Drug Target Interaction Prediction

A project submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
In
INFORMATION TECHNOLOGY

Submitted by:

Mr Parimi Vignesh and Mr Ginjupalli Srikanth

Roll No : 12012025, 12012042

Supervised by:

Dr. Ramesh Saha

Assistant Professor

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
SONEPAT -131021, HARYANA, INDIA

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. I would like to thank Prof **Dr. Ramesh Saha**, Assistant Professor - IT, and IIIT Sonepat for allowing me to undertake & helping me complete this project. Also, I would like to thank my group member (Parimi Vignesh), and also my batchmates who guided me, helped me and gave ideas and motivation at each step.

Ginjupalli Srikanth

SELF DECLARATION

I hereby declare that the work contained in the project file titled “Drug Target Interaction Prediction” is original. I have followed the standards of research/project ethics to the best of my abilities. I have acknowledged all sources of information that I have used in this project.

Name: Parimi Vignesh , Ginjupalli Srikanth
Roll No.: 12012025, 12012042
Department of Information Technology
Indian Institute of Information Technology,
Sonepat-131201, Haryana, India.

CERTIFICATE

This is to certify that **Mr Ginjupalli Srikanth** has worked on the project entitled "**Drug Target Interaction Prediction**" under my supervision and guidance. The contents of the project, being submitted to the **Department of Information Technology**, **IIIT Sonepat,** for the award of the degree of **B.Tech** in **Information Technology**, are original and have been carried out by the candidate himself. This project has not been submitted in full or part for the award of any other degree or diploma to this or any other university.

Dr Ramesh Saha
Supervisor
Department of Information Technology
Indian Institute of Information Technology,
Sonepat-131201, Haryana, India

ABSTRACT

Name of the student: **Ginjupalli Srikanth**

Roll No.: **12012042**

Degree for which submitted: **B.Tech (IT)**

Department of **Information Technology**, **IIIT Sonepat**.

Project Title: **Drug Target Interaction Prediction**

Name of the thesis supervisor: **Dr. Ramesh Saha**

Month and year of the thesis submission: **December 2023**

In this study, we present a novel approach using graph convolutional neural networks (GCNNs) for the prediction of drug-target interactions accurately which is a critical aspect to find the binding affinity in drug discovery process. We have inputted the features in the form of graphs into the network architecture that follows hierarchical splitting. Here, we included important features of drug smiles like chirality, aromaticity, formal charge which helps to achieve better results in predicting drug-target interactions. For protein structures, we can visualize the features using a list of size 20 containing only the zeros and one based on enumerating alphabets of protein sequences. In the evolving era of deep learning, neural networks have played an important role in improving the research area of drug-target interactions. Graph convolutional neural networks help to handle large amounts of data easily which is essential for the exploration in the field of bioinformatics. We will also examine the future possibilities of graph convolutional neural networks in predicting drug-target interactions.

Table Of Contents

1. INTRODUCTION

It is very important to find out the accurate affinity of drug-target interactions. Lots of difficulties arise when we experiment practically with different types of drugs to predict the suitable interactions. This eats up a lot of investment, time and is difficult to test it on humans. Using machine learning and deep learning techniques have improved the space for discovery of various drugs for the ligands without testing them practically on humans. These computation approaches are developing significantly after the evolution of procedures of artificial intelligence. Several properties of certified drugs can be discovered for unexpected targets. Also, these advanced methods help us in easy determination of drug-target pairs when we want to explore properties of a newly discovered drug.

The enlargement in size of drug and protein datasets has increased the scope of finding out drugs through these computer-controlled methods. These datasets contain the information of drugs and proteins along with labels which helps in construction of models that determine the properties of new drugs. The machine and deep learning models have produced some encouraging results. However, quality of the dataset is particularly important while working with these methods. The success of these deep learning models hugely rely on the same factor.

Graph convolutional neural networks help us to deal with large data when compared to some of the other techniques like support vector machines and random forest. Here, data is sent to the layers of the network architecture in the form of batches. The features are then subjected to go through filters in a hierarchical manner. The accuracy of the output also depends on the number of epochs. However, repeated training of the data may affect the results as it may lead to overfitting. We can analyze the features of drugs and proteins by representing them in the form of dictionary objects without using the inbuilt functions that are difficult to visualize novel features.

2. LITERATURE SURVEY

2.1 Existing Problem :

The interaction between drugs and proteins is a complex process that can give rise to various challenges in drug development and therapeutic applications. Here are some existing problems in the context of drug interaction with proteins:

Drugs commonly interact with unintended protein targets, resulting in off-target effects. Predicting and mitigating off-target interactions is a significant challenge in drug development.

Incomplete knowledge of protein functions and pathways can result in unexpected interactions. Understanding the full spectrum of a protein's functions is crucial for predicting and avoiding adverse effects.

Proteins can adopt multiple conformational states, and their structures can change dynamically. Predicting drug interactions with proteins in different structural states is a challenging task.

Accurately predicting the binding affinity between drugs and proteins is challenging. Computational methods often face difficulties in precisely quantifying the strength of drug-protein interactions.

Many drugs interact with multiple proteins, leading to polypharmacology. Predicting and managing these interactions to optimize therapeutic outcomes while minimizing side effects is a complex problem.

Identifying rare or novel drug-protein interactions is challenging due to limited data. Discovering and characterizing these interactions are important for understanding potential side effects and improving drug safety.

Interactions between drugs and proteins can be context-dependent, influenced by factors such as tissue-specific expression and cellular microenvironments. Predicting context-specific interactions is a key challenge.

The development of drug resistance is a significant problem in cancer and infectious disease treatment. Understanding the mechanisms of resistance and predicting how drugs interact with mutated proteins is critical for overcoming this challenge.

Post-translational modifications (PTMs) can alter protein structures and functions. Predicting how drugs interact with proteins undergoing PTMs adds another layer of complexity to drug-protein interaction studies.

Integrating diverse data types, including structural information, omics data, and clinical information, for a comprehensive understanding of drug-protein interactions poses challenges in terms of data integration and analysis.

Achieving personalized medicine goals requires predicting how individual genetic variations may impact drug-protein interactions. Tailoring drug therapies to individual genetic profiles requires a deeper understanding of these interactions.

Experimental methods for studying drug-protein interactions have limitations, such as the inability to observe interactions in real-time or in specific cellular contexts. Overcoming these limitations is crucial for accurate prediction and validation.

Addressing these challenges requires a multidisciplinary approach involving computational methods, experimental techniques, and a deep understanding of both drug and protein biology. Advances in this field have the potential to improve drug safety, efficacy, and the overall success of drug development efforts.

2.2 Proposed Solution :

Addressing the challenges in drug interaction with proteins requires a multifaceted approach. Firstly, leveraging advanced computational methods, such as machine learning algorithms and molecular docking simulations, can enhance the prediction of drug-protein interactions, aiding in the identification of potential off-targets and estimating binding affinities more accurately. Additionally, advancements in structural biology techniques, including cryo-electron microscopy and X-ray crystallography, can provide high-resolution insights into dynamic protein structures, enabling a better understanding of conformational changes and context-dependent interactions. Integrating multi-omics data and information on post-translational modifications can contribute to a more comprehensive view of drug-protein interactions, helping to predict personalized responses and address issues related to polypharmacology. Collaborative efforts between computational biologists, experimental researchers, and clinicians are essential for overcoming these challenges, fostering innovation in drug development, and ultimately improving the safety and efficacy of therapeutic interventions.

3. LITERATURE REVIEW

1. In DTiGEMS+: drug-target interaction prediction using graph embedding, graph mining, and similarity-based techniques paper, most of the approaches followed for DTi using target-based, drug-based, known DTi information are discussed. Also, interactions of drug-drug and protein-protein interactions were explained briefly. Parameters like recall, precision and FPR are calculated. One main flaw is that this approach is based on graph embedding which might not be applicable to all types of drug-target interactions. Moreover, only a limited amount of data is examined.

2. In Graph Convolutional Neural Networks for predicting Drug-Target interactions paper, importance of GNNs and the way they operate on graph data is explained. This also described how GNNs use message-passing algorithms to propagate information between nodes in a graph. RFScore is used as a scoring function to find the binding affinity. This paper showed that graph convolutional neural networks achieved some better performance when compared to other conventional methods. More area under the ROC curve indicates higher accuracy of drug-target interaction prediction.

