UNIVERSITÀ DEGLI STUDI FIRENZE

University of Florence
Department of Information Engineering
Ph.D. Program in Smart Computing

Graph Neural Networks for Advanced Molecular Data Analysis

Candidate
Niccolò Pancino

Supervisor
Prof. Monica Bianchini

PhD Coordinator
Prof. Stefano Berretti

Cycle XXXV, years 2019-2023

Ph.D. Program in Smart Computing
Unversity of Florence, University of Pisa, University of Siena

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Smart Computing.

Abstract

Graphs are data structures composed of collections of nodes and edges, which can be used to represent objects, or patterns, along with their relationships. Deep Learning techniques, and in particular Deep Neural Networks, have recently known a great development and have been employed in solving tasks of increasing complexity and variety. In particular, Graph Neural Networks (GNNs) have been extensively studied in the last decade, with many theoretical and practical innovations. Indeed, their main feature is the capability of processing graphs with minimal loss of structural information, which has caused GNNs to be applied to an increasing number of problems of different nature, leading to the development of new theories, models, and techniques. In particular, biological data proved to be a very suitable application field for GNNs, with metabolic networks, molecules, and proteins representing just few examples of data that are naturally encoded as graphs.

In this thesis, a software framework for implementing GNN models was developed and discussed. Furthermore, some applications of GNNs to molecular data, relevant both from the point of view of Deep Learning and Bioinformatics, are discussed. The main focus of the work is on the drug side–effect prediction problem. This is a challenging task, since predictions can be based on homogeneous as well as heterogeneous and complex data, with graphs collecting nodes and edges representing different entities and relationships. On the other side, protein–protein interfaces can be detected by identifying the maximum clique in a correspondence graph of protein secondary structures, a problem which can be solved with Layered Graph Neural Networks (LGNNs). Promising experimental outcomes offer valuable insights and permit drawing interesting conclusions about the abilities of GNNs in analyzing molecular data. Given the growing interest of the AI research community on graph–based models, these applications, inspired by real–world problems, constitute a very good testing ground for evaluating GNN computational ca-

pabilities, in order to improve and evolve the actual models and extend them to more complex tasks, particularly in the biological field.

Contents

List of acronyms used in the Thesis

AI	Artificial Intelligence
ANN	Artificial Neural Network
ASA	Accessible Surface Area
CGNN	Composite Graph Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DSE	Drug Side–Effect
GAN	Generative Adversarial Network
GAT	Graph Attention neTwork
GCN	Graph Convolution Network
GNN	Graph Neural Network
MGSEP	Molecular Graph Side–Effect Predictor
LGNN	Layered Graph Neural Network
LSTM	Long Short Term Memory
MG^2N^2	Molecular Generative Graph Neural Network
ML	Machine Learning
MLP	Multi–Layer Perceptron
MPNN	Message Passing Neural Network
NLP	Natural Language Processing
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SSE	Secondary Structure Element
SVM	Support Vector Machine
VAE	Variational AutoEncoder

Chapter 1

Introduction

Machine Learning (ML) — especially if declined in the current Deep Learning (DL) framework — is an ever–evolving field which has seen many breakthrough discoveries in recent years, allowing for solutions to be found for an increasing variety and complexity of problems. Particularly, attention is given to Graphs Neural Networks (GNNs) [1], which are a powerful tool for understanding complex relationships within graph data, and have been proven to be effective in a wide range of applications.

This thesis aims to explore the potential of GNNs in the field of biology and contribute to the ongoing research in this area.

1.1 GNNs in Bioinformatics

A graph is a data structure composed of a collection of nodes and edges which can be used to represent objects, or patterns, along with their relationships [2]. Nodes and edges can be associated with feature vectors, describing their attributes. Nowadays, graphs play an important role in many modern applications, since they are widely used to describe the information of interest in many real–world problems in different fields, including physics, social science [3], economics, and bioinformatics. For instance, for biological or chemical data, nodes denote entities, such as atoms, proteins, or genes, while edges represent chemical bonds, physical contacts, or metabolic interactions. Actually, graphs constitute the natural data domain in many bioinformatics applications, such as, for instance, molecular property prediction [4], identification of interfacing amino acids in Protein–Protein Interaction (PPI)

[5, 6], prediction of polypharmacy side effects [7], and drug design [8]. More generally, it can be observed that a graphical representation allows information from different sources to be merged in a natural way (e.g. gene regulatory networks, metabolic networks, any drugs taken) while preserving the interactions that naturally occur in a complex biological environment.

Traditional machine learning methods are not able to process graph–structured data in its native form, and require them to be encoded into vectors, with an inevitable loss of useful information and preventing models from considering the relational information inherent in the task.

GNNs are powerful connectionist models for graph–structured data, which have become practical tools for any problem involving graphs, thanks to their capability of processing relational data directly in graph form and calculating an output at each node or edge, with minimal loss of information. Indeed, GNNs assume that the input domain is represented by a set of entities and relationships between them.

Since the seminal work in [1, 9], many GNN models have been proposed, such as Graph Convolution Networks (GCN) [10, 11, 12], GraphSAGE [13], and Graph Attention Networks (GATs) [14]. GNNs have been successfully applied to a wide variety of tasks [15], spreading from computer vision [16, 17, 18] and social and recommendation systems [19, 20, 21], to biological and chemical tasks in protein–related problems [22, 4, 23, 24] and drug—discovery applications [25, 8].

Three types of tasks can be faced by means of GNNs, i.e. node–focused, edge–focused, and graph–focused problems. In particular, node–focused problems concern situations in which all the nodes of a graph, or a subset of them, are associated with a target value: intuitively, an output must be produced in correspondence of each targeted node, which can be used for classification, regression, or clustering purposes. For example, localizing a particular compound in a macromolecule, when the molecule is represented as a graph, is a node–focused task. On the other hand, edge–focused problems concern tasks in which the targets are associated to the edges: the GNN must classify, cluster, or even predict the existence of relationships between patterns. Predicting the nature of chemical bonds between atoms or molecules represents an edge–focused task. Finally, a graph–focused task concerns problems in which a unique target is associated to the whole graph, and the goal is to predict a property or to cluster the complex object represented by the graph. Predicting the mutagenicity of a particular compound

[4] is an example of graph–focused problem.

GNNs can therefore be applied to many types of problems, in both the regression and classification settings, as well as in both the inductive and mixed inductive–transductive learning framework, which makes these models extremely adaptable and capable to match the extraordinary variety of biological problems on graphs. GNNs can also be modified in order to solve even more specific tasks: they can be used to develop graph generative models [26]; attention mechanisms can make the models explainable [27]; hierarchical versions can process large graphs with complex structures [28]; composite GNNs can process heterogeneous graphs [29].

1.2 Thesis Summary

The primary objective of this thesis is to investigate the use of graph neural networks in solving biological tasks inspired by real–world problems and possibly related to molecular data. A comprehensive overview of the various ML algorithms which can be applied to structured data as well as a detailed review of the related literature in the field is provided, with a specific focus on GNNs and biological problems which can be addressed by them, in order to establish a solid foundation for the subsequent presentation and discussion of the work.

Firstly, the development of a software framework for the implementation of the original (recurrent) GNNs for a wide variety of possible scientific applications is discussed. The software has been used in all the other research works presented in this thesis. Four applications, which are relevant from the point of view of ML as well as from that of bioinformatics, are then illustrated in the following chapters.

Three applications consist in the prediction of Drug Side–Effects (DSEs) with GNNs. Predicting side–effects is a key problem in drug discovery: an efficient method for anticipating their occurrence could cut the costs of the experimentation on new drug compounds, avoiding predictable failures during clinical trials. A dataset is built for this task, which includes relevant heterogeneous information coming from multiple well known and publicly available online resources. The dataset consists of a single heterogeneous graph, where genes and drugs are represented as nodes, connected by three different sets of edges, with relationships based on gene–gene interactions, drug–gene interactions, and drug–drug similarity. Drug nodes are associated to a set of 360 common side–effects, in a multi–class multi–label node classification setting, exploiting a composite GNN model, specialized on heterogeneous graph–structured data. Since the purpose of the model is that of predicting DSEs of new drugs based on those of previously known compounds, transductive learning is exploited to better adapt the model to this setup. Given its nature, the problem is particularly challenging in the ML scope, providing an interesting application case of GNNs to a complex task (multiple non–mutually exclusive classes to be predicted in parallel), on heterogeneous data, and in a mixed inductive–transductive setting. Results have been promising in terms of accuracy compared to both other graph–based models and traditional ML methods, which are not able to use relational information, showing that encoding data in a graph structure brings an advantage over unstructured data, and that GNNs exploit the given information better than concurrent models. The method is adaptable and can be easily expanded to include more node attributes and edges without changing the ML framework. However, DruGNN operates as a black–box approach, like many other DNN–based methods: producing a much more interpretable and trustworthy version of the model in the future would boost the usability of the approach. The second application of GNNs for DSE prediction is addressed as a graph–focused multi–class and multi–label classification problem. As the drug structure can be efficiently encoded by a graph, which retains all structural information associated with the drug and which can also be enriched with relevant chemical features of the compound, GNNs are employed to predict DSEs based solely on the drug molecular graph representation. A novel dataset of molecular graphs is then introduced for the task. The experimental results show that the DSE prediction can be effectively accomplished with this method. Finally, the last application for DSE prediction includes both the setting of the previous two applications, being based on the same heterogeneous graph — including two types of nodes, representing drugs and genes, and three types of edges, for drug–gene, drug–drug, and gene–gene relationships — on which the first application is trained, and on the molecular graph representation considered in the second one. Formally, the task is still a node–focused, multi–class, multi–label classification problem: the GNN model is trained to predict the DSEs associated with the drug nodes, by processing directly graph–structured data and by generating neural fingerprints at learning time, which can be adapted to the task the model is trained for. This results in further improvements over the other methods, showing promising performance in predicting DSEs. It is a matter of future research to specialize the predictors, e.g. considering tissue specific data and DSE supervisions, for more accurate and informative insights, as well as to provide interpretability and explainability of the models, in order to improve the usability of the approaches and to facilitate the development of safer and more effective drugs.

The last GNN application consists in the prediction of protein–protein interfaces. Simulating the structural conformation of a protein in a computer environment (in silico) can be difficult and requires a significant amount of resources. Having a reliable method for predicting interfaces could enhance the prediction of protein's quaternary structures and functions. By representing a pair of interacting peptides as graphs, a correspondence graph can be created, describing their interaction and accounting for structural similarities and contacts between the pair. The correspondence graph is then analyzed in search of the maximum clique (the largest fully–connected sub–graph within a give graph), which corresponds to the location of the interface between the two peptides. Identifying cliques in this context can be challenging due to the imbalanced distribution of data (only a small number of nodes belong to cliques) and the complex nature of the data structures. GNNs provide a viable solution for dealing with this task, with their capability of approximating functions on graphs. The experimental results show that this solution is very promising, also compared to other methods available in the literature.

The proposed applications, additionally to their inherent relevance from the point of view of research on molecular data, are a good testing ground for GNNs. The first three applications are strictly correlated, as they propose three different models in order to face the same problem. In particular, the first application demonstrates the capabilities of GNNs in processing a heterogeneous relational dataset, built from multiple different data sources; the second one is based on a homogeneous relational dataset, yet it allows to tackle the same task; finally, the third one further broadens the field of study by including both the previously described settings. This application is more classical, yet it allows to face a relevant biological task, and to test GNNs on the maximum clique detection problem (known to be NP–complete) on an unbalanced dataset. Therefore, all the presented real–world applications allow to obtain relevant results, and to draw interesting conclusions, from the point of view of ML.

Additionally, this thesis also explores other works, mainly related to bioinformatics and structured data. These studies are briefly summarized and discussed in Chapter 7. The rest of this section summarizes the contributions of the thesis, presented in Section 1.2.1, and the thesis structure, described in Section 1.2.2.

1.2.1 Main Contributions of the Thesis

The main contributions of the thesis are summarized in the following.

1. A software framework, described in the publication [**A2**], based on TensorFlow 2 and Keras, for the implementation of Recurrent GNNs in multiple application scenarios, for node–focused, edge–focused, and graph–focused applications, for homogeneous and heterogeneous graph processing, and for both inductive and mixed inductive–transductive learning settings.

2. Three GNN–based DSE predictors, described in the publications [**A1**], [**A4**], and [**A7**]:

   (a) A novel relational dataset which integrates multiple information sources into a network of drugs and genes, with different types of features and connections;

   (b) Two novel GNN–based predictors on such dataset, which deal with a multi–class multi–label classification task on heterogeneous data, both in the inductive and a mixed inductive–transductive learning framework;

   (c) Ablation studies on DSEs and data sources, which underline their importance for the learning process;

   (d) Experimental validation of the three predictors, with interesting results highlighting the gap with a non–graph–based method.

3. A GNN–based method for the identification of protein–protein interfaces, described in the publication [**P1**]:

   (a) A novel dataset of graphs describing protein–protein interactions, built on reliable and publicly available sources;

(b) A GNN predictor trained on this dataset, which successfully detects cliques (corresponding to interactions) in a highly unbalanced setting, with very promising results.

1.2.2 Structure of the Thesis

The thesis is divided into chapters, sections, and subsections to present a clear and organized structure for the research conducted during the Ph.D. program.

After the present introductory chapter, Chapter 2 describes the general concepts of DL and its application to structured data, with a focus on the GNN model and its application in Biology and Bioinformatics. A review of the literature on the topics relevant to this thesis is provided in Chapter 3, with some interesting insights on theoretical and practical applications of GNNs to biological problems, as well as on the classical methodologies and state–of–the–art approaches for the tasks addressed in this thesis.

The main contributions of this work are presented and discussed in Chapters 4, 5, and 6. In particular, Chapter 4 describes a Python–based software framework for the implementation of the original recurrent GNN model. Then, three different GNN–based DSE predictors are proposed and described in Chapter 5, along with a brief description of the publicly available data sources used for such applications. Two novel relational datasets have been built from the aforementioned databases: the former, exploited both in Section 5.3 and Section 5.5, is composed of drugs and human genes, to consider all the heterogeneous data relevant for the prediction of DSEs; the latter is a homogeneous dataset composed of molecular graphs and it is described in Section 5.4. The GNN predictors have been trained and tested on these datasets, leading to very promising results. Finally, the last contribution of this thesis is discussed in Chapter 6, in which a GNN model for the prediction of protein–protein interfaces is presented, with interesting experimental results. In this application, the GNN model is trained on a dataset composed of multiple homogeneous graphs — accounting for structural similarities and contacts between pairs of peptides — in order to detect the maximum clique on such graphs, corresponding, from a biological point of view, to the localization of the interfaces between the two peptides.

Other works, carried out during the Ph.D., not related to the main contributions of the present work, yet still relevant to ML and bioinformatics, are briefly described in Chapter 7.

Lastly, Chapter 8 draws the conclusions of this thesis, suggesting possible future developments, and discussing their relevance and meaning.

Chapter 2

Deep Learning on Structured Data

This chapter introduces the fundamentals of deep learning (DL) and discusses its application to structured data, specifically graphs. It also provides an overview of the types of biological data that are particularly well suited for DL approaches and introduces the main applications that are discussed in the thesis. In particular, Section 2.1 describes DL in the wider framework of Artificial Intelligence (AI) and Machine Learning (ML), briefly introducing it from a historical point of view in Section 2.1.1, along with some insight on deep neural networks and on *ad hoc* learning algorithms (Section 2.1.2). Models and algorithms for structured data are introduced in Section 2.2. In Section 2.3, Graph Neural Networks (GNNs) are introduced, sketching the general model in Section 2.3.1, its learning procedure in Section 2.3.2, and its deeper version represented by the Layered Graph Neural Networks (LGNNs) in Section 2.3.3, as well as the composite model for heterogeneous graphs in Section 2.3.4. Moreover, a theoretical analysis of the approximation capabilities of GNNs and an overview of the principal models and applications are given in Sections 2.3.5 and 2.3.6, respectively. Finally, Section 2.4 describes how GNNs are applied to molecular data, reviewing biological problems involving graph data in Section 2.4.1, giving a deeper view on graph–based drug related tasks in Section 2.4.2, and introducing the applications of GNNs to some bioinformatics problems in Section 2.4.3.

2.1 Deep Learning

DL is a subset of a larger family of ML methods based on ingesting data representations with deep architectures, i.e. ANNs with three or more layers, to solve complex tasks: while a neural network with a single layer can still make approximate predictions, additional hidden layers can help to refine the knowledge, extracting more and more abstract and high level information from data. These neural networks attempt to simulate the behavior of the human brain, learning from large amounts of data, ranging from systems that simply replicate solutions designed by human experts, to ML models that learn their solutions from experience.

DL architectures, for example, have been applied in computer vision, automatic spoken language recognition, natural language processing, audio recognition, and in bioinformatics, which is the science that studies the use of computer tools to analyze biological phenomena such as gene interactions, protein composition and structure, and biochemical processes in cells.

From a mathematical point of view, a ML model learns a function f associating an output y to an input x, according to its parameters θ, as described in Eq. (2.1):

$$f(x, \theta) = y \tag{2.1}$$

The ML and DL models are also capable of performing different types of learning, which are usually classified as supervised learning, unsupervised learning, and reinforcement learning. Supervised learning utilizes labeled datasets, where the correct value $\hat{y}$ of $f(x)$ is known, to categorize or make predictions; this requires some kind of human intervention to label input data correctly. In contrast, unsupervised learning does not require labeled datasets, and instead, it detects patterns in the data, clustering them by any distinguishing characteristics. Therefore, unsupervised models learn from unlabeled data, using only the available information on examples. Reinforcement learning is a process in which a model learns to become more accurate for performing an action in an environment based on feedback, in order to maximize the reward.

2.1.1 From Artificial Intelligence to Deep Learning

The history of AI can be traced back to 1943, when Walter Pitts and Warren McCulloch created a neuron model inspired by the neural cells of the human brain [30], with a combination of algorithms and mathematics to mimic the thought process.

In 1950, Alan Turing wrote a paper [31] suggesting how to test a *thinking* machine: he believed a machine could be described as thinking if it could carry on a conversation imitating a human with no noticeable differences, by way of a teleprinter. His paper was followed, in 1952, by the Hodgkin–Huxley model [32] of the brain as composed by neurons forming an electrical network, with individual neurons firing on/off pulses. These combined events, discussed at a conference sponsored by Dartmouth College in 1956, helped to spark the idea of AI.

The concept of ML was first introduced in a 1959 study [33], which can be considered one of the first works on learning algorithms, although it contained only basic concepts and ideas, without proposing a learning mechanism usable and efficient.

AI has had a mostly steady evolution with two substantial interruptions known as the *AI winters*. The first AI winter occurred from 1974 to 1980, as the US government halted financing for AI research. AI researchers faced two very basic obstacles: not enough memory and processing speed which would be considered appalling by modern standards. With the promising debut of "Expert Systems" (ES), which were created and rapidly adopted by leading competitive organizations around the world, the first AI winter came to a close. The issue of gathering information from multiple specialists and disseminating it to its consumers was the main focus of AI research.

Anyway, the real breakthrough did not come before the 1980s, with the introduction of the BackPropagation algorithm [34], i.e. the essence of neural network training, based on the backward propagation of errors for tuning the parameters, reducing error rates and making the model reliable by increasing its generalization capabilities.

The AI industry experienced a severe decline in research from 1987 to 1993, which coincided with the increasing popularity of desktop PCs and the perception that expert systems were too expensive and difficult to maintain. DARPA also redirected its funding to other initiatives, leading to a reduction in funding for AI research and the Second AI Winter began. However, some individuals continued to work on machine learning, resulting in significant progress, such as the development of the support vector machine model in 1995 [35] and the LSTM (Long Short–term Memory) recurrent networks in 1997 [36]. Ironically, AI thrived in the absence of government funding and public outcry.

Overall, many of AI's historic goals were achieved in the 1990s and 2000s: Grandmaster and then world chess champion Gary Kasparov lost to IBM's Deep Blue, a chess–playing computer program, in 1997. Kasparov's defeat had worldwide resonance and represented a sensational boost for the development of AI.

ML made significant progress when GPUs were created in 1999, allowing for faster processing of data and the development of deep learning algorithms. Over a ten–year period, faster processors with GPUs increased computational speed 1,000 times. Neural networks became a contender against support vector machines, with superior results over time as more training data was provided, resulting in a collection of algorithms to model data high–level abstractions through the use of deep architectures, spreading the concept of DL and Deep Neural Networks (DNNs).

Before DL could know the fast and revolutionary development it has known in the 2010s (which is still ongoing at the present day), the issue to be solved were long–term dependencies in data, and the consequent vanishing gradient problem [37]. Many different solutions were proposed, with the development of models based on different paradigms, and designed for different data, that will be overviewed in Section 2.2.1.

ML includes many different paradigms of learning machines:

- RL [38] investigates how agents can be taught to operate so that to solve a problem by means of rewards (reinforcements);
- Support Vector Machines (SVMs) are trained to discriminate data linearly, or with kernels accounting for non–linearity [35];
- ANNs combine artificial neurons [30] to approximate non–linear functions of variable complexity;
- Self–Organizing Maps (SOMs) are a special type of ANN, based on competitive rather than inductive learning; they are an unsupervised model: the neurons are arranged into a regular grid (map) and are able to fire based on the input and the activation of their neighbors [39];
- Clustering methods are unsupervised learning algorithms that can be trained to associate data entities based on their vicinity in the feature space [40];

- Random Forests (RFs) [41] learn to build ensembles of decision trees that fit training data according to supervisions;
- Gradient boosting techniques [42] realize a strong decision model by building an ensemble of several weak decision models (usually decision trees).

A particular family of machine learning models, known as ANNs, has gained a lot of popularity in recent years for addressing problems of increasing complexity, frequently surpassing other ML techniques. ANNs use artificial neurons, the basic processing units which apply a function to the weighted sum of its inputs [30]. ANNs are commonly organized into layers of neurons, and their architecture determines their ability to resolve problems of varying complexity. More layers and units per layer improve ANN computing power, but training becomes more challenging due to theoretical and practical constraints such as memory requirements and processing speed. The number of layers defines the depth of the network: DNNs refer to networks with multiple hidden layers, e.g. layers which are not externally observable and are located between the input and the output layer. The more layers a neural network has, the "deeper" it is. A simple type of ANN is the Multi–Layer Perceptron (MLP), which takes an array of features as input and computes an output function defined according to the problem at hand. MLPs have been proven to be universal approximators for Euclidean data [43, 44, 45]: this means that the mapping between input and output vectors can be learned by an MLP, with a sufficient number of neurons [44] and at least one hidden layer, with any degree of accuracy. The first example of a DNN can be represented by the four–layer Cognitron [46]. More complex ANNs and DNNs have been developed to handle structured data and are considered in Section 2.2.

2.1.2 Learning with Deep Models

Feedforward models are the simplest type of ANNs, which consist of multiple layers of computational units, through which the information flows unidirectionally, from the input to the output layer. Each neuron in one layer has directed connections to the neurons of the subsequent layer. More specifically, in this network, the input signal is propagated through the N layers $L_1, L_2, ...L_N$ in only one direction — forward — from the input nodes, through the hidden nodes (if any) and to the output nodes, obtaining the network output y. There are no cycles or loops in the network. In a supervised learning setting, the output y is then compared to the supervision $\hat{y}$, by means of an error function $E(y, \hat{y})$, also called *loss function* or cost function. The model is optimized with a gradient descent algorithm in order to minimize E: the process starts from the calculation of the derivative $\partial E/\partial y_i$ for each unit i of the output layer L_N, which can then be used to calculate the contribution to the error of the units belonging to the last hidden layer $\partial E/\partial y_j$, for each unit j of the hidden layer L_{N-1}, as in Eq. (2.2), and so on:

$$\frac{\partial E}{\partial x_j} = \sum_{i \in L_N} \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial x_j} \tag{2.2}$$

In fact, the process is repeated in cascade — as a backward calculation of the error contributions — and is therefore called BackPropagation [34]. These contributions are then exploited to learn a better configuration and therefore to update the network parameters θ. DNNs, where the number of layers N is large, are characterized by the long–term dependency problem [37], which prevent the model to learn the dependencies between neurons located in distant layers due to the so called vanishing gradient problem: the derivative of the error gets so small when backpropagating to lower network layers, that the layers closer to the input cannot be trained from experience. The source of the problem turned out to be the use of certain activation functions, which condensed their input, in turn reducing the output range in a somewhat chaotic fashion. This produced large areas of input to be mapped over an extremely small range. In these areas of the input space, a large change will be reduced to a small change in the output space, resulting in a vanishing gradient. This behaviour prevents traditional ML algorithms, like standard BackPropagation, from successfully training DNN models [37].

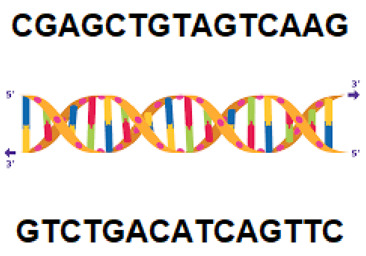
Over the years, in order to solve these issues, new ad–hoc training methods, activation functions, such as the Rectified Linear Unit (ReLU), and batch normalization, were introduced.

2.2 Machine Learning for Structured Data

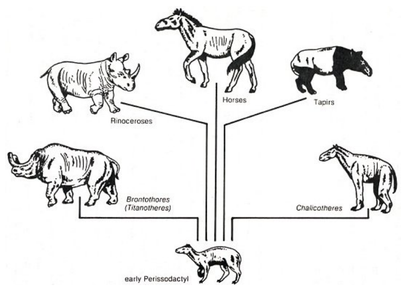
Structured data are everywhere and they are playing an increasingly important role in our daily lives as the world becomes more interconnected. As the amount and variety of data related to any given problem increases, the use of relational data and data structures becomes vital. Indeed, structured data are particularly useful in organizing and managing large amounts of information, allowing for more efficient analysis and decision–making. In a variety of fields, such as finance, healthcare, and transportation, structured data are essential for optimizing processes and achieving better outcomes. As our reliance on data continues to grow, the importance of structured data will only continue to increase [15].

In the field of AI and ML, data are often organized and processed in a particular way. The most common data type, called Euclidean, consists of entities described by a simple vector of feature values. This type of data is often used by traditional ML models and ANNs. However, real–world data can be represented with many different structures, including sequences, trees, graphs, and images. A sequence is a type of data structure that represents each entity as part of a temporal or spatial succession of similar entities, where each entity is followed (and preceded) by only one other entity: examples of data that can be represented as sequences include nucleic acids, proteins, temporal sequences of weather observations, and item queues. Trees are a generalization of sequences, where each entity can be followed by multiple entities (but preceded by only one *parent* entity), such as phylogenetic trees, decision trees and, in general, data admitting a hierarchical organization. Graphs are a further generalization of trees, where entities are represented as nodes and relationships between entities are represented as edges. Graph data examples can include protein structures, metabolic networks, molecules, power grids, traffic systems, citation networks, knowledge graphs, social networks, and even the Internet. Images are a specific type of graph, where pixels are represented as nodes and edges only exist between nearby pixels. Images can represent a wide variety of subjects, and can be used for helping with the identification of tumors through radiography and skin images, analyzing road usage through images of vehicles, or detecting emotions through photos of human faces. Often, combinations and hierarchies of these structured data types can be found. For instance, videos are sequences of images, networks of interacting compounds can be seen as graphs of graphs, phylogenetic trees based on DNA are trees of sequences, and the weather can be analyzed using sequences of graphs. Visual examples of these structured data types can be found in Fig. 2.1.

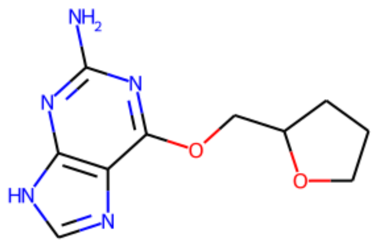
Traditionally, these structures are encoded into Euclidean feature vectors using specialized algorithms: however, this process can result in a loss of information. Deep learning solutions, on the other hand, can analyze and make decisions based on structured data in its natural form, potentially improving the accuracy and effectiveness of the analysis. Such solutions will be analyzed in Section 2.2.1.



(a) Sequence



(b) Tree



(c) Graph

(d) Image

Figure 2.1: Some structured data examples. (A) Sequences: The two strands of a DNA double helix. (B) Tree: a cladogram. (C) Graph: the structural formula of a molecule. (D) Image: a photo is a collection of numerical values of pixel colour levels.

2.2.1 Structure–Oriented Models

Deep neural networks are a type of artificial neural networks which can be designed with complex architectures to better fit the structure of the data they are intended to process. One of the earliest approaches for training neural networks with BackPropagation, the BackPropagation Through Time (BPTT) algorithm, was published in 1990 [47], just a few years after the introduction of BackPropagation for feedforward neural networks [34]. One of the first types of structured data that was processed with artificial neural networks — namely Recurrent Neural Networks (RNNs) — using the BPTT algorithm was sequences [48]. RNNs use the concept of recursion by replicating the same layer (or group of layers) for each element in an input sequence. In 1997, the introduction of LSTM networks [49] revolutionized the field by introducing the concept of cell gates, which are units that can control the flow of signals (and gradients) through the network, allowing it to store information over an extended period of time [49]. Gated recurrent units (GRUs) are similar to LSTMs, but they only have a single gate per unit (the forget gate) [50]. RNNs have been used for tasks such as natural language processing [51], protein secondary structure prediction [52, 53], motion recognition [54], stock market prediction [55], and speech recognition [56]. These same application areas have also been explored using one–dimensional Convolutional Neural Networks (CNN–1D) [57] and transformers [58]. Transformers have had a significant impact on the field by introducing an attention mechanism that allows the network to weigh the importance of different elements in a sequence and overcome biases introduced by the order of the elements [58].

