Youssef Ezz Eldeen Ezzat

MASTER'S DEGREE IN SECURITY ENGINEERING AND ARTIFICAL INTELLIGENCE (MESIIA)

MASTER'S Thesis FINAL PROJECT

Directed by Prof. Francesc Serratosa

Prediction of drug–protein binding affinity using existing graph convolutional neural network models on URV In-house dataset

Tarragona
June 10, 2024

Contents

1 Introduction

A virus encodes one or more proteases which are enzymes that spur the formation of new protein products, thus play crucial roles in virus replication , and are important targets for the design and development of potent antiviral agents or drugs. Binding affinity is the strength of the binding interaction between a single biomolecule (e.g., a virus protein) to its ligand or binding partner (e.g., a drug). it is a key to appreciating the intermolecular interactions driving biological processes and measured as part of the drug discovery process to help design drugs that bind their targets selectively and specifically.

The development of new drugs is costly, time consuming and often accompanied with safety issues. Drug repurposing can avoid the expensive and lengthy process of drug development by finding new uses for already approved drugs. In order to repurpose drugs effectively, it is useful to know which proteins are targeted by which drugs which is the definition of Drug-Target-Affinity(DTA)[1]

there is a strong motivation to build computational models that can estimate the interaction strength of new drug–target pairs based on previous drug–target experiments.

the DTA prediction problem as a regression task where the input is a drug–target pair and the output is a continuous measurement of binding affinity for that pair.

2 Aim

The aim of this thesis is to make use of four existing convolutional neural network(CNN) models following the paper in [1] - to take advantage of the fact that they represents drugs as graphs and use graph neural networks to predict DTA- in order to train a neural network on a new small in-house URV dataset of sample size 332 patterns. but since the CNN models typically require large datasets for training like the datasets davis and kiba that have sample sizes of 30058 (25047 train + 5011 test) and 118257 (98547 train + 19710 test) respectively used to train the CNN models in [1] the approach taken will be:

- Apply k-folding with k = 10 to get 10 different combinations of train and test patterns for URV dataset

Figure 1: k-folding.

the GIT repository of the code [2] used in the referenced paper [1] was forked to Git repository [3] to experiment with the code in the referenced paper and obtain the results. code refactoring was done to run experiments in python notebooks instead of terminals

3 URV dataset

3.1 protein representation

the protein data are initially provided as **Protein Data Bank(PDB)** files. PDB is a file format used to store 3D structural information about proteins. It is one of the most widely used formats for representing and exchanging structural data in bioinformatics and structural biology. PDB files contain atomic coordinates of atoms in a molecule, along with metadata and additional information such as experimental methods used to determine the structure.

the PDB file name is named after the protein ID. if a PDB file e.g. **6M2N_protein.pdb** is open with any text editor e.g. notepad++, the metadata including its ID, name, type are in the header and title tags as shown in figure 2 for protein ID **6M2N**

6M2N_protein.pdb

```
1  HEADER    VIRAL PROTEIN                           28-FEB-20   6M2N
2  TITLE     SARS-COV-2 3CL PROTEASE (3CL PRO) IN COMPLEX WITH A NOVEL INHIBITOR
3  COMPND    MOL_ID: 1;
4  COMPND   2 MOLECULE: 3C-LIKE PROTEINASE;
5  COMPND   3 CHAIN: A, B, C, D;
6  COMPND   4 SYNONYM: 3CL-PRO, 3CLP, MAIN PROTEASE, MPRO, NON-STRUCTURAL PROTEIN
7  COMPND   5 5, NSP5, SARS CORONAVIRUS MAIN PROTEINASE;
8  COMPND   6 EC: 3.4.22.69;
9  COMPND   7 ENGINEERED: YES
10 SOURCE    MOL_ID: 1;
11 SOURCE   2 ORGANISM_SCIENTIFIC: SEVERE ACUTE RESPIRATORY SYNDROME CORONAVIRUS
12 SOURCE   3 2;
13 SOURCE   4 ORGANISM_COMMON: 2019-NCOV;
14 SOURCE   5 ORGANISM_TAXID: 2697049;
15 SOURCE   6 EXPRESSION_SYSTEM: ESCHERICHIA COLI BL21(DE3);
16 SOURCE   7 EXPRESSION_SYSTEM_TAXID: 469008;
17 SOURCE   8 EXPRESSION_SYSTEM_STRAIN: BL21(DE3)
18 KEYWDS    SARS-COV-2, 3CL PRO, VIRAL PROTEIN
19 EXPDTA    X-RAY DIFFRACTION
20 AUTHOR    H.X.SU,W.F.ZHAO,M.J.LI,H.XIE,Y.C.XU
```

Figure 2: PDB file example.

more informtion can be viewed at the PDB website [4] for the same protein ID **6M2N**

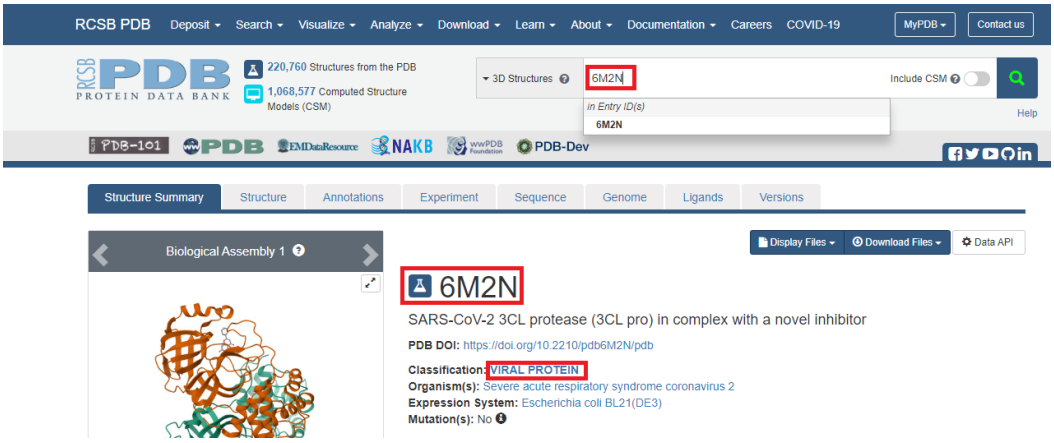


Figure 3: PDB file example.

a protein can be either Single Chain or Multi-Chain, so it can be composed of one chain of amino acids(sequence) or more usually two, we pick the longest sequence as the representation for the protein this is done using **Bio.PDB** module in the **Biopython** library that provides tools for working with Protein Data Bank (PDB) files in python. classes used are:

- **PDBParser** used to parse PDB files, create a structure object and read the atomic coordinates and other structural information from a PDB file and converts it into a hierarchical structure object, which can be easily manipulated and analyzed.
- **PDBBuilder** used to identify and construct polypeptides (chains of amino acids) from a structure object (such as a protein structure obtained from a PDB file)

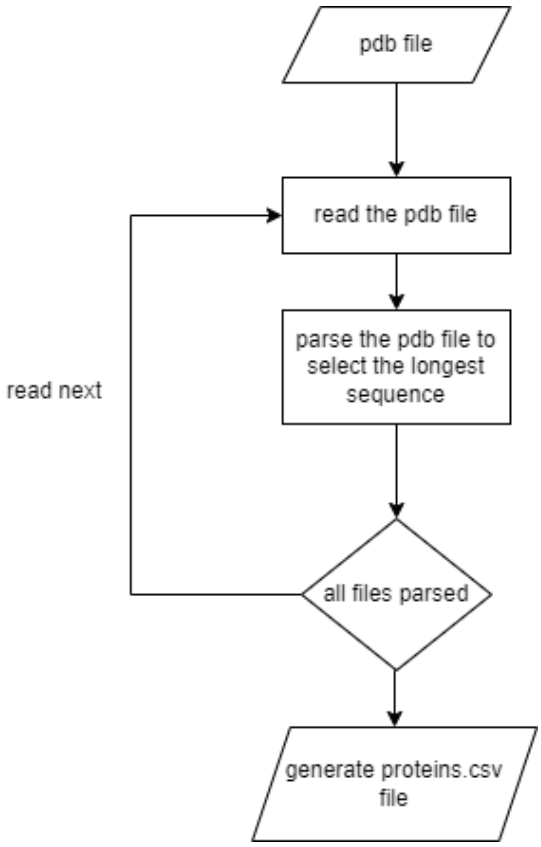


Figure 4: PDB file example.

3.2 drug(ligand) representation

the ligand data are provided as **Structure Data file(SDF)** file format used for representing chemical compounds and their associated data. It is primarily used to store information about molecules, including their structure and various properties, in a structured way.

the SDF file name is named after the protein ID with which the ligand is paired. consider SDF file **6M2N_ligand.sdf** shown in fig 5 where its name does not indicate any information about the ligand but suggests it will be paired with previous protein to get information about the ligand itself, it can be open using a text editor e.g. notepad++ and at the end of the file information that identify the ligand can be found e.g. **chemical formula** and **Simplified Molecular Input Line Entry System(SMILES)** notation which is is a chemical notation that allows a user to represent a chemical structure in a way that can be used by the computer.