3. How Powerful are Graph Neural Networks: The paper outlines the information that combines the theoretical analysis and observational validation to measure the ability of the GNNs in capturing the different graph structures and also showed that graph neural networks can be as effective as the Weisfeiler-Lehman test in recognising different graph structures, and developed a simple neural model called the Graph Isomorphism Network that has the same power as the WL test, also identifying the limitations in popular GNN variants like GCN (Graph Convolutional Networks) and GraphSAGE.

4. Drug-Target affinity prediction using graph neural networks and contact maps: The most important factors that affect the performance of a model include the architecture of GNN and the number of layers. The datasets used are KIBA and Davis. Methods used are DGraphDTA (double graph DTA predictor), a three layer GCN that combines both molecular and protein graphs. While coming to the metrics, they used concordance index, squared correlation coefficient, mean squared error, Pearson correlation coefficient has significantly improved the prediction for both datasets.

5. DeepNC: a framework for drug-target interaction prediction with graph neural networks: DeepNC (Deep Neural Computation) an advanced deep learning model, combines three GNN algorithms: Generalized Aggregation Networks (GENConv), Graph Convolutional Networks(GCNConv), Hypergraph Convolution Hypergraph Attention (HypergraphConv). Datasets used are KIBA, Davis, Allergy. The framework learns drug and target features using GNN layers and a 1D convolution network. Then, it uses fully-connected layers to predict

binding affinity values. DeepNC enhances performance in terms of mean square error (MSE) and concordance index (CI).

6. Predicting target–ligand interactions with graph convolutional networks for interpretable pharmaceutical discovery: PLA-Net, a two-module deep GCN to deal with TLI (Therapeutic Target-Ligand Interaction) prediction in a curated version of the AD dataset. Approach combines valuable data from both ligand and protein graphs using deep GCN modules and also introduced a method to create adversarial molecule augmentations that maintain biologically relevant details, further used in training for improving models performance.

7. Graph convolutional networks for computational drug development and discovery: This paper explains how GCNs can assist in drug-related tasks, including predicting molecular properties and activities, interaction prediction, synthesis prediction, and drug design. They tested various databases like PubChem, NCI, FreeSol for comparing results. They also discussed the foundations and various GCN architectures, highlighted real-world applications, and addressed the challenges and future prospects of using GCNs in drug discovery.

8. GNNExplainer: Generating Explanations for Graph Neural Networks: The datasets described in this paper are REDDIT-BINARY, MUTAG, BA-COMMUNITY, BA-SHAPES, TREE-GRID, TREE-SHAPE. The method described is GNNExplainer which is a model-agnostic method that provides clear explanations for predictions made by any GNN-based model on graph-based tasks. It identifies important subgraph structures and key node features that influence predictions and can provide consistent explanations for multiple instances. The approach is formulated as an optimization task, and experiments show that it outperforms other methods in explaining GNN predictions by up to 43.0%.

9. Prediction of drug‐drug interaction events using graph neural network based feature extraction: DDI prediction involves two primary tasks: first, identifying interactions among drugs, and the second, determining the type or nature of these interactions. DrugBank and KEGG databases are used. Accuracy, Area Under the Precision-Recall-Curve (AUPR), Area under the ROC curve (AUC), F1 score, Precision and Recall are the multiple metrics used.

10. Machine Learning for Drug-Target Interaction Prediction: In DTI prediction, the machine learning process includes three key steps: data preprocessing for drugs and targets, model training using learning rules, and making predictions for test data using the trained model. Used quantitative bioactivity data improved prediction accuracy and reliability. It addressed the remaining challenges and future prospects for machine learning in DTI prediction, aiming to offer valuable insights and guidance for future researchers.

11. ICAN: Interpretable cross-attention network for identifying drug and target protein interactions: Interpretable Cross-Attention Network (ICAN) which predicts DTIs using the simplified molecular input line entry system of drugs and amino acid sequences of target proteins. This model has three parts: an embedding layer, an attention layer, and an output layer.

DAVIS, BindingDB, and BIOSNAP are the three datasets taken into the test.

12. CAT-CPI: Combining CNN and transformer to learn compound image features for predicting compound-protein interactions: CAT-CPI, a novel molecular image-based model that combines CNN for local feature learning with a transformer encoder for capturing semantic relationships in molecular images, and a k-gram-based method with a transformer encoder to extract protein sequence features. The proposed model consists of three modules: one for extracting compound features, another for extracting protein features, and a third for the FR module. Datasets used for the CPI task namely Human, Celegans, Davis.

4. METHODOLOGY

4.1 Preliminaries :

Here, we have used the Biosnap dataset which contains the data in forms of train, test and validation files. For drugs, we extracted the features like symbol, degree, chirality, number of hydrogens, aromaticity, implicit valence and formal charge. Feature matrices are used to represent the symbol for the protein sequences. Among the drug features, chirality and formal charge are very helpful to get better results.

Symbols = `['C', 'N', 'O', 'S', 'F', 'Si', 'P', 'Cl', 'Br', 'Mg',`

`'Na', 'Ca', 'Fe', 'As', 'Al', 'I', 'B', 'V', 'K', 'Tl', 'Yb', 'Sb',`
`'Sn', 'Ag', 'Pd', 'Co', 'Se', 'Ti', 'Zn', 'H', 'Li', 'Ge', 'Cu',`
`'Au', 'Ni', 'Cd', 'In', 'Mn', 'Zr', 'Cr', 'Pt', 'Hg', 'Pb',`
`'Unknown']`

4.2 Problem Statement:

Using graph convolutional neural networks to train the model for better prediction of drug-target binding affinity. Working with GCNNs for the construction of the model as they are very handy in managing large data. Better accuracy is observed while using GCNNs when compared to other drug-target based methods.

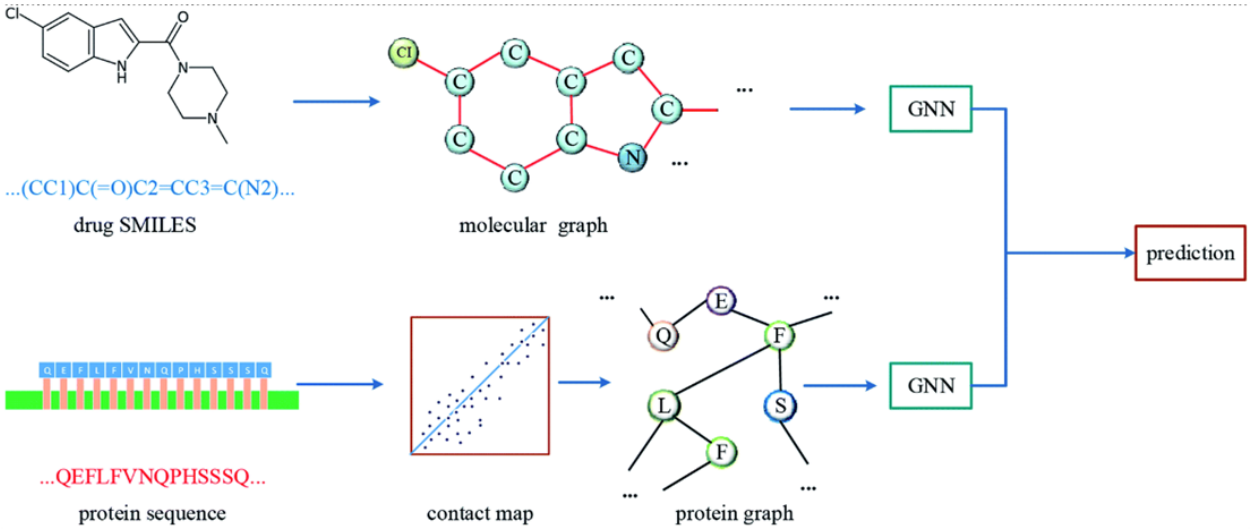
4.3 Model Architecture and Explanation:

Our methodology comprises three main components:

To begin with, we have concatenated three different types of datasets and subjected them to web crawling. In this step, we extracted the drug smiles and protein sequences from two different websites and placed them into a single file. This file contains smiles, sequences along with labels (0 or 1).

Secondly, we have preprocessed the dataset to convert both the smiles and sequences into graph format. This involves using the graph function from the networkx module. Post processing, we get a graph dataset which contains information in graph format. To visualize this information, another process is done to display the features of drugs and proteins which doesn't involve any inbuilt graph function.