Images are traditionally more complex to process, since they can be large in term of size and memory consumption. Moreover, learning on images with traditional ML models such as MLPs is not efficient, since slicing them to fit into vectors can lead to structural information loss, as well as processing them can be computationally expensive or even infeasible. Convolutional neural networks were developed as a solution to this problem. Based on a theory from 1989 [59] and first introduced in 2012 [60], CNNs use small patches of pixels as inputs and produce a single output value, repeated across the whole image, without any need to slice the image. This process can be repeated with different convolutional filters in the same layer, or with multiple convolutional layers. CNNs also often include pooling layers, to reduce the input size, and dense layers. There are many different types of CNN architectures that have been developed for a range of tasks, including image classification [60], segmentation [61], object detection [62], and image generation [63], introducing deeper models as the research goes on [64]. These models change the paradigm of forward propagation: each residual block (composed of a small number of layers) learns to refine the output of the previous residual block, shortening the gradient path in BackPropagation [65]. CNNs can be very deep, with hundreds of layers, and often use residual connections between layers to improve their performance. They can also be used to process 1D sequences, 3D images, and videos, by generating a recurrent CNN. In this regard, 1D, 2D and 3D grids can be considered as particular kinds of graphs, yet CNNs cannot be used to process any type of graph. Processing and learning on graphs are challenging tasks in ML, and various methods have been developed to extract Euclidean information from them, including techniques based on random walks [66] and kernels which can approximate functions on graphs [67], with many applications in bioinformatics [68]. Some specific types of graphs, like trees and directed acyclic graphs, can be processed using models from the recurrent neural network family [69, 70]. In this context, the breakthrough discovery was the theorization of neural networks which can process graphs by adapting their architecture to the input graph topology, namely GNNs [1]. These models will be extensively described and discussed in the following Section 2.3.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a well–known class of machine learning models for graph–structured data processing. The first theorization of GNNs dates back to 2005 [71] — with the full mathematical formulation proposed in 2009 [1] — and describe them as networks which replicate the topology of the input graph, and exchange messages between neighbor nodes to produce an output on selected data. GNNs typically work on a graph dataset D composed of one or more graphs, processed independently.

A graph is a non–linear (non–Euclidean) data structure composed of a collection of nodes and edges. Formally, it is defined as a tuple $G = (N, E)$, where N is the set of nodes and $E \subseteq N \times N$ is the set of edges. Nodes represent objects or entities, and relationships between them are represented by edges. Nodes and edges can be associated with values or vectors of values, describing their attributes, and defined as node feature vectors, l_n, $\forall n \in N$, and an edge feature vectors, $e_{n,m}$, $\forall (m, n) \in E$, respectively. Moreover, a neighborhood function Ne, which associates each node n to the set of its neighbor nodes $Ne(n) \subset N$ can be defined, based on the edges E.

2.3.1 The Graph Neural Network Model

A GNN assigns a state s_n to each node $n \in N$ and updates it iteratively by sending messages through the edges connecting n to its neighbors $Ne(n)$. The GNN theoretical model is fully described by two parametric functions, f_w and g_w, which, respectively, regulate the state updating process and the output calculation.

The GNN model approximates an output function g_w, expressing a property of the whole graph G, of its nodes or a subset of its nodes $N_{out} \subseteq N$, or of its edges or a subset of its edges $E_{out} \subseteq E$. To do so, a state $x_n \in \mathbb{R}^{d_x}$ is associated to each node $n \in N$, and then iteratively updated by sending messages through the edges connecting n to its neighbors $Ne(n)$.

The state dimension d_x as well as the number of state updating iterations K are hyperparameters of the GNN, while the states are usually initialized by sampling from a random distribution centered on the origin of $\mathbb{R}^{d_x}$. Given the randomly sampled initial states x_n^0, $\forall n \in N$, the state of a generic node n at iteration t can be calculated as in Eq. (2.3):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ A(\{(M(n,\ m,\ t)) : m \in Ne(n)\})) \tag{2.3}$$

where $M(\cdot)$ and $A(\cdot)$ are the the function defining the message coming from the neighbourhood $m \in Ne(n)$ to node n, and the aggregating function defining how these messages are aggregated, respectively. M can be any function returning the message in the form of a vector whose computation is based on the destination node n, the source node m, and the label $e_{m,n}$ of the edge (m, n) connecting the two nodes. M could even be learned with a neural network. In this thesis, M always takes the general form defined in Eq. (2.4) with the possibility of excluding l_m or $e_{m,n}$ (when nodes/edges are not labeled) from the computation:

$$M(n,\ m,\ t) = (x_m^{t-1},\ l_m,\ e_{m,n}) \tag{2.4}$$

On the other hand, the aggregation function A can be any function defined on a set of vectors (the messages), each having dimension d_m and returning a single vector of the same dimension. A is usually the sum, average, or maximum of the single components of the message, but it could even be learned with another neural network, as for example in the (convolutional) aggregations of GraphSAGE [13]. In this thesis, A is either the sum or average of the messages, as described by Eq. (2.5):

$$\begin{aligned} A_{sum} &= \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m,\ e_{m,n}) \\ A_{avg} &= \frac{A_{sum}}{|Ne(n)|} \end{aligned} \tag{2.5}$$

Since the two aggregations are similar, a hyperparameter a equal to $1/|Ne(v_i)|$ or 1 can be defined to select the aggregation function, obtaining the average or the sum, respectively. Combining all these concepts, the state updating function in its general final form can be rewritten as in Eq. (2.6):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m,\ e_{m,n})) \tag{2.6}$$

The GNN implements a recurrent algorithm for exchanging useful information between nodes and their neighbors, for a pre–set number of iterations T or until the state computation dynamics reaches a stable equilibrium point at the iteration $t \leq T$; then, the final versions of the node states x_n^T, $\forall n \in N$, are fed in input to the output network, approximating the output function g_w, which can be defined for the three types of problems addressed by GNNs: node focused, edge focused or graph focused. In node focused problems, the output is defined for each node $n \in N_{out}$, as a function of its state and label, as in Eq. (2.7):

$$y_n = g_w(x_n^T,\ l_n) \tag{2.7}$$

In edge focused problems, the output is defined for each edge $(m, n) \in E_{out}$, as a function of the states of both the source node m and the destination node n, and, if it exists, the label $e_{m,n}$, as in Eq. (2.8):

$$y_{m,n} = g_w(x_n^T,\ x_m^T,\ e_{m,n}) \tag{2.8}$$

Finally, in graph focused problems, the output is calculated over each node $n \in N_{out}$ as in node based problems, and then averaged over the output nodes. This is defined in Eq. (2.9):

$$y_G = \frac{1}{|N_{out}|} \sum_{n \in N_{out}} g_w(x_n^T,\ l_n) \tag{2.9}$$

2.3.2 Learning with Graph Neural Networks

Based on an information diffusion mechanism, GNNs can process homogeneous and heterogeneous graph–structured data. In particular, GNNs create an *encoding network*, a recurrent neural network architecture which replicates the topology of the input graph, by means of two MLP units, one implementing the state transition function f_w of Eq. (2.6) at each node and the other implementing the output function g_w (on targeted nodes or edges). The GNN replicates the MLP units on each node of the input graph and unfolds itself in time and space, generating a feedforward architecture, known as the *unfolding network*, in which each layer contains copies of all the elements of the encoding network and represents an iteration of the implemented algorithm. Connections between neurons belonging to subsequent layers reproduce exactly those of the encoding network. Weight sharing is exploited between all the copies of the MLPs, allowing to manage long–term dependencies between distant nodes in the graph, for both the state and output MLPs. The information associated with each node can thus be propagated through the whole graph in a sufficient number of iterations; then an output on nodes, edges, or whole graphs — depending on the problem under analysis — is produced by the MLP implementing g_w.

A sketch of the unfolded encoding network is provided in Fig. 2.2.

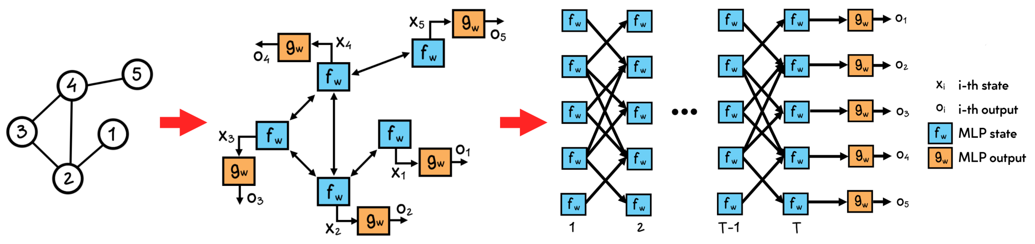


Figure 2.2: Sketch of how the GNN model produces the encoding network and its unfolded version on an example input graph (for a node focused problem). In the unfolded network, the topology–like replica of the input graph is guaranteed by the connections between layers.

Note that the unfolded encoding network corresponds to a DNN with recurrent layers: given the numbers of layers in the state MLP, indicated with L_f, and in the output MLP, denoted with L_g, the network has a depth of $K \times L_f + L_g$ layers. Weight sharing in space and time makes the model scalable, invariant in the number of parameters to graph size and number of iterations, and less prone to overfitting. Consequently, to optimize the parameters of the network, and therefore implement the learning process, it is sufficient to apply a standard optimization algorithm based on Back-Propagation. Typical examples include stochastic gradient descent or the Adam optimizer [72]. A loss function, depending on the problem under analysis (e.g. cross–entropy for classification), is applied to the outputs and the targets, computing the error. The error gradient is then calculated with respect to all parameters of the network, averaging the contribution over all the replicas of each parameter, and applying the resulting value to all the replicas of the same weight.

2.3.3 Layered Graph Neural Networks

In 2010, the base GNN model was further refined by introducing its deeper version, the Layered Graph Neural Networks (LGNNs) [73], to face the Long–Term Dependency problem, which corresponds to the inability of the network to correctly process a complex structure, due to the local nature of the GNN training process, which does not allow state computation to be influenced by labels and states of distant nodes.

The LGNN architecture is characterized by multiple GNNs connected in cascade, in which each layer can be trained separately using always the same target, and using invariably the original graph as input, though with node labels enriched by the results calculated in the previous layer. Formally, the first layer is a standard GNN operating on the original input graphs. Each layer after the first is trained on an enriched version of the graphs, in which the information obtained from the previous layer is concatenated to the original node labels. This additional information consists in, either, the node state, the node output, or both. Formally, the label of node n in the i–th layer, $i > 1$, is $l_n^i = [l_n, x_n^{i-1}]$ or $l_n^i = [l_n, o_n^{i-1}]$ or $l_n^i = [l_n, x_n^{i-1}, o_n^{i-1}]$, where x_n^{i-1}, o_n^{i-1} are, respectively, the state and the output of node n at layer $i-1$ of the cascaded architecture. Given the operation described for the labels, all the mathematical formulations remain valid for each layer. The output of a node in a GNN, y_n, is influenced by the node state x_n, which in turn is influenced by the node's neighbors $Ne(n)$: if this information is included in the training data for each GNN layer, then the model is able to expand the node's neighborhood, layer by layer.

This means that the LGNN can progressively widen the neighborhood of a generic node, continually increasing the amount of information it takes

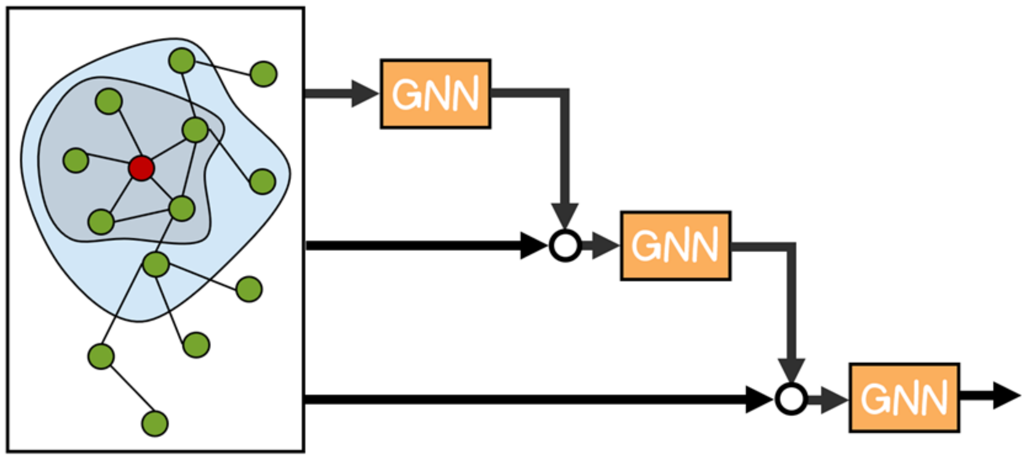


Figure 2.3: Example of an LGNN with 3 layers.

into account from nodes which are further and further away from the one under analysis. Moreover, the solution proposed by the previous layer (in the form of states or output vector, or both of them) is integrated to the input of each layer after the first, significantly addressing the long–term dependency issue: each GNN, therefore, becomes an expert which solves the considered problem using the original data and the experience obtained from the GNNs of the previous layers so as to "correct" the errors made by the previous networks, rather than solving the whole problem. As a result, LGNNs progressively refine the model's output and obtain a greater learning capability with respect to a single–layered GNN [73].

The architecture of a LGNN model applied to a generic graph is shown in Fig. 2.3.

2.3.4 Composite Graph Neural Networks

GNNs and LGNNs can be extended to the domain of heterogeneous graphs, where multiple types of nodes coexist, as they represent different kinds of objects and may have different numbers and types of features: the GNN model for heterogeneous graphs [29] is known as a Composite Graph Neural Network (CGNN), while the LGNN model in this case is called Composite Layered Graph Neural Networks (CLGNN). Heterogeneous graphs are often used to represent information about different types of entities and/or relationships: typical examples of this setting are knowledge graphs, in which entities of different types need to be encoded into a single relational graph, or molecular graphs, since atom species can be distinguished and represented by different node types.

Without loss of generality, in this section, only the case of CGNNs will be analyzed. CGNNs process graphs with edges representing different types of relationships considering a one–hot encoding of the relationship type as a label, possibly to be concatenated to the edge features, if any. Node types instead are treated as subsets of the node set N, and each type has a dedicated state network. In this setting, given the number of node types nt in the dataset, the CGNN learning process is based on multiple MLPs for the nodes state updating process, one for each type of nodes, and a variable number of output networks, depending on the problem under analysis. Each state network F_i learns its own version $f_{w,i}$ of the state updating function in Eq. (2.6). In this way, each node type is associated with a unique MLP which learns a unique state transition function. All the F_i are characterized by the same output dimension, corresponding to the CGNN hyperparameter d_x. To allow nodes belonging to different types to exchange coherent messages, the node label is not considered as a part of the message, as it can assume different dimensions and meaning for different node types. The state updating function $f_{w,i}$ can therefore be rewritten as in Eq. (2.10):

$$x_n^t = f_{w,i}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ e_{m,n}))\ :\ i = T[n] \tag{2.10}$$

where, to select the correct $f_{w,i}$, a vector T associating each node $n \in N$ to its type i is given to the GNN as part of the dataset.

The output functions described in Eq. (2.7), (2.8), and (2.9) are still valid, with the only difference represented by the number of output network used for calculating the output: in the most general case, for node focused applications in which N_{out} contains nodes of all the types defined in the dataset, the CGNN model requires nt output networks; in the simple case, in which N_{out} refers to a single node type — as in all the applications described in this thesis —, the model still requires nt different state networks, but only a single output network is needed, since the label of only one node type is considered in calculating the output for each $n \in N_{out}$. As a consequence, in the latter case the parameters the GNN has to learn are distributed in $nt + 1$ MLPs, in contrast to the two MLPs of the homogeneous setting.

The rest of the learning process is exactly the same as the homogeneous case, as explained in Section 2.3.2, with the only difference consisting in the number of state networks — and possibly of output networks — to build the encoding network and consequently the unfolding network. The CGNN learning process scheme on a heterogeneous graph is depicted in Fig. 2.4

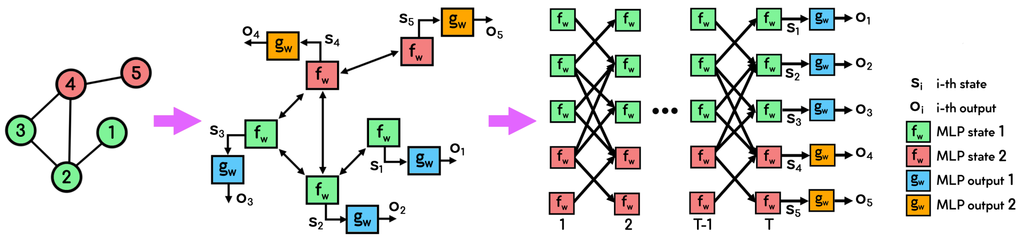


Figure 2.4: Scheme of a CGNN learning process on a heterogeneous graph. From the left: The heterogeneous input graph, where green and red circles represent two types of nodes, and connections belong to a single undirected edge type. The encoding network, generated by means of the MLPs implementing the state transition functions $f_{w,i}$ (red and green rectangles, one for each node type) and the output functions $g_{w,i}$ (blue and orange rectangles). The unfolding network, in which the information can flow from the input to the output, passing through the aforementioned MLPs and the graph connections defining this feedforward network. Note that for a non–composite GNN, learning on a homogeneous graph, the scheme is the same, with a unique MLP for state transition, since only one type of nodes is admitted.

2.3.5 Approximation Power of GNNs

The approximation capabilities of DNNs, and in particular MLPs, have been studied and described in various theoretical works [43, 44, 74]. However, since GNNs operate on the graph domain, they have to withstand different challenges, also concerning symmetries in the data structures. In [9], published in parallel with the original model [1], the computational power of GNNs was analyzed.

Given a graph $G = (N, E)$, to analyze the computation of the state of a generic node n in the graph, the concept of unfolding tree is introduced: the unfolding tree is built by taking n as the root, adding its neighbors as child nodes, and then recursively adding the neighbors of each child node as its children. In other words, an unfolding tree is obtained by unfolding G up to an arbitrary depth k using n as the starting point, as described in Eq. (2.11):

$$T_n^k = \begin{cases} \text{Tree}(x_n, \ \{T_m^{k-1}, \forall m \in Ne(n)\}) & \text{if } k > 1 \\ \text{Tree}(x_n) & \text{if } k = 1 \end{cases} \tag{2.11}$$

The unfolding tree T_n^k represents all the information on node n available to the GNN after k iterations. For k larger or equal to the diameter of G, two nodes n and m with identical unfolding trees $T_n^k = T_m^k$ are indistinguishable to the network and are said to be unfolding equivalent. It was demonstrated that GNNs are universal approximators for all functions that preserve the unfolding equivalence (i.e. functions that produce the same result on any pair of unfolding equivalent nodes) [9]. An alternative way to assess the computational capabilities of GNN models has been presented in recent publications [75, 76]. It is based on the Weisfeiler–Lehman graph isomorphism test [77], which associates a unique, canonical representation to each graph, based on the connections between nodes (i.e. the node adjacency). If two graphs have the same representation, they are considered isomorphic, meaning that they have the same structure. However, the one–dimensional version of the Weisfeiler–Lehman test (1–WL) cannot distinguish between all possible pairs of graphs, because the same representation can be shared by multiple non–isomorphic graphs. Therefore, higher–dimensional versions of the test, such as the D–dimensional test (D–WL), have been developed. These tests are based on sets of D nodes and are more effective at distinguishing between non–isomorphic graphs. The ability to distinguish between non–isomorphic graphs increases with D.

GNN models can be classified based on their ability to mimic the Weisfeiler–Lehman test: those which can replicate the 1–WL test are classified as 1–WL models, which means they are at least as powerful as the one–dimensional Weisfeiler–Lehman test; however, many currently used GNN models are not as powerful as 1–WL, because they are unable to simulate the test. Interestingly, the 1–WL test is analogous to an iteration of neighborhood aggregation in recurrent GNNs: as a result, these models have been demonstrated to be all 1–WL if injective neighborhood aggregation and output functions [75] are used. Currently, no GNN model has been able to simulate higher–order Weisfeiler–Lehman tests like, 2–WL or more [75], although some research has been done on developing higher–order GNNs using non–local neighborhood aggregation [78]. Moreover, it has been demonstrated that both unfolding trees and the Weisfeiler–Lehman test can be used to evaluate the approximation power of GNNs [79].

2.3.6 Applications of Graph–based Models

Since the seminal work [1, 9], many GNN models have been introduced [80], which can be classified, depending on how the information is propagated among nodes, in two broad families: recurrent GNNs, and convolutional GNNs. The former, which also include the original model [1], are based on message passing between nodes and apply the same weight matrices in the recurrent calculation of node states, whereas the latter apply different weights at each time–step t. Moreover, Convolutional GNNs, also known as Graph Convolution Networks (GCNs), are based on the concept of graph convolution: similarly to what happens in CNNs, a convolutional filter is applied on each node (and its neighborhood) to calculate its label in the subsequent layer, or its output.

One key difference between standard convolutional layers and graph convolutional layers is the way they handle spatial relationships between data points. In standard convolutional layers, the spatial relationships between data points are fixed and predefined by the grid structure of the data. In graph convolutional layers, on the other hand, the spatial relationships between data points are dynamic and can be learned from the data itself, since they are defined by the edges of the graph. This makes graph convolutional layers well–suited for tasks where the spatial relationships between data points are complex or not well–defined.

Examples of recurrent GNNs are Graph Nets [81], Gated Graph Sequence Neural Networks [82], Message Passing Neural Networks [83], and Graph Isomorphism Networks [75]. The first convolutional GNNs to be introduced were standard GCNs [10], followed by spectral convolution models [11, 12]. GraphSage generalized the concept of convolutional GNN by introducing different types of neighbor aggregation [13]. GCNs were also combined with attention mechanisms in Graph Attention Networks [14], improving the predictions with information on important relationships [84] and dealing with explainability issues [27].

Graph–based models can be applied on any type of graph, both in real–world applications and synthetic problems. For example, in the Web domain, GNNs have been used for spam detection [85], community detection [10], content interaction prediction [86] and sentiment analysis [3]. They have been employed also in prediction of logical relations in knowledge graphs and future links in citation networks [87], and as recommendation systems

[88, 89]. Many node or graph classification and regression tasks have been solved by applying GNNs [15], also in a heterogeneous setting [90]. Models of the GNN family have showed performance at or close to the state–of–the–art in problems of graph matching [91], weather forecasting [92], power grid analysis [93], and many others. In the biological domain, GNNs can calculate molecule properties [83, 94], predict protein–protein interfaces [23], and classify compounds according to their mutagenicity [4] or activity against HIV [95, 22], just to list a few applications.

2.4 Biological Problems on Graphs

Computational methods have enabled new research avenues in biology and medicine, resulting in the development of interdisciplinary fields such as bioinformatics and medical informatics. DL techniques are becoming more widely used in these fields, offering innovative solutions to previously untreatable problems, reducing the expenses and duration of traditional methods, and improving existing processes by making them more efficient.

2.4.1 Graph Data in Biology

Biological data are often naturally represented with graphs. Molecules have always been perceived as graphs, in which atoms correspond to nodes (possibly labelled with some chemical and physical information such as atom type, molecular weight and polar surface area), and chemical bonds correspond to edges, whose label can be the type of bond. These structures are also known in the literature as molecular graphs. Molecular graphs allow to predict many properties of each molecule, such as mutagenicity, anti–HIV or anti–cancer action, and other levels of activity. Other structures like polymers, proteins, and nucleic acids can be effectively represented as graphs, with nodes representing different structural levels (e.g. substructures, protein secondary structures, DNA blocks) and edges representing interactions between these components. This type of representation is useful for understanding and solving key biological problems, such as predicting interactions between proteins and ligands, proteins and proteins, and proteins and nucleic acids. Additionally, a hierarchical model can be used, in which each node is expanded into its own graph substructure, such as, for example, when a graph containing nodes that describe interacting proteins is expanded by representing each protein as a molecule — in this case it is referred to as graph of graphs.

Graphs can also be used to represent logical information, such as knowledge graphs of biological entities. In these graphs, each node represents an entity and each edge represents a relationship between two entities. ML models can then be trained to predict new, biologically relevant relationships between entities based on the known relationships: this is often done in a heterogeneous setting, where the entities come from different types of data sources.

2.4.2 Graphs in Drug Discovery

Drug discovery is the discipline which focuses on how to develop new drug compounds. Discovering a new molecule is a long and expensive process [96], often involving researchers, companies, and national agencies [97]. Drug discovery is a field in constant development which is increasingly influenced by the use of ML techniques [25], and GNNs in particular [8], since drugs and other molecules are efficiently represented with graphs. These methods are increasingly needed to reduce the enormous monetary and time costs of developing new pharmaceutical compounds [96, 98]. In particular, deep learning models for drug discovery [99] focus on computer aided drug design, where in silico experiments allow for a faster search and delivery of new candidate drugs [100]. Moreover, DNNs can be used to predict the properties of new compounds in silico, to estimate their activity levels in different settings, to predict their side–effects, and to generate candidate molecular structures [25]. This thesis focuses on drug side–effect predictors, and on a predictor of protein–protein interfaces, all developed with GNNs, that will be presented in Chapter 5, and Chapter 6 respectively.

2.4.3 Bioinformatics and GNNs

GNN applications to bioinformatics are not limited to drug discovery. As discussed in Section 2.4.1, graphs are commonly used in biology and medicine, and GNNs can be used to address a wide range of problems in these fields. Some examples of open problems that GNNs could potentially contribute to solve in the future are listed below.

- Protein folding prediction: the term protein folding refers to the process by which proteins fold into their 3D structures, which determine their function. Predicting how proteins fold is an important problem in bioinformatics, as it can help to understand how proteins work and how they interact with other molecules. Traditionally, protein folding prediction has been approached using heuristics, which are methods that work well in practice but do not guarantee the optimal solution. More recently, DL methods, including GNNs, have been used to address this problem, as models can be trained on known protein structures to learn how proteins fold, and can then be used to predict the structures of novel proteins. Indeed AlphaFold is a recent AI system developed by DeepMind [101] that predicts a protein 3D structure from its amino acid sequence. It regularly achieves accuracy competitive with experiments. Nonetheless, the computational power needed to run AlphaFold is not available in common practice. Therefore, predicting protein folding is still a challenging problem due to the large number of possible conformations that a protein can take, which makes it difficult to find the optimal solution using traditional optimization techniques. However, AI methods, particularly GNNs, have the potential to provide a more efficient way to search the solution space and could lead to significant advancements in the discovery of new drugs and therapies.

- Protein–protein interaction prediction: predicting protein–protein interactions involves identifying which proteins bind to each other and determining the specific binding sites on the proteins [23]. This is an important problem in bioinformatics as protein–protein interactions are involved in many biological processes, including signaling pathways, gene regulation, and metabolism. There are several methods that can be used to predict protein–protein interactions, including clique detection or structural similarity analysis. GNNs can learn from known interactions to identify patterns indicative of protein–protein interactions and can be used to predict new interactions. Predicting protein–protein interactions can help to improve our understanding of biological processes and may also lead to the development of new drugs and therapies, for instance able to inhibit the formation of protein complexes which are fundamental for a virus transmission [102].

- AI–driven molecular dynamics simulations: molecular dynamics (MD) simulations are computer–based models that allow scientists to study the movements and interactions of molecules over time. These simulations are an important tool in bioinformatics and other scientific fields, as they can help to understand and predict the behavior of molecules in different environments and under different conditions. Traditionally, MD simulations have been computationally intensive, requiring large amounts of memory and time to run. This has limited the length of time during which the dynamics can be simulated, making it difficult to study processes that occur over long time scales. GNNs have the potential to significantly improve the efficiency of MD simulations, since they can be used to reduce the memory and time requirements. This could lead to the development of new drugs [103, 24, 26] and therapies and improve our understanding of how molecules function in different environments.

- Multi–omics analysis: multi–omics data refers to data from multiple types of omics studies, such as genomics, transcriptomics, proteomics, and metabolomics. These studies generate large amounts of data that can be difficult to analyze and integrate. GNNs can be trained to identify patterns and relationships in the data and to predict properties of an organism or a tissue based on multi–omics information. This can be useful for understanding the mechanisms underlying biological processes and for predicting the behavior of molecules under different conditions.

GNNs are versatile and can be considered an attractive option to modeling many different biological problems [29]. Moreover, different models have been employed on different tasks, such as different types of convolutional or recurrent GNNs. Ad–hoc models can also be developed on a task specific basis or on a broader biological setting [83].

Chapter 3

Machine Learning Applications to Molecular Data

This chapter provides a thorough literature review concerning the research applications constituting the main focus of this thesis. Relevant literature on ML–based applications to drug discovery are introduced in Section 3.1, with a particular focus on drug side–effect prediction (Section 3.2). Eventually, Section 3.3 deals with the protein–protein interface prediction.

3.1 ML in Drug Discovery

In recent years, the drug discovery field has seen a significant shift towards the use of ML techniques [99], due to a number of factors, including the increasing complexity and cost of traditional drug development technologies, the growing availability of computational resources, and the rapid advancements in AI and DL. As a result, DL methods are being increasingly employed to enhance and even replace traditional processes in drug discovery [8]. One of the main areas where DL is being used in drug discovery is in silico experimentation, such as the identification of potential drug compounds [104], prediction of drug–target interactions [105], virtual screening [106], the analysis of binding pockets exploiting 3D CNNs [107] or druggability predictors based on ANNs [108], as well as reverse docking techniques to identify target proteins using a library of known drugs [109]. The use of DL in these areas has the potential to greatly improve the speed and efficiency of the drug discovery process, as well as increase the likelihood of success in identifying effective drugs.

3.2 DSEs Prediction with DL

Drug side–effect prediction using ML is an important area of research in drug discovery. Predicting side effects of drugs in silico, before they are tested in humans, is a crucial task which can greatly improve the safety and efficacy of drugs, as well as reduce the risk of adverse events in patients. Many approaches has been proposed, from simpler methods based on Euclidean data [110, 111, 112, 113], to similarity scores between drugs [114]. The increasing in both quantity and heterogeneity of data, with the availability of processing resources, have helped in spreading ML–based approaches in this direction, from SVMs [115] and clustering techniques [116] to more complex predictors such as random forests [117] and deep MLPs [118] or NLP–based models [119]. Many methods that rely on biological information use protein-target as features, with the assumption that drugs with similar in vitro protein–binding profiles tend to exhibit similar side effects [120].

