Toxicity Classification Using GNNs

1805006 - Tanjeem Azwad Zaman
1805030 - Md Toki Tahmid

Problem Definition

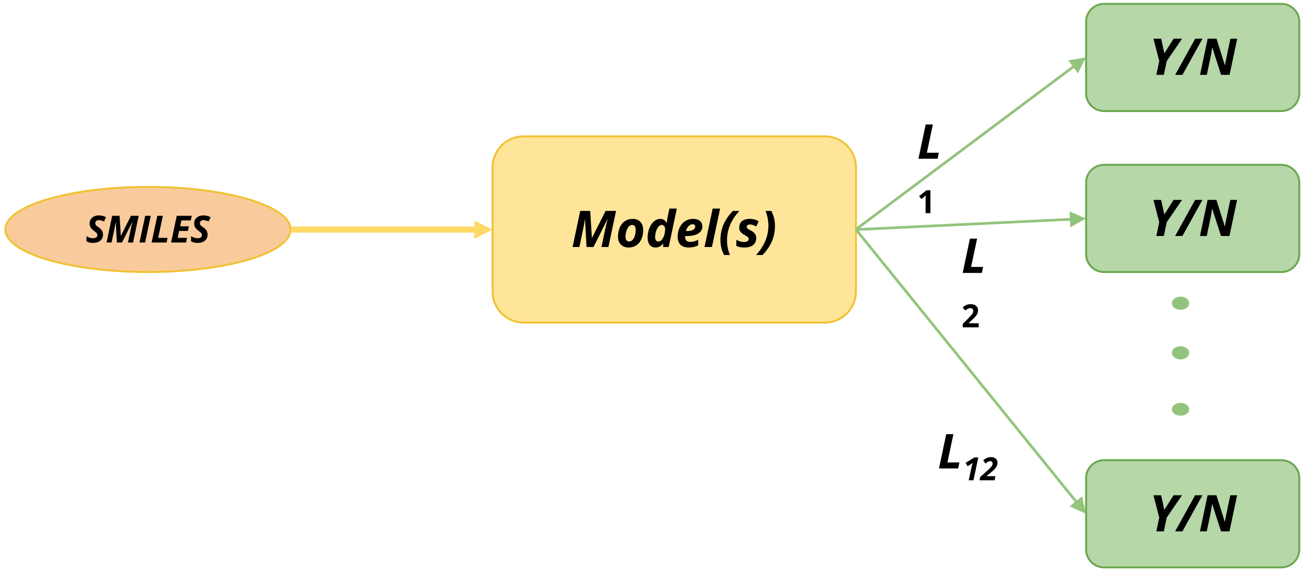
Basic Concepts

- ***SMILES:***
  - Simplified Molecular Input Line Entry System
  - String representation of a compound/mixture
  - Example: ***Aspirin*** is ***CC(=O)OC1=CC=CC=C1C(=O)***
- ***Multi-Label Binary Classification Task***
  - Many output labels/Columns
  - For each input, a single ***Yes/No*** for EACH output column

Input	Output		
	Acidic?	Reductant?	Soluble?
CH3-CH2-OH	Yes	No	Yes

Problem Definition

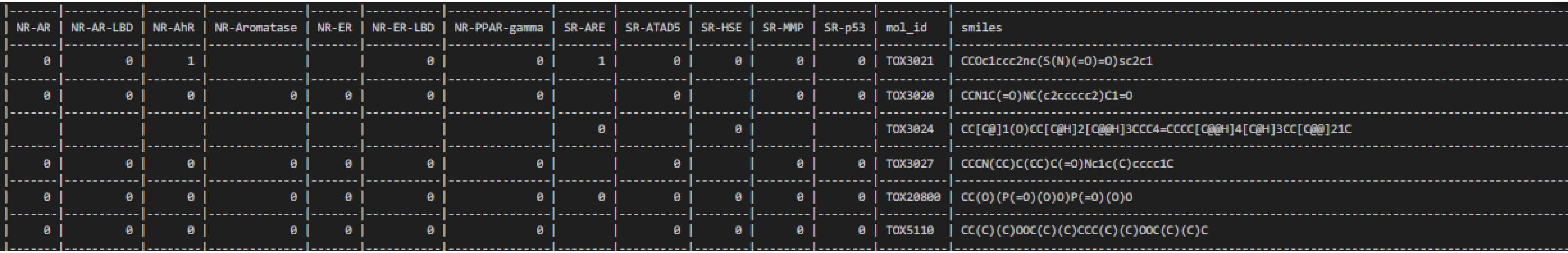
- A Multi-label Binary Classification Task on the Tox-21 Dataset
- Ie. ***Input*** -> *SMILES*; ***Output*** -> an *Yes/No* for 12 labels each



