without the *Variational Loss* at the latent space, because it was quickly evident, that it didn't produce better results.

Moving on to more complex architectures, we see that the **Feature-VGAE** and the **Combined** architecture (which are the first ones that implement a Feature decoder as well), already provide a boost across all three metrics. Specifically, looking at NDCG (the most representative metric among the three), both are higher when compared to the two initial models. Even compared to **ARVGA**, we can see that the models with the Feature-Decoder score better at NDCG.

The biggest performance boost is given at the final two architectures, where the Adversarial Training from ARVGA, is combined with a Feature Decoder and a GNN-based Edge Decoder. The two **Combined-ARVGA** architectures have the best scores among the GNNs, across all three metrics. Specifically, the GCN version of Combined-ARVGA scores the best NDCG and MAP across all the GNN models.

		MAP@10	MRR	NDCG@10
Shortest Path Kernel		0.3086	0.3747	0.0852
Random Walk Kernel		0.0532	0.0693	0.0177
Weisfeiler-Lehman Kernel		0.3372	0.4011	0.1102
Graphlet Sampling Kernel		0.1468	0.1770	0.0374
Neighborhood Hash Kernel		**0.4449**	**0.5440**	**0.1568**
VGAE	GCN	0.4941	0.5873	0.1989
	GAT	0.4972	0.5736	0.1851
	GATv2	0.5020	0.6046	**0.2030**
	GIN	**0.5053**	**0.6067**	0.1914
ARVGA	GCN	0.4678	0.5554	0.1757
	GAT	0.4813	0.5540	0.1744
	GATv2	0.4956	0.5754	0.1911
	GIN	**0.5247**	**0.6325**	**0.2015**
Combined-ARVGA	GCN	0.4828	0.5783	0.1907
	GAT	0.4757	0.6060	0.2236
	GATv2	0.5064	0.5915	0.2156
	GIN	**0.5441**	**0.6067**	**0.2254**
Combined-ARVGA-MLP	GCN	0.5078	0.5874	0.1910
	GAT	0.5250	0.6115	0.2326
	GATv2	**0.5591**	**0.6580**	**0.2438**
	GIN	0.5099	0.5947	0.2299

Table 8.4: Final Metric Scores for the Graph Kernels and the Optimal GNN models, on the Dense Subset. Bold denotes the best result for each architecture.

But we can still see, that the Graph Kernels are still better, when it comes to NDCG. Specifically, the **Neighborhood Hash** Kernel scores about 2% more on NDCG than the best GNN model, but it scores less on MAP and MRR. If we recall, MAP's interpretation, is that it tells us how many relevant objects we have in the predicted ranking, and MRR only takes into account the position of the first relevant object retrieved. So if we look at those two specific aspects of Information Retrieval, then the more complex GNN models are better than the Kernels. But on the other hand, NDCG is the only metric that, basically, takes into account everything from the predicted ranking (the position of the objects, the relevance score of the objects, if an object is relevant or not etc.). So if we are interested in an all-around better Information Retrieval system,

then the **Neighborhood Hash** Graph Kernel is still the better option.

As for the specific GNN modules, the most surprising result, is the under-performance of the **GIN** variants, almost in every architecture. GIN was by far the model with the best theoretical foundation among the four, so it comes as a surprise to score lower than the other, on almost all cases. The one that was generally the best-performing was **GCN**, with **GATv2** also scoring equally high as well.

The results from these first experiments is what led us to conduct experiments on more dense graphs, and assess the GNNs' behavior on these as well. As already mentioned, we didn't test all the possible architectures again on the Dense Set, only the most fundamental and the most promising ones: VGAE, ARVGA, Combined-ARVGA, Combined-ARVGA-MLP. The results on the **Dense Set** are shown in Table 8.4.

		Train Time	**Inference Time**
Shortest Path Kernel		-	117 ms
Random Walk Kernel		-	16.5 sec
Weisfeiler-Lehman Kernel		-	338 ms
Graphlet Sampling Kernel		-	3.51 sec
Neighborhood Hash Kernel		-	263 ms
VGAE	GCN	1min 3sec	680 ms
	GAT	54.1 sec	836 ms
	GATv2	1min 13sec	899 ms
	GIN	56 sec	275 ms
ARVGA	GCN	30 sec	649 ms
	GAT	3min 41sec	855 ms
	GATv2	1min 7sec	905 ms
	GIN	4min 35sec	299 ms
Combined-ARVGA	GCN	4min 30sec	723 ms
	GAT	5min 23sec	888 ms
	GATv2	3 min	971 ms
	GIN	1min 6sec	320 ms
Combined-ARVGA-MLP	GCN	4min 29sec	697 ms
	GAT	4min 16sec	916 ms
	GATv2	5min 48sec	964 ms
	GIN	16 sec	327 ms

Table 8.5: Train/Inference Times of Graph Kernels and GNN models, on Dense Subset

The major difference here, is that GNN models out-perform all the Graph Kernels, across all three metrics. And they achieve that, even from the first architecture, the simple **VGAE**. And the best GNN model, the **GATv2** variation of the **Combined-ARVGA-MLP** achieves 11.5% higher MAP, 11% higher MRR and 9% higher NDCG. So we can safely say, that it is an all-around much better Retrieval system than the Graph Kernels for this case. Also, we notice that in both cases (Random and Dense graphs), the module that was most assisted by the MLP addition to the Combined-ARVGA architecture, is the GATv2.

