Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Εργαστηριο Ψηφιακης Επεξεργασιας Εικονας και Σηματων

Scene Graph Retrieval for Counterfactual Explanations Using Graph Neural Networks

Diploma Thesis

by

Angeliki Dimitriou

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Πληροφορικής
Εργαστήριο Ψηφιακής Επεξεργασίας Εικόνας και Σημάτων

Scene Graph Retrieval for Counterfactual Explanations Using Graph Neural Networks

Diploma Thesis

by

Angeliki Dimitriou

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Οκτωβρίου, 2022.

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

Αθήνα, Οκτώβριος 2022

.....

Αγγελικη Δημητριου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι Εξηγήσεις με Αντιπαράδειγμα παρέχουν αιτολογία πίσω από την απόφαση ενός μοντέλου να κάνει μια διαφορετική πρόβλεψη, προτείνοντας τις αλλαγές που πρέπει να γίνουν. Όταν το εν λόγω μοντέλο είναι ταξινομητής μαύρου-κουτιού και η είσοδος αποτελείται από εικόνες, για να απαντηθεί πώς μια οντότητα πρέπει να τροποποιηθεί με ελάχιστο τρόπο ώστε να ταξινομηθεί διαφορετικά, απαιτείται να βρεθεί η πιο παρόμοια εικόνα που ανήκει σε άλλη κατηγορία. Ένας σημασιολογικά πλούσιος τρόπος για την επίτευξη αυτού, δίνοτας ταυτόχρονα βάση στις αλληλεπιδράσεις μεταξύ των απεικονιζόμενων αντικειμένων, είναι η σύγκριση των αντίστοιχων γραφημάτων σκηνής των εικόνων, δηλαδή γραφημάτων που περιγράφουν τα αντικείμενα σε μια σκηνή και πώς αυτά σχετίζονται μεταξύ τους. Το πρόβλημα της Ομοιότητας Γραφημάτων ή της Αντιστοίχισης Γραφημάτων με Ανεκτικότητα σε Σφάλματα έχει αντιμετωπιστεί κατά τη διάρκεια των χρόνων χρησιμοποιώντας μετρικές όπως η Απόσταση Επεξεργασίας Γραφήματος (ΑΕΓ) ή μεθόδους όπως οι Πυρήνες Γράφων.

Σε αυτή τη διατριβή, προτείνουμε τη χρήση των πρόσφατα ακμάζων μοντέλων βαθιάς μάθησης που λειτουργούν ειδικά σε δεδομένα δομημένα σε γράφους, που ονομάζονται Νευρωνικά Δίκτυα Γράφων (ΝΔΓ). Παρουσιάζουμε ένα πλαίσιο ΝΔΓ που λαμβάνει ζεύγη γραφημάτων ως είσοδο και ενσωματώνει κάθε μέλος σε ένα χώρο που αντιστοιχίζει παρόμοια γραφήματα πιο κοντά με βάση τη μετρική που χρησιμοποιείται κατά την εκπαίδευση ως σήμα εποπτείας. Εκπαιδεύουμε αυτό το μοντέλο σε ένα μικρό υποσύνολο ζευγών γράφων χρησιμοποιώντας την ΑΕΓ ως ετικέτα και εξάγουμε ενσωματώσεις γράφων που μπορούν να συγκριθούν μεταξύ τους χρησιμοποιώντας απλές μετρικές όπως η ομοιότητα συνημιτόνου. Επομένως, παράγονται ταξινομήσεις παρόμοιων γραφημάτων για κάθε δείγμα του συνόλου δεδομένων και μπορεί να προσδιοριστεί η καλύτερη αντιστοίχιση. Κατά τη διάρκεια του πειραματισμού, είμαστε σε θέση να χρησιμοποιήσουμε πολλές διαφορετικές παραλλαγές συνελικτικών ΝΔΓ και να βγάλουμε σημαντικά συμπεράσματα σχετικά με την αποτελεσματικότητα και την εκφραστικότητά τους. Τα μοντέλα ΝΔΓ συγκρίνονται τόσο μεταξύ τους όσο και με μεθόδους πυρήνα γράφων και αξιολογούνται ποσοτικά, χρησιμοποιώντας έναν προσεγγιστικό αλγόριθμο ΑΕΓ ως βασική αλήθεια, και ποιοτικά με παρατήρηση αντίστοιχων εικόνων. Τα μοντέλα μας είναι σε θέση να ξεπεράσουν τις προηγουμένως χρησιμοποιούμενες μεθόδους πυρήνα και στις δύο περιπτώσεις και να παράγουν ενσωματώσεις που είναι ωφέλιμες για τη δημιουργία εξηγήσεων με αντιπαράδειγμα και δυνητικά εφαρμόσιμες σε πολλά άλλα προβλήματα.

Λέξεις-κλειδιά — Νευρωνικά Δίκτυα Γράφων, Αντιστοίχιση Γράφων με Ανοχή Λάθους, Ομοιότητα Γραφημάτων, Ανάκτηση Γραφήματος, Γράφοι Σκηνής, Εξηγήσεις με Αντιπαράδειγμα

Abstract

Counterfactual explanations provide reasoning in the form of changes needed to be made in order for a model to make a different decision. When the model in question is a black-box classifier and the input consists of images, to answer how an instance should be modified in a minimal way so as to be classified differently, one is required to find the most similar image in the other category. A semantically meaningful way to do that, while simultaneously attending to the interactions between depicted objects, is by comparing the images' corresponding scene graphs, i.e. graphs which describe object instances in a scene and how they relate to each other. The problem of Graph Similarity or Error-tolerant Graph Matching has been tackled throughout the years by measures like Graph Edit Distance (GED) or methods like Graph Kernels.

In this thesis, we propose using the recently thriving deep learning models which specifically operate on graph structured data, called Graph Neural Networks (GNN). We present a GNN framework which takes graph pairs as input and embeds each counterpart in a space which maps more similar graphs closer based on the metric used during training as a supervision signal. We train this model on a small subset of graph pairs using GED as their label and extract graph embeddings which can be compared to one another using simple metrics like cosine similarity. Therefore, rankings of similar graphs are produced for each instance in the dataset and the best match can be determined. During experimentation, we are able to utilize several different convolutional GNN variants and draw important conclusions about their effectiveness and expressivity. The GNN models are compared to each other and to graph kernel methods and evaluated both quantitatively, using an approximate GED algorithm as the ground truth, as well as qualitatively by observing corresponding images. Our models are able to outperform the previously used kernel methods in both cases and produce embeddings which are beneficial for creating counterfactual explanations and potentially applicable to many other tasks.

Keywords — Graph Neural Networks, Error-tolerant Graph Matching, Graph Similarity, Graph Retrieval, Scene Graphs, Counterfactual Explanations

Ευχαριστίες

Αυτό το έργο δεν θα ήταν δυνατό χωρίς την υποστήριξη πολλών ανθρώπων. Ευχαριστώ πολύ τον επιβλέποντα μου, κ. Στάμου Γεώργιο, για την πολύτιμη καθοδήγηση του στην εκπόνηση αυτής της εργασίας. Ευχαριστώ επίσης την Μαρία Λυμπεραίου για τη στενή συνεργασία και την υποστήριξη καθ' όλη τη διάρκεια εξερεύνησης των καινούριων αυτών αντικειμένων, καθώς και τους Γεώργιο Φιλανδριανό, Κωνσταντίνο Θωμά και Έντυ Ντερβάκο που μοιράστηκαν τη δουλειά και τις ιδέες τους μαζί μου.

Πολύ σημαντική για εμένα ήταν ακόμα η συναισθηματική συνεισφορά της οικογένειας και των φίλων μου που μου παρείχαν αγάπη, στήριξη και γέλιο. Ιδιαίτερα, θα ήθελα να ευχαριστήσω τους φίλους μου Νικόδημο, Αφροδίτη, Λευτέρη, Άννα και Ευγενία με τους οποίους μοιραστήκαμε αυτό το ακαδημαϊκό ταξίδι από την αρχή μέχρι το τέλος και βγήκαμε νικητές.

Δημητρίου Αγγελική, Οκτώβρης 2022

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό υπόβαθρο

Τα τελευταία χρόνια, η πρόοδος στην υπολογιστική ισχύ και την αποθήκευση καθώς και στους αλγόριθμους βαθιάς μάθησης έχουν οδηγήσει σε μια εντυπωσιακή διείσδυση εφαρμογών τεχνητής νοημοσύνης στην καθημερινή ζωή των ανθρώπων. Η αυξανόμενη παρέμβαση των ευφυών συστημάτων στον τρόπο με τον οποίο οι άνθρωποι καταναλώνουν μέσα και ψυχαγωγία, δημιουργούν τέχνη, αγοράζουν προϊόντα ή ακόμη και λαμβάνουν υγειονομική περίθαλψη δημιουργεί την αναπόφευκτη ανάγκη δόμησης σχέσεων εμπιστοσύνης με αυτά τα εργαλεία. Έτσι, η παροχή εξηγήσεων σχετικά με το γιατί τα συστήματα τεχνητής νοημοσύνης παίρνουν τις αποφάσεις τους έχει γίνει εξαιρετικά σημαντική.

Ο τομέας του Explainable AI στοχεύει να παρέχει πληροφορίες για τα μοντέλα «μαύρου κουτιού» της Μηχανικής Μάθησης και να εξηγήσει τον λόγο πίσω από τις ενέργειές τους. Μία από τις πιο ενδιαφέρουσες τεχνικές που χρησιμοποιούνται για αυτήν την εργασία είναι οι εξηγήσεις με αντιπαράδειγμα που παρέχουν πληροφορίες σχετικά με το πώς τα δεδομένα εισόδου θα μπορούσαν να είχαν αλλάξει προκειμένου ένα μοντέλο να λάβει διαφορετικές αποφάσεις. Αυτές οι αλλαγές συχνά επιδιώκεται να είναι ελάχιστες και μπορούν να βοηθήσουν πολύ στον εντοπισμό προτιμήσεων.

Η επικρατούσα μέθοδος για τον προσδιορισμό της ομοιότητας γραφημάτων, που αποτελεί βήμα στην δόμηση των εξηγήσεων, είναι μέσω πυρήνων γραφήματος. Σε αυτή την εργασία, θα συγκρίνουμε τόσο προσεγγίσεις πυρήνα όσο και κλασικούς αλγόριθμους που καθορίζουν την Απόσταση Επεξεργασίας Γράφων (Graph Edit Distance - GED) με νευρωνικές προσεγγίσεις. Το επίκεντρο είναι η χρήση των Νευρωνικών Δικτύων Γράφων (ΝΔΓ - GNN) για την ολοκλήρωση αυτής της εργασίας.

Αυτή η διατριβή παρέχει μια διεξοδική εξερεύνηση των παραλλαγών Νευρωνικών Δικτύων γράφων και τονίζει και συγκρίνει την εκφραστική τους δύναμη σε δεδομένα γραφημάτων. Υπογραμμίζει τη χρήση μοντέλων GNN στην εργασία της ομοιότητας γράφων, η οποία έχει ως επί το πλείστον αντιμετωπιστεί χρησιμοποιώντας μη νευρωνικές μεθόδους. Επιπλέον, προσπαθούμε να παρουσιάσουμε τον συνδυασμό GNN με δεδομένα γράφων σκηνής αλλά όχι με σκοπό τη δημιουργία τους, η οποία είναι η πιο διαδεδομένη εργασία στη σχετική βιβλιογραφία. Τέλος, μέσω της διαδικασίας εύρεσης των πιο όμοιων γράφων, είμαστε σε θέση να εξαγάγουμε ενσωματώσεις. Αυτές οι ουσιαστικές αναπαραστάσεις είναι σε θέση να συλλάβουν τη σημασιολογική ομοιότητα των δεδομένων εισόδου με βάση την απόσταση επεξεργασίας γραφήματος, στην οποία έχουν εκπαιδευτεί. Οι ενσωματώσεις μπορούν στη συνέχεια να χρησιμοποιηθούν ως αναπαραστάσεις γραφημάτων για άλλες εργασίες.

1.1.1 Γράφοι Σκηνής

Η δομή ενός γράφου εστιάζει στις σχέσεις μεταξύ οντοτήτων, καθιστώντας την έτσι ένα κατάλληλο μέσο αναπαράστασης δεδομένων σε πολλά πεδία. Σε αυτή τη διατριβή η δομή του γραφήματος θα χρησιμοποιηθεί για να αναπαραστήσει τη σκηνή που απεικονίζεται σε μια εικόνα. Αυτός ο τύπος γραφήματος, γνωστός ως γράφος σκηνής, είναι μια δομή δεδομένων που περιγράφει τα στιγμιότυπα αντικειμένων σε μια σκηνή και πώς αυτά τα αντικείμενα σχετίζονται μεταξύ τους. Είναι ένα ισχυρό εργαλείο που αναπτύχθηκε για πρώτη φορά για να βοηθήσει στον τομέα της οπτικής κατανόησης και συλλογισμού υψηλότερου επιπέδου, καθώς αντιπροσωπεύει τη σημασιολογία μιας σκηνής με απεριόριστο και λεπτομερή τρόπο.

Από μια πιο τεχνική άποψη, ένα γράφημα σκηνής είναι ένα κατευθυνόμενο γράφημα στο οποίο οι κόμβοι είναι αντικείμενα σε μια σκηνή, όπως «άνθρωπος» ή «τραπέζι», και οι ακμές είναι οι σχέσεις μεταξύ τους που συχνά περιγράφουν θέσεις ή ενέργειες. Ένα τυπικό παράδειγμα ενός γραφήματος σκηνής και της αντίστοιχης εικόνας του φαίνονται στο Σχήμα 1.1.1.

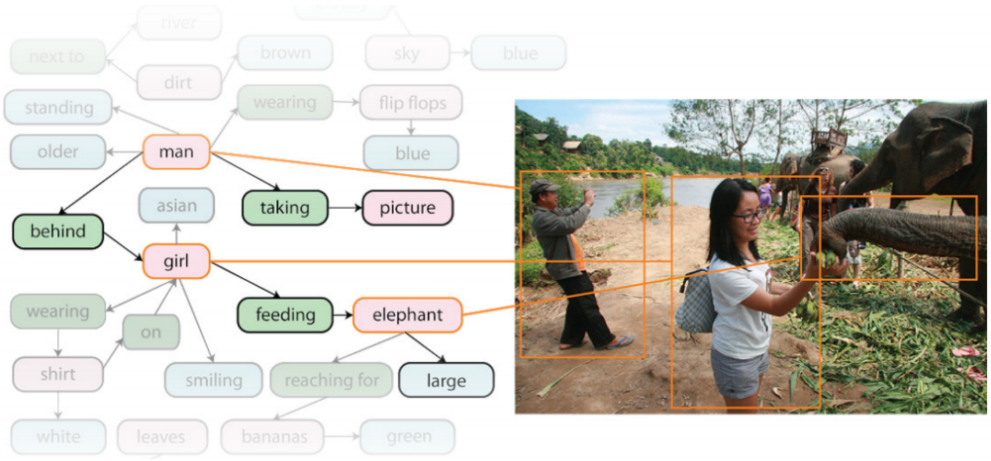


Figure 1.1.1: Γράφος Σκηνής του Visual Genome [44]

Τα γραφήματα σκηνής μπορούν να εφαρμοστούν σε μια ποικιλία εργασιών όρασης και κειμένου που σχετίζονται με την κατανόηση σκηνής, δηλαδή τη διαδικασία ανίχνευσης και ονομασίας των αντικειμένων, των ιδιοτήτων τους και περιγραφή των σχέσεών τους [12]. Η πιο σημαντική βρίσκεται στο πεδίο της Όρασης υπολογιστή και περιλαμβάνει τη δημιουργία του ίδιου του γραφήματος σκηνής χρησιμοποιώντας εικόνες και αντίστροφα.

1.1.2 Νευρωνικά Δίκτυα Γράφων

Δεδομένου ότι η δομή του γράφου αναδύεται φυσικά παντού γύρω μας, εφευρέθηκαν νευρωνικά δίκτυα που λειτουργούν απευθείας σε δεδομένα αυτού του τύπου. Τα γραφήματα είναι μη ευκλείδεια δεδομένα και επομένως τα GNN μπορούν να ομαδοποιηθούν στην ευρύτερη κατηγορία της Γεωμετρικής Μάθησης [9]. Τα Νευρωνικά Δίκτυα Γράφων (GNN) είναι γνωστά για την εκφραστική τους ισχύ και πρόσφατα κερδίζουν δημοτικότητα λόγω των αυξανόμενων δυνατοτήτων τους σε διάφορες εφαρμογές όπως τα συστήματα συστάσεων και το μοριακό δακτυλικό αποτύπωμα [108].

Τα GNN δημιουργήθηκαν γιατι οι περισσότεροι συμβατικοί αλγόριθμοι Machine ή Deep Learning είναι ειδικά κατασκευασμένοι για να καλύπτουν συγκεκριμένο τύπο δεδομένων, όπως εικόνες ή κείμενο, όχι όμως γράφους. Οι περισσότερες αναπαραστάσεις δεδομένων μπορούν να γενικευθούν σε γράφους, αλλά το αντίθετο δεν ισχύει. Στη γενική περίπτωση, τα γραφήματα είναι πιο πολύπλοκα, έχοντας έναν μη σταθερό αριθμό μη ταξινομημένων κόμβων μέσα σε γειτονιές μεταβλητού μεγέθους, και επομένως τα υπάρχοντα μοντέλα δεν μπορούν να τα χειριστούν. Επιπλέον, οι περισσότεροι κοινοί αλγόριθμοι υποθέτουν την ανεξαρτησία στιγμιοτύπων. Αυτό δεν ισχύει όταν εκτελούνται εργασίες σε επίπεδο κόμβου όπου ένα γράφημα είναι η είσοδος του νευρωνικού δικτύου και τα στιγμιότυπα είναι οι κόμβοι του. Τέλος, τα κλασικά Συνελικτικα Νευρωνικά Δίκτυα λειτουργούν σε εικόνες ή γενικότερα κανονικά πλέγματα. Η έλλειψη εντοπιότητας με την παραδοσιακή έννοια στα δεδομένα γράφων, το αυθαίρετο μέγεθος και η αμετοβλητότητα τους σε μεταθέσεις καθιστούν δύσκολη την εκτέλεση της κανονικής συνέλιξης.

Τα Νευρωνικά Δίκτυα Γράφων μπορούν να ταξινομηθούν με διάφορους τρόπους: α) ανάλογα με το επίπεδο του γράφου στο οποίο λειτουργούν σε επιπέδου κόμβου, ακμής ή γραφου, β) ανάλογα με την αρχιτεκτονική που ακολουθούν σε συνελικτικά, επαναλαμβανόμεσα, αυτοκωδικοποιητές και χωροχρονικά και γ) ανάλογα με τον τρόπο εκπαίδευσης σε επιβλεπόμενα, μη επιβλεπόμενα και μερικώς επιβλεπόμενα. Παρακάτω θα αναλύσουμε τις τρεις εκδοχές συνελικτικών δικτύων που θα χρησιμοποιηθούν στο πειραματικό μέρος.

Το **Graph Convolutional Network (GCN)** παρουσιάζει την ιδέα της χρήσης μιας προσέγγισης πρώτης τάξης του ChebNet προκειμένου να μετριαστεί η υπερπροσαρμογή. Στην πραγματικότητα, υποθέτει $K = 1$ και $\lambda_{max} = 2$. Στην ίδια κατεύθυνση το μοντέλο επιβάλλει τον περιορισμό $\theta = \theta_0 = -\theta_1$. Μετά την επιβολή αυτών των περιορισμών, η λειτουργία συνέλιξης είναι:

$$x *_G g_\theta = \theta(I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x \tag{1.1.1}$$

Αφού διαπιστώθηκε εμπειρικά ότι ο όρος $I_n + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ προκαλεί αριθμητική αστάθεια, χρησιμοποιήθηκε ένα τέχνασμα *επανακανονικοποίησης*. Ο όρος $D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = \bar{A}$ αντικαταστάθηκε από $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} = \hat{A}$ όπου $\tilde{A} = I_n + \bar{A}$ και $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Όλα τα παραπάνω μπορούν να περιγραφούν με αυτή τη συμπαγή εξίσωση:

$$H = X *_G g_\theta = f(\hat{A}X\Theta) \tag{1.1.2}$$

όπου το f είναι μια συνάρτηση ενεργοποίησης και επιτρέπονται πολλαπλές είσοδοι και έξοδοι λόγω της χρήσης πινάκων.

Το GCN είναι μια ειδική περίπτωση φασματικής προσέγγισης αφού μπορεί να εκληφθεί και ως χωρική. Στην παρακάτω εξίσωση, μπορούμε να δούμε πώς θα γίνει η συγκέντρωση πληροφοριών εντός της γειτονιάς. Σε αυτήν την περίπτωση ο ίδιος ο κόμβος θεωρείται επίσης ως γείτονάς του εαυτού του, ενός βήματος.

$$h_v = f(\Theta^T(\sum_{u \in N(u) \cup v} \hat{A}_{v,u}x_u)) \quad \forall u \in V \tag{1.1.3}$$

Αυτό το μοντέλο χρησιμοποιείται πολύ συχνά ως μέρος πιο σύνθετων αρχιτεκτονικών στη λογοτεχνία λόγω της απλότητας και της καλής πειραματικής του απόδοσης.

Το **Graph Attention Network (GAT)** [83] υιοθετεί την ιδέα της προσοχής που προτείνεται από το [82] προκειμένου να αποφασίσει ποια μέλη της γειτονιάς ενός κόμβου έχουν πιο σημαντικές πληροφορίες. Στόχος του είναι να μάθει τα σχετικά βάρη μεταξύ γειτονικών κόμβων και επομένως διαφέρει από προηγούμενες προσεγγίσεις όπως το GCN και το GraphSAGE επειδή η έννοια της γειτονιάς δεν είναι προκαθορισμένη ή πανομοιότυπη.

Η συνεκτική λειτουργία ορίζεται ως:

$$h_v^{(k)} = \sigma(\sum_{u \in N(u) \cup v} \alpha_{vu}^{(k)} W^{(k)} h_u^{(k-1)}) \tag{1.1.4}$$

όπου τα βάρη προσοχής για κάθε κόμβο v μπορούν να οριστούν ως:

$$\alpha_{vu}^{(k)} = softmax(LeakyReLU(a^T[W^{(k)}h_v^(k-1)||W^{(k)}h_u^{(k-1)}])) \tag{1.1.5}$$

Η μεταβλητή a αντιπροσωπεύει ένα σύνολο παραμέτρων με δυνατότητα εκμάθησης. Η αναπαράσταση των κρυφών επιπέδων αρχικοποιείται με τα χαρακτηριστικά κάθε κόμβου και η συνάρτηση softmax διασφαλίζει ότι τα βάρη της προσοχής αθροίζονται σε ένα.

Ο παραπάνω μηχανισμός ονομάζεται *self-attention*, αλλά το GAT χρησιμοποιεί επιπλέον *multi-head attention* για να σταθεροποιήσει τη μάθηση και να κάνει το μοντέλο πιο εκφραστικό. Οι ακριβείς εξισώσεις βρίσκονται στο [83].

Το GAT είναι αποτελεσματικό αφού από τα ζεύγη κόμβου-γείτονα μπορούν να υπολογιστούν ταυτόχρονα. Επιπλέον, τα μεγέθη της γειτονιάς του είναι αδιάφορα και μπορεί να εφαρμοστεί εύκολα σε επαγωγικά μαθησιακά προβλήματα.

Το **Graph Isomorphism Network (GIN)** [101] είναι η πρώτη χωρική προσέγγιση που αντιμετωπίζει την αδυναμία προηγούμενων χωρικών μοντέλων να κάνουν διάκριση μεταξύ διαφορετικών δομών γράφων με βάση τις ενσωματώσεις που παράγονται. Για να γίνει αυτό, το GIN χρησιμοποιεί μια απλή τεχνική, προσθέτοντας μια παράμετρο βάρους για τον κεντρικό κόμβο της συνέλιξης. Η λειτουργία ορίζεται παρακάτω όπου $\epsilon^{(k)}$ είναι το βάρος.

$$h_v^{(k)} = MLP((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N(u)} h_u^{(k-1)}) \tag{1.1.6}$$

Το GIN αποδεικνύεται ότι είναι εξίσου ισχυρό με το τεστ ισομορφισμού γραφήματος Weisfeiler-Lehman, δηλαδή παράγει διαφορετικές ενσωματώσεις κόμβων όταν ασχολούμαστε με μη ισομορφικά γραφήματα. Αυτό καθιστά

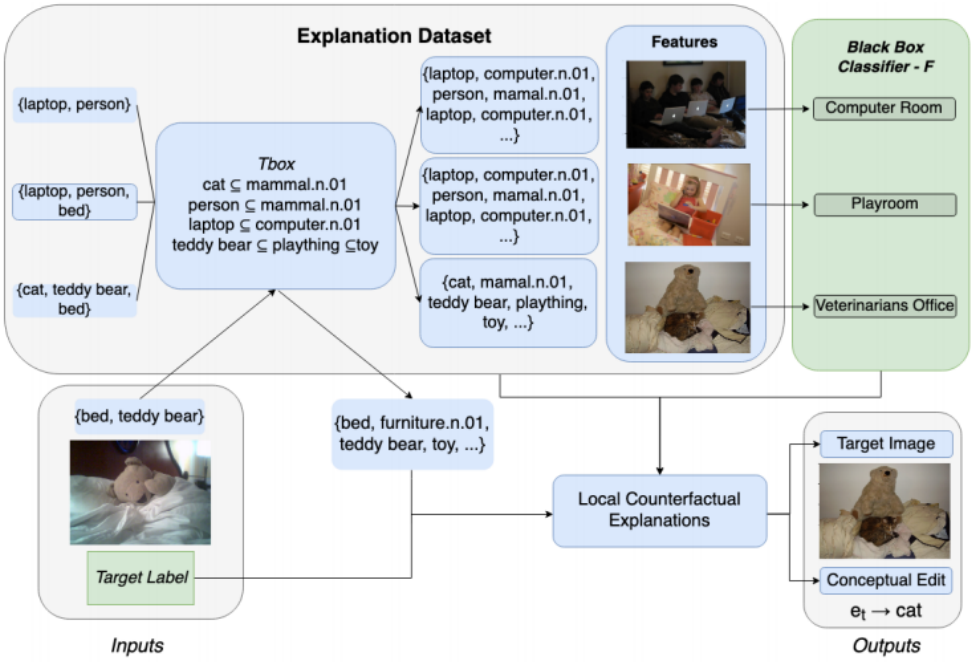


Figure 1.1.2: Πλαίσιο της Εννοιολογικής Επεξεργασίας για Εξηγήσεις με Αντιπαράδειγμα [22].

αυτό το μοντέλο εξαιρετικά ισχυρό και επομένως το πιο εκφραστικό μεταξύ των παραλλαγών. Το GIN χρησιμοποιεί το Perceptron πολλαπλών επιπέδων (MLP) και τη συνάρτηση αθροίσματος ως συσσωρευτή. Για κάθε επίπεδο, οι ενσωματώσεις κόμβων αθροίζονται και το αποτέλεσμα συνενώνεται. Έτσι, η εκφραστικότητα του τελεστή του αθροίσματος συνδυάζεται με τη μνήμη των προηγούμενων επαναλήψεων χρησιμοποιώντας συνένωση. Παρόλα αυτά, θα πρέπει να έχουμε κατά νου ότι η θεωρητική ισχύς του GIN δεν εμφανίζεται πάντα στην πράξη.

1.1.3 Εξηγήσεις με Αντιπαράδειγμα

Οι εξηγήσεις με αντιπαράδειγμα (counterfactual explanations) στον τομέα της Επεξηγησιμότητας στην Τεχνητή Νοημοσύνη (explainable AI) στοχεύουν να δώσουν μια εξήγηση για το "Τι θα πρέπει να αλλάξει προκειμένου το μοντέλο να λάβει μια διαφορετική απόφαση". Επομένως, μπορούν ουσιαστικά να εξηγήσουν προβλέψεις μεμονωμένων περιπτώσεων, όπου οι αιτίες του προβλεπόμενου αποτελέσματος είναι συγκεκριμένες τιμές χαρακτηριστικών αυτής της περίπτωσης. Είναι αντιθετικές και επιλεκτικές, που σημαίνει ότι βρίσκουν τις ελάχιστες αλλαγές στον χώρο των χαρακτηριστικών. Ταυτόχρονα, είναι εύκολα κατανοητές από τους ανθρώπους και συνήθως προσφέρουν πολλαπλές διαφορετικές απαντήσεις για την ίδια περίπτωση που την εξηγούν εξίσου καλά.

Conceptual Edits as Counterfactual Explanations - Εννοιολογική Επεξεργασία για Εξηγήσεις με Αντιπαράδειγμα

Η δουλειά αυτή εμπνέεται από και βρίσκει εφαρμογή στην προσέγγιση που παρουσιάζεται από τους Φιλανδριανός και λοιποί [22]. Οι ίδιοι προτείνουν ένα θεωρητικό πλαίσιο για τον υπολογισμό των εξηγήσεων με αντιπαράδειγμα μέσω εννοιολογικών επεξεργασιών. Έννοιες ονομάζουμε τις γενικές αναπαραστάσεις των αντικειμένων που υπάρχουν στα δεδομένα εισόδου και συνδέονται με ιεραρχική εξωτερική γνώση από το WordNet [58]. Το περίγραμμα του παρουσιαζόμενου πλαισίου περιγράφεται στο Σχήμα 1.1.2.

Είναι ευδιάκριτο ότι το μοντέλο μαύρου-κουτιού που χρησιμοποιείται σε αυτήν την εφαρμογή είναι ένας ταξινομητής, ο οποίος κατηγοριοποιεί τις εικόνες από το σύνολο δεδομένων COCO ανάλογα με τον τύπο δωματίου που απεικονίζεται. Έτσι, απαντάται η ερώτηση "Τι θα έπρεπε να αλλάξει ώστε κάτι να ταξινομηθεί ως X αντί για Υ».

Συνολικά, ο υπολογισμός των counterfactuals συνεπάγεται:

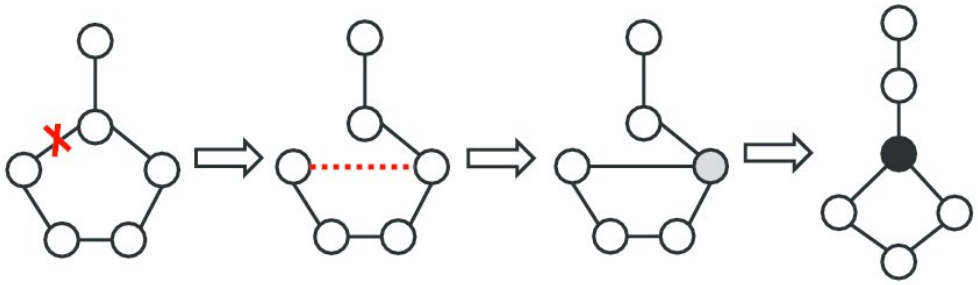


Figure 1.1.3: Απόσταση Επεξεργασίας μεταξύ δυο Γράφων. [4]

Το ελάχιστο GED απαιτεί 3 λειτουργίες επεξεργασίας και εάν όλες ήταν εξίσου σταθμισμένες η τιμή του θα ήταν 3.

- Εύρεση της απόστασης μεταξύ όλων των παρόντων εννοιών χρησιμοποιώντας τα συντομότερα μονοπάτια στο μη κατευθυνόμενο γράφημα ιεραρχίας εννοιών - TBox - που λαμβάνεται με τον αλγόριθμο του Dijkstra.
- Υπολογισμό της απόστασης επεξεργασίας των συνόλων εννοιών (Concept Set Edit Distance) από το ένα σύνολο εννοιών στο άλλο χρησιμοποιώντας τον αλγόριθμο του Karp για την πλήρη αντιστοίχιση ελάχιστου βάρους. Αυτό απαιτεί την κατασκευή ενός πλήρους διμερούς γραφήματος από τα δύο σύνολα.
- Λήψη τοπικών εξηγήσεων με αντιπαράδειγμα βρίσκοντας τα συντομότερα μονοπάτια στο ήδη κατασκευασμένο γράφημα (χρησιμοποιώντας ξανά τον αλγόριθμο του Dijkstra).

,

Η συνεισφορά μας σε αυτή την προσέγγιση είναι διπλή: i) χρήση της δομής γράφου που περιγράφει τις σχέσεις μεταξύ των συνόλων εννοιών και επομένως είναι πιο πλούσια αναπαράσταση και ii) εύρεση των πιο όμοιων ζευγών δειγμάτων εισόδου και επομένως μόνο υπολογισμός των τροποποιήσεων για αυτά τα ζεύγη, τα οποία θεωρούνται ελάχιστα. Φυσικά, τα πιο παρόμοια ζεύγη λαμβάνονται με χρήση GNNs.

1.1.4 Ομοιότητα Γράφων

Η *Ομοιότητα Γράφων ή Αντιστοίχιση Γράφων* ορίζεται ως το πρόβλημα εύρεσης ομοιοτήτων μεταξύ γραφημάτων, δηλαδή εύρεσης αντιστοίχισης $s : G \times G \to \mathbb{R}$ για ένα ζεύγος γράφων, χαρακτηριστικής του πόσο όμοιοι ή ανόμοιοι είναι. Η *Ακριβής Αντιστοίχιση Γράφων* είναι ουσιαστικά το πρόβλημα του *Ισομορφισμού Γράφων*. Ωστόσο, η ακριβής αντιστοίχιση δεν είναι πάντα δυνατή. Για παράδειγμα, τα γραφήματα θα μπορούσαν να έχουν διαφορετικούς αριθμούς κόμβων ή ακμών ή να διαθέτουν χαρακτηριστικά. Αυτό το πρόβλημα ορίζεται ως *Ανακριβής ή Ανεκτική σε Σφάλματα Αντιστοίχιση Γράφων* και συνεπάγεται την εύρεση της καλύτερης δυνατής αντιστοίχισης.

Η **Απόσταση Επεξεργασίας Γράφων ή GED** [74] είναι ανακριβής τεχνική και ορίζεται παρακάτω. Η GED του ζεύγους γραφημάτων G_1 και G_2 συμβολίζεται ως $GED(G_1, G_2)$, οι λειτουργίες επεξεργασίας είναι e_i, το κόστος τους είναι $c(e_i)$ και $P(G_1, G_2)$ υποδηλώνει το σύνολο των διαδρομών επεξεργασίας που μετατρέπουν το πρώτο γράφημα σε ισομορφικό του δεύτερου. Οι λειτουργίες επεξεργασίας περιλαμβάνουν την εισαγωγή, διαγραφή και αντικατάσταση κορυφών και ακμών και το κόστος τους καθορίζεται από τον χρήστη.

$$GED(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in P(G_1,G_2)} \sum_{i=1}^{k} c(e_i) \tag{1.1.7}$$

Η GED είναι ένα υπολογιστικά ακριβό NP-δύσκολο πρόβλημα. Ακόμη κι η προσέγγισή του είναι είναι δύσκολη με αποτέλεσμα να μπορεί να θεωρηθεί πως ανήκει στην κατηγορία πολυπλοκότητας APX-hard. Έχουν υπάρξει πολλές προσεγγιστικές λύσεις που επιτυγχάνουν κυβικές πολυπλοκότητες στην πλειοψηφία τους. Εμείς θα χρησιμοποιήσουμε μια πολύ γνωστή προσέγγιση που βασίζεται στη διμερή αντιστοίχιση γράφου μέσω του αλγόριθμου εκχώρησης Volgenant-Jonker [18].