The prediction of DSEs requires knowledge of various biological entities, including genes, proteins, drugs, and pathways. The data used for such predictions is therefore naturally relational, thus being possible to represent it as a graph. GNNs have been found to be efficient in these types of scenarios; however, there is currently no known use of node–focused GNN–based approaches for predicting the side effects of individual drugs, although they have been employed in a similar but distinct context, specifically for the prediction of polypharmacy side effects — i.e. those resulting from the concurrent use of multiple drugs. Polypharmacy is nowadays a common practice in modern medicine which can lead to a large number of potential interactions, making it difficult to predict which combinations may lead to DSEs, given the complexity of the data. One approach to predicting polypharmacy DSEs using DL is to analyze a subset of possible drug pairs in the dataset, using graph attention networks (GATs) to measure the graph co–attention on the molecular graphs of the two drugs in each pair [121, 122, 123]. Another approach is to build a graph that accounts for the interaction between protein targets and drugs, and the known side effects of the single drugs. This graph is then analyzed using a GCN to predict the polypharmacy DSEs as links between drug nodes [7]. In this context, matrix factorization is a commonly employed technique for predicting links: in this cases networks are portrayed as matrices with cells representing relationships between them. In such scenarios, link prediction can be treated as a problem of matrix completion [124] based on Singular Value Decomposition (SVD) [125] or Non–negative Matrix Factorization (NMF) techniques [126, 127]. Given the graph of drugs and DSEs, composed of two inner networks — a DSEs–drug bipartite sub–graph and drug–drug similarity sub–graph —, the task is then to predict the links in the bipartite network between drugs and DSEs by means of heat diffusion–based or similarity–based approaches [123]. In recent years, hybrid approaches have been developed to jointly learn drug features from both the macroscopic biological network and the microscopic drug molecules [128]: to predict potential connections between drugs and DSEs, the model calculates molecular structure embeddings and fingerprints based on the drug's SMILES, while side effects are represented as unique random vectors. In the corresponding bipartite graph, the GNN model is exploited to update nodes representation, which is eventually multiplied by the side effect feature matrix to recreate the adjacency matrix, solving the link prediction task.

3.3 Prediction of Protein–Protein Interfaces

Predicting protein–protein interfaces is crucial in structural biology and bioinformatics, since by identifying the specific residues that form the interface of a protein complex, researchers can gain a deeper understanding of the molecular interactions that drive biological processes. This knowledge can have significant implications for the development of new drugs, as it can help identify potential drug targets and design drugs that specifically bind to those targets. Furthermore, predicting the interface of a protein complex can also aid in the prediction of the overall 3D structure of the complex, which is crucial for understanding the function of the proteins involved. In addition, understanding protein–protein interactions can lead to the design of new proteins with improved properties: this can be used in the development of new enzymes for industrial applications and in the design of new vaccines.

Though detecting the interface of two monomers is important to predict the quaternary structure and functionality of proteins [129], this can be a hard task to be faced with traditional techniques [130]. Protein–protein interfaces can be predicted with a variety of approaches: based on sequence homology [131], Bayesian methods [132], analyzing combined docking simulations [133], or using SVM predictors [134]. Recently, GNN–based predictors have been developed for the prediction of molecular interactions [135, 136, 137], although no specific GNN methods have been reported for the detection of interfaces between monomers. In this context, the biological problem of detecting the interfaces between the two proteins can be reformulated as a maximum clique search problem [6], by constructing a so–called correspondence graph from the graph representations of the two interacting monomers, in which secondary structures elements are considered [5]. The interface will then correspond to the maximum clique in the correspondence graph [6]. Clique detection problems have already been addressed with GNNs [138], and, more recently, also in a transductive learning setup [139]. Finally, this strategy was also further refined by exploiting LGNNs [73].

Chapter 4

GNNkeras: A Keras–based Library for Graph Neural Networks

In this chapter, GNNkeras, the software framework for the implementation of GNNs, described in publication [**A2**], is described. This framework allows to easily define GNN models for homogeneous and heterogeneous graph processing, represented by GNN/LGNN and CGNN/CLGNNN (see Section 2.3.4) models, respectively. All the implemented GNN models can be used for classification, regression, and clustering on nodes, edges or whole graphs, both in inductive and mixed inductive–transductive learning settings [139]. In the original framework, GNNs are inductively trained based on a supervised learning environment. However, GNNs and LGNNs can also take advantage of transductive learning [140], thanks to the natural way the information flows and spreads across the graph. In the transductive framework, the training set nodes and their targets are used in conjunction with the test patterns. In particular, the feature vectors of a subset of the training nodes — called transductive nodes — are enriched with their targets, to be explicitly exploited in the diffusion process, yielding a direct transductive contribution.

The rest of this chapter is organized as follows: Section 4.1 introduces the motivation behind the development of GNNkeras, Section 4.2 describes the software in detail and its usability, then Section 4.3 draws conclusions on this work.

4.1 Motivation and Significance

In the context of ML research on graphs, it is important for researchers and software developers to have adequate and flexible tools that support the development of applications with current GNN models and possibly favor the study of new versions of GNNs. For this reason, a new Keras library was developed, based on the original GNN model [1], which allows to implement the whole subclass of recurrent GNNs [80], and LGNNs [73].
GNNkeras users can easily access a huge number of ML features. This fact is guaranteed by Keras, which is built on top of TensorFlow 2.x [141] and is one of the most used and complete software libraries for ML. As far as the author's knowledge goes, GNNkeras is the first tool specifically designed for implementing recurrent GNNs, while other tools exist for building models of the subclass of convolutional GNNs. Finally, GNNkeras is flexible in that it permits to manage a variety of activities, graph domains and learning approaches.

The characteristics of GNNkeras are many and can be summarized in the following points.

- GNNkeras allows to develop and deploy GNN models easily, in a few lines of code, and with high versatility. Representing a GNN as a GNNkeras model gives a considerable advantage compared to previous common solutions, which were manually written from scratch with TensorFlow.
- All the three different types of deep learning problems on graphs are implemented: node–based, edge–based, graph–based.
- GNNs can be layered, implementing the LGNN version for more complex problems.
- GNNs and LGNNs can be applied to heterogeneous graphs.
- All the three super–categories of deep learning tasks can be tackled with GNNs: regression, classification, and generation.
- Inductive and mixed inductive–transductive learning can be adopted.

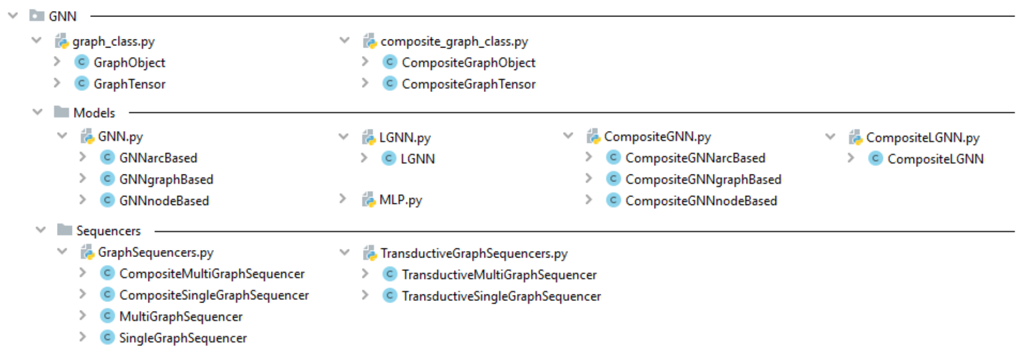
Although there are already several excellent GNN libraries available, based either on PyTorch (PyTorch Geometric [142], Deep Graph Library

DGL [143], etc.) or Tensorflow (Spektral [144], etc.), GNNkeras has been developed as a new library, rather than contributing to the existing ones. GNNKeras was born to address specific use cases or applications which are not fully supported by existing libraries, in particular to handle heterogeneous graph data, which is not possible with the other Tensorflow–based libraries. For this reason, GNNkeras can provide a tailored solution, with specialized APIs and workflows, that are designed to make it more efficient and effective for users to build GNN models on heterogeneous graphs.

The expected impact of GNNkeras is mainly related to its capability of helping its users in speeding up the proposal of new research and the development of advanced software.

4.2 Software Description

The software implements the GNN model formulation described in Section 2.3.1, the CGNN model described in Section 2.3.4, and LGNNs [73] as described in Section 2.3.3.



```
GNN
  graph_class.py
    GraphObject
    GraphTensor
  composite_graph_class.py
    CompositeGraphObject
    CompositeGraphTensor
  Models
    GNN.py
      GNNarcBased
      GNNgraphBased
      GNNnodeBased
    LGNN.py
      LGNN
    MLP.py
    CompositeGNN.py
      CompositeGNNarcBased
      CompositeGNNgraphBased
      CompositeGNNnodeBased
    CompositeLGNN.py
      CompositeLGNN
  Sequencers
    GraphSequencers.py
      CompositeMultiGraphSequencer
      CompositeSingleGraphSequencer
      MultiGraphSequencer
      SingleGraphSequencer
    TransductiveGraphSequencers.py
      TransductiveMultiGraphSequencer
      TransductiveSingleGraphSequencer
```

Figure 4.1: Software architecture: the main *GNN* directory contains graph data representation classes; the *Models* sub–directory provides MLP, GNN, LGNN, CGNN and CLGNN implementations; the *Sequencers* sub–directory provides graph sequencers for feeding models with GraphObject/CompositeGraphObject data. Note that the MLP model is a function which returns a Keras Sequential model, meaning that every Sequential model can be used for implementing the state transition network f_w and the output network g_w. In the latest version of the software, no distinction is made between multi–graph and single–graph sequencers, since only GraphSequencer/CompositeGraphSequencer and TransductiveGraphSequencer/CompositeTransductiveGraphSequencer are provided.

To parallelize software execution on modern CPUs and GPUs, all the operations are based on matrix multiplications. Fig. 4.2 shows the processing scheme of a heterogeneous graph by a CGNN model implemented with GNNkeras. The homogeneous case is analogous to a CGNN with a single node type.

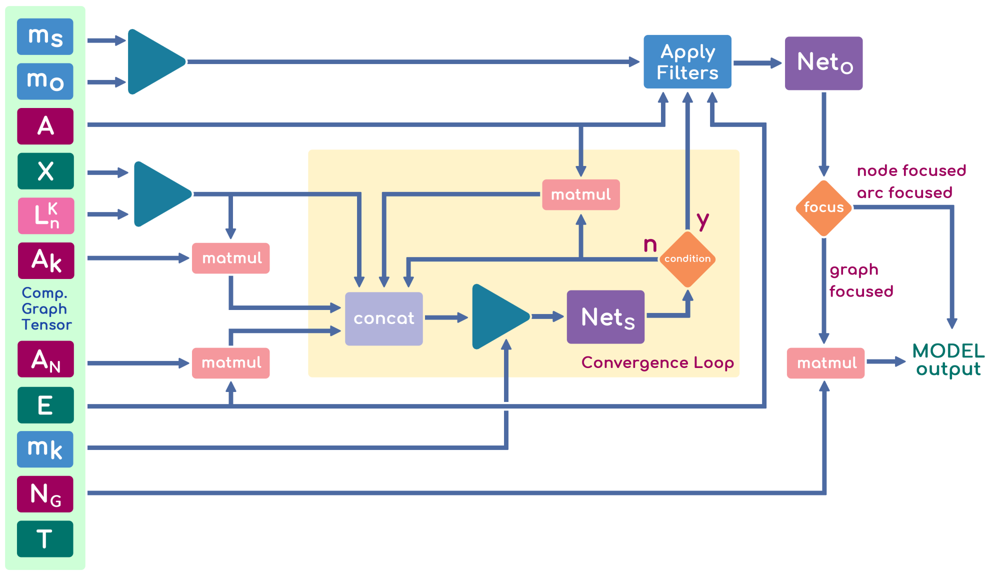


Figure 4.2: CGNN model software scheme. The GraphSequencer generates batches of GraphTensors which are presented to the model as input. All quantities pass through multiple operations (matrix multiplications, boolean mask filtering and concatenating processes) to form the input to f_w and g_w.

GNNkeras has been implemented as a module using the Python3 programming language, and it includes GNN models for node–focused, edge–focused, and graph–focused applications, working in homogeneous and heterogeneous graph domains, both in inductive and transductive learning contexts. It is based on NumPy, SciPy, and TensorFlow libraries, as NumPy and SciPy provide efficient numerical routines for dense and sparse data, while TensorFlow and Keras provide a simple and smart way to define and manage models, as well as to simplify the learning and evaluation processes. In particular, TensorFlow [141] is an open-source set of libraries for creating and working with neural networks, developed since 2017 by Google Brain, a deep learning and artificial intelligence research team from Google AI, the research division at Google formed in 2011 and dedicated to artificial intelligence. Since 2019 with its 2.0 version, TensorFlow has adopted Keras as its official high-level API, as it simplifies the implementation of complex neural networks with its easy to use framework. Developed by Google, Keras provides a python frontend with a high level of abstraction while including inbuilt and optimized modules for all neural network computations. Indeed, it provides numerous implementations of commonly used neural-network models or building blocks such as layers, objectives, activation functions, optimizers. Keras is scalable and it has native support for mixed–precision training on Nvidia GPUs and TPUs for speeding up the learning process. While there are many deep learning frameworks available today, Keras along with TensorFlow 2 has greater adoption in both the industry and the research community than any other deep learning solution.

4.2.1 GraphObject and GraphTensor

The software relies on a custom graph representation, which is implemented in `graph_class` and defined by the GraphObject class. An instance of GraphObject is therefore a compact representation of a generic graph $G = (V, E)$, which is initialized at least by a node feature matrix $\mathbf{X}$, an arc feature matrix $\mathbf{E}$, and by a target matrix $\mathbf{T}$.

Since not all $v_i \in V$ or $e_{ij} \in E$ are necessarily associated with a target value, a boolean output mask $\mathbf{m}_o \in \mathbb{B}$ is included in the GraphObject, to define whether or not a target value $t_i \in \mathbf{T}$ is associated with a specific node or arc. Moreover, when the dataset is composed of only one single graph, a boolean set mask $\mathbf{m}_s \in \mathbb{B}$ is included, so as to specify the subset of nodes or arcs belonging to a specific data set. Note that, for the graph to be correctly processed, the dimensions of $\mathbf{m}_s$ and $\mathbf{m}_o$ must match, while $\mathbf{m}_o$ must contain as many true values as the number of rows in $\mathbf{T}$.

During the initialization phase, a GraphObject defines automatically three SciPy sparse matrices in coordinate format: the adjacency matrix $\mathbf{A}$ and an arc–node matrix $\mathbf{A_N}$, which are used in the aggregation message procedure in the state transition phase — affected by a hyperparameter `aggregation_mode`, whose value defines the φ version to be used for aggregating the messages —, and the node–graph matrix $\mathbf{N_G}$, which is used by the GNN models in graph–based applications, and in particular, by the MLP implementing the output function, to convert a node–based output to an overall graph–based output.

The GraphObject class can be extended to the heterogeneous domain with the CompositeGraphObject class, provided in `composite_graph_class`, which includes a boolean type mask $\mathbf{m}_K \in \mathbb{B}^{n\times K} = \{m_{ik}^K \in \mathbb{B}\}$, to specify the type for each node, such that $m_{ik}^K = 1$ if and only if node $v_i = v_i^k$, otherwise $m_{ik}^K = 0$. In other words, each column of $\mathbf{m}_K$ is a boolean array which defines the type of node each node in the graph belongs to: there are no overlapping columns in $\mathbf{m}_K$, namely they are mutually exclusive, meaning that a node can belong to only one node type. A heterogeneous or *composite* graph is therefore described by its composite node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times L_n}$, where $L_n = \max(L_n^K)$ — as a zero padding is added to shorter feature vectors — and by its arc feature matrix $\mathbf{E} \in \mathbb{R}^{m \times l_e}$. The incoming neighborhood set can be composed of nodes of different type $\text{in}_v(v_i) = \{\text{in}_v^k(v_i) \subseteq \text{in}_v(v_i) : \text{in}_v^k(v_i) \cap \text{in}_v^h(v_i) = \emptyset,\ \forall k \neq h,\ k, h \in K\}$, where $\text{in}_v^k(v_i) = \{v_j^k \in V : e_{ji} \in E,\ k \in K\}$ has size $|\text{in}_v^k(v_i)|$. Moreover, a composite adjacency matrix set can be defined, exploited by the CGNN to process the incoming message of a node and based on the number and type of the neighbors, $\mathbf{A}_K = \{\mathbf{A}_k \in \mathbb{R}^{n \times n}\}$, where $\mathbf{A}_k = \{a_{ij}^k \in \mathbb{R} : k \in K, 0 \leq i \leq n-1, 0 \leq j \leq n-1\}$ is the composite adjacency matrix of type k. In particular, if there exists an arc $(v_i^k, v_j) \in E$, connecting two generic nodes $v_i^k, v_j \in V$, then $a_{ij}^k \in \mathbb{R} \neq 0$, otherwise $a_{ij}^k = 0$. Instances of GraphObject and CompositeGraphObject can also be saved in a single NumPy uncompressed/compressed npz file — or in a folder of text files — which includes all the necessary matrices for their complete representation. Given a dataset of graphs, in the form of a list of graph data elements, these classes also provide a smart way to save the entire dataset in a single folder, from which it can be loaded when needed. For speeding up the learning procedure, before feeding a GNN model, a GraphObject is converted in another custom graph representation, called GraphTensor, which contains a tensor–based description of all the attributes for the graph to be correctly and quickly processed by the GNN model: although GraphObjects and GraphTensors are essentially the same object, they differ in the data type used for their attributes, as GraphObjects are described by NumPy arrays and SciPy sparse matrices while GraphTensors by TensorFlow constant and sparse tensors. In order to be correctly processed by the GNN models, GraphObjects and CompositeGraphObjects are required to be fed to a special data handler, the Graph Sequencer, described in the following.

4.2.2 GraphSequencer

A GraphSequencer is a data manager for fitting to a sequence of data — such as a dataset of graphs — which is fed with a single GraphObject element or a list of multiple GraphObject elements to generate batches of GraphTensors, whose attributes are presented as input to a given GNNkeras model. In the first version of the software, described in [**A2**], a total of six GraphSequencer are provided, for multi–graph and single–graph–based datasets, in homogeneous and heterogeneous graph domains, and for inductive and a mixed inductive–transductive learning approaches. It is worth noting that the transductive one is a special class of Sequencer, which in the homogeneous case is fed with homogeneous GraphObjects while generating heterogeneous graph data. Indeed, for each epoch and batch, it splits the graph training supervised nodes into two subsets of inductive and transductive nodes: in the latter, the target of a node is integrated within its feature vector. Therefore, for that node, no supervision is considered in the learning process. Instead, no operation is carried out on the (supervised) inductive set, which is used in the learning process for adapting the model's parameters. As a consequence, the new graph cannot be represented by a homogeneous GraphTensor, since two types of nodes are present — described by feature vectors of different lengths — thus making necessary the CompositeGraphTensor representation for the CGNN learning process. In the composite case described by the CompositeTransductiveGraphSequencer, both the input and output data are represented by heterogeneous graphs, as the procedure of splitting into transductive and inductive sets can be performed on one specific type of nodes among all the available types in a CompositeGraphObject instance, from which batches of CompositeGraphTensors are generated, whose node types are the same as input heterogeneous graphs, integrating a new node type which defines the transductive subset for that specific node type.

4.3 Conclusions

In this chapter, a new general GNN programming framework has been presented, which provides multiple Keras–based GNN models for homogeneous and heterogeneous graph processing, for both inductive and mixed inductive–transductive learning settings. GNNkeras has been designed with the aim of simplifying the development of software applications in the field of ML for graphs, to simplify the approach to this kind of machine learning models for those who are already familiar with Keras models as well as for those who want to enter the graph domain of Artificial Intelligence.

Indeed, due to the mentioned characteristics in Section 4.1, GNNkeras is a flexible and suitable tool which can be used by researchers in ML to test new models and to design new applications for relational data. Finally, the exceptional interest in ML for graphs is a measure of the size and growth of the community operating in the sector and for which GNNkeras can be useful. In this perspective, a possible future development could be to include all its functionalities in other existing wide–used TensorFlow–based libraries. Contributing to an existing library, to add support for heterogeneous graph data processing, could be a valuable approach to ensure that the library is maintained and improved by the research community. This would avoid fragmenting the field and creating duplication of effort, and would allow users to benefit from a unified and well–maintained library. Adding support for heterogeneous graph data processing to an existing library would require significant effort, but could be accomplished through contributions from a community of developers and researchers. This could involve adding specialized APIs and workflows that are optimized for working with heterogeneous graph data, as well as incorporating relevant research advancements and best practices. By contributing to an existing library, researchers can benefit from the existing community and infrastructure, and can leverage the knowledge and experience of other developers and users, leading to a more robust and well–supported library that is able to keep up with the latest developments in the field of GNNs.

Chapter 5

Drug Side–Effect Prediction with Graph Neural Networks

This chapter describes three predictors of DSEs based on GNNs, trained on both homogeneous and heterogeneous relational datasets integrating multiple data sources. All the proposed models are the subjects of publications [**A1**], [**A4**] and [**A7**].

Drug side effects (DSEs), also known as adverse drug reactions (ADRs), are unwanted and undesirable effects that are possibly related to a drug, and they have become a major modern healthcare concern, particularly given the increasing complexity of therapeutics; thus, they have a high impact on the costs of the public health care system [145] and drug discovery processes [146, 97]. They can arise when a drug is used either as indicated, or as a consequence of improper dosages or multiple drug interactions, ranging from minor problems such as a runny nose to life–threatening events, such as a heart attack or liver damage: although fatal ADRs are rare once a drug is launched on the market, they are estimated to be one of the most common causes of death in the United States [147].

As prescription drug use is increasing [148], DSEs are becoming a central problem for pharmaceutical companies, since their occurrence is one of the most important reasons for candidate drug elimination during clinical trials, accounting for around 30% of failures [149, 97] and preventing candidate molecules from being selected as commercial drugs. The development of computational tools that can predict adverse effects might thus reduce the attrition rate, avoiding health risks for participants and cutting drug development costs [121].

DL methods are becoming increasingly important for understanding complex biological processes which trigger DSEs [118]. These processes involve various entities such as drugs, proteins, genes, and metabolic processes which interact with each other. An effective predictor should be able to handle heterogeneous data and consider the relationships between data types. In general, DSE computational prediction methods range from simple predictors to machine learning (ML) techniques such as support vector machines (SVMs) or multilayer perceptrons (MLPs), and to more complex models based on random forests or deep learning [115, 150, 7]. However, most of the DSE prediction techniques are still limited by the reliance on euclidean data [111, 114], which prevents models from considering relational information in the task, since molecular data must undergo preprocessing to be encoded into vectors, with an inevitable loss of possibly useful information. In addition, preprocessing methods usually require re–thinking when new features are added. GNNs, instead, can process relational data directly in graph form, exploiting all the structural information.

GNNs were already used for a related but different task, namely the prediction of polypharmacy side–effects. Polypharmacy side–effects are triggered by the combined use of two or more drugs. Direct and indirect interactions between the drugs are the key mechanisms behind these adverse reactions, which can be foreseen based on the structures of the drugs and their interactions with human genes. Predicting the probability of such events before prescribing the drugs can save the patient's health. This problem was addressed with GNNs both by analyzing the network of drugs and protein targets [7], and by applying a graph co–attention model over the two graphs describing a pair of drugs' structural formulas [121]. Differently, the side–effects of a single drug are mainly triggered by its interactions with the human organism, and can therefore be determined based on these interactions, on the structural features of the drug, and on similarities with other drugs for which the side–effects are known. Nevertheless, to the best of the author's knowledge, no single–drug side–effect predictor based on GNNs has been proposed yet.

This chapter presents three GNN–based methods for predicting the single DSEs, after a short description of the mixed inductive–transductive training procedure which has been used in some of the presented works and a brief overview of the databases from which the data has been retrieved, in Sections 5.1 and 5.2, respectively. In particular, Section 5.3 is the subject of publication [**A1**] and proposes a node–focused GNN classifier working on a heterogeneous graph involving drug–gene, drug–drug, and gene–gene relationships; Section 5.4 is the subject of publication [**A7**] and deals with a graph–focused classification task of the graph–based representations of drug structures; Section 5.5 further broadens the field of study by including both the graph–based representation of drug structures and the relationships they have with genes and other chemical compounds, as presented in publication [**A4**]. With reference to the previously mentioned order, all sections share the same structure: first, the related dataset is described (Sections 5.3.1, 5.4.1, and 5.5.1); then the specific model is presented, in Sections 5.3.2, 5.4.2, and 5.5.2, giving implementation details of the GNN model specific of the application as well as the setup employed in the experimentation (Sections 5.3.3, 5.4.3, and 5.5.3). Eventually, results are reported and discussed in Sections 5.3.4, 5.4.4, and 5.5.4 with a comparison with other ML methods in Sections 5.3.5, 5.4.5 and 5.5.5, and a focus on an ablation study in Sections 5.3.6, and 5.5.6. Please note that, since in [**A7**] only the graph–based structures of molecules are considered, with only one set of data retrieved from the databases described in Section 5.2, no ablation study is carried out in Section 5.4. The expected use of the method is then discussed in Sections 5.3.7, 5.4.6, and 5.5.7, respectively. Finally, Section 5.6 draws overall conclusions on the methods and the results.

5.1 Mixed Inductive–Transductive Learning

A mixed inductive–transductive learning scheme [140] is employed in the works presented in this chapter. In this setting, a double mechanism is exploited by the network to learn node–class associations. In standard inductive learning, predictions of side–effects of drugs are made based on node features of drugs and genes, and graph connectivity. In a transductive learning setup, instead, predictions are made based on known side–effects of other drugs. In this mixed inductive–transductive learning scheme, both mechanisms are utilized simultaneously. Specifically, the feature vectors of a subset of the training nodes, referred to as transductive nodes, are enhanced with their targets in order to be explicitly exploited into the diffusion process, yielding a direct transductive contribution.

In the learning process, the training set is split into ten batches. The network learns the input–supervision association on one training batch at a

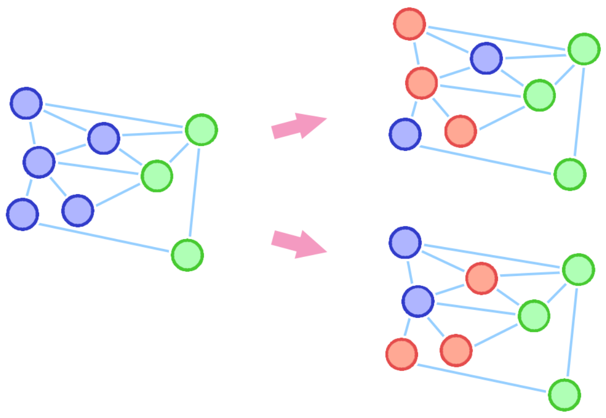


Figure 5.1: An example of two–batch generation for the mixed inductive–transductive learning setting. From the left: the original graph, with no transductive nodes; the generated batches, with both inductive and transductive nodes. Blue, red, and green nodes represent inductive drug nodes, transductive drug nodes, and gene nodes, respectively. Note that the transduction is performed only on drug nodes, since no prediction is performed on gene nodes.

time, while the other nine batches are exploited as a transduction set. The features of each drug node in the transduction set are augmented with the transductive features, corresponding to the occurrence of the side effects on that node. An example of two–batch generation for the mixed inductive–transductive learning setting is depicted in Fig. 5.1. When evaluating the validation set, the entire training set is used as the transduction set in the same manner as previously described. When evaluating the test set, the transduction set is composed of both the validation set and the training set. The idea is to exploit known DSE associations to predict DSEs for new drugs: since the mixed inductive–transductive scheme replicates this behavior during training, validation, and test times, this method is suitable for the use case of the following datasets and tools.

5.2 Data Sources

Having multiple databases which can exchange information on heterogeneous data is crucial for bioinformatics, as it allows for a more comprehensive understanding of biological systems. By combining data from different sources, researchers can gain a broader perspective on the interactions and relationships between different molecules, genes, and other biological entities, since the complexity of biological systems can make it difficult to understand their underlying mechanisms. In the following the main data sources used for retrieving data for publications [**A1**], [**A7**], and [**A4**] are described.

5.2.1 PubChem

PubChem [151] is an open open chemistry database which provides information on millions of chemical substances, including their structures, properties, and biological activities. In addition, PubChem also includes information on the physical properties of chemicals, such as melting point, boiling point, and solubility, as well as links to other resources, such as patents, literature references, and safety information. PubChem is a valuable resource for researchers, students, and the general public interested in chemistry and the biological effects of chemicals. The database is constantly being updated with new information, making it a valuable resource for staying up–to–date on the latest research in the field. Launched in 2004 as a component of the Molecular Libraries Program (MLP) of the US National Institutes of Health (NIH), the system is maintained by the National Center for Biotechnology Information (NCBI) and involves 885 contributing organizations, including university labs, government agencies, pharmaceutical companies and chemical vendors and publishers. As of January 2023, PubChem contains more than 298 million substance descriptions, 112 million unique chemical structures, and 301 million bioactivity data points from 1,5 million biological assays experiments, involving 103.988 Genes and 185.153 proteins.

5.2.2 Ensembl

Ensembl [152] is a comprehensive and freely available database of genomic and biological data for vertebrates and other eukaryotic organisms. Started as a project in 1999 to automatically annotate the genome, it is one of the most widely used resources for genetic and genomic information, and is particularly useful for understanding the structure and function of genes and their products (such as proteins). Ensembl is a collaborative project between the European Bioinformatics Institute (EBI) — part of the European Molecular Biology Laboratory (EMBL) — and the Wellcome Trust Sanger Institute, and it is maintained and updated by a team of bioinformatics analysts and software developers. The database includes a range of data types, including DNA and protein sequences, gene structures, regulatory elements, and functional annotations. It also provides tools for data visualization and analysis, including BLAST searches, multiple alignment tools, and gene expression data. Ensembl is an important resource for researchers in many fields, including genetics, genomics, and molecular biology. It is regularly updated with new data and features, and is an essential tool for anyone working with genomic data.

One of the most useful features of Ensembl is to access and query large datasets of biological data in a user–friendly way with BioMart web–based tool. BioMart is designed to be easy to use, even for people with little or no programming experience, and it provides a wide range of features for searching, filtering, and downloading data, including DNA and protein sequences, gene structures, functional annotations, and gene expression data. Users can also use BioMart to search for specific genes or genomic regions of interest, and to filter data based on various criteria such as species, tissue type, or gene function.

5.2.3 Gene Ontology

The Gene Ontology (GO) [153] is a controlled vocabulary used to describe the biological roles of genes and their products, typically proteins, in an organism. It is a database that provides a standardized system for annotating and querying the function of genes across different organisms. GO is made up of three main components: the Biological Process (BP), Molecular Function (MF) and Cellular Component (CC). These three ontologies provide a consistent way to describe the different aspects of gene product behavior and location: the BP ontology defines high–level biological processes in which a gene or gene product participates, such as metabolism or cell division; the MF ontology describes the molecular activity of a gene product, such as catalytic activity or binding; the CC ontology describes the cellular location or complex in which a gene product is active, such as the cytoplasm or a cell membrane. GO is a widely used resource in the bioinformatics community, with annotation for many sequenced organisms including humans, mice, fruit flies, and yeasts.