This also a definite proof, of the importance of the Dataset quality for all the Machine Learning models. Particularly for the GNNs, it is evident that the **Density** of the training graphs, plays an important role, because the Message-Passing method can't be efficiently utilized when there are numerous isolated nodes. Also the constraint of the graph size also played an important role, as the final Graph Embeddings are more sufficiently comparable, when the graph sizes are of the same order.

Another major difference are the scores of the **GIN** variants. In this case, they are the ones that produce the best results in most cases, while they performed quite poorly in the case of the Random Set. This is another evidence that supports the claim we made above, that the Density of the graphs plays a key role in allowing them to utilize their expressive power, and learn meaningful features.

We will also provide the runtimes for the Graph Kernels, as well as the Train and Test times for the GNN models, in Table 8.5. The only important thing to notice here, is that the GNN models' Training, takes from a few seconds, up to a few minutes. So the trade-off, between the runtime and results of the Kernels, and

the Train time and results of the GNNs, is definitely worth it, especially in the case of the Dense Subset. We also shouldn't forget, that the biggest advantage of GNNs over Graph Kernels, is that they provide general-purpose Embeddings, that can be used for any downstream task.

In conclusion, we can see that the GNNs prove to be a really good solution, when it comes to the **Graph Similarity** task, especially if we have a reassurance of the quality of the graphs at our disposal. In terms of Train time, they also compete with all the Supervised Similarity frameworks, which would take a much longer time to train, because the use pairs of graphs as samples, and not just graphs (quadratically increasing the amount of training samples).

8.3.2 Qualitative Analysis

For the Qualitative assessment of the frameworks, we will utilize the images that correspond to the Scene Graphs, retrieved by the GNNs and Kernels. We will showcase Query Images, and the first few retrieved Images, both for successful and unsuccessful cases. It is important to remember here, that **all the frameworks mentioned and tested, didn't have access to the Images, only to corresponding Scene Graphs**. This obviously leads to cases where the Images are not alike, but the Graphs' similarity leads the model to rank them high. Below, we present pairs of Query-Answers, exemplifying the scenarios explained above.

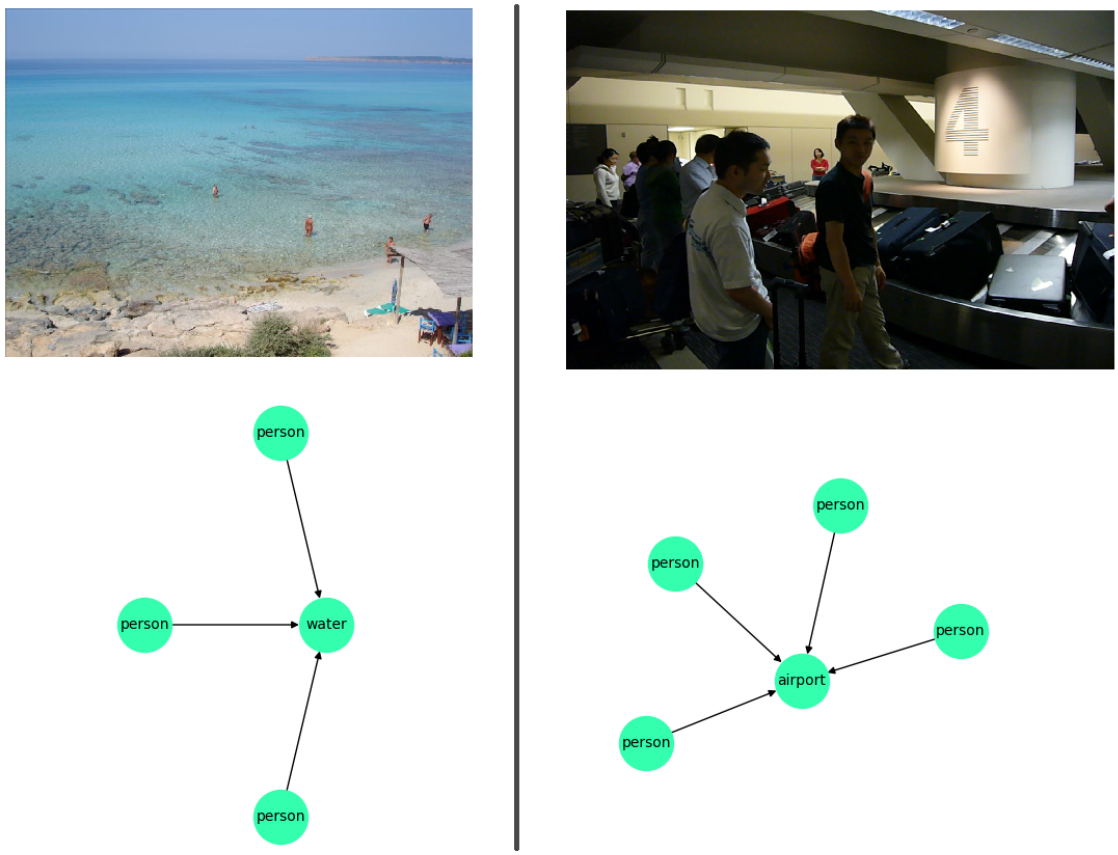


Figure 8.3.1: Example where the query (left) and the first retrieved object (right) according to the best GNN model, have structurally similar Scene Graphs, but quite dissimilar Images

