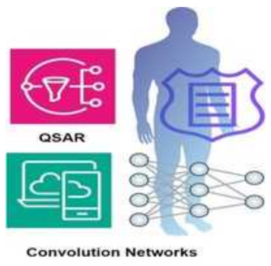
Dec 19, 2023

Graph Neural Network Framework for Web-Based Prediction of Protein-Ligand Docking Scores across multiple organs



DOI
dx.doi.org/10.17504/protocols.io.j8nlkoy9xv5r/v1

Anagha S Setlur[1], Vidya Niranjan[1], Arjun Balaji[2], Chandrashekar K[1]

[1]Department of Biotechnology, RV College of Engineering, Bangalore- 560059, affiliated to Visvesvaraya Technological University (VTU), Belagavi- 590018;
[2]Department of Electronics & Telecommunication, RV College of Engineering, Bangalore- 560059, affiliated to Visvesvaraya Technological University (VTU), Belagavi- 590018

Vidya Niranjan
R V College of Engineering

DOI: https://dx.doi.org/10.17504/protocols.io.j8nlkoy9xv5r/v1

Protocol Citation: Anagha S Setlur, Vidya Niranjan, Arjun Balaji, Chandrashekar K 2023. Graph Neural Network Framework for Web-Based Prediction of Protein-Ligand Docking Scores across multiple organs. **protocols.io** **https://dx.doi.org/10.17504/protocols.io.j8nlkoy9xv5r/v1**

Protocol status: Working
We use this protocol and it's working

Created: December 15, 2023

Last Modified: December 19, 2023

Protocol Integer ID: 92373

Keywords: QSAR, machine learning and deep learning, graph convolution networks, graph neural networks, data pre-processing, human organs, web-based predictions, effective molecular representation learning, molecular property prediction, field of molecular property prediction, graph neural network, graph neural network framework, graph neural network framework for web, molecular data, prediction of protein, ligand docking score, docking score between protein, graph convolution network, application of graph neural network, ligand docking scores across multiple organ, assessing molecular interaction, deep learning, molecular interaction, graph format, smiles representation, protein, drug design, using smiles representation, based drug design, molecule, qsar, docking score, trained model

Abstract

Estimating the docking score between proteins and drugs is very important in the application of structure-based drug design. This project explores the application of Graph Neural networks (GNN) in the field of molecular property prediction using SMILES representation, the trained models are then deployed on a web-based platform for broader accessibility and use. The primary dataset utilized in this study includes molecular data represented by MolPort IDs and associated docking scores, which are critical in assessing molecular interactions. A significant aspect of this project is data preprocessing, where each molecule, initially represented as a SMILES string, is converted into a graph format. Effective molecular representation learning is pivotal to facilitate molecular property prediction. Models are then evaluated based on various performance metrics and deployed on the web-based platform.

Keywords: QSAR, machine and deep learning, graph convolution networks, graph neural networks, data pre-processing, human organs, web-based predictions

Guidelines

QSAR modeling should be performed for each protein under each organ first. The ligand IDs and SMILES structures are the preferred columns to be present in the analytical dataset.

Troubleshooting

Safety warnings

NA

Ethics statement

None.

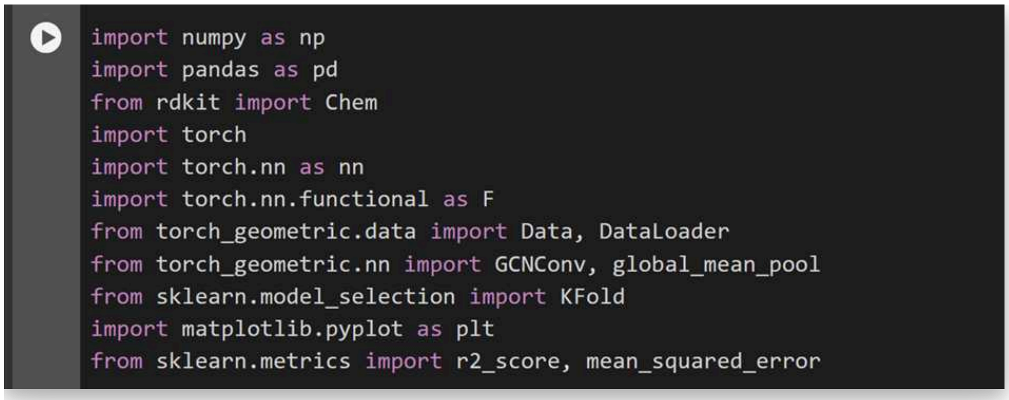
Before start

Check system compatibility to run pre-processing of models and GCN/hybrid GCN models.