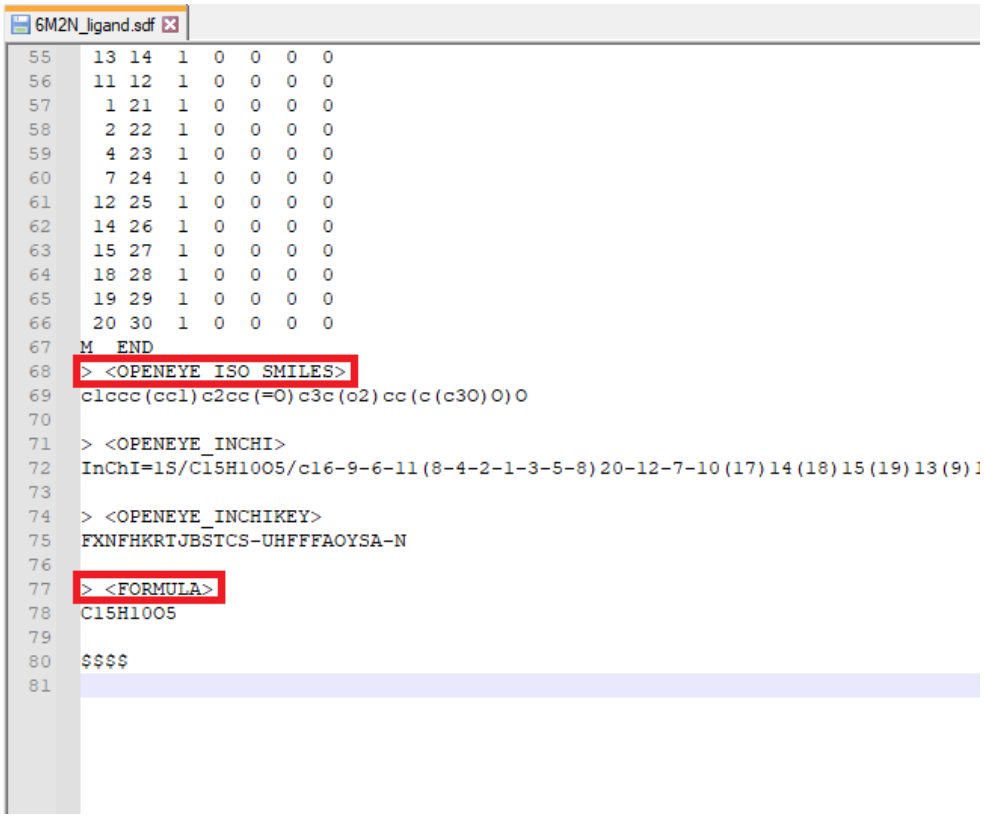


Figure 5: SDF file example.

while the chemical formula may not be unique, the SMILES notation is always unique. so it can be used in many websites e.g. the chemspider website[6] to search for more information about the ligand

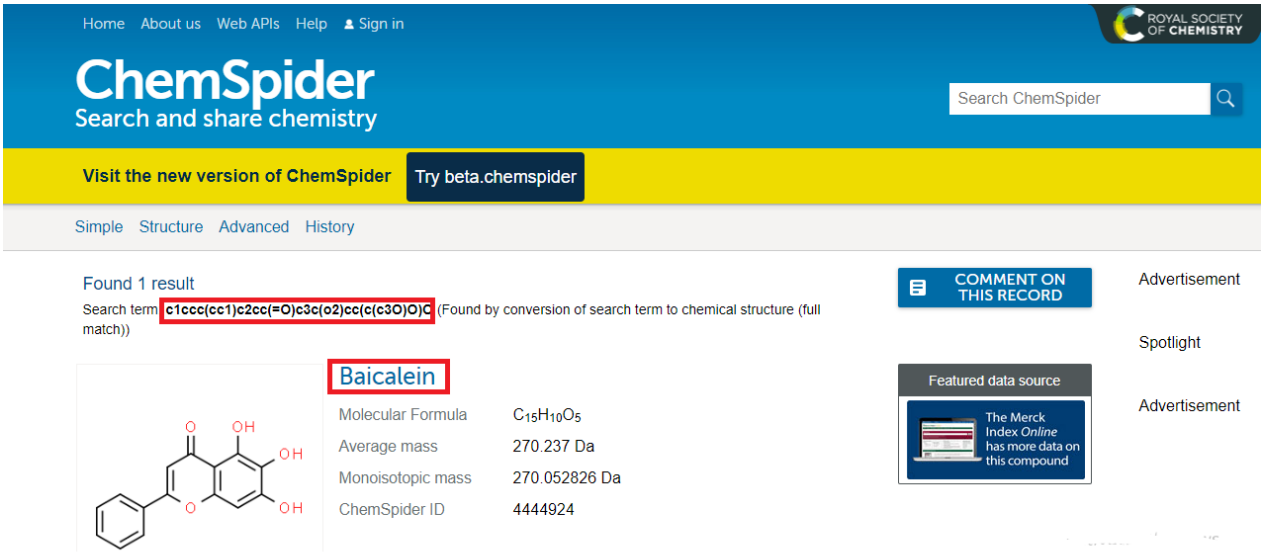


Figure 6: SDF website.

An SDF ligand file is read and first the molecule is validated(sanitized) to check if error exist in the structure, if the structure is valid the SMILES notation is extracted and appended to valid ligands SDF file

else it is excluded and added to invalid ligands SDF file.

The **rdkit.Chem** package [7] was used to read the molecule using Chem.SDMolSupplier function, validate it using Chem.SanitizeMol and convert it to SMILES notation using function Chem.MolToSmiles

- **Chem.SDMolSupplier** used to read the molecule.
- **Chem.SanitizeMol** used to validate the molecule
- **Chem.MolToSmiles** obtain the SMILES notation of the molecule

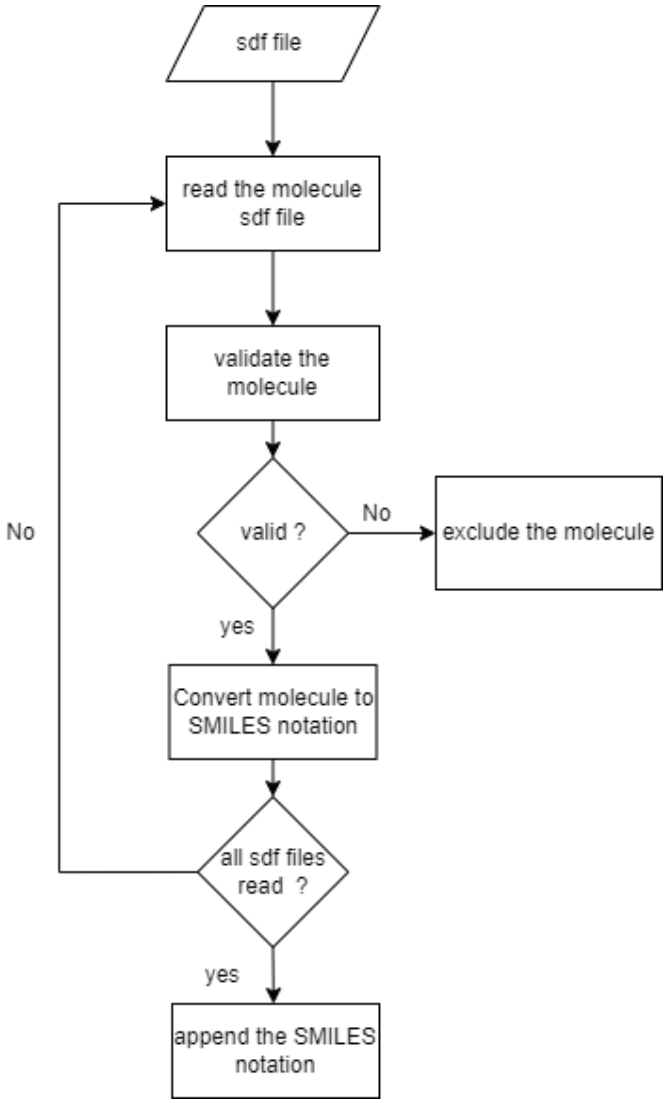


Figure 7: SDF workflow.

3.3 affinity representation

affinity is the continouos variable that represents the interaction between the ligand and the protein smaller value means less affinity and higher value indicates strong affinity.

affinity values are provided in csv file linked with protein ID and the protein ID in turn is found in the names of the files of both ligand and the protein

the following figures explain statistics of the affinity values of the 322 pairs, note that jitter was added to one version of the box plot for the sake of clarity

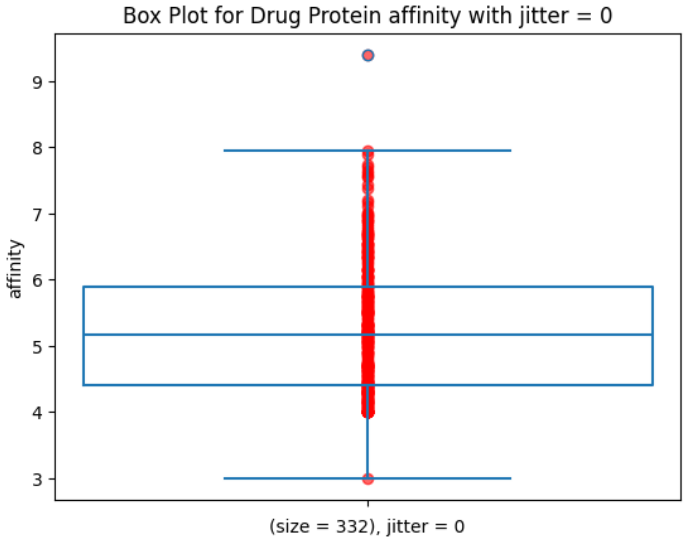


Figure 8:

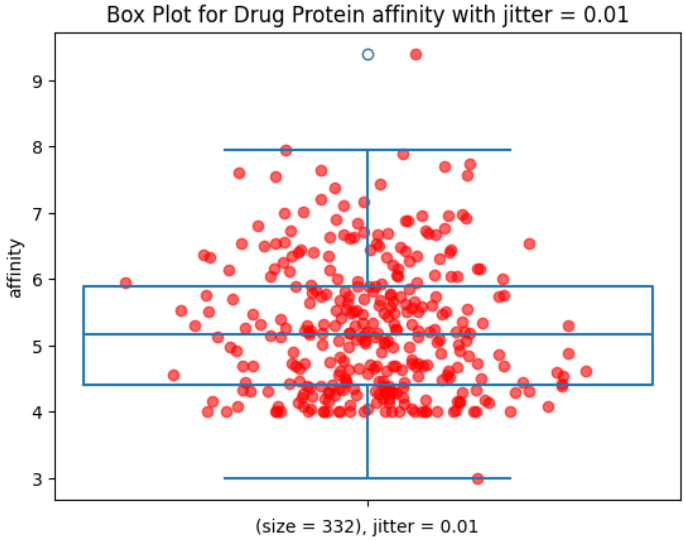
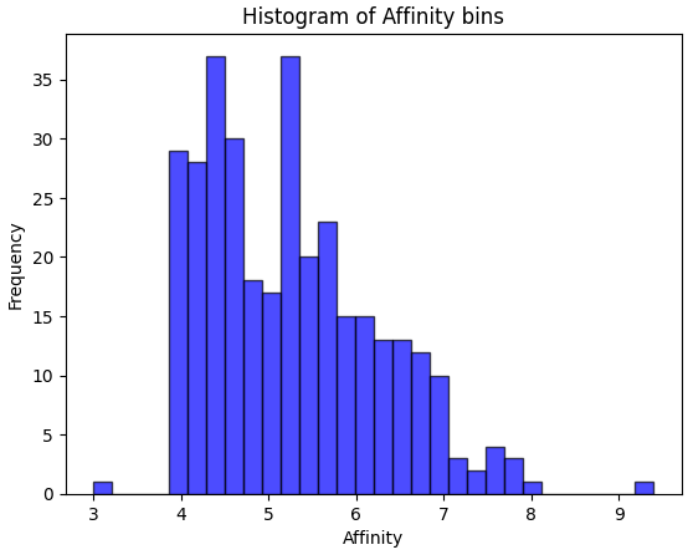
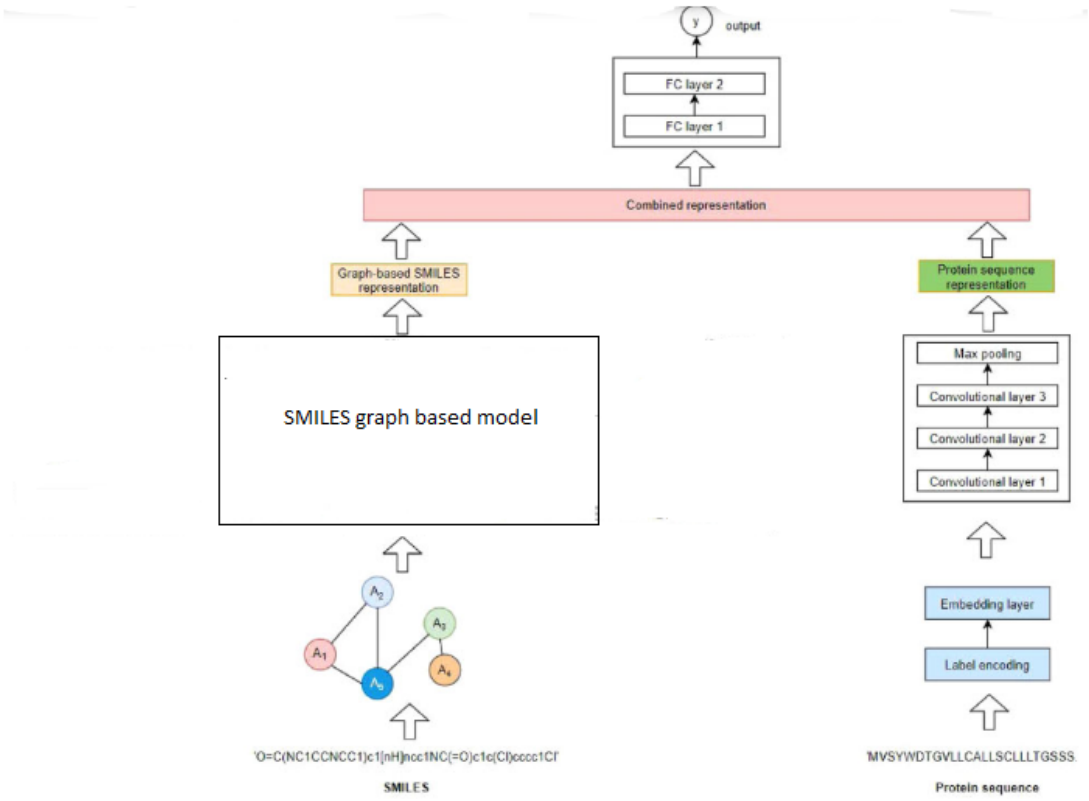


Figure 9:



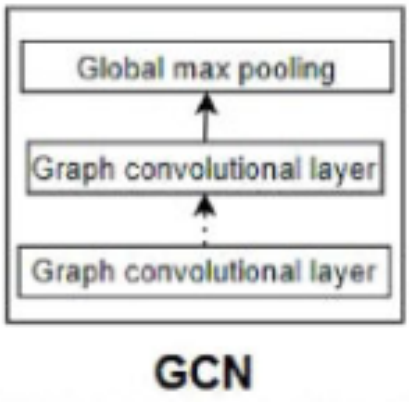
4 Network architecture

The overall architecture combines the information from the molecular graph and the protein sequence to make a prediction. it is a PyTorch implementation of a Graph Convolutional Network (GCN) based model



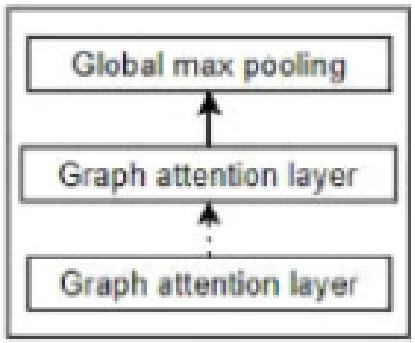
The architecture consists of two main branches where the SMILES graph based branch could be replaced by any of the four SMILES graph branch models :

- **SMILES graph branch**: operates on the molecular graph representation of the input SMILES. could be any of four models
  - **GCN model**
    1. It uses three GCN layers (GCNConv) to extract features from the graph. The number of filters and feature dimensions are increased in the subsequent layers.
    2. After the GCN layers, the extracted features are passed through two fully connected layers (fc_g1 and fc_g2) to obtain a fixed-size output representation.



  - **GAT model**
    1. The model starts with two GAT layers, gcn1 and gcn2, which operate on the input graph data.
    2. The first GAT layer, gcn1, takes the initial node features (num_features_xd) and outputs a representation with 10 heads, effectively increasing the feature dimensionality by a factor of 10.

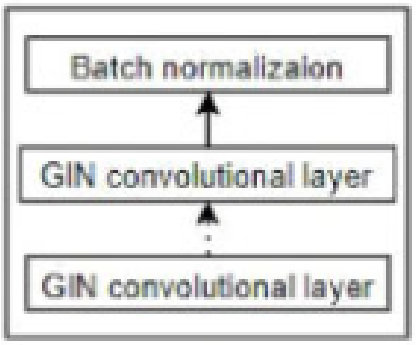
3. The second GAT layer, gcn2, then reduces the feature dimensionality to output_dim.
4. After the GAT layers, a fully connected layer fc_g1 is applied to the output of the second GAT layer.



GAT

– **GIN model**

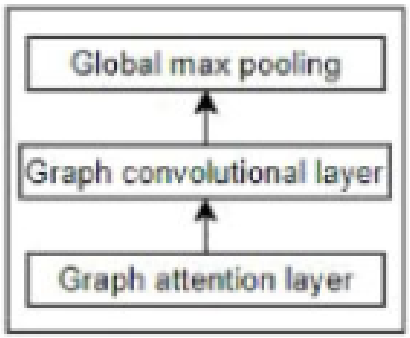
1. The model has 5 GINConv layers, each with a sequential neural network (nn1, nn2, ..., nn5) that consists of a Linear layer, a ReLU activation, and another Linear layer.
2. Each GINConv layer is followed by a BatchNorm1d layer to normalize the output.



GIN

– **GAT-GCN model**

1. The first layer is a GATConv layer, which implements the Graph Attention Network mechanism. It takes the input node features (num_features_xd) and outputs features with the same dimensionality, but with 10 attention heads.
2. The second layer is a GCNConv layer, which applies a standard Graph Convolutional Network operation. It takes the concatenated output of the previous GAT layer (10 times the original number of features) and outputs the same number of features.



GAT-GCN

- **Protein sequence branch**: operates on the protein sequence input, which is represented as a sequence of integer indices of characters representing amino acids.

1. The protein sequence is passed through an embedding layer (embedding_xt) to convert the integer indices into a dense vector representation.
2. The embedded sequence is then passed through a 1D convolutional layer (conv_xt_1) to extract features from the protein sequence.
3. The convolutional features are flattened and passed through a fully connected layer (fc1_xt) to obtain a fixed size output representation.

After obtaining the output representations from the two branches, the model concatenates them and passes the combined features through two more fully connected layers (fc1 and fc2) to produce the final output.

The model uses ReLU activations, dropout for regularization, and global max pooling (gmp) to aggregate the graph-level features.

4.1 GCN-based graph representation learning

The GCNNet class is a neural network model designed for processing two types of input data: molecular graphs (represented by SMILES strings) and protein sequences. This model uses a combination of Graph Convolutional Networks (GCNs) for the molecular graphs and 1D Convolutional Networks (Conv1D) for the protein sequences.