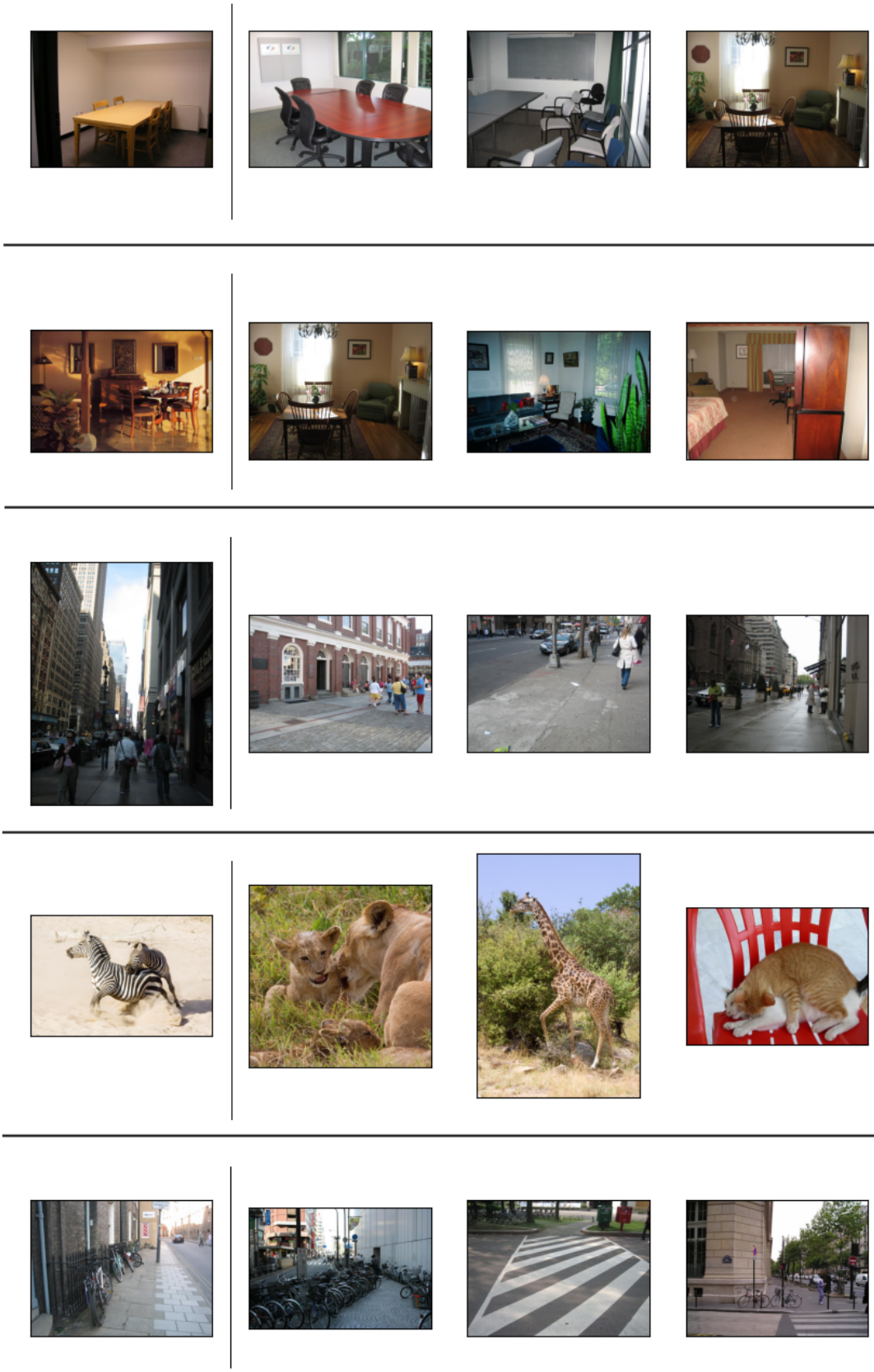


Figure 8.3.2: Examples of different Queries (left), with the top-3 results (right), as retrieved by the best GNN model

Figure 8.3.3: Two example queries, where the top-2 retrieved objects by the best GNN model (top) are visually closer to the query image (left), compared to the top-2 retrieved objects by the best Graph Kernel (bottom)

Chapter 9

Conclusion

9.1 Reflecting on the Findings

In this thesis, we tackled the problem of Graph Similarity, under the purview of constructing **Information Retrieval** systems. Specifically, we considered the Ground Truth to be the similarity computed by the *Graph Edit Distance* (GED) algorithm, whose goal is to transform a given graph, into an isomorphic to another graph. This has been proved to be NP-Complete, so various approximate variations of GED are commonly used. Here, we use the *Bipartite Matching* variation, that approximates the optimal graph edit distance, by taking into consideration mostly the node information, and the edits attributed to them. For the insertion/deletion/substitution costs of the scene graphs, we incorporate the Concept Hierarchy provided by *WordNet*, along with "path similarity" algorithms applied on this hierarchy. Even though variations of GED provide us with the most accurate results, the classical technique for tackling Graph Similarity is through the use of **Graph Kernels**. Their diminished computational complexity, capacity for result interpretability, and sustained high performance have positioned them as an enticing choice for many years now. We choose to evaluate five popular Graph Kernels, three of which are based on fundamental Graph metrics, and the other two rely on information propagation.