Οι **Πυρήνες Γράφων** είναι συναρτήσεις πυρήνα που χρησιμοποιούνται σε γραφήματα, οι οποίες μετρούν την ομοιότητα τους σε πολυωνυμικό χρόνο. Παρέχουν μια αποτελεσματική, εκφραστική και ευρέως εφαρμόσιμη

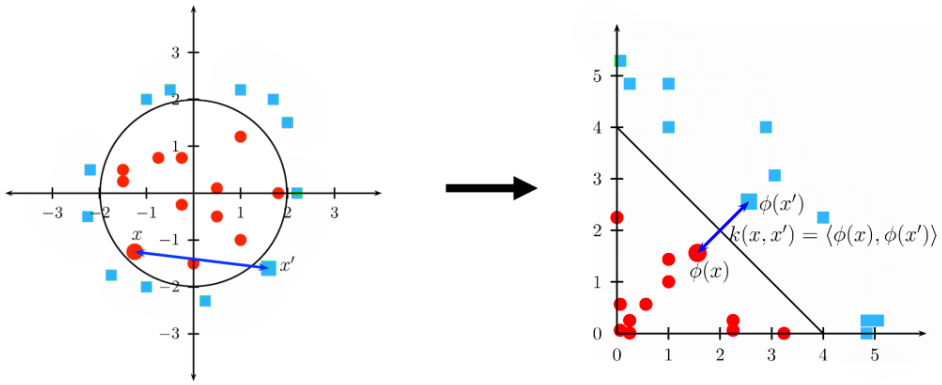


Figure 1.1.4: Παραδειγματική Απεικόνιση του Kernel Trick [38]

εναλλακτική λύση στο GED.

Ένας θετικά ορισμένος πυρήνας σε ένα μη κενό σύνολο X είναι μια συμμετρική συνάρτηση $K : X \times X \to \mathbb{R}$ δεδομένου ότι η 1.1.8 ισχύει, όπου $x_i \in X$, $n \in \mathbb{N}$ και c_i είναι πραγματικοί αριθμοί. Συχνά γίνεται διάκριση μεταξύ θετικά οριστισμένων (p.d.) πυρήνων για τους οποίους ισχύει μόνο η ισότητα και θετικά ημι-ορισμένων (p.s.d.) πυρήνων για τους οποίους ισχύει το αντίθετο. Σε αυτή τη διατριβή, θα διερευνήσουμε πέντε μεθόδους πυρήνα γράφων, τρεις από τις οποίες μπορούν να χειριστούν γραφήματα με χαρακτηριστικά κόμβων.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \tag{1.1.8}$$

1.2 Προτεινόμενο Μοντέλο

1.2.1 Συνεισφορά

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

- Χρησιμοποιούμε το πρόβλημα της Ομοιότητας Γράφων για να εξερευνήσουμε μια πληθώρα παραλλαγών Νευρωνικών Δικτύων Γράφων και εν σύνεχεία να χρησιμοποιήσουμε μερικές από τις πιο κυρίαρχες στη βιβλιογραφία για την αντιμετώπιση του. Εστιάζουμε σε συνελικτικά μοντέλα, τα οποία συγκρίνονται μεταξύ τους και επομένως, συνάγεται η εκφραστικότητα και η χρησιμότητα του καθενός.
- Η χρήση GNN για την αντιμετώπιση της Αντιστοίχισης Γράφων είναι μια αρκετά πρόσφατη προσέγγιση. Το μεγαλύτερο μέρος της βιβλιογραφίας επικεντρώνεται σε μη νευρωνικές μεθόδους. Αντίθετα, αυτή η εργασία προτείνει τη χρήση μοντέλων GNN, που πρόσφατα βρίσκονται στο επίκεντρο της προσοχής, όχι μόνο για τη σύγκριση γραφημάτων αλλά και για την εξαγωγή σημαντικών αναπαραστάσεων. Η προσέγγισή μας ξεπερνά τις κλασσικές μεθόδους τόσο ποιοτικά όσο και ποσοτικά.
- Η σύγκριση και η ομοιότητα γράφων σκηνής είναι μια νέα εργασία. Η βιβλιογραφία για γράφους σκηνής επικεντρώνεται κυρίως στη δημιουργία γραφημάτων, στην ανάκτηση κειμένου ή στο VQA.
- Τα συγκριτικά αποτελέσματα που εξάγονται χρησιμοποιούνται τελικά για την παραγωγή εξηγήσεων με αντιπαράδειγμα. Από όσο γνωρίζουμε, δεν υπάρχει προηγούμενη βιβλιογραφία που να συνδυάζει αυτές τις πτυχές στον τομές του explainable AI.

1.2.2 Μοντέλο GNN

Το GNN μοντέλο που προτείνουμε ακολουθεί το περίγραμμα που παρουσιάζεται στο σχήμα 1.2.1. Η είσοδος αποτελείται από ζεύγη γράφων σκηνής τα οποία στη συνέχεια υποβάλλονται σε επεξεργασία από πανομοιότυπα μοντέλα ενσωμάτωσης GNN. Αυτά τα μοντέλα αποτελλούνται από στοιβαγμένα επίπεδα μιας παραλλαγής GNN

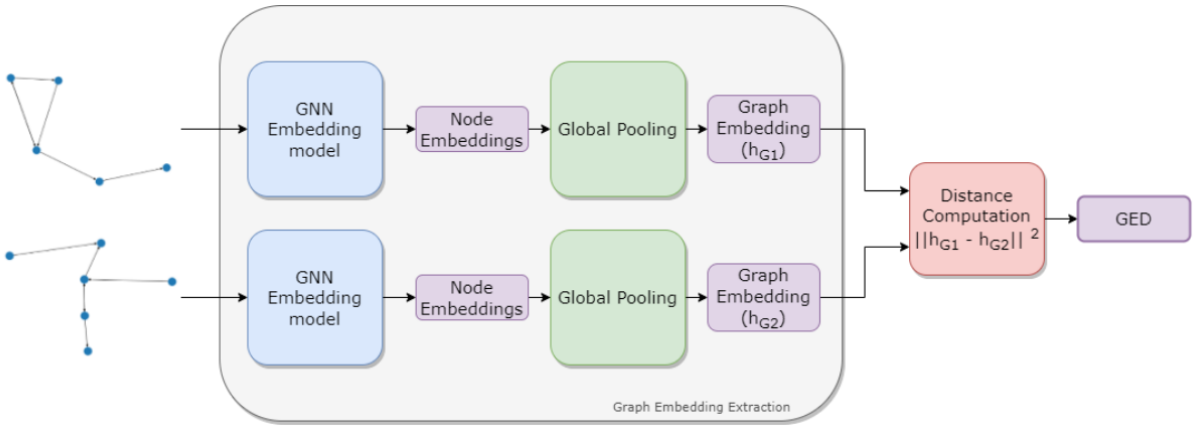


Figure 1.2.1: Προτεινόμενο GNN μοντέλο για την Ομοιότητα Γράφων Σκηνής.

σε συνδυασμό με άλλους τύπους τυπικών επιπέδων όπως ενεργοποίησης, κανονικοποίησης και dropout. Το μοντέλο GNN παράγει ενσωματώσεις σε επίπεδο κόμβου οι οποίες με τη σειρά τους δειγματίζονται για τη δημιουργία καθολικών ενσωματώσεων γραφημάτων. Έτσι ολοκληρώνεται το πρώτο μέρος του προτεινόμενου μοντέλου που εκτελεί την εξαγωγή ενσωματώσεων σε επίπεδο γράφου. Αυτό θα χρησιμοποιήσουμε τελικά για να εξαγάγουμε τις ενσωματώσεις για όλα τα γραφήματα του συνόλου δεδομένων. Το ζεύγος των ενσωματώσεων (h_{G1}, h_{G2}) χρησιμοποιείται στη συνέχεια για τον υπολογισμό της απόστασης μεταξύ των διανυσμάτων και κατά συνέπεια για την πρόβλεψη μιας τιμής που αντιπροσωπεύει την υπολογισμένη απόσταση επεξεργασίας γράφου.

Η διαδικασία εκπαίδευσης περιλαμβλανει την τροφοδοσία ενός μικρού υποσυνόλου ζευγών γράφων στο μοντέλο, τον υπολογισμό της απώλειας μεταξύ της υπολογισμένης απόστασης εξόδου και της πραγματικής απόστασης επεξεργασίας γράφου και την εκ νέου διάδοση της μέσω του δικτύου. Αυτή η διαδικασία διακόπτεται μετά από ένα ορισμένο αριθμό εποχών, που καθορίζονται από τον χρήστη. Μετά την ολοκλήρωση της εκπαίδευσης, όλα τα γραφήματα περνούν μεμονωμένα μέσω της Ενότητας Εξαγωγής Ενσωματώσεων Γράφων. Οι παραγόμενες ενσωματώσεις μπορούν να συγκριθούν χρησιμοποιώντας ομοιότητα συνημιτόνου.

Το **Μοντέλο Ενσωμάτωσης GNN** είναι η καρδιά της υλοποίησης. Στα Σχήματα 1.2.2 και 1.2.3 μπορούμε να δούμε μια πιο λεπτομερή προβολή του σχεδιασμού της παραλλαγής **GCN/GAT** [41, 83] και της παραλλαγής **GIN** αντίστοιχα. Και στις δύο περιπτώσεις, μπορεί να υπάρχει μια σειρά από πανομοιότυπα επίπεδα, αλλά οι διαστάσεις εισόδου και εξόδου τους ποικίλλουν. Τα επίπεδα στο 8.2.2 αποτελούνται από συνελίξεις GCN ή GAT που ακολουθούνται από τη συνάρτηση ενεργοποίησης ReLU. Προσθέτουμε επίσης dropout, που προσφέρει τη δυνατότητα "απενεργοποίησης" νευρώνων με πιθανότητα p που καθορίζεται από τον χρήστη, προκειμένου να αποφευχθεί η υπερπροσαρμογή. Η παραλλαγή GAT απαιτεί πρόσθετη προσοχή στις διαστάσεις ορισμού. Λόγω της χρήσης multi-head attention, οι διαστάσεις εισόδου πολλαπλασιάζονται με τον αριθμό των κεφαλών σε κάθε στρώμα. Η παραλλαγή GIN έχει πιο περίπλοκη αρχιτεκτονική. Συγκεκριμένα, η συνέλιξη του GIN απαιτεί το σχεδιασμό ενός μοντέλου MLP για την εκπαίδευση της παραμέτρου βάρους του κεντρικού κόμβου της συνέλιξης. Όπως φαίνεται στο σχήμα, αποφασίσαμε να χρησιμοποιήσουμε δύο διαδοχικά πλήρως συνδεδεμένα επίπεδα σε συνδυασμό με λειτουργίες ενεργοποίησης ReLU, το πρώτο από τα οποία περιέχει επίσης κανονικοποίηση παρτίδας (batch normalization). Αυτές οι επιλογές ήταν εμπνευσμένες από τη δημοσίευση που παρουσίασε το GIN [101]. Σε αυτή την εργασία, στοίβαξαν 5 πανομοιότυπα στρώματα που περιέχουν τα προαναφερθέντα συστατικά, ενώ εμείς επιλέγουμε να πειραματιστούμε με την ποσότητα αυτών.

Είναι αξιοσημείωτο ότι οι ενσωματώσεις κόμβων που παράγονται από κάθε στρώμα του μοντέλου Ενσωμάτωσης GNN συνενώνονται πριν δειγματοληφθούν. Αυτή είναι μια προσπάθεια διατήρησης περισσότερων πληροφοριών που συγκεντρώθηκαν κατά τη διάρκεια της διαδικασίας και κατά συνέπεια δημιουργίας πιο εκφραστικών ενσωματώσεων γράφων. Κοινή πρακτική χρησιμοποιείται επίσης στο [101].

Το **Global Pooling** που χρησιμοποιείται για τη δειγματοληψία των ενσωματώσεων κορυφών που εξάγονται από τις παραλλαγές GNN διαφέρει ανάλογα με το μοντέλο. Για το GCN/GAT προτιμήθηκε το μέσο - average pooling, εμπνευσμένο από το [79], ενώ για το GIN επιλέξαμε να αθροίσουμε τις ενσωματώσεις κόμβων όπως προτάθηκε από τους δημιουργούς του προκειμένου να αυξήσουμε την εκφραστικότητα. Όλες οι παραπάνω

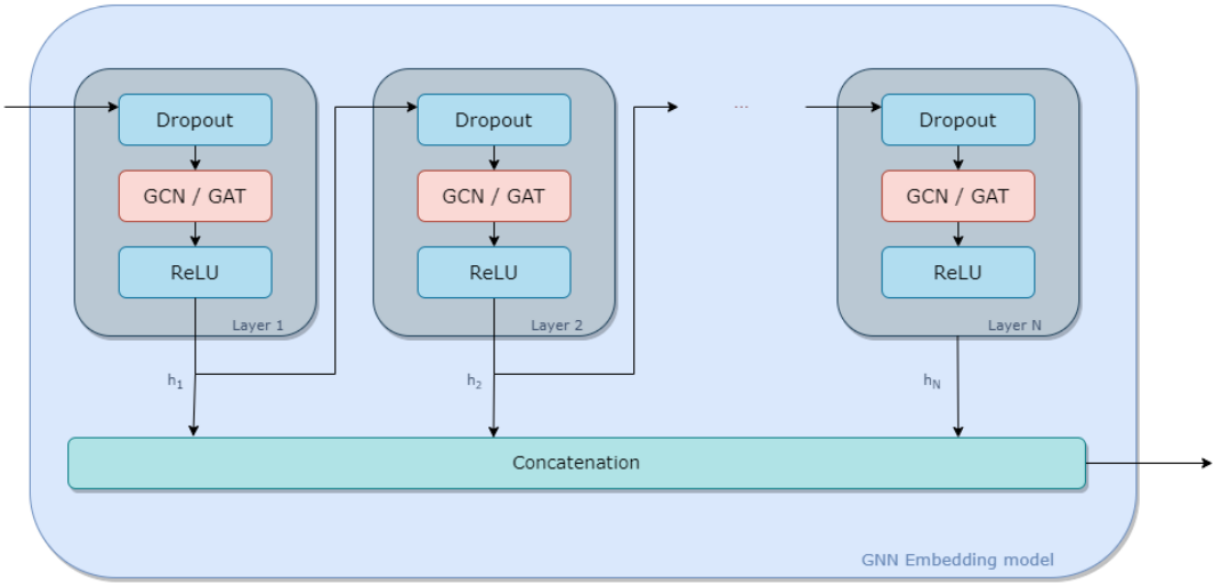


Figure 1.2.2: Σχεδιασμός μοντέλου με τις παραλλαγές GCN/GAT.

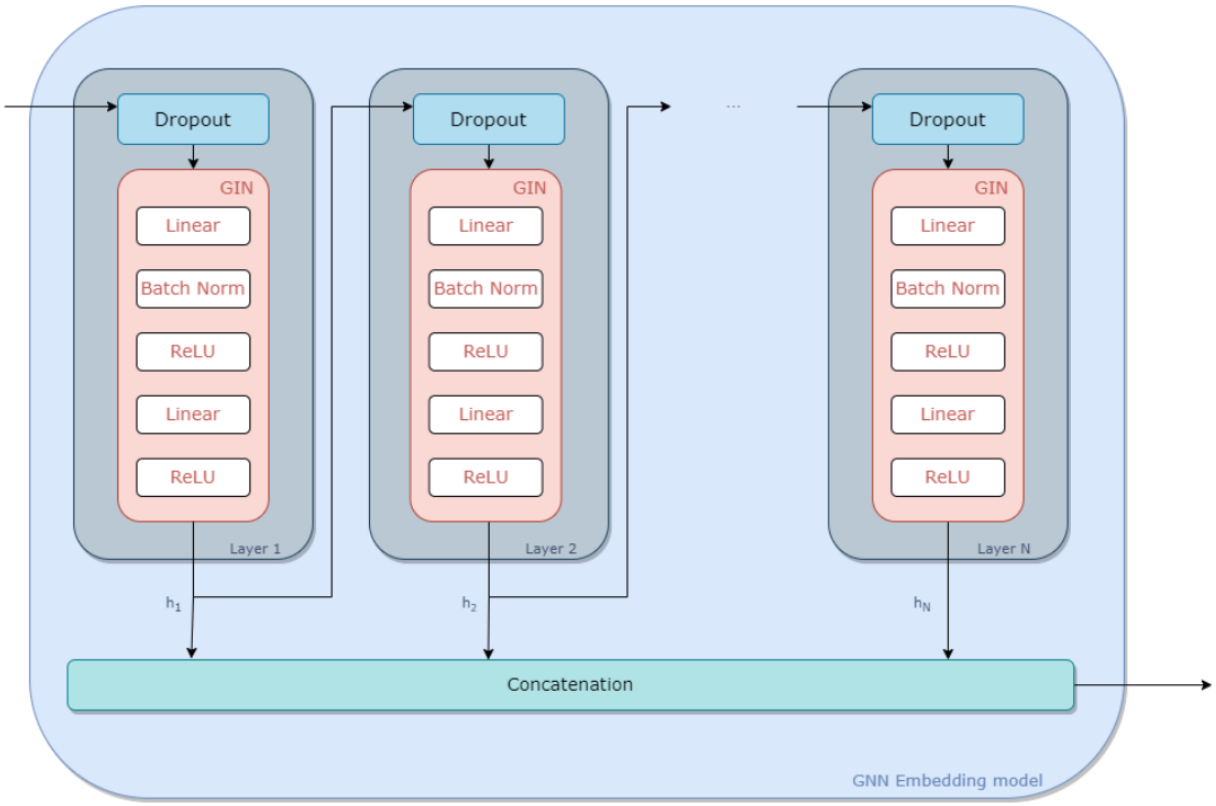


Figure 1.2.3: Σχεδιασμός μοντέλου με την παραλλαγή GIN.

Table 1.1: Στατιστικά Γράφων

	Μέσος όρος	**Min**	**Max**
Πυκνότητα	0.20	0.14	0.47
Κόμβοι	7.25	6	15
Ακμές	9.04	5	36
Απομονωμένοι Κόμβοι	0.47	0	3

λειτουργίες εκτελούνται καλύτερα σε παρτίδες για λόγους αποτελεσματικότητας.

Η διαδικασία **Εξαγωγής Ενσωμάτωσεων Γράφου** ακολουθείται από τον υπολογισμό της απόστασης μεταξύ των διανυσμάτων προκειμένου να πραγματοποιηθεί παλινδρόμηση. Η απόσταση των διανυσμάτων ορίζεται ως η νόρμα L_2 της αφαίρεσής τους. Αυτή η μέθοδος χρησιμοποιήθηκε επίσης στο [5] που επηρεάζει άμεσα την προσέγγισή μας, μαζί με την αρχιτεκτονική και το στυλ εκπαίδευσης που προτείνονται στο [79]. Η χρήση της απώλειας Μέσου Τετραγωνικού Σφάλματος μαζί με αυτήν τη μετρική απόστασης οδηγεί σε μια τεχνική που ονομάζεται **Multi-Dimensional Scaling (MDS)** που χρησιμοποιείται για μείωση διαστάσεων. Έτσι, οι αποστάσεις διατηρούνται με μεγαλύτερη ακρίβεια.

Η διαδικασία εκπαίδευσης ολοκληρώνεται μετά από προκαθορισμένο αριθμό εποχών. Η πλέον εκπαιδευμένη μονάδα Εξαγωγής Ενσωματώσεων Γράφου χρησιμοποιείται για τον υπολογισμό των ενσωματώσεων για όλα τα γραφήματα. Αυτές οι ενσωματώσεις μπορούν στη συνέχεια να συγκριθούν ως κλασικά διανύσματα χρησιμοποιώντας την ομοιότητα συνημιτόνου.

Το μοντέλο μπορεί να εκπαιδευτεί με οποιοδήποτε μέτρο ομοιότητας γραφήματος. Σε αυτήν την περίπτωση, χρησιμοποιούμε την GED που λαμβάνει υπόψη τόσο τη δομή όσο και τη σημασιολογία των ζευγών γραφημάτων. Είναι επίσης καλά ορισμένη και δε χρησιμοποιείται σε κάποιο συγκεκριμένο κλάδο.

1.3 Πειράματικό Μέρος

1.4 Σύνολο Δεδομένων και Μετρικές

Το **σύνολο δεδομένων** που μας παρέχει τους γράφους σκηνής σε αυτή τη διατριβή είναι το Visual Genome [44]. Το Visual Genome είναι επί του παρόντος το μεγαλύτερο σύνολο δεδομένων περιγραφών εικόνων, αντικειμένων, χαρακτηριστικών, σχέσεων και ζευγών απαντήσεων ερωτήσεων και περιέχει πάνω από 108K εικόνες. Όλες οι προαναφερθείσες οντότητες που μπορούν να κανονικοποιηθούν έχουν αντιστοιχιστεί σε σύνολα του WordNet.

Για αυτήν την εφαρμογή, χρησιμοποιούμε ένα υποσύνολο 500 γράφων από τα χιλιάδες γραφήματα του αρχικού συνόλου δεδομένων. Το σύνολο των γραφημάτων έγινε συμβατό με το WordNet [58]. αφαιρώντας όλους τους κόμβους ή τις ακμές χωρίς τα σύνολα που τους έχουν εκχωρηθεί. Ορισμένα σχετικά στατιστικά στοιχεία βρίσκονται στον Πίνακα 1.1.

Οι μέθοδοι που θα χρησιμοποιηθούν για την αντιμετώπιση του προβλήματος της ομοιότητας γράφων υποθέτουν την ύπαρξη χαρακτηριστικών κόμβων για τα γραφήματα. Η πηγή των χαρακτηριστικών για τους κόμβους των γραφημάτων ήταν επίσης θέμα πειραματισμού. Λόγω του γεγονότος ότι τα επίσημα γραφήματα σκηνής του Visual Genome δεν περιλαμβάνουν χαρακτηριστικά κόμβου ή ακμών, τα χαρακτηριστικά επιλέχθηκε να είναι προεκπαιδευμένες ενσωματώσεις. Εν τέλει, δοκιμάστηκαν οι ενσωματώσεις της ιεραρχίας ουσιαστικών του WordNet χρησιμοποιώντας path2vec [46] που εκπαιδεύτηκαν στην ομοιότητα Wu-Palmer [98] προκειμένου να μιμηθούν τις εσωτερικές λειτουργίες της βασικής αλήθειας καθώς και τα γνωστά στον τομέα της Επεξεργασίας Φυσικής Γλώσσας GloVe Embeddings [67].

Δεδομένου ότι οι μέθοδοι που θα δοκιμαστούν παράγουν λίστες κατάταξης ή προτάσεις με τις πιο παρόμοιες εικόνες για έναν δεδομένο στόχο, αποφασίστηκε να χρησιμοποιηθούν οι παρακάτω μετρικές για τη σύγκριση με τη βασική αλήθεια:

- **Hit Percentage / Ποσοστό Επιτυχίας** - Αυτή η μετρική αντιπροσωπεύει το μέσο ποσοστό των

κοινών προτάσεων μεταξύ των κορυφαίων k αποτελεσμάτων για κάθε εικόνα.

- **Rank Biased Overlap (RBO) / Μεροληπτική Επικάλυψη Κατάταξης** [87] - Αποτελεί μέτρο σύγκρισης ταξινομημένων λιστών που μπορεί να χρησιμοποιηθεί για ημιτελείς ταξινομήσεις, να χειριστεί την απουσία συνάφειας - ύπαρξη διαφορετικών στοιχείων στις δύο βαθμολογίες και να αποδώσει μεγαλύτερη σημασία σε υψηλότερες βαθμίδες, όντας μονότονο με αυξανόμενο βάθος αξιολόγησης.
- **Normalized Discounted Cumulative Gain (NDCG) / Κανονικοποιημένο Προεξοφλημένο Αθροιστικό Κέρδος** [86] - Πρόκειται για μέτρο που αξιολογεί την ποιότητα της κατάταξης. Βαθμολογεί την αξία της σύστασης ενός στοιχείου σε μια συγκεκριμένη θέση, σταθμίζοντας τα στοιχεία υψηλότερης κατάταξης περισσότερο.

1.5 Περιγραφή Πειραμάτων

Βασική Αλήθεια

Για να αξιολογηθούν τα αποτελέσματα των μοντέλων που θα χρησιμοποιηθούν, πρέπει να καθοριστεί ένα ποσοτικό μέτρο και κατά συνέπεια να κατασκευαστεί η βασική αλήθεια. Σε αυτήν την περίπτωση, η βασική αλήθεια θεωρείται ότι είναι ένας πίνακας των αποστάσεων επεξεργασίας μεταξύ κάθε ζεύγους γραφημάτων στο σύνολο δεδομένων. Αυτός ο πίνακας παράγεται χρησιμοποιώντας μια προσέγγιση του ακριβούς Αλγόριθμου Επεξεργασίας Απόστασης Γράφων που βασίζεται στη διμερή αντιστοίχιση γραφήματος μέσω του αλγόριθμου εκχώρησης Volgenant-Jonker [37]. Το κόστος εισαγωγής, διαγραφής και αντικατάστασης κόμβων/ακμών υπολογίζεται ως η απόσταση μεταξύ των αντικειμένων που αντιπροσωπεύουν έναν κόμβο ή ακμή, η οποία ορίζεται από την εννοιολογική τους απόσταση στο γράφημα WordNet Tbox όπως προτείνεται στο [22]. Η υλοποίηση του αλγόριθμου, τον οποίο θα ονομάσουμε *BIP-GED*, παρέχεται από τη βιβλιοθήκη Deep Graph Learning της python - DGL [84].

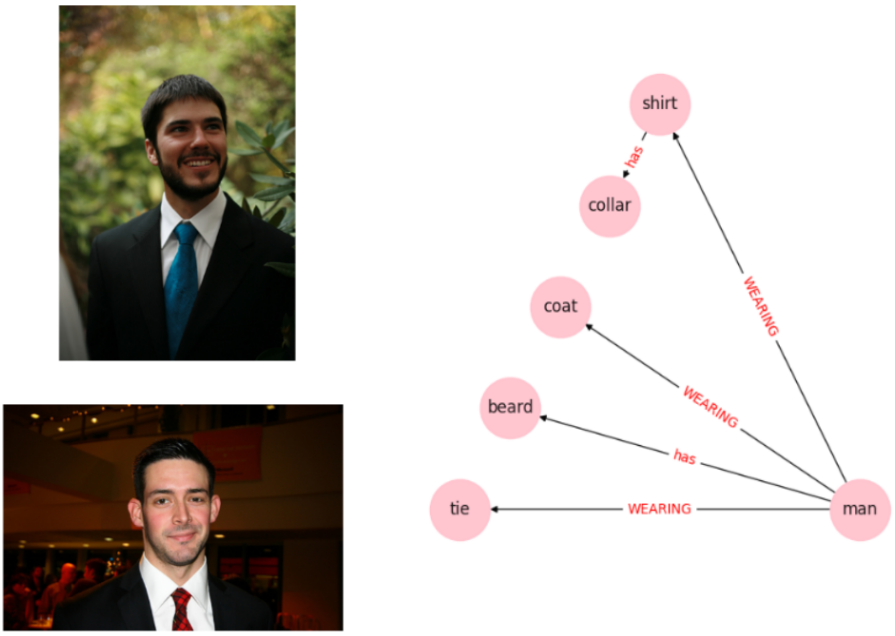


Figure 1.5.1: Εικόνες με κοινούς γράφους σκηνής.

Για να πιστοποιηθεί η ποιότητα του *BIP-GED* ως μέτρο ομοιότητας και επομένως η επιλογή μας να το χρησιμοποιήσουμε ως βασική αλήθεια, παρέχουμε μερικές υποδειγματικές εικόνες. Η μέθοδος αυτή είναι σε θέση να προσδιορίσει ότι η απόσταση επεξεργασίας δύο ισομορφικών γράφων είναι στην πραγματικότητα 0 1.5.1. Από το σχήμα 1.5.3 καταλαβαίνουμε πως το *BIP-GED* δίνει προτεραιότητα τόσο στη δομική ομοιότητα όσο και στη σημασιολογία ώστε να παράγει αξιόπιστα αποτελέσματα για την αξιολόγηση του μοντέλου μας. Συγκεκριμένα, η

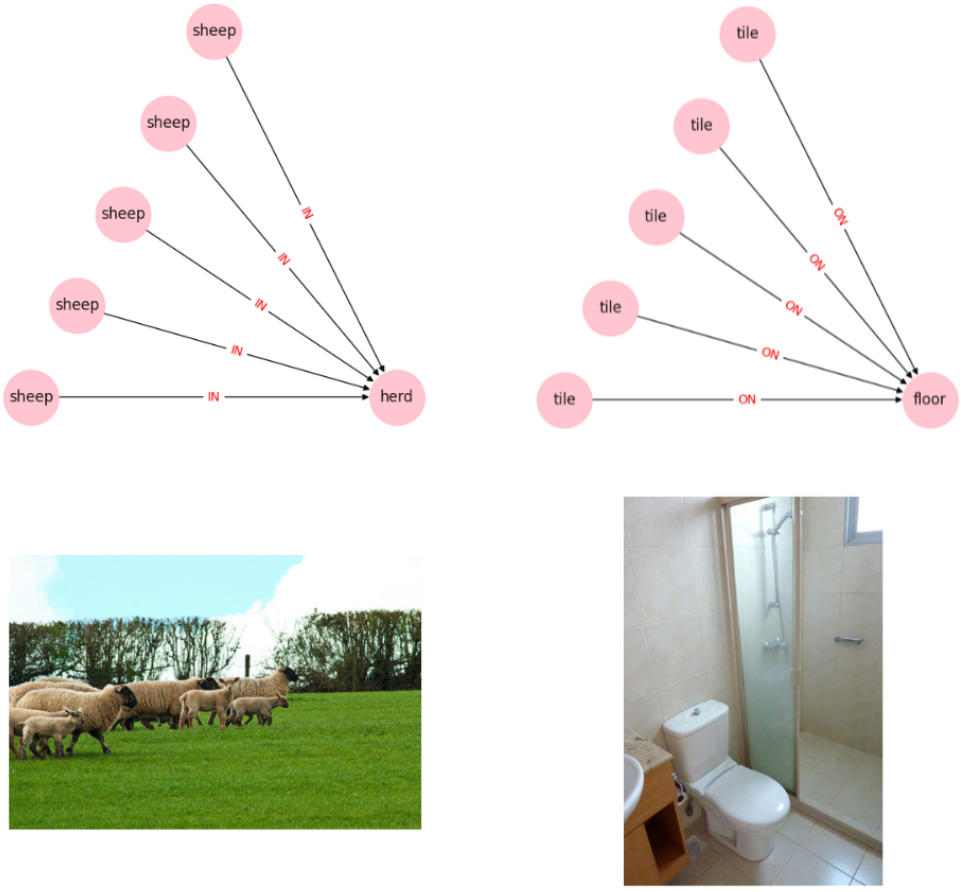


Figure 1.5.2: Παράδειγμα ανόμοιων εικόνων.

έμφαση στη σημασιολογία των αντικειμένων και των σχέσεων αποδεικνύεται κι από την εικόνα 1.5.2, όπου ενώ οι δύο γράφοι είναι δομικά κοντά, τους αποδίδεται μεγάλο GED σκορ μιας και τα απεικονιζόμενα αντικείμενα είναι πολύ μακριά στην ιεραρχία WordNet.

Μέθοδοι Πυρήνα Γράφων

Δεδομένου του ότι η επικρατούσα μέθοδος για τον προσδιορισμό της ομοιότητας γράφων είναι μέσω πυρήνων, οι ακόλουθοι πυρήνες δοκιμάστηκαν στο υποσύνολο των γράφων μας χρησιμοποιώντας τη βιβλιοθήκη της python GraKeL [77]: πυρήνες με δεδομένα που έχουν ετικέτες όπως ο πυρήνας Weisfeiler-Lehman (WL) και ο πυρήνας Pyramid Match (PM) καθώς και πυρήνες για γράφους με χαρακτηριστικά όπως το Propagation Attribute (PA), το Subgraph Matching (SM) και το GraphHopper (GH).

Στα Σχήματα 1.5.5, 1.5.6 έχουμε παράσχει μερικά παραδείγματα κορυφαίων 5 αποτελεσμάτων που παράγονται από τις πέντε μεθόδους πυρήνα, σε σύγκριση με την αλήθεια. Όλες οι μέθοδοι αγνοούν την ιδέα της οντολογικής ομοιότητας που παρέχεται από το WordNet και προτείνουν παρόμοια αποτελέσματα. Οι μέθοδοι που χρησιμοποιούν χαρακτηριστικά αντί για ετικέτες δεν φαίνεται να αξιοποιούν τις ιεραρχικές ενσωματώσεις path2vec αποτελεσματικά. Συγκεκριμένα, από το σχήμα 1.5.6 φαίνεται ότι ένα πιο σύνθετο γράφημα οδηγεί σε απροσδόκητα αποτελέσματα και πιστοποιείται οπτικά η καλή επίδοση του πυρήνα PM και η κακή επίδοση του GH, όπως θα φανεί και ποσοτικά.