Finally, we pass these graph features into a graph convolutional neural network model to train the model which gives out certain parameters like accuracy, precision, recall and F1 score. This binary classification model has three convolutional layers and twenty percent of the dataset is used for testing. The number of epochs can be mentioned and training loss as well as accuracies are calculated for each iteration. A drop out fraction of 0.2 is used to prevent overfitting. Number of epochs should be mentioned.



5. DEEP CHEM AND NETWORKX MODULE

5.1 DeepChem:

1. Overview:

DeepChem is an open-source library that provides tools and modules for deep learning in the domain of chemistry and life sciences. It is developed to enable researchers and practitioners to leverage deep learning techniques for various tasks related to molecular science.

2. Key Features:

Flexible Architecture: DeepChem offers a flexible and modular architecture that supports the development of a wide range of deep learning models for chemical informatics.

Data Handling: It includes functionalities for handling molecular datasets, featurizing molecules, and preparing data for training deep learning models.

Model Zoo: DeepChem provides a Model Zoo, which is a collection of pre-trained models for various chemical tasks, including molecular property prediction and compound generation.

3. Applications:

Drug Discovery: DeepChem is widely used in drug discovery and development. It aids in predicting molecular properties, screening potential drug candidates, and understanding the structure-activity relationships (SAR) of compounds.

Cheminformatics: The platform supports various cheminformatics tasks such as molecular fingerprinting, similarity search, and molecular representation.

4. Supported Tasks:

DeepChem supports a range of tasks, including but not limited to:

Molecular Property Prediction: Predicting properties like solubility, toxicity, or bioactivity.

QSAR (Quantitative Structure-Activity Relationship): Modeling the relationship between chemical structure and biological activity.

Virtual Screening: Identifying potential drug candidates through computational screening.

5. Integration with Deep Learning Frameworks:

DeepChem integrates with popular deep learning frameworks like TensorFlow and PyTorch, allowing users to leverage the capabilities of these frameworks for developing and training deep learning models.

6. Community and Collaboration:

Being an open-source project, DeepChem benefits from community contributions and collaborations. Researchers and developers worldwide contribute to its development, making it a dynamic and evolving platform.

7. Challenges and Future Developments:

Like any tool or platform, DeepChem faces challenges, including the need for continuous updates to keep pace with advancements in deep learning and molecular science. Future developments may involve improving model interpretability, handling larger datasets, and expanding the range of supported tasks.

5.2 NetworkX:

NetworkX stands as a foundational and versatile Python library essential for analyzing, modeling, and visualizing complex networks or graphs. It serves as a pivotal toolkit, offering a rich set of functionalities for researchers, data scientists, and analysts across diverse domains where understanding interconnected data structures is crucial.

At its essence, NetworkX excels in its capability to handle a wide range of graph structures. Whether dealing with directed, undirected, weighted, or bipartite graphs, NetworkX provides an extensive collection of data structures tailored to various network modeling needs. This adaptability is invaluable in real-world applications, where networks often exhibit diverse and intricate relationships.

One of NetworkX's fundamental features is its ability to facilitate graph visualization. It enables users to create graphical representations that offer insights into the structure and patterns within complex networks. These visualizations play a crucial role in conveying information derived from network analyses, aiding in understanding and decision-making.

However, NetworkX's true strength lies in its comprehensive suite of algorithms designed for network analysis. Covering a wide spectrum of functionalities, these algorithms include measures for centrality, shortest path calculations, clustering, community detection, and more. These capabilities empower users to perform in-depth analyses on networks, extracting key information, and revealing valuable insights.

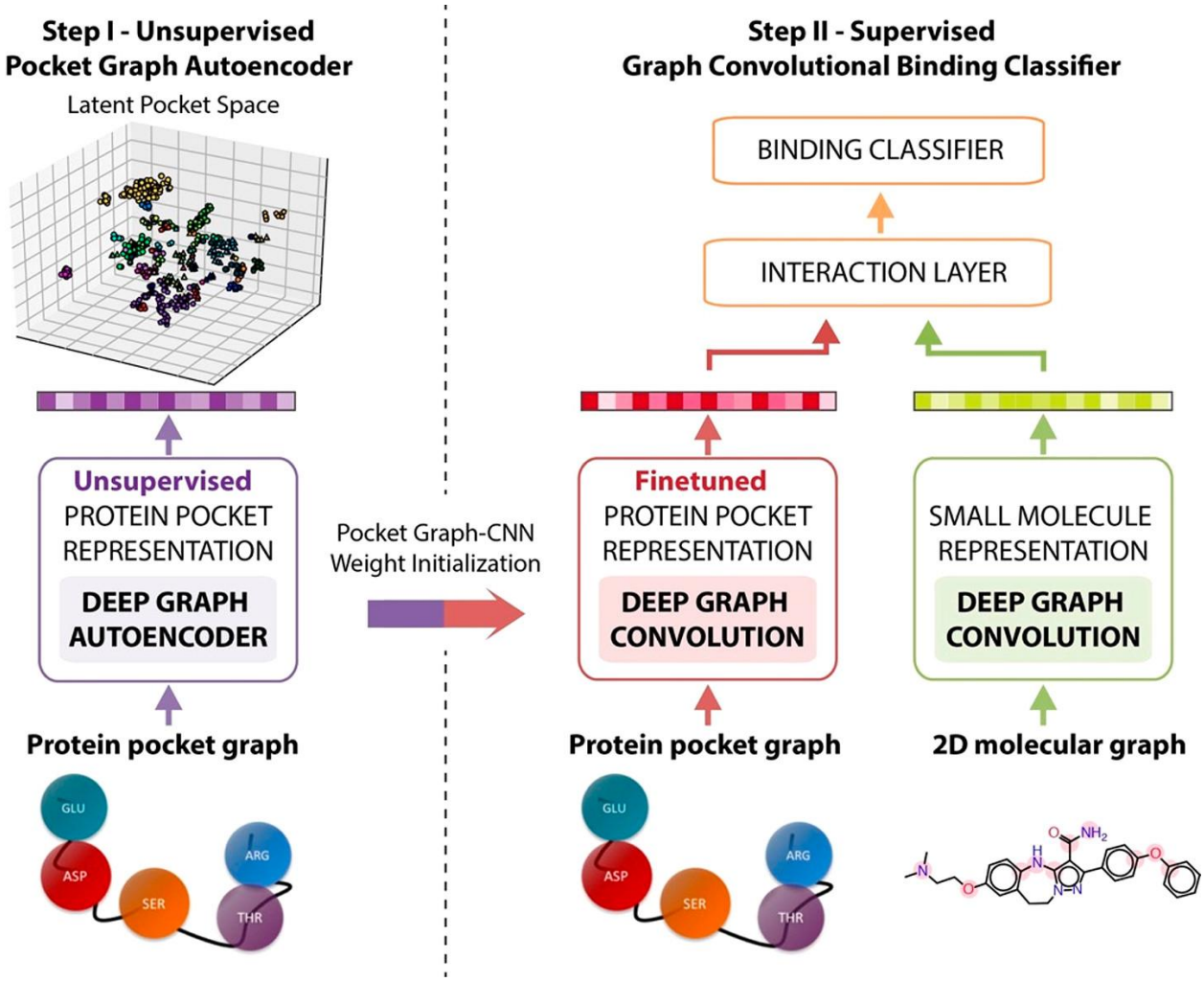
The library's ease of use and flexibility allow for rapid prototyping and experimentation. It integrates seamlessly with other Python libraries and tools, enabling its incorporation into diverse workflows and analytical pipelines.

NetworkX finds applications across a multitude of domains. In social network analysis, it helps in understanding connections between individuals or communities. In biology, it aids in modeling molecular interactions and biological pathways. Moreover, in transportation or infrastructure planning, it assists in modeling connectivity and optimizing networks.

The library's versatility and robustness make it an indispensable tool for exploring hidden patterns, understanding relationships, and deriving actionable insights from intricate networked datasets.

In conclusion, NetworkX emerges as a cornerstone in network analysis, providing an extensive repertoire of functionalities and tools to explore, understand, and derive meaningful conclusions from complex network structures. Its capacity to handle diverse graph structures.