The key point point of this work, is to analyze several **Unsupervised Graph Autoencoder** architectures, in order to thoroughly assess and compare their performance to the aforementioned Graph Kernels. The GNNs that we propose and experiment with, consist of several building blocks, each focusing on a different aspect of *representation learning*. Specifically, the final model that propose, the **Combined-ARVGA**, is based on three key components:

- The initial Graph Autoencoder, along with the **Inner-Product Decoder**, responsible for the structural reconstruction of the graph
- The **Feature Decoder**, responsible for the Node Information reconstruction of the graph
- The **Discriminator**, responsible for regularizing the latent embeddings produced by the encoder, through the process of adversarial training.

For the actual GNN modules used in the final Graph Autoencoders, we test four of the most popular ones: *GCN*, *GAT*, *GATv2* and *GIN*. Regarding the input graphs, we utilize the **GloVe word embeddings**, and we remove the attributes and edge information from the original scene graphs, further simplifying their structure. We also acquire two subsets of Visual Genome, the **Random** Subset and the **Dense** subset, in order to measure the impact of the dataset's quality, on the final performance of the models. For the evaluation process, we used three of the most widely used metrics for assessing Information Retrieval systems: **MAP** (measuring the *Precision* of the final ranking), **MRR** (measuring the position of the first retrieved relevant object) and **NDCG** (the all-around order-aware metric, that takes into account positions, relevance scores).

From the experiments on the Random Subset, we can conclude that Combined-ARVGA was the best performing architecture, with GCN and GATv2 being the most competent GNN modules. By conducting an

ablation study on the several architectures, we argue that the use of a **Feature Decoder** alongside an **Adversarial Regularization** method, play a paramount role in learning meaningful representations for graph nodes, which can be easily extended to producing Edge and Graph Embeddings as well. As for the Graph Kernels, the expressive power of the Information Propagation method is obvious in the final result, as the Weisfeiler-Lehman and Neighborhood-Hash Graph Kernels, outperformed the other kernels by a substantial margin. When comparing the best Kernel with the best GNN model, we find that **Combined-ARVGA GCN** scored 1% higher in MAP, 3% higher in MRR, but 1.5% lower in NDCG. The fact that GNNs couldn't score as high as Graph Kernels at NDCG, is what led us to conduct a more thorough analysis of the Random Subset that was originally used. It was found that the scene graphs had really low Density, a substantial number of Isolated Nodes, as well as a wide variety of graph sizes, making them difficult to compare. Therefore, we created a second dataset, the Dense Subset, in order to test whether the data quality can negatively affect the performance of GNN models, to the point where Graph Kernels outperform them.

This hypothesis was thoroughly validated through the additional experiments we conducted on the Dense Subset. Specifically, comparing the **Combined-ARVGA-MLP** that uses the GATv2 module to the best-performing Graph Kernel, the GNN scored 11% higher on MAP, 11% higher on MRR, and 9% higher on NDCG. So we can safely conclude that Graph Autoencoders can learn meaningful representations of Graphs, and use them to out-perform Graph Kernels on the task of Graph Similarity, given that the quality of the dataset has been analyzed and validated. Besides that, the major advantage of such Graph Autoencoders, is that the final Embeddings have not been trained on any downstream task, but only for general-purpose Representation Learning. This allows us to utilize these trained models, and perform traditional Node-Level, Edge-Level or Graph-Level tasks, on the graphs that we have at our disposal.

Finally, as mentioned in Section 6.2.3, the ultimate goal for training these GNN models and evaluating them on the Information Retrieval task, is mainly to implement them within a Counterfactual-Explanations Framework (such as the one proposed in [21]), which utilizes external knowledge that provides explanations for the samples of an arbitrary dataset. If this external knowledge is formulated in graph structures, then GNN models, similar to the ones analyzed in this thesis, can be utilized to provide better performance and lower computation times.

9.2 Future Research

In the pursuit of advancing the field of computing Graph Similarity using Graph Autoencoders, this thesis has laid a foundational framework and provided valuable insights into the effectiveness and potential of such models in Information Retrieval problems. However, as with any scientific endeavor, the exploration of this domain is far from exhaustive, and numerous uncharted avenues beckon for further investigation. In this chapter, we delve into the promising directions and untapped potential for future research, aiming to build upon the findings and methodologies presented in this work. Specifically, the key areas of research that hold particular interest are the following:

- First and foremost, whether the performance of these GNN models in an **Inductive Information Retrieval** task is competent, is the most important follow-up question after the work presented here. We formulated the problem as a Transductive one, and proved the superiority of Graph Autoencoders to Graph Kernels, but there need to be thorough experimentation and evaluation to make the same claim for Inductive inference.
- The use of **Relational GNN Modules**, should also be considered, because they allow for a much more expressive representation for the Edges of the dataset. Specifically, Relational GNNs, formulate the input graph as a *heterogeneous* graph, where there are numerous Edge Types, each with a unique adjacency matrix. The utilization of such models, could allow us to change the preprocessing of the graphs, and take advantage of the information on the Edges.
- Autoencoders in general, have been extensively used in **Conditional Generation tasks**, and Graph Autoencoders are no exception. Even though we didn't delve into the Generation capabilities of Graph Autoencoders in this thesis, it could prove to be a powerful tool, especially in the context of Explainability, where many proposed frameworks' goal is to generate Counterfactual Explanations.