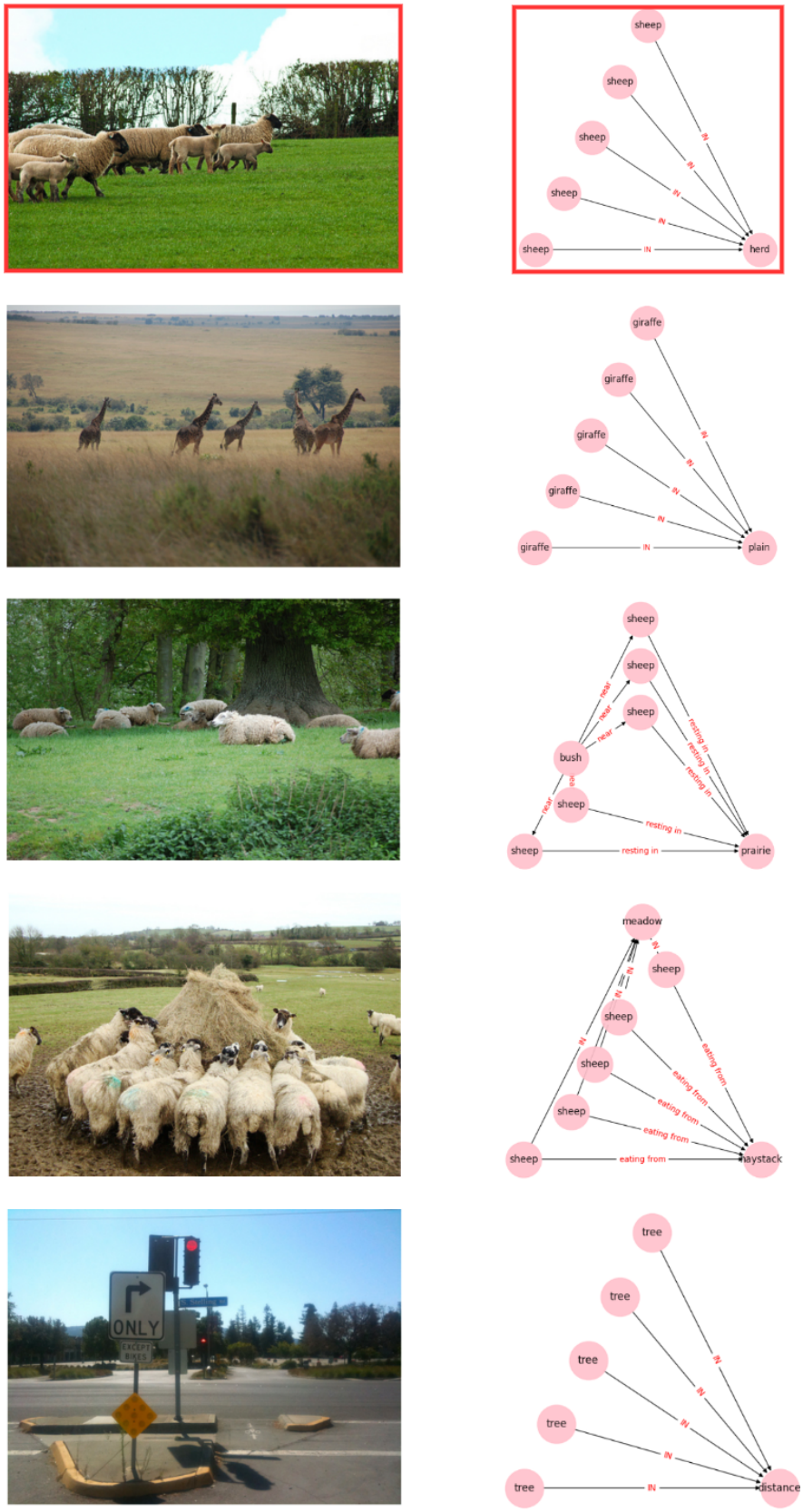


Figure 1.5.3: Κορυφαία 4 αποτελέσματα για μια εικόνα στόχο χρησιμοποιώντας το *BIP-GED*

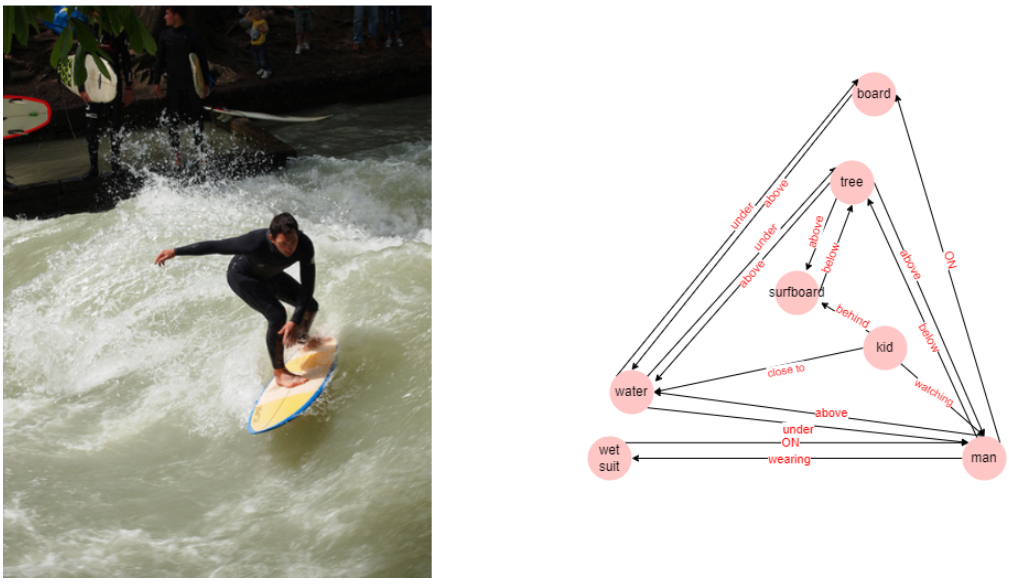


Figure 1.5.4: Εικόνα άντρα σε σανίδα του σερφ και αντίστοιχος γράφος σκηνής.

Μοντέλα με GNN

Το πλαίσιο που περιγράφεται στο 1.2.2 κωδικοποιήθηκε σε Python χρησιμοποιώντας τη βιβλιοθήκη εκμάθησης σε γράφους Pytorch Geometric [21]. Η βασική αρχιτεκτονική που απεικονίζεται στο σχήμα 1.2.1 καθιερώθηκε και οι επιλογές για κάθε στοιχείο αποτέλεσαν υπερπαραμέτρους.

Τα πειράματα πραγματοποιήθηκαν ξεχωριστά για καθεμία από τις τρεις **παραλλαγές** που εξετάσαμε - *GCN, GAT και GIN*. Σε κάθε εκτέλεση υπήρχαν ορισμένες κοινές υπερπαράμετροι προς προσδιορισμό, όπως **ο αριθμός των εποχών, το μέγεθος παρτίδας, η πιθανότητα dropout και οι διαστάσεις και ο αριθμός των συνελικτικών στρωμάτων**. Όλα τα μοντέλα βελτιστοποιήθηκαν χρησιμοποιώντας Adam και απώλεια MSE προκειμένου να επιτευχθεί Πολυδιάστατη Κλιμάκωση. Έτσι, πειραματιστήκαμε επίσης με τις παραμέτρους του βελτιστοποιητή, δηλαδή τον **ρυθμό μάθησης** και την **αποσύνθεση βάρους**. Μια σημαντική επιλογή για τα μοντέλα ήταν επίσης ο **αριθμός προπονητικών ζευγαριών** που έπρεπε να χρησιμοποιηθεί για την προσαρμογή τους. Οι τιμές που χρησιμοποιήσαμε διατηρήθηκαν αυστηρά κάτω από 10% του συνόλου δεδομένων, όχι μόνο για να διατηρηθούν μικρότεροι χρόνοι εκπαίδευσης, αλλά και για να αποδειχθεί η ικανότητα των μοντέλων GNN να δημιουργούν έναν καλό χώρο ενσωμάτωσης χρησιμοποιώντας μόνο ένα μικρό υποσύνολο των γράφων σκηνής.

Η επιλογή των αρχικών **χαρακτηριστικών** για τους κόμβους γραφήματος αποτελεί επίσης μέρος της διαμόρφωσης του μοντέλου. Όπως αναφέρθηκε προηγουμένως, πειραματιστήκαμε με ενσωματώσεις συνόλων που προέρχονται από το WordNet χρησιμοποιώντας path2vec καθώς και ενσωματώσεις GloVe που παράγονται με βάση την ιδέα της συν-εμφάνισης λέξεων σε διάφορους τύπους σωμάτων. Τέλος, ορισμένες από τις παραλλαγές που χρησιμοποιούνται έχουν υπερπαραμέτρους ειδικές για την αρχιτεκτονική τους. Για να είμαστε πιο ακριβείς, οι παραλλαγές GAT που χρησιμοποιούν προσοχή πολλαπλών κεφαλών απαιτούν την επιλογή του αριθμού των **κεφαλών**, ενώ οι παραλλαγές GIN προσφέρουν την επιλογή να είναι εκπαιδεύσιμη ή όχι η **παράμετρος βάρους** ϵ.

1.5.1 Αποτελέσματα

Σύγκριση παραλλαγών

Πρώτον, αναφέρουμε αποτελέσματα και συγκριτικές παρατηρήσεις μεταξύ μοντέλων GNN με βάση τις παραλλαγές και τις υπερπαραμέτρους που χρησιμοποιούνται. Από αυτό το σημείο και μετά, θα αναφερόμαστε στο προτεινόμενο μοντέλο ως *GNN-COMP* όπου το GNN μπορεί να είναι οποιαδήποτε από τις τρεις παραλλαγές - GCN, GAT ή GIN.

Figure 1.5.5: Κορυφαία 5 αποτελέσματα εικόνας με κοπάδι προβάτων χρησιμοποιώντας πυρήνες γράφων.

Η βασική αλήθεια κυκλώνεται με κόκκινο ορθογώνιο. Οι προβλέψεις των μεθόδων πυρήνα ακολουθούν τη σειρά WL-, PM-, PA-, SM-, GH-kernel από πάνω προς τα κάτω.

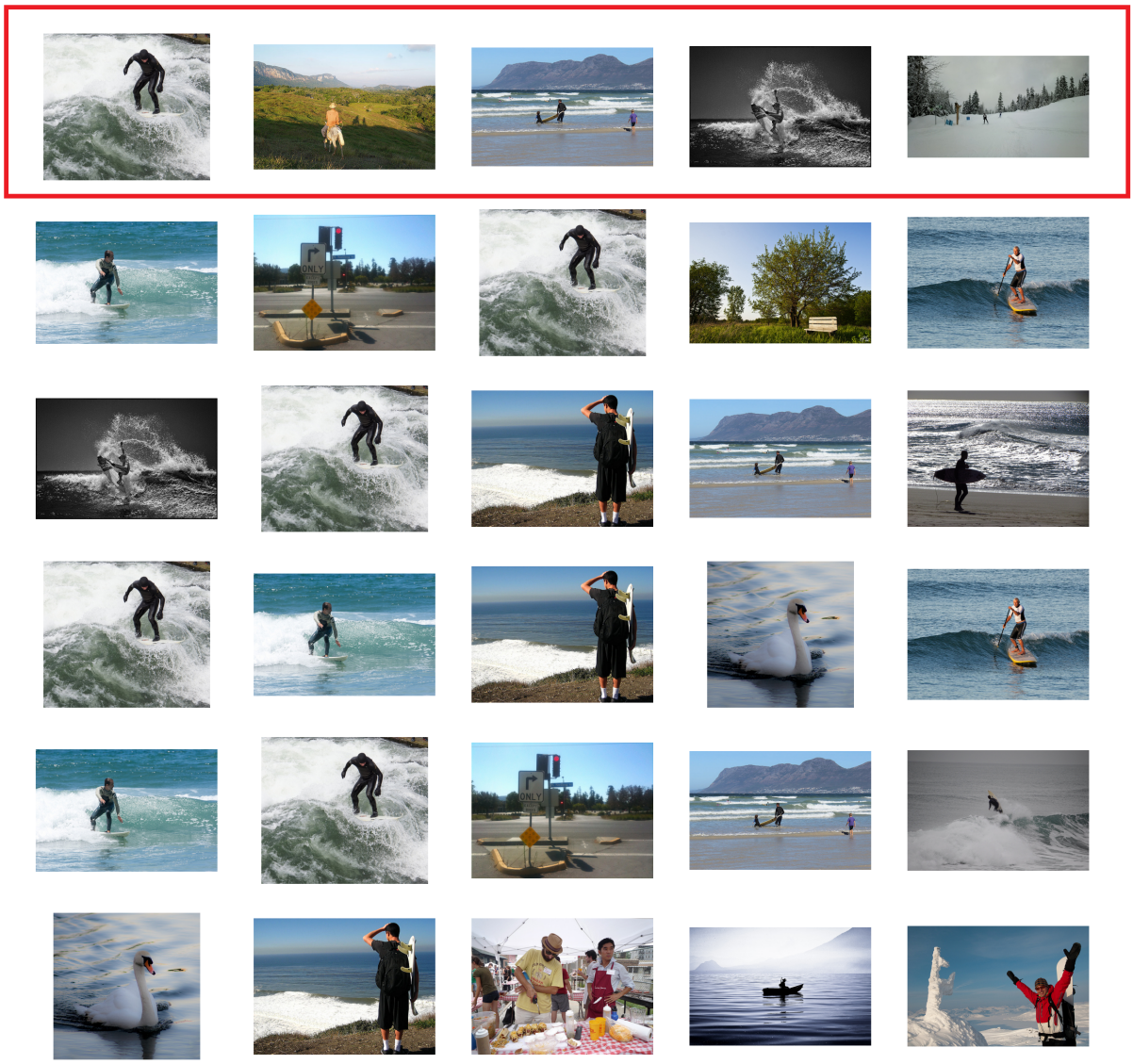
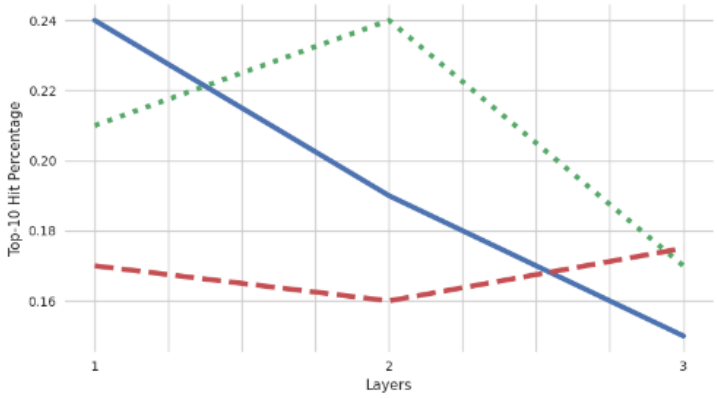
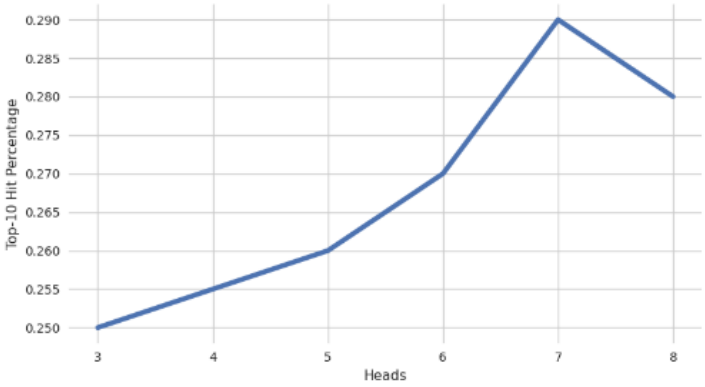


Figure 1.5.6: Κορυφαία 5 αποτελέσματα εικόνας με άντρα σε σανίδα του σερφ χρησιμοποιώντας πυρήνες γράφων.

Η βασική αλήθεια κυκλώνεται με κόκκινο ορθογώνιο. Οι προβλέψεις των μεθόδων πυρήνα ακολουθούν τη σειρά WL-, PM-, PA-, SM-, GH-kernel από πάνω προς τα κάτω.



(a) Απόδοση με βάση τον αριθμό των επιπέδων για διαφορετικές παραλλαγές.



(b) Κεφαλές προσοχής για μοντέλα *GAT-COMP*.

Figure 1.5.7: Κορυφαία-10 αποτελέσματα ποσοστού επιτυχίας σύμφωνα με διάφορες παραμέτρους.

Η πρώτη παρατήρηση που έγινε ήταν ότι ο αριθμός των επιπέδων GNN έχει άμεσο αντίκτυπο στην απόδοση του μοντέλου. Στην πραγματικότητα, οι διαφορετικές παραλλαγές εμφάνισαν διαφορετικές συμπεριφορές σε σχέση με αυτήν την υπερπαράμετρο. Το σχήμα 1.5.7a παρέχει αποτελέσματα του ποσοστού επιτυχίας (HIT-P) των κορυφαίων-10 εικόνων για κάθε παραλλαγή για αριθμό επιπέδων 1 έως 3, δεδομένου ότι όλες οι άλλες υπερπαράμετροι είναι σταθερές. Είναι εύκολα αντιληπτό ότι τα μοντέλα *GCN-COMP* δεν ευνοούν τη στοίβαξη. Οι παραλλαγές *GAT-COMP* γενικά παρουσίασαν παρόμοια συμπεριφορά, με εξαίρεση το μοντέλο 2 επιπέδων του οποίου τα αποτελέσματα απεικονίζονται στο γράφημα. Σε αντίθεση με τα δύο άλλα μοντέλα, οι παραλλαγές *GIN-COMP* φαινόταν να παρουσιάζουν αυξημένο ποσοστό επιτυχίας όταν ο αριθμός των επιπέδων έγινε μεγαλύτερος, κάτι που συμφωνεί με την πρακτική που χρησιμοποιούσε και η δημοσίευση που εισήγαγε αυτή την παραλλαγή.

Όσον αφορά άλλες υπερπαραμέτρους, όπως εποχές ή αριθμό προπονητικών ζευγαριών, η αύξησή τους οδήγησε σε καλύτερα αποτελέσματα. Ένα παρόμοιο εύρημα ισχύει για την αύξηση των διαστάσεων του στρώματος. Ωστόσο, το dropout φαίνεται ότι δεν χρειαζόταν. Αυτό συμβαίνει επειδή τα μοντέλα *GNN-COMP* δεν είναι επιρρεπή σε υπερπροσαρμογή, λόγω του μικρού όγκου των παραδειγμάτων εκπαίδευσης. Η σύγκριση μεταξύ των ενσωματώσεων path2vec και GloVe ως χαρακτηριστικα των κόμβων των γράφων αποτυπώνεται στον Πίνακα 1.2. Γίνεται σαφές ότι τα μοντέλα GNN ευνοούν τον δεύτερο τύπο embeddings. Για όλες τις παραλλαγές και τις μετρικές, οι ενσωματώσεις GloVe οδηγούν σε καλύτερη απόδοση - κάτι που δεν είναι διαισθητικό.

Τέλος, όσον αφορά την εξερεύνηση υπερπαραμέτρων που αντιστοιχούν μοναδικά σε κάθε μοντέλο, παρέχεται το σχήμα 1.5.7b. Σε αυτό το σχήμα απεικονίζεται η απόδοση των μοντέλων *GAT-COMP* που έχουν εκπαιδευτεί με 5000 ζεύγη σε σχέση με τον αριθμό των κεφαλών που χρησιμοποιούνται από τον μηχανισμό προσοχής. Δεν αποτελεί έκπληξη το γεγονός ότι η αύξηση των κεφαλών οδηγεί σε βελτιωμένη απόδοση. Όσον αφορά την παράμετρο βάρους ϵ των μοντέλων *GIN-COMP*, παραδόξως, οδηγεί σε χειρότερα αποτελέσματα όταν ορίζεται

Table 1.2: Σύγκριση ποσοστού επιτυχίας μεταξύ γραφημάτων με ενσωματώσεις Path2vec και GloVe με τα κορυφαία-10 αποτελέσματα μοντέλων GNN-COMP

	WU-P Path2vec		**GloVe**	
	HIT-P	**mRBO**	**HIT-P**	**mRBO**
GCN-COMP	0.212	0.1594	0.2412	0.1719
GAT-COMP	0.1936	0.1364	0.2088	0.1463
GIN-COMP	0.1518	0.0826	0.173	0.093

Table 1.3: Συνολικά αποτελέσματα κορυφαίων 10, 5 και 2 προβλέψεων για τις ποσοτικές μετρικές.

	Top-10			**Top-5**			**Top-2**		
	HIT-P	**mRBO**	**NDCG**	**HIT-P**	**mRBO**	**NDCG**	**HIT-P**	**mRBO**	**NDCG**
WL-KERNEL	0.1982	0.1574	0.5299	0.1656	0.1266	0.4984	0.101	0.0905	0.4614
PM-KERNEL	0.2032	0.1752	0.5299	0.1796	0.1547	0.4984	0.134	0.129	0.4614
PA-KERNEL	0.1646	0.1241	0.5272	0.124	0.0964	0.4958	0.077	0.0735	0.4591
SM-KERNEL	0.1806	0.1409	0.5272	0.1412	0.1144	0.4958	0.093	0.0845	0.459
GH-KERNEL	0.0714	0.0492	0.9999*	0.0484	0.0363	0.9999*	0.03	0.027	0.9999*
GCN-COMP	**0.3142**	**0.2441**	**1.0**	**0.2476**	**0.1988**	**1.0**	0.171*	0.1515	**1.0**
GAT-COMP	0.3046*	0.2367*	**1.0**	0.2324*	0.1935*	**1.0**	**0.177**	0.1555*	**1.0**
GIN-COMP	0.2704	0.2216	**1.0**	0.2228	0.1899	**1.0**	0.165	**0.1595**	**1.0**

Τα καλύτερα αποτελέσματα σε κάθε στήλη έχουν έντονη γραφή και τα δεύτερα καλύτερα σημειώνονται με $*$.

σε εκπαιδεύσιμη, κάτι που είναι μια άμεση αντίφαση με τις προσδοκίες μας.

Λαμβάνοντας υπόψη όλα τα προαναφερθέντα δεδομένα, είναι προφανές ότι το GIN έχει τη χειρότερη απόδοση μεταξύ των παραλλαγών όταν παρέχεται μικρότερος αριθμός ζευγαριών εκπαίδευσης. Το GCN και το GAT από την άλλη πλευρά, παράγουν αρκετά παρόμοια αποτελέσματα. Παρά τον τίτλο του ως ένα από τα πιο ισχυρά μοντέλα, το GIN δεν κυριαρχεί έναντι των απλούστερων μοντέλων όσον αφορά τις μετρικές που εξετάζονται.

1.5.2 Συνολική απόδοση

Η συνολική απόδοση των μοντέλων *GNN-COMP* βρίσκεται στον Πίνακα 1.3. Αξιολογήσαμε τα μοντέλα με βάση το ποσοστό επιτυχίας, το μέσο RBO και το NDCG των κορυφαίων 10, 5 και 2 προτάσεων των πιο παρόμοιων ζευγών γράφων. Τα ίδια αποτελέσματα αναφέρονται εδώ για όλες τις μεθόδους πυρήνα για εύκολη σύγκριση. Τα αποτελέσματα *GNN-COMP* που παρέχονται είναι τα καλύτερα επιτεύγματα σε κάθε περίπτωση με διαφορετικές διαμορφώσεις για κάθε παραλλαγή.

Αρχικά, είναι προφανές ότι μεταξύ των μεθόδων πυρήνα γραφήματος, ο πυρήνας αντιστοίχισης πυραμίδας ξεπερνάει σχεδόν κάθε φορά τους άλλους. Ωστόσο, οι βαθμολογίες NDCG που προκύπτουν είναι πολύ χαμηλές σε σύγκριση με τις άλλες μεθόδους. Αυτό είναι μια σαφής ένδειξη ότι παρόλο που η σχετικότητα της κατάταξης είναι ακριβής, οι βαθμολογίες που παράγονται δεν αντικατοπτρίζουν την αλήθεια. Αντίθετα, η ακριβώς αντίθετη παρατήρηση γίνεται στην περίπτωση του πυρήνα GraphHopper. Η κακή του απόδοση στις άλλες δύο μετρήσεις, οι οποίες έχουν μεγαλύτερη σημασία για το πρόβλημα, ωστόσο, αποδεικνύουν ότι δεν είναι κατάλληλος για αυτά τα δεδομένα.

Μεταξύ των μεθόδων που χρησιμοποιούν GNN, το *GIN-COMP* είναι το λιγότερο ικανό συνολικά, ποσοτικά. Τόσο το *GAT−* όσο και το *GCN-COMP* παράγουν παρόμοια αποτελέσματα, με το καθένα να είναι καλύτερο σε διαφορετικές περιπτώσεις. Για παράδειγμα, το GCN ξεπερνά το GAT τόσο στο ποσοστό επιτυχίας όσο και στο μέσο RBO για τα κορυφαία 5 και 10 αποτελέσματα, αλλά το GAT παράγει καλύτερες τοπ-2 προτάσεις. Μια ενδιαφέρουσα παρατήρηση είναι ότι όσο πλησιάζουμε στις κορυφαίες συστάσεις, τόσο πιο παρόμοιες γίνονται οι βαθμολογίες των GNN. Συγκεκριμένα, η παραλλαγή GIN καταφέρνει να φτάσει σχεδόν τις άλλες δύο παραλλαγές στην κατάταξη των τοπ-5 και τελικά γίνεται η καλύτερη όσον αφορά το RBO για τις καλύτερες 2 συστάσεις. Παρόλο που το *GIN-COMP* καταφέρνει να ξεπεράσει τα άλλα μοντέλα στις κορυφαίες-2 συστάσεις, η εκφραστική του δύναμη δεν τονίζεται σε αυτήν την εργασία όπως θα περιμέναμε. Όπως αναφέρεται συχνά στη βιβλιογραφία, η θεωρητική του ικανότητα δεν αντικατοπτρίζεται πάντα στην πράξη.

Συγκρίνοντας τις μεθόδους πυρήνα γράφων και GNN, καταλήγουμε στο εξής ενδιαφέρον συμπέρασμα από τον Πίνακα 1.2. Ακόμη και με 2000 ζεύγη εκπαίδευσης, τα οποία αποτελούν μόνο το 1,6 % του συνόλου των ζευγών, το GAT και το GCN υπερτερούν της καλύτερης μεθόδου πυρήνα όσον αφορά το top-10 HIT-P. Είναι προφανές ότι τα μοντέλα GNN και η εκφραστικότητά τους είναι ένα ισχυρό εργαλείο ομοιότητας. Ωστόσο, μπορούμε επίσης να παρατηρήσουμε ότι η κατάταξη δεν είναι απαραίτητα ιδανική όταν το υποσύνολο των ζευγών είναι τόσο μικρό, δεδομένων των βαθμολογιών mRBO.

Είναι ασφαλές να πούμε ότι οι μέθοδοι GNN ξεπερνούν σταθερά αυτές του πυρήνα και μπορούν να επιτύχουν σημαντικές βελτιώσεις. Για να είμαστε πιο ακριβείς, υπάρχει 11% αύξηση του ποσοστού επιτυχίας στις τοπ-10 προτάσεις μεταξύ του καλύτερου πυρήνα γράφων και της μεθόδου GNN και 7% αύξηση του mRBO. Το *GCN-COMP* παρέχει όχι μόνο ακριβέστερες προβλέψεις, αλλά και καλύτερη σχετική κατάταξη. Έχει ένα σαφές πλεονέκτημα σε σύγκριση με τις άλλες παραλλαγές με εξαίρεση τα κορυφαία-2 αποτελέσματα. Αναμφισβήτητα, οι βαθμολογίες που επιτυγχάνονται με τις μεθόδους GNN σε αυτήν την περίπτωση είναι πολύ παρόμοιες, που σημαίνει ότι οι επιδόσεις τους είναι σχεδόν ίσες. Ωστόσο, δεδομένης της σημασίας της τελικής ανίχνευσης του πιο παρόμοιου (top-1) ζεύγους γραφημάτων, ακόμη και η παραμικρή βελτίωση που επιτυγχάνεται από τις δύο άλλες παραλλαγές μπορεί να αποδειχθεί σημαντική. Στην πραγματικότητα, η παραλλαγή GIN επιτυγχάνει 2% καλύτερη πρόοδο μεταξύ των μοντέλων GNN για την εύρεση του ίδιου πιο παρόμοιου γράφου με το *BIP-GED*, με βαθμολογία 0,154.

Είναι αξιοσημείωτο ότι όλες οι μέθοδοι GNN έχουν τέλειο σκορ NDCG, ξεπερνώντας όλες τις μεθόδους πυρήνα. Αναμένεται η καλή τους απόδοση σε αυτή τη μετρική, καθώς αυτά τα μοντέλα είναι εκπαιδευμένα να μιμούνται τις βαθμολογίες ομοιότητας της βασικής αλήθειας και έτσι να δημιουργούν μια πολύ παρόμοια κατανομή.

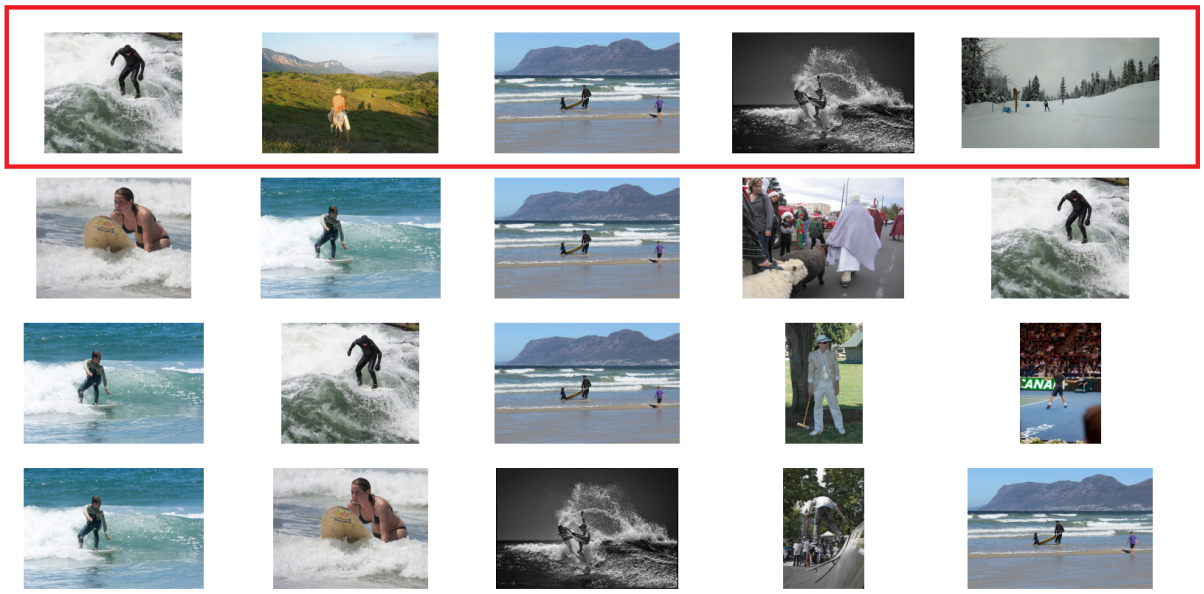
1.5.3 Ποιοτικά αποτελέσματα

Τέλος, θα εξετάσουμε ορισμένα οπτικά αποτελέσματα των κατατάξεων που παράγονται από τα τρία καλύτερα μοντέλα GNN. Στο σχήμα 1.5.8 μπορούν να βρεθούν οι κορυφαίες-5 προβλέψεις για τις γνωστές εικόνες στόχους του κοπαδιού των προβάτων και του ανθρώπου που κάνει σερφ. Υπάρχει μια σημαντική διαφορά στις συστάσεις που παράγονται από τις μεθόδους πυρήνα γράφων. Τα κορυφαία αποτελέσματα δεν κυριαρχούνται από εικόνες με το ίδιο αντικείμενο. Η ιεραρχία ή η σημασιολογία που χρησιμοποιείται για την κατασκευή της βασικής αλήθειας αντικατοπτρίζεται επίσης στα αποτελέσματα από τα GNN. Για παράδειγμα, στο σχήμα 1.5.8a συνιστώνται εικόνες με παρόμοια γραφήματα, αλλά διαφορετικά είδη ζώων. Με παρόμοιο τρόπο, στο Σχήμα 1.5.8b φαίνονται άνθρωποι σε άλλους τύπους σανίδων, π.χ. σε σκέιτμπορντ ή σε αλληλεπίδραση με άλλο αθλητικό εξοπλισμό. Αυτά τα αποτελέσματα αντιπροσωπεύουν με ακρίβεια την πολυπλοκότητα της σημασιολογίας πίσω από αντικείμενα, ενώ ταυτόχρονα λαμβάνουν υπόψη την τοπολογία του γραφήματος.

Όπως έχει ήδη τονιστεί από τα ποσοτικά αποτελέσματα, οι ταξινομήσεις που παράγονται με μεθόδους GNN συμφωνούν με τη βασική αλήθεια σε μεγαλύτερο βαθμό από τις συστάσεις των πυρήνων γράφων. Αυτή η παρατήρηση αντικατοπτρίζεται και στα ποιοτικά αποτελέσματα. Επιπλέον, ακόμη και οι συστάσεις που δεν ευθυγραμμίζονται με τα αποτελέσματα του GED φαίνεται να έχουν διαισθητικά νόημα για την ανθρώπινη αντίληψη. Για παράδειγμα, οι εικόνες ενός κοριτσιού και ενός αγοριού που κάνουν σερφ, οι οποίες συγκαταλέγονται στις κορυφαίες-2 εικόνες που προβλέπονται από τα GNN στο Σχήμα 1.5.8b, είναι πράγματι πολύ κοντά στην εικόνα στόχο. Μερικές φορές, στην πραγματικότητα, τα αποτελέσματα του GNN τείνουν να είναι πιο έγκυρα από τα αποτελέσματα της αλήθειας. Στο σχήμα 1.5.9, το *BIG-GED* προτείνει εικόνες σκαφών και υδάτινων μαζών ως τις πιο παρόμοιες με την εικόνα στόχο ενός αγοριού που βουρτσίζει τα δόντια του, ενώ το *GIN-COMP* προτείνει εικόνες με οδοντόβουρτσες σε νεροχύτες και μικρά παιδιά. Είναι αναμφισβήτητα προφανές ότι τα τελευταία αποτελέσματα είναι πιο λογικά. Ο αλγόριθμος επεξεργασίας απόστασης γράφου αγνοεί τις εικόνες των οδοντόβουρτσων επειδή οι γράφοι σκηνής τους είναι πυκνοί και περίπλοκοι, ενώ περιέχουν πολύ συγκεκριμένες ετικέτες, όπως τρίχες ή υπόλειμμα οδοντόκρεμας. Ένας αλγόριθμος επεξεργασίας βρίσκει ευκολότερο να αντιστοιχίσει γραφήματα με πιο παρόμοια δομή σε σχέση με έννοιες σε αυτήν την περίπτωση, κάτι που είναι μη διαισθητικό για τον άνθρωπο. Αυτό το παράδειγμα είναι ένας δείκτης της ικανότητας των GNN να καταγράφουν ομοιότητες τόσο στη σημασιολογία όσο και στην τοπολογία και ιδιαίτερα την ικανότητα του GIN να κατανοεί καθολικές έννοιες.

Κατά την εξέταση των οπτικών αποτελεσμάτων, δεν υπάρχει άμεσος τρόπος να αποφασιστεί ποια παραλλαγή GNN είχε καλύτερη απόδοση. Μια παρατήρηση είναι ότι το *GIN-COMP* δεν ήταν τόσο συνεπές στην παροχή λογικών συστάσεων. Συγκεκριμένα, μερικές φορές ήταν εξαιρετικά ακριβές, ενώ άλλες συνιστούσε φαινομενικά

(a) Κορυφαία-5 αποτελέσματα για εικόνα κοπαδιού προβάτων.



(b) Κορυφαία-5 αποτελέσματα για εικόνα άντρα σε σανίδα σερφ.

Figure 1.5.8: *GNN-COMP* κορυφαία-5 αποτελέσματα για εικόνες στόχους.

Η βασική αλήθεια κυκλώνεται με κόκκινο ορθογώνιο. Οι προβλέψεις των μεθόδων πυρήνα ακολουθούν τη σειρά GCN-, GAT- και GIN-COMP από πάνω προς τα κάτω.

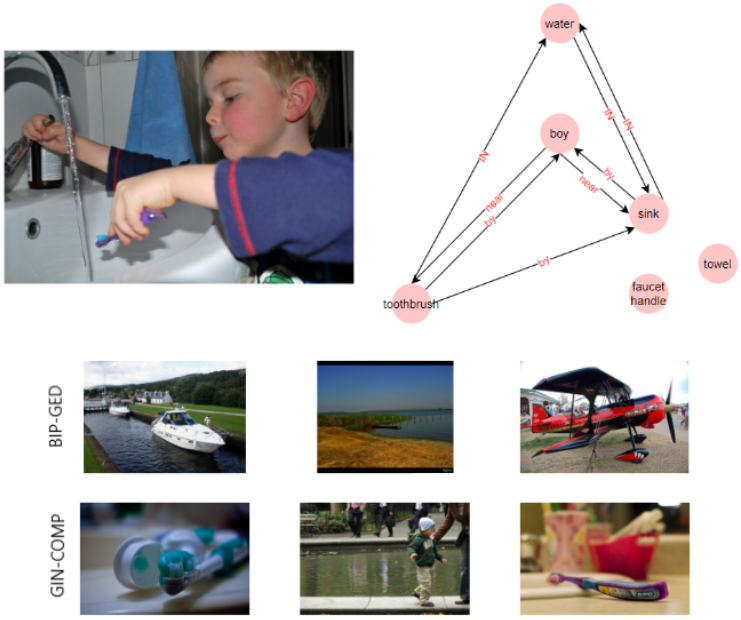


Figure 1.5.9: Παράδειγμα μοντέλου GNN που παρέχει καλύτερα κορυφαία-3 αποτελέσματα από την αλήθεια.

τυχαίες εικόνες. Πρέπει να σημειωθεί, ωστόσο, ότι το τελευταίο φαινόμενο ήταν σπάνιο.