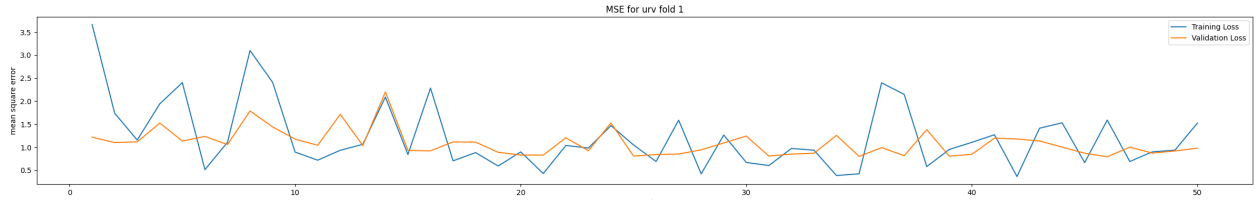
5 Results

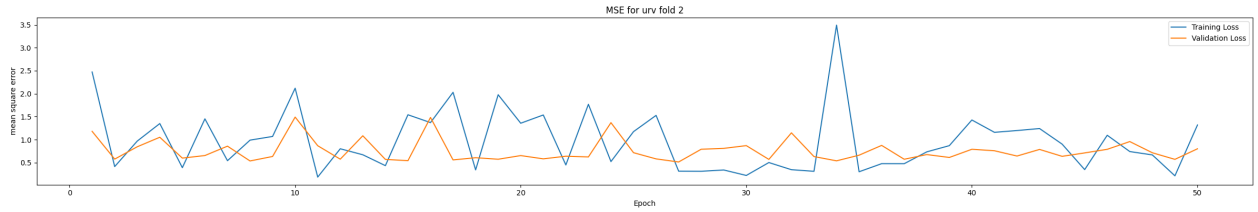
5.1 train and test GCN model on the URV dataset with k folds = 10

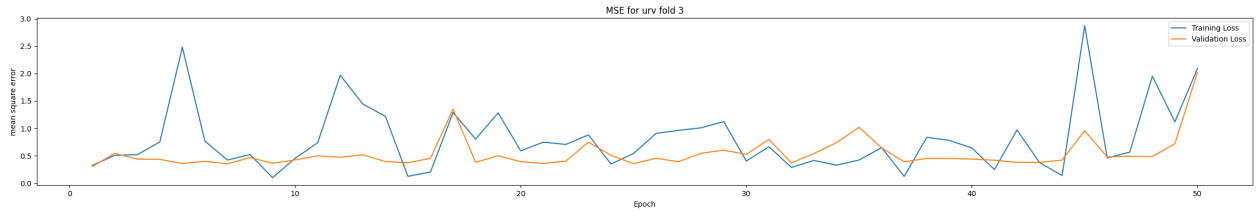
Apply k-folding with k = 10 to get 10 different combinations of train and test patterns for URV dataset using GCN model gives the following error evolutions and real predicted affinity

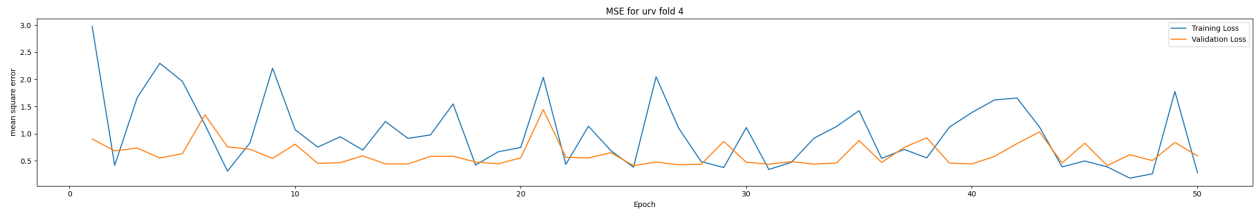
the mean MSE is 0.84

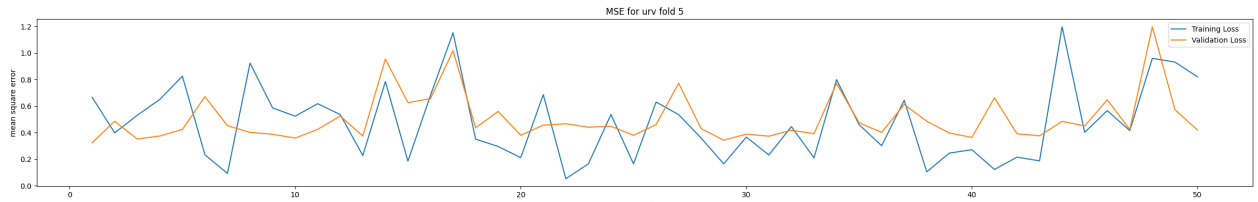
the mean standard deviation is 0.97

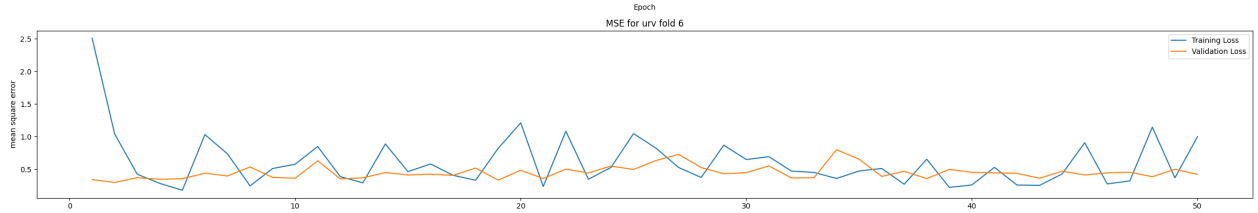

MSE for urv fold 1
Training Loss
Validation Loss
mean square error
Epoch

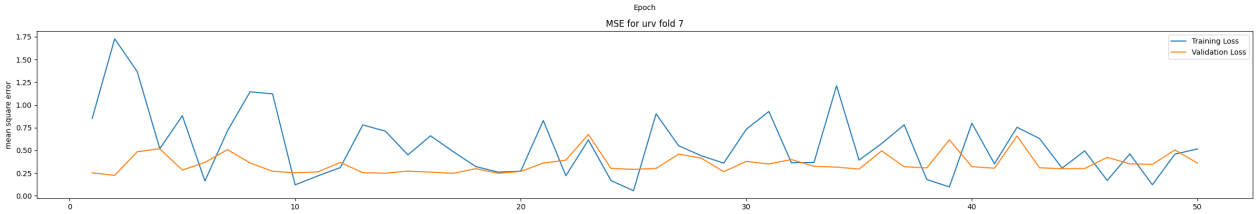

MSE for urv fold 2
Training Loss
Validation Loss
mean square error
Epoch

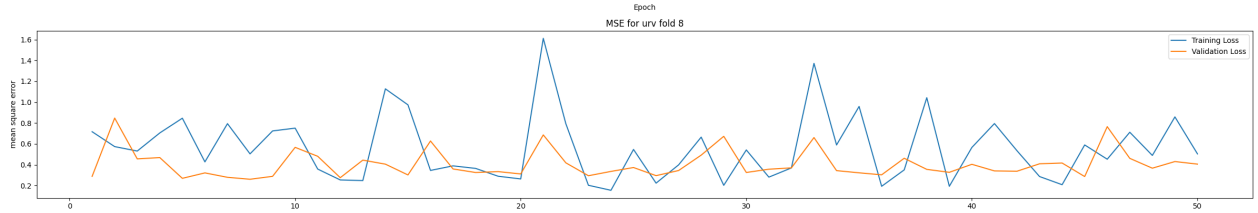

MSE for urv fold 3
Training Loss
Validation Loss
mean square error
Epoch

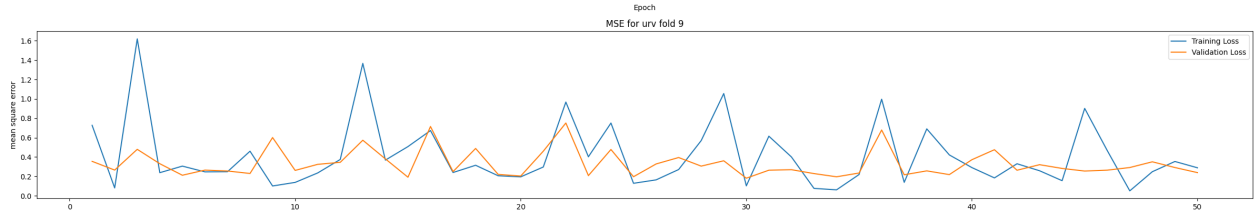

MSE for urv fold 4
Training Loss
Validation Loss
mean square error
Epoch

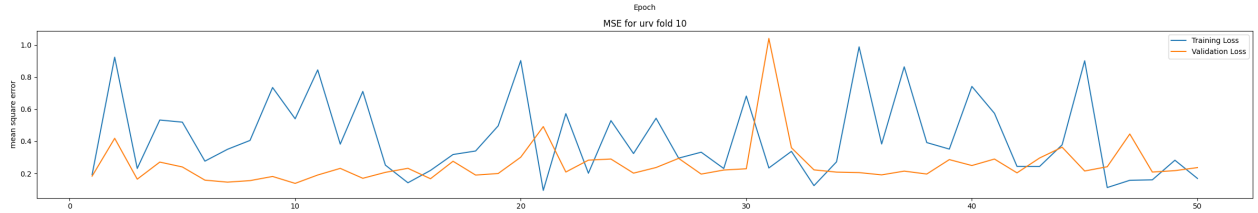

MSE for urv fold 5
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 6
Training Loss
Validation Loss
mean square error
Epoch

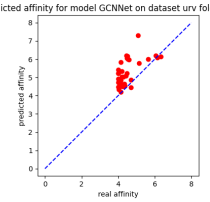

MSE for urv fold 7
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 8
Training Loss
Validation Loss
mean square error
Epoch

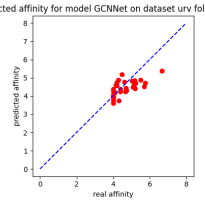

MSE for urv fold 9
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 10
Training Loss
Validation Loss
mean square error
Epoch

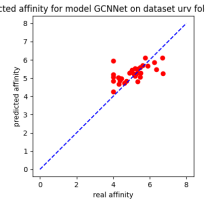
real vs predicted affinity for model GCNNet on dataset urv fold 1 with MSE 0.95



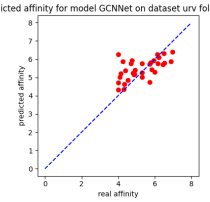
real vs predicted affinity for model GCNNet on dataset urv fold 2 with MSE 0.27



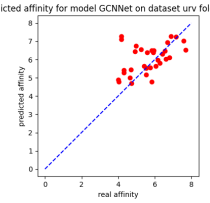
real vs predicted affinity for model GCNNet on dataset urv fold 3 with MSE 0.41