- Within the scope of the thesis, we only experimented with the most popular and fundamental **GNN Modules**, which are the GCN, GAT, GATv2 and GIN. Several more refined versions have been proposed[32, 15], that try to tackle the problem of oversmoothing when stacking many layers. The use of these GNN Modules, could allow for scalability of the models proposed here, in order to achieve better performance in Representation Learning tasks.

Chapter 10

Bibliography

[1] Bach, F. *Graph kernels between point clouds*. 2007. arXiv: 0712.3402 [cs.LG].

[2] Bhatti, U. et al. "Deep Learning with Graph Convolutional Networks: An Overview and Latest Applications in Computational Intelligence". In: *International Journal of Intelligent Systems* 2023 (Feb. 2023). DOI: 10.1155/2023/8342104.

[3] Bird, S., Klein, E., and Loper, E. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[4] Borgwardt, K. M. et al. "Protein function prediction via graph kernels". In: *Bioinformatics* 21.Suppl 1 (June 2005), pp. i47–i56. DOI: 10.1093/bioinformatics/bti1007. URL:

[5] 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.

[6] BORGWARDT, K. M. et al. "GRAPH KERNELS FOR DISEASE OUTCOME PREDICTION FROM PROTEIN-PROTEIN INTERACTION NETWORKS". In: *Biocomputing 2007*. WORLD SCIENTIFIC, Dec. 2006. DOI: 10.1142/9789812772435_0002. URL:

[7] Boser, B. E., Guyon, I. M., and Vapnik, V. N. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL:

[8] Brody, S., Alon, U., and Yahav, E. "How Attentive are Graph Attention Networks?" In: *International Conference on Learning Representations*. 2022. URL:

[9] Brughmans, D., Leyman, P., and Martens, D. "Nice: an algorithm for nearest instance counterfactual explanations". In: *Data Mining and Knowledge Discovery* (2023), pp. 1–39.

[10] Bruna, J. et al. *Spectral Networks and Locally Connected Networks on Graphs*. 2014. arXiv: 1312.6203 [cs.LG].

[11] Cao, Z. and Schmid, N. "Heterogeneous sharpness for cross-spectral face recognition". In: May 2017, 102020Q. DOI: 10.1117/12.2261984.

[12] Ceroni, A., Costa, F., and Frasconi, P. "Classification of small molecules by two- and three-dimensional decomposition kernels". In: *Bioinformatics* 23.16 (June 2007), pp. 2038–2045. DOI: 10.1093/bioinformatics/btm298. URL:

[13] Chang, X. et al. "A Comprehensive Survey of Scene Graphs: Generation and Application". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (Jan. 2023), pp. 1–26. DOI: 10.1109/tpami.2021.3137605. URL:

[14] Chen, L. et al. "Graph Edit Distance Reward: Learning to Edit Scene Graph". In: (2020). arXiv: 2008.06651 [cs.CV].

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

[16] Chen, S. et al. "Discrete Signal Processing on Graphs: Sampling Theory". In: *IEEE Transactions on Signal Processing* 63.24 (Dec. 2015), pp. 6510–6523. DOI: 10.1109/tsp.2015.2469645. URL:

[17] Cheung, M. et al. "Graph Signal Processing and Deep Learning: Convolution, Pooling, and Topology". In: (July 2020).