Το τελευταίο πράγμα που πρέπει να έχουμε κατά νου είναι ότι αυτά τα πειράματα πραγματοποιήθηκαν σε ένα μικρό υποσύνολο των γράφων του Visual Genome. Έτσι, ορισμένες εικόνες και τα αντίστοιχα γραφήματα δεν είχαν καλές αντιστοιχίσεις. Οι αλγόριθμοι που δοκιμάστηκαν μπόρεσαν να αποδώσουν ικανοποιητικά και σε αυτήν την περίπτωση, προτείνοντας τις πιο παρόμοιες σκηνές. Ωστόσο, ένας άνθρωπος δεν θα θεωρούσε απαραίτητα τις σκηνές που απεικονίζονται ως παρόμοιες. Για παράδειγμα, στην Εικόνα 1.5.10 βλέπουμε ένα κορίτσι να τρώει μια πίτσα. Παρά τις πίτσες και τα παιδιά που υπάρχουν στο υποσύνολο των γραφημάτων που χρησιμοποιούνται, οι σκηνές που συνδυάζουν αυτές τις δύο έννοιες είναι ανύπαρκτες. Στην πραγματικότητα, η πιο κοντινή εικόνα είναι αυτή ενός πουλιού που τρώει φαγητό από ένα πιάτο. Δύο στα τρία μοντέλα αποτυγχάνουν να βρουν αυτή την εικόνα και αντ' αυτού εστιάζουν σε μεμονωμένες πτυχές της. Οι προβλέψεις που ακολουθούν την κορυφαία αποτελούνται κυρίως από φαγητό σε τραπέζια ή είναι οπτικά άσχετες, ακόμη και για την GED.

1.6 Συμπεράσματα

1.7 Συζήτηση

Πειραματιστήκαμε με τρεις διαφορετικές παραλλαγές συνελικτικού GNN για το προτεινόμενο μοντέλο μας και συγκρίναμε την αποτελεσματικότητα, την πολυπλοκότητα και την εκφραστικότητά τους. Διαπιστώσαμε ότι όλες τους, ποσοτικά και ποιοτικά, ξεπερνούν τις μεθόδους πυρήνα γραφου στο πρόβλημα της ομοιότητας γράφων και προσφέρουν εύχρηστες και ουσιαστικές αναπαραστάσεις για δομές γραφήματος. Εμπλουτίσαμε τους γράφους με χαρακτηριστικά κόμβου και με έκπληξη διαπιστώσαμε ότι οι κλασικές ενσωματώσεις GloVe ήταν πιο ωφέλιμες για όλα τα μοντέλα *GNN-COMP* παρά τα ιεραρχικά path2vec, που εκπαιδεύτηκαν στο WordNet. Μεταξύ των παραλλαγών, μπορέσαμε να διαπιστώσουμε ότι το GCN και το GAT είχαν συγκρίσιμες επιδόσεις στις χρησιμοποιούμενες μετρικές, το ποσοστό επιτυχίας και το μέσο RBO, για τα κορυφαία-10 και κορυφαία-5 αποτελέσματα. Ωστόσο, όσον αφορά τα πιο παρόμοια γραφήματα, όλοι πέτυχαν εξαιρετικά παρόμοια αποτελέσματα, με το GIN να παράγει το καλύτερο. Στην πραγματικότητα, το *GCN-COMP* πέτυχε σχεδόν 11% αύξηση σε σχέση με τον πυρήνα Pyramid Match σε ποσοστό επιτυχίας στα τοπ-10 αποτελέσματα και το *GIN-COMP* έφτασε στο 15,4 % στην πρόβλεψη της ίδιας ακριβώς εικόνας με την προσέγγιση GED που αποτελούσε την βασική αλήθεια. Παρά τη φήμη του GIN ως ένα από τα πιο ισχυρά GNN, μπορέσαμε να δούμε μόνο ένα κομμάτι

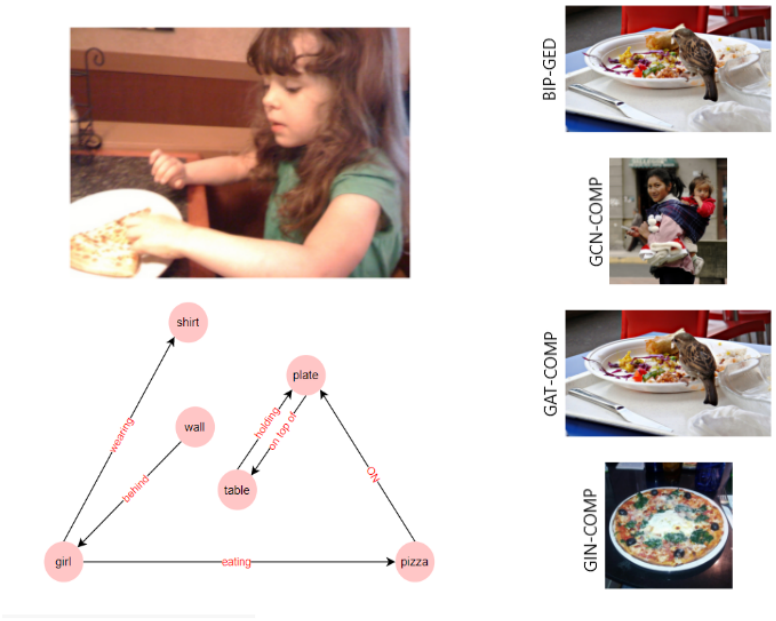


Figure 1.5.10: Παράδειγμα ανόμοιας εικόνας με το υπόλοιπο σύνολο δεδομένων.

από τις ικανότητές του. Κατάφερε να συλλάβει το γενικό πλαίσιο με αποτελεσματικό τρόπο. Παρόλα αυτά, συνολικά, όλες οι παραλλαγές παρήγαγαν κάτι παραπάνω από ικανοποιητικά αποτελέσματα. Είναι αξιοσημείωτο ότι το GCN το έκανε στον μισό χρόνο.

Οι ταξινομήσεις που παράγονται σε αυτή τη διατριβή έγιναν με σκοπό να βοηθήσουν στον αποτελεσματικό και πιο ουσιαστικό υπολογισμό των εξηγήσεων με αντιπαράδειγμα. Συγκεκριμένα, η ανίχνευση του πιο παρόμοιου ζεύγους μιας εικόνας μας δίνει την ευκαιρία να υπολογίσουμε μόνο μία επεξεργασία ή εξήγηση, αντί για όλες τις πιθανές. Ως εκ τούτου, η προτεινόμενη προσέγγιση, η οποία χρησιμοποιεί τον αντίστοιχο γράφο σκηνής μιας εικόνας για την ανάκτηση του καλύτερου ζεύγους, εισάγει μια σημαντική επιτάχυνση σε αυτή τη διαδικασία, αξιοποιώντας παράλληλα τις πλούσιες πληροφορίες που παρέχει η δομή του γραφήματος σχετικά με τις σχέσεις μεταξύ των εννοιών που απεικονίζονται. Επιπλέον, ο τελικός χρήστης μένει με ένα επαγωγικό μοντέλο, που σημαίνει ότι το εκπαιδευμένο *GNN-COMP* μπορεί να αξιοποιηθεί για την παραγωγή ενσωματώσεων για καινούριους γράφους σκηνής.

Συμπερασματικά, μπορούμε να επιβεβαιώσουμε ότι η χρήση των GNN όχι μόνο για τον υπολογισμό της εγγύτητας γραφήματος, αλλά και για τη δημιουργία αναπαραστάσεων γράφων είναι εξαιρετικά επωφελής. Μέσα από τα πειράματά μας συνειδητοποιήσαμε την εκφραστική δύναμη των GNN, τη σημασία της αρχικοποίησης των χαρακτηριστικών του κόμβου και τη διαφορετική απόκρισή τους στον αριθμό δειγματων εκπαίδευσης. Ήμασταν σε θέση να συνδυάσουμε τις έννοιες των γράφων σκηνής και των Νευρωνικών Δικτύων Γράφων για να προτείνουμε μια νέα προσέγγιση Ομοιότητας Γράφων με σκοπό τελικά να βοηθήσουμε στη δημιουργία εξηγήσεων με αντιπαράδειγμα. Οι αναπαραστάσεις που λαμβάνονται μπορούν να αποδειχθούν πολύτιμες και για πολλές άλλες εργασίες κατάντη, κατεύθυνση την οποία ελπίζουμε να εμπνεύσουμε.

1.8 Μελλοντικές Κατευθύνσεις

Κλείνοντας αυτή τη διατριβή θα θέλαμε να προτείνουμε μερικές κατευθύνσεις για περαιτέρω βελτίωση αυτής της εργασίας ώστε να εμπνεύσουμε διαφορετικές ενδιαφέρουσες προσεγγίσεις. Πρώτον, θα μπορούσε να διερευνηθεί η χρήση περισσότερων παραλλαγών Νευρωνικών Δικτύων Γράφων, όπως παραλλαγές που αξιοποιούν πληροφορία στις ακμές. Με αυτόν τον τρόπο, θα αποκτήθούν πληροφορίες σχετικά με τον τύπο των σχέσεων μεταξύ των αντικειμένων και θα δημιουργηθούν πλουσιότερες ενσωματώσεις. Μια άλλη προσέγγιση θα ήταν να δοκιμαστούν παραλλαγές όπως οι Graph Autoencoders που ενσωματώνουν τα γραφήματα με μη επιβλεπόμενο τρόπο, ή ακόμα

και παραλλαγές αυτο-επίβλεψης με βάση την αντίθετική μάθηση και να συγκριθεί η απόδοσή τους με τις τεχνικές που χρησιμοποιήσαμε εμείς. Όσον αφορά τους γράφους, η επιλογή των αρχικών χαρακτηριστικών αποδείχθηκε να είναι κρίσιμη για την απόδοση των μοντέλων, επομένως θα μπορούσαν να χρησιμοποιηθούν διαφορετικές ιεραρχικές ενσωματώσεις. Επιπλέον, θα μπορούσε να πραγματοποιηθεί μια λεπτομερής στατιστική και οπτική ανάλυση των ζευγών γράφων εκπαίδευσης προκειμένου να διαπιστωθεί πώς γραφήματα με διαφορετικές ιδιότητες επηρεάζουν τα μοντέλα GNN. Τέλος, ο σχεδιασμός μιας προσαρμοσμένης απώλειας για ομοιότητα θα μπορούσε επίσης να βοηθήσει στην ικανότητα του μοντέλου να παράγει πιο ακριβείς κατατάξεις ομοιότητας.

Οι ενσωματώσεις γραφημάτων που παράγονται θα μπορούσαν να χρησιμοποιηθούν για άλλες εφαρμογές εκτός από τον υπολογισμό εξηγήσεων με αντιπαράδειγμα. Συγκεκριμένα, η ιδέα της ομοιότητας θα μπορούσε να οριστεί εκ νέου για άλλες εργασίες. Οι γράφοι σκηνής θα μπορούσαν να συνδυαστούν με άλλες μεθόδους για την αποτύπωση ιδιοτήτων όπως το χρώμα, το μέγεθος ή τη θέση των αντικειμένων στην εικόνα με σκοπό τον εμπλουτισμό των ενσωματώσεων. Επιπλέον, δεδομένου ότι αυτή η προσέγγιση ανιχνεύει τα περισσότερα παρόμοια γραφήματα σε ένα δεδομένο σύνολο δεδομένων, θα μπορούσε να χρησιμοποιηθεί για άλλες εργασίες τύπου ανάκτησης και να επεκταθεί για να λειτουργήσει σε γραφήματα γνώσης ή γραφήματα εγγράφων. Τέλος, θα μπορούσε να αναπτυχθεί μια μέθοδος αποτελεσματικής ομαδοποίησης ενσωματώσεων για την εξαγωγή πληροφοριών σχετικά με τις εγγενείς ιδιότητες που τοποθετούν αυτά τα διανύσματα πιο κοντά μεταξύ τους στο χώρο. Η ιδέα της ομαδοποίησής τους με ουσιαστικό τρόπο θα οδηγούσε επίσης στο κλάδεμα των γράφων που θεωρούνται παρόμοιοι εισάγοντας σημαντική επιτάχυνση. Τέλος, αυτή η εργασία θα μπορούσε να εμπνεύσει την εξερεύνηση του αντίστροφου προβλήματος, δηλαδή να απαντήσει στο ερώτημα γιατί διαφορετικές παραλλαγές GNN πιστεύουν ότι ορισμένα γραφήματα είναι παρόμοια.

Chapter 2

Introduction

In recent years, advances in computing power and storage as well as deep learning algorithms have resulted in an impressive infiltration of artificial intelligence applications in people's every day lives. The increasing interference of intelligent systems in the way people consume media and entertainment, create art, purchase products or even receive healthcare creates the inevitable need to build trust with these tools. Thus, providing explanations on why AI systems make their decisions has become utterly important.

The field of Explainable AI aims to provide insight on Machine Learning "black box" models and explain the reason behind their actions. One of the more interesting techniques used for this task are counterfactual explanations which provide feedback on how the input data could have been altered in order for a model to make different decisions. Those changes are often sought to be minimal and can be of great assistance in detecting biases.

In this thesis, we attempt to find the most similar and therefore least semantically distant instances in a dataset of images as a step of building counterfactual explanations. We focus on comparing the scenes portrayed in the images and not the counterfactual explanation itself. This work is directly inspired by [22], in which instances are compared based on conceptual edits. Provided that the input images contain scenes and are also available in scene-graph format, we compare them using their corresponding graph representation. The extra information regarding the relations between present objects, naturally provided by the graph structures, makes this approach semantically richer.

The prevailing method to determine graph similarity is through graph kernels. In this work, we will directly compare kernel approaches and classic algorithms which determine the edit distance of graphs with neural approaches. The focal point is the use of Graph Neural Networks (GNN) for accomplishing this task.

This thesis provides a thorough exploration of Graph Neural Network variants and highlights and compares their expressive power on graph data. It underlines the use of GNN models in the task of graph similarity, which has been mostly tackled using non-neural methods. Furthermore, we attempt to showcase the combination of GNNs with scene graph data for a task other than their generation, which is the most prevalent in related literature. Finally, through the process of finding the most similar scene graphs, we are able to extract embeddings. These meaningful representations are able to capture the semantic similarity of the input data based on graph edit distance, which they are trained on. The embeddings can then be utilized as graph representations for other downstream tasks.

The outline of this thesis is as follows:

- We will firstly provide all the background needed in basic Machine Learning algorithms and concepts as well as scene graphs in order to be able to explain and justify the idea of Graph Neural Networks. After doing so, we will provide a thorough description of GNN variants relevant to this work.
- We will give a more detailed definition of Counterfactual Explanations and related work, focusing on the ideas presented in [22]. In a similar fashion, we will formally explain the task of graph similarity and provide the theoretical background for methods already used to tackle it, such as Graph Kernels

and Graph Edit Distance algorithms.

- Lastly, we will propose our GNN-based model for graph similarity and highlight the performance of different variants used for its layers. We will compare these results with the more conventional methods described in previous sections and draw conclusions based on their expressiveness and evaluation on quantitative and qualitative results.

Chapter 3

Machine Learning

The topic of machines with human-like capabilities such as thinking has been prevalent among engineers for centuries. In more recent years, after the invention of computers, the field of Artificial Intelligence (AI) has not only been established but also thriving. According to [27] Artificial Intelligence, or the development of computer systems which tackle tasks normally requiring human intelligence, at first focused on solving problems difficult for humans, but elementary for computers. However, it soon became apparent that the opposite problem is also in need of attention, i.e. teaching machines to solve problems which for human beings are intuitive due to their acquired experience.

Machine Learning (ML) is a branch of AI which employs empirical or historical data in order to make predictions with statistical models. In a broader sense, as the name implies, it helps a machine learn through statistics in order to build its experience level without needing to be explicitly programmed to do so [91]. These models perform a type of predictive analysis, processing data using different algorithms, extracting patterns from raw data and finally generating outputs or predictions for a number of tasks.

Machine Learning tasks vary from relatively simple to more convoluted or subjective. For example, some basic tasks involve classifying or grouping data in categories, while others include generating unique brand new data, like images. The more intuitive the task the more challenging it tends to be for computers. The difficulty of the task is also related to the type of input data, which can be images, text, speech, timeseries or even graphs. According to the data representation, a task is grouped with other similar ones belonging to the same field. The most popular ones are Computer Vision, Natural Language Processing, Speech Recognition, Recommendation Systems and more recently Geometric Learning. The task itself is most of the times strongly connected to the machine learning category, as explained in the next section.

Contents

3.1 Learning Categories

Machine Learning algorithms can be categorized according to the experience the model is provided with during training. The three broad types of machine learning algorithms are supervised, unsupervised and reinforcement learning, while semi- and self- supervision can be considered variants of the above.

Supervised Learning

In this type of learning the input data, comprised of multiple features, is associated with a label or target. Let the feature vector be x and the target be y, the model will learn to predict y from x using an array of examples. In most cases, this is achieved by learning the distribution function $p(y|x)$. Some of the most prominent supervised learning tasks are considered to be classification, regression and forecasting.

Unsupervised Learning

Unsupervised learning algorithms do not take labels attached to feature vectors of the dataset into account. Instead, they attempt to implicitly or explicitly learn the probability distribution of the entire dataset $p(x)$ and in turn give insight about the underlying structure of data. Popular tasks of this category include clustering and dimensionality reduction.

Semi-supervised Learning

Semi-supervised learning can be considered a special type of supervised learning where most samples of the dataset used for training are not associated with targets. The small amount of labeled data can be attributed to either difficulty of acquiring said information or desire for better accuracy [95]. Common semi-supervised learning tasks include link prediction in graphs or fraud detection.

Self-supervised Learning

Self-supervised learning is a learning category lying between supervised and unsupervised. It makes use of unlabeled data and leverages supervision signals stemming from the structure of the data itself, by using pseudolabels. This form of learning mostly consists of solving pre-text tasks, i.e. tasks specifically crafted to help a model learn the inner-workings of a dataset, and using the rich information obtained by the originally unlabeled data to later solve other downstream tasks [51], like the ones mentioned in previous sections.

Reinforcement Learning

In reinforcement learning the dataset is not fixed, but rather it receives feedback from changes in its environment. This type of learning will not be considered in this thesis.

3.2 Training a Neural Network

Neural Networks are a subset of machine learning, whose operation resembles that of the human neural system. In this section the general idea of the learning process of such a system will be described as well as the potential challenges.

3.2.1 Basic Concepts

Firstly, we will explain the basic operation of a shallow neural network to provide background information needed to understand similar architectures on graph structured data. The following concepts mostly adhere to supervised learning tasks, but are broadly relevant for most approaches described later on in this thesis. The majority of information below was sourced from [27].

In supervised learning, the model uses a fixed amount of N samples from the training dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ and their corresponding labels to compute a function $f : X \to Y$ which maps the input X to the output Y and its trainable parameters are often called weights. In order to evaluate f, a

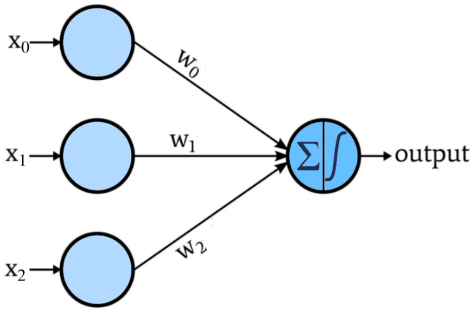


Figure 3.2.1: Shallow Neural Network of One Neuron - Perceptron [39]

function is established to determine the error of training, called loss function L. The goal during training is the calculation of the weights of f in a way that ensures the minimization of the loss function .

The output of each neuron in a neural network is not solely determined by the computation of the weighted sum of the inputs x_i. On the contrary, a different function is employed called **activation function** to define how to transform the weighted sum to an output. An activation function plays a critical role in the network's performance and therefore should be chosen carefully. It can be linear or non-linear and the most commonly used examples can be seen below.

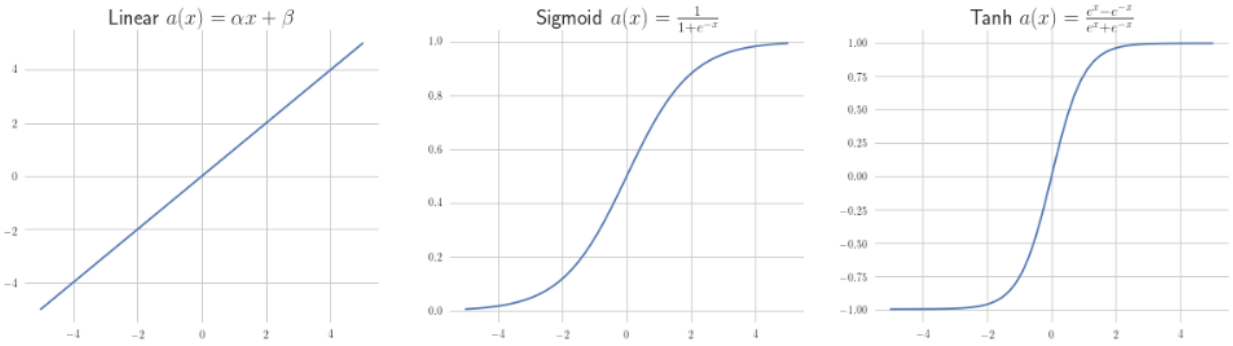


Figure 3.2.2: Examples of activation functions.

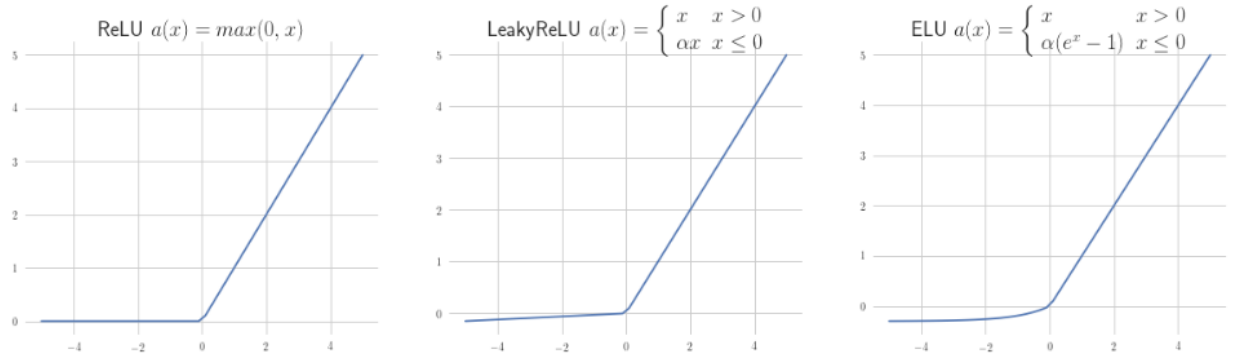


Figure 3.2.3: ReLU activation function and variants.

The linear activation function exhibits a number of limitations, one of the most important being the fact that it makes backpropagation, which will be explained later, impossible since its derivative is a constant. It leaves the weighted sum of the input practically unchanged. In most cases, non-linear activation functions

are used due to the fact that they ultimately allow the stacking of multiple layers (deep networks) and the creation of more complex mappings. Sigmoid and tanh, although being popular non-linearities for certain problems, face the problem of vanishing gradients [96], making training unstable.

The most widely adopted choice of activation function is ReLU and its variants. It overcomes the problems of aforementioned functions and leads to more computationally efficient training. Some of its variants include LeakyReLU and ELU which can be seen in Figure 3.2.3. ReLU also suffers from limitations, such as the dying ReLU problem which causes the existence of non-active neurons [94]. This problem is combated by its variants.

As mentioned previously, the **loss function** or cost function is used to determine how close the model's prediction is to the truth. It maps $Y \times Y$ to a non-negative real number, or $L(y_i, f(x_i))$ for the i_{th} sample. Neural models are trained repeatedly for a number of iterations, called epochs, until meeting an objective or when the maximum amount of iterations has been reached. In general, the total loss of the model in each epoch can be defined as a normalized average of the cost function for each piece of data on the training set. In an optimal scenario, the parameters which minimize this function will be discovered.

The choice of cost function heavily depends on the task carried out. The most notable loss functions are listed below but it is important to note that often custom loss functions are created to cater for more complex tasks.

Mean Squared Error is one of the simplest cost functions and it is often used in regression tasks.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{n} \tag{3.2.1}$$

Mean Absolute Error or *L1 loss* is similar to MSE in the aspect that it ignores the direction of error [92]. This cost function is more robust to outliers.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - f(x_i)|}{n} \tag{3.2.2}$$

Cross Entropy Loss or *Negative Log Likelihood* is a loss function commonly used in classification tasks. It focuses on penalizing predictions that are confident but incorrect and in the case of non-binary classification in M classes can be defined as:

$$NLL = -\sum_{c=1}^{M} p_{y_i,c} log(p_{f(x_i),c}) \tag{3.2.3}$$

The next step after the establishment of the cost function is its minimization. In order to achieve optimization, **gradient-based** methods are employed. The use of gradients is crucial for such problems since derivatives can give us insight on how to scale small changes in the input so as to eventually reach the desirable output [27]. More specifically, the objective function is minimized by iteratively computing the value of the loss function for all training samples and in turn the gradients of the model's parameters and finally updating the parameters in the opposite direction of the gradient.

The most popular gradient method, which adopts the aforementioned process, is *Gradient Descent*. Its definition can be found in Equation 3.2.4, where ϑ represents the model's parameters and ϵ corresponds to the learning rate. The learning rate is a small positive constant which is chosen during training, often by trying several values and keeping the best. Its choice can prove crucial to the model's performance.

$$\theta' = \theta - \epsilon \nabla_\theta L(\theta) \tag{3.2.4}$$

There are many other gradient based optimizers often used in applications, most of which are extensions or inspired by gradient descent. One of the most popular ones is *Stochastic Gradient Descent* (SGD) which differs in the way that it computes gradients and updates parameters for each (x_i, y_i) pair in batches, instead

of the whole dataset. This practice gives it a significant speed advantage. Other honorable mentions include *AdaGrad, RMSProp, AdaDelta* and *Adam*.

Even though the computation of an analytical expression for gradients is simple, its numerical evaluation is quite expensive. For this reason the algorithm of **back-propagation** was proposed [73]. In this approach the chain rule needed to be applied for gradients is computed in a specific order of operations that is highly efficient, making use of computational graphs and storing already computed values.

The process of computing outputs and adjusting weights through back-propagation is often repeated for several epochs until the loss function converges or another threshold is met. The neural network's performance can then be evaluated using a variety of metrics and data which was not encountered during training.

3.2.2 Generalization and Overfitting

A machine learning algorithm must be able to succeed easily on previously unseen data, different from those it was trained with. The ability of a model to perform well on new inputs drawn from the same distribution as the one used to create it is called generalization. To achieve good generalization an algorithm must result in both low train and test error, where train error is defined as the loss during training and test error is the loss obtained from newly observed data, after the training process.

The lack of generalization of a model is often attributed to underfitting or overfitting. Underfitting is caused by high train error, meaning the model has not sufficiently learnt the data distribution whereas overfitting is the product of a large gap between train and test error, given the train error is low. The latter exposes the fact that the model has learnt the data too well, including existing noise, thus having a negative impact on its performance.

It is apparent that the training of a neural network is not just a simple optimization problem, but rather the pursuit of a good trade-off between test and train error. The most prominent technique to handle overfitting is regularization. The concept of regularization is based on Occam's razor, the notion that the simplest solution must be chosen or in this case the smallest in way of parameters. This idea is implemented by adding an extra term $\lambda R(\theta)$ in the loss function which penalizes larger more complex models, while favoring low values of loss. The hyperparameter λ of the term is chosen during training in order to favor the model's performance and R is the regularization function which is often a norm of the weights. The two most prevalent regularizers are L_1 and L_2 norm. In the equations below W represents the weight matrix of the model.

L_1 *Regularization* or *Lasso Regularization* tends to favor sparse solutions, i.e. solutions containing many zero values, by penalizing both uniformly low and high parameter values.

$$R_{L_1}(W) = \sum_{i,j} |W_{i,j}| \tag{3.2.5}$$

L_2 *Regularization* or *Ridge Regularization* punishes high values heavily. It is often referred to as weight decay.

$$R_{L_2}(W) = \sum_{i,j} W_{i,j}^2 \tag{3.2.6}$$

Finally, another way of achieving generalization is by using *dropout*. With this technique, a number of layer outputs are randomly ignored in order to ensure that the model does not rely on specific neurons. By doing so, noise is added to the training process which in turn makes the neural network more robust.

3.3 Deep Learning

With the knowledge of all the basic steps of training a shallow neural network, some principal models will be explained in this section. All of them are examples of Deep Learning algorithms, i.e. neural networks in which multiple layers of neurons are used in order to extract progressively higher level features from the input data. Even though the complexity of said algorithms is increased, the general idea remains the same.

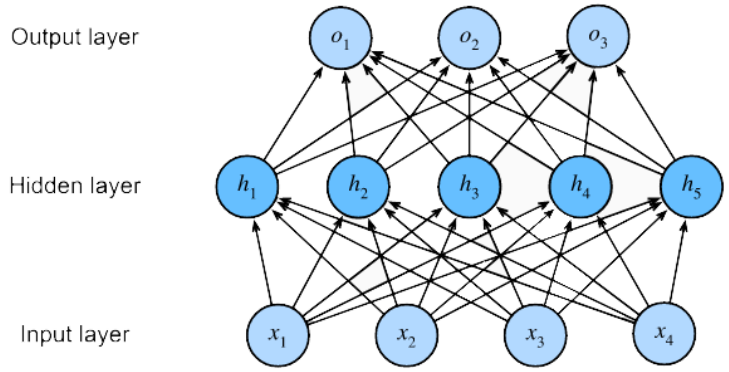


Figure 3.3.1: Multi-layer Perceptron with one hidden layer of 5 units. [105]

3.3.1 Multi-Layer Perceptron (MLP)

The concept of the Multi-Layer Perceptron was introduced as a way of overcoming the limitations of linear models. Specifically, linear models imply monotonicity, i.e. that any change in features always causes change in the same direction for the model's output - decrease or increase [105]. This generic assumption is not true for the majority of problems to be solved. In order to tackle this obstacle, the stacking of multiple fully connected layers was proposed.

The MLP, often referred to as a Feed Forward Neural Network (FFNN), is an artificial neural network in which the connections between nodes do not form a cycle [89]. The information is only processed in one direction from the nodes of the input layer, through the hidden nodes to the output layer nodes. As explained in previous sections, each node computes the weighted sum of inputs and produces an ouput using an activation function, which in this case needs to be non-linear. Otherwise, since the sum of two linear functions is also linear the existence of layers would be unnecessary.

The simplest MLP contains only one hidden layer and is called a *single-layer perceptron*. An example can be seen in Figure 3.3.1. The input layer comprises of the input vectors left unchanged, each of which are passed to all the neurons in the hidden layer. The hidden layer processes the inputs it is given and extracts features from them. If multiple layers exist, the closer the hidden layer is to the output the higher-level the features it extracts. The output layer computes the output of the model by using the processed data passed from the previous layer.

There is no limitation to the number of layers or number of units in each layer. This is a matter of experimentation during the training process.

3.3.2 Convolutional Neural Networks (CNN)

Multi-layer Perceptrons are useful for dealing with tabular data, i.e. data in the form of rows and columns which consist of a number of samples and their corresponding figures. However, if the input data consists of high-dimensional data such as images, an appropriate MLP would have to be enormous in size with millions of parameters. Considering this and the fact that visual data exhibits interactions between features due to the locality of pixels, a new type of neural network was introduced, the Convolutional Neural Network (CNN) [47].

CNNs are rooted in the idea of using a new type of layer which performs convolutions. A *convolutional layer* contains a set of trainable filters. These filters are convolved or in practice cross-correlated with input from the previous layer producing what is called a feature or activation map. They are generally small in size, leading to lower numbers of parameters, since they make the assumptions of translation invariance and locality, which are true for images. The process of cross-correlation consists of simply computing the dot product between filters and the input across both dimensions and subsequently producing a 2-dimensional activation map of that filter. The network is able to learn filters corresponding to different features in specific positions of the input. The local focus of the CNN also aids in the problem of overfitting.

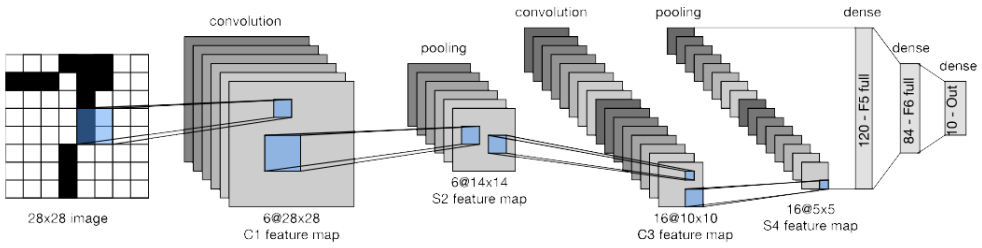


Figure 3.3.2: Typical CNN architecture - LeNet. [105]

In a typical CNN the convolutional layers are followed by pooling layers. The act of pooling serves the purpose of both alleviating the local sensitivity of convolutional layers and spatially downsampling representations [105]. Pooling filters are non trainable and perform the deterministic operation of aggregating elements present in a fixed-size window of the feature maps they receive. The types of aggregation most commonly used are the computation of maximum or average, resulting in the so-called max and mean pooling filters respectively.

In Figure 3.3.2 we can see the architecture of the first and simplest CNN. Nowadays, the CNNs used in applications are much deeper, meaning they consist of tens of alternating convolutional and pooling layers, followed by fully connected -or dense- ones. The dense layers contribute in learning combinations of the features extracted and serve as a classification head.

3.4 Embedding

A term which we will refer to a lot in this dissertation is *Embeddings*. An embedding is a relatively low-dimensional space used for the translation of higher-dimensional vectors. The goal of embeddings is to bring semantically similar objects closer together in the embedding space, by effectively capturing the semantics of the input. The dimensionality reduction offered by this method makes machine learning easier if the original input comprises of sparse high-dimensional vectors [17].

Embeddings are most commonly used in the field of natural language processing (NLP), serving as word or sentence representations. They generally are real-valued vectors representing the semantic meaning of words in such a way that maps words with similar meaning closer together in the vector space [97]. The generation of embeddings is possible through an array of methods such as dimensionality reduction on the word co-occurrence matrix, probabilistic models or even neural networks. The most popular approaches include Word2vec [57], GloVe [67], BERT [69] and Principal Component Analysis (PCA).

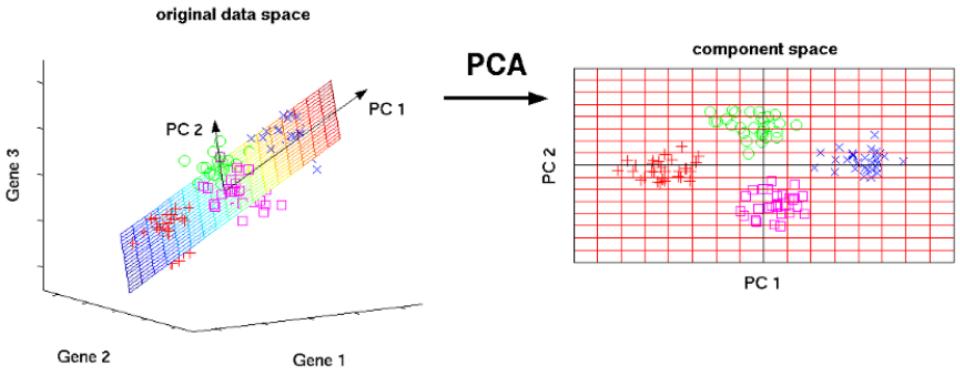


Figure 3.4.1: Visualization of Embedding using PCA. [36]

In the sections below we will explain some of the most notable embedding methods:

Principal Component Analysis

PCA is a standard mathematical technique used for dimensionality reduction. It focuses on finding dimensions of the input data which are highly correlated and therefore can be represented as one. PCA is defined as an orthogonal linear transformation which maps data to a new coordinate space. The dimensions of this space are computed as the eigenvectors corresponding to the greatest eigenvalues of the covariance features matrix. These vectors are chosen to be orthogonal, meaning the features are independent, and result in the least amount of error for approximating the data.