IMPORTING LIBRARIES

1 **Import all necessary libraries**

Ensure the installation and importation of all the necessary libraries needed for both the data preprocessing and the model training and evaluation. Provided below is a screenshot of the required libraries to be imported.



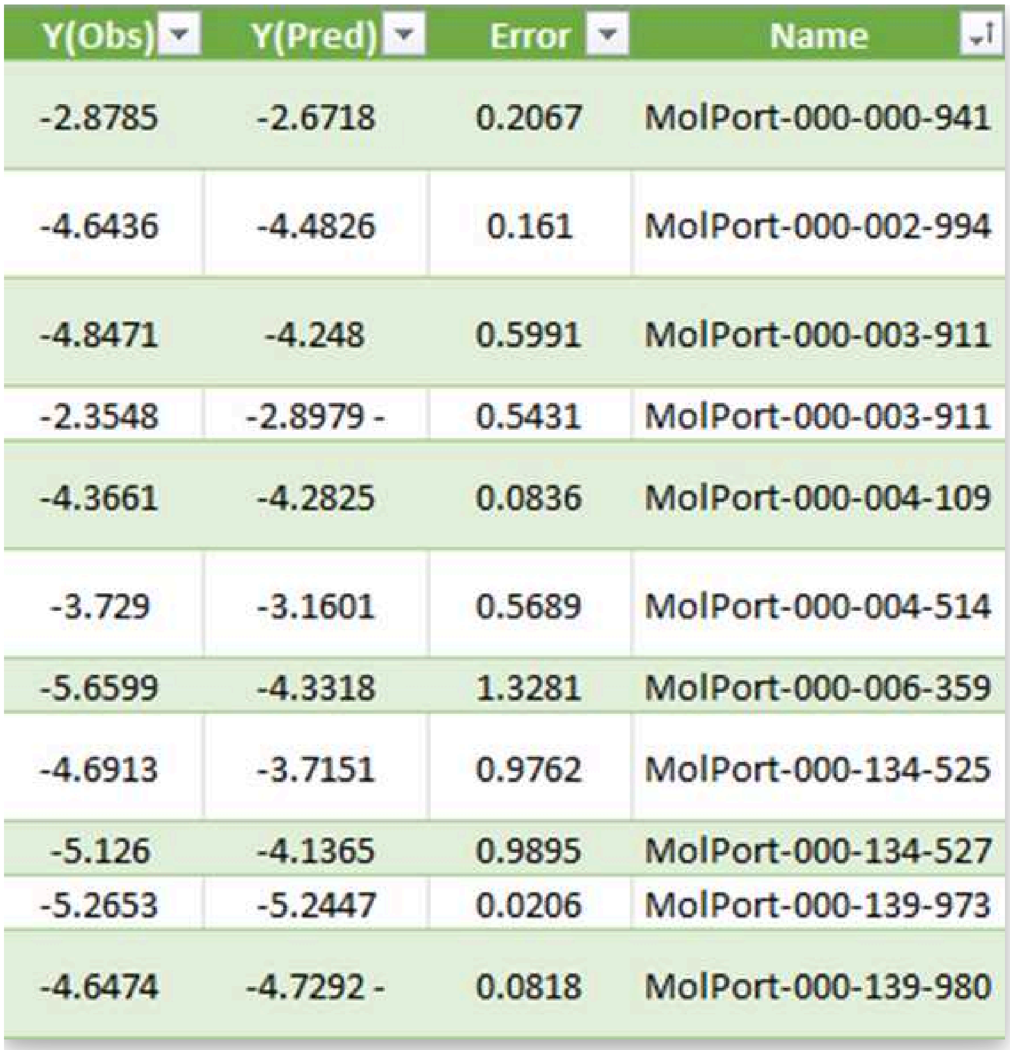
```
import numpy as np
import pandas as pd
from rdkit import Chem
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.data import Data, DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error
```

Importing required libraries

DATASET CREATION

2 In the present scenario, Quantitative Structure Activity Relationship (QSAR) data generated from Schrodinger Maestro was used for dataset creation. QSAR models were first generated for specific proteins and by taking a set of ligands from MolPort.