6. FLOW CHART



Overview of the proposed two-step graph-convolutional framework. In step I, an unsupervised pocket graph autoencoder is trained on a representative druggable pocket set to learn general pocket features and to embed protein pockets into a fixed-size latent space. In step II, we constructed a pocket Graph-CNN and a ligand Graph-CNN to extract features from the pocket graphs and 2D ligand graphs, respectively. To allow the network to recognize diverse pocket features, the pocket Graph-CNN is initialized with learned weights from step I. The interaction layer then integrates features learned from the pocket and ligand Graph-CNNs. Finally, the

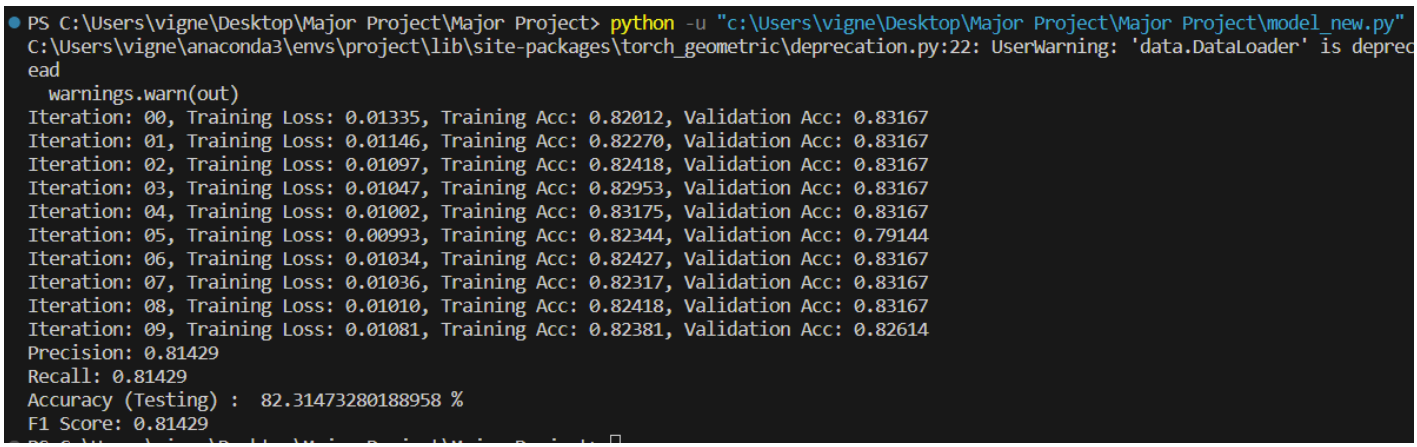
classifier takes in the learned interactions to perform binding predictions. In step II, the model training is driven by the binding classification labels. Therefore, the model will automatically extract task-specific features characterizing interactions between target and ligands. Furthermore, because the model takes in pocket and ligand graphs separately, the model does not require protein-ligand complexes as input.

We visualize the learned pocket embedding in 2D space and show that the latent representations reflect biological pocket similarity. We then perform head-to-head comparisons of prediction performance between our Graph-CNNs, 3DCNN ligand-scoring, AutoDock Vina, RF-Score, and NNScore, on the Database of Useful Decoys: Enhanced (DUD-E) and Maximum Unbiased Validation (MUV) data set and showed that our model achieved better or comparable performances on both data sets without requiring cocrystal structures. In addition to the conventional virtual screening setting, we further examined the ability of our model to predict binding propensity of a given compound to different targets to estimate specificity and showed that the model can predict meaningful ligand-target binding profiles. Finally, we visualized individual contributions of each pocket residue and ligand atom to the classification decisions and showed that our networks can capture key interface residues and key ligand atoms in protein pockets and ligands.

7. RESULT

Training loss, training accuracy and validation accuracy can be observed for each epoch (Iteration). Precision, Recall and F1-Score values are around 82.7%. Training loss, training accuracy and validation accuracy can be observed for each epoch (iteration). Increasing the number of features for drugs helps to distinguish the compounds easily and thereby helps to improve the accuracy.

New Model Result:



GCNN(Graph Convolutional Neural Networks) Result:

```
PS C:\Users\vigne\Desktop\Major Project\Major Project> python -u "c:\Users\vigne\Desktop\Major Project\Major Project\GraphModel.py"
C:\Users\vigne\anaconda3\envs\project\lib\site-packages\torch_geometric\deprecation.py:22: UserWarning: 'data.DataLoader' is deprec
ead
  warnings.warn(out)
Epoch: 000, Train Loss: 0.01343, Train Acc: 0.81818, Val Loss: 0.37539, Val Acc: 0.83167
Epoch: 001, Train Loss: 0.01107, Train Acc: 0.82815, Val Loss: 0.38040, Val Acc: 0.83167
Epoch: 002, Train Loss: 0.01031, Train Acc: 0.83544, Val Loss: 0.39325, Val Acc: 0.82171
Epoch: 003, Train Loss: 0.01054, Train Acc: 0.84079, Val Loss: 0.38408, Val Acc: 0.84090
Epoch: 004, Train Loss: 0.01022, Train Acc: 0.85215, Val Loss: 0.37158, Val Acc: 0.83869
Epoch: 005, Train Loss: 0.00962, Train Acc: 0.86405, Val Loss: 0.42280, Val Acc: 0.82503
Epoch: 006, Train Loss: 0.00902, Train Acc: 0.87125, Val Loss: 0.39726, Val Acc: 0.84238
Epoch: 007, Train Loss: 0.00882, Train Acc: 0.87697, Val Loss: 0.44680, Val Acc: 0.84681
Epoch: 008, Train Loss: 0.00875, Train Acc: 0.87716, Val Loss: 0.47624, Val Acc: 0.84976
Epoch: 009, Train Loss: 0.00863, Train Acc: 0.88288, Val Loss: 0.41115, Val Acc: 0.84976
Precision: 0.78506
Recall: 0.78506
Test Accuracy: 0.83614
F1 Score: 0.78506
PS C:\Users\vigne\Desktop\Major Project\Major Project>
```

Here we can observe that we are getting more accuracy in the GCNN(Graph Convolutional Neural Networks) Model , so we are choosing GCNN Model for Prediction.

8. CONCLUSION AND FUTURE SCOPE

8.1 Conclusion:

This study has advanced our understanding of drug target interactions and prediction methodologies.The strength of binding affinity between these drugs and proteins not only influences the pace of disease mitigation but also underscores the pivotal role played by binding affinity in modulating their interaction dynamics.Throughout this research, we tried to use the modern computation techniques to predict the results efficiently. We used a binary classification approach for the nature of the interaction between drugs and their respective targets. Methods and the approach we used for prediction improves and gives an accuracy which is far more better than others. The number of epochs, a critical parameter in our model, plays a crucial role in shaping the efficiency of our predictions.

8.2 Future Scope:

The study of how drugs interact with specific targets in our bodies and predicting these interactions is an evolving field and has a lot of potential for future research.

1. Computational techniques help in predicting interactions between multiple drugs and targets simultaneously, leading to the development of drugs that can cure multiple diseases.

2. Prediction of personalized drug-target interactions can lead to the creation of targeted medicines based on an individual's genetic features and other environmental factors.

3. Encourage scientists and companies to share their data so that we have more information to work with.

4. It's important to test the predictions made by computers in real life experiments to make sure they are right .

5. Think about the rules and guidelines we need to make sure all of tsi is done ethically and safely.

6. Look at how we can use computers to find helpful substances in nature and understand how they work.

9. APPLICATIONS

9.1 Objective of Application:

The project exploring drug-target interaction prediction using Graph Convolutional Neural Networks (GCNNs) holds substantial potential across various critical applications in pharmaceutical and biomedical fields. Firstly, it could revolutionize drug discovery pipelines by significantly expediting the identification of potential drug candidates. By accurately predicting interactions between compounds and target proteins, this technology enables computational screening of vast compound libraries, saving time and resources compared to traditional experimental methods. Moreover, personalized medicine stands to benefit immensely from this advancement. The ability to forecast precise drug-target interactions based on an individual's genetic makeup and specific environmental factors can pave the way for tailored treatments, enhancing therapeutic outcomes and minimizing adverse effects. Furthermore, the computational predictions can aid in repurposing existing drugs for novel indications, contributing to more cost-effective and time-efficient strategies in pharmaceutical research and development. Ethical considerations and data sharing initiatives also gain prominence, fostering collaborative efforts and ensuring responsible utilization of predictive models in drug development and healthcare.