[18] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: (2016). arXiv: 1511.07289 [cs.LG].
[19] Cortes, C. and Vapnik, V. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/bf00994018. URL:
[20] Defferrard, M., Bresson, X., and Vandergheynst, P. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG].
[21] Dervakos, E. et al. "Choose your Data Wisely: A Framework for Semantic Counterfactuals". In: (2023). arXiv: 2305.17667 [cs.AI].
[22] Dhurandhar, A. et al. *Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives*. 2018. arXiv: 1802.07623 [cs.AI].
[23] Dosovitskiy, A. et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: (2021). arXiv: 2010.11929 [cs.CV].
[24] Dutta, S. et al. "Data-driven reduced order modeling of environmental hydrodynamics using deep autoencoders and neural ODEs". In: (July 2021).
[25] Eldawy, E. O. et al. "FraudMove: Fraud Drivers Discovery Using Real-Time Trajectory Outlier Detection". In: *International Journal of Geo-Information* 10 (Nov. 2021). DOI: 10.3390/ijgi10110767.
[26] Esmail, W., Stockmanns, T., and Ritman, J. "Machine Learning for Track Finding at PANDA". In: (Oct. 2019).
[27] Fankhauser, S., Riesen, K., and Bunke, H. "Speeding Up Graph Edit Distance Computation through Fast Bipartite Matching". In: (2011), pp. 102–111. DOI: 10.1007/978-3-642-20844-7_11. URL:
[28] Fernández, R. R. et al. "Random forest explainability using counterfactual sets". In: *Information Fusion* 63 (2020), pp. 196–207.
[29] Filandrianos, G. et al. "Conceptual Edits as Counterfactual Explanations." In: *AAAI Spring Symposium: MAKE*. 2022.
[30] Fischer, A. et al. "A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance". In: (2014), pp. 83–92. DOI: 10.1007/978-3-662-44415-3_9. URL:
[31] Gärtner, T., Flach, P., and Wrobel, S. "On Graph Kernels: Hardness Results and Efficient Alternatives". In: *Learning Theory and Kernel Machines*. Springer Berlin Heidelberg, 2003, pp. 129–143. DOI: 10.1007/978-3-540-45167-9_11. URL:
[32] Gasteiger, J., Bojchevski, A., and Günnemann, S. *Predict then Propagate: Graph Neural Networks meet Personalized PageRank*. 2022. arXiv: 1810.05997 [cs.LG].
[33] Gaüzère, B., Brun, L., and Villemin, D. "Two New Graphs Kernels in Chemoinformatics". In: *Pattern Recognition Letters* 33 (Nov. 2012). DOI: 10.1016/j.patrec.2012.03.020.
[34] Ghosh, S. et al. "Generating Natural Language Explanations for Visual Question Answering using Scene Graphs and Visual Attention". In: (2019). arXiv: 1902.05715 [cs.CL].
[35] Gilmer, J. et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].
[36] Grauman, K. and Darrell, T. "The Pyramid Match Kernel: Efficient Learning with Sets of Features". In: *Journal of Machine Learning Research* 8.26 (2007), pp. 725–760. URL:
[37] Grover, A. and Leskovec, J. "node2vec: Scalable Feature Learning for Networks". In: (2016). arXiv: 1607.00653 [cs.SI].
[38] Guidotti, R. "Counterfactual explanations and how to find them: literature review and benchmarking". In: *Data Mining and Knowledge Discovery* (Apr. 2022). DOI: 10.1007/s10618-022-00831-6. URL:
[39] Guidotti, R. et al. "Factual and counterfactual explanations for black box decision making". In: *IEEE Intelligent Systems* 34.6 (2019), pp. 14–23.
[40] Hamilton, W. L., Ying, R., and Leskovec, J. "Inductive Representation Learning on Large Graphs". In: (2018). arXiv: 1706.02216 [cs.SI].
[41] Harchaoui, Z. and Bach, F. "Image Classification with Segmentation Graph Kernels". In: July 2007, pp. 1–8. ISBN: 1-4244-1180-7. DOI: 10.1109/CVPR.2007.383049.
[42] Hasibi, R. and Michoel, T. "A Graph Feature Auto-Encoder for the prediction of unobserved node features on biological networks". In: *BMC Bioinformatics* 22.1 (Oct. 2021). DOI: 10.1186/s12859-021-04447-3. URL:
[43] Herzig, R. et al. "Learning Canonical Representations for Scene Graph to Image Generation". In: (2020). arXiv: 1912.07414 [cs.CV].
[44] 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.

[45] Hochreiter, S. and Schmidhuber, J. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.
[46] Jiexia, Y. et al. "How to Build a Graph-Based Deep Learning Architecture in Traffic Domain: A Survey". In: (May 2020).
[47] Johnson, J., Gupta, A., and Fei-Fei, L. "Image Generation from Scene Graphs". In: (2018). arXiv: 1804.01622 [cs.CV].
[48] Johnson, J. et al. "Image Retrieval Using Scene Graphs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.
[49] Jonker, R. and Volgenant, A. "A shortest augmenting path algorithm for dense and sparse linear assignment problems". In: *Computing* 38.4 (Dec. 1987), pp. 325–340. DOI: 10.1007/bf02278710. URL:
[50] Karimi, A.-H. et al. *Model-Agnostic Counterfactual Explanations for Consequential Decisions*. 2020. arXiv: 1905.11190 [cs.LG].
[51] Karp, R. M. "An algorithm to solve the m × n assignment problem in expected timeO(mn logn)". In: *Networks* 10.2 (1980), pp. 143–152. DOI: 10.1002/net.3230100205. URL:
[52] Keane, M. T. and Smyth, B. "Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI)". In: *Case-Based Reasoning Research and Development: 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8–12, 2020, Proceedings 28*. Springer. 2020, pp. 163–178.
[53] Kingma, D. P. and Welling, M. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].
[54] Kipf, T. N. and Welling, M. *Variational Graph Auto-Encoders*. 2016. arXiv: 1611.07308 [stat.ML].
[55] Kipf, T. N. and Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].
[56] Klambauer, G. et al. "Self-Normalizing Neural Networks". In: (2017). arXiv: 1706.02515 [cs.LG].
[57] Kriege, N. and Mutzel, P. "Subgraph Matching Kernels for Attributed Graphs". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 291–298. ISBN: 9781450312851.
[58] Krishna, R. et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV].
[59] Kumaran, A. and Haritsa, J. "Multilingual Semantic Matching Operator in SQL". In: (Sept. 2023).
[60] Lash, M. T. et al. "Generalized Inverse Classification". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, June 2017, pp. 162–170. DOI: 10.1137/1.9781611974973.19. URL:
[61] Laugel, T. et al. *Inverse Classification for Comparison-based Interpretability in Machine Learning*. 2017. arXiv: 1712.08443 [stat.ML].
[62] Lecun, Y. et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.
[63] Levie, R. et al. *CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters*. 2018. arXiv: 1705.07664 [cs.LG].
[64] Li, J. et al. "LogKernel: A Threat Hunting Approach Based on Behaviour Provenance Graph and Graph Kernel Clustering". In: *Security and Communication Networks* 2022 (Sept. 2022), pp. 1–16. DOI: 10.1155/2022/4577141.
[65] Li, J. et al. "Predicting User Activity Intensity Using Geographic Interactions Based on Social Media Check-In Data". In: *ISPRS International Journal of Geo-Information* 10 (Aug. 2021), p. 555. DOI: 10.3390/ijgi10080555.
[66] Li, Q., Han, Z., and Wu, X.-M. *Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning*. 2018. arXiv: 1801.07606 [cs.LG].
[67] Liao, J., Yang, C., and Yang, H. "Evaluation of Aircraft Environmental Control System Order Degree and Component Centrality". In: *Aerospace* 10 (May 2023), p. 438. DOI: 10.3390/aerospace10050438.
[68] Lin, T.-Y. et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].
[69] Looveren, A. V. and Klaise, J. *Interpretable Counterfactual Explanations Guided by Prototypes*. 2020. arXiv: 1907.02584 [cs.LG].
[70] Loshchilov, I. and Hutter, F. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[71] Mahboubi, A., Brun, L., and Dupé, F.-X. "Object classification based on graph kernels". In: *2010 International Conference on High Performance Computing & Simulation*. 2010, pp. 385–389. DOI: 10.1109/HPCS.2010.5547109.