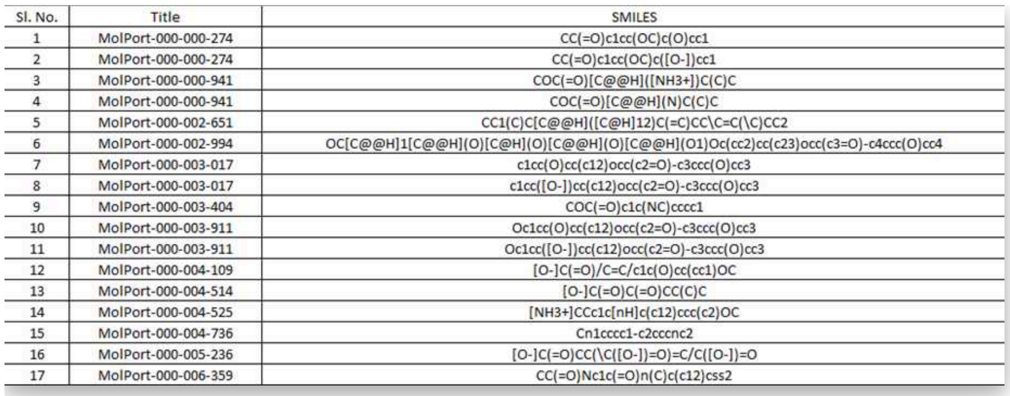
Taking an example for Brain, O14672. Here, Y(Obs) is the docking score. This dataset has the MolPort IDs and the docking scores obtained from QSAR modeling data.



Y(Obs)	Y(Pred)	Error	Name
-2.8785	-2.6718	0.2067	MolPort-000-000-941
-4.6436	-4.4826	0.161	MolPort-000-002-994
-4.8471	-4.248	0.5991	MolPort-000-003-911
-2.3548	-2.8979 -	0.5431	MolPort-000-003-911
-4.3661	-4.2825	0.0836	MolPort-000-004-109
-3.729	-3.1601	0.5689	MolPort-000-004-514
-5.6599	-4.3318	1.3281	MolPort-000-006-359
-4.6913	-3.7151	0.9762	MolPort-000-134-525
-5.126	-4.1365	0.9895	MolPort-000-134-527
-5.2653	-5.2447	0.0206	MolPort-000-139-973
-4.6474	-4.7292 -	0.0818	MolPort-000-139-980

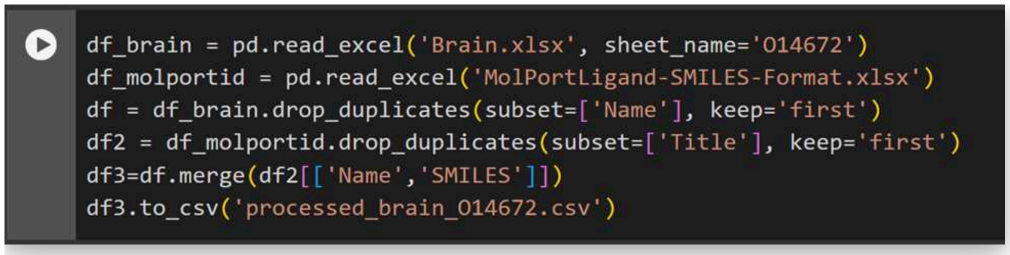
2.1 Creation of analytical dataset

Using the second dataset containing MolPort IDs and the SMILES string. An analytical dataset was created.



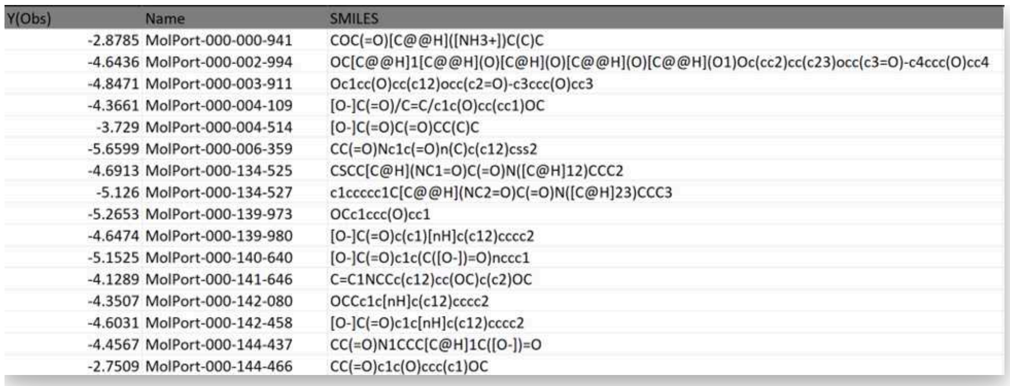
Sl. No.	Title	SMILES
1	MolPort-000-000-274	CC(=O)c1cc(OC)c(O)cc1
2	MolPort-000-000-274	CC(=O)c1cc(OC)c([O-])cc1
3	MolPort-000-000-941	COC(=O)[C@@H]([NH3+])C(C)C
4	MolPort-000-000-941	COC(=O)[C@@H](N)C(C)C
5	MolPort-000-002-651	CC1(C)C[C@@H]([C@H]12)C(=C)CC\C=C(\C)CC2
6	MolPort-000-002-994	OC[C@@H]1[C@@H](O)[C@H](O)[C@@H](O)[C@@H](O1)Oc(cc2)cc(c23)occ(c3=O)-c4ccc(O)cc4
7	MolPort-000-003-017	c1cc(O)cc(c12)occ(c2=O)-c3ccc(O)cc3
8	MolPort-000-003-017	c1cc([O-])cc(c12)occ(c2=O)-c3ccc(O)cc3
9	MolPort-000-003-404	COC(=O)c1c(NC)cccc1
10	MolPort-000-003-911	Oc1cc(O)cc(c12)occ(c2=O)-c3ccc(O)cc3
11	MolPort-000-003-911	Oc1cc([O-])cc(c12)occ(c2=O)-c3ccc(O)cc3
12	MolPort-000-004-109	[O-]C(=O)/C=C/c1c(O)cc(cc1)OC
13	MolPort-000-004-514	[O-]C(=O)C(=O)CC(C)C
14	MolPort-000-004-525	[NH3+]CCc1c[nH]c(c12)ccc(c2)OC
15	MolPort-000-004-736	Cn1cccc1-c2cccnc2
16	MolPort-000-005-236	[O-]C(=O)CC(\C([O-])=O)=C/C([O-])=O
17	MolPort-000-006-359	CC(=O)Nc1c(=O)n(C)c(c12)css2