9.2 Graph Convolution Neural Network:

Graph Convolutional Neural Networks (GCNNs) represent a pivotal advancement in machine learning, particularly in handling and analyzing graph-structured data. Unlike traditional neural networks designed for grid-like data such as images or sequences, GCNNs are tailored for graph data, allowing them to effectively capture and utilize the intricate relationships and dependencies present within graphs. They operate by aggregating and propagating information across nodes and edges in a graph, enabling tasks like node classification, graph classification, and link prediction. GCNNs leverage convolutional operations, adapted to graphs, to extract meaningful features from the nodes and their local neighborhoods. This unique architecture endows GCNNs with the capability to discern patterns, hierarchies, and structures inherent in complex networks, making them invaluable in diverse domains such as social networks, bioinformatics, recommendation systems, and chemical analysis. Their flexibility, efficiency in handling large-scale graph data, and ability to learn representations from graph-structured inputs underscore their significance as a powerful tool for analyzing and understanding complex relational data. Ongoing research continues to refine GCNN architectures and explore their applications in multifaceted domains, promising further advancements in leveraging graph-data for machine tasks

10. CODE SCREENSHOTS

Web Scraping Code:

web_scrap.py > ...

```python
from collections import Counter
import ast
import numpy
import pandas

info = pandas.concat([pandas.read_csv("biosnap/train.csv")[["Drug_ID", "Protein_ID"]],pandas.read_csv("biosnap/val.csv")[["Drug_ID", "Pro
temp = numpy.array(list(Counter(numpy.concatenate((info["Drug_ID"].values, info["Protein_ID"].values))).keys())) #indexing

# from msilib import sequence
# from bs4 import BeautifulSoup
# from urllib.request import Request, urlopen
# from time import*
# from typing import Sequence

# seq = {}
# baseUrl = "http://www.uniprot.org/uniprot/"
# maxTries = 4
# for i in tqdm(prot_ID):
#     currentUrl = baseUrl + i + ".fasta"
#     delay = 4
#     for _ in range(maxTries):
#         try:
#             page = urlopen(currentUrl)
#             html = page.read().decode("utf-8")
#             soup = BeautifulSoup(html, "html.parser")
#             seq[i] = soup.text
#             break
#         except:
#             sleep(delay)
#             delay *= 3

# for i in seq.keys():
#     seq[i] = ''.join(seq[i].split("\n")[1:])

# file = open('proteins.txt', 'wt')
# file.write(str(seq))
```

```
# file.close()

# db_seq = {}
# notFetched = []


# baseUrl = "https://go.drugbank.com/drugs/"
# for i in (DB_ID):
#     try:
#         url = baseUrl + i
#         req = Request(url , headers={'User-Agent': 'Mozilla/5.0'})
#         page = urlopen(req)
#         html = page.read().decode("utf-8")
#         soup = BeautifulSoup(html, "html.parser")

#         d = soup.find_all("div", attrs={"class": "wrap"})
#         db_seq[i] = d[2].text
#     except:
#         notFetched.append(i)
#         continue

# file = open('drugs.txt', 'wt')
# file.write(str(db_seq))
# file.close()

data = open('proteins.txt', 'r')
prot_seq_from_txt = ast.literal_eval(data.read())

Data = open('drugs.txt', 'r')
drug_seq_from_txt = ast.literal_eval(Data.read())

INFO = pandas.concat([pandas.read_csv("biosnap/train.csv")[["Drug_ID", "Protein_ID", "label"]], pandas.read_csv("biosnap/val.csv")[["Drug
```

```
INFO = INFO[INFO["Drug_ID"].isin(set(list(drug_seq_from_txt.keys())))]
INFO = INFO[INFO["Protein_ID"].isin(set(list(prot_seq_from_txt.keys())))]

InfoNew = pandas.DataFrame(columns=["Drug", "Protein", "label"])
InfoNew["Drug"] = INFO["Drug_ID"].apply(lambda A: drug_seq_from_txt[A])
InfoNew["Protein"] = INFO["Protein_ID"].apply(lambda A: prot_seq_from_txt[A])
InfoNew["label"] = INFO["label"]

InfoNew.to_csv("data.csv", index=False)
```

Random Forest Code:

RandomForest.py > ...

```python
# import pandas as pd
# from rdkit import Chem
# from sklearn.model_selection import train_test_split
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.metrics import accuracy_score, classification_report

# data = pd.read_csv('data.csv')

# AA_MAPPING = {
#     'A': 0,
#     'C': 1,
#     'D': 2,
#     'E': 3,
#     'F': 4,
#     'G': 5,
#     'H': 6,
#     'I': 7,
#     'K': 8,
#     'L': 9,
#     'M': 10,
#     'N': 11,
#     'P': 12,
#     'Q': 13,
#     'R': 14,
#     'S': 15,
#     'T': 16,
#     'V': 17,
#     'W': 18,
#     'Y': 19
# }


# mol_objects = []
# for smile in data['Drug']:
#     mol = Chem.MolFromSmiles(str(smile))
#     mol_objects.append(mol)

# seq_objects = [str(seq) for seq in data['Protein'].tolist()]

# labels = data['label'].tolist()

# # Initialize empty list to store graph representations of molecules
# graphs = []

# # Loop over all molecule objects and create graph representations
# for i, mol in enumerate(mol_objects):
#     if mol is None:
#         continue
#     graph = {}

#     # Add node features for atoms
#     for atom in mol.GetAtoms():
#         symbol = atom.GetSymbol()
#         charge = atom.GetFormalCharge()
#         if charge == 0:
#             feature = [1, 0]  # neutral atom
#         elif charge > 0:
#             feature = [0, 1]  # positively charged atom
#         else:
#             feature = [0, -1]  # negatively charged atom
#         graph[atom.GetIdx()] = {'symbol': symbol, 'Charge': feature}

#     # Add edge features for bonds
#     for bond in mol.GetBonds():
#         begin_atom = bond.GetBeginAtomIdx()
#         end_atom = bond.GetEndAtomIdx()
#         bond_type = bond.GetBondTypeAsDouble()
#         graph[begin_atom][end_atom] = {'bond_type': bond_type}
#         graph[end_atom][begin_atom] = {'bond_type': bond_type}

#     # Add node features for amino acids
#     seq = seq_objects[i]
#     for j, aa in enumerate(seq):
```

```
#         if aa in AA_MAPPING:
#             feature = [0] * 20
#             feature[AA_MAPPING[aa]] = 1
#             graph[j + mol.GetNumAtoms()] = {'symbol': aa, 'feature': feature}

#     # Add additional features
#     graph['size'] = mol.GetNumAtoms()
#     num_rings = mol.GetRingInfo().NumRings()
#     graph['num_rings'] = num_rings
#     for atom in mol.GetAtoms():
#         if atom.GetChiralTag() != Chem.ChiralType.CHI_UNSPECIFIED:
#             graph[atom.GetIdx()]['chiral_center'] = True
#         else:
#             graph[atom.GetIdx()]['chiral_center'] = False
#     for atom in mol.GetAtoms():
#         if atom.GetIsAromatic():
#             graph[atom.GetIdx()]['aromatic'] = True
#         else:
#             graph[atom.GetIdx()]['aromatic'] = False

#     # Add label for drug target interaction
#     graph['label'] = labels[i]

#     graphs.append(graph)

# # Extract features and labels from the graphs
# features = []
# labels = []
# for graph in graphs:
#     features.append([node['symbol'] for node in graph.values() if isinstance(node, dict)])
#     labels.append(graph['label'])


# # Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# # Convert the features into a numerical representation
```

```
# # Convert the features into a numerical representation
# X_train_encoded = pd.get_dummies(pd.DataFrame(X_train).stack(), prefix_sep='').sum(level=0)
# X_test_encoded = pd.get_dummies(pd.DataFrame(X_test).stack(), prefix_sep='').sum(level=0)

# # Align the encoded test data with the encoded training data to ensure consistent feature representation
# X_test_encoded = X_test_encoded.reindex(columns=X_train_encoded.columns, fill_value=0)


# # Initialize the Random Forest classifier
# classifier = RandomForestClassifier()

# # Train the classifier
# classifier.fit(X_train_encoded, y_train)

# # Make predictions on the test set
# y_pred = classifier.predict(X_test_encoded)

# # Calculate accuracy and print classification report
# accuracy = accuracy_score(y_test, y_pred)
# report = classification_report(y_test, y_pred)
# print("Accuracy:", accuracy)
# print("Classification Report:")
# print(report)

import pandas as pd
from rdkit import Chem
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

data = pd.read_csv('data2.csv')

AA_MAPPING = {
    'A': 0,
    'C': 1,
    'D': 2,
```

```
    'E': 3,
    'F': 4,
    'G': 5,
    'H': 6,
    'I': 7,
    'K': 8,
    'L': 9,
    'M': 10,
    'N': 11,
    'P': 12,
    'Q': 13,
    'R': 14,
    'S': 15,
    'T': 16,
    'V': 17,
    'W': 18,
    'Y': 19
}

mol_objects = []
for smile in data['Drug']:
    mol = Chem.MolFromSmiles(str(smile))
    mol_objects.append(mol)

seq_objects = [str(seq) for seq in data['Protein'].tolist()]

labels = data['label'].tolist()

graphs = []
for i, mol in enumerate(mol_objects):
    if mol is None:
        continue
    graph = {}

    for atom in mol.GetAtoms():
        symbol = atom.GetSymbol()
        charge = atom.GetFormalCharge()
```

```
        if charge == 0:
            feature = [1, 0]  # neutral atom
        elif charge > 0:
            feature = [0, 1]  # positively charged atom
        else:
            feature = [0, -1]  # negatively charged atom
        graph[atom.GetIdx()] = {'symbol': symbol, 'Charge': feature}

    for bond in mol.GetBonds():
        begin_atom = bond.GetBeginAtomIdx()
        end_atom = bond.GetEndAtomIdx()
        bond_type = bond.GetBondTypeAsDouble()
        graph[begin_atom][end_atom] = {'bond_type': bond_type}
        graph[end_atom][begin_atom] = {'bond_type': bond_type}

    seq = seq_objects[i]
    for j, aa in enumerate(seq):
        if aa in AA_MAPPING:
            feature = [0] * 20
            feature[AA_MAPPING[aa]] = 1
            graph[j + mol.GetNumAtoms()] = {'symbol': aa, 'feature': feature}

    graph['label'] = labels[i]

    graphs.append(graph)

features = []
labels = []
for graph in graphs:
    features.append([node['symbol'] for node in graph.values() if isinstance(node, dict)])
    labels.append(graph['label'])

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# Convert the features into a numerical representation (if needed)
# ...
```

```
220
221  # Initialize the Random Forest classifier
222  classifier = RandomForestClassifier()
223
224  # Train the classifier
225  classifier.fit(X_train, y_train)
226
227  # Make predictions on the test set
228  y_pred = classifier.predict(X_test)
229
230  # Calculate accuracy and print classification report
231  accuracy = accuracy_score(y_test, y_pred)
232  report = classification_report(y_test, y_pred)
233  print("Accuracy:", accuracy)
234  print("Classification Report:")
235  print(report)
```