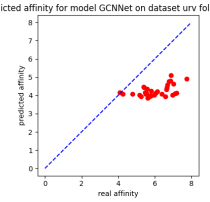
real vs predicted affinity for model GCNNet on dataset urv fold 4 with MSE 0.58



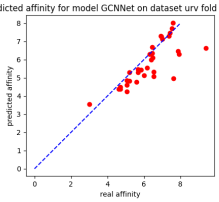
real vs predicted affinity for model GCNNet on dataset urv fold 5 with MSE 1.08



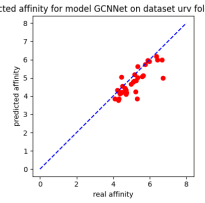
real vs predicted affinity for model GCNNet on dataset urv fold 6 with MSE 3.65



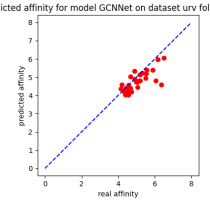
real vs predicted affinity for model GCNNet on dataset urv fold 7 with MSE 0.84



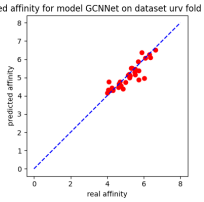
real vs predicted affinity for model GCNNet on dataset urv fold 8 with MSE 0.30



real vs predicted affinity for model GCNNet on dataset urv fold 9 with MSE 0.24



real vs predicted affinity for model GCNNet on dataset urv fold 10 with MSE 0.11

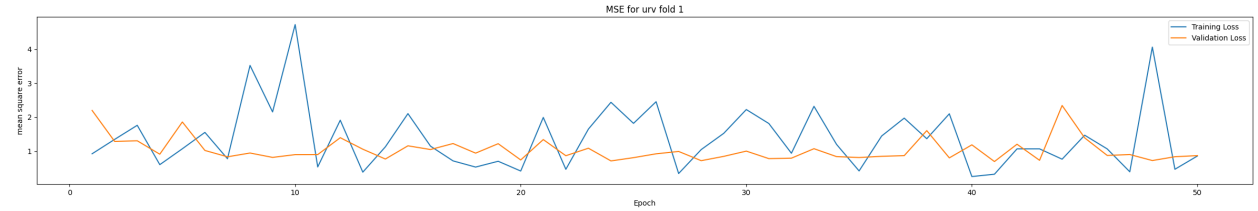


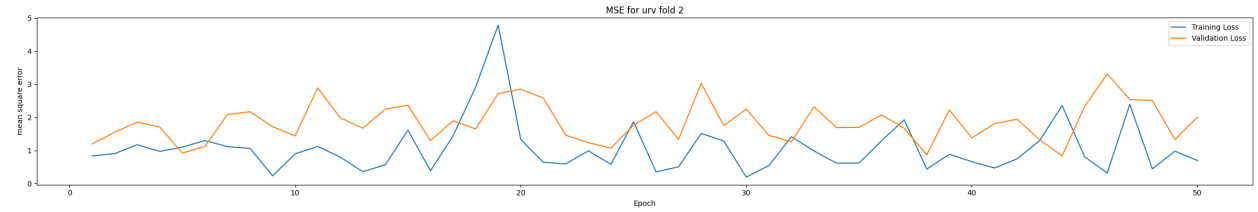
5.2 train and test GAT model on the URV dataset with k folds = 10

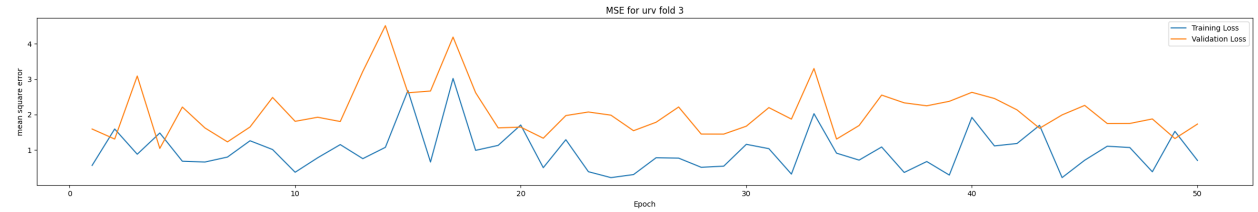
Apply k-folding with k = 10 to get 10 different combinations of train and test patterns for URV dataset using GAT model gives the following error evolutions and real predicted affinity

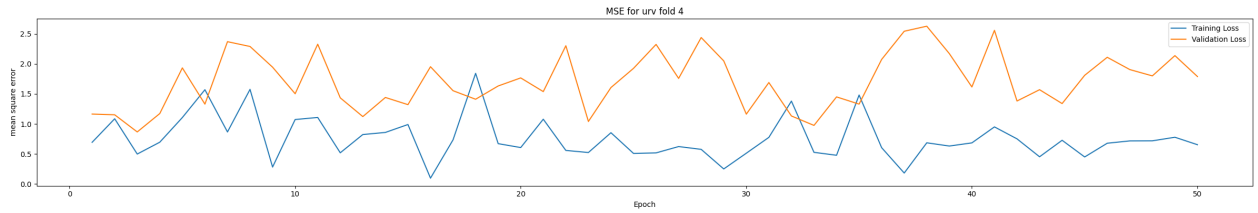
the mean MSE is 1.50

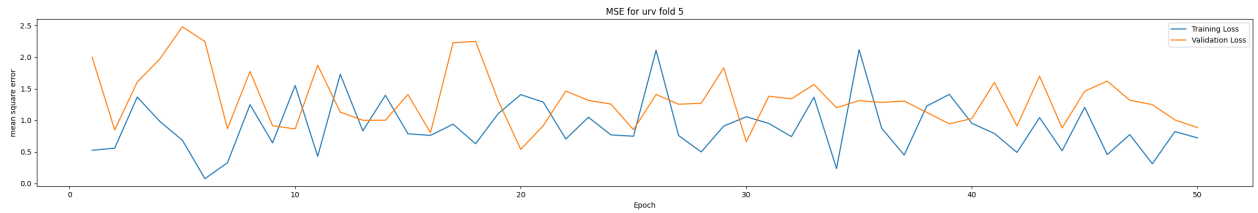
the mean standard deviation is 1.13

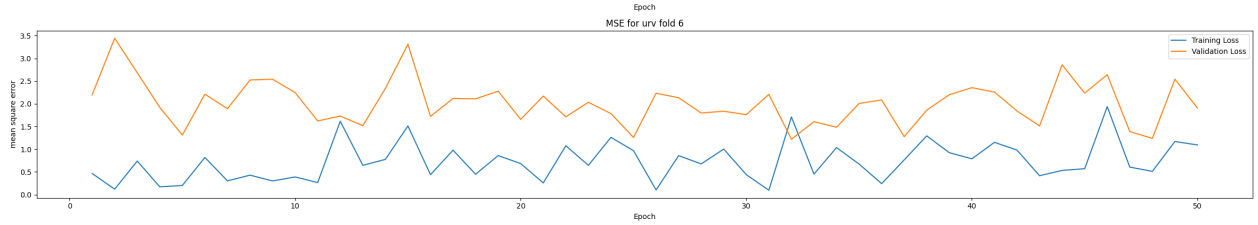

MSE for urv fold 1
Training Loss
Validation Loss
mean square error
Epoch

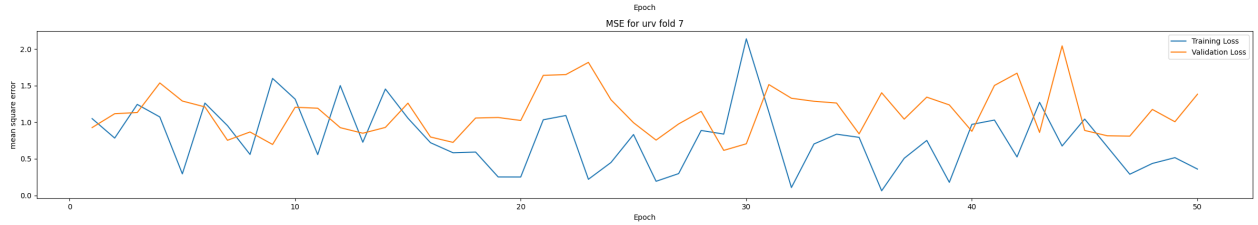

MSE for urv fold 2
Training Loss
Validation Loss
mean square error
Epoch

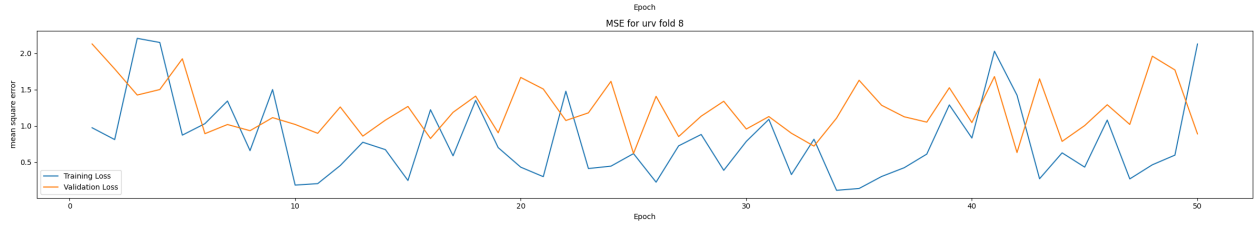

MSE for urv fold 3
Training Loss
Validation Loss
mean square error
Epoch

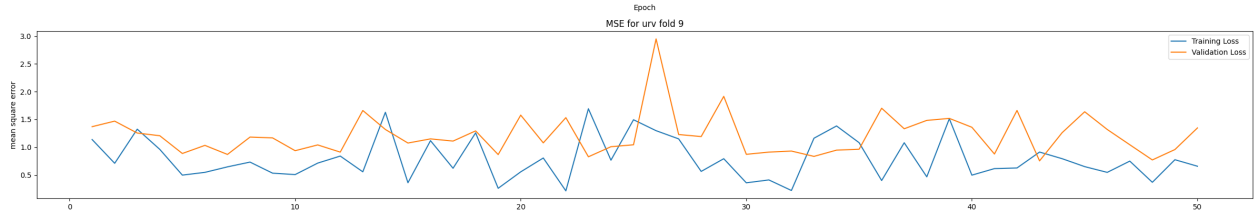

MSE for urv fold 4
Training Loss
Validation Loss
mean square error
Epoch

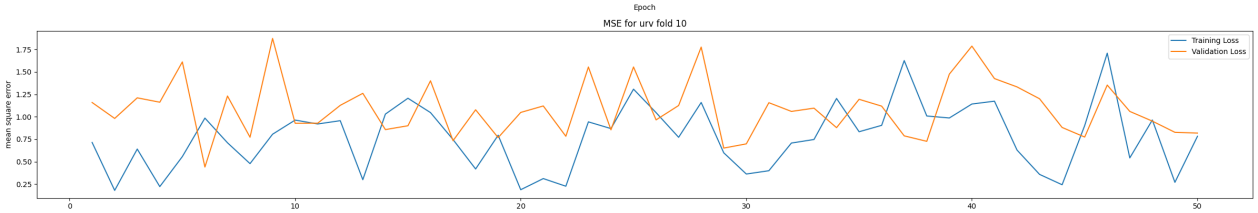

MSE for urv fold 5
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 6
Training Loss
Validation Loss
mean square error
Epoch