The following is performed to prepare an analytical dataset:



```
df_brain = pd.read_excel('Brain.xlsx', sheet_name='014672')
df_molportid = pd.read_excel('MolPortLigand-SMILES-Format.xlsx')
df = df_brain.drop_duplicates(subset=['Name'], keep='first')
df2 = df_molportid.drop_duplicates(subset=['Title'], keep='first')
df3=df.merge(df2[['Name','SMILES']])
df3.to_csv('processed_brain_014672.csv')
```

The processed dataset looks as follows:



Y(Obs)	Name	SMILES
-2.8785	MolPort-000-000-941	COC(=O)[C@@H]([NH3+])C(C)C
-4.6436	MolPort-000-002-994	OC[C@@H]1[C@@H](O)[C@H](O)[C@@H](O)[C@@H](O1)Oc(cc2)cc(c23)occ(c3=O)-c4ccc(O)cc4
-4.8471	MolPort-000-003-911	Oc1cc(O)cc(c12)occ(c2=O)-c3ccc(O)cc3
-4.3661	MolPort-000-004-109	[O-]C(=O)/C=C/c1c(O)cc(cc1)OC
-3.729	MolPort-000-004-514	[O-]C(=O)C(=O)CC(C)C
-5.6599	MolPort-000-006-359	CC(=O)Nc1c(=O)n(C)c(c12)css2
-4.6913	MolPort-000-134-525	CSCC[C@H](NC1=O)C(=O)N([C@H]12)CCC2
-5.126	MolPort-000-134-527	c1ccccc1C[C@@H](NC2=O)C(=O)N([C@H]23)CCC3
-5.2653	MolPort-000-139-973	OCc1ccc(O)cc1
-4.6474	MolPort-000-139-980	[O-]C(=O)c(c1)[nH]c(c12)cccc2
-5.1525	MolPort-000-140-640	[O-]C(=O)c1c(C([O-])=O)nccc1
-4.1289	MolPort-000-141-646	C=C1NCCc(c12)cc(OC)c(c2)OC
-4.3507	MolPort-000-142-080	OCCc1c[nH]c(c12)cccc2
-4.6031	MolPort-000-142-458	[O-]C(=O)c1c[nH]c(c12)cccc2
-4.4567	MolPort-000-144-437	CC(=O)N1CCC[C@H]1C([O-])=O
-2.7509	MolPort-000-144-466	CC(=O)c1c(O)ccc(c1)OC

DATA PRE-PROCESSING

3 **SMILES to graph conversion**

Data preprocessing is a pivotal step in this model. Each molecule represented by a SMILES string is converted into a graph, with atoms as nodes and chemical bonds as edges. This graph representation is essential for the GNN to accurately interpret molecular structures.