Data Preprocessing:

preprocessing.py > ...

```
1   import torch
2   import pandas
3   import numpy
4   from networkx import Graph
5   from rdkit import Chem
6   from torch_geometric import data as data_fun_torch
7
8   CsvData = pandas.read_csv('data.csv')
9   info = CsvData.values
10  array = []
11
12
13  def one_hot_encoding(x, allowable_set):
14      if x not in allowable_set:
15          raise Exception(
16              "input {0} not in allowable set{1}:".format(x, allowable_set))
17      return list(map(lambda s: x == s, allowable_set))
18
19  def func_get(a, requiredSet):
20      if a not in requiredSet:
21          a = requiredSet[-1]
22      return list(map(lambda s: a == s, requiredSet))
23
24  def is_chiral_atom(atom):
25      chiral_tag = atom.GetChiralTag()
26      if chiral_tag in [Chem.ChiralType.CHI_TETRAHEDRAL_CW, Chem.ChiralType.CHI_TETRAHEDRAL_CCW]:
27          return 1  # Atom is chiral
28      else:
29          return 0  # Atom is not chiral
30
31
32  def AtomFeatures(atom):
33      l= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
34      return numpy.array(func_get(atom.GetSymbol(), ['C', 'N', 'O', 'S', 'F', 'Si', 'P', 'Cl', 'Br', 'Mg', 'Na', 'Ca', 'Fe', 'As', 'Al', 'I
35                      one_hot_encoding(atom.GetDegree(), [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) +
36                      func_get(is_chiral_atom(atom), [0, 1]) +
37                      func_get(atom.GetTotalNumHs(), [0, 1, 2, 3, 4]) +
```

```
                    func_get(atom.GetFormalCharge(), [-4, -3, -2, -1, 0, 1, 2, 3, 4]) + # v- NB -B/2
                    func_get(atom.GetImplicitValence(), [0, 1, 2, 3, 4, 5, 6, 7, 8]) +
                    [atom.GetIsAromatic()])

def Molecules_Smiles(DrugSmile):
    DrugMolecule = Chem.MolFromSmiles(DrugSmile)     #get molecules from smiles string
    length = DrugMolecule.GetNumAtoms()
    matrix = []
    for node in DrugMolecule.GetAtoms():
        vector = AtomFeatures(node)           #feature vector
        matrix.append(vector / sum(vector))  #Normalising

    links = []    #stores atom indices
    for bond in DrugMolecule.GetBonds():
        links.append([bond.GetBeginAtomIdx(), bond.GetEndAtomIdx()]) #stored as pair in a list
    edg = Graph(links).to_directed()          #form directed edges of the graph
    Edgindex = []         #stores edge indices
    for Edge1, Edge2 in edg.edges:  #iterating edges
        Edgindex.append([Edge1, Edge2])

    return length, matrix, Edgindex   #no.of atoms, features, indices of edges


SequenceDict = {b: (a+1) for a, b in enumerate("ACDEFGHIKLMNPQRSTVWY")} #B J O U X Z
SeqMaxLen = 1000

def SequenceCat(prot):                        #maps characters to integers
    p = numpy.zeros(SeqMaxLen)
    for a, b in enumerate(prot[:SeqMaxLen]):
        p[a] = SequenceDict[b]
    return p

arrayValues = []
for data in range(info.shape[0]):
    try:
        DrugSmiles = info[data, 0]
```

```
        ProSequence = info[data, 1]
        ClassLabel = info[data, 2]
        length, matrix, Edgindex = Molecules_Smiles(DrugSmiles)
        GraphConvolutionData = data_fun_torch.Data(x=torch.tensor(matrix, dtype=torch.float), edge_index=torch.tensor(Edgindex, dtype=tor
        result = SequenceCat(ProSequence)
        GraphConvolutionData.target = torch.LongTensor([result])
        arrayValues.append(GraphConvolutionData)
    except:
        pass
torch.save(arrayValues, 'preprocessed_dataset.pt')
```

Data Preprocessing 2:

preproces2.py > ...

```python
import pandas as pd
from rdkit import Chem
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

data = pd.read_csv('data.csv')

AA_MAPPING = {
    'A': 0,
    'C': 1,
    'D': 2,
    'E': 3,
    'F': 4,
    'G': 5,
    'H': 6,
    'I': 7,
    'K': 8,
    'L': 9,
    'M': 10,
    'N': 11,
    'P': 12,
    'Q': 13,
    'R': 14,
    'S': 15,
    'T': 16,
    'V': 17,
    'W': 18,
    'Y': 19
}


mol_objects = []
for smile in data['Drug']:
    mol = Chem.MolFromSmiles(str(smile))
    mol_objects.append(mol)

```

```python
seq_objects = [str(seq) for seq in data['Protein'].tolist()]

labels = data['label'].tolist()

# Initialize empty list to store graph representations of molecules
graphs = []

# Loop over all molecule objects and create graph representations
for i, mol in enumerate(mol_objects):
    if mol is None:
        continue
    graph = {}

    # Add node features for atoms
    for atom in mol.GetAtoms():
        symbol = atom.GetSymbol()
        charge = atom.GetFormalCharge()
        if charge == 0:
            feature = [1, 0]  # neutral atom
        elif charge > 0:
            feature = [0, 1]  # positively charged atom
        else:
            feature = [0, -1]  # negatively charged atom
        graph[atom.GetIdx()] = {'symbol': symbol, 'Charge': feature}

    for bond in mol.GetBonds():
        begin_atom = bond.GetBeginAtomIdx()
        end_atom = bond.GetEndAtomIdx()
        bond_type = bond.GetBondTypeAsDouble()
        graph[begin_atom][end_atom] = {'bond_type': bond_type}
        graph[end_atom][begin_atom] = {'bond_type': bond_type}

    seq = seq_objects[i]
    for j, aa in enumerate(seq):
        if aa in AA_MAPPING:
            feature = [0] * 20
```

```
                feature[AA_MAPPING[aa]] = 1
                graph[j + mol.GetNumAtoms()] = {'symbol': aa, 'feature': feature}

        graph['size'] = mol.GetNumAtoms()


        for atom in mol.GetAtoms():
            if atom.GetIsAromatic():
                graph[atom.GetIdx()]['aromatic'] = True
            else:
                graph[atom.GetIdx()]['aromatic'] = False

        # Add label for drug target interaction
        graph['label'] = labels[i]

        graphs.append(graph)

    print(graphs[1])
```