PCA captures linear correlations and therefore if the dataset has underlying non-linearities this approach will fail. Although simple in conception, if not performed appropriately it can lead to information loss [93].

Autoencoder

The *Autoencoder* is a dimensionality reduction method using neural networks. It succeeds in overcoming PCA's limitations since it basically is a non-linear generalization of it. Specifically, if it were to be linear it would produce any orthogonal basis in a non-deterministic way.

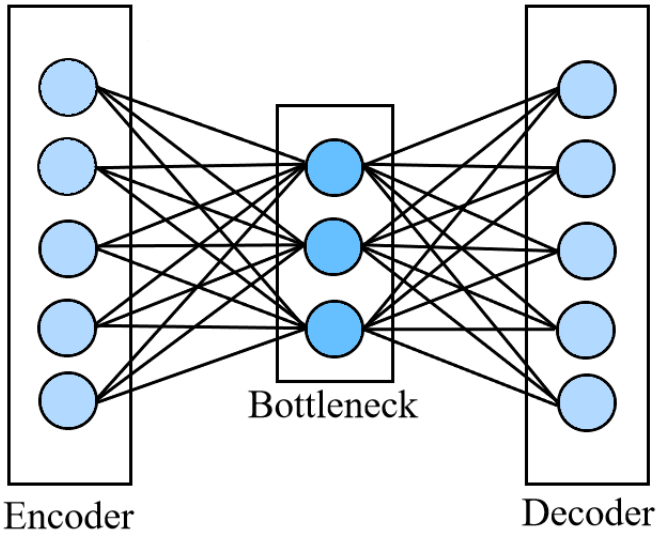


Figure 3.4.2: Autoencoder Architecture.

The autoencoder is a bottleneck architecture whose framework is composed by an *Encoder* and a *Decoder*. The first component transforms the high-dimensional input into a latent low-dimensional code, whereas the second reconstructs the input data using this code [33]. Both components are neural networks which are iteratively optimized using a custom loss function, called reconstruction loss, which computes the distance between the input and reconstructed data by the decoder.

The encoder and decoder can be stacks of different types of layers, even convolutional if the input is an image. There are various different models adopting the autoencoder architecture such as the undercomplete, sparse, contrastive and variational autoencoder. These models primarily differ on the construction of the loss function. For example, in the Variational Autoencoder there is a regularization term aiming to tackle overfitting. This feature combined with the encoding of the input as a distribution - and not points - makes this particular variant well-suited not only for the embedding of existing data but also for the generation of new ones.

This architecture provides a trainable way of embedding data. Its most important feature is the idea of the bottleneck, which only lets vital semantic information go through and eventually be reconstructed. Control of the bottleneck layer by size reduction helps to alleviate overfitting. The choice of depth and dimensions is also crucial.

Trained Embeddings as Part of a Larger Model

Another way of producing embeddings is implicitly while training a neural network for another task. A way to achieve this is by adding a special type of layer, defined by the library used to train, with dimension d in order to create a d-dimensional embedding. Alternatively, the embeddings could be extracted by any other given layer deemed fit for capturing data semantics.

This approach provides specific embeddings which capture semantic similarity of the input data determined by the training task. In many applications, like the one presented in this thesis, this is a desirable trait. However, in most cases the training of the larger model is more computationally expensive than the training of embeddings separately [17].

Chapter 4

Graphs

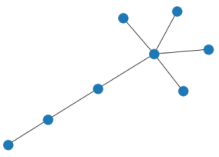
4.1 Graph Theory Basics

A graph, denoted G, is a non-linear data structure comprising of a set of objects called nodes or vertices which can be connected to one another, forming an ordered pair called an edge. In other words:

$$G(V, E) = \{(u, v) : u, v \in V, (u, v) \in E\} \tag{4.1.1}$$

where V is the set of vertices and E is the set of edges with $|V| = N, |E| = M$. In literature, the term "graph" is often used as a synonym for a simple graph, i.e. a graph without any self-loops (edges connecting a vertex to itself) and no more than one edge connecting any pair of vertices.

A visual representation of a graph can be seen in Figure 4.1.1a. However, graphs can also be described using an adjacency matrix A, a square array of dimensions $N \times N$ whose non-zero elements indicate the existence of a link between vertices. In some cases links between nodes can be assigned weights, which hold a relevant meaning to what the graph represents. Thus, the weight value for the corresponding edge would be present for each node pair in the adjacency matrix.



(a) Visual Representation

```
[[0 1 0 0 0 0 0 0]
 [1 0 1 0 0 1 1 1]
 [0 1 0 0 0 0 0 0]
 [0 0 0 0 1 1 0 0]
 [0 0 0 1 0 0 0 0]
 [0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0]]
```

(b) Adjacency Matrix

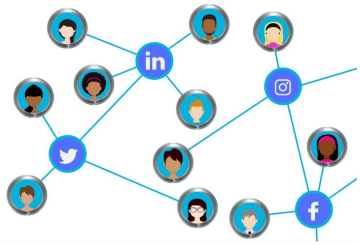
Figure 4.1.1: Representation of undirected graph.

A graph may additionally have node and/or edge attributes. In this case, each node (edge) is characterized by a feature vector of dimension D, resulting in a node (edge) feature matrix X of dimension $N \times D$ (X^e of dimension $M \times D$).

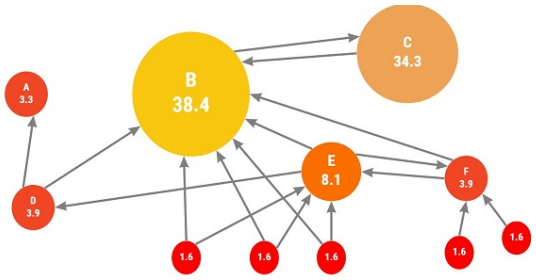
Graphs are categorized according to the direction of connections present between nodes. More specifically, an edge between nodes u, v is called undirected if both ordered pairs (u, v) and (v, u) belong in the set of edges if u and v are connected whereas is called directed if only one of them is present. Therefore, a graph is said to be undirected when all its edges are undirected and directed (or digraph) if at least one of them is not. A property of an undirected graph is the symmetry of its adjacency matrix.

The neighborhood $N(u)$ of an entity u in a graph is defined as the set of nodes adjacent to it. A path is a sequence of vertices $u_1, u_2, ..., u_n$ such that u_i and u_{i+1} are neighbors for all $1 \leq i \leq n - 1$. An undirected

(a) Molecule [25]



(b) Social Network [72]



(c) PageRank on websites [62]

Figure 4.1.2: Graph reprentation examples

graph is connected if every pair of nodes are connected through a path. In the case of digraphs, there are two distinct versions of connectedness, weak and strong. A directed graph is weakly connected if for every pair of vertices u and v there exists a path either from u to v or from v to u. In contrast, a strongly connected digraph should contain paths leading both ways.

A distinct feature of graphs that sets them apart from other types of data is the fact that they are generally non-euclidean. Non planar graphs cannot be mapped on a two-dimensional, non-curved space because they cannot be drawn on the plane so that links intersect only at their endpoints. This attribute which stems from the dependency of objects in graphs creates limitations in many established data processing models and results in the need for new ones.

4.2 Scene Graphs

The graph data structure focuses on the relations between entities, thus making it a suitable means of data representation in many fields. For example, in computer science graphs are used to represent flow of computation within programs, in recommendation systems to rank results such as the web pages presented by search engines or content in social media. Furthermore, graphs can be utilized by the social sciences to represent interactions between people (social network), in research to reflect relations between papers through citation networks or even in chemistry to represent the structure of molecules.

In this dissertation, the graph structure will be used to represent the scene depicted in an image. This graph type, known as a scene graph, is a data structure which describes the object instances in a scene and how these objects relate to each other. It is a powerful tool first developed to aid in the field of higher level visual

understanding and reasoning, since it represents the semantics of a scene in an unrestricted and detailed manner.

From a more technical perspective a scene graph is a directed graph in which nodes are objects in a scene, like 'man' or 'table', and edges are the relations between them often describing positions or actions. A typical example of a scene graph and its corresponding image can be seen in Figure 4.2.1.

Some notable observations made from this figure are the following. There exists another type of vertex representing attributes, descriptive words for the present entities - adjectives in the majority. These nodes are not part of the well known and commonly used structure of the triplet <subject, relation, object> used in literature and can be ignored if irrelevant to the task the graph is being used for. It is also discernible that relations do in fact have a direction, from subject to object, and sometimes are depicted as vertices themselves. This practice is mostly a visual aid.

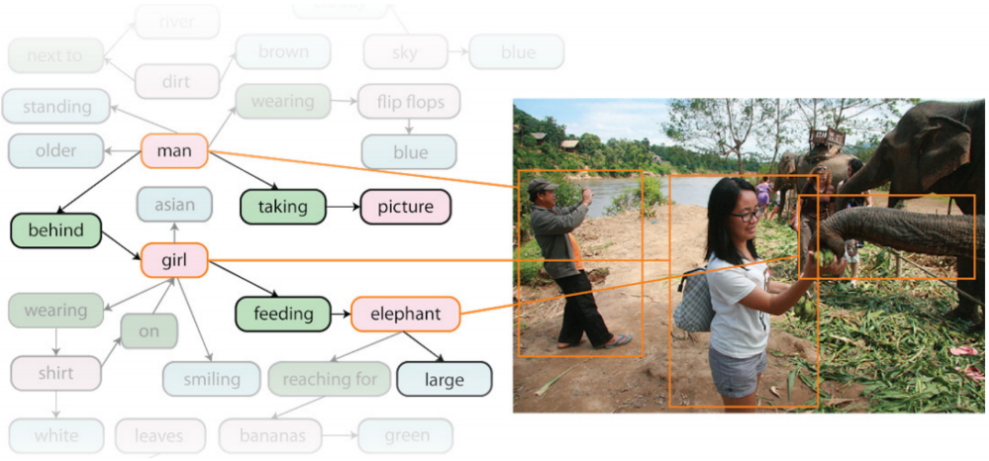


Figure 4.2.1: A Visual Genome Scene Graph [44]

Scene graphs are applicable in a variety of visual and textual tasks which relate to scene understanding, i.e. the process of detecting and naming objects, recounting their attributes and describing their relations [12]. The most prominent one lies in the field of computer vision and involves the generation of the scene graph itself using images and vice versa. There is a great amount of literature focusing on the task of Scene Graph Generation using both traditional and neural approaches. Especially in the case of generating an image, the inverse process, the use of a graph instead of plain text provides additional details concerning object relations which sentences cannot capture. Moreover, another task in which scene graphs are proven useful is Visual Question Answering. In this case, the target image and the semantics of the scene depicted are more effectively understood due to the fact that it is not viewed as a set of pixels, but as a set of correlated entities. Additional applications of the scene graph include image and video captioning, image reasoning and description and image or text retrieval, proving its value even in the field of Natural Language Processing.

Chapter 5

Graph Neural Networks (GNN)

It has been established from the previous section that the graph structure naturally emerges all around us. For this reason, neural networks operating directly on graph data were invented. Graphs are non-euclidean data and thus GNNs can be grouped in the broader category of Geometric Learning [9].

Graph Neural Networks (GNN) are known for their expressive power and recently have been gaining popularity due to their growing capabilities in various applications like recommendation systems and molecular fingerprinting. GNN applications range from chemistry and physics to traffic networks. These models can extract features from knowledge graphs and perform several graph mining tasks. They are widely used in computer vision for visual reasoning and semantic segmentation, in NLP for relation extraction or text classification and in combinatorial optimization to solve graph-related problems [108].

In the following sections we will review their unique features that make them so influential, explain how they are used and offer a review of the most important GNN variants.

Contents

5.1 Unique Characteristics

5.1.1 Motivation

The first question to be answered is why GNNs were created, even though there is already a substantial amount of neural network architectures. Most other data representations can be generalized to graphs but the opposite in not true. Most conventional Machine or Deep Learning algorithms are specifically crafted to cater to a certain type of data, such as images or text. Images can be perceived as fixed-size grid graphs and text or even speech can be thought of as line graphs. But in the general case, graphs are more complex, having a non-fixed number of unordered nodes within neighborhoods of variable size, and therefore existing models cannot handle them.

Another reason why machine learning is more complex on graphs is that most common algorithms assume instance independence. This is not true when performing node-level tasks where one graph is the input of the neural network and the instances are its nodes. Vertices are obviously related to one another with directed or undirected links.

The main motivation behind GNNs are Convolutional Neural Networks. Classic CNNs operate on images, or more broadly regular grids. As explained previously, they take advantage of the spatial locality of pixels which are nodes on the grid by sliding rectangular kernels with a small receptive field over the image to produce feature maps. The lack of locality in the traditional sense in graph data, their arbitrary size and invariance make regular convolution difficult to perform.

5.1.2 Permutation Invariance

Graphs are represented with adjacency matrices, a format that is permutation invariant. A Graph Neural Network is a transformation on nodes, edges or global-context that should preserve the graph symmetries - or permutation invariances - through the process of optimization.

More formally, for a graph with n number of nodes, their features can be represented by the feature matrix $X = [x_1, ..., x_n]^T$ where x_i are the feature vectors of each node. As explained the order of the nodes should not be important. However, by using this representation an ordering is unavoidably enforced. In order to overcome this problem the network should learn a function which is not affected by this ordering and therefore preserves permutation invariance.

A *permutation invariant* function f can be defined as:

$$f(PX) = f(X) \tag{5.1.1}$$

where X is any input matrix - in this case the feature matrix of a graph - and P is a permutation matrix. Permutation matrices contain exactly one non-zero element in each row and column and intuitively they scramble the rows of X resulting in different orderings.

A more thorough examination of the above function leads to the conclusion that operators like f are very limited. Basically such functions belong to a two-dimensional vector space containing summing the diagonal and summing the off diagonal of X. For example, a graph network consisting of only invariant layers would not even be able to discriminate between two graphs with the same number of nodes and edges, because it observes nodes as sets and not individually [56].

For all the above, in practice we want to combine invariant functions with equivariant functions which reflect the permutation of the node features in the output. *Equivariant operators* can be defined as:

$$f(PX) = Pf(X) \tag{5.1.2}$$

It is notable that a permutation in the node ordering could also be reflected on the adjacency matrix A of the graph. In this case the permutation will be enforced as PAP^T and result in $f(X, A)$ for invariance and $Pf(X, A)$ for equivariance.

Figure 5.1.1: Permutation in the adjacency matrix.

Left: Original Graph, Right: Graph obtained by permutation PAP^T

In practice graph neural networks often work on neighborhoods and not isolated nodes. For this reason, if we consider only the immediate 1-hop neighbors as the neighborhood N_i of i, the neighborhood feature matrix X_{N_i} would be a set of the neighbors' collective features. Each N_i would be processed separately using a local function g and finally the output would be a stack of the intermediate local results. To achieve f to be permutation equivariant, g should be invariant.

5.1.3 Weisfeiler-Lehman Test

The Weisfeiler-Lehman (WL) Graph Isomorphism Test [88] is used for the discrimination of non-isomorphic graphs. Specifically, it is able to distinguish between graphs that are not isomorphic, but cannot exactly provide proof that two graphs are in fact isomorphic. This capability gives significant insight on the graph's structure and aids in graph comparison.

In terms of graph theory, two graphs G and H are called isomorphic when an isomorphism exists between them. An isomorphism of graphs is a bijection between the vertex sets of G and H

$$f : V(G) \to V(H) \tag{5.1.3}$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H [90].

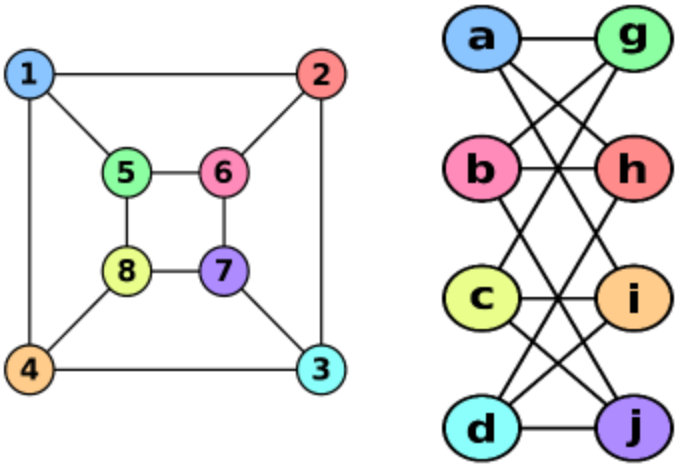


Figure 5.1.2: Two isomorphic graphs [90]

The principal idea of the Weisfeiler-Lehman test is to replace the label of each vertex with a multiset label consisting of the original and the sorted set of labels belonging to members of the neighborhood. This process

of relabeling is repeated for a given number of times simultaneously for all the graphs. If the outcome is graphs with different node labels, one can safely say that the input graphs were not isomorphic.

The problem that the WL test tackles is very challenging with a computational complexity in the class NP-intermediate. However, the WL test provides its answer in polynomial time. This test is also relevant to GNNs since it closely resembles the general idea of the GNN learning process of message passing. In fact, it was proven that a Graph Neural Network can at best be as good as the WL test at solving the graph isomorphism problem [101]. Thus, GNNs and the WL test are often compared based on their expressivity.

5.2 Taxonomy

Developments in the field of Geometric Learning in recent years have resulted in the creation of an abundance of different GNN models. Each one specializes in a certain task or offers a specific optimization. In order to explain the many GNN variants in the next section, we will explain on which bases they can be categorized:

5.2.1 Task Type

GNN variants tackle an array of graph analytics tasks each of which may focus on different attributes of the graph structure.

- **Node-Level tasks** require models which predict some property for each node in a graph, like its identity or role. With a broader perspective, node-level models build node representations. Analogous tasks would be image segmentation in computer vision or parts-of-speech prediction in NLP. In the supervised case, the most common tasks include node regression and node classification.
- **Edge-Level tasks**, in a similar fashion, call for models to predict the property or presence of edges. Subsequently, these tasks include edge classification and link prediction. An example of edge-level inference is scene understanding, trying to predict the existence and/or label of relation between objects in a scene.
- **Graph-Level tasks** entail the prediction of a single property for the whole graph or more vaguely attempt to extract graph representations. In order to do so, the model utilized is combined with pooling or readout operations. *Graph pooling* is a form of down-sampling which creates coarser graphs, whereas *Readout* instantly collapses node representations into a singular global graph representation. These tasks include graph classification or regression. The topic of this dissertation is also a graph-level task, which specifically employs information stemming from graph pairs to extract graph representations.

5.2.2 Architecture

Another way of categorizing Graph NNs is by prioritizing the type of framework or architecture they are using. This is the taxonomy [99] suggests and it is the following:

- **Convolutional graph neural networks (ConvGNNs)** inspired by classic CNNs are based on using the convolution operation, as defined for graph data. A basic outline of the function of the majority of these variants is: they define a node's neighborhood, aggregate information between neighboring nodes and finally generate each node's representation. Just like CNNs in standard Deep Learning, multiple graph convolutional layers are stacked in order to extract high-level node features the closer we get to the ouput. ConvGNNs are some of the most important building blocks for more advanced models. ConvGNNs can further be labelled as **spatial** or **spectral**. The distinction lies on whether the model makes use of spatial graph convolutions - spatial - or treats graphs as signals in the frequency domain - spectral.
- **Recurrent graph neural networks (RecGNNs)** make use of the idea of using information from previous units as influence for the current one, just like in classic Machine Learning. Specifically, they assume a node constantly trades information within its neighboorhood until a stable equilibrium is reached. This idea of *message passing* is one of the first in the field of GNNs and has been the basis for most other models, especially spatial-based convolutional ones.

- **Graph autoencoders (GAEs)** are based on the same premise as the Autoencoder explained in previous sections, i.e. are unsupervised learning frameworks which encode graphs/nodes into a low-dimensional space and attempt to reconstruct them. GAEs are mostly used on tasks like: network embedding - creating node representations through adjacency matrix reconstruction - and graph generation in a step by step manner.
- **Spatial-temporal graph neural networks (STGNNs)** are architectures which work on spatial-temporal graphs, i.e. graphs that change in structure overtime. To do so, they consider both spatial dependency as well as temporal dependency, often combining main ideas from convolutional and recurrent graph nets. They have recently become more significant due to their involvement in applications like traffic forecasting.

5.2.3 Training Type

Last but least, it is common to separate GNN variants according to what kind of signals they are trained with. This is is not a strict grouping since some models can be applied to multiple of the categories below.

- **Semi-supervised learning** tasks on graphs primarily work on graphs which are partially labeled, meaning only some nodes have a target class. The task in this case is node classification, which is achieved in a more robust way. Link prediction can also be carried out in a semi-supervised manner, in a similar fashion.
- **Supervised learning** tasks are based on the fact that the data is labeled. So they comprise of classification and regression tasks in node, edge or graph level.
- **Unsupervised learning** tasks are characterized by a lack of labels for the graph attributes. Some examples of tasks are node clustering and graph embedding. For the latter instance, the embedding can be learned in a purely unsupervised way using an end-to-end encoder-decoder based framework or by contrastive learning using negative samples. There also exist a lot of Graph Self-Supervised learning applications, which are mostly encoder-decoder based [51], and can be loosely grouped in this category.

5.3 GNN Models

In this section we will describe some of the most important Graph Neural Network architectures. Starting from the framework which introduced the concept of GNNs, we will expand to the many GNN variants and finally to general GNN frameworks. The most emphasis will be placed on models which will be used later on in our proposal.

5.3.1 Original Graph Neural Network

The original Graph Neural Network was presented by Scarselli et al. [75]. It falls under the category of RecGNNs and is an extension of prior recurrent models in order for them to work on graphs. It is based on the use of information diffusion within the graph in a recurrent manner until eventually a stable equilibrium is achieved.

Using notation from [108], the original model tries to learn a state embedding $h_v \in \mathbb{R}^d$ for every node v which includes not only its own information but also the neighborhood's N_v. It is defined as:

$$h_v = f(x_v, x_{co[v]}, h_{N_v}, x_{N_v}) \tag{5.3.1}$$

where x_v represents the features of node v, $x_{co[v]}$ are the features of its corresponding edges while h_{N_v} and x_{N_v} refer to the set of neighbors of v and represent their collective states and features respectively.

The function f is called *local transition function* and it is a parametric function common for all nodes which performs the state update by utilizing the neighborhood's information. In practice the neighborhood's information is summed in order to preserve invariance [99].

The state embedding can be used to produce an output related to the task. The output o_v is defined as:

$$o_v = g(h_v, x_v) \tag{5.3.2}$$

where g is the *local output function* and it describes how the output will be produced. The term function is used loosely and can even refer to an FFNN.

A compact representation of the above equations after stacking them for all nodes of the graph can be seen in Equation 5.3.3. H, O, X, X_N are the matrices corresponding to states, outputs, features and neighborhood features after the stacking, while F, G are the *global* versions of f, g which are also constructed by the combination of the local functions for each node.

$$\begin{aligned} H &= F(H, X) \\ O &= G(H, X_N) \end{aligned} \tag{5.3.3}$$

The state in each iteration of this recurrent model is computed using Banach's theorem [40], since H is the fixed point of the above equation. In order to find a unique solution, F must be a contraction map. H is randomly initialized and if the criterion for F is met, then the convergence is exponentially fast regardless of the initial value. After the fixed point is found, the last step node hidden states are forwarded to a readout layer extracting a global representation. The state update is defined as:

$$H^{t+1} = F(H^t, X) \tag{5.3.4}$$

In the case of supervised learning, the loss function looks like this:

$$\sum_{i=1}^{n} (t_i - o_i)^2 \tag{5.3.5}$$

As common in neural network applications, the cost function may include a penalty term to control other properties of the model. The optimization is gradient-based and determines the weights of the parametric functions f, g.

Overall, the learning algorithm iteratively updates the states using 5.3.3 until convergence to the fixed point. Then, gradients are computed and the weights are updated accordingly.

The original GNN presented crucial ideas which are utilized by many of the approaches which will be explained later and so is some of the terminology. However, it has some important limitations:

- The requirement of f being a contraction map limits the model's abilities.
- GNN is computationally expensive due to the iterative updates towards the fixed point.
- It is unsuitable for node-level tasks because of the fixed point requirement. The representation obtained is smooth and therefore not informative for each node.
- There is no representation of edge features.

5.3.2 Variants

In this section we will present some of the most important GNN variants presented in order to alleviate the aforementioned challenges or cater to unique cases. We will group them according to their architecture, as proposed in Section 5.2.2. STGNNs and RecGNNs will not be further elaborated on since they are out of the scope of this thesis.

Spectral-based ConvGNNs

As mentioned before, spectral ConvGNNs treat the graph as a signal and therefore are strictly mathematical approaches. They make use of the Laplacian matrix L of the graph, which captures its key properties, and use the Fourier transform to project the graph to the orthonormal space defined by the eigenvectors obtianed after factorizing the Laplacian. The formal definition of the Laplacian matrix can be seen below, where D is the degree matrix and A the adjacency matrix of the graph.

$$L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{5.3.6}$$

All spectral convolutional approaches for GNNs use the graph convolution defined in 5.3.7, but they differ in the choice of the filter g_θ. In this equation x is the input signal to be convolved, U is the matrix of eigenvectors ordered by eigenvalues resulting from the factorization of the normalized Laplacian matrix of the input graph and g_θ is the convolution filter, for which $g_\theta = diag(U^T g)$ is true.

$$x *_G g_\theta = U g_\theta U^T x \tag{5.3.7}$$

Spectral Convolutional Neural Network (Spectral CNN) [10] assumes that the filter is a diagonal matrix with learnable parameters. However, this initial approach suffers from computational inefficiency, dependence on the input graph's structure and non-spatial locality.

Chebyshev Spectral CNN (ChebNet) [14] approximates g_θ by Chebyshev polynomials of the diagonal matrix of eigenvalues Λ ($L = U \Lambda U^T$). The convolution can be defined as:

$$x *_G g_\theta = \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) x \tag{5.3.8}$$

where $\tilde{L} = 2L/\lambda_{max} - I_n$ with λ_{max} being the largest eigenvalue of L and I_n the identity matrix of dimension n. $T_k(x)$ represents the Chebysev polynomial for the k-th order and in the above equation it can be proven by induction that:

$$T_k(\tilde{L}) = U T_k(\tilde{\Lambda}) U^T \tag{5.3.9}$$

$T_k(x)$ can be computed as:

$$T_k(x) = 2x T_{k-1}(x) - T_{k-2}(x) \quad \text{with} \quad T_0(x) = 1 \quad \text{and} \quad T_1(x) = x \tag{5.3.10}$$

This model improves on the spatial locality problem of Spectral CNN. Specifically, the filters defined by ChebNet are K-localized, since the operation is a K^{th}-order polynomial of the Laplacian, allowing this model to obtain local features regardless of the graph size.

Graph Convolutional Network (GCN) presents the idea of using a first order approximation of ChebNet in order to alleviate overfitting. In fact, it assumes $K = 1$ and $\lambda_{max} = 2$. In the same direction, the model enforces the constraint $\theta = \theta_0 = -\theta_1$. After imposing these restraints on Equation 5.3.9 and taking into consideration 5.3.10, the convolution operation is:

$$x *_G g_\theta = \theta(I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) x \tag{5.3.11}$$

After empirically finding that the term $I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ causes numerical instability, a *renormalization* trick was used. The term $D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = \bar{A}$ was replaced with $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} = \hat{A}$ where $\tilde{A} = I_n + \bar{A}$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. All of the above can be described by this compact equation:

$$H = X *_G g_\theta = f(\hat{A} X \Theta) \tag{5.3.12}$$

where f is an activation function and multiple inputs and ouputs are allowed due to the matrix form.

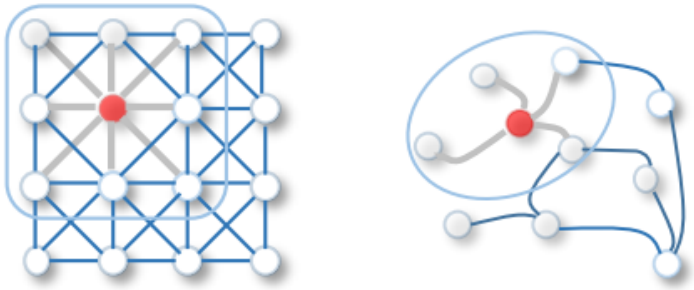


Figure 5.3.1: Comparison of 2D and Graph Convolution [99]

Left: 2D Convolution in CNNs, Right: Graph Convolution

The GCN is a special case of a spectral approach since it can also be perceived as a spatial one. In the equation below, we can see how the aggregation of information within the neighborhood would be performed. In this case the node itself is also regarded as its 1-hop neighbor.

$$h_v = f(\Theta^T(\sum_{u \in N(u) \cup v} \hat{A}_{v,u} x_u)) \quad \forall u \in V \tag{5.3.13}$$

This model is very frequently used as a part of more complex architectures in literature due to its simplicity and good experimental performance.

Spatial-based ConvGNNs

Spatial approaches define the graph convolution operation taking into account the spatial locality of nodes, i.e. the neighborhood, making them adjacent to conventional CNNs. However, they are also heavily inspired by the RecGNN pioneering ideas. Specifically, spatial ConvGNNs convolve the central and neighboring nodes' features, as seen in Figure 5.3.1, to obtain the updated representation which ultimately leads to information propagation along the edges of the graph.

Diffusion Convolutional Neural Networks (DCNN) [3] conceive the convolution as a diffusion process and define the diffusion convolution function as:

$$H^{(k)} = f(W^{(k)} \odot P^k X) \tag{5.3.14}$$

where f is an activation function and $P = D^{-1}A$ and is called the *probability transition matrix*. During the information distribution each node exchanges information with one of its neighbors with a certain transition probability provided by P. After an amount of iterations an equilibrium is reached and the diffusion is over. The update rule of the original GNN is not used here, instead each hidden representation $H^{(k)}$ is computed independently and all of them are concatenated to obtain the final one.

GraphSAGE [31] learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. This framework introduces the idea of sampling the node's neighborhood in order to obtain a fixed number of neighbors in each iteration. The convolution process can be summarized in:

$$h_v^{(k)} = \sigma(W^{(k)} \cdot f_k(h_v^{(k-1)}, \{h_u^{(k-1)}, \forall u \in S_{N(v)}\})) \tag{5.3.15}$$

where f_k is an aggregation function, σ is the sigmoid function and $S_{N(v)}$ is a random sample of the neighborhood of v. We assume that h_v is initialized with the node's original features. The choice of aggregation function is important since it should be permutation invariant, such as a mean, sum or max function.

GraphSAGE is practically an extension of the GCN to inductive unsupervised learning. It is proposed that it should be trained using an unsupervised graph-based loss, but can also be trained in a supervised manner if needed. The model provides low-dimensional embeddings for downstream tasks which reflect local and global characteristics of the graph. It focuses on attributed graphs - with features - but can also work without node attributes by using handcrafted structural information. GraphSAGE does not train the embeddings, but the aggregator function with which inference is performed.

This approach is more efficient for unseen data than older transductive approaches and it is also invariant to orthogonal transformations. With adjustments it is an instance of the Weisfeiler-Lehman test.

Graph Attention Network (GAT) [83] adopts the idea of attention proposed by [82] in order to decide which members of a node's neighborhood have more important information. It aims to learn the relative weights between adjacent nodes and therefore differs from previous approaches like GCN and GraphSAGE because the concept of the neighborhood is not pre-defined or identical.

The convolutional operation is defined as:

$$h_v^{(k)} = \sigma(\sum_{u \in N(u) \cup v} \alpha_{vu}^{(k)} W^{(k)} h_u^{(k-1)}) \tag{5.3.16}$$

where the attention weights for each node v can be defined as:

$$\alpha_{vu}^{(k)} = softmax(LeakyReLU(a^T [W^{(k)} h_v^(k-1) || W^{(k)} h_u^{(k-1)}])) \tag{5.3.17}$$

The variable a represents a set of learnable parameters. The hidden representation is initialized with the features of each node and the softmax function ensures that attention weights sum to one.

The mechanism above is called *self-attention*, but GAT additionally uses *multi-head attention* in order to stabilize learning and make the model more expressive. The exact equations can be found in [83].

GAT is efficient since the node-neighbor pairs can be computed simultaneously. Moreover, it is indifferent to neighborhood sizes and can be applied to inductive learning problems easily.

Graph Isomorphism Network (GIN) [101] is the first spatial approach that addresses the inability of previous spatial models to discriminate between different graph structures based on the embeddings produced. In order to do that, GIN uses a simple technique, adding a learnable weight parameter for the central node of the convolution. The operation is defined below where $\epsilon^{(k)}$ is the weight.

$$h_v^{(k)} = MLP((1 + \epsilon^{(k)}) h_v^{(k-1)} + \sum_{u \in N(u)} h_u^{(k-1)}) \tag{5.3.18}$$

GIN is proved to be as powerful as the WL graph isomorphism test, i.e. produces different node embeddings when dealing with non-isomorphic graphs This makes this model maximally powerful and therefore the most expressive among the variants. The most discriminative GNN should use an injective multiset function in order to distinguish rooted subtree structures. For this reason, GIN uses the Multi-Layer Perceptron and the sum function as the aggregator. For each layer, node embeddings are summed and the result is concatenated. Thus, the expressiveness of the sum operator is combined with the memory of previous iterations by using concatenation. All things considered, one should keep in mind that the theoretical power of the GIN does not always translate in practice

GAEs

In this section we will present the most popular graph autoencoder architectures presented in the same paper [42].

Graph AutoEncoder (GAE) improves on previous autoencoder-based techniques by leveraging the information provided by node features. It uses two GCN layers to capture node structural and node feature information at the same time as seen in the equation for the encoder below.

$$Z = enc(X, A) = Gconv(ReLU(Gconv(A, X; \Theta_1)); \Theta_2) \tag{5.3.19}$$

where Z is the graph's embedding matrix.

The decoder aims to utilize the embeddings in order to reconstruct the graph adjacency matrix. The model is trained with a CrossEntropy loss between the real adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$. Its equation is visible below, where z_v refers to the embedding of node v.

$$\hat{A}_{v,u} = dec(z_v, z_u) = \sigma(z_v^T z_u) \tag{5.3.20}$$

Variational Graph AutoEncoder (VGAE) is the variational version of the previous model which learns the data distribution. It mainly attempts to alleviate overfitting, but additionally gives the opportunity for the model to be used for generative tasks.

Aside from employing distributions, the main difference from GAE is the use of the Kullback-Leibler divergence function which measures the distance between two distributions as a regularizer, in a similar fashion to the conventional VAE.

Special GNN variants

All of the variants above are made for static homogeneous graphs with node features. But graphs in real life are not always that simplified. For this reason, dedicated efforts have been made to create GNN variants for complex graphs. These efforts can significantly contribute to the adoption of GNNs in a broader range of applications [53].

Heterogeneous graphs are commonly used to represent the relations between papers or authors. For this reason, many GNN approaches, such as the GAT-filter, have been modified to accommodate them. This is possible with the use of meta-paths [35] which capture various relations between nodes with different semantics. The original graph is split into a set of homogeneous graphs which are dealt with separately. In a similar manner, if a graph is bipartite the graph convolution operation is split into two components, one for each set of nodes.