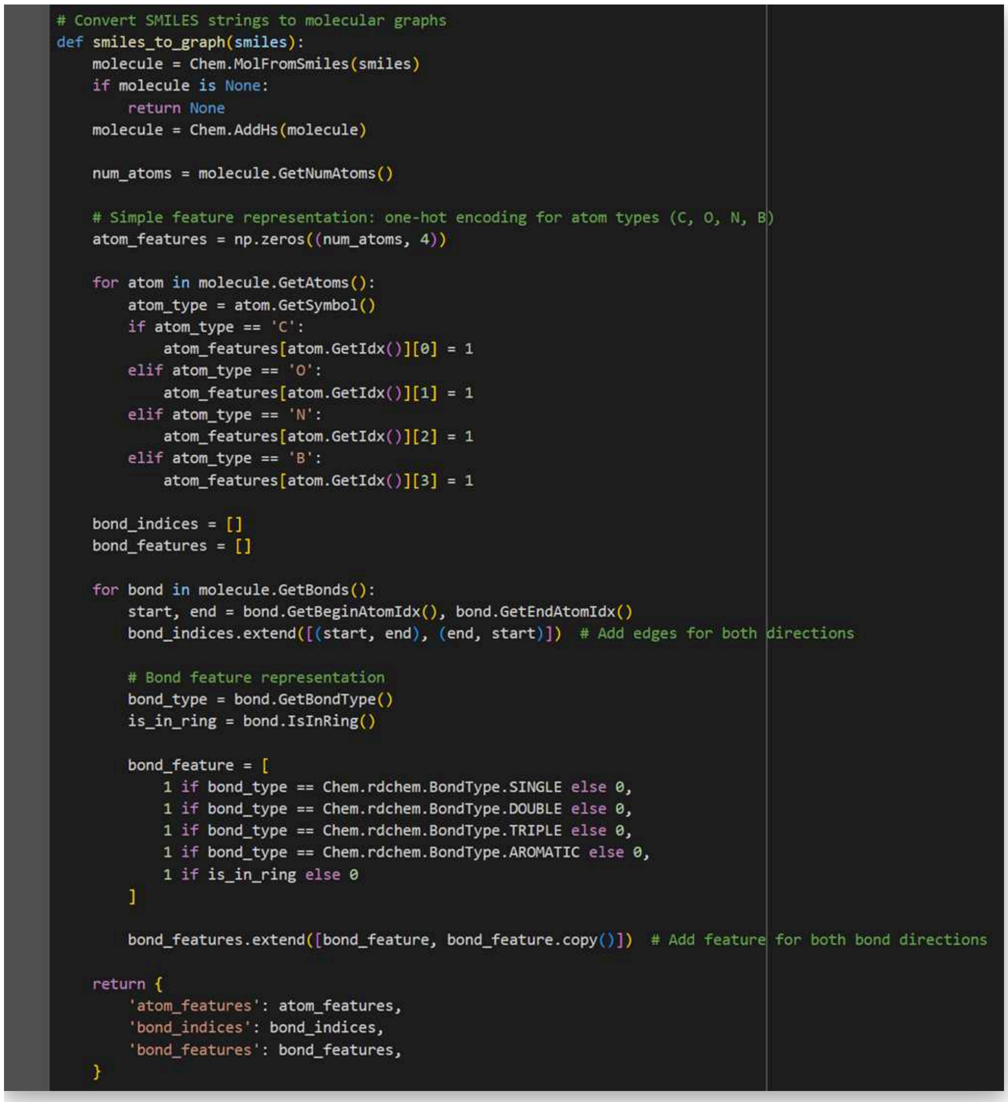
Feature Representation

- **Atom Features:** Each atom is represented by a one-hot encoded feature vector, indicating the atom type. The model considers four types of atoms (C, O, N, B), leading to a 4-dimensional feature vector for each atom.
- **Bond Features:** Bonds are characterized by their type (single, double, triple, aromatic) and their inclusion in a ring structure. Each bond is represented by a 5-dimensional feature vector.

	A	B
	Feature	Dimensions
	One-hot encoding of atom types (C, O, N, B)	4
	Edge features for bond types (single, double, triple, aromatic)	4
	Edge features for bond presence in a ring structure	1
	Atom features for atom presence in a ring structure	1
	Bond indices for atom connectivity	2 per bond

3.1 **Using RDKit library for feature representation**

So, to represent all these features, we utilize the functionalities of the RDKit library. The function converts a SMILES string into a molecular graph, encoding atom types using one-hot encoding and representing bonds with their types and ring membership.



```python
# Convert SMILES strings to molecular graphs
def smiles_to_graph(smiles):
    molecule = Chem.MolFromSmiles(smiles)
    if molecule is None:
        return None
    molecule = Chem.AddHs(molecule)

    num_atoms = molecule.GetNumAtoms()

    # Simple feature representation: one-hot encoding for atom types (C, O, N, B)
    atom_features = np.zeros((num_atoms, 4))

    for atom in molecule.GetAtoms():
        atom_type = atom.GetSymbol()
        if atom_type == 'C':
            atom_features[atom.GetIdx()][0] = 1
        elif atom_type == 'O':
            atom_features[atom.GetIdx()][1] = 1
        elif atom_type == 'N':
            atom_features[atom.GetIdx()][2] = 1
        elif atom_type == 'B':
            atom_features[atom.GetIdx()][3] = 1

    bond_indices = []
    bond_features = []

    for bond in molecule.GetBonds():
        start, end = bond.GetBeginAtomIdx(), bond.GetEndAtomIdx()
        bond_indices.extend([(start, end), (end, start)])  # Add edges for both directions

        # Bond feature representation
        bond_type = bond.GetBondType()
        is_in_ring = bond.IsInRing()

        bond_feature = [
            1 if bond_type == Chem.rdchem.BondType.SINGLE else 0,
            1 if bond_type == Chem.rdchem.BondType.DOUBLE else 0,
            1 if bond_type == Chem.rdchem.BondType.TRIPLE else 0,
            1 if bond_type == Chem.rdchem.BondType.AROMATIC else 0,
            1 if is_in_ring else 0
        ]

        bond_features.extend([bond_feature, bond_feature.copy()])  # Add feature for both bond directions

    return {
        'atom_features': atom_features,
        'bond_indices': bond_indices,
        'bond_features': bond_features,
    }
```