5.2.4 STITCH

The search tool for interactions of chemicals (STITCH) [154] is one of the most complete and up–to–date database of known and predicted interactions between over 2 million chemical compounds and over 5 million proteins. The interactions include physical and functional associations, derived from scientific literature, patents, computational prediction, knowledge transfer between organisms, and interactions aggregated from other primary databases. STITCH integrates information about interactions from metabolic pathways, crystal structures, binding experiments and drug–target relationships, which are also know as drug–protein interactions (DPIs). This database also provides information on the strength and type of each interaction, as well as any known physiological effect of the interaction. This can be helpful for researchers trying to understand the potential mechanisms of action of a chemical compound or the potential side effects of a drug.

5.2.5 HuRI

The Human Reference Protein Interactome Mapping Project (HuRI) [155] is a repository, created and maintained by the Center for Cancer Systems Biology (CCSB) at Dana–Farber Cancer Institute, which stores the data from the ongoing work on the first complete reference map of the human protein–protein interactome network. All pairwise combinations of human protein–coding genes are systematically being interrogated to identify which are involved in binary protein–protein interactions (PPIs). HuRI has grown in several distinct stages primarily defined by the number of human protein–coding genes amenable to screening for which at least one Gateway–cloned Open Reading Frame (ORF) was available at the time of the project. The database currently includes three proteome–scale human PPI datasets, as well as smaller datasets generated from screens of specific protein–coding genes. In addition to these datasets, HuRI also includes information about PPIs involving different isoforms of the same gene produced through alternative splicing. HuRI is an important fully–open resource for researchers studying the human proteome, as it provides a wealth of information about how different proteins work together to perform essential functions and contribute to disease processes. By making this information available, HuRI enables researchers to identify potential therapeutic targets and to develop new treatments for diseases. Additionally, the database is constantly updated as new information becomes available, so it is a valuable resource for staying current with developments in the field. Overall, the HuRI database is a key tool for advancing our understanding of the human proteome and for improving human health.

5.2.6 SIDER

The Side Effect Resource database (SIDER) [156] contains information on the side effects of drugs available in the European Union and the European Economic Area (EEA), maintained by the European Medicines Agency (EMA). The database includes information on over 500 drugs and more than 4.000 side effects derived from a variety of sources, including reports submitted to national regulatory authorities, published literature, and other public sources. Moreover, SIDER includes information on the indications (uses) for each drug, as well as information on the drug's pharmacology (how it works in the body) and its contraindications (situations in which the drug should not be used). As a result, SIDER is intended to be a resource for pharmacovigilance, which is the study of the safety of drugs after they have been approved for marketing, and to help healthcare professionals, researchers, and others to identify and assess the potential risks of drugs, and to make informed decisions about their use. It is regularly updated with new information on drugs and their side effects, as it becomes available. In addition, the EMA also provides a number of tools and resources to help users of SIDER in accessing and interpreting the data, including guidance documents and training materials, as well as technical support for users who have questions or need assistance.

5.3 Modular Multi–Source Prediction

In this section DruGNN, a new method for single DSE prediction, based on GNNs and a novel graph dataset accounting for drug–gene, drug–drug, and gene–gene relationships, is presented. The first novelty of this work is the creation of a dataset for predicting DSEs using the publicly available data described in Section 5.2. This dataset, described in detail in Section 5.3.1, is composed of a single graph with two types of nodes (drugs and genes) and three types of edges (connections between drugs, genes, and both), where each node is described by a specific set of features, such as chemical properties for drugs and characteristics/functions for genes. Another key aspect of this work is the development of DruGNN, a GNN–based ML method for DSE prediction on the aforementioned graph dataset. Although the idea of a GNN for heterogeneous data is not novel, to the author's knowledge there are no GNN–based approaches, especially recurrent GNN models, for the prediction of side effects of single drugs exploiting a heterogeneous dataset composed of drugs and genes. Application specific implementation details of the GNN model are given in Section 5.3.2. The prediction process is set up as a multi–class multi–label node–focused classification problem in which drug nodes are associated to a binary vector defining DSE classes. A mixed inductive–transductive learning approach [139] is used, incorporating information on known DSEs in order to predict new DSEs.

The method is flexible and can be easily expanded to include more node attributes and edges without altering the ML framework [140], as well as to predict DSEs of new drugs with no need of retraining. Experimental results have shown promising accuracy and the method has been compared to similar graph–based models and a traditional ML method which cannot exploit relational information. Two ablation studies have also been conducted over both the DSE set and the feature set to demonstrate the model's robustness and the importance of each data source in the training process.

The rest of this section is organized as follows: Section 5.3.1 describes the dataset, its construction process, and the data sources. Implementation details of the GNN model specific of this application are given in Section 5.3.2, while the experimental setup is described in Section 5.3.3. In Section 5.3.4 the obtained results are presented, with a discussion on their relevance and meaning. A comparison with other models is carried out in Section 5.3.5, while Section 5.3.6 focuses on an ablation study. Eventually, Section 5.3.7 discusses the expected use of the method.

5.3.1 Dataset

The dataset consists of a single heterogeneous graph where genes and drugs are represented as nodes, connected by three different sets of edges, with relationships based on gene–gene interactions, drug–gene interactions, and drug–drug similarity. A sketch of the graph is provided in Fig. 5.2.

Gene node features are retrieved from the BioMart database [157] and consist of chromosome and strand location encoded by a one–hot vector of 25

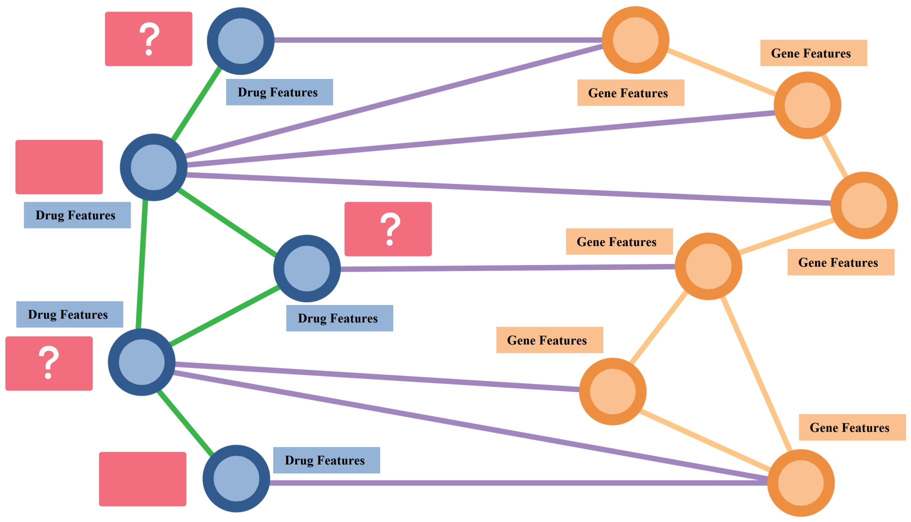


Figure 5.2: Illustration of the graph composition. Blue and orange nodes represent drugs and genes, respectively, labelled with their own features. Red rectangles represent DSEs classes.

features (22 regular chromosomes, plus X, Y, and mitochondrial DNA) and by a single value $+1$ or -1, respectively, the percentage of GC content, and a one–hot vector representing the molecular function ontology term, clustered on the 113 unique top–level terms from Gene Ontology's molecular function ontology [153] using DAVID [158, 159], for a total of 140 features on each gene node.

Drug nodes are described by a set of chemico–physical molecular descriptors provided by the PubChem database [151], such as molecular weight, polar surface area, xlogp coefficient, heavy atom and rotatable bond counts, numbers of hydrogen bond donors and acceptors, concatenated to the MF of the drug. Only drug nodes with at least one occurring DSE are taken into consideration.

Two MFs of different lengths are extracted from the SMILES strings provided by PubChem — describing each molecular structure of the drugs — using the RdKit python library [160], of sizes 128 and $2,048$ elements, used for different purposes. The shortest one is concatenated to the drug features in order to keep the feature vector size of drug nodes similar to that of gene nodes and therefore bringing the total size of the drug feature vector to 135. The longest MF allows to better estimate and to define the drug-–drug relationships, based on the drug similarity measured in terms of Tanimoto Similarity (TS) [161], the most common measure of similarity between molecules, so that an edge between a generic node pair is generated if and only if the TS value of the two related MFs is greater than or equal to a fixed TS threshold at graph construction.

Drug–gene relationships are based on drug–to–protein interactions (DPI) extracted from the STITCH database [154] and BioMart [157], allowing to produce a mapping between drugs and genes whose products are the proteins the drugs interact with. For the aforementioned dataset, this mapping consists of 314, 369 DPI links. Finally, gene–gene relationships are retrieved from HuRI [155] and from a gene–to–protein mapping provided by BioMart. Gene–gene interaction links combined to gene features such as the GO term allow to reconstruct the metabolic network.

The supervisions for the drug nodes are extracted from the SIDER database [156]. At the time of this study, SIDER contained 5, 868 side–effects occurring on 1, 430 drugs, with a total of 139, 756 entries, each accounting for the association of a single drug to a specific side–effect. These associations are retrieved from medical documents, and in most cases do not report the concentration at which the side effects appear. Drugs supervisions are defined as all the most occurring DSEs found on SIDER with a minimum of 100 occurrences over the drugs of the dataset, whose original distribution is reported in Fig. 5.3.

After all the filtering steps, these result in 360 side–effects , which may (positive class) or may not occur (negative class) for each drug node. The choice for the DSE filtering threshold represents a good compromise between number of considered DSEs and the balancement of the dataset, since lowering this threshold resulted in a much more unbalanced supervision. The associations between drugs and side–effect are modeled independently of each other, both on the drug and on the side–effect axes. The final task is then a multi–label, multi–class, node–focused classification, with 96, 477 total positive occurrences, and 515, 160 negative ones. In particular, belonging to the positive class for a generic drug means that it produces the particular side effect, corresponding to the value 1 in the related position of the target vector. Note that each drug can cause multiple side effects. These labels are also used, according to the inductive–transductive scheme, as either transductive features for known drugs, or class supervisions for new drugs.

The final dataset consists of a single heterogeneous graph composed of 1, 341 drug nodes and 7, 881 gene nodes as well as 12, 002 PPIs links, 314, 369

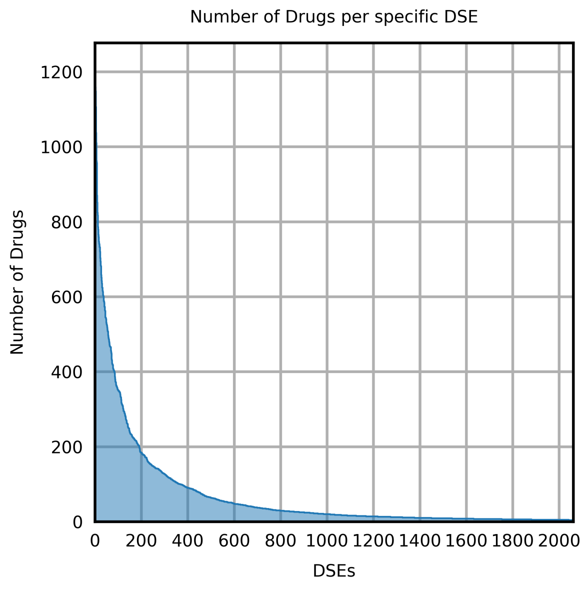


Figure 5.3: The number of drugs associated to each specific DSE. Most DSEs are associated with few drugs.

DPIs links, and $5,252$ drug–drug similarities, for a total of $9,222$ featured nodes and $331,623$ unfeatured edges. In generating the drug–drug arcs, a TS threshold equal to 0.7 has been selected, as to avoid either an excessive number of drug–drug links or isolated drug nodes, which in the mixed inductive–transductive learning policy would have meant too much DSE transductive information passing between nodes, or no information passing at all, respectively. The dataset construction, with all the source databases and preprocessing steps, is sketched in Fig. 5.4.

5.3.2 Model

This subsection describes the application–specific implementation of the GNN model formulated in Section 2.3.1. In particular, since in this section the dataset is a heterogeneous graph with two types of nodes (drugs and genes), the GNN model is a Composite GNN, as formulated in Section 2.3.4. The state updating functions are therefore implemented by two MLPs, one for each node type. The general formulation given in Eq. (2.10) is specified for this application in Eq. (5.1), where N_d and N_g represent the subsets of drug and gene nodes, respectively. Edge features are not used in

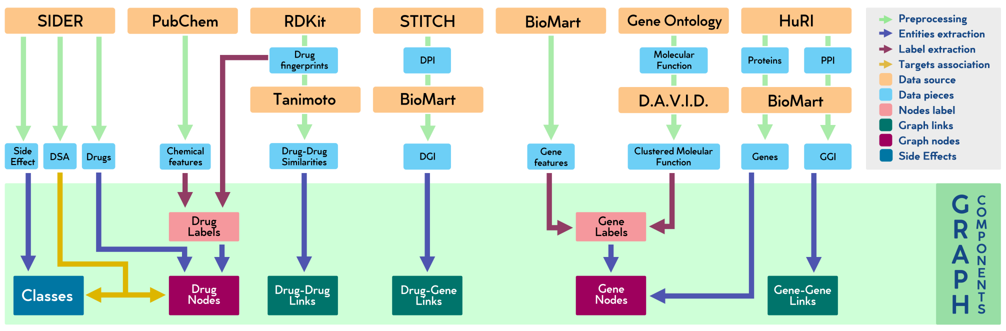


Figure 5.4: Sketch of the DruGNN original dataset construction. The information flows from the orange and cyan rectangles representing data sources and data pieces to the final graph representation, in which purple and pink rectangles represent graph nodes and their features, green rectangles stand for the edges between nodes, and blue rectangles represent the labels of a subset of the drug nodes.

this formulation, as edges are not labeled in the final dataset:

$$\begin{aligned} x_n^t &= f_{w,d}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ l_n)) \ \text{ if } \ n \in N_d \subset N \\ x_n^t &= f_{w,g}(x_n^{t-1},\ l_n,\ a \sum_{m \in Ne(n)} (x_m^{t-1},\ l_n)) \ \text{ if } \ n \in N_g \subset N \end{aligned} \tag{5.1}$$

During the training phase, the two MLPs will learn two different versions of the state updating function: f_w^d computes the states of all drug nodes $N_d \subset N$, while f_w^g the states of all gene nodes $N_g \subset N$. The output network is then applied to the subset of drug nodes in the graph for which an output is requested. Since this is a problem of node classification the output function is exactly the one described in Eq. (2.7).

5.3.3 Experimental setup

The final task is to predict DSEs on drug nodes in a node–based classification problem with multiple classes and a multi–label setting, as each drug can cause multiple side–effects among the 360 considered DSEs. A random dataset split, consistently used throughout the experimentation, is performed to ensure reproducible and comparable results. 10% of the drug nodes are designated as the test set, and only employed during testing phase. Another 10% of the drug nodes are reserved as a validation set to monitor overfitting and stop the training process accordingly. The remaining 80% of the

Hyperparameter	**Values**
Activation Function	relu, selu, tanh, sigmoid
Initial Learning Rate	10^{-2}, 10^{-3}, 10^{-4}
State Dimension	10, 50, 100, 200
Hidden Units	100, 200, 500
Neighborhood Aggregation	average, sum
Dropout Rate in net_s	0.0, 0.1, 0.3, 0.5

Table 5.1: Hyperparameter values analyzed during the grid search procedure, with the best configuration in terms of performance highlighted in red.

nodes are exploited as the training set. The network hyperparameters were tuned with an extensive grid–search procedure. In particular, experiments were carried out on all the hyperparameter values described in Table 5.1 and their combinations. Each element in the grid was analyzed by measuring the average model accuracy in a 5–fold cross–validation procedure.

To assess the performance of the DruGNN on the dataset and investigate the impact of varying the number of side–effects on the learning process, a series of experiments were conducted. An ablation study was conducted on the set of side–effects, in which the model was trained and tested on versions of the dataset featuring progressively fewer side–effects. Only the k most prevalent side–effects were retained in each iteration, with k assuming the following values: $360, 240, 120, 80, 40, 20, 10, 5$.

To assess the significance of the contributions from the various data sources, an additional ablation study was carried out: node features and edges were organized in groups by source and were removed from the dataset, one group at a time, evaluating the model's performance in the absence of such a group. The difference in performance provides an estimate of the significance of the excluded features. Seven feature/edge groups characterize the dataset, and each group was analyzed in a 5–fold cross–validation procedure. The same dataset split and transductive learning scheme described in the previous experimentation (as outlined in Section 5.1) were consistently applied.

Eventually, the DruGNN was compared to other competitive GNN models with different characteristics, in order to assess its performance with respect to the alternative solutions. In particular, two powerful models were considered: GCNs [10], which exploit convolutions to aggregate information coming from different locations across the graph, and have shown competitive performance on many different tasks; GraphSAGE [13], which are versatile networks that can be configured with various aggregation and state updating functions, being potentially competitive on every graph dataset. In addition, a comparison with a simple MLP was also performed, in order to evaluate the difference between a graph–based model and a Euclidean predictor: in particular, a three–layered MLP was used after a small optimization over the validation set. Due to the unique and graph–structured nature of the dataset, it was not possible to incorporate previously published predictors for DSEs in the comparison as the feature sets would not be adaptable. It should also be noted that no graph–based predictor has been published for this task thus far.

5.3.4 Results and Discussion

The hyperparameter search described in Section 5.3.3 produced a model with an accuracy over the validation set of 87.22%. The same model, evaluated on the held–out test set scored an accuracy of 86.30%.

5.3.5 Comparison with Other Models

To evaluate the capabilities of DruGNN with respect to other GNN variants and non–graph based Euclidean models, a comparison was made with GraphSAGE [13], GCNs [10], and a simple MLP model trained on a vectorized version of the drug data. The MLP serves as a measure of the results that can be achieved using traditional Euclidean predictions on the dataset. The GCN and GraphSAGE were trained using the same inductive–transductive scheme as DruGNN. All models were trained using the binary cross–entropy loss function, Adam optimizer [72], and an initial learning rate of 10^{-4}. A maximum of 500 epochs were allowed for each model, with early stopping implemented based on the validation loss and recovery of the best weights. As expected, all graph–based models outperformed the standard MLP, highlighting the benefits of representing the dataset as relationships on a graph and learning directly on the graph structure. Additionally, using a GraphSAGE or GCN approach did not yield the same results as obtained with DruGNN, as shown in Table 5.2. This can be attributed to the Re-

Model	**Configuration**	**Avg. Acc.**
MLP	$DL = 3 \times 25$	0.7798 ± 0.014
GCN	$CL = 2 \times 36,\ DL = 116$	0.8294 ± 0.004
GraphSAGE	$CL = 2 \times 72,\ DL = 1 \times 168$	0.8311 ± 0.004
DruGNN	$K = 6, SD = 50,\ DL = 1 \times 200$	0.8630 ± 0.006

Table 5.2: Comparison between different models of the GNN family. Model configuration is reported; all of the models were optimized on a limited hyperparameter search space. The parameters used in the DruGNN model include K (maximum number of state update iterations), SD (state dimension), DL (number of dense layers and the number of units in each layer), and CL (number of convolutional layers and the number of units in each layer). For GCN and GraphSAGE models, the dense layer is the last one before the output layer.

current GNN model being particularly efficient in node property prediction tasks, while the other GNN models tend to aggregate nodes on a larger scale, providing an advantage in graph property prediction tasks. This is also consistent with theoretical studies on GNNs that demonstrate processing capabilities by simulating the Weisfeiler–Lehman test [75] (for more details please refer to Section 2.3.5). Models evaluation is based on the average accuracy percentage obtained over a 10–fold cross–validation procedure on the same dataset split.

5.3.6 Ablation Study

The contribution of side–effects to the network's learning capability was investigated using the best model configuration obtained from the first set of experiments. The side–effects were ranked by occurrence and the set was gradually reduced by selecting only the most common ones. The average accuracy was measured over five repetitions on the held–out test set and the results are presented in Table 5.3.

The multi–class multi–label classification task can be seen as a set of independently and parallel problems involving all the class memberships. It was initially expected that an increase in the number of classes would result in the need for the network to learn a more complex algorithm. However, the results in Table 5.3 show an improvement in performance for larger sets of side–effects. This unexpected behavior can be attributed to the network's

DSE	360	240	120	80	40	20	10	5
Acc.	0.863	0.815	0.732	0.685	0.630	0.618	0.671	0.747
Bal.	0.581	0.586	0.600	0.605	0.621	0.599	0.577	0.562

Table 5.3: Average accuracy (Acc.), and average balanced accuracy (Bal.) obtained on the test set by training and testing the model on progressively smaller sets of DSEs.

ability to learn intermediate solutions useful for all or large subsets of the classes, similar to transfer learning. This is particularly effective in this case, as transfer learning between classes is crucial due to the relatively small dimension of the set of drugs, with the additional bonus of avoiding overfitting. However, at lower set dimensions (up to 20), transfer learning becomes less efficient and the network instead learns to treat each class independently.

The unbalanced nature of the problem also contributes to the observed behavior. The side–effects with fewer occurrences are highly unbalanced towards the negative class, while the side–effects with more occurrences are skewed towards the positive class. As less common side–effects are removed, the balance shifts, likely playing a significant role in this phenomenon.

A second ablation study was conducted on the feature/edge groups from various data sources. The model's accuracy, trained and tested without the specific data group, was evaluated and averaged over five runs of the same experiment. To better weigh the importance of each group, the DPF (Difference Per Feature) score was also calculated by dividing the performance difference with respect to the complete model by the number of features in the group. The results of the ablation study for each data group, including the observed performance loss, are presented in Table 5.4.

Table 5.4 demonstrates that each data source has a positive impact on the GNN learning process. The deletion of drug fingerprints results in the largest decrease in performance: this can be attributed to the significance of drug substructures in identifying side effects, as well as the large number of features assigned to this data group (128 in total). Additionally, analysis of the DPF score indicates that the seven PubChem descriptors have the greatest contribution, as expected due to their chemical relevance. The gene features also have a notable impact on performance, with the BioMart–retrieved features having a DPF equivalent to that of drug fingerprints. Edges also

Group	DPI	PPI	DDS	FP	PC	GO	BM
Type	E	E	E	DF	DF	GF	GF
Acc.	0.8614	0.8618	0.8623	0.8581	0.8620	0.8617	0.8620
Diff.	0.16	0.12	0.07	0.49	0.10	0.13	0.10
Count	-	-	-	128	7	113	27
DPF	-	-	-	0.004	0.014	0.001	0.004

Table 5.4: Average accuracy (Acc) and relative difference from the model trained on the full set of features (Diff.) when the model is trained and tested on the dataset with the absence of a specific feature/edge group (Group). The Group column is divided into three types: E (Edges), DF (Drug Features), GF (Gene Features). For DF and GF data groups, the number of features (Count) and the difference per feature (DPF) are also reported. DPI (Drug–Protein Interactions), PPI (Protein–Protein Interactions), DDS (Drug–Drug Similarity), FP (FingerPrints), PC (PubChem), GO (Gene Ontology), and BM (BioMart) are the different data groups used in the study.

showed to be important, as the deletion of each edge set leads to a decrease in performance. Notably, the results suggest that drug similarity relations are less important, which may be due to the network's ability to infer similarity based on fingerprints and drug–gene interactions.

The results of the study indicate that although each group of features and edges contributes positively to the model's performance, the minimal decrease in performance observed when they are removed suggests the robustness of the model: in other words, the model maintains a high level of performance even when certain sets of edges or features are excluded. One potential explanation for this robustness is the general tendency for GNNs to be robust, as previously demonstrated by systematic ablation studies on various types of graph datasets [75]. Another possibility is that the large number of features and edges, as well as the diversity of data sources used in the model, enhance its robustness.

5.3.7 Usability

DruGNN is designed as a practical tool that can aid healthcare and pharmacology professionals in predicting the side effects of newly discovered compounds or those not yet classified as commercial drugs. The dataset and software are publicly accessible on GitHub[1], allowing for further scientific research and utilization by the community. Furthermore, both the dataset and algorithm are scalable, with the ability to add new compounds for prediction without compromising network usability, as there is no need to re–train it from scratch.

An example of such usage is represented by the prediction of the side–effects of Amoxicillin (PubChem CID: 2171), which is part of the held–out test set (and therefore never seen during the training or validation phases). Amoxicillin has been determined to be similar to the following drugs, listed by PubChem CID: 2173, 2349, 2559, 4607, 4730, 4834, 8982, 15232, 22502, 6437075. It also interacts with 76 genes. No other information but the fingerprint and PubChem features of Amoxicillin are available to the model. The network correctly predicts 22 side–effects, among which (listed by SIDER id) quite common and expectable ones, like C0000737 (Abdominal pain) and C0038362 (Stomatitis), but also non–obvious ones, like C0002871 (Anaemia) and C0917801 (Insomnia). However, it fails to predict 6 side–effects: C0002994 (Angioedema), C0008370 (Cholestasis), C0009319 (Colitis), C0011606 (Dermatitis exfoliative), C0014457 (Eosinophilia), C0036572 (Convulsion). Please notice that Angioedema, Colitis, Dermatitis exfoliative, and Convulsion are indicated as very rare for Amoxicillin. Cholestasis has relatively few occurrences in the dataset, and is therefore difficult to predict. Moreover, the network shows good predictive capabilities on side–effects which are common in the whole drug class Amoxicillin belongs to (represented by the similar compounds in the dataset). In addition, the network predicts only one side–effect which is not associated to Amoxicillin in the supervision: C0035078 (Renal failure).

As shown in the example, to predict the side–effects of a new compound, it is sufficient to retrieve information (coming from wet–lab studies and from the literature) on its interactions with genes, and to know its structural formula. The fingerprint of the compounds composing the dataset as well as similarities between them can be calculated by using RdKit python library. The PubChem features can either be obtained from a database, or calculated with RdKit. By incorporating the compound into the dataset, its DSEs can be predicted with DruGNN. The results obtained from DruGNN and the relevant information from the database can be utilized by medical professionals such as doctors and pharmacists for visual analysis.

[1]`https://github.com/PietroMSB/DrugSideEffects`

5.4 DSE prediction on Molecular Graphs

In this section, GNN–MGSEP (Graph Neural Network — Molecular Graph Side–Effect Predictor) is described, for the DSE prediction task, addressed as a graph–focused multi–class multi–label classification problem. As the drug structure can be efficiently encoded by a molecular graph, GNNs are employed to predict DSEs on single drugs based solely on the drug structure. This methodology differs from the GNN–based DSE predictor described in Section 5.3, since DruGNN predicts DSEs on a large knowledge graph integrating drug features, gene features, gene–gene interactions, drug–gene interactions and drug–drug similarities [103]. DruGNN, despite its ability to integrate information from various domains through Composite Graph Neural Networks (CGNNs), is limited in its ability to exploit full molecular information as it encodes the drug structure as a fingerprint vector. Moreover, using SMILES, like other non–graph–based methodologies, also results in loss of information. In contrast, molecular graphs retain all structural information associated with each drug compound, which can be exploited to predict DSEs. A novel dataset of molecular graphs is introduced and described in Section 5.4.1: it can be used for the prediction of DSEs with any predictor model that accepts the drug structure in input; molecular graphs can also be enriched with relevant chemical features of the compound. Although the model is not a novelty in this field, to the author's knowledge there are no GNN–based approaches for the prediction of side effects of single drugs exploiting only their molecular structure, in a graph–focused, homogeneous setting. The rest of the section is organized as follows: Section 5.4.2 explains the methodology and describes the GNN model used for carrying out the predictions; Section 5.4.3 describes the experimental setup; Section 5.4.4 discusses the relevance of the results, with a focus on the prediction performance in comparison with other available methods in Section 5.4.5; finally, Section 5.4.6 describes the usability of the method. Since only the graph–based structures of molecules are considered, and no data is collected from most of the databases described in Section 5.2, no ablation study is carried out in this work.

5.4.1 Dataset

In this framework, only the chemical structure of a drug and known DSE associations are required. Data was obtained from the SIDER database [156], which contained information on $1,430$ drugs, $5,880$ Adverse Drug Reactions (ADRs), and $140,064$ drug–ADR pairs, at the time of the study. The "stereo" version of the STITCH compound identifier was used as a key on the PubChem database [151]. A preprocessing procedure was implemented on the SIDER database to filter out duplicates of drug–ADR pairs. This was done by removing all associations referring to lowest level terms (LLTs) for DSEs. The dataset is therefore composed of drug–ADR pairs in which side–effects are expressed via a primary term (PT). This ensures that duplicate pairs are not present and that the learning procedure is not negatively affected by unnecessary data. Additionally, a constraint was applied on DSE occurrences, dropping DSEs with less than 5 occurrences in the dataset. A series of procedures were implemented which resulted in a significant reduction of the number of associations and side–effects in the dataset. Specifically, the number of associations was reduced from nearly $309,000$ to $157,000$, and the quantity of DSEs with a number of occurrences greater or equal to 5 was reduced from $4,251$ to $2,055$. In addition, compounds which lacked intramolecular bonds were removed from the dataset. This resulted in a final dataset of $157,000$ associations. Furthermore, an additional filter was applied to the dataset for Experiment B1, resulting in the removal of drugs that were associated with either less than 5 or more than 400 side–effects. The distribution of number of side–effects associated for each drug is shown in Fig. 5.5.

A grouping system was developed for similar chemical elements in order to be viewed as the same type by the GNN model based on their chemico–physical properties (Table 5.5 for details), due to the non–uniform distribution of chemical elements within the dataset. This is crucial as some elements have limited occurrences and too many node types would increase the complexity of the learning problem.

Molecular chemical structures were represented by graphs to minimize loss of information that would occur when compressing the molecular graph into other data structures. To retrieve the graph representation of a specific molecule, intermediate steps were taken such as using PubChemPy[2] to re-

[2]PubChemPy documentation is available at `http://pubchempy.readthedocs.io/`

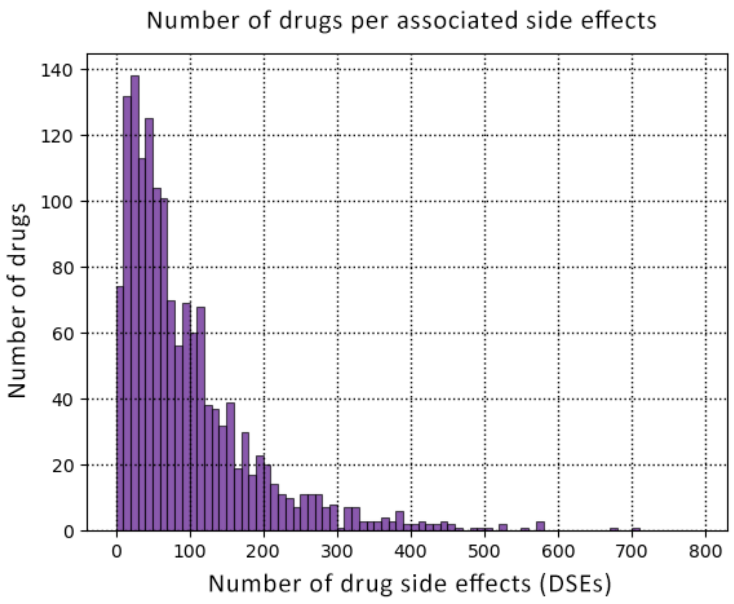


Figure 5.5: The number of side–effects per drug follows a non uniform skewed distribution. Most drugs have few DSEs, while few drugs are associated to a large number of DSEs. This causes imbalancement in the class distributions which can lead to a bias in the model.