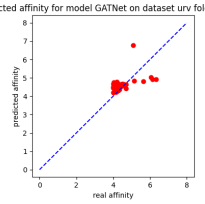

MSE for urv fold 7
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 8
Training Loss
Validation Loss
mean square error
Epoch

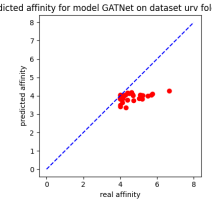

MSE for urv fold 9
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 10
Training Loss
Validation Loss
mean square error
Epoch

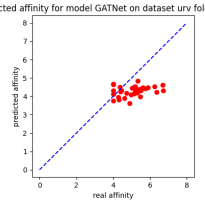
real vs predicted affinity for model GATNet on dataset urv fold 1 with MSE 0.34



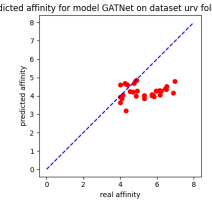
real vs predicted affinity for model GATNet on dataset urv fold 2 with MSE 0.82



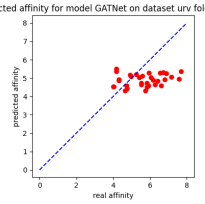
real vs predicted affinity for model GATNet on dataset urv fold 3 with MSE 1.17



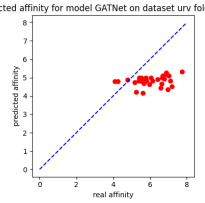
real vs predicted affinity for model GATNet on dataset urv fold 4 with MSE 1.92



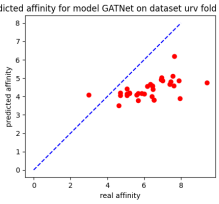
real vs predicted affinity for model GATNet on dataset urv fold 5 with MSE 1.58



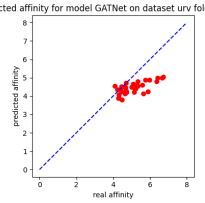
real vs predicted affinity for model GATNet on dataset urv fold 6 with MSE 2.07



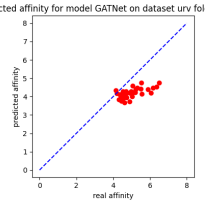
real vs predicted affinity for model GATNet on dataset urv fold 7 with MSE 4.53



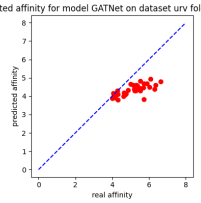
real vs predicted affinity for model GATNet on dataset urv fold 8 with MSE 0.74



real vs predicted affinity for model GATNet on dataset urv fold 9 with MSE 0.89



real vs predicted affinity for model GATNet on dataset urv fold 10 with MSE 0.96

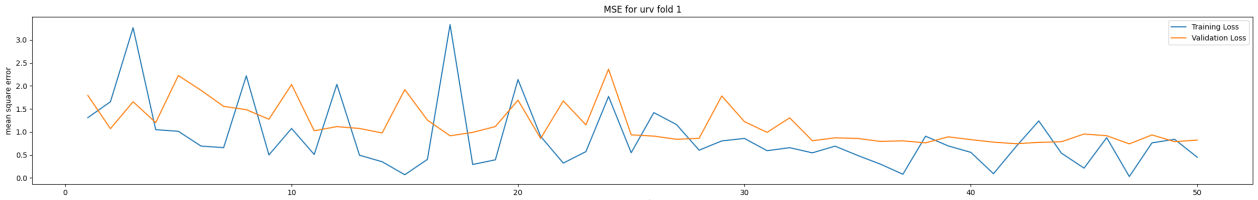


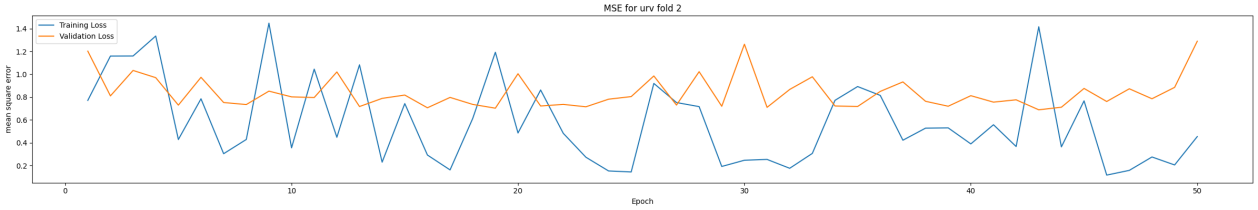
5.3 train and test GATGCN model on the URV dataset with k folds = 10

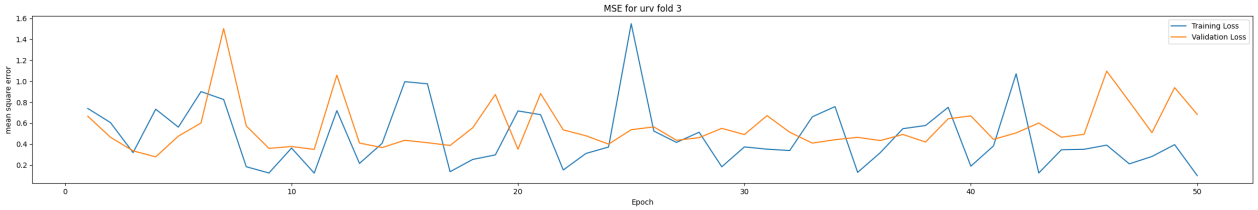
Apply k-folding with k = 10 to get 10 different combinations of train and test patterns for URV dataset using GATGCN model gives the following error evolutions and real predicted affinity