Using RDKit for feature representation

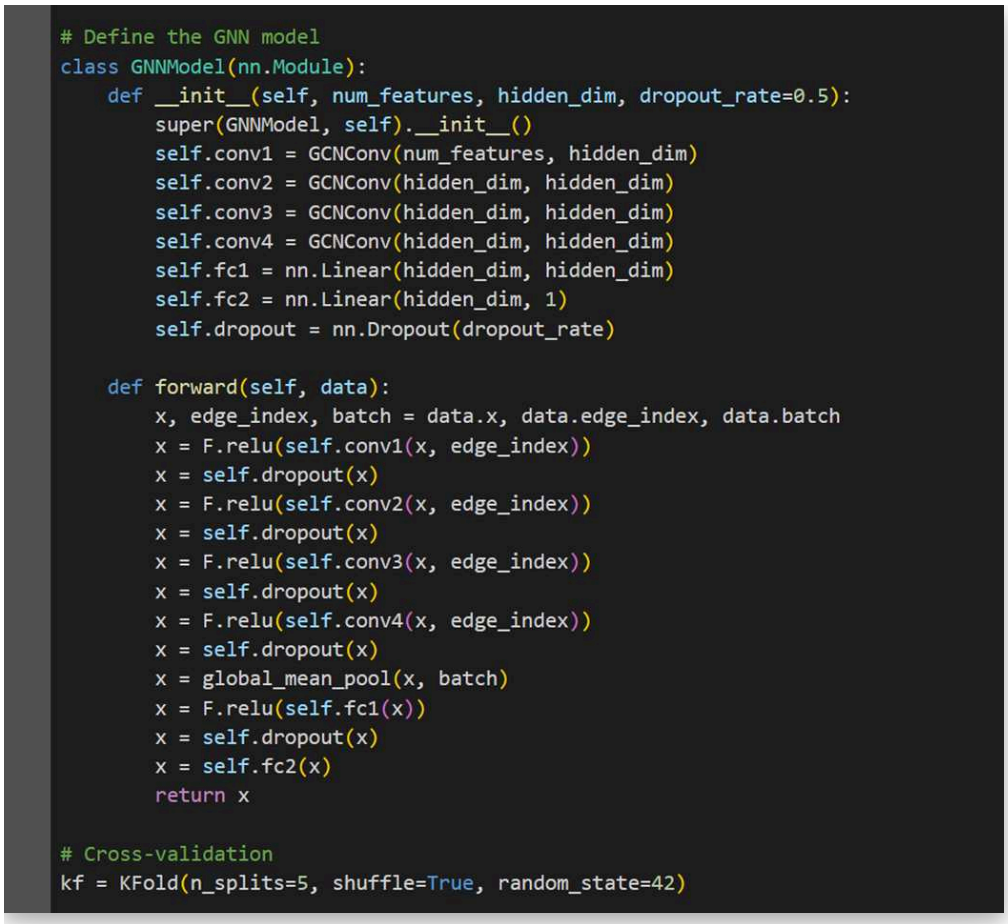
MODEL TRAINING AND EVALUATION

4 **Model defining and training**

Define the models and train with early stopping along with appropriate parameters.