Multi-dimensional graphs are graphs containing different types of edges.In this case, it is necessary to consider both within and across-dimension interactions, i.e. relations between nodes which are of the same type as well as are not. Towards this direction, [54] was presented. Moreover, the type of graph containing two types of relations denoted as positive and negative is called signed and is a common representation used in social network theory. For those types of graphs we cannot treat the positive and negative subgraphs independently because they interact. Works like [15] leverage the balance theory to model these interactions.

Hypergraphs, also often representing author and paper networks, are graphs containing hyperedges, i.e. edges connecting any number of nodes. These structures can be transformed into simple graphs by pairwise relation extraction, as proposed by [19, 102].

Finally, graphs can also have temporal characteristics meaning they can evolve overtime. Dynamic graphs are graphs with many real world applications and require learning multiple models for different snapshots in time. An interesting recent dynamic or STGNN model is [66].

5.3.3 General Frameworks

The existence of such an abundance of GNN variants led to the creation of various general frameworks which group them together. This way, we can study them more efficiently and draw significant conclusions. In this section, we will coarsely describe some of the most important ones belonging to the category of spatial ConvGNNs.

Mixture model network (MoNet) [60] is a framework combining several non-euclidean models, such as CNNs for manifolds and GNNs. A new pseudo-coordinate system is created, with its origin being each point on a manifold or each vertex on a graph. The neighboring points/nodes are defined accordingingly. GCN can be formulated as an instance of this framework.

Message Passing Neural Network (MPNN) [24] is probably the most well-known family of GNNs. This framework uses two phases: the message passing phase and the readout phase for graph-level tasks.

In the first phase, information is aggregated to each node's neighbors by using a message function M_k which is essentially the graph convolution. Then, an update function U_k is used to update the hidden state. This is a K-step process defined by the equation below.

$$h_v^{(k)} = U_k(h^{(k-1)}, \sum_{u \in N(v)} M_k(h_u^{(k-1)}, x_{vu}^e)) \tag{5.3.21}$$

The readout phase is optional depending on whether node or graph-level inference is required. In the second case, the representation $h_v^{(K)}$ is passed to a readout function in order to create a graph representation.

$$h_G = R(h_v^{(k)} | u \in G) \tag{5.3.22}$$

where R is the readout function.

MPNN can describe a variety of different spatial convolutional GNNs which generally adopt this message passing process. This is done by defining the functions M_k, U_k and R in appropriate ways. One of the networks belonging to this group is GCN and also several other models used on molecular structures which have not been mentioned.

It is notable that GIN has been proven to be more powerful than all MPNN-based methods at least in theory.

Non-Local Neural Network (NLNN) [85] uses a non-local operation to compute the hidden state at a position as a weighted sum of features at all possible positions in space, time or spacetime. Consequently, the NLNN essentially groups all "self-attention" methods, including GAT.

Graph Network (GN) [7] is an even more general framework which groups models according to the type of task they fulfill, i.e. node-level, edge-level and graph-level. It generalizes other frameworks, including the aforementioned ones.

This framework is based on the use of a computational unit called GN block. This block defines three update and three aggregation functions, each referring to a different level. By choosing these functions, a significant array of models or other frameworks can be defined.

Chapter 6

Counterfactual Explanations

The field of AI interpretability has been gaining increasing attention due to the growing realization that in order to confidently be able to count on the impressive outputs provided by intelligent systems, appropriate explanations are needed [2]. AI applications are taking over almost every aspect of everyday life and it is significant to assure that these models and the respective input datasets are not biased. AI Explainability is crucial not only for detecting biases, but also for increasing social acceptance, establishing safety measures for applications like self-driving cars and finally for the enhancement of the machine learning systems themselves using the information learnt [59].

In this thesis, we will focus on a specific explainability technique called Counterfactual Explanations. In the following sections we will define this type of AI explanations and describe the way they relate to our proposed model.

Contents

6.1 Definitions

A general definition for a *counterfactual explanation* - or simply a *counterfactual* - is that it describes the causation of a situation by assuming "If X had not happened, Y would not have occurred". Its name derives from the need to imagine a counterfeit reality which contradicts our perception [59].

In terms of AI explainability, the counterfactual aims to provide an explanation for "What would need to change in order for the model to make a different decision". Therefore, they essentially can explain predictions of individual instances, where the causes of the predicted outcome are particular feature values of this instance.

Counterfactuals are contrastive and selective, meaning they find minimal changes in the feature space. Thus, they are human-friendly. However, there are usually multiple different counterfactual explanations for the same instance which explain it equally well. This creates contradictions which can be overcome simply by providing all possible truths and letting the end user decide on establishing a criterion to choose the best one.

There are both *model-agnostic* and *model-specific* counterfactual explanation methods. Model-agnostic approaches have a chief advantage over solely using interpretable models, which is their flexibility. They can be tested on several different machine learning models and evaluated on the explainability of an array of tasks. The method which we will be focusing on is also model-agnostic.

A good counterfactual explanation should first and foremost be able to produce the predefined prediction. It should provide a minimal explanation, i.e. the closest one in terms of features as determined by some distance metric. Moreover, its features should have possible values. All three of these criteria are met by [22]. Finally, in some cases it is desirable for a CE method to provide multiple explanations for the decision-subject to choose.

6.2 Conceptual Edits

In this section, we will elaborate on the counterfactual explanation method which inspired this work. Essentially, our goal is to produce embeddings of the input graphs which capture the graph similarity in terms of edit distance in order to facilitate the counterfactual application in this paper.

Filandrianos et al. [22] propose a theoretical framework for the computation of counterfactual explanations through conceptual edits. In this context, concepts are the general representations of objects present in the input data and are linked with external knowledge, in the form of concept hierarchies. The outline of the presented framework is described in Figure 6.2.1

It is discernible that the black-box model used in this application is a classifier, which categorizes images from the COCO dataset depending on what type of room they are set in. Thus, the counterfactual answers the question "What would have to change for something to be classified as X instead of Y". For better comprehension of what is depicted in the figure, we will further explain some of the terminology used.

An *Explanation Dataset* is a set of ordered pairs comprising of samples of the input data to the black-box classifier and the subset of concepts associated with each instance. More formally, if the classifier is $F : D \rightarrow [0,1]^c$ where D is the domain to be explained, c the number of classes and CN is set of all atomic concepts, we can define the explanation dataset as:

$$(x_i, C_i), \quad x_i \in D \quad and \quad C_i \subseteq CN \tag{6.2.1}$$

A *Tbox* is a set of terminological axioms described as $A \subseteq B$, where $A, B \subseteq CN$, as they are defined in the WordNet [58] noun hierarchy. The Tbox can also be represented using a directed graph $G(V, E)$ in which each node represents a different atomic concept or the universal concept $\top$ and the edges have a direction from A to B. This property is also true for transitive inclusions.

A *Conceptual Edit* is one of the following three types of change which transforms a set of concepts A:

- *Replacement* of concept $A \in A$ with concept $B \notin A$

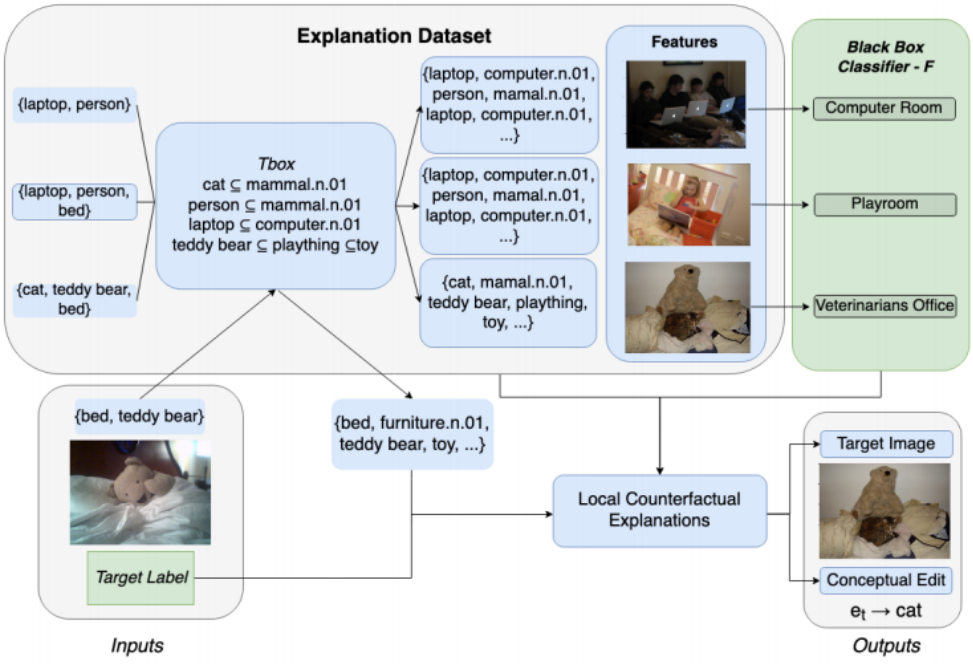


Figure 6.2.1: Conceptual Edits as Counterfactual Explanations framework [22].

- *Insertion* of concept $B \notin A$
- *Deletion* of concept $A \in A$

Edits are defined as $\epsilon_{x \to y}$ where $x, y \in CN$ and y is the concept replacing x. In the case of insertion, x will become the universal concept $\top$ and as for deletion y will become $\top$. For example, in the figure, $\epsilon_{\top \to cat}$ means the insertion of a cat in the image. The cost of concept set edits can be derived from the smallest path between the two concepts in the Tbox graph.

Finally, *Local Counterfactual Explanations* constitute the set of conceptual edits that transform sets of concepts to other sets of concepts classified in a different class in a minimal way, i.e. with the smallest cost. A stricter definition involves constructing a graph G(V,E) where $V = D$ and E contains edges of weight $1/\sigma(a, b)$ for every pair of $a, b \in D$. The denominator is called significance of transformation and its definition can be seen in Equation 6.2.2. Paths between between elements of class c_1 to any other element of class c_2 are counterfactual explanations, but our goal is to find optimal explanations and therefore shortest paths.

$$\sigma(a, b) = \frac{|F(x_a) - F(x_b)|}{D_T(C_a, C_b)} \tag{6.2.2}$$

where $a = (x_a, C_a), b = (x_b, C_b)$ and D_T is the distance of the concepts.

The paper also proposes a way to compute *Generalized Counterfactual Explanations*, i.e. computation of counterfactuals for a subset of instances that satisfy a specific query and the importance of each concept. The importance is a number that signifies how often an object is deleted or inserted in order to lead to a transformation belonging to a specific class.

Overall, The computation of counterfactual explanations entails:

- Finding the concept distance between all present concepts using shortest paths on the undirected TBox graph obtained with Dijkstra's algorithm.
- Computing Concept Set Edit Distance from one set of concepts to the other using Karp's algorithm for minimum weight full matching. This requires constructing a complete bipartite graph from the two sets.

- Obtaining local counterfactual explanations by finding shortest paths on the already constructed graph (using Dijkstra's algorithm again).

,

Our contribution to this approach is dual: i) using graph data which contain relations between the sets of concepts and are therefore a richer representation and ii) finding the most similar input instance pairs and therefore only computing the edits for these pairs, which are considered minimal. Of course, the most similar pairs are obtained using GNNs.

6.3 Related Work

After examining the concepts of Conceptual Edits as Counterfactual Explanations, we are going to present other relevant work based on counterfactuals, use of knowledge bases and the utilization of GNNs in such approaches.

Firstly, there are many works focused on counterfactual explanations in recent literature. An approach by Goyal et al. [28] introduces the idea of detecting the regions of the image requiring change by leveraging the low-order features extracted by the deep neural network used. Moreover, the minimum-edit method is also used, but the edits in this case are pixel-level and depend on the information provided by the "black-box" model. Poyiadzi et al. [70] propose Feasible and Actionable Counterfactual Explanations, meaning they enforce restraints such as following the data distribution and guaranteeing feasibility and actionability of explanations provided. The idea of providing a minimal set of changes is also proposed in [26] with the difference of it operating on numeric data dependent on the classifier's input feature space. In the method presented by [107], the use of external knowledge as aid to a text-to-image generative adversarial network is highlighted. This recent survey [13] provides further detailed descriptions of models and categorizes them based on their properties.

Next, we will present some relevant work involving counterfactuals with external knowledge from knowledge graphs, which provide symbolic knowledge in a way that facilitates both machines and humans. Silva et al. [78], for example, uses WordNet in the task of task entailment with great results while simultaneously explaining predictions. Other approaches like [16, 50] focus on mimicking a black-box classifier's behaviour with semantic query answering over external knowledge in order to provide explanations. Finally, in approaches like [1] the knowledge from task-specific ontological bases is used for error explanation. A more thorough look on the use of knowledge graphs in interpretable AI is provided by [81].

Finally, the intersection of counterfactual explanations and Graph Neural Networks mostly contains works which attempt to interpret GNNs and not the other way around. For instance, Lucic et al. [52] proposes counterfactual explanation of any GNN model by perturbating the adjacency matrix of input graphs edge-by-edge until a change of prediction. Similarly, Bajaj et al. [6] suggests finding a subset of edges which upon deletion changes the model's prediction. This set needs to be small and robust to noise caused by perturbations. One of the only works which utilizes GNNs in order to ultimately provide explanations is Holzinger et al. [34] which proposes the construction of counterfactual graphs for explainability and causability in AI medical applications. These graphs are multi-modal and are created using GNNs.

Chapter 7

Graph Similarity

Graph Similarity or *Graph Matching* is the problem of finding a similarity between graphs, i.e. finding a mapping $s : G \times G \to \mathbb{R}$ for a pair of graphs, characteristic of how similar or dissimilar they are.

The matching of graphs is categorized according to the plausible accuracy which can be achieved. *Exact Graph Matching* is essentially the problem of *Graph Isomorphism* explained in Section 5.1.3. In this context, the goal would be to recognize isomorphic graphs, given that the appropriate conditions are met, which are maximally similar. However, exact matching is not always possible. For example, the subject graphs could have different numbers of nodes or edges or just be attributed. This problem is defined as *Inexact or Error-tolerant Graph Matching* and it entails finding the best possible match.

The process of inexact graph matching is fundamentally an optimization problem which can be tackled using many different algorithms. For instance, *Graph Edit Distance* is an appropriate similarity measure based on counting operations that are necessary to transform one graph to the other and selecting the minimum one. This method along with the polynomial alternative of *Graph Kernels* will be explored in this thesis and compared to the idea of using GNNs to map each graph to a feature vector and use distance metrics on vectors to obtain similarities. In the sections below we will elaborate on the aforementioned methods.

Contents

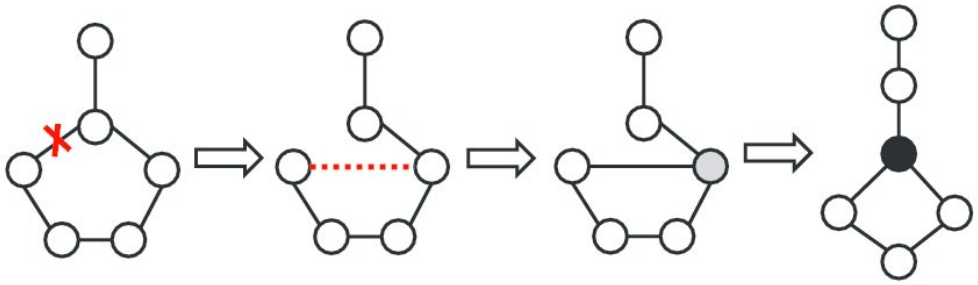


Figure 7.1.1: Graph Edit Distance Between Two Graphs. [4]

Minimum GED requires 3 edit operations and if they were all equally weighted its value would be 3.

7.1 Graph Edit Distance

Graph Edit Distance (GED) is a measure of similarity (or rather dissimilarity) between pairs of graphs. It can be considered a generalized version of other distances, such as string edit distance, tree edit distance [106] or Hamming distance [32], if graphs with proper constraints are constructed.

The general formal definition of GED, based on the paper which introduced it [74], can be seen below. The Graph edit Distance between the graph pair G_1 and G_2 is denoted as $GED(G_1, G_2)$, the edit operations are e_i, their costs are $c(e_i)$ and $P(G_1, G_2)$ denotes the set of edit paths transforming the first graph to an isomorphic of the second. Edit operations include insertion, deletion and substitution of vertices and edges. The costs of edit operations are specific to the input graph format and therefore defined by the user.

$$GED(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in P(G_1,G_2)} \sum_{i=1}^{k} c(e_i) \tag{7.1.1}$$

Exact algorithms for computing GED typically aim to minimize the cost of the edit path from one graph of the pair to the other. The methods used for this computation are either pathfinding search or shortest paths and they often utilize the A* search algorithm. GED is a computationally expensive NP-hard problem. Additionally, its approximation is also in a difficult class and thus GED can be placed in the APX-hard complexity class. There have been many graph edit distance approximation approaches which achieve cubic complexities in the majority. For instance, some of the most popular ones are Hungarian [45], Hausdorff [23] and BP-Beam [63].

The GED algorithm used in this thesis is a well-known approximation based on bipartite graph matching by means of the Volgenant-Jonker assignment algorithm [18]. Assignment algorithms lead to polynomial time complexities because the process of assigning nodes can be solved as a Linear Sum Assignment Problem (LSAP). LSAP requires the construction of a cost matrix whose elements are the assignment costs between two sets. The goal is to find the permutation which minimizes the overall cost. Volgenant-Jonker (VJ) [37] is an LSAP-based algorithm, popular for its efficiency. It achieves a complexity of $O(n^3)$, but in practice it is much faster. Moreover, it is effective for both sparse and dense graphs, and it is insensitive to the cost value range. VJ is a three step process, consisting of preprocessing to obtain a partial solution, sparsification followed by improvement through augmentation and finally determination of shortest paths. Thus, this algorithm is also called shortest augmenting path. Augmentation comprises of the construction of the auxiliary network graph and computation of minimal cost alternating paths between unassigned rows and columns which are subsequently used to improve on the solution. The bipartite matching heuristic firstly focuses on the creation of the cost matrix. In the case of GED, LSAP works on the two sets of vertices of the graphs and the cost matrix is expanded to accommodate deletions and insertions. The node edit scores include edge edit costs for adjacent nodes. Secondly, VJ is used to compute the node assignments. This way, while node operations are explicitly computed, the edge operations are inferred. This results in a great speed-up, but is also the reason for the suboptimality of the solution. However, this approach offers great results in practice.

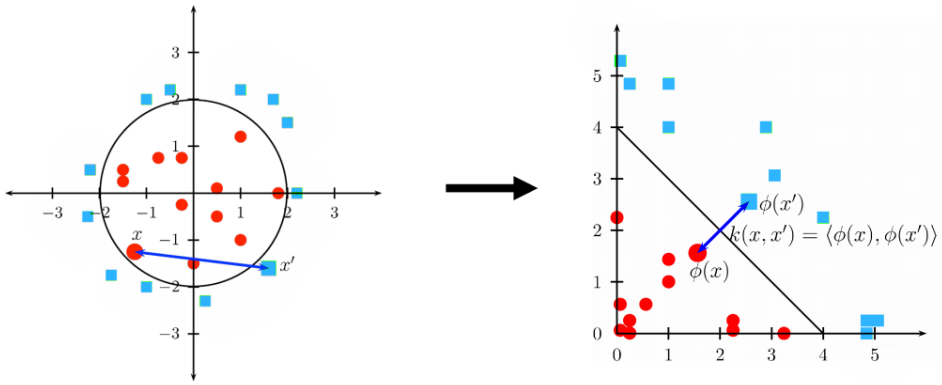


Figure 7.2.1: Illustration of Kernel Trick [38]

7.2 Graph Kernels

Even though Graph Edit Distance algorithms effectively compute graph similarity, they are not computationally efficient. The problem of GED essentially includes subgraph isomorphism check as an intermediate step which is NP-hard. Furthermore, choosing edit operation costs is a difficult task which requires mindful attention.

Graph Kernels are kernel functions used on graphs, which measure the similarity of graphs in polynomial time. They provide an efficient, expressive and widely applicable alternative to GED.

A positive definite kernel on a non-empty set X is a symmetric function $K : X \times X \to \mathbb{R}$ if 7.2.1 holds, where $x_i \in X$, $n \in \mathbb{N}$ and c_i are real numbers. A distinction is often made between positive-definite (p.d.) kernels for which the equality only holds and positive semi-definite (p.s.d.) kernels for which the opposite is true. Intuitively, p.d. kernels only have positive eigenvalues, while p.s.d can also have zero values. Kernel functions are essentially similarity functions. .

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \tag{7.2.1}$$

A common application of kernel functions is the "kernel trick". With the help of the kernel, data is mapped to a feature space with higher dimensions where the inner product of all pairs can be computed fairly easily. Thus, the coordinates of the data do not need explicit computation and the measurement of similarity in the feature space is possible by computing the kernel in the input space:

$$k(x, x') =< \phi(x), \phi(x') > \tag{7.2.2}$$

where $\phi : X \to \mathbb{H}$ and $\mathbb{H}$ is a Hilbert space.

Graph kernels can be defined as convolution kernels on pairs of graphs and are also represented by 7.2.2. Similarly to GED algorithms, there is a connection to the graph isomorphism problem. Specifically, if ϕ is injective in the equation above, the input graphs, namely G and G' are isomorphic [38].

In this thesis, we will explore five graph kernel methods, three of which can handle graphs with node attributes. We will elaborate on them below, starting with approaches on labeled graphs and continuing with attributed graph methods.

Weisfeiler-Lehman Kernel

This kernel is based on the Weisfeiler-Lehman algorithm, explained in section 5.1.3. This framework was introduced by [76] and it defines graphs as:

$$\{G_0, G_1, ..., G_h\} = \{(V, E, l_0), (V, E, l_1), ..., (V, E, l_h)\} \tag{7.2.3}$$

where $G_0 = G$ the graph with original labels, $G_1 = r(G_0)$ the graph after the first iteration and relabelling. The graphs are represented with triplets (V, E, l_i) where l_i is the set of labels and the WL algorithm is repeated h times.

The WL kernel operates on top of an existing kernel, called the base kernel, and denoted as k. Then, the Weifeiler-Lehman for h iterations is defined as seen below and proven to be p.s.d.

$$k_{WL}^{(h)}(G, G') = k(G_0, G_0') + k(G_1, G_1') + ... + k(G_h, G_h') \tag{7.2.4}$$

Pyramid Match Kernel

The Pyramid Match (PM) graph kernel is the extended version for graph data of the well-known computer vision algorithm with the same name [29, 65]. It operates by initially embedding each graph's nodes in a d-dimensional vector space using the absolute eigenvectors of the largest eigenvalues of the adjacency matrix. The sets of graph vertices are compared through the mapping of the corresponding points in the d-dimensional hypercube to multi-resolution histograms. The comparison is achieved with a weighted histogram intersection function, as defined in Equation 7.2.5.

$$I(H_G^l, H_{G'}^l) = \sum_{i=1}^{D} min(H_G^l(i), H_{G'}^l(i)) \tag{7.2.5}$$

The process of comparing histograms and finding matches occurs in several levels, corresponding to different regions of the feature space with increasing size. The algorithm counts new matches at each level, i.e. points in the same region, and weights them according to the size of the level. The cells/regions double in size in each iteration of the algorithm. In the equation above H_G^l denotes the histogram of G while $H_G^l(i)$ denotes the number of vertices of G in the i^{th} cell. The PM kernel is computed as:

$$k_{PM}(G, G') = I(H_G^L, H_{G'}^L) + \sum_{l=0}^{L-1} \frac{1}{2^{L-1}} (I(H_G^l, H_{G'}^l) - I(H_G^{l+1}, H_{G'}^{l+1})) \tag{7.2.6}$$

where there are L levels altogether.

The above equation holds true in the case of labeled graphs as well, but in order for a match to exist the two points should also have the same label. Finally, the PM kernel is computed overall by summing the results for each discrete label, as seen below.

$$k_{PM}(G, G') = \sum_{i=1}^{c} k_{PM}^i(G, G') \tag{7.2.7}$$

where there are c distinct labels.

The kernel's complexity is $O(ndL)$ which comparable to other kernels is quite computationally expensive.

Propagation Attribute Kernel

The Propagation Attribute (PA) graph kernel [64] is based on information propagation between nodes, similarly to some GNN variants. The nodes in this case have attributes which, if the graph is labeled, are One-Hot-Vectors of the initial dictionary. The nodes are perceived as a probability distribution of size $n \times d$ where n corresponds to the number of vertices and d to the dimension of attributes. Information diffusion is enforced, as defined in 7.2.8.

$$P_{t+1} = TP_t \tag{7.2.8}$$

T is a transition matrix which is either user-defined or $T = D^{-1}A$ where $D = diag(\sum_j A_{ij})$ and P_0 is the original graph attributes.

The kernel in each iteration is computed as:

$$k_{PA}(G, G')^t = k_{PA}(G, G')^{t-1} \sum_{u \in G_t^{(i)}} \sum_{v \in G_t^{(j)}} k(u, v) \tag{7.2.9}$$

where t is the iteration number and $k(u, v)$ is found through binning. The technique used in called Locally Sensitive Hashing (LSH) and helps preserve similar diffusion patterns in the same bins.

Subgraph Matching Kernel

The Subgraph Matching (SM) [43] kernel counts the number of matchings between subgraphs of bounded size between two graphs, as suggested by Levi [48]. It can be applied to graphs that contain node labels, edge labels, node attributes or edge attributes.

The common subgraph isomorphism kernel can be computed as described by 7.2.10 when the constraint for the vertex and edge kernel functions κ_V, κ_E in 7.2.11 is satisfied. The equations below assume that an isomorphism between $G(V, E)$ and $G'(V', E')$ is a bijection $\phi : V \to V'$ which preserves adjacencies and labels. The function $\psi : V \times V \to V' \times V'$ is the mapping of vertex pairs implicated by the bijection ϕ such that $\psi((v, u)) = (\phi(v), \phi(u))$.

$$k_{SM}(G, G') = \sum_{\phi \in \mathfrak{B}(G,G')} \lambda(\phi) \prod_{v \in S} \kappa_V(v, \phi(v)) \prod_{e \in S \times S} \kappa_E(e, \psi(e)) \tag{7.2.10}$$

where $S = dom(\phi)$ and S the subset of vertices belonging to a subgraph of G. $\mathfrak{B}(G, G')$ is the set of all bijections between S, S' and λ is a weighting function.

$$\kappa_V(v, v') = \begin{cases} 1, & if \quad l(v) \equiv l(v'), \\ 0, & otherwise \end{cases} \qquad \kappa_E(e, e') = \begin{cases} 1, & if \quad e \in E \wedge e' \in E' \wedge l(e) \equiv l(e') \quad or \quad e \notin E \wedge e' \notin E', \\ 0, & otherwise \end{cases} \tag{7.2.11}$$

There is also a more general subgraph matching kernel which builds a weighted product graph to allow a more flexible scoring of bijections.

GraphHopper Kernel

The GraphHopper (GH) kernel [20] compares shortest paths between node pairs of the input graphs, while considering both path lengths as well as which vertices were encountered while "hopping" along shortest paths. It is practically equivalent to a weighted sum of node kernels.

More formally, it can be defined as:

$$k_{GH}(G, G') = \sum_{\pi \in P} \sum_{\pi' \in P'} k_p(\pi, \pi') \tag{7.2.12}$$

where π, π' are paths and P, P' are families of shortest paths in G, G' respectively. Namely, the path kernel k_p is defined as follows. The notation used below represents discrete paths as $|\pi|$. This definition essentially shows that k_p is a sum of node kernels k_n on vertices encountered while simultaneously hopping along paths π and π' of equal discrete length.

$$k_p(\pi, \pi') = \begin{cases} \sum_{j=1}^{|\pi|} k_n(\pi(j), \pi'(j)), & if \quad |\pi| = |\pi'|, \\ 0, & otherwise \end{cases} \tag{7.2.13}$$

7.3 Related Work

The task of graph similarity has been thoroughly explored throughout the years, primarily through the use of non-neural algorithms. Among the first techniques to be proposed as a graph similarity metric was Graph Edit Distance [74], which was explained in Section 7.1 and directly relates to this work because it will be used as a point of reference in the evaluation process. Other early graph similarity metrics are Maximum Common Subgraph [11] and Graph Isomorphism [8]. As established, these types of methods rely on the problem of isomorphic graphs and therefore have NP-hard complexities. Approximate methods using heuristics are proven effective in the acceleration of the process. These mostly comprise of GED approximations and include both improvements on the A^* search algorithm as well as node assignment methods. Examples were provided in previous sections [45, 63, 23], with the most relevant being the approach of using the bipartite matching heuristic in combination with the Jonker-Volgenant assignment algorithm [37]. Despite the enhancement in runtime, these methods are still best used for small graphs.

Methods based on graph theory compute similarity values but fail to produce representations of the graphs. Some well-known approaches involve learning latent representations of the vertices of only one graph at a time. These include node2vec [30], DeepWalk [68], LINE [80] and NetMf [71]. However, there is no straightforward way to compare the node embeddings produced in this case, given the lack of permutation invariance. One way to combat this setback is by embedding the graph as a whole, like graph2vec does [61]. Finally, an even better idea is to transform similarity estimation to a task involving the mapping of graph pairs in a different space and the use of these representations to finally produce the similarity score. One end-to-end method we already explained is that of graph kernels, with an abundance of different techniques [103].

Finally, the most relevant methods are those involving Graph Neural Networks. GNNs intrinsicly embed graphs at the node level. Therefore, the plain use of GCN [41] or VGAE [42] would create permutation invariant representations. This simple approach encourages learning on one graph at a time and therefore does not compare graphs during the training process. Other methods suggest making the models supervised and aim to improve graph classification accuracy, such as DiffPool [104], which is a pooling model made to operate in unison with other GNN architectures, and CAPSGNN [100] which aims to create more meaningful graph embeddings through the use of a capsule network. The closest approaches to the one presented in this thesis are trained on graph pairs. SimGNN [4] proposes combining graph level embeddings with fine-grained node information and the use of attention to focus on the most important nodes based on the similarity metric which is chosen to train the model. Similarly to our proposed model it is trained in a supervised manner, using a predefined similarity measure. Graph Matching Networks [49] also presents the idea of computing a similarity score jointly on the pair through a cross-graph attention-based matching mechanism and compares it to independently mapping each graph to a vector and obtaining a score afterwards. The learning process is performed either pairwise with the notion of negative and positive pairs or triplet-wise by establishing a relative similarity between the target graph and a negative and positive counterpart. UGraphEmb [5] uses a siamese GNN to train graph pairs on a predefined GED measure and implicitly trains embeddings in the process. Therefore, this is by far the closest approach to ours with the addition of Multi-Scale Node Attention and a customized loss based on inter-graph proximity preservation. However, this paper focuses on embedding extraction, rather than the task of similarity itself. Additional and more detailed information can be found in Ma et al. [53].

An interesting observation to be made is that all the aforementioned approaches are run on graph datasets representing proteins, citation networks etc. In other words, none of them operate on scene graphs. This observation sets our work apart because it results in a better insight on the use of GNNs for similarity in visual concepts. Besides, it was made apparent in section 4.2 that scene graph data has not been the center of attention for graph similarity focused applications. In fact, papers combining the two concepts are quite rare in literature, with the most relevant one to our work being [55], which proposes using GCN in a weak supervision setting to produce scene graph embeddings. However, in this paper the supervision signal is the similarity score of the image captions, meaning the model is not trained on graph but rather on textual similarity.

Chapter 8

Proposal

In this section, we propose the Graph Neural Network model with which we will tackle the problem of Scene Graph Similarity. The model is trained with a Graph Edit Distance supervision signal and the embeddings extracted are used to compare input graphs, in order to produce counterfactual explanations.

We first highlight the main contributions of this thesis and then explain the proposed model in detail.

8.1 Contributions

The contributions of this dissertation are multiple and can be summarized as follows:

- We use the task of Graph Similarity to explore a plethora of Graph Neural Network variants and in turn utilize some of the most predominant ones in literature to tackle this problem. The focus is put on convolutional variants which have the ability to capture and represent both structure and semantics through information propagation in the confines of a node's neighborhood, thus taking advantage of locality while preserving invariances. The variations of the model are compared and the expressiveness and utility of each one is deduced.
- Using GNNs to address graph matching is a reasonably recent approach. Most of the literature focuses on non-neural methods, such as approximation algorithms for graph distance or kernels to accelerate the process. In contrast, this work proposes utilizing models from the newly thriving field of GNNs not only to compare graphs but also to extract meaningful representations. Our approach outperforms these methods both visually and quantitatively.
- Scene graph comparison and similarity in particular is a novel task. Previous work on scene graph data mostly focuses on graph generation, text retrieval or VQA.
- The comparative results extracted are ultimately used to produce counterfactual explanations. To our knowledge, there exists no previous literature combining these aspects to facilitate this AI interpretability task.

8.2 Proposed Model

The GNN-based model we propose follows the outline presented in Figure 8.2.1. The input consists of scene graph pairs which are then processed by identical GNN Embedding models. These models are stacked layers of a GNN variant combined with other types of standard layers like activation, normalization and dropout. The GNN model produces node-level embeddings which in turn are sampled to create global graph embeddings. This concludes the first part of the proposed model which performs the Graph Embedding Extraction and is the one we will ultimately use to extract the embeddings for all the graphs. The pair of graph embeddings (h_{G1}, h_{G2}) is then used to compute the distance between the vectors and consequently predicting a value which represents the computed Graph Edit Distance.

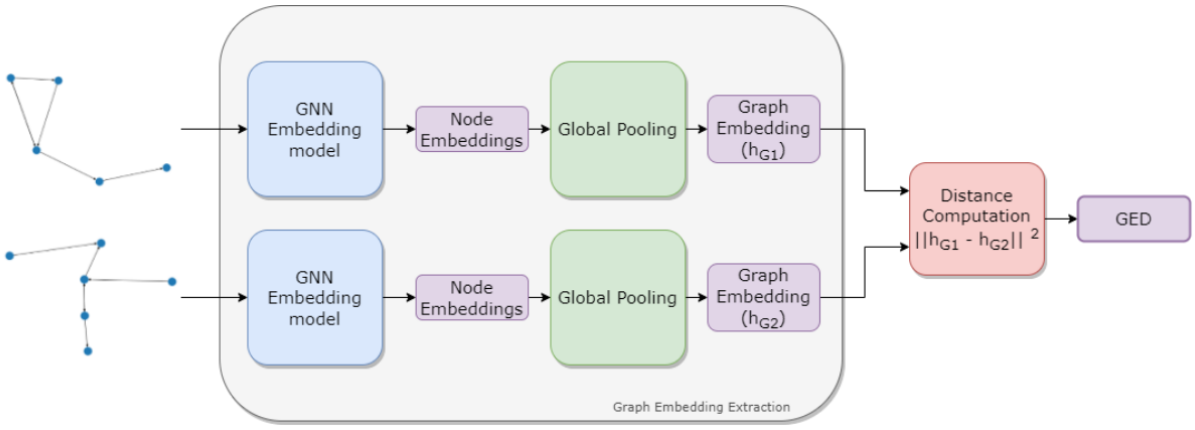


Figure 8.2.1: Proposed GNN model for Scene Graph Similarity

The training process consists of feeding a small subset of graph pairs through the model, computing the loss between the output distance and the real graph edit distance between the input graphs and back-propagating it through the network. This process is stopped after a certain amount of epochs, determined by the user. After training is over, all of the graphs are individually passed through the Graph Embedding Extraction module. The produced embeddings can be utilized separately for other downstream tasks or, in this case, compared using cosine similarity.