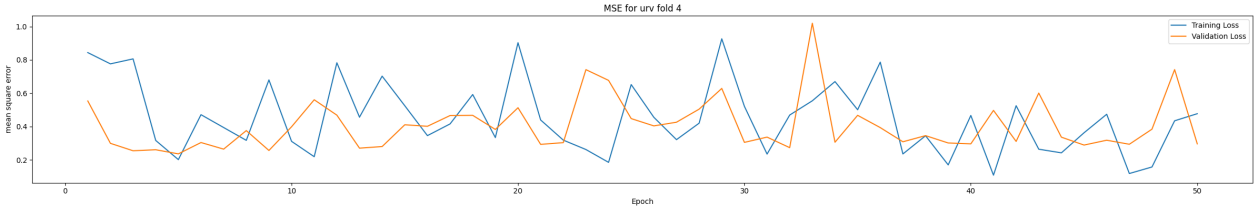
the mean MSE is 0.44

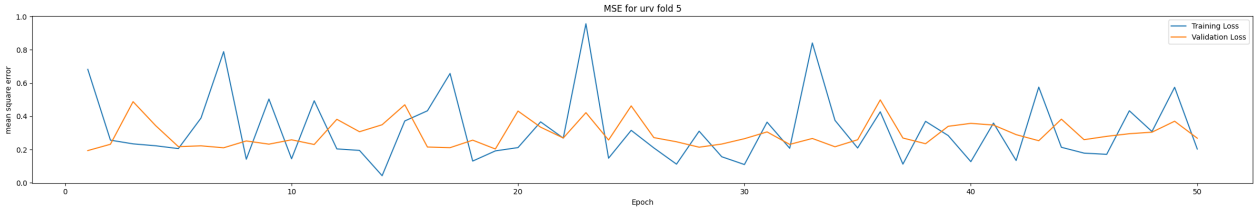
the mean standard deviation is 0.24

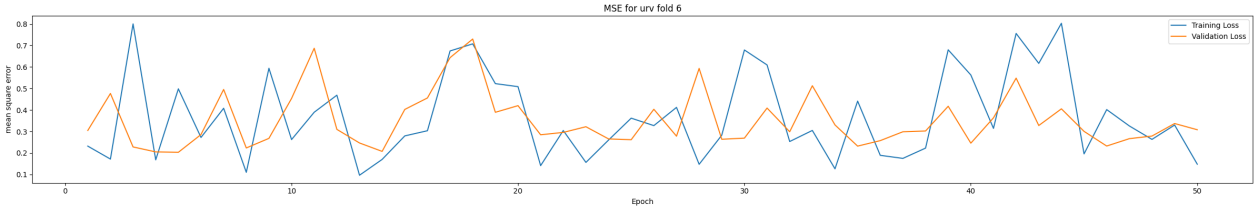

MSE for urv fold 1
Training Loss
Validation Loss
mean square error
Epoch

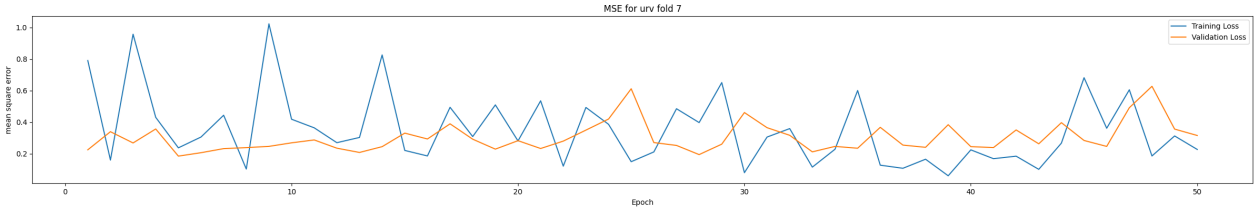

MSE for urv fold 2
Training Loss
Validation Loss
mean square error
Epoch

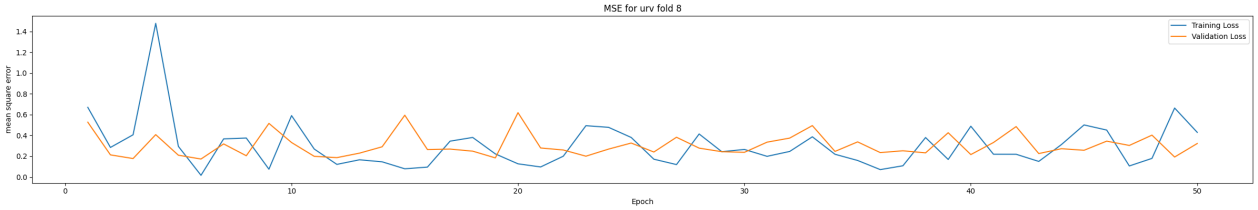

MSE for urv fold 3
Training Loss
Validation Loss
mean square error
Epoch

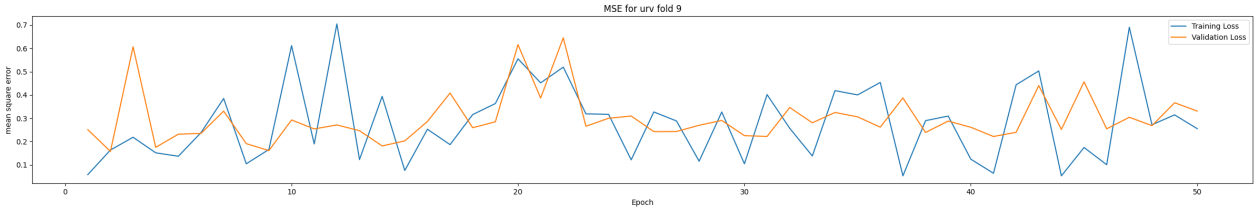

MSE for urv fold 4
Training Loss
Validation Loss
mean square error
Epoch

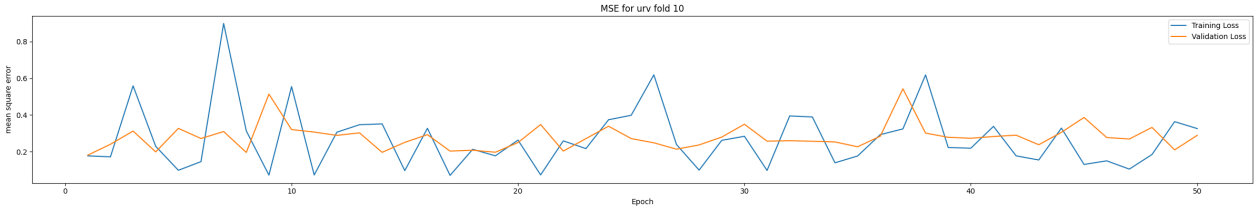

MSE for urv fold 5
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 6
Training Loss
Validation Loss
mean square error
Epoch

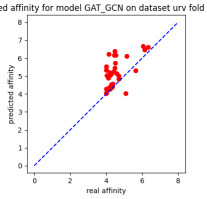

MSE for urv fold 7
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 8
Training Loss
Validation Loss
mean square error
Epoch

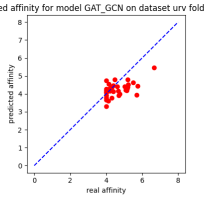

MSE for urv fold 9
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 10
Training Loss
Validation Loss
mean square error
Epoch

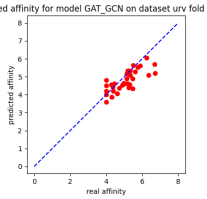
real vs predicted affinity for model GAT_GCN on dataset urv fold 1 with MSE 0.89



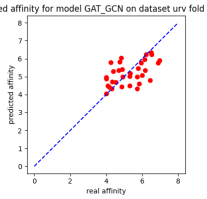
real vs predicted affinity for model GAT_GCN on dataset urv fold 2 with MSE 0.44



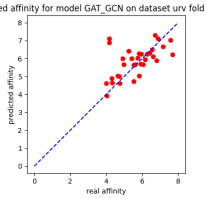
real vs predicted affinity for model GAT_GCN on dataset urv fold 3 with MSE 0.33



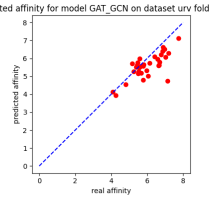
real vs predicted affinity for model GAT_GCN on dataset urv fold 4 with MSE 0.62



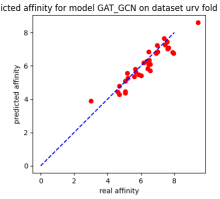
real vs predicted affinity for model GAT_GCN on dataset urv fold 5 with MSE 0.80



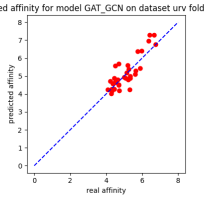
real vs predicted affinity for model GAT_GCN on dataset urv fold 6 with MSE 0.48



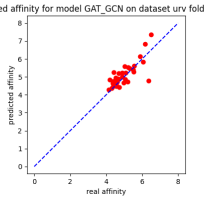
real vs predicted affinity for model GAT_GCN on dataset urv fold 7 with MSE 0.25



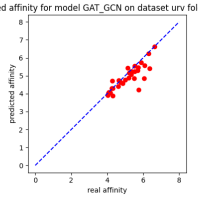
real vs predicted affinity for model GAT_GCN on dataset urv fold 8 with MSE 0.23



real vs predicted affinity for model GAT_GCN on dataset urv fold 9 with MSE 0.21



real vs predicted affinity for model GAT_GCN on dataset urv fold 10 with MSE 0.20

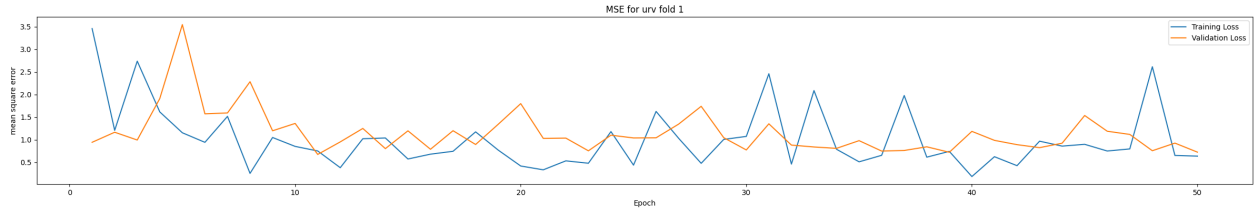


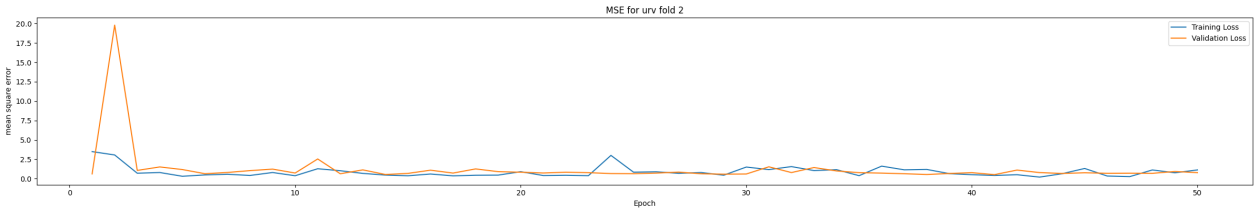
5.4 train and test GINCONV model on the URV dataset with k folds = 10

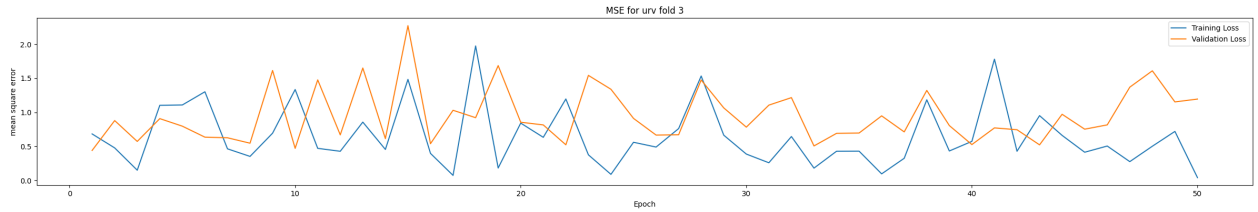
Apply k-folding with k = 10 to get 10 different combinations of train and test patterns for URV dataset using GINCONV model gives the following error evolutions and real predicted affinity