4.1 **MODEL 1- GRAPH CONVOLUTION NETWORK (GCN)**

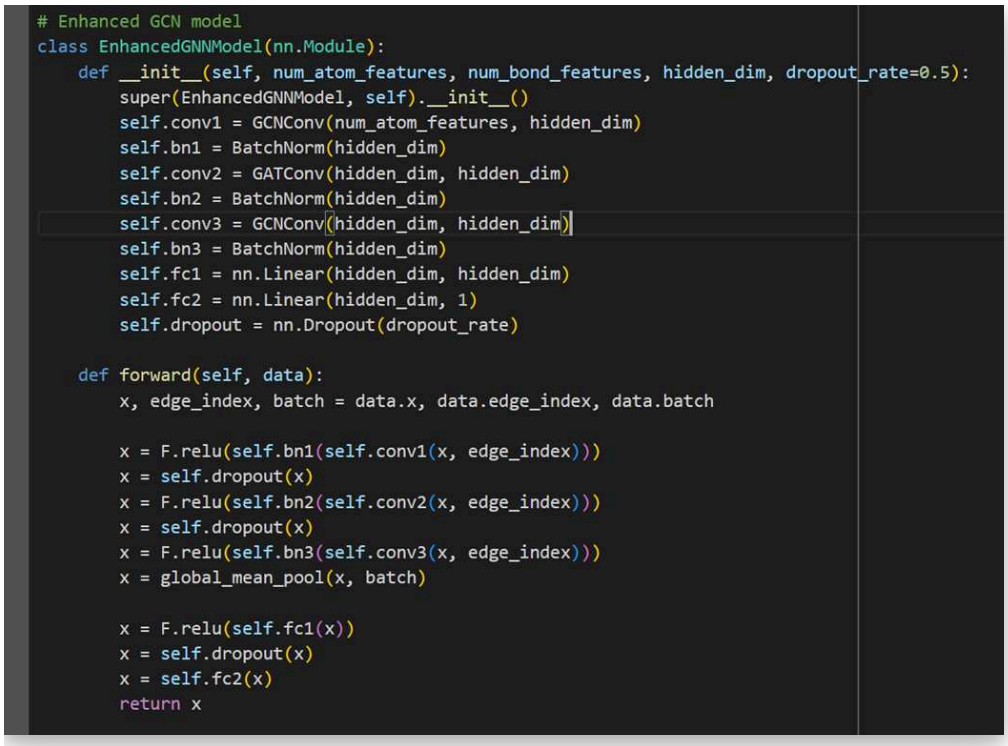
The first model we explore is a Graph Convolution Network (GCN) with 2 convolution layers.



```
# Define the GNN model
class GNNModel(nn.Module):
    def __init__(self, num_features, hidden_dim, dropout_rate=0.5):
        super(GNNModel, self).__init__()
        self.conv1 = GCNConv(num_features, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.conv3 = GCNConv(hidden_dim, hidden_dim)
        self.conv4 = GCNConv(hidden_dim, hidden_dim)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, 1)
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = F.relu(self.conv1(x, edge_index))
        x = self.dropout(x)
        x = F.relu(self.conv2(x, edge_index))
        x = self.dropout(x)
        x = F.relu(self.conv3(x, edge_index))
        x = self.dropout(x)
        x = F.relu(self.conv4(x, edge_index))
        x = self.dropout(x)
        x = global_mean_pool(x, batch)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

4.2 **MODEL 2- HYBRID GCN**

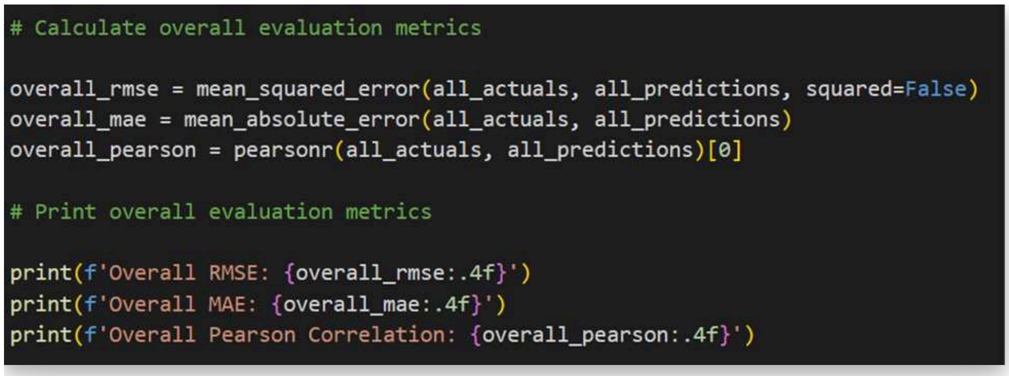
The second model we explore is a hybrid GCN model:



```python
# Enhanced GCN model
class EnhancedGNNModel(nn.Module):
    def __init__(self, num_atom_features, num_bond_features, hidden_dim, dropout_rate=0.5):
        super(EnhancedGNNModel, self).__init__()
        self.conv1 = GCNConv(num_atom_features, hidden_dim)
        self.bn1 = BatchNorm(hidden_dim)
        self.conv2 = GATConv(hidden_dim, hidden_dim)
        self.bn2 = BatchNorm(hidden_dim)
        self.conv3 = GCNConv(hidden_dim, hidden_dim)
        self.bn3 = BatchNorm(hidden_dim)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, 1)
        self.dropout = nn.Dropout(dropout_rate)

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

        x = F.relu(self.bn1(self.conv1(x, edge_index)))
        x = self.dropout(x)
        x = F.relu(self.bn2(self.conv2(x, edge_index)))
        x = self.dropout(x)
        x = F.relu(self.bn3(self.conv3(x, edge_index)))
        x = global_mean_pool(x, batch)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

4.3 **5-fold cross validation**

Utilizing 5-Fold cross-validation for training enhancing its robustness and reliability. This method ensured a comprehensive evaluation by systematically partitioning the data into distinct subsets for both training and validation.