The **GNN Embedding model** is the heart of the implementation. In Figures 8.2.2 and 8.2.3 we can see a more detailed view of the design of the **GCN/GAT** [41, 83]variant and the **GIN** variant respectively. In both cases, there can be a variant amount of identical layers, but their input and output dimensions can vary. The layers in 8.2.2 comprise of GCN or GAT convolutions which are followed by the ReLU activation function. We also attach dropout, with a possibility of "turning off" neurons of p determined by the user, in order to prevent overfitting. The GAT variant requires additional regard in the definition of dimensions. Because of the employment of multi-head attention, input dimensions are multiplied by the number of heads in each layer. The GIN variant has a more complicated architecture. Specifically, the GIN convolution requires designing an MLP model to train the weight parameter of the central node of the convolution. As discernible in the figure, we decided to use two consecutive fully connected layers combined with ReLU activation functions, the first of which also contains batch normalization. These choices were inspired by the paper which introduced GIN [101]. In this paper, they stacked 5 identical layers containing the aforementioned components, whereas we choose to experiment with their amount. Dropout is utilized once again.

It is notable that the node embeddings produced by each layer of the GNN embedding model are concatenated before being pooled. This is an attempt to preserve more information gathered throughout the process and consequently create more expressive graph embeddings. This is a common practice also used in Xu et al. [101].

The **Global Pooling** used to sample the embeddings of vertices extracted by the GNN variants differ according to the model. For GCN/GAT average pooling was preferred inspired by [79], while for GIN we chose to sum node embeddings as proposed by its creators in order to increase expressivity. All the above operations are better performed in batches for efficiency purposes.

The **Graph Embedding Extraction** process is followed by the computation of distance between them in order to perform regression. The distance of vectors is defined as the L_2 norm of their subtraction. This method was also used in [5] which is a direct influence to our approach, along with the architecture and training style proposed in [79]. Using Mean Squared Error loss along with this distance metric results in a technique called **Multidimensional scaling (MDS)** which is employed in dimensionality reduction. Thus, distances are more accurately preserved.

The training process is finished after a predetermined amount of epochs. The now trained Graph Embedding Extraction component is subsequently used to compute embeddings for all graphs. These embeddings can then be compared as classic vectors and we decided to use the cosine similarity.

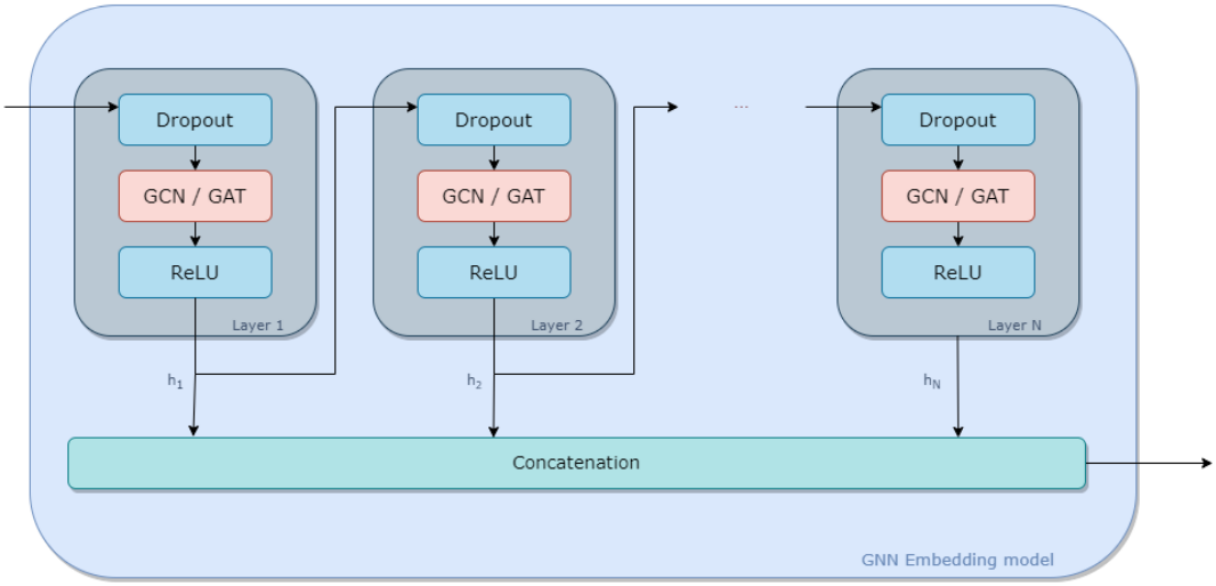


Figure 8.2.2: Design of GAT/GCN model variants.

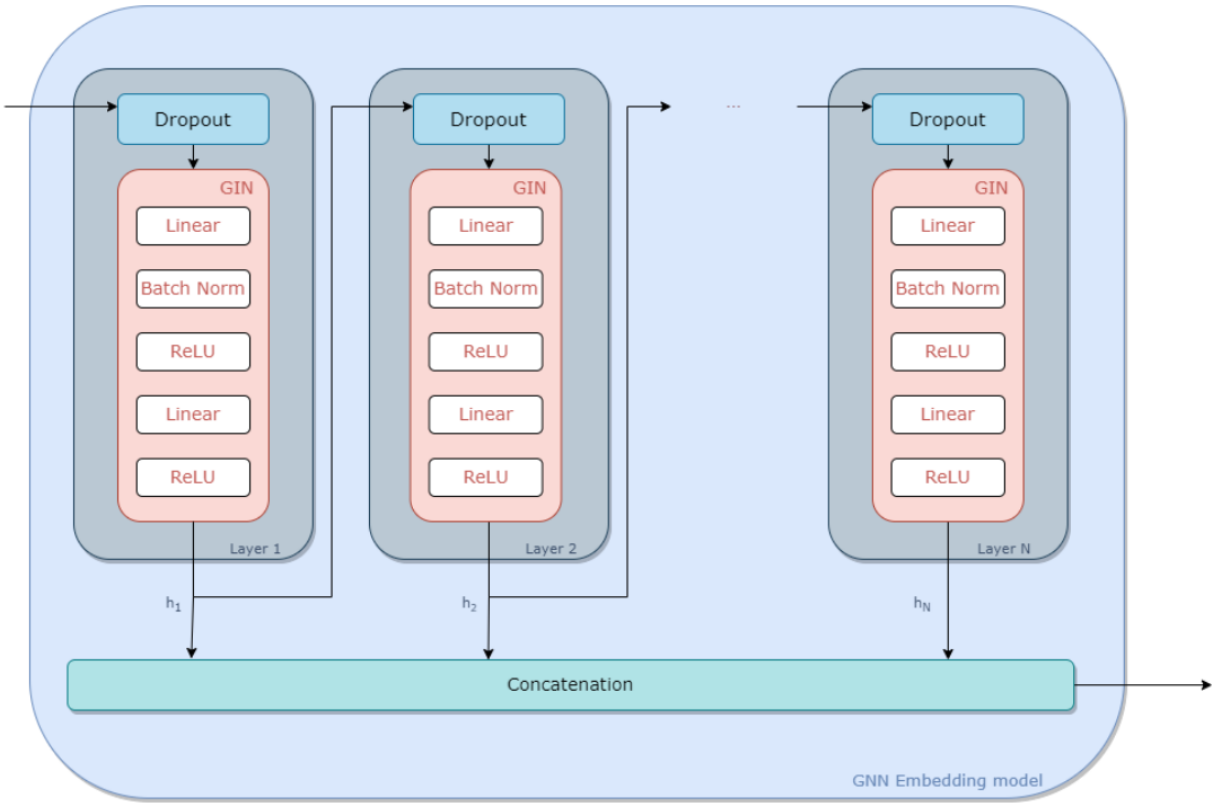


Figure 8.2.3: Design of GIN model variant.

The model can be trained with any Graph Similarity measure. In this proposal, we use Graph Edit Distance which takes both structure and semantics of graph pairs into account. It is also well-defined and not domain specific.

Chapter 9

Experiments

In order to evaluate the proposed model and compare it to other baseline techniques for graph similarity we carried out various experiments. In this section, some preliminary information will be presented about the dataset and metrics to be utilized as well as the preparation of ground truth and production of results using graph kernel approaches.

With the basics put in place, we will analyze how we experimented with the GNN-model and its variants and finally present evaluation results. In addition to the quantitative results, we will present visual representations of most similar images for a more intuitive understanding of the effectiveness of our approach.

Contents

Table 9.1: Graph Statistics

	Average	Min	Max
Density	0.20	0.14	0.47
Nodes	7.25	6	15
Edges	9.04	5	36
Isolated Nodes	0.47	0	3

9.1 Preliminaries

9.1.1 Dataset

Visual Genome

The dataset which provides us with the scene graphs used in this dissertation is Visual Genome [44]. Visual Genome is currently the largest dataset of image descriptions, objects, attributes, relationships, and question answer pairs, containing over 108K images. As mentioned, it does not solely comprise of scene graphs, but rather holds seven main components relating to images: region descriptions, objects, attributes, relationships, region graphs, scene graphs, and question answer pairs. All of the aforementioned entities which can be canonicalized have been mapped to WordNet synsets.

For this application, it was deemed that only a subset of the thousands of graphs of the original dataset would be used. The entirety of the graphs were made WordNet [58] compatible by removing all nodes or edges without synsets assigned to them. After experimentation and analysis of the transformed data, 500 scene graphs were chosen. These graphs were picked because of the density of their edges and the decreased number of isolated nodes combined with their fairly small size. Some relevant statistics can be found in Table 9.1.

Low number of neighbourless nodes signifies higher connectedness and therefore graphs richer in relations. This is an important feature for this application because it aims to extend the simple notion of unrelated concepts in a scene. Additionally, the smaller graphs make computation time more manageable since in general the graph's more expressive nature makes it more difficult to work with.

Graph Attributes

Both neural as well as some kernel methods which will be employed to tackle the problem of graph similarity assume the existence of node attributes for the graphs. These are mostly used as initial features of the nodes before the operation of each algorithm is performed and sometimes can be randomly chosen or set to be one-hot-encoded vectors of labels. Due to the fact that we strive for a solution which captures semantic similarity as well as structural, we opt for more expressive features.

The source of attributes for the nodes of the graphs was also a matter of experimentation. Due to the fact that the official Visual Genome scene graphs do not include node or edge attributes, the features would have to be hand-crafted or pretrained embeddings. With hopes of better performance and in order to avoid picking the features ourselves, as commonly done in previous work, the latter option was chosen. More specifically, embeddings of the synsets provided for each object were sought giving us the opportunity to focus on the semantics of each object and not the structural meaning of each node (i.e. degree, clustering coefficient). Embeddings of the WordNet noun hierarchy using path2vec [46] trained on Wu-Palmer similariity [98] were tested in order to mimic the inner workings of the ground truth as well as the well-known in the field of Natural Language Processing GloVe Embeddings [67]. The latter do not capture the similarity of objects in fashion of the WordNet noun subtree, but provide a well-established baseline trained on the co-occurence of words and therefore the concepts they represent.

9.1.2 Evaluation Metrics

The methods to be tested ultimately output ranked lists or recommendations of the most similar images for a given target. Thus, the following metrics were used to compare the results with the ground truth established

by the classic GED algorithm.

Hit Percentage

This metric represents the percentage of common recommendations among the top k results for each image. When mentioning hit percentage throughout this essay we are referring to a mean hit percentage, an average value for each target image. The hit percentage is the metric valued the most when assessing results because agreement with the ground truth on the most similar images is of utmost importance for this task.

Rank Biased Overlap

Rank Biased Overlap (RBO) [87] is a measure for comparing ranked lists first introduced to evaluate search engine results. Some of its characteristics are that it can be used for incomplete rankings, it handles non-conjointness - existence of different elements in the two rankings, it assigns more importance to higher ranks and is monotonic with increasing depth of evaluation. Furthermore, it is extended to be able to handle ties as well as rankings of different lengths. Its definition for infinite lists is:

$$RBO(S,T,p) = (1-p)\sum_{d=1}^{\infty} p^{d-1} * A_d$$

where S and T are the two rankings, p is a parameter determining the steepness of weights, d is the depth of the item in the list and A_d is the agreement at depth d, i.e. the overlap of the two lists at a given depth. RBO ranges between 0 and 1.

In this dissertation it is used as an additional more expressive metric to the simple hit percentage in order to ensure that the top k most similar images not only have common elements to the ground truth, but also are ranked in a similar fashion, with a focus on the higher ranks.

Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) [86] is another measure for ranking quality. In a similar manner as RBO it scores the value of the recommendation of an element in a specific position, weighting higher ranked items more heavily. One crucial difference between the two metrics is that the ranked lists used for NDCG consist of relevance scores, or in our case similarity scores of images/graphs, and not indices.

Standard NDCG can be defined as the fraction of the Discounted Cumulative Gain (DCG) over an ideal version of it. DCG can be defined as seen in the second equation below.

$$NDCG(f, S_n) = \frac{DCG(f, S_n)}{IDCG(S_n)}$$

$$DCG(f, S_n) = \sum_{i=1}^{n} \frac{rel_i}{log(i+1)}$$

where f is a ranking function, S_n is a dataset of n elements and rel_i is the relevance score of the i_th element as determined by the function f. NDCG is called standard in this case because the discount function used is the inverse logarithm decay. The IDCG is the maximum possible DCG for the given dataset obtained by using another ranking function f'.

In our experiments f' is the ground truth GED algorithm, S_n is the set of graphs and f is the method tested at each step of the process. NDCG score will be used as a quality metric, in order to evaluate the quality of the similarity scores in addition to the the aforementioned metrics which assess the rankings themselves.

9.1.3 Ground Truth

In order to evaluate the results of the models to be used, a quantitative measure needs to be established and consequently the ground truth must be constructed. In this case the ground truth is considered to be a matrix of the edit distances between each pair of graphs in the dataset. This matrix is produced by employing an

approximation of the exact Graph Edit Distance Algorithm which is based on bipartite graph matching by means of the Volgenant-Jonker assignment algorithm [37]. The costs of insertion, deletion and substitution of nodes/edges are calculated as the distance between objects, representing a node or edge, which is defined by their concept distance in the WordNet Tbox graph as proposed in [22].

We opted for using an approximation algorithm instead of the exact one due to prohibitive time it would require. For the subset of 500 graphs, the GED algorithm has to be executed 124750 times and given the NP-hard complexity of the exact algorithm, the approximate one was chosen. To be certain about the quality of results from the GED approximation in practice, we compared the two algorithms in question on a smaller but sizeable subset of pairs and were able to find that 1.5% of pairs had a difference of 20 and above, and merely 16% had a difference of 5 and above. The above is true when the average GED is about 170.

The implementation of the bipartite graph matching edit distance algorithm, which we will call *BIP-GED*, was provided by the python Deep Graph Learning library - DGL [84].

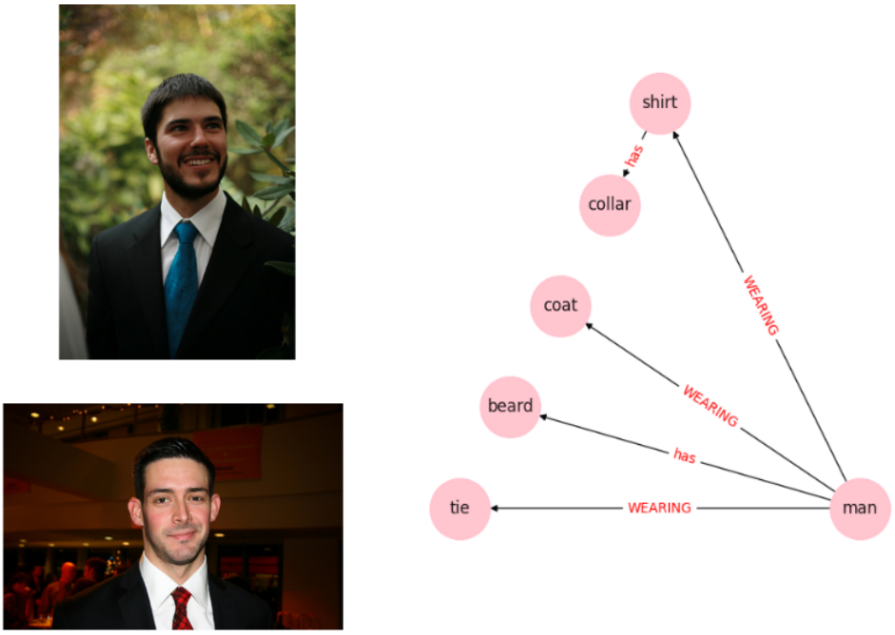


Figure 9.1.1: Images with identical scene graphs.

Below we provide some exemplary images and discuss their similarity score based on the Graph Edit Distance metric we established. In Figure 9.1.1, two different images with identical scene graphs are depicted. *BIP-GED* truthfully determines that their edit distance is in fact 0, based on the graph which represents them. Figure 9.1.3 shows the top-4 results for the top image highlighted with the red rectangle. In order to understand this ranking we also provide the scene graph for each image. As we can see, even though the top result contains another type of animal, the relations between objects are incredibly similar to the target. Thus, this pair has one of the lowest GED scores in the dataset, which is 24. Recommendations 2 and 3, with scores 48 and 55 respectively, contain the same subgraph structurally and furthermore five common objects, i.e. the sheep. The cost of deletion of additional nodes and edges is what makes them less similar than the top result. Finally, the image with a GED of 79 initially seems like an unfortunate matching, but upon inspection of its scene graph it is obvious that its graph structure is almost identical to the target - with the same edge labels, just different objects. Given that the objects are trees which are not very far from animals, the GED is not too large. The last figure, 9.1.2, shows an image with a scene graph structurally similar to the fourth result discussed previously. However, this pair receives a much larger score of 172 and the image of a bathroom floor does not even appear in the top 10. This observation proves that the semantics of objects and relations play an important role, i.e. a tile is much further in the WordNet hierarchy from a sheep than a tree is.

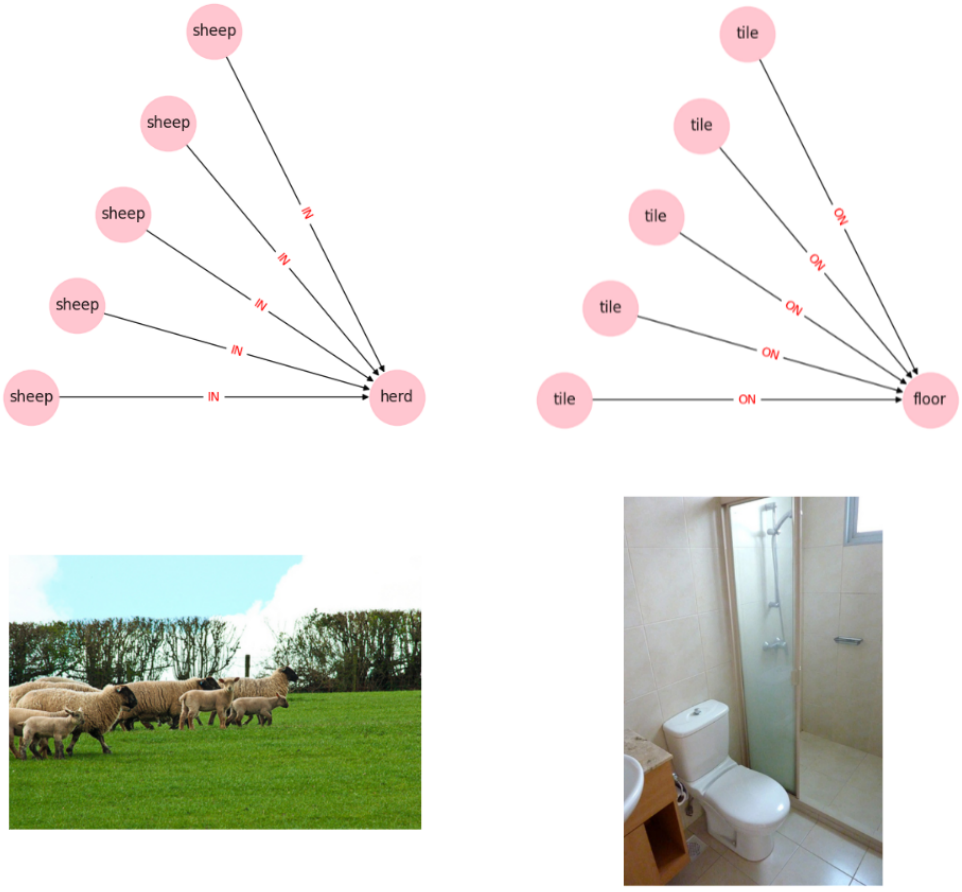


Figure 9.1.2: Example of dissimilar images.

The above observations prove the quality of *BIP-GED* as a similarity measure and therefore our choice to use it as a ground truth. It is able to prioritize both structural similarity as well as semantics and produce reliable results to evaluate our model with.

9.1.4 Graph Kernels

The experiments carried out included establishing baseline methods for solving the problem of graph comparison. As explained in section 7.2, the prevailing method to determine graph similarity is through graph kernels. As a result the following kernels were tested on our subset of graphs using the python library GraKeL [77]: kernels focused on labeled data such as the Weisfeiler-Lehman (WL) Kernel and Pyramid Match Kernel (PM) as well as kernels which consider attributed graphs such as Propagation Attribute (PA), Subgraph Matching (SM) and GraphHopper (GH).

The only kernel which can handle edge labels from the above is the Subgraph Matching kernel. In order to achieve the full potential of this method we will provide edge labels to the graphs as part of the experiments, if possible.

The operation of all kernels above has been described in the corresponding section. In a similar fashion to the ground truth, graph kernels produce a matrix of similarities between each graph pair. However, it is important to note that graph kernels provide similarity scores whereas GED is a dissimilarity measure. Their inverse nature should be taken into consideration when comparing results and therefore some form of normalization should be employed, such as max normalization.

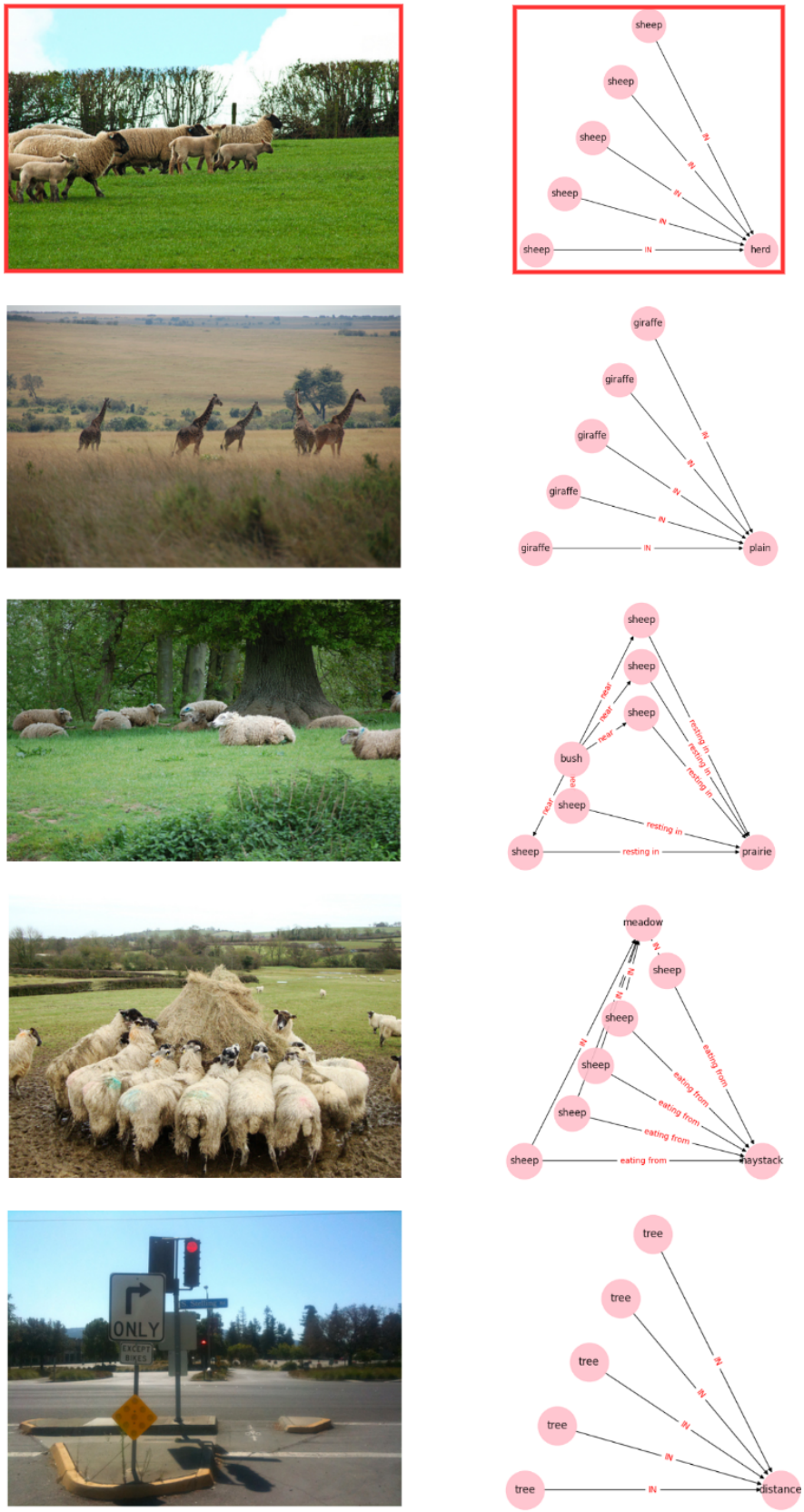


Figure 9.1.3: Top 4 results for a target image using *BIP-GED*

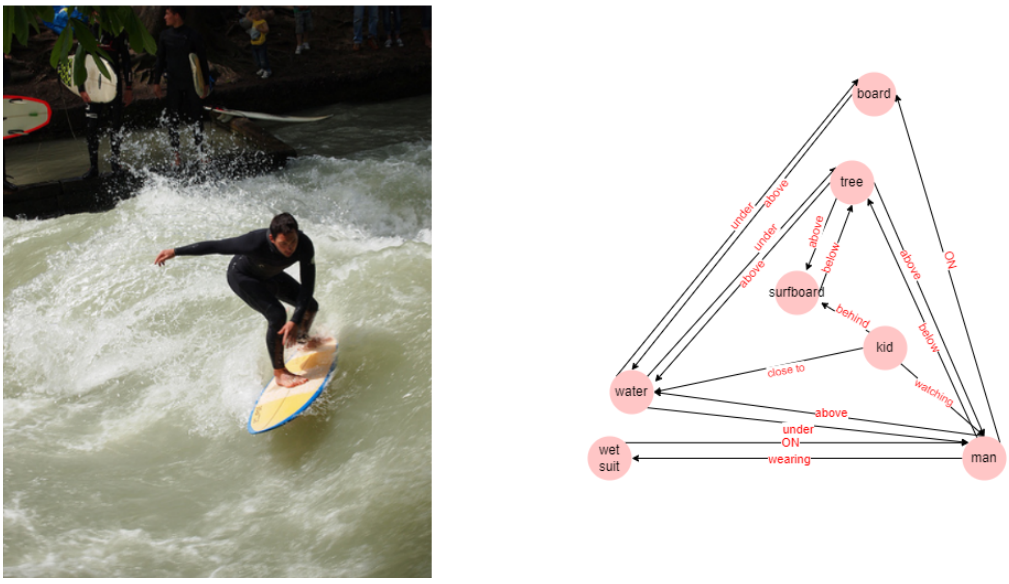


Figure 9.1.4: Image of man on surfboard and its scene graph.

The kernel methods provide adequate results on the scene graph similarity task. Firstly, all but one - the GraphHopper kernel - are able to detect isomorphic graphs which exist in the dataset and deem them as each other's most similar pair. Examining the similarity matrices produced by each kernel, it is apparent that there are many zero values present. In other words, kernels believe that some graphs could never match. The large amount of zero values and the low variance exhibited by the kernels lead to low NDCG scores. As explained previously, NDCG is used in this case as a quality measure of the scores, and if low values are demonstrated it means that despite the ability of the model to provide a satisfactory ranking, it does not provide an accurate similarity distribution between images.

In Figures 9.1.5, 9.1.6 we have provided some examples of top-5 results produced by the five kernel methods, compared to the ground truth. Starting from the image discussed in the previous section, it is rather obvious that all kernel methods provide very similar results. A common factor is that all of the results contain sheep. This observation is expected because the idea of the ontological similarity provided by WordNet is not captured. On one hand WL- and PM-kernel purely operate using labels and on the other hand PA-, SM- and GH-kernel seem to not be able to utilize the semantic information provided by the path2vec attributes in order to match graphs containing node concepts similar to that of a sheep. In contrast, in the case of the image of a man surfing, the graph kernel methods provide more diverse results. This is essentially because its scene graph representation is much more complex. Figure 9.1.6 directly reflects that a more complex graph leads to unexpected results, i.e. the image of trees in the distance or people preparing food. However, once again images of surfboards dominate the recommendation lists.

The small sample of these two comparative figures points to the efficiency of the Pyramid Match kernel and at the same time the inability of the GraphHopper kernel to provide results reflecting the ground truth or even reality in some cases. This observation will also become clear by examining the final reports. Overall results based on metrics and comparison with our proposed model are provided in section 9.3.

9.2 Model Experiments

For the evaluation of the proposed model, the framework described in 8.2 was coded using Python and the graph learning library Pytorch Geometric [21]. The basic architecture depicted in Figure 8.2.1 was established and the choices for each component were left as a hyperparameter.

Experiments were carried out separately for each one of the three **variants** we examined - *GCN, GAT and GIN*. In each run there were some common hyperparameters to determine, such as the number of epochs, the batch size, the probability of dropout and the dimensions and number of convolutional layers. The **batch**

Figure 9.1.5: Graph Kernel top-5 results on image of herd of sheep.

The ground truth is highlighted red. Kernel predictions follow from top to bottom in the order of WL-, PM-, PA-, SM-, GH-kernel.

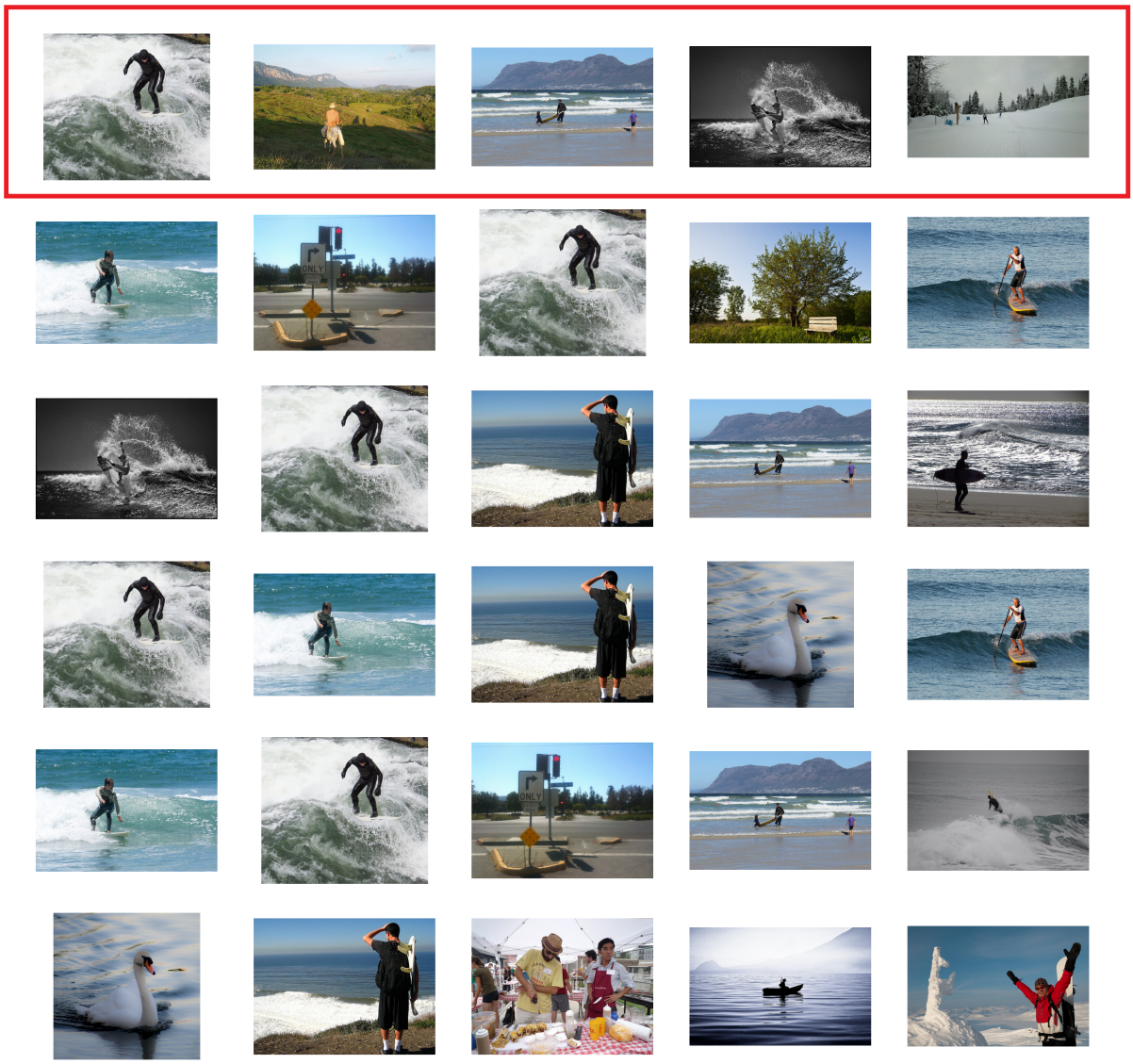


Figure 9.1.6: Graph Kernel top-5 results on image of man on surfboard.

The ground truth is highlighted red. Kernel predictions follow from top to bottom in the order of WL-, PM-, PA-, SM-, GH-kernel.

size was kept small, around 10 for most runs, sacrificing speed for model accuracy. The number of **epochs** was set to values between 20, 50 and 70, while dropout was preferred at a low level. We experimented with up to 4 **layers**, being more reserved for computationally heavier models, like GIN variants. **Dimensions** were chosen between 256 and 2048. All models were optimized using Adam and an MSE loss in order to achieve Multi-Dimensional Scaling. Thus, we also experimented with the parameters of the optimizer, i.e. the **learning rate** and **weight decay**.

An important choice to make for the models was the **number of training pairs** to use in order to fit them. The values we used were strictly kept below 10% of the dataset, not only so as to maintain smaller training times but also to instill the ability of the GNN models to create a well-mapped scene graph embedding space utilizing only a small subset of the graphs. The numbers we used in fact ranged between 2000 to 10000 pairs - the latter only in the case of GCN which is otherwise the most lightweight variant. The subset of training pairs was chosen randomly at the start of each run in order to ensure lack of bias for certain pair types.

Choosing the initial **attributes** for graph nodes is also a part of the model configuration. As mentioned before, we experimented with synset embeddings stemming from WordNet using path2vec as well as GloVe embeddings which are produced based on the idea of word co-occurence in several types of corpora. Even though, the first embeddings are more adjacent to the idea of leveraging hierarchical similarity of concepts, the latter ones are very well-trained and more established. Examining their effectiveness in combination with these Graph NN models and the differences in the quality of rankings can lead to interesting observations.

Finally, some of the variants used have hyperparameters specific to their architecture. To be more precise, GAT variants which use multi-headed attention require the choice of the number of **heads**, while GIN variants offer the option of the **weight parameter** ϵ being trainable or not. Therefore, we experimented with multiple head numbers and both trainable and non-trainable ϵ.