Element group	**Element**	**Element group**	**Element**
1	C	9	Br
2	N	10	Na, K, Li
3	O	11	Ca, Mg, Ba, Sr
4	S, Se	12	Co, Tc, Mn, Fe
5	F	13	Au, Ag, Pt, Zn
6	P	14	B, Ge, In, Tl
7	Cl	15	La, Gd
8	I		

Table 5.5: Grouping of the elements used in this work.

trieve the SMILES string associated with the compound and transforming it into a RWMol using the RdKit library [160]. This RWMol was then used to build a NetworkX [162] graph of the molecular structure, which was finally converted into a GraphObject for use as a structured graph representation in GNNKeras [29]. The graph representation used in this work is made of three different components:

- the node matrix, where rows represent the chemical element (or chemical element grouping) of the specific node. The following general rule applies to the node matrix:

$$n_{ij} = \begin{cases} 1 & \text{if the } i\text{–th node belongs to the } j\text{–th element group} \\ 0 & \text{otherwise} \end{cases}$$

- the edge list, made of arrays of length 6 that collect initial and final node of the edge, along with a label indicating the chemical bond it represents. More specifically, an edge label is defined as $\{n_h,\ n_k,\ b\}$, in which n_h and n_k are respectively the initial and final nodes ids, while $b = \{b_1, b_2, b_3, b_4\}$ is a one–hot array expressing the bond type which can be single, double, triple or aromatic;

- the target vector, employed to carry out supervised learning; it consists in a binary vector of 2055 entries (i.e., one entry per side–effect) such that

$$t_i = \begin{cases} 1 & \text{if the drug can cause the } i\text{–th side–effect} \\ 0 & \text{otherwise} \end{cases}$$

5.4.2 Model

This subsection describes the application–specific implementation of the GNN model formulated in Section 2.3.1: in particular, since in this section the dataset is composed of a collection of homogeneous graphs representing molecules, the GNN model is a normal GNN, as formulated in Section 2.3.1. The state updating functions are therefore implemented by a single MLP, as only one type of nodes is present, representing the atoms the molecules are composed of. The general formulation given in Eq. (2.6) and still holds for this application, as features are associated to each edge, representing the bond type between pairs of atoms. Since this is a graph classification task, the output function is exactly the one described in Eq. (2.9).

5.4.3 Experimental Setup

In this work, DSE prediction is modelled as a graph–focused classification problem. Therefore, it requires an adequate loss function to be optimized during learning. The best fitting choice in this scenario is the binary cross–entropy, being capable of handling each DSE independently from the others by taking into consideration the entries of the network's output one by one. The loss function was optimized via the Adam optimizer (from Adaptive moment estimation), which has proved to be highly efficient, requires little memory, and is appropriate for problems with noisy and/or sparse gradients [72].

The following settings differ from each other on various parameters, such as number of epochs, batch size and stopping criteria. These differences are presented and compared in Table 5.6.

Parameter	**Exp. A**	**Exp. B**	**Exp. B1**	**Exp. C**
Batch size	32	32	32	16
Threshold loss	0.15	0.15	0.15	0.14
Epochs	8000	10000	10000	7500
Patience	2000	2000	2000	1000

Table 5.6: Hyperparameters of the experiments discussed in this work. Exp. B and B1 were carried out using the same parameters but different datasets.

Various metrics were employed to evaluate different aspects of the model's performance. One of them is the binary accuracy, which simply calculates how often predictions match binary labels as a percentage. When using such metric, entries of target arrays are considered independent from each other, by performing an element–wise comparison between the predicted array and the desired output. However, this metric does not provide enough information about the model's performance due to the nature of the target distribution, as each target array contains $2,055$ entries, and each chemical compound in the filtered dataset causes approximately 97 side–effects on average[3], which accounts for only 4.71% of the total number of target array entries. Therefore, high values in binary accuracy do not necessarily indicate good network performance as they could be achieved even if the network's predictions were vectors full of zeros. Hence, metrics such as the Area Under ROC Curve (AUC) and the Area Under Precision Recall Curve (AUPR) were also employed, which are obtained by plotting the true positive rate against the false positive rate and the positive predicted value against the true positive rate, respectively.

[3]More precisely, the mean results in $96,744$ DSEs per compound.

Metric	Exp. A	Exp. B	Exp. B1	Exp. C
Bin Acc	0.9516 ± 0.004	0.9513 ± 0.004	0.9525 ± 0.006	0.9494 ± 0.03
AUC	0.8613 ± 0.004	0.8611 ± 0.010	0.8673 ± 0.005	0.8586 ± 0.003
AUPR	0.2913 ± 0.022	0.2885 ± 0.018	0.2854 ± 0.035	0.2682 ± 0.017

Table 5.7: Results for each experiment.

Metric	Exp. A	Exp. B	Exp. B1
PPV	0.4756 ± 0.26	0.4541 ± 0.24	0.4512 ± 0.24
NPV	0.9589 ± 0.04	0.9600 ± 0.04	0.9614 ± 0.04
TPR	0.9905 ± 0.01	0.9891 ± 0.02	0.9886 ± 0.02
TNR	0.2132 ± 0.15	0.2382 ± 0.15	0.2068 ± 0.14

Table 5.8: Further analysis for the best experimental settings. PPV: Positive Predicted Value; NPV: Negative Predicted Value; TPR: True Positive Rate, or Sensitivity; TNR: True Negative Rate, or Specificity.

5.4.4 Results and Discussion

The results presented in the following are obtained from a 5–fold cross–validation procedure, for an unbiased evaluation of GNN–MGSEP.

Table 5.7 shows the results obtained in each experiment: Exp. B1, which was carried out after a further filtering of the data, provided the best performance in terms of binary accuracy and AUC, while Exp. A resulted in a better AUPR.

The best three experiments underwent a further analysis: Table 5.8 reports positive predicted value, negative predicted value, sensitivity and specificity of experiments A, B and B1. It is worth noting how different the positive predicted value and the specificity are, compared to the negative predicted value and the sensitivity: therefore, it is possible to hypothesize that currently the GNN model, in the framework of side–effect prediction, is better at detecting negative associations with respect to positive ones due to the unbalanced distribution of drug side–effect associations.

An analysis focused on the side–effects revealed that the relative frequency of each adverse reaction highly influences the ability of the model in detecting cases of positive associations regarding such side–effects. Table 5.9 shows such results, by considering the 10 most frequent and less frequent side–effects (DSEs) in the datasets and reporting the ratio between true positive predictions and the number of occurrences of such adverse reactions.

Metric	**Exp. A**	**Exp. B**	**Exp. B1**
Most Freq. DSEs	0.6931 ± 0.11	0.8559 ± 0.15	0.8381 ± 0.16
Less Freq. DSEs	0.0150 ± 0.03	0.0297 ± 0.05	0.0568 ± 0.07
Overall Average	0.1118 ± 0.19	0.1206 ± 0.22	0.1090 ± 0.19

Table 5.9: Influence of the relative frequency of each adverse reaction on the model ability in detecting positive associations.

The difference in the "detectability" of side–effects based on their number of occurrences in the dataset is clearly shown.

The experimental results show that the DSE prediction task can be effectively accomplished using only the drug structure (molecular graph) as GNN–MGSEP does. A similar task had been previously accomplished using GNN (DruGNN) with a large knowledge graph containing seven main information resources such as drug structural fingerprints, chemical properties, molecular function ontology, genomic information, gene–gene interactions, drug–drug similarity and drug–gene interactions [103]. The setup presented in this work is simpler, requiring only drug structures for prediction, yet the molecular graph conveys structural information which is crucial in determining drug functionality, resulting in a simpler yet efficient prediction framework as indicated by the metrics.

5.4.5 Comparison with Other Models

For performance comparison, other predictors not based on GNNs are also considered, such as Pauwels [112], which is a structure–based predictor that uses Sparse Canonical Correlation Analysis of structural fingerprints; DrugClust [116], which is a predictor based on clustering and Gene Expression data, and DeepSide [118], a more complex predictor based on DL which integrates data from various sources. These methods, along with DruGNN, provide a valuable set of models for evaluating the capabilities of GNN–MGSEP in comparison to previous work. While a direct comparison is not possible as all of these methods use different data types and were trained and tested on datasets of different nature, a comparison can be made by assessing the differences in data types, dataset sizes, and predicted DSEs. The number of drugs, predicted side–effects, information used, and methods used for each predictor are described in Table 5.10.

Predictor	**Drugs**	**DSEs**	**Method**	**Data Types**
Pauwels	888	1,385	SCCA	SF
DrugClust	1,080	2,260	Clustering	CF DPI GEX
DeepSide	791	1042	MLP	GEX GO SF
DruGNN	1,341	360	CGNN	CF SF GO PPI DPI DDS
GNN–MGSEP	1,397	2,055	GNN	MG

Table 5.10: Comparison of data and methodology for each predictor. Drugs and DSEs refer to the number of drugs and side–effects, respectively. SF: Structural Fingerprints; MG: Molecular Graphs; GO: Gene Ontology data; CF: Chemical Features; DPI: Drug–Protein Interactions, DDS: Drug–Drug Similarity; PPI: Protein–Protein Interactions.

As demonstrated by models such as DeepSide and DruGNN, integrating increasing amounts of heterogeneous information has been the key to improving DSE predictors. The former was one of the first DL approaches to the problem, while the latter used GNNs to analyze the knowledge graph of DSEs. GNNs can also be used to analyze the structure of each molecule, as molecules are naturally represented by graphs. Moreover, molecular graphs convey full structural information of the molecule more efficiently than structural fingerprints, which are commonly used in this field. This leads to comparable performance between DeepSide, DruGNN, and GNN–MGSEP, which uses a much simpler load of information consisting only of the molecular graph of each compound, making it a user–friendly predictor in comparison to the other two. The performance of each model is reported in Table 5.11.

5.4.6 Usability

GNN–MGSEP is user–friendly as it requires minimal information to accurately predict the occurrence of side–effects. To estimate the probable DSEs of a new drug, it is only necessary to submit its molecular graph to the model. Retraining the model is not necessary every time a new drug is introduced in the dataset, however, when a large number of new drugs with their labels become available, retraining will improve the model's performance for future predictions. The model is lightweight and does not require significant resources for training, as it can be done on a commercial laptop without a GPU. Once the model is trained, obtaining predictions with GNN–MGSEP is even more efficient as the whole dataset of molecular graphs does not need

Model	Binary Accuracy	AUC	AUPR
Pauwels	-	0.8932	-
DrugClust	-	0.9138	0.3336
DeepSide	-	0.8090	-
DruGNN	0.8630	0.7715	-
GNN–MGSEP	0.9525	0.8673	0.2913

Table 5.11: Comparison of prediction performance with respect to the other described methodologies. Please notice that each predictor was trained and tested on its own dataset, with different data types, number of drugs, and number of DSEs compared to the others. Three metrics are considered for the evaluation: Binary Accuracy, AUC (Area Under ROC Curve), and AUPR (Area Under Precision–Recall curve). Only the metrics proposed by the respective authors are reported for each predictor, as different problem formulations do not always allow to use the same metrics.

to be loaded. The model can be used as a simple screening service to predict the occurrence of side–effects on a large number of molecular graphs, at the early stages of a drug discovery pipeline.

5.5 DSE Prediction with DL Molecular Embedding

In this section, the MolecularGNN method for DSE prediction is described [163], which tries to overcome the limits of the works described in Sections 5.3 and 5.4. Usually, a molecular structure is encoded in a computer–readable format such as the SMILE string or International Chemical Identifier (InChI) keys, which are strings of symbols of variable length, used by computer software to facilitate the search for molecular information in databases and for creating two or three–dimensional models. Although these formats represent molecules in their entirety, fixed–sized encoding strategies are widespread in drug discovery and in drug design applications, as they are better suited for computational methods. The most popular are the molecular fingerprints (MFs), mainly used for assessing similarities between molecules in virtual screening (VS) processes and chemical analysis. MFs represent molecules with fixed–size vectors of real numbers, encoding chemical properties of the molecules and their structural and sub–structural features. Fingerprints of this kind are the results of mostly hand–crafted algorithms based on sub–

graph structure detection [160]. However, in recent years, many machine learning–based approaches offered an alternative strategy: neural fingerprints (NF), which are obtained by training neural networks on a specific task [164, 165]; therefore, the choice of machine learning architecture for the fingerprint production becomes crucial in order to obtain a fingerprint with the least loss of information.

Following the approach proposed in Section 5.3, the data of interest consist of a heterogeneous graph including two types of nodes (drugs and genes) and three types of edges (drug—gene, drug-–drug, and gene-–gene relationships). Formally, the task is a node–focused, multi–class, multi–label classification problem: namely, the GNN model is trained to predict the DSEs associated with the drug nodes, by processing directly graph–structured data and by generating NFs at learning time. In this context, the structure of a generic drug can be represented as a single graph, consisting of nodes representing atoms and linked by chemical bonds, making any GNN–based model a proper choice for this task. Indeed, since GNNs can handle graph–structured molecules directly and learn how to encode them during the training process, they are ideal to overcome the limits in DSE prediction and in the calculation of fingerprints, by adapting the molecule representation to the task they are being trained for — thus capturing structural and relational information relevant for the specific task, which is something traditional fingerprints and euclidean–based ML models cannot do.

In the presented approach, by exploiting both the relational dataset described in Section 5.3.1 and the structural representations of drugs as described in Section 5.4.1, a richer learning domain is obtained, which is composed of a graph of graphs: each graph corresponds to its molecular graph at the lower level and to a node of the knowledge graph at the higher level. In the latter, drugs are linked to each other by similarity and are linked to gene nodes according to their interactions. A new GNN classifier is exploited for a single–DSE prediction called MolecularGNN that processes such heterogeneous and composite graph–structured data.

Notice that the use of a graph–of–graphs domain is an interesting research topic, as GNNs have been applied in very few cases to such a peculiar domain in the literature [166, 167, 168, 169]. Moreover, in such a context, the GNN model is supposed to automatically extract a neural fingerprint (NF) from each molecule.

Just like the previous sections, the rest of the section is organized as follows: Section 5.5.1 describes the dataset on which the model defined in Section 5.5.2 is trained, while Section 5.5.3 defines the experimental setup; Section 5.5.4 presents and details the relevant experimental results, with a comparison with other available methods in Section 5.5.5 and an ablation study discussed in Section 5.5.6. Eventually, Section 5.5.7 discusses the usability of the method.

5.5.1 Dataset

The dataset for the DSE prediction task was the one constructed in [103] and described in Section 5.3.1. It consists of a single heterogeneous graph where genes and drugs are represented as nodes, connected by three different sets of edges, with relationships based on gene–gene interactions, drug–gene interactions, and drug–drug similarity. At a high–resolution level, drug nodes are represented as a single homogeneous graph each, where nodes represent atoms and edges represent chemical bonds.

In particular, the molecules' SMILES strings are used to obtain graph–based representations of the drug molecules, from which a GNN–based NF is extracted. Even if the graph representation of a drug as a molecule may seem natural, it can still be extremely complex: a simple way to represent them is to set each atom as a node (except for hydrogen atoms) and each bond between two atoms as an edge in the graph, both associated with feature vectors. Since some atoms are quite rare in the dataset — for instance, arsenic is present in only one molecule — they are grouped in accordance with their chemical families, which are encoded by a 1–hot vector: in total, 18 classes of atoms have been identified, as shown in Fig. 5.6.

Moreover, atom feature vectors are enriched by additional chemical information, such as the degree of the atom (number of electrons involved in chemical bonds), the number of hydrogen atoms linked to it, the number of radical electrons around the atom (which constitute a rare feature, as they introduce instability), and their formal charges. Edge features describe the type (simple, double, triple, or aromatic) of bond the edge represents, encoded by a 1–hot vector: this is the only feature used on edges, as other important quantities should be deductible from the atoms on both sides of the edge, and from their neighborhood.

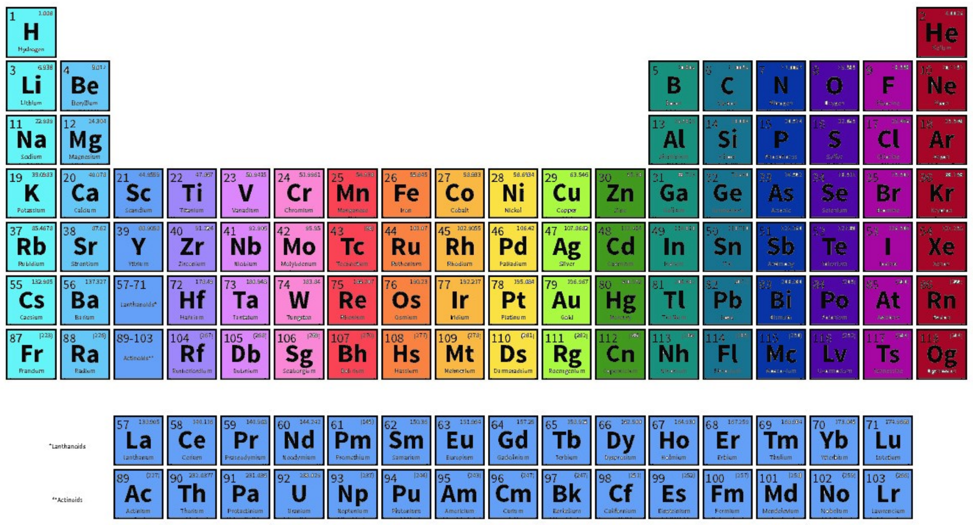

H He
Li Be B C N O F Ne
Na Mg Al Si P S Cl Ar
K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr
Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I Xe
Cs Ba 57-71 Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
Fr Ra 89-103 Rf Db Sg Bh Hs Mt Ds Rg Cn Nh Fl Mc Lv Ts Og
*Lanthanoids La Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Lu
**Actinoids Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr

Figure 5.6: Grouping of atoms according to their chemical family. Each color represents a single atom class.

5.5.2 Model

This subsection describes the application–specific implementation of the GNN model formulated in Section 2.3.1. Both the implementation described in Section 5.3.2 and Section 5.4.2 are exploited in this application, as described in the following. To improve the performance by exploiting information from both the molecular structures and the graph, a new GNN–based model, MolecularGNN, is proposed. This model is an improvement of DruGNN [103] and GNN–MGSEP [170], as it is capable of extracting information from both the genes–drugs relational graph as well as directly from the graph–based molecular structure representation, by processing it with a GNN sub–model in order to produce a NF which considers both node features and relational data, thereby improving the overall predictions.

An overview of the architecture and processing scheme is given in Fig. 5.7.

The MolecularGNN model is composed of two GNN sub–models:

- A molecular embedding module: A GNN–based architecture fed with the graph representations of the molecular structures—extracted from the SMILES strings describing the drug molecules—to produce a task–based NF, which is concatenated to the drug node features and used by the following sub–model to predict DSEs on drug nodes; this approach

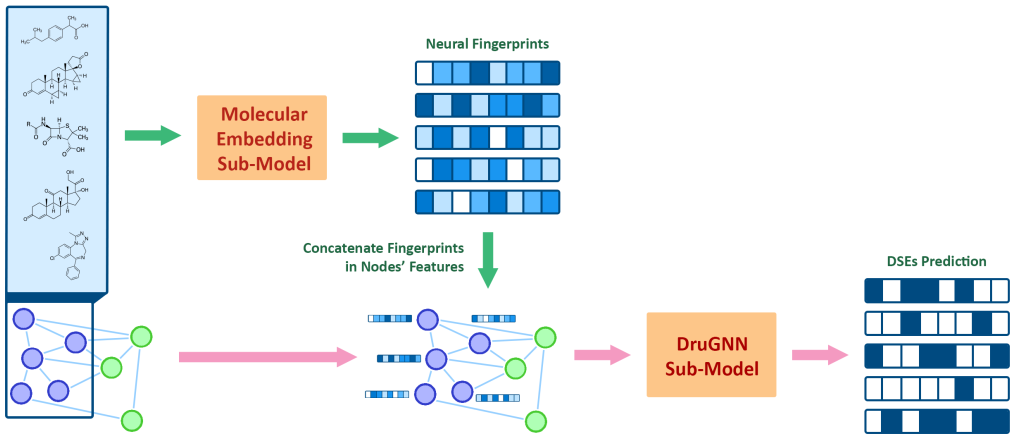

Molecular Embedding Sub-Model
Neural Fingerprints
Concatenate Fingerprints in Nodes' Features
DruGNN Sub-Model
DSEs Prediction

Figure 5.7: MolecularGNN architecture and processing scheme. Molecular graph structures — representing a single drug node each — are processed by the Molecular Embedding sub–model to produce NFs which are then concatenated to the drug node feature vectors. The enriched graph–structured dataset is then processed by the DruGNN module to produce predictions on drug node DSEs. A new drug molecule can be added to the dataset by calculating its possible relationships with other nodes in the graph and extrapolating its graph representation from its SMILES code, and the related DSEs can be predicted by just applying the pre–trained model to the new dataset, composed of the one on which the model has been trained, and enriched by the new information derived from the new drug molecule.

should encourage the molecular embedding sub–model to extract information, which is not captured by the MF used in the original work, as it remains the same overall training procedure. In this case, since the NF is produced by a ML model, it is dynamically adapted during the learning procedure for the DSE prediction task to maximize the performance of the general model. Implementation of this sub–module is described in Section 5.4.2.

- DruGNN: The model is similar to the one introduced in [103]. This network is a recurrent GNN model [1] which exploits an inductive–transductive learning scheme to predict DSEs on drug nodes. More details about its implementation and its hyperparameters can be found in Section 5.3.3. The main difference with the original model is the fingerprint used as part of the drug node features, since in this work it is not a pre–calculated or standard MF, but a NF produced by the molecular embedding sub–model. Implementation of this sub–module is described in Section 5.3.2.

To ensure that the information extracted from the molecule complements the information contained in the DruGNN graph, the two aforementioned modules are jointly trained as a single model.

5.5.3 Experimental Setup

A grid–search procedure has been carried out based on the hyperband algorithm [171], together with an early stopping callback, monitoring the F1–score on a validation set. The hyperparameter search space is summarized in Table 5.12.

Hyperparameter	**Values**
M number of layers in net state	$[1, 2, 3, 4]$
M number of layers in net output	$[1, 2, 3]$
M batch normalization layer before net state	$[True, False]$
M dropout in net state	$[0.0, 0.1]$
M L2 regularization	$[0.0, 0.001, 0.01, 0.1]$
M activation function	$[relu, tanh, selu]$
M state dimension	$[50, 70, 100, 120, 150]$
M max iteration	$[2, 3, 4, 5, 6, 7]$
D number of layers in net state	$[1, 2, 3, 4]$
D number of layers in net output	$[1, 2, 3]$
D batch normalization layer before net state	$[True, False]$
D dropout in net state	$[0.0, 0.1]$
D L2 regularization	$[0.0, 0.001, 0.01, 0.1]$
D activation function	$[relu, tanh, selu]$
D state dimension	$[50, 70, 100, 120]$
D max iteration	$[2, 3, 4, 5, 6, 7]$

Table 5.12: Hyperparameter search space for MolecularGNN. M = molecular embedding sub–model; D = DruGNN sub–model. In red the best configuration obtained, corresponding to the final architecture of the MolecularGNN model.

Moreover, three strategies were adopted during the learning procedure to deal with the data distribution imbalance: using binary focal cross–entropy instead of simple binary cross–entropy, not considering the DSEs with less than 100 positive occurrences, and using a weighting scheme to encourage better prediction of rarer side effects.

Focal cross–entropy was first introduced in [172] for object detection applications. The only difference from the standard cross–entropy loss is the introduction of a factor $(1-p)^\gamma$ for positive example and p^γ for negative examples, as defined in Eq. (5.2):

$$Loss(p,y) = \begin{cases} -(1-p)^\gamma \log(p), & \text{if } y = 1 \\ -(p)^\gamma \log(1-p), & \text{otherwise} \end{cases} \tag{5.2}$$

From a practical point of view, it results in a loss which penalizes big errors and gives little to no importance to small ones in predictions. Despite its early development purpose, it can be effectively applied on any classification task where the model has to learn from unbalanced datasets, as the additional factor encourages the model to take some risks by predicting the minority class more often. Effects of γ on accuracy and F1–score metrics in the grid search procedure are shown in Fig. 5.8.

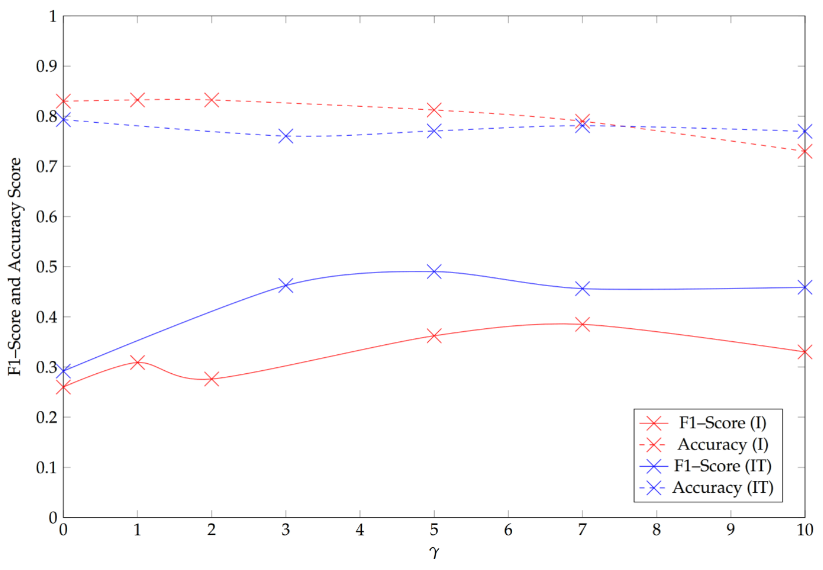


Figure 5.8: Effects of γ on the accuracy and F1–score metrics. I = only inductive learning approach; IT = mixed inductive–transductive learning approach.

In the dataset, some DSEs occur in only 100 cases, and others in more than 1000, suggesting that the difficulty of prediction for a given DSE is somehow related to the number of positive examples. To tackle this disparity, the model should give more relevance to rarer DSEs when predicting class probabilities: the ideal behavior should be for the network to have close to

Metric	**DruGNN**	**MolecularGNN**
Accuracy	0.8630 ± 0.006	0.7746 ± 0.007
AUC	0.7715 ± 0.012	0.6846 ± 0.014
F1–score	0.2680 ± 0.010	0.4716 ± 0.008
Precision	0.5934 ± 0.014	0.4425 ± 0.012
Recall	0.1859 ± 0.03	0.5104 ± 0.021
PatR	0.2002 ± 0.010	0.2591 ± 0.004

Table 5.13: Comparison of MolecularGNN and DruGNN models.

uniform predicting power over the whole drug set, as in perfectly balanced datasets. To achieve this, a weighting scheme inversely proportional to the number of positive examples has been used, as defined in Eq. (5.3):

$$w = \left(\frac{100}{n_{positive}}\right)^{\mu} \tag{5.3}$$

where w is the weight given to a DSE with the $n_{positive}$ positive example, and μ is a geometric coefficient which determines the weight of this weighting process (no weighting for $\mu = 0$ and inversely proportional weighting for $\mu = 1$).

All the experiments have been carried out on a Linux–operated machine, equipped with an Intel® Core™ i9–10920X CPU @ 3.50GHz (12 cores/24 threads), 128 GB DDR4 memory, and two NVIDIA Titan RTX with 24 GB VRAM GDDR6 each. A single graphics card was able to carry out an instance of the learning procedure and the related evaluation of the model, leading to an average of about 5 h and 30 min per experiment.

5.5.4 Results and Discussion

MolecularGNN has been compared with the DruGNN model using accuracy, area under the ROC curve (AUC), micro F1–score (F1–score), micro–precision (precision), micro–recall (recall), and precision at recall of 90% (PatR), in a 10–fold cross–validation procedure. The results are presented in Table 5.13.

MolecularGNN performed significantly better in terms of F1–score compared to the DruGNN model: indeed, the main improvement of MolecularGNN over DruGNN was in the recall, with an improvement of 0.32, which

Model	**Drugs**	**DSEs**	**Method**	**Data Types**
Pauwels	888	1385	SCCA	SF
DrugClust	1080	2260	Clustering	CF DPI GEX
DeepSide	791	1042	MLP	GEX GO SF
DruGNN	1341	360	CGNN	SF CF GO PPI DPI DDS
GNN–MGSEP	1,397	2,055	GNN	MG
MolecularGNN	1384	360	CGNN NF	MG CF GO PPI DPI DDS

Table 5.14: Comparison of data and methodology for each predictor. NF stands for neural fingerprints, SF for structural fingerprints, MG for molecular graphs, GO for Gene Ontology data, CF for chemical features, DPI for drug–protein interactions, DDS for drug–drug similarity, PPI for protein–protein interactions.

compensated a drop in precision of 0.15, resulting in an almost doubled F1–score. The PatR metric is particularly interesting, as it well simulates a real world scenario: in this scope, the ideal model (i.e., the one with the maximum score) should correctly classify 90% of the side effects, to make clinical tests to filter out the false positives. This metric shows a notable increase in 0.6. The threshold of 90% is still a somewhat low value for such a use case; however, it constitutes a robust metric for predictors. Some metrics, such as the accuracy, do not improve, resulting in a drop in performance of about 10 percentage points: that is in line with the expectations, as the dataset is characterized by an imbalanced class distribution — very common in most real–life classification problems or datasets — thus making the F1–score a better evaluation metric, since it gives a better measure of the incorrectly classified cases than the accuracy metric.