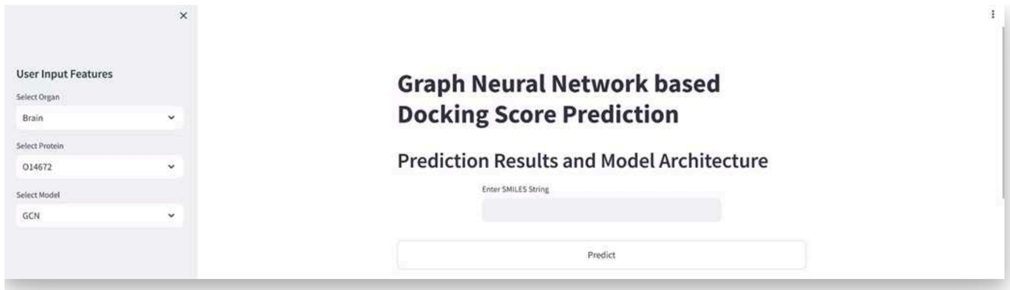


```python
# Calculate overall evaluation metrics

overall_rmse = mean_squared_error(all_actuals, all_predictions, squared=False)
overall_mae = mean_absolute_error(all_actuals, all_predictions)
overall_pearson = pearsonr(all_actuals, all_predictions)[0]

# Print overall evaluation metrics

print(f'Overall RMSE: {overall_rmse:.4f}')
print(f'Overall MAE: {overall_mae:.4f}')
print(f'Overall Pearson Correlation: {overall_pearson:.4f}')
```

The model's performance was further evaluated using metrics like Root Mean Squared Error (RMSE) and Mean Average Error(MAE), providing insights into its predictive accuracy and overall performance.

PICKING THE BEST MODEL AND UPLOADING IN REPOSITORY

5 The best possible model was picked and the weights were saved. Then, these weights were uploaded onto the Streamlit repository.


User Input Features
Select Organ: Brain
Select Protein: O14672
Select Model: GCN

Graph Neural Network based Docking Score Prediction

Prediction Results and Model Architecture

These same steps were repeated across different proteins, datasets and models to integrate all models from each human organ into a single platform.

CONCLUSION

6 This protocol briefs the steps required to integrate all predicted QSAR data from each organ into a single, all-in-one platform for all human organs and proteins associated with them, to enable users to provide a SMILES structure and estimate the predicted docking score after mapping with the integrated models. Data pre-processing is the primary step in this protocol, followed by creation of analytical dataset for conversion into graphs. Advanced machine and deep learning technique called the graph convolution network (GCN) is shown as model 1, where high dimensional data is converted to low dimensional data and the graphs are correlated to the target variables (in this case, docking scores). The hybrid model, shown as model 2, also adds an additional concept of attention mechanism, that employs positional encoding along with traditional GCN. The web-application allows users to choose which model to utilise for their prediction. This protocol allows for the direct binding affinity predictions of small molecules to important proteins in the human organs, thereby, providing an overall safety information on the small molecules.

ACKNOWLEDGEMENTS

7 The authors thank Mr. Akshay Uttarkar for providing inputs throughout.

Protocol references

1. Kaplan Z, Ehrlich S, Leswing K (2021) Benchmark study of DeepAutoQSAR, ChemProp, and DeepPurpose on the ADMET subset of the Therapeutic Data Commons. **https://newsite.schrodinger.com/life-science/learn/white-papers/benchmark-study-deepautoqsar-chemprop-and-deeppurpose-admet-subset-therapeutic-data/**

2. Gion K, Gattani S, Kaplan Z (2022) DeepAutoQSAR hardware benchmark. **https://newsite.schrodinger.com/materials-science/learn/white-papers/deepautoqsar-hardware-benchmark/**

3. Schrödinger Release 2023-4: DeepAutoQSAR, Schrödinger, LLC, New York, NY, 2023.

4. https://www.molport.com/shop/index

5. Wu F, Souza A, Zhang T, Fifty C, Yu T, Weinberger K. Simplifying graph convolutional networks. International conference on machine learning 2019 May 24 (pp. 6861-6871). PMLR.

6. Javeed A. A hybrid attention mechanism for multi-target entity relation extraction using graph neural networks. Machine Learning with Applications. 2023 Mar 15;11:100444.