New Model:

model_new.py > ...

```
import torch
import torch.nn as nn
from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from torch_geometric.data import DataLoader
from torch_geometric.nn import GCNConv, global_max_pool
import warnings

dataset = torch.load('preprocessed_dataset.pt')
train_dataset, test_dataset = train_test_split(dataset, test_size=0.2, random_state=42)
train_dataset, val_dataset = train_test_split(train_dataset, test_size=0.2, random_state=42)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, drop_last=True)
val_loader = DataLoader(val_dataset, shuffle=True, drop_last=True)
test_loader = DataLoader(test_dataset, shuffle=True, drop_last=True)

warnings.filterwarnings("ignore")


def Model(): #defines and returns neural network model
    class GCNNet(torch.nn.Module): #This class inherits from torch.nn.Module which has several functions of neural network architecture
        def __init__(self, n_output=2, n_filters=32, embed_dim=128, drug_feat=81, prot_feat=20, output_dim=128,
                     dropout=0.2): #dropping some fraction of neurons to prevent overfitting

            super(GCNNet, self).__init__()

            self.n_output = n_output #assigns the no.of output classes
            self.conv1 = GCNConv(drug_feat, drug_feat)
            self.conv2 = GCNConv(drug_feat, drug_feat * 2)
            self.conv3 = GCNConv(drug_feat * 2, drug_feat * 4)
            self.fc_g1 = torch.nn.Linear(drug_feat * 4, 1024) #These lines define fully connected layers that follow the graph convolutio
            self.fc_g2 = torch.nn.Linear(1024, output_dim)
            self.relu = nn.ReLU() #This line defines the activation function (Rectified Linear Unit, ReLU) that is used after each layer'
            self.dropout = nn.Dropout(dropout)

            self.embedding_xt = nn.Embedding(prot_feat + 1, embed_dim) #embedding layers which convert discrete inputs into continuous em
            self.conv_xt_1 = nn.Conv1d(in_channels=1000, out_channels=n_filters, kernel_size=8)
            self.fc1_xt = nn.Linear(32 * 121, output_dim)
```

```
            self.fc1 = nn.Linear(2 * output_dim, 1024)
            self.fc2 = nn.Linear(1024, 512)
            self.out = nn.Linear(512, self.n_output)

        def forward(self, data): #here, input data is unpacked and passed
            x, edge_index, batch = data.x, data.edge_index, data.batch #batch contains information about which nodes belong to which grap
            target = data.target

            x = self.conv1(x, edge_index) #These lines perform graph convolutional operations
            x = self.relu(x)
            x = self.conv2(x, edge_index)
            x = self.relu(x)
            x = self.conv3(x, edge_index)
            x = self.relu(x)            #This sequence of operations extracts hierarchical graph features from the input data.

            x = global_max_pool(x, batch) #Global max pooling computes the maximum value for each feature across all nodes within each gr
            x = self.relu(self.fc_g1(x))
            x = self.dropout(x)
            x = self.fc_g2(x)
            x = self.dropout(x)

            embedded_xt = self.embedding_xt(target) #an embedding layer that converts the discrete text data into continuous embeddings
            conv_xt = self.conv_xt_1(embedded_xt)
            xt = conv_xt.view(-1, 32 * 121)
            xt = self.fc1_xt(xt) #1D convolutional layer (self.conv_xt_1) is applied, followed by reshaping (view) and linear transformat

            xc = torch.cat((x, xt), 1)
            xc = self.fc1(xc) #. The concatenated features are then passed through fully connected layers (self.fc1 and self.fc2) with Re
            xc = self.relu(xc)
            xc = self.dropout(xc)
            xc = self.fc2(xc)
            xc = self.relu(xc)
            xc = self.dropout(xc)
            out = self.out(xc) # the output is passed through the last fully connected layer (self.out), which produces the model's predi
            return out
```

```
    model = GCNNet()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    loss_function = nn.CrossEntropyLoss()
    return model, optimizer, loss_function #The model, optimizer, and loss_function are then returned as a tuple from the Model function.

def TrainTheModel(num_itr=2):
    model, optimizer, loss_function = Model() #It calls the Model() function to initialize the neural network model (model), optimizer (o
    train_loss = torch.zeros(num_itr)
    valid_loss = torch.zeros(num_itr)
    train_acc = torch.zeros(num_itr)
    valid_acc = torch.zeros(num_itr) #initialising 4 tensors
    Final_acc = 0
    Final_model = None
    for itr in range(num_itr):
        model.train() #sets the model in training mode
        for z, data in enumerate(train_loader): #z is just an enumeration variable, and data represents batch of training data.
            optimizer.zero_grad() #clears gradients of prev. parameters
            out = model(data) #computes model prediction
            loss = loss_function(out, data.y) #It calculates the loss between the model's predictions (out) and the true labels (data.y)
            loss.backward() #It computes the gradients of the loss with respect to the model's parameters
            optimizer.step() #Updates the model's parameters using the optimizer based on the obtained gradients.
            train_loss[itr] += loss.item() #training loss
            pred = out.max(1)[1] #It calculates the predicted class labels by taking class with highest probability
            train_acc[itr] += pred.eq(data.y).sum().item() #It computes the training accuracy by comparing the predicted labels (pred) wi
        train_loss[itr] /= len(train_loader.dataset) #training loss and accuracy for the current epoch are normalized by dividing by the
        train_acc[itr] /= len(train_loader.dataset)

        model.eval() #sets in evaluation mode
        for data in val_loader: #It computes the model's predictions (out) for the current batch of validation data.
            out = model(data)
            loss = loss_function(out, data.y)
            valid_loss[itr] += loss.item()
            pred = out.max(1)[1]
            valid_acc[itr] += pred.eq(data.y).sum().item()
        valid_loss[itr] /= len(val_loader.dataset)
        valid_acc[itr] /= len(val_loader.dataset)
```

```
        if (valid_acc[itr] > Final_acc):
            Final_model = model
        print(f'Iteration: {itr:02d}, Training Loss: {train_loss[itr]:.5f}, Training Acc: {train_acc[itr]:.5f}, Validation Acc: {valid_ac
    return Final_model, train_loss, valid_loss, train_acc, valid_acc

model, train_loss, valid_loss, train_acc, valid_acc = TrainTheModel()

torch.save(model.state_dict(), 'dataset.pt')

model.load_state_dict(torch.load('dataset.pt'))
model.eval()
test_acc = 0
precision = 0
recall = 0
for data in test_loader:
    out = model(data)
    pred = out.max(1)[1]
    test_acc = test_acc + pred.eq(data.y).sum().item()
    precision = precision + precision_score(data.y, pred)
    recall = recall + recall_score(data.y, pred)

precision = precision/len(test_loader.dataset) #normalising
recall = recall/len(test_loader.dataset)
test_acc = test_acc/len(test_loader.dataset)

f1_score = 2*(precision*recall)/(precision + recall)

print(f'Precision: {precision:.5f}')
print(f'Recall: {recall:.5f}')
print("Accuracy (Testing) : ", test_acc*100,"%")
print(f'F1 Score: {f1_score:.5f}')
```

GCNN(Graph Convolutional Neural Networks):

GraphModel.py > ...