5.5.5 Comparison with Other Models

Additionally, a comparison with previously developed methods for drug side effect prediction is provided in Table 5.14 and Table 5.15. More precisely, MolecularGNN was compared to the same models described and used for comparison in Section 5.4.5.

It is worth noticing that each method was developed on different datasets and with different and often heterogeneous data types. Moreover, these differences led to different dataset sizes, as not all the data available online can be exploited in all the settings. Please keep in mind that the different nature of the predictors, the data used by each, and the number of examples mean that the comparison is purely qualitative. As a matter of fact, it is common

Model	**Binary Accuracy**	**AUC**	**AUPR**
Pauwels	-	0.8932	-
DrugClust	-	0.9138	0.3336
DeepSide	-	0.8090	-
DruGNN	0.8630	0.7715	-
GNN–MGSEP	0.9525	0.8673	0.2913
MolecularGNN	0.7746	0.6846	-

Table 5.15: Comparison of prediction performance with respect to the other described methodologies. Please notice that each predictor was trained and tested on its own dataset, with different data types, number of drugs, and number of DSEs compared to the others. Three metrics are considered for the evaluation: Binary Accuracy, AUC (Area Under ROC Curve), and AUPR (Area Under Precision–Recall curve). Only the metrics proposed by the respective authors are reported for each predictor, as different problem formulations do not always allow to use the same metrics.

practice to test the models for DSE prediction only on the dataset for which they are conceived, as each model usually uses a unique combination of the many different pieces of information needed to model these complex biological phenomena. The comparison was made in terms of AUC metric, since this is the only metric that was measured for each of the predictors involved.

The presented method, with an unbalanced dataset, did not reach an AUC comparable to simpler predictors operating on less unbalanced datasets. The F1–score was the dominant metric in this case, and it was significantly better than the F1–score of DruGNN — the only other predictor trained on a dataset with a comparable level of imbalancement between classes.

5.5.6 Ablation Study

An ablation study was carried out to assess the importance of the contributions of the different changes in the learning procedure and in the dataset. The results from a 10–fold cross–validation procedure are reported in Table 5.16. As expected, the mixed inductive–transductive learning approach resulted in the biggest improvements over the basic model. The most notable results are the increases of 0.167 in the AUC, 0.138 in the F1–score, and 0.046 in the PatR. Those huge improvements show the importance of exploiting all the available information, including the actual labels of the neighboring drugs. The molecular embedding developed in Section 5.5.2

Configuration	**Accuracy**	**AUC**	**F1-Score**
Baseline	0.7746 ± 0.007	0.6846 ± 0.014	0.4716 ± 0.008
$\mu = 0$	−0.003 ± 0.008	−0.001 ± 0.006	−0.001 ± 0.004
$\gamma = 0$	0.013 ± 0.012	−0.040 ± 0.015	−0.037 ± 0.008
All 360 side effects	−0.002 ± 0.007	−0.022 ± 0.030	−0.013 ± 0.021
Only inductive	−0.021 ± 0.030	−0.167 ± 0.001	−0.138 ± 0.020
Using fingerprint	0.019 ± 0.006	−0.014 ± 0.005	−0.034 ± 0.014
No fingerprint	0.002 ± 0.007	−0.042 ± 0.020	−0.042 ± 0.010

Configuration	**Precision**	**Recall**	**PatR**
Baseline	0.4425 ± 0.012	0.5104 ± 0.021	0.2591 ± 0.004
$\mu = 0$	−0.006 ± 0.013	0.0020 ± 0.014	−0.001 ± 0.003
$\gamma = 0$	0.021 ± 0.024	−0.097 ± 0.050	−0.015 ± 0.005
All 360 side effects	−0.006 ± 0.010	−0.016 ± 0.042	−0.007 ± 0.008
Only inductive	−0.050 ± 0.040	−0.196 ± 0.043	−0.046 ± 0.003
Using fingerprint	0.0380 ± 0.017	−0.100 ± 0.032	−0.002 ± 0.003
No fingerprint	0.001 ± 0.019	−0.079 ± 0.045	−0.017 ± 0.005

Table 5.16: Ablation study of MolecularGNN. Values are shown as the (mean ± std) of the results of 10 learning procedures each. The best result for each metric is highlighted in red. Baseline is the MolecularGNN model architecture, as described in Table 5.12.

has an excellent effect on the F1–score (0.042) and on the PatR (0.017%), bringing the second–best improvement for both metrics, only behind the inductive–transductive learning approach. However, it also caused notable decreases in accuracy, AUC, and precision. This was probably the consequence of a mix of overfitting and the gained expressiveness, which was mainly directed toward the F1–score due to changes to the loss function. This was further reinforced by the simultaneous use of both molecular embedding and sklearn fingerprint, which did not increase accuracy but instead decreased the F1–score, the AUC, and the PatR.

The modifications to the loss function brought notable improvements: the use of focal cross–entropy described in Section 5.5.3 caused big increases in AUC (0.04), F1–score (0.04), and recall (0.10). It also caused the third–biggest increase in PatR, which is arguably the most important metric in the scope of real–world usage. All these came at the cost of slight decreases in accuracy and precision at a classification threshold equal to 0.5. Considering the massive increase in recall, the reduction in precision should be easily compensated by increasing the threshold. In contrast, the effect of weighting the loss based on the number of positive examples, as described in Section 5.5.3, is very small, but it is mostly beneficial, with the exception of the recall which showed a very slight improvement using $\mu = 0$ (no weighting).

5.5.7 Usability

The model presented in the section also brings consistent improvement to the usability of the method, as it integrates two successful strategies for solving the same problem: exploiting the molecular graph to predict the side effects of the drug [170] and predicting the side effects on a wide graph that integrates heterogeneous information on the drug and the genes it interacts with [103]. These two strategies have been applied separately so far, but they can be combined thanks to the ability of GNNs to process different forms of graph structured data, which is a very promising solution. This was demonstrated by the improvements in F1–score and PatR score shown in the results and will lead to better usability in future real–world scenarios. One of such scenarios, as already partially discussed in Section 5.3.7, will consist of building a fully automated pipeline of deep learning–aided drug discovery, in which a molecular graph generator such as GraphVAE [173], JTVAE [174], or MG^{2}N^2 [26] could generate large quantities of potential drug candidates. After a first filtering step for discarding the compounds with low QED scores [175] or low druggability scores [107, 108], MolecularGNN could be employed to screen out all the molecules with relevant probabilities of producing side effects. This pipeline could provide a large quantity of potential drug candidates with good drug–likeness and small side effect profiles, thereby constructing a reliable chemical space from which drug candidates can be drawn by experts, contributing to cutting the costs and difficulties of drug discovery research.

5.6 Conclusions and Future Work

Combining data from multiple sources is essential for a DNN to effectively learn the complex mechanisms regulating the occurrence of DSEs. The relational information of interactions between drugs and genes can be effectively represented by a graph structure. By integrating these entities and their relations, a graph dataset was constructed for training and testing graph–based DSE predictors. The GNNs demonstrated strong learning capabilities on this dataset, indicating that a predictor based on this type of model could aid in predicting the occurrence of side–effects. Additionally, its application to new candidate drugs could save time and money in drug discovery studies, as well as prevent health concerns for individuals participating in clinical tests.

DruGNN is a modular approach for predicting DSEs and is robust to ablation. Moreover, it is easily adaptable to new drug compounds by simply adding the new drug, along with its features and gene interactions, as a node in the graph and running the prediction of its classes. The model does not require retraining and the same inductive–transductive learning process can be applied for future additions of compounds and predictions of their side–effects. The prediction is based on a modular multi–omics robust approach, using information obtained from publicly available sources. In principle, the same graph could also be used to predict drug–gene interactions of new compounds by applying link prediction to the gene set.

At a higher resolution, the DSE prediction task can be tackled with GNN–MGSEP, based on the molecular graph describing the drug structure. A dataset of molecular graphs and associated side–effects was built in order to train and test the model. The experimental results show that the model is capable of very good performance on the task of DSE prediction. Exploiting only the molecular graph, it is able to obtain comparable performance, and in some cases even better performance, with respect to the state of the art methods in such task, which need large loads of information from heterogeneous sources to formulate their predictions — even though a direct comparison is not possible due to the different nature of the data used by each predictor.

Combining these two settings, it is possible to produce an enhanced and innovative version of DruGNN: the new model, called MolecularGNN, is able to work directly on the molecular structure and to work on unbalanced datasets and a graph–of–graphs domain. The novelty of this model is represented by the generation of drug structural fingerprints, which can be adapted to the task the general model is trained for. In particular, a drug's molecular structure is processed by a GNN sub–model, producing a NF which is then used to enrich drug node features in the DruGNN graph dataset, which are eventually processed by the following GNN sub–model to predict the DSEs of drug nodes. This new technique, trained with an inductive–transductive approach, together with a graph–based NF generation, succeeded in identifying more experimentally observed DSEs than DruGNN: this task is considered much more important than identifying DSEs which do not arise for a given drug, as a non predicted DSE could cause little harm or be potentially fatal. Although MolecularGNN is still limited, with an F1–score of less than 0.5, its precision at recall of 90% performance indicates that it could possibly be used as an aide during drug development processes.

Since drug structures proved to be one of the most important parts in the dataset for DL, an interesting future direction is represented by the augmentation with features coming from the gene side and drug–gene relations. In this scope, the algorithm could even be combined with generative models, like MG^{2}N^2 [26], which generate molecular graphs of possible drug candidates in large quantities. The task of the DSE predictor would be to screen out all the candidate compounds with high probabilities of occurrence of particular side–effects.

Another potential avenue of research is to specialize some of the predictors discussed in this work by incorporating tissue–specific data, such as gene expression, to fine–tune a dedicated version of the model for each tissue. This could be achieved by utilizing tissue–specific side–effect targets, resulting in a more detailed prediction that could also be personalized based on an individual's gene expression values, as it is relevant in the context of precision medicine.

Chapter 6

Graph Neural Networks for the Prediction of Protein–Protein Interfaces

In this chapter, a predictor of protein–protein interfaces based on GNNs, corresponding to publication [**P1**] is presented.

Proteins are fundamental molecules for life. They are involved in any biological process that takes place in living beings, carrying out a huge variety of different tasks. In these molecules, functionality and structural conformation are strictly correlated [176]. Therefore, analyzing structural features of proteins is often useful in understanding which biological processes they are involved in, which ligands they bind to, and which molecular complexes they form.

The structure of a protein can be described at three different levels: the primary structure corresponds to the sequence of amino acids it is composed of; the secondary structure corresponds to the local conformation of the peptide chain, in the shape of α–helices, β–sheets or coils; the tertiary structure represents the three–dimensional configuration of the molecule. Often, two or more molecules bind together to form a protein complex, whose shape goes under the name of quaternary structure. Dimers are the simplest protein complexes, as they are composed of just two monomers. To form such complexes, monomers interact through specialized parts of their surface, called binding sites or interfaces. These interactions can be studied with the help of graph theory. Indeed, each monomer can be represented as a graph, with nodes corresponding to Secondary Structure Elements (SSEs), while edges stand for spatial relationships between adjacent SSEs, which can be parallel, anti–parallel or mixed. Using graphs of two different monomers, a correspondence graph can be built, whose nodes describe all the possible pairs of SSEs from the two different subunits [5]. Based on the correspondence graph, identifying binding sites on protein surfaces can be reformulated as a maximum clique search problem [6].

The maximum clique problem is known to be an NP–complete problem, meaning that, except for very small graphs, traditional operations research algorithms [177] will employ a prohibitive amount of time before solving it. From this consideration stemmed the idea of using a ML method to solve the problem with reasonable computational costs. In particular, GNNs [1] look like the perfect model, with their ability to process graph–structured inputs. In this scope, the maximum clique problem consists of a binary classification between the nodes which belong to the maximum clique and those which do not. In particular, the solution proposed in this chapter entails applying LGNNs [73] to solve the maximum clique problem.

The rest of the chapter is organized as follows: Section 6.1 illustrates the method, sketching the data acquisition and processing operations. The implementation details on GNNs and LGNNs is given in Section 6.2, while the experimental methodology is described in Section 6.3. Finally, Section 6.4 presents and gives interesting insights on the results of the work, and Section 6.5 discusses the results of the approach and draws conclusions.

6.1 Dataset

In this section, the data and the experimental methodology used in this work is described, with a particular focus on the dataset construction for the protein–protein interface prediction task, in which pairs of monomers are associated to a correspondence graph.

To build the dataset, heterodimers (i.e. dimers formed by two different monomers) characterized by the absence of disulfide bridges, the presence of salt bridges, and protein–protein interaction sites were searched in the Protein DataBank in Europe (PDBePISA) [178]. A database of 6,695 known protein has been obtained for a total of 160,680 monomeric interfaces. To guarantee biological significance, some criteria were enforced: an area of at least 200 Å^2, $\langle x, y, z \rangle$ symmetry, and only two interacting protein molecules. After this operation, a set of 12,455 interfaces has been obtained. For every

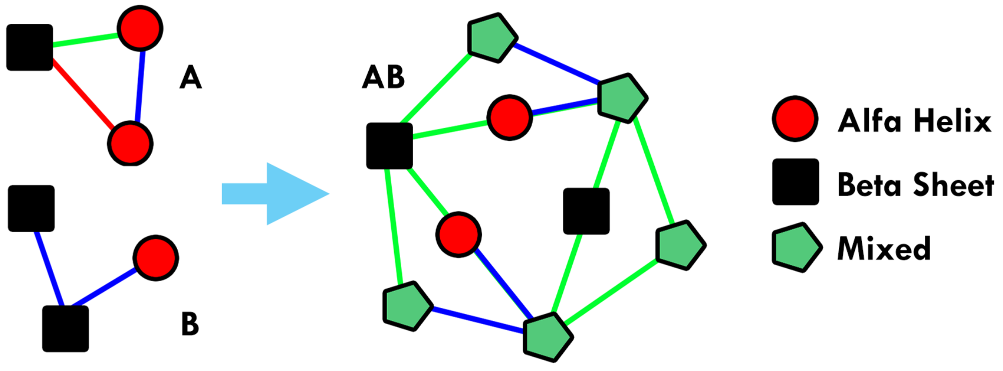


Figure 6.1: Example of the construction of a correspondence graph from two generic monomers graphs.

interface, two protein graphs were built, representing two polypeptide chains which interact on the binding site.

The monomeric graphs were built using VPLG [179], with PDB [180] and DSSP [181] files representing the whole protein. Each node v is labeled with a feature vector l_v which consists of: an ID number, the SSE type, the number of occurrences of cysteine and that of the aromatic amino acids (tyrosine, tryptophan and phenylalanine), the percentage of amino acids taking part in the interface and the overall hydrophobicity [182], the charge and Accessible Surface Area (ASA) of the SSE, respectively as the sum of hydropathic indexes, charges and accessible surface areas of each amino acid at pH 7.

Once the graph has been produced for both monomers, it is possible to build the correspondence graph [6, 5]. Let $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ be the graphs representing two protein chains and $G = (N_G, E_G)$ be the correspondence graph of G_1 and G_2. Let $v_i, u_i \in N_i$ be two generic nodes in G_i with $i = 1, 2$. Therefore, two nodes $v = (v_1, v_2)$, $u = (u_1, u_2) \in N_G$ are connected by an edge $(v, u) \in E_G$ if and only if $\exists(v_1, u_1) \in E_1$ and $\exists(v_2, u_2) \in E_2$. An example of the construction of a correspondence graph from two generic graphs is depicted in Fig. 6.1.

The edge label $e_{u,v}$ is a one–hot representation of the spatial relationship between two adjacent nodes in G, which depends on the labels e_{v_1,u_1} and e_{v_2,u_2}, so that $e_{v,u}$ is the same edge label if both the edge labels in G_1 and G_2 are equal, *mixed* otherwise. The label of node $v \in N_G$ consists of: an

Dataset	**Graphs**	**Edges**	**Nodes**	**Nodes0**	**Nodes1**	**%Nodes1**
Before Pruning	512	441,203	328,629	325,798	2,831	0.86 %
After Pruning	1,044	274,608	166,424	163,593	2,831	1.7 %

Table 6.1: Dataset statistics before and after data cleaning. Nodes0 and Nodes1 represent negative and positive nodes, respectively.

ID number, a one–hot representation of the SSE type, the differences in the occurrences of cysteine and the aromatic amino acids, the arithmetic mean of the two hydrophobicity values, the minimum of the ASAs and the sum of the charges of the two SSEs. In particular, the SSE type of the node $v \in N_G$, which represents $v_1 \in N_1$ and $v_2 \in N_2$, is the same as that of the nodes v_1 and v_2 if both belong to the same SSE class, while it is defined as *mixed* if they belong to different SSE classes.

The node targets were generated with the Bron and Kerbosch algorithm [183], which identified the cliques within each correspondence graph, with a minimum size of three nodes. Subsequently, these cliques were analyzed, in order to determine whether or not they were biologically significant. In this context, a clique is defined as positive or biologically significant if and only if all the nodes belonging to that clique represent pairs of SSEs of different monomeric graphs, that contain both at least one residue that is part of the interface. Hence, the target attached to each node is a two–dimensional vector containing a one–hot encoding of the two classes: positive if the node belongs to a biologically significant clique, negative otherwise. 512 correspondence graphs have been obtained with this procedure, each containing at least one biologically significant clique (and any number of negative cliques) composed of three or more nodes. These were not completely connected, often being made of multiple separated connected components. Since many connected components did not include cliques, a pruning strategy was adopted, in order to clean the dataset. The correspondence graphs were split, obtaining a graph for each connected component, as depicted in Fig. 6.2.

Those not containing at least one clique have been filtered. As a result, the dataset is composed of graphs containing at least one clique, whether positive or not. This operation produced the final dataset of 1044 connected graphs, 537 of which contain a positive clique, while the remaining 507 contain only negative cliques. Table 6.1 provides numerical information on the dataset, before and after the pruning process.

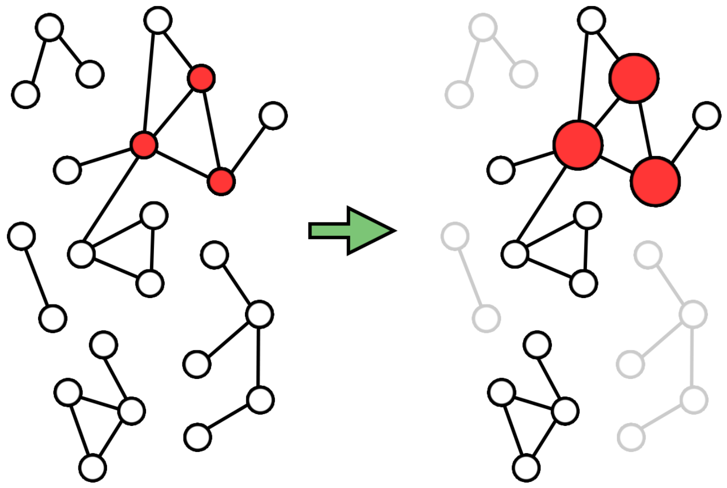


Figure 6.2: Pruning strategy applied to the dataset. Graphs were split into their connected components, filtering those not containing positive or negative cliques, as well as isolated nodes. Moreover, nodes belonging to a positive clique were weighted more in the learning procedure.

6.2 Model

In the work described in this chapter, GNNs are implemented as formulated in Section 2.3.1, with no features on the edges. Moreover, only the sum of the incoming message is used as their neighborhood aggregation function, namely in all the experiments the hyperparameter a has value $a = 1$. The state updating function f_w defined in Eq. (2.6) can be re–written as in Eq. (6.1):

$$x_n^t = f_w(x_n^{t-1},\ l_n,\ \sum_{m \in Ne(n)} (x_m^{t-1},\ l_m)) \tag{6.1}$$

Since this is a node classification task, the output function g_w is implemented as defined in Eq. (2.7).

6.3 Experimental Setup

A binary GNN classifier was developed for the detection of maximum cliques in the correspondence graphs, which addresses the problem as a node–based classification task. Usually, in a classification task, the performance is measured in terms of accuracy. This metric, though, is not precise on unbalanced datasets, like the one under analysis. Therefore, to evaluate the model's performance the F1–score has been used, which combines precision and recall to provide a balanced measure.

The architecture of the MLP module dedicated to the output function g_w was kept fixed, using a single level MLP and the softmax activation function. On the contrary, a 10–fold cross–validation was performed in order to determine the best hyperparameters for the MLP implementing the state updating function f_w. According to the cross–validation results, the MLP architecture with better performance has got a single hidden–layer with logistic sigmoid activation functions. This setup was used also to test a 5–layered LGNN network, where each GNN layer shares the same architecture.

In order to evaluate the performance of the LGNN, a 10–fold cross–validation was carried out. The LGNN is composed of 5 GNN layers, each with state dimension equal to 3. The state is calculated by a one–layer MLP with logistic sigmoid activation, while the output is calculated with a one–layer MLP with softmax activation. Since the negative/positive examples ratio is quite large, the weight of positive examples is fixed to the 10% of this ratio, against a weight of 1 for negative examples, in order to balance the learning procedure. The model is trained with the Adam optimizer [72] and the cross–entropy loss function.

6.4 Results and Discussion

The best performance is obtained with LGNNs integrating only the state in the node labels. There are slight improvements in precision and more tangible improvements in recall, which gains more than 10 percentage points in the second GNN level, and then continues to grow and stabilize in the following levels, as shown in Fig. 6.3. This architecture is the only one in which a significant increase of the F1–Score is observable, getting more than 6 percentage points from nearly 35% of the first GNN level to more than 40% in the final GNN level, as reported in Table 6.2.

Contrariwise, integrating in the node labels only the output or both the state and the output, the F1–score decreases through the LGNN layers. The other parameters remain almost stable, except for recall, which slightly increases through the LGNN layers. However, the standard deviation of the recall tends to grow, suffering from a marked dependence on the initial conditions of the experiment. The results confirm the expectations based on biological data and show good performance in determining the interaction sites, recognizing on average about 60% of the interacting nodes.

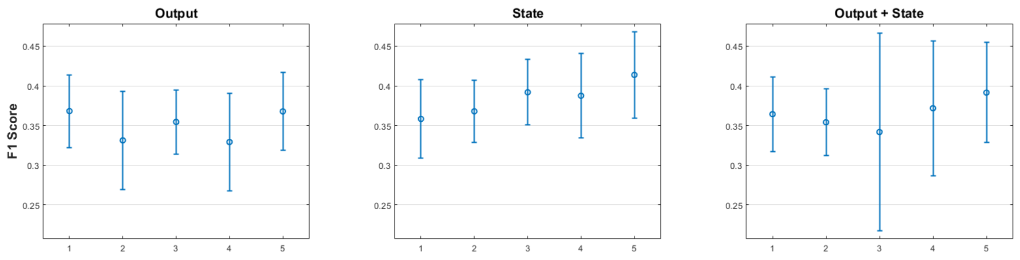


Figure 6.3: 5 levels LGNN 10–fold cross validation results: F1–score

Output	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Precision	0.319 ± 0.069	0.271 ± 0.058	0.287 ± 0.049	0.266 ± 0.046	0.295 ± 0.070
Recall	0.455 ± 0.048	0.447 ± 0.111	0.476 ± 0.061	0.446 ± 0.101	0.517 ± 0.059
F–Score	0.368 ± 0.046	0.331 ± 0.062	0.354 ± 0.040	0.329 ± 0.062	0.368 ± 0.049

State	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Precision	0.310 ± 0.061	0.279 ± 0.045	0.322 ± 0.052	0.295 ± 0.053	0.328 ± 0.061
Recall	0.436 ± 0.063	0.558 ± 0.056	0.524 ± 0.087	0.585 ± 0.080	0.571 ± 0.067
F–Score	0.358 ± 0.050	0.368 ± 0.039	0.392 ± 0.041	0.387 ± 0.053	0.414 ± 0.055

Both	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Precision	0.308 ± 0.063	0.273 ± 0.052	0.261 ± 0.106	0.301 ± 0.064	0.296 ± 0.060
Recall	0.460 ± 0.056	0.544 ± 0.109	0.520 ± 0.185	0.518 ± 0.168	0.597 ± 0.096
F–Score	0.364 ± 0.047	0.354 ± 0.042	0.342 ± 0.125	0.372 ± 0.085	0.392 ± 0.063

Table 6.2: Results obtained with three different LGNN settings: propagating the output, the state or both from one layer to the next

6.5 Conclusions and Future Work

The problem of protein–protein interface detection has been addressed as a search for the maximum clique in the interface correspondence graph [23]. Although the problem is NP–complete, the described method, based on

GNNs, can find the maximum clique in affordable time. The performance of the model was measured in terms of F1–score and shows that the approach described in this chapter is very promising, though it can be further improved. One key idea in this direction is that of using graphs in which the nodes correspond to single amino acids, rather than to SSEs. Although this latter approach would increase the complexity of the problem, it would avoid the loss of information in compressing amino acid features into SSE nodes. Moreover, predictions obtained in this setting would be more accurate, describing the binding site at the amino acid level.

Chapter 7

Other Works

This chapter provides an overview of the other activities carried out during the Ph.D. and not strictly related to the content of the thesis. In Section 7.1, subject of [**P3**], a proof of concept of a GNN–based mobile app is proposed, which could make it easier for caregivers of rare disease patients to share knowledge and assistance.

Then, in Section 7.2 two statistical approaches for sequential data are provided: the former, subject of the publications [**A3**] and [**A5**], is described in Section 7.2.1 and overviews a method for differentiating between classes of patients in medical applications, based on data collected by an eye–tracker device; the latter is the topic of publication [**A6**] and illustrated in Section 7.2.2, describes a statistical and DL–based method for validating sequences produced by the Ribo–Seq Profiling technique.

Eventually, Section 7.3 provides a description of two applications of DL techniques to image analysis tasks coming from the real world: in Section 7.3.1 a CNN model for dragonfly action recognition [**P2**] is described, while Section 7.3.2 illustrates a DL approach for the segmentation and analysis of oocytes images [**P4**].

7.1 GNN–Based Caregiver Matching

A rare disease is defined as a disorder having a low occurrence in the population and is often chronic and potentially life–threatening. There are estimated to be over 6,000 rare diseases, with a prevalence ranging from 3.5% to 5.9% depending on local definitions. This results in 263–446 million people

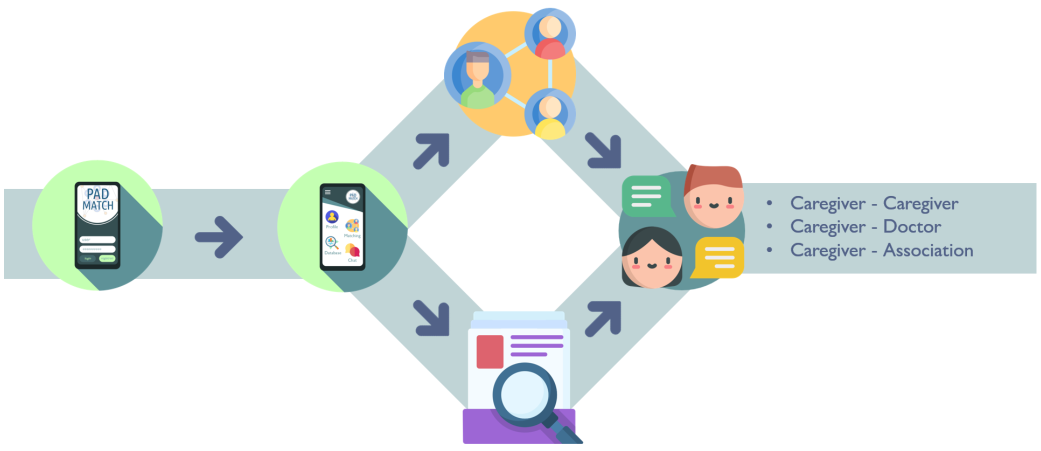


Figure 7.1: General architecture of CaregiverMatcher mobile app. From the left: to access the platform, caregivers log in with username and password. Four sections are available in the home page: Profile, to manage personal and patient data; Chat, where all messages and chat conversations are stored; Get Informed to retrieve rare diseases information as well as associations or doctors contacts; Match to start the matching process. As a result, caregivers can then connect with patient associations, specialized clinicians and other caregivers.

affected worldwide [184]. Caregivers, who can be family members or hired professionals, provide daily assistance to those affected by rare diseases. The constant attention required and social isolation experienced by caregivers can present obstacles in their daily assistance [185]. To address these issues, a network of caregivers is considered to be extremely valuable [186].

In this section, derived from publication **[P3]**, the proof of concept of a cross–platform application in support of the caregiver's experience is presented. The proposal is called CaregiverMatcher [20], and its aim is to create a network of caregivers of rare disease patients. This project was presented at the Rare Disease Hackathon 2020, organized by the Italian Forum Sistema Salute, reaching the final stage reserved to the best 8 proposals, among more than 250 projects. CaregiverMatcher exploits GNNs [1] to link each caregiver with other caregivers that face similar issues in daily assistance, building a network of contacts, based on information on the assisted patients. The design of this application is intended to be user–friendly and present informative sections to improve the knowledge about rare diseases, as show in Fig. 7.1.

The core, and main novelty, of CaregiverMatcher is the idea of connecting caregivers with GNNs. From a practical point of view, the application builds links between caregivers based on both patient personal information and health condition. The GNN model is asked to predict whether an edge exists between each pair of caregiver nodes. The predicted presence or absence of an edge represents the existence of a caregiver–caregiver relationship, and it is weighted according to a real–valued similarity score describing how compatible their profiles are: the higher the score, the higher the compatibility between the connected users. Eventually, once the matching process has been completed, the user is returned a list of similar caregivers, filtered as needed by setting some parameters in a dedicated section.

It is well established that support groups for caregivers can have a positive impact on their mental health, level of burden, and overall social well–being [187]. However, it's important to note that the full potential of the application will only be realized after an unspecified period of time, as the GNN model on which the system is based needs a sufficient number of users in order to effectively match caregivers within the network.

In summary, CaregiverMatcher has the potential to provide numerous advantages to caregivers in various areas of their lives. This includes improvements in mental health, as the application offers both psychological and practical support as well as easy access to credible educational resources provided by experts and organizations.

7.2 Multi–Modal Data Analysis

This section presents two mixed statistical and ML approaches applied on sequential data. Statistical analysis obtained from visual tests [188, 189] are discussed in Section 7.2.1 to identify patterns within the data in order to differentiate between people affected by chronic pain, extrapyramidal disorders or healthy people. Then, statistical and ML approaches are described in Section 7.2.2 to validate data produced by translation procedure analysis.