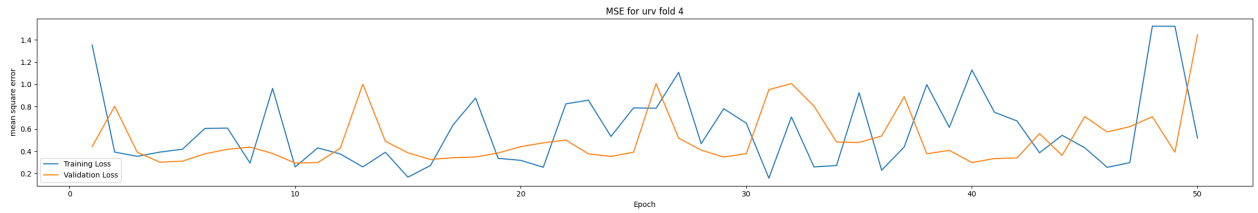
the mean MSE is 0.76

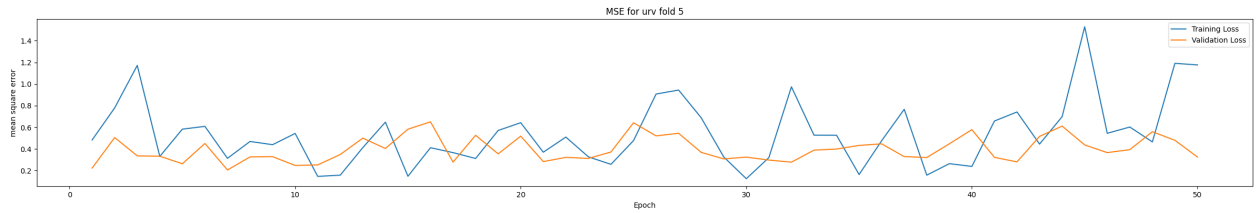
the mean standard deviation is 0.56

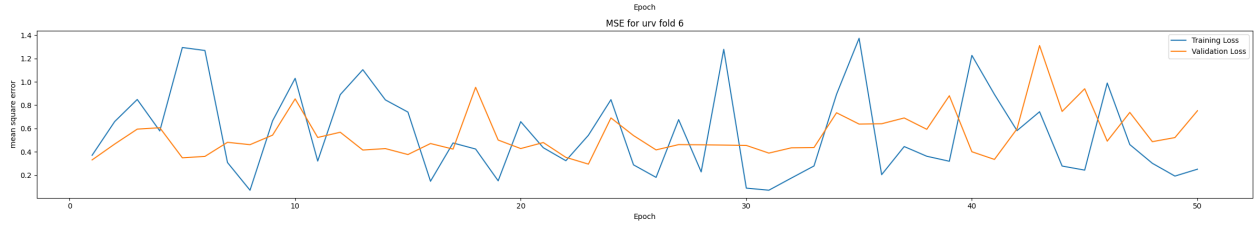

MSE for urv fold 1
Training Loss
Validation Loss
mean square error
Epoch

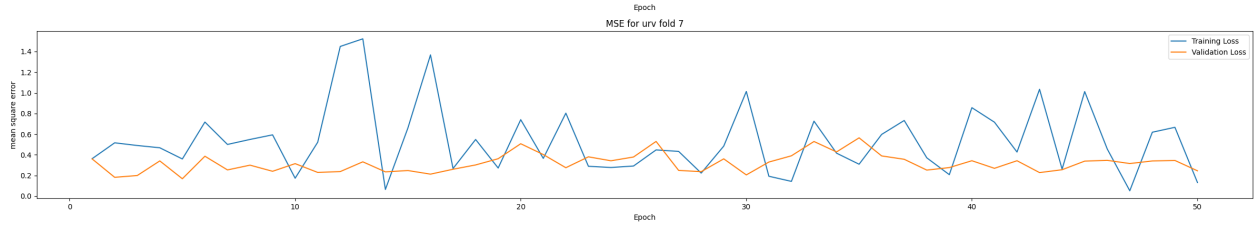

MSE for urv fold 2
Training Loss
Validation Loss
mean square error
Epoch

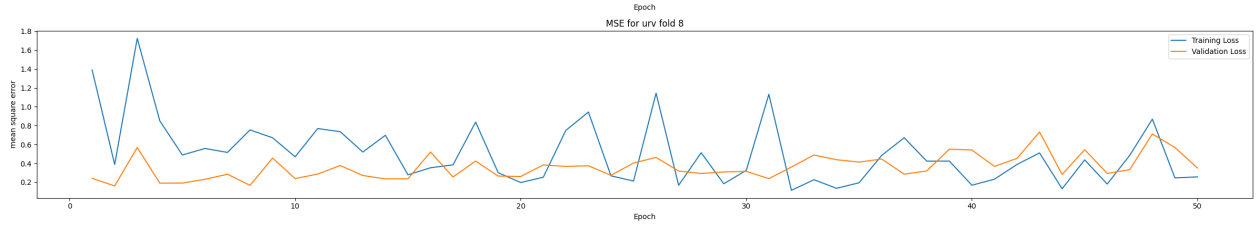

MSE for urv fold 3
Training Loss
Validation Loss
mean square error
Epoch

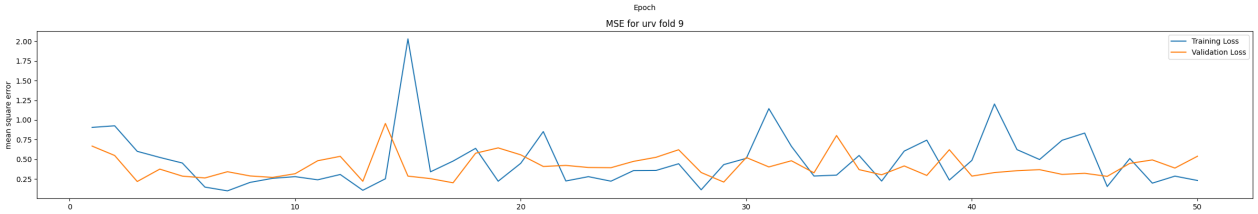

MSE for urv fold 4
Training Loss
Validation Loss
mean square error
Epoch

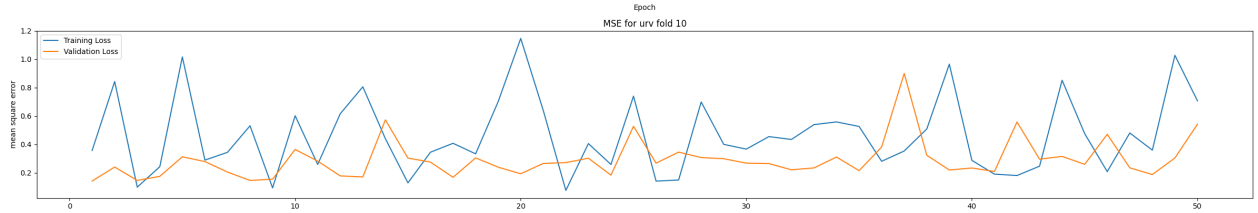

MSE for urv fold 5
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 6
Training Loss
Validation Loss
mean square error
Epoch

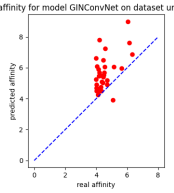

MSE for urv fold 7
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 8
Training Loss
Validation Loss
mean square error
Epoch

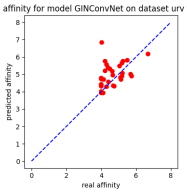

MSE for urv fold 9
Training Loss
Validation Loss
mean square error
Epoch


MSE for urv fold 10
Training Loss
Validation Loss
mean square error
Epoch

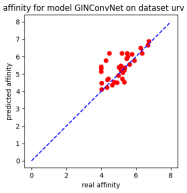
real vs predicted affinity for model GINConvNet on dataset urv fold 1 with MSE 2.01



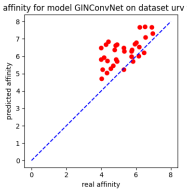
real vs predicted affinity for model GINConvNet on dataset urv fold 2 with MSE 0.62



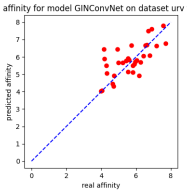
real vs predicted affinity for model GINConvNet on dataset urv fold 3 with MSE 0.44



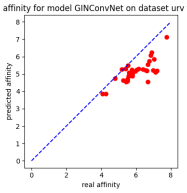
real vs predicted affinity for model GINConvNet on dataset urv fold 4 with MSE 1.51



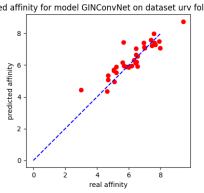
real vs predicted affinity for model GINConvNet on dataset urv fold 5 with MSE 0.62



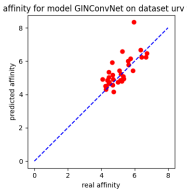
real vs predicted affinity for model GINConvNet on dataset urv fold 6 with MSE 1.06



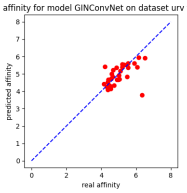
real vs predicted affinity for model GINConvNet on dataset urv fold 7 with MSE 0.32



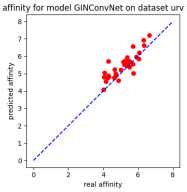
real vs predicted affinity for model GINConvNet on dataset urv fold 8 with MSE 0.41



real vs predicted affinity for model GINConvNet on dataset urv fold 9 with MSE 0.35



real vs predicted affinity for model GINConvNet on dataset urv fold 10 with MSE 0.26



References

[1] [Thin et al.] Thin Nguyen; Hang Le; Thomas P. Quinn; Tri Nguyen; Thuc Duy Le; and Svetha Venkatesh, "GraphDTA: predicting drug–target binding affinity with graph neural networks", Journal Bioinformatics, Volume 37, Issue 8, March 2021, Pages 1140–1147 Available at https://academic.oup.com/bioinformatics/article/37/8/1140/5942970.

[2] Related data, pre-trained models and source code in [1] are publicly available at https://github.com/thinng/GraphDTA.

[3] the repository forked from [2] to perform experiments in this thesis available at https://github.com/YoussefEzz/GraphDTA_forked.

[4] PDB website: https://www.rcsb.org/.

[5] Bio.PDB Biopython module: https://biopython.org/wiki/The_Biopython_Structural_Bioinformatics_FAQ.

[6] chemspider website: https://www.chemspider.com/.

[7] chemspider website: https://www.rdkit.org/docs/source/rdkit.Chem.html.