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from sklearn.metrics import*
from sklearn.model_selection import train_test_split
from torch_geometric.data import DataLoader
from torch_geometric.nn import GCNConv, global_max_pool
import warnings


dataset = torch.load('preprocessed_dataset.pt')
train_dataset, test_dataset = train_test_split(dataset, test_size=0.2, random_state=42)
train_dataset, val_dataset = train_test_split(train_dataset, test_size=0.2, random_state=42)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, drop_last=True)
val_loader = DataLoader(val_dataset, shuffle=True, drop_last=True)
test_loader = DataLoader(test_dataset, shuffle=True, drop_last=True)

warnings.filterwarnings("ignore")

def Model():
    class GCNNet(torch.nn.Module): #nf labels #nf filters #dimensionality of embed  #                                    #dropping frac
        def __init__(self, n_output=2, n_filters=32, embed_dim=128, num_features_xd=81, num_features_xt=25, output_dim=128, dropout=0.2):

            super(GCNNet, self).__init__()

            self.n_output = n_output
            self.conv1 = GCNConv(num_features_xd, num_features_xd)
            self.conv2 = GCNConv(num_features_xd, num_features_xd*2)
            self.conv3 = GCNConv(num_features_xd*2, num_features_xd * 4)
            self.fc_g1 = torch.nn.Linear(num_features_xd*4, 1024)
            self.fc_g2 = torch.nn.Linear(1024, output_dim)
            self.relu = nn.ReLU()
            self.dropout = nn.Dropout(dropout)

            self.embedding_xt = nn.Embedding(num_features_xt + 1, embed_dim)
            self.conv_xt_1 = nn.Conv1d(in_channels=1000, out_channels=n_filters, kernel_size=8)
            self.fc1_xt = nn.Linear(32*121, output_dim)
```

```
            self.fc1 = nn.Linear(2*output_dim, 1024)
            self.fc2 = nn.Linear(1024, 512)
            self.out = nn.Linear(512, self.n_output)

        def forward(self, data):
            x, edge_index, batch = data.x, data.edge_index, data.batch
            target = data.target

            x = self.conv1(x, edge_index)
            x = self.relu(x)
            x = self.conv2(x, edge_index)
            x = self.relu(x)

            x = self.conv3(x, edge_index)
            x = self.relu(x)
            x = global_max_pool(x, batch)
            x = self.relu(self.fc_g1(x))
            x = self.dropout(x)
            x = self.fc_g2(x)
            x = self.dropout(x)

            embedded_xt = self.embedding_xt(target)
            conv_xt = self.conv_xt_1(embedded_xt)
            xt = conv_xt.view(-1, 32 * 121)
            xt = self.fc1_xt(xt)

            xc = torch.cat((x, xt), 1)
            xc = self.fc1(xc)
            xc = self.relu(xc)
            xc = self.dropout(xc)
            xc = self.fc2(xc)
            xc = self.relu(xc)
            xc = self.dropout(xc)
            out = self.out(xc)
            return out
```

```
    model = GCNNet()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    loss_function = nn.CrossEntropyLoss()
    return model, optimizer, loss_function


def TrainTheModel(num_epochs=2):
    model, optimizer, loss_function = Model()
    train_loss = torch.zeros(num_epochs)
    valid_loss = torch.zeros(num_epochs)
    train_acc = torch.zeros(num_epochs)
    valid_acc = torch.zeros(num_epochs)
    best_acc = 0
    best_model = None
    for epoch in range(num_epochs):
        model.train()
        for _, data in enumerate(train_loader):
            optimizer.zero_grad()
            out = model(data)
            loss = loss_function(out, data.y)
            loss.backward()
            optimizer.step()
            train_loss[epoch] += loss.item()
            pred = out.max(1)[1]
            train_acc[epoch] += pred.eq(data.y).sum().item()
        train_loss[epoch] /= len(train_loader.dataset)
        train_acc[epoch] /= len(train_loader.dataset)

        model.eval()
        for data in val_loader:
            out = model(data)
            loss = loss_function(out, data.y)
            valid_loss[epoch] += loss.item()
            pred = out.max(1)[1]
            valid_acc[epoch] += pred.eq(data.y).sum().item()
        valid_loss[epoch] /= len(val_loader.dataset)
        valid_acc[epoch] /= len(val_loader.dataset)
```

```
        if (valid_acc[epoch] > best_acc):
            best_model = model
        print('Epoch: {:03d}, Train Loss: {:.5f}, Train Acc: {:.5f}, Val Loss: {:.5f}, Val Acc: {:.5f}'.format(
            epoch, train_loss[epoch], train_acc[epoch], valid_loss[epoch], valid_acc[epoch]))
    return best_model, train_loss, valid_loss, train_acc, valid_acc


model, train_loss, valid_loss, train_acc, valid_acc = TrainTheModel()

torch.save(model.state_dict(), 'dataset.pt')

model.load_state_dict(torch.load('dataset.pt'))
model.eval()
test_acc = 0
precision = 0
recall = 0
for data in test_loader:
    out = model(data)
    pred = out.max(1)[1]
    test_acc += pred.eq(data.y).sum().item()
    precision += precision_score(data.y, pred)
    recall += recall_score(data.y, pred)
test_acc /= len(test_loader.dataset)
precision /= len(test_loader.dataset)
recall /= len(test_loader.dataset)

print('Precision: {:.5f}'.format(precision))
print('Recall: {:.5f}'.format(recall))
print('Test Accuracy: {:.5f}'.format(test_acc))

f1_score = 2*(precision*recall)/(precision+recall)
print('F1 Score: {:.5f}'.format(f1_score))
```

11. APPENDIX

11.1 Referred Websites:

1. Wen Torng and Russ B. Altman; Graph Convolutional Neural Networks for Predicting Drug-Target Interactions (2019).

2. Hiroyuki Kurata, Sho Tsukiyama; ICAN: Interpretable cross-attention network for identifying drug and target protein interactions (2022).

3. Mohammad HussainAl‐Rabeah & Amir Lakizadeh; Prediction of drug‐drug interaction events using graph neural networks based feature extraction (2022).

4. Maha A. Thafar, Rawan S. Olayan, Somayah Albaradei, Vladimir B. Bajic ,Xin Gao, Takashi Gojobori, and Magbubah Essack; DTiGEMS+: drug–target interaction prediction using graph embedding, graph mining, and similarity-based techniques (2019).

5. Huu Ngoc Tran Tran, J. Joshua Thomas, and Nurul Hashimah Ahamed Hassain Malim; DeepNC: a framework for drug-target interaction prediction with graph neural networks (2022).

6. Hafez Eslami Manoochehri and Mehrdad Nourani; Drug-target interaction prediction using semi-bipartite graph model and deep learning (2020).

7. Mingjian Jiang, Zhen Li, Shugang Zhang, Shuang Wang, Xiaofeng Wang, Qing Yuan and Zhiqiang Wei; Drug–target affinity prediction using graph neural network and contact maps (2020).

8. Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka; HOW POWERFUL ARE GRAPH NEURAL NETWORKS? (2019).

9. Mengying Sun, Sendong Zhao, Coryandar Gilvary, Olivier Elemento, Jiayu Zhou and Fei Wang; Graph convolutional networks for computational drug development and discovery (2019).

10. Paola Ruiz Puentes, Laura Rueda‐Gensini, NataliaValderrama, Isabela Hernández, CristinaGonzález, Laura Daza, Carolina Muñoz‐Camargo, Juan C. Cruz, PabloArbeláez; Predicting target–ligand interactions with graph convolutional networks for interpretable pharmaceutical discovery (2022).

11. Ying Qian, Jian Wu and Qian Zhang; CAT-CPI: Combining CNN and transformer to learn

compound image features for predicting compound-protein interactions (2022).

12. Ruolan Chen, Xiangrong Liu, Shuting Jin, Jiawei Lin, and Juan Liu; Machine Learning for Drug-Target Interaction Prediction (2018).

13. Abbasi, Karim; Razzaghi, Parvin; Poso Antti; Ghanbari-Ara, Saber; Masoudi-Nejad, Ali; Deep Learning in Drug Target Interaction Prediction: Current and Future Perspectives (2021).

14. Lei Xu, Xiaoqing Ru, Rong Song; Application of Machine Learning for Drug-Target Interaction Prediction (2021).

15. Jeongtae Son, Dongsup Kim; Development of a graph convolutional neural network model for efficient prediction of protein-ligand binding affinities (2021).

16. Yang Li, Guanyu Qiao, Keqi Wang, Guohua Wang; Drug-Target interaction prediction via multi-channel graph neural networks (2022).

11.2 Reference for code:

https://towardsdatascience.com/coding-a-graph-convolutional-neural-network-gcn-using-keras-sequential-api-ec5211126875