7.2.1 Visual Sequential Search Test

An algorithmic approach for the analysis of the Visual Sequential Search Test (VSST) is proposed in this subsection, to have an insight into the patient condition and to offer a support for clinical application [188, 189]. Eye–

tracking can offer a novel clinical practice and a non–invasive tool to detect neuro–pathological syndromes, with its capacity of detecting eye position and speed of movements [190]. This study focuses on the analysis of data coming from the Visual Sequential Search Test (VSST), a neuro–psychological test which measures the ability to detect and locate a target stimulus in images, to assess cognitive impairment or working memory. The test consists of a series of trials in which a target stimulus (such as a letter or shape) is presented among a set of distractors. The participant is asked to identify the target stimulus as quickly and accurately as possible. The time taken to complete each trial is recorded and the test is designed to measure the speed and accuracy of visual search performance.

In this work, VSST data is analyzed, examining both the blinking behaviour and the pupil size of the subjects, to gain an insight into the patient condition and offer a support for the clinical practice. For this purpose, several indicators to distinguish among classes of patients have been compared.

Data has been collected from people affected by extrapyramidal syndrome, by chronic pain or from healthy people (46, 284 and 46 examples, respectively, for a total of 376 participants), and consist of multiple time–sampled series provided by the eye–tracker instrument, containing information on bidimensional gaze position, left and right pupil sizes, fixation IDs and the stimulus code). Individuals affected by extrapyramidal symptoms suffer from tremors, rigidity, spasms, decline in cognitive functions (dementia), affective disorders, depression, amnesia, involuntary and hyperkinetic jerky movements, slowing of voluntary movements, and postural abnormalities. Conversely, there are several mechanisms underlying chronic pain; more often an excessive and persistent stimulation of the “nociceptors” or a lesion of the peripheral or central nervous system, but there are also forms of chronic pain that do not seem to have a real, well–identified cause (neuropathic pain). Therefore, chronic pain can be related to a variety of diseases, with very different severity, from depression, to chronic migraine and to cancer.

A Kruskall–Wallis test [191] was applied to the distributions of the blinking rate, maximum pupil size variations and mean blinking duration. The analysis showed the presence of some statistically significant differences between the groups under analysis. In particular, no significant differences are found between healthy subjects and patients affected by extrapyramidal syndrome considering the three indicators. Conversely, a significant statistical difference between healthy controls and chronic pain patients was found for the rate of blinking and the variation of pupil size. Concerning the comparison between patients affected by chronic pain and extrapyramidal syndrome, a significant difference was detected both in the maximum pupil size variation and in the blinking average duration.

Limitations of this work are mainly due to the small dataset available. Variations of the VSST could be implemented and standardised, to avoid biases due to the fact that no instructions were given concerning the number of times the patients should have completed the sequence during the data acquisition time. Therefore, future research and extensions will concern new standardised data collection for further testing and a more extensive validation of the approach based on a wider experimentation, for example by considering more than three mutual exclusive classes, so as to include co–morbidities, i.e., cases in which additional conditions are concurrent to the primary one.

7.2.2 Validation of Ribo–Seq Profiles

This subsection describes a work, the subject of publication [**A6**], in which statistical analysis and ML models have been used to validate the quality of sequences produced by the so–called Ribosome Profiling Technique (Ribo–Seq). Ribosomes perform protein synthesis from mRNA templates by a highly regulated process called translation, which plays a key role in the regulation of gene expression, both in physiological and pathological conditions. In recent years, Ribo–Seq has emerged as a powerful method for globally monitoring the translation process in vivo at single nucleotide resolution. Interestingly, the nucleotide—level resolution of Ribo—seq experiments reveals the density of the ribosomes at each position along the mRNA template. Local differences in the density of Ribosome Protected Fragments (RPFs) along the Open Reading Frame (ORF) reflect differences in the speed of translation and elongation, thus determining regions where the translation is slower or faster. Unfortunately, the reproducibility of Ribo—seq experiments can be affected by multiple variables due to the complexity of the experimental protocol and the lack of standardization in computational data analysis.

Inspired by the seminal work [192], a novel analysis procedure for Ribo–Seq data is performed. Based on the collected data from Escerichia Coli (E. coli) sequences — 40 highly reproducible profiles resulted from the Rib–

Seq procedure —, a statistical analysis has been carried out that gave new insights on the dynamics of the ribosome translation, showing a statistically significant difference in the nucleotide composition between sub–sequences characterized by different translation speeds [193]. Data has been generated from E. coli sequences retrieved from different datasets available online, in the form of 40 digitalized highly reproducible Ribo–Seq profile sequences from which 264 sub–sequences characterized by a slow translation and 195 by a fast translation have been extracted, for a total of 459 sub–sequences of variable length.

The statistical analysis consisted in computing the relative frequencies of each nucleotide in the fast and slow sub–sequences, respectively, showing a significant larger concentration of the adenine w.r.t the other nucleotides. To assess the significance of the results, a statistical test has been performed in order to determine whether the frequencies obtained are specific characteristics of fast and slow sub–sequences or they are simply a product of chance. The obtained results suggests that the relative frequencies of the four nucleotides are significantly different from those occurring randomly. In particular, nucleotides A and T show a higher frequency in fast sequences than G and C. Instead, the frequency of nucleotides G and C is significantly higher in slow sequences than those of A and T. Based on this evidence, the proposed method for the identification of reproducible Ribo–Seq profiles was able to correctly detect sub–sequences characterized by a higher information content with respect to random sub–sequences, confirming the validity of the new approach.

Another experiment has been carried out, in which the informative content of the obtained data has been validated using two ML–based approaches, in the form of two binary classifiers whose task is to predict the translation speed class of the sub–sequences. The two neural network architectures, which are capable of processing both plain and sequential data, are highly effective in obtaining high accuracy. Furthermore, it is shown that fundamental information is contained in both the nucleotide composition of the sequences and the order in which nucleotides appear within each sequence.

This research opens new frontiers in the analysis of ribosome translation dynamics in different organisms. Despite the complexity of the human data, further analysis is deemed necessary in order to improve performance. This may include the use of specialized neural architectures and hyperparameter search. It is notable that the proposed method represents an effective approach for analyzing any type of Ribo–seq data and investigating the open question of what features influence the speed of ribosome during translation.

7.3 DL Applications for Image Analysis

This section presents two DL techniques applied on image data. Sequences of images representing dragonflies are analyzed in order to recognize the actions the dragonflies are performing [194]. Then, an oocyte classifier based on CNNs and attention mechanism is proposed [195], in order to support human operators in in-vitro fertilization (IVF) procedures.

7.3.1 Dragonfly Action Recognition

This subsection presents a work, described in publication [**P2**], in which a CNN–based classifier for recognizing the different phases of dragonfly flight in images is presented. Dragonflies are very complex and advanced flying organisms: they are characterized by a long and thin body, two large multifaceted eyes, two pairs of transparent wings and six legs. They can move the four wings in a fully independent way, giving them extraordinary agility, and obtaining formidable performance in flight and hunting, where they can perform backward movements, very narrow vertical and horizontal loops and stops in mid–air. All these features make dragonfly flight very interesting to study, while also making dragonfly action recognition a difficult task. The development of a reliable system for recognizing dragonfly actions is useful as it has wide interest in the biological research community. Indeed, research in different fields could benefit from the recognition system to test hypotheses on dragonfly anatomy, flight dynamics and predatory behaviors.

More specifically, the proposed model classifies video frames in five classes: take–off, flight, landing, stationary and absent (frames in which the dragonfly is not present). A limitation in the development of a DL model is the requirement for a large set of fully annotated data. In this case, there was not a publicly available labeled dataset of dragonfly images. In order to train a DL architecture, a sufficient quantity of samples were gathered from online videos. These samples were preprocessed and labeled on a frame–by–frame basis. Two distinct classifier networks were evaluated for action recognition: a standard CNN, which processes one frame at a time, and an LSTM, which processes sequences of frames.

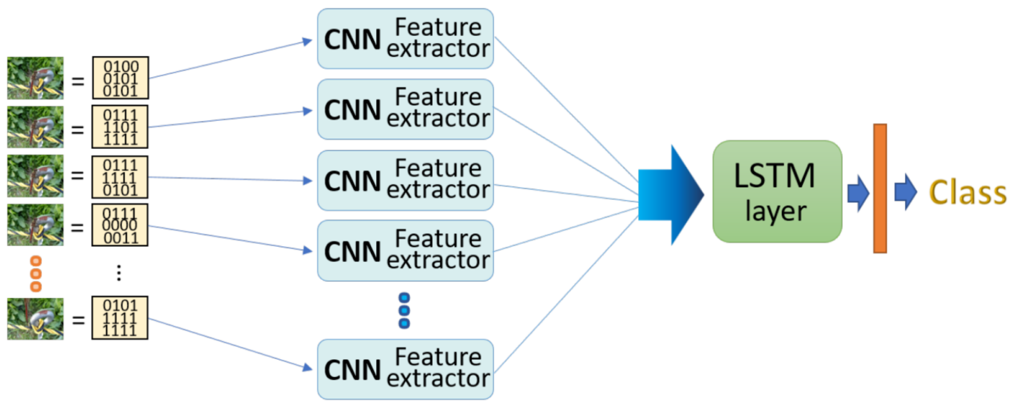


Figure 7.2: Architecture of the CNN + LSTM hybrid model. The images are converted into 224×224×3 matrices and processed separately by replicas of the same CNN model (in blue). The results are then processed by the LSTM layer (in green). Finally, dense layers (in orange) re–elaborate the information and perform the classification.

A first set of experiments was carried out with the aim of identifying a good combination of hyperparameters, such as the number of convolutional blocks or the number of feature maps. Transfer learning was then applied, exploiting pre–trained models that could extract low–level image features efficiently and with short training times. Three CNN models which are well known in the literature were used for transfer learning: the MobileNet–v2 [196], the VGG16 [197] and the DenseNet121 [198]. All of them are pre— trained on the ImageNet dataset [199].

The experimentation on LSTMs was carried out on the same dataset, but taking into account the sequentiality of frames. Since the recurrent layers are not able to process multidimensional data, a CNN–based feature extractor was employed, in order to transform each frame into a feature vector, compatible with the recurrent layers of the LSTM model. The sequence of vectors was then analyzed with the LSTM model, characterized by 2 dense layers composed of 100 and 5 neurons, respectively, as depicted in Fig. 7.2 To analyze the entire sequence, a sliding window with a size of 7 frames was employed.

The experimental results showed that both models (CNNs and LSTMs) reached an accuracy score of 73.9 %, yet with some difficulties in recognizing specific classes. This work demonstrated that DL techniques can be successfully applied to the dragonfly action recognition task. In particular, a good set of guidelines for the automatic analysis of dragonfly flight has been provided. These guidelines include new instructions for setting up a dataset, as well as useful considerations for the calibration, design and implementation of deep models to face this complex task.

7.3.2 Oocyte Segmentation

Medical Assisted Procreation (MAP) has seen a sharp increase in demand over the past decade, due to a variety of reasons, including genetic factors, health conditions altered by stress and pollution, as well as delayed pregnancy and age–related loss of fertility. The success of MAP techniques is strongly correlated to the dexterity of human operators, who are asked to classify and select healthy oocytes to be fertilized and returned to the uterus. This work describes a deep learning approach to the segmentation of oocyte images, to support operators in their selection, to improve the success probability of MAP. This work focuses on second level in–vitro fertilization (IVF), a set of pharmacological therapies, surgical and laboratory procedures, and in particular on ICSI (IntraCytoplasmic Sperm Injection) procedure, in which the spermatozoa are directly injected into the oocyte. It is worth noting that the success of ICSI depends, among the other genetic factors, on the capability of the human operator to perform a good selection of embryos: this work therefore aims at showing how a semantic segmentation DL approach can be used to support the operators, to improve the success probability of the IVF procedure.

The dataset was initially composed by 40x inverted images captured with a Leica Micromanipulator with Nikon camera (Fig. 7.3a), provided with respective masks highlighting inclusions (Fig. 7.3b).

Split	W/ Inclusion	Resolution
Training Original	18	256×256
Training Augmented	1000	256×256
Training Synthetic	1000	256×256
Test Original	8	256×256

Table 7.1: Dataset organization. Training Original are the set of microscope images; Training Augmented are the images from the training set augmented by randomly applying flips and rotations; Training Synthetic are the set of synthetic images augmented by generation; Test Original represents the microscope images set used as a test set.

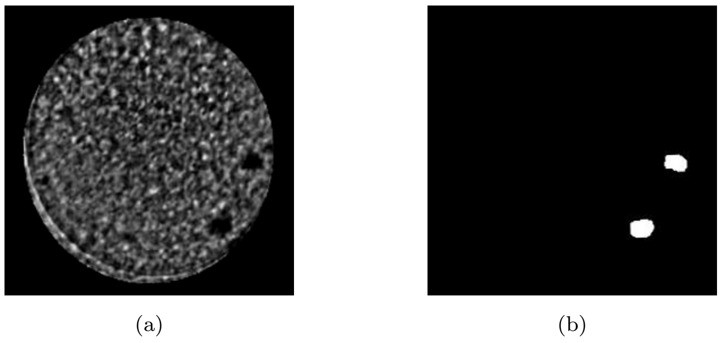


Figure 7.3: Sample image from the training set in (a); the corresponding mask in (b)

The semantic segmentation task has been tackled by the means of a Segmentation Multiscale Attention Network (SMANet) [200], a DL architecture for image segmentation, formed by three main components: a ResNet encoder, a multiscale attention module, and a convolutional decoder. In particular, a ResNet50 architecture [65] has been used as the encoder in order to perform feature extraction of the input image. The convolutional encoder is followed by an attention module and a convolutional layer providing pixel–wise prediction, which learns to focus on regions containing the desired object, and by a two level decoder to recover small details at a higher resolution.

In order to successfully train a deep segmentation network, a large amount of supervised data is usually required. Nonetheless, data deficiency is a typical drawback of DL biomedical applications, due to privacy issues and to the inherent difficulty on obtaining supervised data. Therefore, a data augmentation procedure was carried out, based on a DL image generation technique, to face the lack of a large set of annotated data. In particular, an image–to–image translation approach was used, to transform semantic label maps in fake oocyte images, in order to enlarge the small available training set.

The label maps are created by randomly placing from one to five circular spots with variable diameter (from 2 to 7 pixels), since this closely resembles the true distribution of the inclusions in the oocyte which, based on the opinion of experts, is totally random.

The experimental setup has been divided in three cases, involving training only on real images, training only on synthetic images, and performing fine–tuning, in which a model pre–trained on synthetic images was fine–tuned on real images. As a first outcome, it is worth noting that synthetic images can be effectively used as data augmentation. Indeed, better results are achieved by training on synthetic images rather than using real data, demonstrating the high quality of the generated images. Moreover, the results can be further improved by fine–tuning the model, pre–trained on the syntethic data, on the real images.

This work can be considered as a first step towards a more complex tool, capable to characterise oocytes in a complete manner and give the human operator a quality score, obtained through an objective and reproducible procedure. As a matter of further research, the collection of original images will be enlarged, which is expected to greatly improve the performance of the network. Furthermore, other parameters having a strong influence on the evaluation of oocytes may be considered in the segmentation task: to this aim, more detailed segmentation maps will be drawn, taking into account other characteristics and abnormalities of oocytes.

Chapter 8

Conclusions and Future Developments

This thesis is focused on GNNs and their applications in bioinformatics. A software framework, described in Chapter 4, for designing and deploying GNNs for research purposes has been developed and used to implement some applications, presented and discussed in the other chapters of this thesis. The software provides multiple Keras–based GNN models for homogeneous and heterogeneous graph processing and for both inductive and mixed inductive–transductive learning settings, thus providing a simple tool for the development of software applications in the field of ML for graphs. In particular, GNNs are employed for predicting the side–effects of drugs and for identifying protein–protein interfaces. The two applications are described in Chapter 5 and Chapter 6, respectively. Some specific conclusions on each of these works are drawn at the end of the respective chapters, and summarized in the following. Three DSE predictors are proposed and discussed in Chapter 5.

The first one, called DruGNN and described in Section 5.3, combines data from heterogeneous sources into a relational dataset allows to represent the complex features and interactions involved into the occurrence of DSEs. CGNNs are ideal to process this kind of dataset, following a mixed inductive–transductive learning scheme, and produce accurate predictions in this multi–class multi–label node classification task. Moreover, DruGNN is modular: data can be added without having to re–train the model, and robust to ablation. The proposed method can be easily used to predict the side–effects of a candidate drug, by inserting the molecule and its interaction data. A future research direction consists in specializing the predictor, by taking into account tissue specific data and DSE supervisions, obtaining more accurate and informative insights.

The second DSE predictor, GNN–MGSEP, is described in Section 5.4. It is based only on the molecular graphs of the drug structures. A dataset of molecular graphs and associated side–effects was built in order to train and test the model. The experimental results show that the model is capable of very good performance in this task. In the future, the model can be further developed in multiple directions. On the one hand, introducing a larger amount of drug examples could improve performance of GNN–MGSEP while retaining the same simplicity and lightweight style. On the other hand, the model can be refined by integrating heterogeneous data as it is the case for DeepSide and DruGNN. The more straightforward addition that could be made is constituted by the chemical features of drugs, that can be retrieved from PubChem and integrated inside the molecular graph. Other data, describing drug–protein interactions, metabolomics, gene expression, and ontologies, could be integrated as well, though this would imply a rethinking of the model to attach these pieces of information to a molecular graph.

The last proposed method for the prediction of DSEs is MolecularGNN, described in Section 5.5, which combines the DruGNN and GNN–MGSEP settings, to produce an enhanced model, able to work directly on the molecular structure and on unbalanced datasets, in a graph–of–graphs domain. The novelty of this model is represented by the generation of drug structural fingerprints, which can be adapted to the task the general model is trained for. Possible future developments in this context include a dynamic learning approach, by considering a dynamic topology of the graph, in which the connections between the drug nodes do not remain the same throughout the learning procedure, since they can be re–calculated on the basis of the similarity between the neural fingerprints generated at each epoch by the appropriate sub–model, possibly adding some constraints to respect a certain level of similarity given by the one obtained with the standard, preprocessed fingerprints.

In future real–world scenarios, as already partially discussed for DruGNN [103], it will be possible to build a fully automated pipeline of deep learning–aided drug discovery, in which a molecular graph generator such as ChemVAE [201], JTVAE [174], CCGVAE [202], GraphVAE [173], MolGAN [203], or the GNN–based MG^2N^2 [26] could generate large quantities of potential drug candidates. These methods can in fact produce massive amounts of possible drug candidates, but often lack the ability of evaluating the possible DSEs of the generated compounds. The drug candidates could then be screened for their drug–likeness, retaining only compounds with a high QED score [175] or druggability score, which can be estimated with various methods, including deep learning predictors [107, 204, 108]. GNN–MGSEP and MolecularGNN could be employed to screen out all the molecules with relevant probabilities of producing side effects. This pipeline could provide a large quantity of potential drug candidates with good drug–likeness and small side effect profiles, thereby constructing a reliable chemical space from which drug candidates can be drawn by experts, contributing to cutting the costs and difficulties of drug discovery research.

However, the black–box nature of these predictors limits their interpretability and trustworthiness: to address this limitation, explainable machine learning (XAI) models, particularly those based on GNNs, have been developed to provide further insight into the model's structure and decision-making process. In the context of drug discovery and side effect prediction, GNN–based XAI models can help identify the molecular features that are most relevant for predicting a particular side effect. By analyzing the importance of different molecular features, drug developers can gain insights into the mechanisms that underlie the side effect and potentially identify ways to mitigate it. GNN–based XAI models can also generate visual explanations of the model's predictions, providing a basis for validating and refining the model's performance. In conclusion, GNN–based XAI models have significant potential in drug discovery and drug side effect prediction. By providing interpretability and explainability, and by improving the transparency of machine learning models, GNN–based XAI can help boost the usability of the approaches and facilitate the development of safer and more effective drugs.

GNNs were also employed for predicting protein–protein interfaces, as described in Chapter 6. This task was formulated as a maximum clique search on the correspondence graph derived from the secondary structures of each pair of proteins. The approach showed very promising results, individuating the secondary structures that participate in each interface, and providing interesting insights for future research. However, further experimentation and different GNN models and architectures could improve the model. Additionally, the current SSE–based representation of the interfaces is at a coarse level and could be refined to include specific amino acids or even atoms for a more precise identification of protein–protein interactions. A first limitation is that current results are preliminary: an improved version of the model could be developed based on a more extensive experimentation, also taking into account more possible GNN models and architectures. The second drawback is represented by the coarse scale of the interface representation predicted: in fact, the same analysis could be refined to the amino acid scale, or even at the atom level, more precisely identifying the residues that take part into protein–protein interfaces.

GNNs have shown their capabilities in various molecular data settings, particularly in the field of drug discovery. Applications such as node classification in a traditional yet imbalanced setting and node classification on a complex, heterogeneous relational dataset are just a few examples of the potential of GNNs in this field. Furthermore, the development of a software framework that allows for an easy Keras–based implementation of recurrent GNNs has made it easier for researchers to access and utilize their properties.

In conclusion, the demonstrations of GNN capabilities in these testing grounds, along with the growing interest and community related to graph–based models, suggest that GNNs will continue to improve and evolve. New theories and models, such as non–local GNNs with higher computational power, will likely be proposed in the future. Additionally, GNNs will be applied to new fields and even more complex data settings, providing significant contributions to various scientific fields, including biology, physics simulations, weather prediction, and social network analysis.

Publications and Activities

Journal Articles

[**A1**] Pietro Bongini, Franco Scarselli, Monica Bianchini, Giovanna Maria Dimitri, **Niccolò Pancino**, Pietro Liò. 2022. Modular multi–source prediction of drug side–effects with DruGNN. Published in early access on IEEE/ACM Transactions on Computational Biology and Bioinformatics.
https://doi.org/10.1109/TCBB.2022.3175362
Candidate's contributions: conceptualization, software implementation, manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Section 5.3.

[**A2**] **Niccolò Pancino**, Pietro Bongini, Franco Scarselli, Monica Bianchini. 2022. GNNkeras: A Keras–based library for Graph Neural Networks and homogeneous and heterogeneous graph processing. SoftwareX, 18, 101061.
https://doi.org/10.1016/j.softx.2022.101061
Candidate's contributions: software implementation (in collaboration), manuscript reviewing and editing.
Thesis relevance: primary software contribution, presented in Chapter 4.

[**A3**] **Niccolò Pancino**, Caterina Graziani, Veronica Lachi, Maria Lucia Sampoli, Emanuel Ştefănescu, Monica Bianchini, and Giovanna Maria Dimitri. 2021. A Mixed Statistical and Machine Learning Approach for the Analysis of Multimodal Trail Making Test Data. Mathematics 9, no. 24: 3159.
https://doi.org/10.3390/math9243159
Candidate's contributions: software implementation (in collabora-

tion), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.2.1.

[A4] **Niccolò Pancino**, Yohann Perron, Pietro Bongini, and Franco Scarselli. 2022. Drug Side Effect Prediction with Deep Learning Molecular Embedding in a Graph–of–Graphs Domain. Mathematics 10, no. 23: 4550.
https://doi.org/10.3390/math10234550
Candidate's contributions: supervision, software implementation (in collaboration), original manuscript draft, manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Section 5.5.

[A5] Emanuel Ștefănescu, Niccolò Pancino, Cateringa Graziani, Veronica Lachi, Maria Lucia Sampoli, Giovanna Maria Dimitri, A Bargagli, Dario Zanca, Monica Bianchini, D. Muresanu, D. and Alessandra Rufa. 2022. Blinking Rate Comparison Between Patients with Chronic Pain and Parkinson's Disease. In: European Journal of Neurolog, Vol. 29, pp. 669–669.
https://onlinelibrary.wiley.com/doi/epdf/10.1111/ene.15466
Candidate's contributions: supervision, software implementation (in collaboration), original manuscript draft, manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.2.1.

[A6] Giorgia Giacomini, Caterina Graziani, Veronica Lachi, Pietro Bongini, **Niccolò Pancino**, Monica Bianchini, Davide Chiarugi, Angelo Valleriani, and Paolo Andreini. 2022. A Neural Network Approach for the Analysis of Reproducible Ribo–Seq Profiles. Algorithms 15, no. 8: 274.
https://doi.org/10.3390/a15080274
Candidate's contributions: metodology, validation, resurces, manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.2.2.

[A7] Pietro Bongini, Elisa Messori, **Niccolò Pancino**, Monica Bianchini. 2022. A Deep Learning Approach to the Prediction of Drug Side–Effects on Molecular Graphs. Submitted to IEEE/ACM Transactions on Computational Biology and Bioinformatics.

Candidate's contributions: model and algorithm design (in collaboration with Pietro Bongini), software implementation (in collaboration with Pietro Bongini), manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Section 5.4.

Conference Papers

[P1] **Niccolò Pancino**, Alberto Rossi, Giorgio Ciano, Giorgia Giacomini, Simone Bonechi, Paolo Andreini, Franco Scarselli, Monica Bianchini, Pietro Bongini. 2020. Graph Neural Networks for the Prediction of Protein–Protein Interfaces. In: 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2020), 127–132.
Candidate's contributions: conceptualization and discussion, model and algorithm design (in collaboration with Pietro Bongini), software implementation (in collaboration with Pietro Bongini), original manuscript draft, manuscript reviewing and editing.
Thesis relevance: primary contribution, presented in Chapter 6.

[P2] Martina Monaci, **Niccolò Pancino**, Paolo Andreini, Simone Bonechi, Pietro Bongini, Alberto Rossi, Giorgio Ciano, Giorgia Giacomini, Franco Scarselli, Monica Bianchini. (2020). Deep Learning Techniques for Dragonfly Action Recognition. In: Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020), 1, 562–569.
Candidate's contributions: conceptualization and discussion (in collaboration), data analysis (in collaboration), original manuscript draft, manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.3.1.

[P3] Filippo Guerranti, Mirco Mannino, Federica Baccini, Pietro Bongini, **Niccolò Pancino**, Anna Visibelli, Sara Marziali. (2021). Caregiver-Matcher: graph neural networks for connecting caregivers of rare disease patients. In: 25th International Conference on Knowledge–Based and Intelligent Information & Engineering Systems (KES 2021). Procedia Computer Science, 192, 1696–1704.
https://doi.org/10.1016/j.procs.2021.08.174

Candidate's contributions: model and algorithm design (in collaboration), conceptualization and discussion (in collaboration), original manuscript draft (sections on Graph Neural Networks and their application), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.1.

[P4] Paolo Andreini, **Niccolò Pancino**, Filippo Costanti, Gabriele Eusepi, Barbara Toniella Corradini. (2022). A Deep Learning approach for oocytes segmentation and analysis. In: 30th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2022).
https://doi.org/10.14428/esann/2022.ES2022-44
Candidate's contributions: conceptualization and discussion (in collaboration), manuscript reviewing and editing.
Thesis relevance: other works, discussed in Section 7.3.2.

Book Chapters

[C1] Pietro Bongini, **Niccolò Pancino**, Franco Scarselli, Monica Bianchini. (2022). BioGNN: How Graph Neural Networks can solve biological problems. In: Cheng Peng Lim et al. Handbook of Artificial Intelligence in Healthcare. Intelligent Systems Reference Library. Springer Nature Switzerland, Cham, CH.
Candidate's contributions: model and algorithm design (in collaboration), conceptualization and discussion (in collaboration), manuscript reviewing and editing.

Acknowledgements

I would like to express my sincere gratitude to my advisors, Prof. Monica Bianchini and Prof. Franco Scarselli, for their guidance, support, and friendship throughout my Ph.D. journey. Their expertise and dedication have been invaluable in shaping my research and professional development. Their keen insights, constructive feedback, and unwavering encouragement have helped me to overcome obstacles and achieve my goals. Their mentorship has been a source of inspiration, and I am grateful for the opportunity to learn from them and I am honored to consider them not only as my advisors but also as my friends.

I would like to extend my sincere thanks to Prof. Pietro Liò for his valuable collaboration and support during my stay at the University of Cambridge. His expertise and insights have greatly contributed to my research and I am grateful for the opportunity to work alongside him.

I would like to thank Prof. Claudio Gallicchio and Prof. Stefano Cagnoni for being part of my Supervisory Committee: I would like to express my appreciation for their time, effort and dedication as well as their suggestions and opinions, which have been fundamental to the success of my research.

I am grateful to the institutions which have enabled me to pursue these studies. Specifically, I would like to thank the Ph.D. program in Smart Computing, which is jointly funded by the Universities of Florence, Siena, and Pisa, and also by Regione Toscana through the European Social Fund. I also want to express my thanks to the University of Florence, the host institution of the Ph.D program, as well as the University of Siena, particularly the Department of Information Engineering and Mathematics, where I completed my degrees and conducted most of my research.

A special thank you goes to all my fellow Ph.D. students at the Siena Artificial Intelligence Laboratory (SAILab). I would risk not delivering my thesis on time if I were to mention each and every one of them individually —

I swear I will do it separately, do not worry —, but I want them to know that each of them has made these last three years unforgettable: in the Lab202 I found authentic friendships and a sense of belonging and brotherhood that I never expected to find in such a place. Special mentions to my friends Pietro Bongini and Alessio D'Inverno, with whom I shared efforts in every work we did together, trying to make everything work, and great laughs both inside and outside the Lab and to Giovanna Maria Dimitri and Enrico Meloni, who always supported me even when I was thousands of kilometers away.

I would like to extend a heartfelt thank you to my dear friends, Michele, the VH group, the Hoppians, the UPD parishioners, as well as the newbies who came in my life in the last year: the hours spent talking to you, together with your laughter and good humor have brightened even the darkest of days. I am so grateful to have you all in my life and I look forward to many more years of friendship and fun. Thank you for being there for me, and for being such an important part of my life.

Finally, my deepest gratitude goes to my family for their endless support and love throughout my journey. My parents, Paolo and Patrizia, have been my constant source of inspiration and encouragement. Their love and guidance have been invaluable to me and I am eternally grateful for all that they have done for me. My sisters, Ottavia, Gioia and Michela, together with Manfredi and Alessandro, have always been my confidants and best friends. Their love and support have been a source of strength for me. I am also grateful to my grandmother for her prayers and for always being there for me, and my nephews Chiara e Francesco for the joy they brought in my life. I am blessed to have such a supportive and loving family, and I could not have achieved this milestone without their love and support. I dedicate this achievement to them, with all my love and gratitude.

Bibliography

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[2] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[3] M. Wang and G. Hu, “A novel method for twitter sentiment analysis based on attentional–graph neural network,” *Information*, vol. 11, no. 2, p. 92, 2020.