[72] Mahé, P. and Vert, J.-P. "Graph kernels based on tree patterns for molecules". In: *Machine Learning* 75 (Oct. 2006). DOI: 10.1007/s10994-008-5086-2.

[73] Mahé, P. and Vert, J.-P. "Graph kernels based on tree patterns for molecules". In: *Machine Learning* 75.1 (Oct. 2008), pp. 3–35. DOI: 10.1007/s10994-008-5086-2. URL:

[74] Mansour, M., Martens, J., and Blankenbach, J. "Hierarchical SVM for Semantic Segmentation of 3D Point Clouds for Infrastructure Scenes". In: (June 2023). DOI: 10.20944/preprints202306.2255.v1.

[75] Martens, D. and Provost, F. "Explaining Data-Driven Document Classifications". In: *MIS Q.* 38.1 (Mar. 2014), pp. 73–100. ISSN: 0276-7783. DOI: 10.25300/MISQ/2014/38.1.04. URL:

[76] Merenda, M., Porcaro, C., and Iero, D. "Edge Machine Learning for AI-Enabled IoT Devices: A Review". In: *Sensors* 20 (Apr. 2020), p. 2533. DOI: 10.3390/s20092533.

[77] Miller, G. A. "WordNet: A Lexical Database for English". In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL:

[78] Mittal, G. et al. "Interactive Image Generation Using Scene Graphs". In: (2019). arXiv: 1905.03743 [cs.CV].

[79] Mnih, V. et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. DOI: 10.1038/nature14236. URL:

[80] Mothilal, R. K., Sharma, A., and Tan, C. "Explaining machine learning classifiers through diverse counterfactual explanations". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. ACM, Jan. 2020. DOI: 10.1145/3351095.3372850. URL:

[81] Neuhaus, M., Riesen, K., and Bunke, H. "Fast Suboptimal Algorithms for the Computation of Graph Edit Distance". In: (2006), pp. 163–172. DOI: 10.1007/11815921_17. URL:

[82] Neumann, M. et al. "Graph Kernels for Object Category Prediction in Task-Dependent Robot Grasping". In: *Mining and Learning with Graphs*. 2013. URL:

[83] Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. "Graph Kernels: A Survey". In: *Journal of Artificial Intelligence Research* 72 (Nov. 2021), pp. 943–1027. DOI: 10.1613/jair.1.13225. URL:

[84] Pan, S. et al. *Adversarially Regularized Graph Autoencoder for Graph Embedding*. 2019. arXiv: 1802.04407 [cs.LG].

[85] Park, J. and Lim, S. "LEHAN: Link-feature Enhanced Heterogeneous Graph Attention Network". In: *IEEE Access* 10 (Jan. 2022), pp. 1–1. DOI: 10.1109/ACCESS.2022.3198941.

[86] Pennington, J., Socher, R., and Manning, C. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL:

[87] Poyiadzi, R. et al. "FACE: feasible and actionable counterfactual explanations". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020, pp. 344–350.

[88] Pržulj, N. "Biological network comparison using graphlet degree distribution". In: *Bioinformatics* 23.2 (Jan. 2007), e177–e183. DOI: 10.1093/bioinformatics/btl301. URL:

[89] Qi, M., Wang, Y., and Li, A. "Online Cross-Modal Scene Retrieval by Binary Representation and Semantic Graph". In: *Proceedings of the 25th ACM International Conference on Multimedia*. MM '17. Mountain View, California, USA: Association for Computing Machinery, 2017, pp. 744–752. ISBN: 9781450349062. DOI: 10.1145/3123266.3123311. URL:

[90] Quinlan, J. R. "Induction of Decision Trees". In: *Machine Learning* 1 (1986), pp. 81–106.

[91] Ralaivola, L. et al. "Graph kernels for chemical informatics". In: *Neural Networks* 18.8 (2005). Neural Networks and Kernel Methods for Structured Domains, pp. 1093–1110. ISSN: 0893-6080. URL:

[92] Riesen, K., Fankhauser, S., and Bunke, H. "Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic." In: (Jan. 2007).

[93] Sandryhaila, A. and Moura, J. M. F. "Discrete Signal Processing on Graphs". In: *IEEE Transactions on Signal Processing* 61.7 (Apr. 2013), pp. 1644–1656. DOI: 10.1109/tsp.2013.2238935. URL:

[94] Sanfeliu, A. and Fu, K.-S. "A distance measure between attributed relational graphs for pattern recognition". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3 (1983), pp. 353–362. DOI: 10.1109/TSMC.1983.6313167.