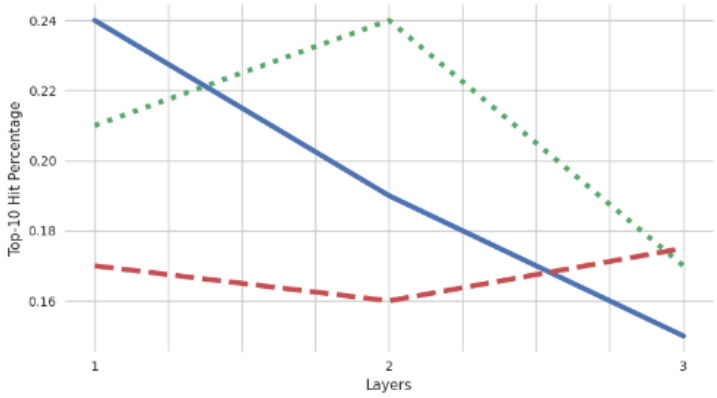
After training, graph embeddings for all graphs of the dataset were produced and compared using cosine similarity to create rankings. The ranked lists of most similar pair recommendations for each graph were juxtaposed to the ground truth and evaluated using hit percentage and mean RBO. The quality of the similarity scores and their distribution among top results were estimated with the aid of NDCG score. Finally, similarity scores directly from the model were produced for specific target pairs, so as to be compared with $BIP - GED$ scores. These results along with several visual representations of the rankings can be found in the next section.

9.3 Results

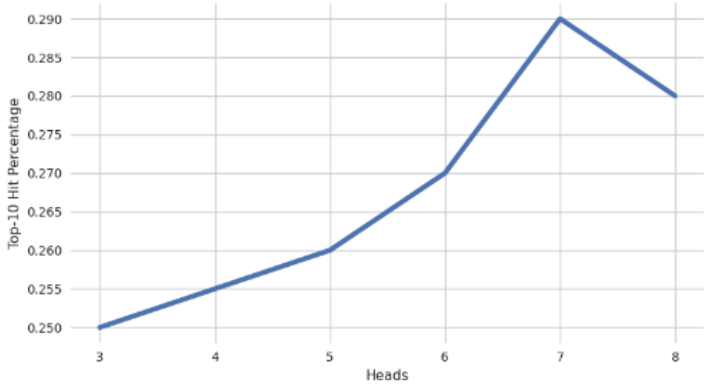
9.3.1 Variant Comparisons

Firstly, we report results and comparative observations between GNN models based on the variants and hyperparameters used. From this point on, we will refer to the proposed model as *GNN-COMP* where GNN can be any of the three variants - GCN, GAT or GIN.

The first observation made was that the number of GNN layers has a direct impact on the performance of the model. In fact, the different variants exhibited different behaviors in relation to this hyperparameter. Figure 9.3.1a provides top-10 hit percentage (HIT-P) results for each variant for layer numbers 1 to 3, given that all the other hyperparameters are fixed. Specifically, all the models were trained with 2000 graph pairs for 50 epochs. The reported values are the maximum ones achieved for this type of configuration. It is easily discernible that *GCN-COMP* models do not favor stacking, but rather accomplish better results with a single high-dimensional layer. *GAT-COMP* variants in general exhibited a similar behavior, with the exception of the 2-layer model whose results are depicted in the graph, with dimensions [512, 512], which managed to perform better. In contrast to the two other models, *GIN-COMP* variants seemed to exhibit increased hit percentage when the number of layers became larger. This observation about the GIN variant is not surprising, since the paper which presented it suggested the stacking of 5 layers to achieve desired results. However, these models tend to become unusable due to their costly nature in terms of time. If a GNN model surpasses the time it takes to find Graph Edit Distances between all pairs, then it is not sensible to prefer it, unless the performance improvement on the metrics used is remarkable. In practice, the best performing models only required one convolutional layer.



(a) Performance based on Number of Layers for different variants.



(b) Attention Heads for *GAT-COMP* models.

Figure 9.3.1: Top-10 Hit Percentage Results regarding different hyperparameters.

Concerning other hyperparameters, such as epochs or number of training pairs, their increase led to better results. Specifically, as expected, the more training pairs we provided, the more hits were observed compared to the ground truth. However, after the mark of 5000 pairs the improvement started becoming less significant. A similar finding holds true for the increase in layer dimensions; nonetheless dropout seemed to not be needed. This is because *GNN-COMP* models are not prone to overfit, due to the small amount of training examples.

As for the comparison between path2vec and GloVe embeddings as graph attributes, it became clear that GNN models favor the latter. Relevant top-10 hit percentage and mean RBO (mRBO) results for different variants are provided in Table 9.2. The reports are once again from models with a configuration of 2000 training pairs and 50 epochs. For all variants and metrics, GloVe embeddings lead to better performance - something counter-intuitive. To be more precise, path2vec embeddings adopt a similar hierarchical idea for the mapping of node objects and are additionally linked to WordNet, just like the ground truth. The better performance of the models using GloVe is most likely a result of the quality of these embeddings themselves.

Lastly, regarding the exploration of model specific hyperparameters, we have provided Figure 9.3.1b. In this figure the performance of *GAT-COMP* models trained with 5000 pairs is depicted in relation to the number of heads used by the multi-headed attention mechanism. It is no surprise that the increase of heads leads to improved performance, given that this additional attention-based feature is added to increase expressivity. However, a drop of hit percentage for a number of heads higher than 7 can be observed. As for the weight parameter ϵ of *GIN-COMP* models, surprisingly, it lead to inferior results when set to trainable, which is a direct contradiction to our expectations.

Taking into account all the aforementioned data, it is obvious that GIN is performing the worst among variants when a smaller amount of training pairs is provided. GCN and GAT on the other hand, produce

Table 9.2: GNN-COMP Top-10 Hit Percentage Comparison between Graphs with Path2vec and GloVe Embeddings

	WU-P Path2vec		**GloVe**	
	HIT-P	**mRBO**	**HIT-P**	**mRBO**
GCN-COMP	0.212	0.1594	0.2412	0.1719
GAT-COMP	0.1936	0.1364	0.2088	0.1463
GIN-COMP	0.1518	0.0826	0.173	0.093

Table 9.3: Top 10, 5 and 2 overall prediction results for all quantitative metrics.

	Top-10			**Top-5**			**Top-2**		
	HIT-P	**mRBO**	**NDCG**	**HIT-P**	**mRBO**	**NDCG**	**HIT-P**	**mRBO**	**NDCG**
WL-KERNEL	0.1982	0.1574	0.5299	0.1656	0.1266	0.4984	0.101	0.0905	0.4614
PM-KERNEL	0.2032	0.1752	0.5299	0.1796	0.1547	0.4984	0.134	0.129	0.4614
PA-KERNEL	0.1646	0.1241	0.5272	0.124	0.0964	0.4958	0.077	0.0735	0.4591
SM-KERNEL	0.1806	0.1409	0.5272	0.1412	0.1144	0.4958	0.093	0.0845	0.459
GH-KERNEL	0.0714	0.0492	0.9999*	0.0484	0.0363	0.9999*	0.03	0.027	0.9999*
GCN-COMP	**0.3142**	**0.2441**	**1.0**	**0.2476**	**0.1988**	**1.0**	0.171*	0.1515	**1.0**
GAT-COMP	0.3046*	0.2367*	**1.0**	0.2324*	0.1935*	**1.0**	**0.177**	0.1555*	**1.0**
GIN-COMP	0.2704	0.2216	**1.0**	0.2228	0.1899	**1.0**	0.165	**0.1595**	**1.0**

Best results in each column are made bold and second best are annotated with $*$.

fairly similar results. Despite being titled as the one of the most powerful, GIN does not dominate over the simpler models with regard to the metrics examined.

9.3.2 Overall Performance

The overall performance of the *GNN-COMP* models can be found in Table 9.3. We evaluated the models on hit percentage, mean RBO and NDCG of the top 10, 5 and 2 recommendations of the most similar graph pairs. The same results are reported here for all the graph kernel methods for easy comparison. The *GNN-COMP* results provided are the best achieved in each case with different configurations for each variant.

At first, it is apparent that among the graph kernel methods the Pyramid Match Kernel outperforms the others almost every time. However, the resulting NDCG scores are very low compared to the other methods. This is a clear indication that even though the relativity of the ranking is accurate, the scores produced do not reflect the ground truth. In contrast, the exact opposite observation is made in the case of the GraphHopper kernel. Its poor performance in the other two metrics, which hold a bigger significance for the task, however, prove it is not well-suited for this data.

Between the GNN-based methods, *GIN-COMP* is the least competent overall, quantitatively speaking. Both *GAT−* and *GCN-COMP* produce similar results, with each being better in different cases. For example, GCN outperforms GAT in both hit percentage and mean RBO for the top 5 and 10 results, but GAT produces better top 2 recommendations. An interesting observation is that the closer we get to the top recommendations, the more similar the scores of the GNNs become. Specifically, the GIN variant manages to almost reach the other two variants in the top-5 rankings and finally even becomes the best performing one in terms of RBO for the top-2 recommendations. Even though *GIN-COMP* manages to outperform the other models on the top-2 recommendations, its expressive power is not highlighted in this task the way we expected it to be. As often reported in literature, its theoretical capacity is not always reflected in practice.

Comparing graph kernel and GNN methods, an interesting deduction is already made by the inspection of Table 9.2. Even with 2000 train pairs, which constitutes a mere 1.6% of the combined total of pairs, GAT and GCN outperform the best graph kernel method in terms of top-10 HIT-P. It is obvious that GNN models and their expressivity are a powerful tool for similarity. However, we can also notice that the ranking is not necessarily ideal when the subset of training pairs is so small, given the mRBO scores.

It is safe to say that the GNN methods steadily outperform the kernel ones and are able to achieve major

improvements. To be more precise, there is an 11% increase of hit percentage in the top-10 recommendations between the best graph kernel and GNN method and a 7% increase of mRBO. *GCN-COMP* provides not only more accurate predictions, but also a better relative ranking. It has a clear advantage compared to the other variants with the exception of the top-2 results. Arguably, the scores achieved by the GNN methods in this case are very similar, meaning their performances are close to equal. However, given the importance of ultimately detecting the most similar (top-1) graph pair, even the slightest improvement maintained by the two other variants can be proven significant. In fact, the GIN variant achieves a 2% advance among the GNN models on the task of finding the same most similar graph as *BIP-GED*, with a score of 0.154.

It is notable that all GNN methods have a perfect NDCG score, surpassing all kernel methods. Their good performance on this metric is expected since these models are trained to mimic the similarity scores of the ground truth and thus create a very similar distribution.

Regarding the optimized models produced using *GNN-COMP*, they all consisted of one GNN layer of dimension 2048. The GIN and GAT variants were trained with 7000 pairs for 50 and 40 epochs respectively, whereas the GCN variant was trained with 10000 pairs for 70 epochs. *GAT-COMP* used 7 attention heads and the weight parameter of *GIN-COMP* was set to non-trainable. *GCN-COMP*, being the simplest of all three, required almost half the training time. The other two models can achieve comparable results with less pairs; nonetheless, they are much more computationally expensive. Therefore, in order not to excessively surpass the time needed to build the ground truth, we do not complicate them even more.

9.3.3 Qualitative Results

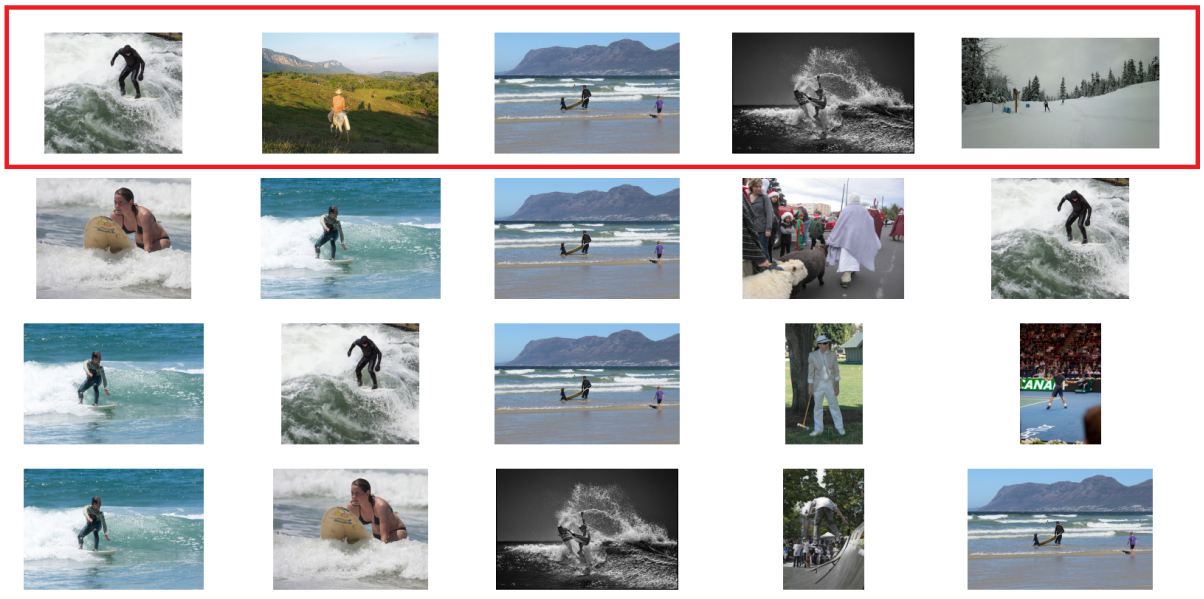
Lastly, we will examine some visual results of the rankings produced by the three best GNN models, i.e. top-k images and their scene graphs. In Figure 9.3.2 the top-5 predictions for the familiar target images of the herd of sheep and the man surfing can be found. There is an important difference to the recommendations produced by the graph kernel methods; top results are not dominated by images with the same object. The hierarchy or semantics used to construct ground truth is also reflected in the results by the GNNs. For example, in Figure 9.3.2a images with structurally similar graphs but other types of animals are recommended. In a similar fashion, in Figure 9.3.2b people are seen on other types of boards, i.e. a skateboard, or interacting with other sports equipment. These results accurately represent the complexity of semantics behind objects, while at the same time considering the graph topology.

As already highlighted by the quantitative results, rankings produced by GNN methods agree with the ground truth more consistently than recommendations by graph kernels. This observation is also reflected in the qualitative results. Moreover, even recommendations which do not fall in line with GED results seem to intuitively make sense to human perception. For instance, the images of a girl and a boy actively surfing, which are among the top-2 images predicted by the GNNs in Figure 9.3.2b, are in essence exceedingly close to the target image. Sometimes, in fact, GNN results tend to be more valid than ground truth ones. In Figure 9.3.3, *BIG-GED* recommends pictures of boats and bodies of water as the most similar to the target image of a boy brushing his teeth, whereas *GIN-COMP* suggests pictures of toothbrushes on sinks and little boys. It is arguably obvious that the latter results are more sensible. The graph edit distance algorithm disregards images of toothbrushes because their scene graphs are dense and complicated, containing very specific annotations, such as bristles or toothpaste residue. An edit algorithm finds it easier to match graphs with more similar structure than concepts in this case, which is counter-intuitive for humans. This example is an indicator of the ability of GNNs to capture similarities in both semantics and topology and especially GIN's capacity to understand global concepts.

When examining visual results, there is no direct way to decide which GNN variant performed better. One observation is that *GIN-COMP* was not as consistent in providing sensible recommendations. Specifically, sometimes it was immensely accurate, while others it recommended seemingly random images. It must be noted, however, that the latter phenomenon was scarce.

The last thing to bear in mind is that these experiments were carried out in a small subset of the Visual Genome graphs. Thus, some images and their respective graphs did not have close matchings. The algorithms tested were able to perform adequately in this case as well, recommending the most similar instances. However, a human subject would not necessarily consider the scenes depicted as similar. For example, in Figure 9.3.4 we see a girl eating a pizza. Despite pizzas and children existing in the subset of graphs used,

(a) Top-5 results on image of herd of sheep.



(b) Top-5 results on image of man on surfboard.

Figure 9.3.2: *GNN-COMP* top-5 results on images.

The ground truth is highlighted red. GNN predictions follow from top to bottom in the order of GCN-, GAT- and GIN-COMP.

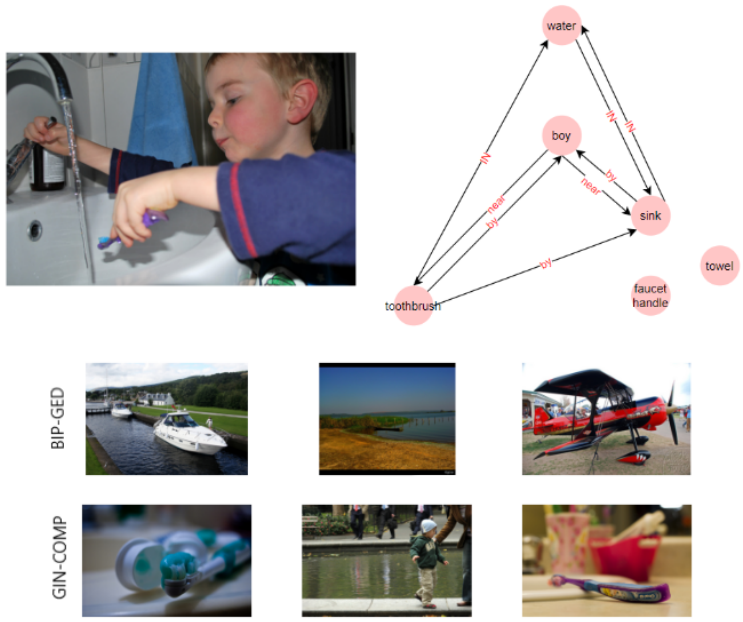


Figure 9.3.3: Example of GNN providing better top-3 results than ground truth.

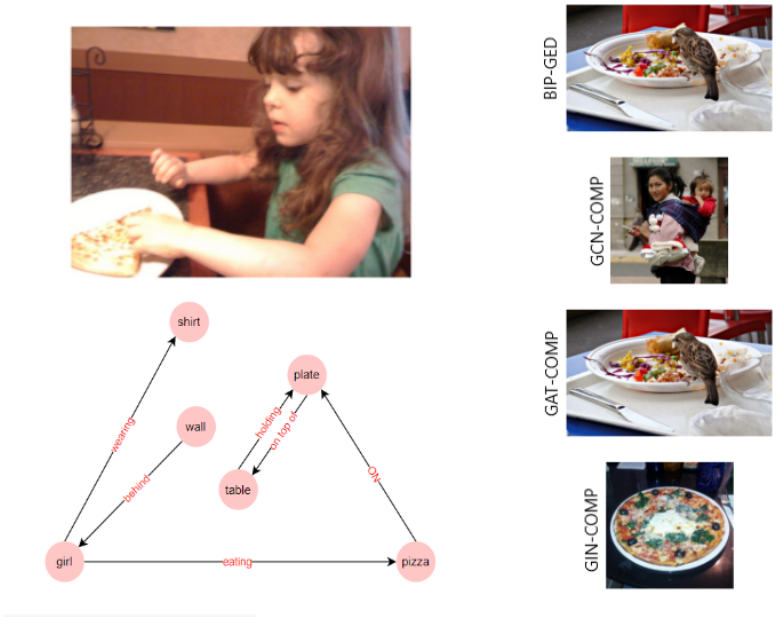


Figure 9.3.4: Example of dissimilar image to the rest of the dataset.

scenes combining those two concepts do not. In fact, the closest image is that of a bird eating food out of a plate. Two out of three models fail to find this image and instead focus on aspects of it. The predictions following the top one mostly consist of food on tables or are visually unrelated, even for the ground truth.

Chapter 10

Conclusion

10.1 Discussion

In this work we explored the use of different methods for the task of Inexact Graph Matching in order to retrieve most similar graph pairs from a given dataset and use them to create Counterfactual Explanations. Graph Similarity is a task tackled by algorithms like Graph Edit Distance or methods like Graph Kernels. However, GED is a computationally expensive NP-hard problem and even though graph kernels provide a polynomial solution, they are purely inductive. Thus, we proposed the use of Graph Neural Networks in order to produce meaningful graph embeddings trained on graph proximity measures, such as GED. We presented a model - referred to as *GNN-COMP* which can utilize various GNN variants in order to embed a graph in a space which captures similarities among graphs based on both structure and semantics and used those embeddings to compare graphs in order to find the most similar pair for each one. A ground truth was established using an approximate Graph Edit Distance algorithm based on bipartite graph matching, enriched with hierarchical semantic information about graph objects obtained by utilizing WordNet. We were able to compare all the aforementioned methods and draw important conclusions about their quality and expressivity.

We experimented with three different convolutional GNN variants for our proposed model and compared their effectiveness, complexity and expressiveness. We found that all of them quantitatively and qualitatively outperform graph kernel methods on the task of graph similarity and offer intuitive and meaningful representations for graph structures. We enriched graphs with node attributes and were surprised to find that classic GloVe embeddings were more beneficial to all *GNN-COMP* models than hierarchical path2vec ones, trained on WordNet. Between the variants, we were able to find that GCN and GAT had comparable performance on the recommendation metrics used, hit percentage and mean RBO, for top-10 and top-5 results. However, regarding the most similar graphs they all achieved exceedingly similar scores, with GIN producing the ultimate best. In fact, *GCN-COMP* achieved an almost 11% increase over the best performing graph kernel, the Pyramid Match kernel, on hit percentage of top-10 results and *GIN-COMP* reached a 15.4% of predicting the exact same most similar image as the ground truth graph edit distance approach. Despite GIN's notoriety as one of the most powerful GNNs, we were able to see only a sliver of its abilities. It managed to capture global context when the other variants did not but, all in all, all of them produced more than adequate results. It is notable that GCN did so in half the time.

The rankings produced in this thesis were made with the intention to aid in the efficient and more meaningful computation of counterfactual explanations. Specifically, detecting an image's most similar pair gives us the opportunity to only compute one edit, or explanation, instead of finding the edit path with every other sample of the dataset. Therefore, the proposed approach, which uses an image's corresponding scene graph for the retrieval of the best pair, introduces a significant speed-up in this process while leveraging the rich information the graph structure provides about the relations between concepts depicted. Moreover, the end user is left with an inductive model, meaning the trained *GNN-COMP* can be leveraged to produce embeddings for unseen scene graphs.

In conclusion, we can affirm that the use of GNNs to not only compute graph proximity, but also create graph representations is highly beneficial. Through our experiments we realized the expressive power of GNNs, the importance of their node feature initialization and their different response on training sample amount. We were able to combine the concepts of scene graphs and graph neural networks to propose a novel approach on graph similarity to aid the AI explainability task of counterfactual explanations. The representations obtained can prove to be valuable for many other downstream tasks as well, direction which we hope to inspire.

10.2 Future Work

In closing this thesis we would like to suggest a few directions to further improve on this work or inspire different interesting approaches. Firstly, the use of more Graph Neural Network variants could be explored, such as variants which leverage edge information. Doing so, will help capture information regarding the type of relationships between objects and create richer embeddings. Another approach would be to try variants like Graph Autoencoders which embed the graphs in an unsupervised manner, or even self-supervised variants based on contrastive learning and compare their performance to the supervised ones we used. As for the graphs themselves, the choice of initial attributes seemed to be crucial for the performance of models, so different hierarchical embeddings could be utilized. Furthermore, a detailed statistical and visual analysis of training graph pairs could be performed in order to find out how graphs with different properties influence the GNN models. Finally, designing a custom fine-grained loss for similarity could also aid in the model's ability to produce more accurate similarity scores.

The graph embeddings produced could be used for other applications in addition to counterfactual explanation computation. For instance, the idea of similarity could be re-imagined for other tasks. Scene graphs could be combined with other modalities to capture properties like color, size or position of objects in the image to enrich the embeddings. Moreover, since this approach detects most similar graphs in a given dataset, it could be used for other retrieval type tasks and extended to operate on knowledge graphs or graphs of documents. Finally, a method of effective clustering of embeddings could be developed in order to extract information about the intrinsic properties that map these vectors closer together. The idea of grouping them in a meaningful way would also lead in the pruning of graphs to be considered as similar and in turn introduce significant speed-up. Finally, this work could invoke the exploration of the inverse problem, i.e. answering the question why different GNN variants believe certain graphs are similar and inspire attempts for GNN explainability.

Chapter 11

Bibliography

[1] Alirezaie, M. et al. "A symbolic approach for explaining errors in image classification tasks". In: *Working Papers and Documents of the IJCAI-ECAI-2018 Workshop on*. 2018.

[2] Arrieta, A. B. et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[3] Atwood, J. and Towsley, D. "Diffusion-convolutional neural networks". In: *Advances in neural information processing systems* 29 (2016).

[4] Bai, Y. et al. "Simgnn: A neural network approach to fast graph similarity computation". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 2019, pp. 384–392.

[5] Bai, Y. et al. "Unsupervised inductive graph-level representation learning via graph-graph proximity". In: *arXiv preprint arXiv:1904.01098* (2019).

[6] Bajaj, M. et al. "Robust Counterfactual Explanations on Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 5644–5655. URL:

[7] Battaglia, P. W. et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[8] Berretti, S., Del Bimbo, A., and Vicario, E. "Efficient matching and indexing of graph models in content-based retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.10 (2001), pp. 1089–1105.

[9] Bronstein, M. M. et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[10] Bruna, J. et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[11] Bunke, H. and Shearer, K. "A graph distance metric based on the maximal common subgraph". In: *Pattern recognition letters* 19.3-4 (1998), pp. 255–259.

[12] Chang, X. et al. "Scene graphs: A survey of generations and applications". In: *arXiv preprint arXiv:2104.01111* (2021).

[13] Chou, Y.-L. et al. "Counterfactuals and causability in explainable artificial intelligence: Theory, algorithms, and applications". In: *Information Fusion* 81 (2022), pp. 59–83.

[14] Defferrard, M., Bresson, X., and Vandergheynst, P. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in neural information processing systems* 29 (2016).

[15] Derr, T., Ma, Y., and Tang, J. "Signed graph convolutional networks". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 929–934.

[16] Dervakos, E. et al. "Computing Rule-Based Explanations of Machine Learning Classifiers using Knowledge Graphs". In: *arXiv preprint arXiv:2202.03971* (2022).

[17] Developers, G. *Foundational Courses - Embeddings*. [Online; accessed 23-September-2022]. 2022.

[18] Fankhauser, S., Riesen, K., and Bunke, H. "Speeding up graph edit distance computation through fast bipartite matching". In: *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer. 2011, pp. 102–111.

[19] Feng, Y. et al. "Hypergraph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3558–3565.
[20] Feragen, A. et al. "Scalable kernels for graphs with continuous attributes". In: *Advances in neural information processing systems* 26 (2013).
[21] Fey, M. and Lenssen, J. E. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.
[22] Filandrianos, G. et al. "Conceptual Edits as Counterfactual Explanations." In: *AAAI Spring Symposium: MAKE*. 2022.
[23] Fischer, A. et al. "Approximation of graph edit distance based on Hausdorff matching". In: *Pattern Recognition* 48.2 (2015), pp. 331–343.
[24] Gilmer, J. et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.
[25] GitHub, c. *GitHub - chemplexity/molecules: chemical graph theory library for JavaScript*. en. [online]. 2022. URL:
[26] Gomez, O. et al. "ViCE: visual counterfactual explanations for machine learning models". In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 2020, pp. 531–535.
[27] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.
[28] Goyal, Y. et al. "Counterfactual Visual Explanations". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 2376–2384.
[29] Grauman, K. and Darrell, T. "The pyramid match kernel: Efficient learning with sets of features." In: *Journal of Machine Learning Research* 8.4 (2007).
[30] Grover, A. and Leskovec, J. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.
[31] Hamilton, W., Ying, Z., and Leskovec, J. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).
[32] Hamming, R. W. "Error detecting and error correcting codes". In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.
[33] Hinton, G. E. and Salakhutdinov, R. R. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.
[34] Holzinger, A. et al. "Towards multi-modal causability with graph neural networks enabling information fusion for explainable AI". In: *Information Fusion* 71 (2021), pp. 28–37.
[35] Hussein, R., Yang, D., and Cudré-Mauroux, P. "Are meta-paths necessary? Revisiting heterogeneous graph embeddings". In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018, pp. 437–446.
[36] James Im. *Introduction to PCA (Principal Component Analysis) - Medium*. [Online; accessed 23-September-2022]. 2018.
[37] Jonker, R. and Volgenant, A. "A shortest augmenting path algorithm for dense and sparse linear assignment problems". In: *Computing* 38.4 (1987), pp. 325–340.
[38] Karsten Borgwardt, O. S. *An Introduction to Graph Kernels*. en. [online]. 2010. URL:
[39] Keim, R. *How to Train a Basic Perceptron Neural Network*. en. [online]. 2019. URL:
[40] Khamsi, M. A. and Kirk, W. A. *An introduction to metric spaces and fixed point theory*. John Wiley & Sons, 2011.
[41] Kipf, T. N. and Welling, M. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).
[42] Kipf, T. N. and Welling, M. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).
[43] Kriege, N. and Mutzel, P. "Subgraph matching kernels for attributed graphs". In: *arXiv preprint arXiv:1206.6483* (2012).
[44] Krishna, R. et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations". In: *International journal of computer vision* 123.1 (2017), pp. 32–73.
[45] Kuhn, H. W. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
[46] Kutuzov, A. et al. "Making fast graph-based algorithms with graph metric embeddings". In: *arXiv preprint arXiv:1906.07040* (2019).

[47] LeCun, Y. et al. "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.
[48] Levi, G. "A note on the derivation of maximal common subgraphs of two directed or undirected graphs". In: *Calcolo* 9.4 (1973), pp. 341–352.
[49] Li, Y. et al. "Graph matching networks for learning the similarity of graph structured objects". In: *International conference on machine learning*. PMLR. 2019, pp. 3835–3845.
[50] Liartis, J. et al. "Semantic Queries Explaining Opaque Machine Learning Classifiers." In: *DAO-XAI*. 2021.
[51] Liu, Y. et al. "Graph self-supervised learning: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2022).
[52] Lucic, A. et al. "Cf-gnnexplainer: Counterfactual explanations for graph neural networks". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 4499–4511.
[53] Ma, G. et al. "Deep graph similarity learning: A survey". In: *Data Mining and Knowledge Discovery* 35.3 (2021), pp. 688–725.
[54] Ma, Y. et al. "Multi-dimensional graph convolutional networks". In: *Proceedings of the 2019 siam international conference on data mining*. SIAM. 2019, pp. 657–665.
[55] Maheshwari, P., Chaudhry, R., and Vinay, V. "Scene graph embeddings using relative similarity supervision". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 3. 2021, pp. 2328–2336.
[56] Maron, H. et al. "Invariant and equivariant graph networks". In: *arXiv preprint arXiv:1812.09902* (2018).
[57] Mikolov, T. et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).
[58] Miller, G. A. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.
[59] Molnar, C. *Interpretable machine learning*. Lulu. com, 2020.
[60] Monti, F. et al. "Geometric deep learning on graphs and manifolds using mixture model cnns". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5115–5124.
[61] Narayanan, A. et al. "graph2vec: Learning distributed representations of graphs". In: *arXiv preprint arXiv:1707.05005* (2017).
[62] Networkx. *PageRank algorithm | NetworkX Guide*. en. [online]. 2022. URL:
[63] Neuhaus, M., Riesen, K., and Bunke, H. "Fast suboptimal algorithms for the computation of graph edit distance". In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2006, pp. 163–172.
[64] Neumann, M. et al. "Propagation kernels: efficient graph kernels from propagated information". In: *Machine Learning* 102.2 (2016), pp. 209–245.
[65] Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. "Matching node embeddings for graph similarity". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.
[66] Pareja, A. et al. "Evolvegcn: Evolving graph convolutional networks for dynamic graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5363–5370.
[67] Pennington, J., Socher, R., and Manning, C. D. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL:
[68] Perozzi, B., Al-Rfou, R., and Skiena, S. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.
[69] Pires, T., Schlinger, E., and Garrette, D. "How multilingual is multilingual BERT?" In: *arXiv preprint arXiv:1906.01502* (2019).
[70] 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.
[71] Qiu, J. et al. "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec". In: *Proceedings of the eleventh ACM international conference on web search and data mining*. 2018, pp. 459–467.
[72] Ribeiro Neto, J. *Social Network — Big Data Success Case*. en. [online]. 2022. URL:
[73] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[74] Sanfeliu, A. and Fu, K.-S. "A distance measure between attributed relational graphs for pattern recognition". In: *IEEE transactions on systems, man, and cybernetics* 3 (1983), pp. 353–362.
[75] Scarselli, F. et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.
[76] Shervashidze, N. et al. "Weisfeiler-lehman graph kernels." In: *Journal of Machine Learning Research* 12.9 (2011).
[77] Siglidis, G. et al. "GraKeL: A Graph Kernel Library in Python." In: *J. Mach. Learn. Res.* 21.54 (2020), pp. 1–5.
[78] Silva, V. S., Freitas, A., and Handschuh, S. "Exploring knowledge graphs in an interpretable composite approach for text entailment". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7023–7030.
[79] StellarGraph. *Unsupervised graph classification/representation learning via distances*. [Online; accessed 6-October-2022]. 2020.
[80] Tang, J. et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 1067–1077.
[81] Tiddi, I. and Schlobach, S. "Knowledge graphs as tools for explainable machine learning: A survey". In: *Artificial Intelligence* 302 (2022), p. 103627.
[82] Vaswani, A. et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
[83] Veličković, P. et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).
[84] Wang, M. et al. "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". In: *arXiv preprint arXiv:1909.01315* (2019).
[85] Wang, X. et al. "Non-local neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7794–7803.
[86] Wang, Y. et al. "A theoretical analysis of NDCG type ranking measures". In: *Conference on learning theory*. PMLR. 2013, pp. 25–54.
[87] Webber, W., Moffat, A., and Zobel, J. "A similarity measure for indefinite rankings". In: *ACM Transactions on Information Systems (TOIS)* 28.4 (2010), pp. 1–38.
[88] 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.
[89] Wikipedia contributors. *Feedforward neural network — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[90] Wikipedia contributors. *Graph isomorphism — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2022]. 2022.
[91] Wikipedia contributors. *Machine learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[92] Wikipedia contributors. *Mean absolute error — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[93] Wikipedia contributors. *Principal component analysis — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-September-2022]. 2022.
[94] Wikipedia contributors. *Rectifier (neural networks) — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[95] Wikipedia contributors. *Semi-supervised learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[96] Wikipedia contributors. *Vanishing gradient problem — Wikipedia, The Free Encyclopedia*. [Online; accessed 22-September-2022]. 2022.
[97] Wikipedia contributors. *Word embedding — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-September-2022]. 2022.
[98] Wu, Z. and Palmer, M. "Verb semantics and lexical selection". In: *arXiv preprint cmp-lg/9406033* (1994).
[99] Wu, Z. et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.
[100] Xinyi, Z. and Chen, L. "Capsule graph neural network". In: *International conference on learning representations*. 2018.
[101] Xu, K. et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).

[102] Yadati, N. et al. “HyperGCN: Hypergraph Convolutional Networks for Semi-Supervised Learning and Combinatorial Optimisation”. In: *arXiv* (2019).
[103] Yanardag, P. and Vishwanathan, S. “Deep graph kernels”. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1365–1374.
[104] Ying, Z. et al. “Hierarchical graph representation learning with differentiable pooling”. In: *Advances in neural information processing systems* 31 (2018).
[105] Zhang, A. et al. “Dive into Deep Learning”. In: *arXiv preprint arXiv:2106.11342* (2021).
[106] Zhang, K. “A constrained edit distance between unordered labeled trees”. In: *Algorithmica* 15.3 (1996), pp. 205–222.
[107] Zhao, W., Oyama, S., and Kurihara, M. “Generating natural counterfactual visual explanations”. In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 2021, pp. 5204–5205.
[108] Zhou, J. et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (2020), pp. 57–81.