[4] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, “Hierarchical representation learning in graph neural networks with node decimation pooling,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[5] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett, “Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm,” *Journal of molecular biology*, vol. 229, no. 3, pp. 707–721, 1993.

[6] E. J. Gardiner, P. J. Artymiuk, and P. Willett, “Clique–detection algorithms for matching three-dimensional molecular structures,” *Journal of Molecular Graphics and Modelling*, vol. 15, no. 4, pp. 245–253, 1997.

[7] M. Zitnik, M. Agrawal, and J. Leskovec, “Modeling polypharmacy side effects with graph convolutional networks,” *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[8] J. Kim, S. Park, D. Min, and W. Kim, “Comprehensive survey of recent drug discovery using deep learning,” *International Journal of Molecular Sciences*, vol. 22, no. 18, p. 9983, 2021.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2009.

[10] T. N. Kipf and M. Welling, "Semi–supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, pp. 3844–3852, 2016.

[13] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, pp. 1024–1034, 2017.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017.

[15] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[16] Y. Wang, KrisKitani, and X. Weng, "Joint object detection and multi-object tracking with graph neural networks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13708–13715, IEEE, 2021.

[17] A. Luo, X. Li, Fan Yang, Z. Jiao, H. Cheng, and SiweiLyu, "Cascade graph neural networks for rgb-d salient object detection," in *European Conference on Computer Vision*, pp. 346–364, Springer, 2020.

[18] WeijingShi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1711–1719, 2020.

[19] Z. Wang, W. Wei, G. Cong, X.-L. Li, X.-L. Mao, and M. Qiu, "Global context enhanced graph neural networks for session-based recommendation," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pp. 169–178, 2020.

[20] F. Guerranti, M. Mannino, F. Baccini, P. Bongini, N. Pancino, A. Visibelli, and S. Marziali, "Caregivermatcher: graph neural networks for connecting caregivers of rare disease patients," *Procedia Computer Science*, vol. 192, pp. 1696–1704, 2021.

[21] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "Fedgnn: Federated graph neural network for privacy-preserving recommendation," *arXiv preprint arXiv:2102.04925*, 2021.

[22] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1666–1674, 2018.

[23] N. Pancino, A. Rossi, G. Ciano, G. Giacomini, S. Bonechi, P. Andreini, F. Scarselli, M. Bianchini, and P. Bongini, "Graph neural networks for the prediction of protein–protein interfaces," in *ESANN*, pp. 127–132, 2020.

[24] P. Bongini, N. Pancino, F. Scarselli, and M. Bianchini, "Biognn: How graph neural networks can solve biological problems," in *Artificial Intelligence and Machine Learning for Healthcare*, pp. 211–231, Springer, 2023.

[25] J. S. Smith, A. E. Roitberg, and O. Isayev, "Transforming computational drug discovery with machine learning and ai," *ACS medicinal chemistry letters*, vol. 9, no. 11, pp. 1065–1069, 2018.

[26] P. Bongini, M. Bianchini, and F. Scarselli, "Molecular generative graph neural networks for drug discovery," *Neurocomputing*, vol. 450, pp. 242–252, 2021.

[27] E. Dai and S. Wang, "Towards self–explainable graph neural network," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 302–311, 2021.

[28] Z. Zhong, C.-T. Li, and J. Pang, "Hierarchical message-passing graph neural networks," 2020.

[29] N. Pancino, P. Bongini, F. Scarselli, and M. Bianchini, "Gnnkeras: A keras-based library for graph neural networks and homogeneous and heterogeneous graph processing," *SoftwareX*, vol. 18, p. 101061, 2022.

[30] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[31] A. Mathison Turing, "Computing machinery and intelligence," *The Turing Test: Verbal Behavior as the Hallmark of Intelligence*, pp. 29–56, 1950.

[32] A. L. Hodgkin and A. F. Huxley, "Propagation of electrical signals along giant nerve fibres," *Proceedings of the Royal Society of London. Series B-Biological Sciences*, vol. 140, no. 899, pp. 177–183, 1952.

[33] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[35] C. Cortes and V. Vapnik, "Support–vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] Y. Bengio, P. Simard, and P. Frasconi, "Learning long–term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[39] T. Kohonen, "Self–organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[40] J. A. Davis, “Clustering and structural balance in graphs,” *Human relations*, vol. 20, no. 2, pp. 181–187, 1967.

[41] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[42] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

[43] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[44] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[45] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[46] K. Fukushima, “Cognitron: A self-organizing multilayered neural network,” *Biological cybernetics*, vol. 20, no. 3, pp. 121–136, 1975.

[47] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[48] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[49] S. Hochreiter and J. Schmidhuber, “Long short–term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[50] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder–decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[51] K. M. Tarwani and S. Edem, “Survey on recurrent neural network in natural language processing,” *Int. J. Eng. Trends Technol*, vol. 48, pp. 301–304, 2017.

[52] S. K. Sønderby and O. Winther, "Protein secondary structure prediction with long short term memory networks," 2014.

[53] A. Visibelli, P. Bongini, A. Rossi, N. Niccolai, and M. Bianchini, "A deep attention network for predicting amino acid signals in the formation of α-helices," *Journal of Bioinformatics and Computational Biology*, vol. 18, no. 05, p. 2050028, 2020.

[54] M. Weber, M. Liwicki, D. Stricker, C. Scholzel, and S. Uchida, "Lstm–based early recognition of motion patterns," in *2014 22nd International Conference on Pattern Recognition*, pp. 3552–3557, IEEE, 2014.

[55] K. Pawar, R. S. Jalem, and V. Tiwari, "Stock market price prediction using lstm rnn," in *Emerging Trends in Expert Applications and Security*, pp. 493–503, Springer, 2019.

[56] A. Graves, N. Jaitly, and A. rahman Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*, pp. 273–278, IEEE, 2013.

[57] J. Zhao, X. Mao, and L. Chen, "Speech emotion recognition using deep 1d & 2d cnn lstm networks," *Biomedical Signal Processing and Control*, vol. 47, pp. 312–323, 2019.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[59] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[61] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[62] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[63] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[64] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[66] D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs," 2002.

[67] D. Haussler, "Convolution kernels on discrete structures," tech. rep., Technical report, Department of Computer Science, University of California, 1999.

[68] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, "Graph kernels for chemical informatics," *Neural networks*, vol. 18, no. 8, pp. 1093–1110, 2005.

[69] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.

[70] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.

[71] F. Scarselli, S. L. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pp. 666–672, IEEE, 2005.

[72] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[73] N. Bandinelli, M. Bianchini, and F. Scarselli, "Learning long–term dependencies using layered graph neural networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2010.

[74] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep–but not shallow–networks avoid the curse of dimensionality: a review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, 2017.

[75] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2018.

[76] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*, 2020.

[77] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI Series*, vol. 2, no. 9, pp. 12–16, 1968.

[78] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4602–4609, 2019.

[79] G. A. D'Inverno, M. Bianchini, M. L. Sampoli, and F. Scarselli, "An unifying point of view on expressive power of gnns," 2021.

[80] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[81] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess,

D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, “Relational inductive biases, deep learning, and graph networks,” 2018.

[82] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” 2015.

[83] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1263–1272, 2017.

[84] S. Munikoti, L. Das, and B. Natarajan, “Scalable graph neural network-based framework for identifying critical nodes and links in complex networks,” *Neurocomputing*, vol. 468, pp. 211–221, 2022.

[85] F. Scarselli, A. C. Tsoi, M. Hagenbuchner, and L. Di Noi, “Solving graph data issues using a layered architecture approach with applications to web spam detection,” *Neural Networks*, vol. 48, pp. 78–90, 2013.

[86] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, “Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 539–548, 2019.

[87] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, “Open graph benchmark: Datasets for machine learning on graphs,” 2020.

[88] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, “Neural graph collaborative filtering,” in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 165–174, 2019.

[89] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, “Kgat: Knowledge graph attention network for recommendation,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 950–958, 2019.

[90] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, pp. 2022–2032, 2019.

[91] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International conference on machine learning*, pp. 3835–3845, PMLR, 2019.

[92] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, "Spatiotemporal graph neural network for performance prediction of photovoltaic power systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15323–15330, 2021.

[93] B. Donon, B. Donnot, I. Guyon, and A. Marot, "Graph neural solver for power systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.

[94] G. Bécigneul, O.-E. Ganea, B. Chen, R. Barzilay, and T. Jaakkola, "Optimal transport graph neural networks," 2020.

[95] B. Wu, Y. Liu, B. Lang, and L. Huang, "Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model," *Neurocomputing*, vol. 321, pp. 346–356, 2018.

[96] O. J. Wouters, M. McKee, and J. Luyten, "Estimated research and development investment needed to bring a new medicine to market, 2009-2018," *Jama*, vol. 323, no. 9, pp. 844–853, 2020.

[97] M. L. Billingsley, "Druggable targets and targeted drugs: Enhancing the development of new therapeutics," *Pharmacology*, vol. 82, no. 4, pp. 239–244, 2008.

[98] M. Dickson and J. P. Gagnon, "Key factors in the rising cost of new drug discovery and development," *Nature reviews Drug discovery*, vol. 3, no. 5, pp. 417–429, 2004.

[99] C. F. Lipinski, V. G. Maltarollo, P. R. Oliveira, A. B. da Silva, and K. M. Honorio, "Advances and perspectives in applying deep learning for drug design and discovery," *Frontiers in Robotics and AI*, vol. 6, p. 108, 2019.

[100] S. J. Y. Macalino, V. Gosu, S. Hong, and S. Choi, "Role of computer-aided drug design in modern drug discovery," *Archives of pharmacal research*, vol. 38, no. 9, pp. 1686–1701, 2015.

[101] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[102] P. Bongini, A. Trezza, M. Bianchini, O. Spiga, and N. Niccolai, "A possible strategy to fight COVID-19: Interfering with spike glycoprotein trimerization," *Biochem Biophys Res Commun.*, vol. 528(1), pp. 35–38, 2020.

[103] P. Bongini, N. Pancino, G. M. Dimitri, M. Bianchini, F. Scarselli, and P. Lio, "Modular multi–source prediction of drug side–effects with drugnn," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–1, 2022.

[104] A. Zhavoronkov, Y. A. Ivanenkov, A. Aliper, M. S. Veselov, V. A. Aladinskiy, A. V. Aladinskaya, V. A. Terentiev, D. A. Polykovskiy, M. D. Kuznetsov, A. Asadulaev, *et al.*, "Deep learning enables rapid identification of potent ddr1 kinase inhibitors," *Nature biotechnology*, vol. 37, no. 9, pp. 1038–1040, 2019.

[105] Q. Feng, E. Dueva, A. Cherkasov, and M. Ester, "Padme: A deep learning-based framework for drug-target interaction prediction," 2018.

[106] T. B. Kimber, Y. Chen, and A. Volkamer, "Deep learning in virtual screening: Recent applications and developments," *International Journal of Molecular Sciences*, vol. 22, no. 9, p. 4435, 2021.

[107] M. Skalic, A. Varela-Rial, J. Jiménez, G. Martínez-Rosell, and G. De Fabritiis, "Ligvoxel: inpainting binding pockets using 3d-convolutional neural networks," *Bioinformatics*, vol. 35, no. 2, pp. 243–250, 2019.

[108] P. Bongini, N. Niccolai, and M. Bianchini, "Glycine–induced formation and druggability score prediction of protein surface pockets," *Journal of bioinformatics and computational biology*, vol. 17, no. 05, p. 1950026, 2019.

[109] X. Zeng, S. Zhu, W. Lu, Z. Liu, J. Huang, Y. Zhou, J. Fang, Y. Huang, H. Guo, L. Li, *et al.*, "Target identification among known drugs by deep learning from heterogeneous networks," *Chemical Science*, vol. 11, no. 7, pp. 1775–1797, 2020.

[110] Y. Yamanishi, M. Kotera, M. Kanehisa, and S. Goto, "Drug–target interaction prediction from chemical, genomic and pharmacological data in an integrated framework," *Bioinformatics*, vol. 26, no. 12, pp. i246–i254, 2010.

[111] S. Mizutani, E. Pauwels, V. Stoven, S. Goto, and Y. Yamanishi, "Relating drug–protein interaction network with drug side effects," *Bioinformatics*, vol. 28, no. 18, pp. i522–i528, 2012.

[112] E. Pauwels, V. Stoven, and Y. Yamanishi, "Predicting drug side–effect profiles: A chemical fragment–based approach," *BMC bioinformatics*, vol. 12, no. 1, pp. 1–13, 2011.

[113] A. Mohsen, L. P. Tripathi, and K. Mizuguchi, "Deep learning prediction of adverse drug reactions in drug discovery using open tg–gates and faers databases," *Frontiers in Drug Discovery*, vol. 1, 2021.

[114] W. Zhang, Y. Chen, S. Tu, F. Liu, and Q. Qu, "Drug side effect prediction through linear neighborhoods and multiple data source integration," in *2016 IEEE international conference on bioinformatics and biomedicine (BIBM)*, pp. 427–434, IEEE, 2016.

[115] I. Shaked, M. A. Oberhardt, N. Atias, R. Sharan, and E. Ruppin, "Metabolic network prediction of drug side effects," *Cell systems*, vol. 2, no. 3, pp. 209–213, 2016.

[116] G. M. Dimitri and P. Liò, "DrugClust: A machine learning approach for drugs side effects prediction," *Computational biology and chemistry*, vol. 68, pp. 204–210, 2017.

[117] A. Cakir, M. Tuncer, H. Taymaz-Nikerel, and O. Ulucan, "Side effect prediction based on drug–induced gene expression profiles and random forest with iterative feature selection," *The Pharmacogenomics Journal*, pp. 1–9, 2021.

[118] O. C. Uner, R. G. Cinbis, O. Tastan, and A. E. Cicek, “DeepSide: A deep learning framework for drug side effect prediction,” 2019.

[119] T. Huynh, Y. He, A. Willis, and S. Rueger, “Adverse drug reaction classification with deep neural networks,” in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, (Osaka, Japan), pp. 877–887, The COLING 2016 Organizing Committee, dec 2016.

[120] A. F. Fliri, W. T. Loging, P. F. Thadeio, and R. A. Volkmann, “Analysis of drug-induced effect patterns to link structure and side effects of medicines,” *Nature chemical biology*, vol. 1, no. 7, pp. 389–397, 2005.

[121] A. Deac, Y.-H. Huang, P. Veličković, P. Liò, and J. Tang, “Drug–drug adverse effect prediction with graph co–attention,” 2019.

[122] S. Bang, J. H. Jhee, and H. Shin, “Polypharmacy side-effect prediction with enhanced interpretability based on graph feature attention network,” *Bioinformatics*, vol. 37, no. 18, pp. 2955–2962, 2021.

[123] M. Timilsina, M. Tandan, M. d’Aquin, and H. Yang, “Discovering links between side effects and drugs using a diffusion based method,” *Scientific reports*, vol. 9, no. 1, p. 10436, 2019.

[124] A. K. Menon and C. Elkan, “Link prediction via matrix factorization,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II 22*, pp. 437–452, Springer, 2011.

[125] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding,” in *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015.

[126] S. Sra and I. Dhillon, “Generalized nonnegative matrix approximations with bregman divergences,” *Advances in neural information processing systems*, vol. 18, 2005.

[127] B. Chen, F. Li, S. Chen, R. Hu, and L. Chen, “Link prediction based on non-negative matrix factorization,” *PloS one*, vol. 12, no. 8, p. e0182968, 2017.

[128] L. Yu, M. Cheng, W. Qiu, X. Xiao, and W. Lin, "idse-he: Hybrid embedding graph neural network for drug side effects prediction," *Journal of Biomedical Informatics*, vol. 131, p. 104098, 2022.

[129] J. Janin, R. P. Bahadur, and P. Chakrabarti, "Protein–protein interaction and quaternary structure," *Quarterly reviews of biophysics*, vol. 41, no. 2, pp. 133–180, 2008.

[130] T. J. Lane, D. Shukla, K. A. Beauchamp, and V. S. Pande, "To milliseconds and beyond: challenges in the simulation of protein folding," *Current opinion in structural biology*, vol. 23, no. 1, pp. 58–65, 2013.

[131] L. C. Xue, D. Dobbs, and V. Honavar, "Homppi: a class of sequence homology based protein-protein interface prediction methods," *BMC bioinformatics*, vol. 12, no. 1, pp. 1–24, 2011.

[132] H. Hwang, D. Petrey, and B. Honig, "A hybrid method for protein–protein interface prediction," *Protein Science*, vol. 25, no. 1, pp. 159–165, 2016.

[133] H. Hwang, T. Vreven, and Z. Weng, "Binding interface prediction by combining protein–protein docking results," *Proteins: Structure, Function, and Bioinformatics*, vol. 82, no. 1, pp. 57–66, 2014.

[134] J. R. Bradford and D. R. Westhead, "Improved prediction of protein–protein binding sites using a support vector machines approach," *Bioinformatics*, vol. 21, no. 8, pp. 1487–1494, 2005.

[135] K. Huang, C. Xiao, L. M. Glass, M. Zitnik, and J. Sun, "Skipgnn: predicting molecular interactions with skip-graph networks," *Scientific reports*, vol. 10, no. 1, pp. 1–16, 2020.

[136] Y. Liu, H. Yuan, L. Cai, and S. Ji, "Deep learning of high–order interactions for protein interface prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 679–687, 2020.

[137] M. Réau, N. Renaud, L. C. Xue, and A. M. J. J. Bonvin, "DeepRank-GNN: a graph neural network framework to learn patterns in protein–protein interfaces," *Bioinformatics*, vol. 39, 11 2022. btac759.

[138] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734, IEEE, 2005.

[139] A. Rossi, M. Tiezzi, G. M. Dimitri, M. Bianchini, M. Maggini, and F. Scarselli, "Inductive–transductive learning with graph neural networks," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 201–212, Springer, 2018.

[140] G. Ciano, A. Rossi, M. Bianchini, and F. Scarselli, "On inductive–transductive learning with graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[141] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

[142] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[143] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[144] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral [application notes]," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 99–106, 2021.

[145] F. R. Ernst and A. J. Grizzle, "Drug–related morbidity and mortality: Updating the cost–of–illness model," *Journal of the American Pharmaceutical Association*, vol. 41, no. 2, pp. 192–199, 2001.

[146] H. Khalil and C. Huang, "Adverse drug reactions in primary care: A scoping review," *BMC health services research*, vol. 20, no. 1, pp. 1–13, 2020.

[147] K. M. Giacomini, R. M. Krauss, D. M. Roden, M. Eichelbaum, M. R. Hayden, and Y. Nakamura, “When good drugs go bad,” *Nature*, vol. 446, no. 7139, pp. 975–977, 2007.

[148] E. D. Kantor, C. D. Rehm, J. S. Haas, A. T. Chan, and E. L. Giovannucci, “Trends in prescription drug use among adults in the United States from 1999–2012,” *Jama*, vol. 314, no. 17, pp. 1818–1830, 2015.

[149] N. P. Tatonetti, T. Liu, and R. B. Altman, “Predicting drug side-effects by chemical systems biology,” *Genome biology*, vol. 10, no. 9, pp. 1–4, 2009.

[150] A. Lavecchia and C. Di Giovanni, “Virtual screening strategies in drug discovery: a critical review,” *Current medicinal chemistry*, vol. 20, no. 23, pp. 2839–2860, 2013.

[151] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, *et al.*, “PubChem in 2021: New data content and improved web interfaces,” *Nucleic acids research*, vol. 49, no. D1, pp. D1388–D1395, 2021.

[152] F. Cunningham, J. E. Allen, J. Allen, J. Alvarez-Jarreta, M. Amode, I. Armean, O. Austine-Orimoloye, A. Azov, I. Barnes, R. Bennett, A. Berry, J. Bhai, A. Bignell, K. Billis, S. Boddu, L. Brooks, M. Charkhchi, C. Cummins, L. Da Rin Fioretto, C. Davidson, K. Dodiya, S. Donaldson, B. El Houdaigui, T. El Naboulsi, R. Fatima, C. G. Giron, T. Genez, J. Martinez, C. Guijarro-Clarke, A. Gymer, M. Hardy, Z. Hollis, T. Hourlier, T. Hunt, T. Juettemann, V. Kaikala, M. Kay, I. Lavidas, T. Le, D. Lemos, J. C. Marugán, S. Mohanan, A. Mushtaq, M. Naven, D. Ogeh, A. Parker, A. Parton, M. Perry, I. Piližota, I. Prosovetskaia, M. Sakthivel, A. Salam, B. Schmitt, H. Schuilenburg, D. Sheppard, J. Pérez-Silva, W. Stark, E. Steed, K. Sutinen, R. Sukumaran, D. Sumathipala, M.-M. Suner, M. Szpak, A. Thormann, F. F. Tricomi, D. Urbina-Gómez, A. Veidenberg, T. Walsh, B. Walts, N. Willhoft, A. Winterbottom, E. Wass, M. Chakiachvili, B. Flint, A. Frankish, S. Giorgetti, L. Haggerty, S. Hunt, G. IIsley, J. Loveland, F. Martin, B. Moore, J. Mudge, M. Muffato, E. Perry, M. Ruffier, J. Tate, D. Thybert, S. Trevanion, S. Dyer, P. Harrison,

K. Howe, A. Yates, D. Zerbino, and P. Flicek, “Ensembl 2022,” *Nucleic Acids Research*, vol. 50, pp. D988–D995, 11 2021.

[153] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, *et al.*, “Gene ontology: Tool for the unification of biology,” *Nature genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[154] D. Szklarczyk, A. Santos, C. Von Mering, L. J. Jensen, P. Bork, and M. Kuhn, “STITCH 5: Augmenting protein–chemical interaction networks with tissue and affinity data,” *Nucleic acids research*, vol. 44, no. D1, pp. D380–D384, 2016.

[155] K. Luck, D.-K. Kim, L. Lambourne, K. Spirohn, B. E. Begg, W. Bian, R. Brignall, T. Cafarelli, F. J. Campos-Laborie, B. Charloteaux, *et al.*, “A reference map of the human binary protein interactome,” *Nature*, vol. 580, no. 7803, pp. 402–408, 2020.

[156] M. Kuhn, I. Letunic, L. J. Jensen, and P. Bork, “The SIDER database of drugs and side effects,” *Nucleic acids research*, vol. 44, no. D1, pp. D1075–D1079, 2016.

[157] D. Smedley, S. Haider, S. Durinck, L. Pandini, P. Provero, J. Allen, O. Arnaiz, M. H. Awedh, R. Baldock, G. Barbiera, *et al.*, “The BioMart community portal: An innovative alternative to large, centralized data repositories,” *Nucleic acids research*, vol. 43, no. W1, pp. W589–W598, 2015.

[158] D. W. Huang, B. T. Sherman, and R. A. Lempicki, “Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources,” *Nature protocols*, vol. 4, no. 1, pp. 44–57, 2009.

[159] D. W. Huang, B. T. Sherman, and R. A. Lempicki, “Bioinformatics enrichment tools: Paths toward the comprehensive functional analysis of large gene lists,” *Nucleic acids research*, vol. 37, no. 1, pp. 1–13, 2009.

[160] G. Landrum, “rdkit/rdkit: 2020_03_1 (q1 2020) release,” mar 2020.

[161] T. T. Tanimoto, “IBM internal report 17th,” tech. rep., IBM, 11 1957.

[162] A. Hagberg, P. Swart, and D. S. Chult, "Exploring network structure, dynamics, and function using networkX," in *In Proceedings of the 7th Python in Science Conference (SciPy)* (J. M. G. Varoquaux, T. Vaught, ed.), pp. 11–15, 2008.

[163] N. Pancino, Y. Perron, P. Bongini, and F. Scarselli, "Drug side effect prediction with deep learning molecular embedding in a graph-of-graphs domain," *Mathematics*, vol. 10, no. 23, p. 4550, 2022.

[164] J. Menke and O. Koch, "Using domain-specific fingerprints generated through neural networks to enhance ligand-based virtual screening," *Journal of Chemical Information and Modeling*, vol. 61, no. 2, pp. 664–675, 2021.

[165] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, 2015.

[166] H. Wang, D. Lian, Y. Zhang, L. Qin, and X. Lin, "Gognn: Graph of graphs neural network for predicting structured entity interactions," *arXiv preprint arXiv:2005.05537*, 2020.

[167] ShonosukeHarada, H. Akita, M. Tsubaki, Y. Baba, I. Takigawa, Y. Yamanishi, and H. Kashima, "Dual convolutional neural network for graph of graphs link prediction," *arXiv preprint arXiv:1810.02080*, 2018.

[168] H. Wang, D. Lian, W. Liu, D. Wen, C. Chen, and X. Wang, "Powerful graph of graphs neural network for structured entity analysis," *World Wide Web*, vol. 25, no. 2, pp. 609–629, 2022.

[169] Y. Wang, Y. Zhao, N. Shah, and T. Derr, "Imbalanced graph classification via graph-of-graph neural networks," *arXiv preprint arXiv:2112.00238*, 2021.

[170] P. Bongini, E. Messori, N. Pancino, and M. Bianchini, "A deep learning approach to the prediction of drug side-effects on molecular graphs," *arXiv preprint arXiv:2211.16871*, 2022.

[171] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[172] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. abs/1708.02002, pp. 2999–3007, 2017.

[173] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks*, pp. 412–422, Springer, 2018.

[174] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *International conference on machine learning*, pp. 2323–2332, PMLR, 2018.

[175] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nature chemistry*, vol. 4, no. 2, pp. 90–98, 2012.

[176] H. Hegyi and M. Gerstein, "The relationship between protein structure and function: a comprehensive survey with application to the yeast genome," *Journal of molecular biology*, vol. 288, no. 1, pp. 147–164, 1999.

[177] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of combinatorial optimization*, pp. 1–74, Springer, 1999.

[178] E. Krissinel, "Crystal contacts as nature's docking solutions," *Journal of computational chemistry*, vol. 31, no. 1, pp. 133–143, 2010.

[179] T. Schäfer, P. May, and I. Koch, "Computation and visualization of protein topology graphs including ligand information," in *German Conference on Bioinformatics 2012*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[180] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[181] W. Kabsch and C. Sander, “Dictionary of protein secondary structure: Pattern recognition of hydrogen–bonded and geometrical features,” *Biopolymers*, vol. 22, no. 12, pp. 2577–2637, 1983.

[182] J. Kyte and R. F. Doolittle, “A simple method for displaying the hydropathic character of a protein,” *Journal of molecular biology*, vol. 157, no. 1, pp. 105–132, 1982.

[183] C. Bron and J. Kerbosch, “Algorithm 457: finding all cliques of an undirected graph,” *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[184] S. N. Wakap, D. M. Lambert, A. Olry, C. Rodwell, C. Gueydan, V. Lanneau, D. Murphy, Y. Le Cam, and A. Rath, “Estimating cumulative point prevalence of rare diseases: analysis of the orphanet database,” *European Journal of Human Genetics*, vol. 28, no. 2, pp. 165–173, 2020.

[185] M. Navaie-Waliser, P. H. Feldman, D. A. Gould, C. Levine, A. N. Kuerbis, and K. Donelan, “When the caregiver needs care: The plight of vulnerable caregivers,” *American journal of public health*, vol. 92, no. 3, pp. 409–413, 2002.

[186] E. Palamaro Munsell, R. P. Kilmer, J. R. Cook, and C. L. Reeve, “The effects of caregiver social connections on caregiver, child, and family well-being.,” *American Journal of Orthopsychiatry*, vol. 82, no. 1, p. 137, 2012.

[187] L.-Y. Chien, H. Chu, J.-L. Guo, Y.-M. Liao, L.-I. Chang, C.-H. Chen, and K.-R. Chou, “Caregiver support groups in patients with dementia: a meta-analysis,” *International journal of geriatric psychiatry*, vol. 26, no. 10, pp. 1089–1098, 2011.

[188] N. Pancino, C. Graziani, V. Lachi, M. L. Sampoli, E. Ștefănescu, M. Bianchini, and G. M. Dimitri, “A mixed statistical and machine learning approach for the analysis of multimodal trail making test data,” *Mathematics*, vol. 9, no. 24, p. 3159, 2021.

[189] E. Ștefănescu, N. Pancino, C. Graziani, V. Lachi, M. L. Sampoli, G. M. Dimitri, A. Bargagli, D. Zanca, M. Bianchini, D. F. Muresanu, *et al.*,

"Blinking rate comparison between patients with chronic pain and parkinson's disease," in *EUROPEAN JOURNAL OF NEUROLOGY*, vol. 29, pp. 669–669, WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, 2022.

[190] R. Kredel, C. Vater, A. Klostermann, and E.-J. Hossner, "Eye-tracking technology and the dynamics of natural gaze behavior in sports: A systematic review of 40 years of research," *Frontiers in psychology*, vol. 8, p. 1845, 2017.

[191] William H Kruskal and A. W. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[192] A. Valleriani and D. Chiarugi, "A workbench for the translational control of gene expression," *bioRxiv*, 2020.

[193] G. Giacomini, C. Graziani, V. Lachi, P. Bongini, N. Pancino, M. Bianchini, D. Chiarugi, A. Valleriani, and P. Andreini, "A neural network approach for the analysis of reproducible ribo–seq profiles," *Algorithms*, vol. 15, no. 8, p. 274, 2022.

[194] M. Monaci, N. Pancino, P. Andreini, S. Bonechi, P. Bongini, A. Rossi, G. Ciano, G. Giacomini, F. Scarselli, and M. Bianchini, "Deep learning techniques for dragonfly action recognition.," in *ICPRAM*, pp. 562–569, 2020.

[195] P. Andreini, N. Pancino, F. Costanti, G. Eusepi, and B. T. Corradini, "A deep learning approach for oocytes segmentation and analysis,"

[196] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[197] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[198] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[199] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[200] S. Bonechi, M. Bianchini, F. Scarselli, and P. Andreini, "Weak supervision for generating pixel–level annotations in scene text segmentation," *Pattern Recognition Letters*, vol. 138, pp. 1–7, 2020.

[201] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[202] D. Rigoni, N. Navarin, and A. Sperduti, "Conditional constrained graph variational autoencoders for molecule design," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 729–736, IEEE, 2020.

[203] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," 2018.

[204] H. A. Hussein, A. Borrel, C. Geneix, M. Petitjean, L. Regad, and A.-C. Camproux, "Pockdrug–server: a new web server for predicting pocket druggability on holo and apo proteins," *Nucleic acids research*, vol. 43, no. W1, pp. W436–W442, 2015.