[95] Sato, K. et al. "Directed acyclic graph kernels for structural RNA analysis". In: *BMC Bioinformatics* 9.1 (July 2008). DOI: 10.1186/1471-2105-9-318. URL:

[96] Schietgat, L., Fannes, T., and Ramon, J. "Predicting Protein Function and Protein-Ligand Interaction with the 3D Neighborhood Kernel". In: Oct. 2015, pp. 221–235. ISBN: 978-3-319-24281-1. DOI: 10.1007/978-3-319-24282-8_19.

[97] Schroeder, B. and Tripathi, S. "Structured Query-Based Image Retrieval Using Scene Graphs". In: (2020). arXiv: 2005.06653 [cs.CV].

[98] Schuster, S. et al. "Generating Semantically Precise Scene Graphs from Textual Descriptions for Improved Image Retrieval". In: *Proceedings of the Fourth Workshop on Vision and Language*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 70–80. DOI: 10.18653/v1/W15-2812. URL:

[99] Shakhnarovich, G., Darrell, T., and Indyk, P. "Nearest-neighbor methods in learning and vision". In: *IEEE Trans. Neural Networks* 19.2 (2008), p. 377.

[100] 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:

[101] Shervashidze, N. et al. "Weisfeiler-Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12.77 (2011), pp. 2539–2561. URL:

[102] Shuman, D. I. et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98. DOI: 10.1109/MSP.2012.2235192.

[103] Siglidis, G. et al. *GraKeL: A Graph Kernel Library in Python*. 2020. arXiv: 1806.02193 [stat.ML].

[104] SMALTER, A., HUAN, J., and LUSHINGTON, G. "GRAPH WAVELET ALIGNMENT KERNELS FOR DRUG VIRTUAL SCREENING". In: *Journal of Bioinformatics and Computational Biology* 07.03 (June 2009), pp. 473–497. DOI: 10.1142/s0219720009004187. URL:

[105] Stumm, E. et al. "Robust Visual Place Recognition with Graph Kernels". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4535–4544. DOI: 10.1109/CVPR.2016.491.

[106] Sugiyama, M. and Borgwardt, K. "Halting in Random Walk Kernels". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL:

[107] Swamidass, S. J. et al. "Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity". In: *Bioinformatics* 21.Suppl 1 (June 2005), pp. i359–i368. DOI: 10.1093/bioinformatics/bti1055. URL:

[108] Thomee, B. et al. "YFCC100M". In: *Communications of the ACM* 59.2 (Jan. 2016), pp. 64–73. DOI: 10.1145/2812802. URL:

[109] Tolomei, G. et al. "Interpretable predictions of tree-based ensembles via actionable feature tweaking". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 465–474.

[110] Tripathi, S. et al. "Compact Scene Graphs for Layout Composition and Patch Retrieval". In: (2019). arXiv: 1904.09348 [cs.CV].

[111] Tripathi, S. et al. "Using Scene Graph Context to Improve Image Generation". In: (Jan. 2019).

[112] Vaswani, A. et al. "Attention is All you Need". In: 30 (2017). Ed. by I. Guyon et al. URL:

[113] Veličković, P. et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[114] Verdu, S. and Weinberger, M. "Inequalities for the L1 Deviation of the Empirical Distribution". In: (Aug. 2003).

[115] Vishwanathan, S. et al. "Graph Kernels". In: *Journal of Machine Learning Research* 11.40 (2010), pp. 1201–1242. URL:

[116] Waa, J. v. d. et al. "Contrastive explanations with local foil trees". In: (2018).

[117] Wachter, S., Mittelstadt, B., and Russell, C. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI].

[118] Watkins, C. J. C. H. and Dayan, P. "Q-learning". In: *Machine Learning* 8.3-4 (May 1992), pp. 279–292. DOI: 10.1007/bf00992698. URL:

[119] 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.

[120] Wu, Z. et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. DOI: 10.1109/tnnls.2020.2978386. URL:
[121] Xu, K. et al. *How Powerful are Graph Neural Networks?* 2019. arXiv: 1810.00826 [cs.LG].
[122] xu, P. et al. "A Survey of Scene Graph: Generation and Application". In: (Apr. 2020). DOI: 10.13140/RG.2.2.11161.57446.
[123] Yang, Z. et al. "Scene Graph Reasoning with Prior Visual Relationship for Visual Question Answering". In: (2019). arXiv: 1812.09681 [cs.MM].
[124] Yuan, W. et al. "A new criterion for defining the failure of a fractured rock mass slope based on the strength reduction method". In: *Geomatics, Natural Hazards and Risk* 11 (Sept. 2020), pp. 1849–1863. DOI: 10.1080/19475705.2020.1814428.
[125] Zeng, Z. et al. "Comparing Stars: On Approximating Graph Edit Distance." In: *PVLDB* 2 (Jan. 2009), pp. 25–36.
[126] Zhang, C., Chao, W.-L., and Xuan, D. "An Empirical Study on Leveraging Scene Graphs for Visual Question Answering". In: (2019). arXiv: 1907.12133 [cs.CV].
[127] Zhang, X., Solar-Lezama, A., and Singh, R. *Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections*. 2018. arXiv: 1802.07384 [cs.LG].
[128] Zhao, B. et al. "Image Generation from Layout". In: (2019). arXiv: 1811.11389 [cs.CV].
[129] Δημητρίου, A. *Scene Graph Retrieval for Counterfactual Explanation Using Graph Neural Networks*. Oct. 2022.