Dataset and its Analysis (Statistics)

Consolidated Dataset

- Per Row: (1 *mol_id* +1 smiles + 12 labels)
- 7831 entries



```
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
| NR-AR | NR-AR-LBD | NR-AhR | NR-Aromatase | NR-ER | NR-ER-LBD | NR-PPAR-gamma | SR-ARE | SR-ATAD5 | SR-HSE | SR-MMP | SR-p53 | mol_id   | smiles
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|     0 |         0 |      1 |              |       |         0 |             0 |      1 |        0 |      0 |      0 |      0 | TOX3021  | CCOc1ccc2nc(S(N)(=O)=O)sc2c1
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|     0 |         0 |      0 |            0 |     0 |         0 |             0 |        |        0 |        |      0 |      0 | TOX3020  | CCN1C(=O)NC(c2ccccc2)C1=O
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|       |           |        |              |       |           |               |      0 |          |      0 |        |        | TOX3024  | CC[C@]1(O)CC[C@H]2[C@@H]3CCC4=CCCC[C@@H]4[C@H]3CC[C@@]21C
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|     0 |         0 |      0 |            0 |     0 |         0 |             0 |        |        0 |        |      0 |      0 | TOX3027  | CCCN(CC)C(CC)C(=O)Nc1c(C)cccc1C
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|     0 |         0 |      0 |            0 |     0 |         0 |             0 |      0 |        0 |      0 |      0 |      0 | TOX20800 | CC(O)(P(=O)(O)O)P(=O)(O)O
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
|     0 |         0 |      0 |            0 |     0 |         0 |             0 |        |        0 |      0 |      0 |      0 | TOX5110  | CC(C)(C)OOC(C)(C)CCC(C)(C)OOC(C)(C)C
|-------|-----------|--------|--------------|-------|-----------|---------------|--------|----------|--------|--------|--------|----------|------------------------------------------------------------------
```

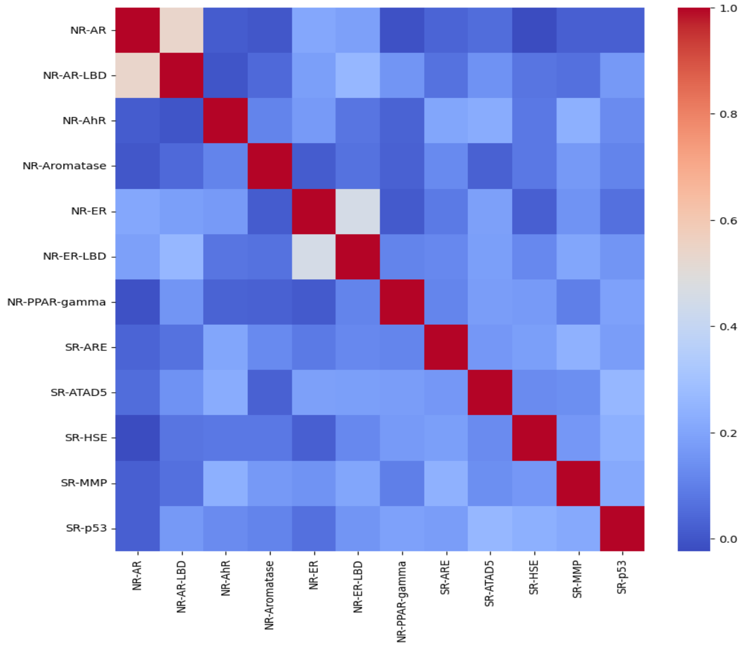
Consolidated Dataset

- Issues:
  - Lots of NaN’s in labels

Attempted Solution	***Issue***
1. Remove Row with a NaN	Dataset reduced
2. Weighted loss function (0 for NaN)	Issue with converging
3. Random fillna with 0/1	Negative correlation Established

Consolidated Dataset

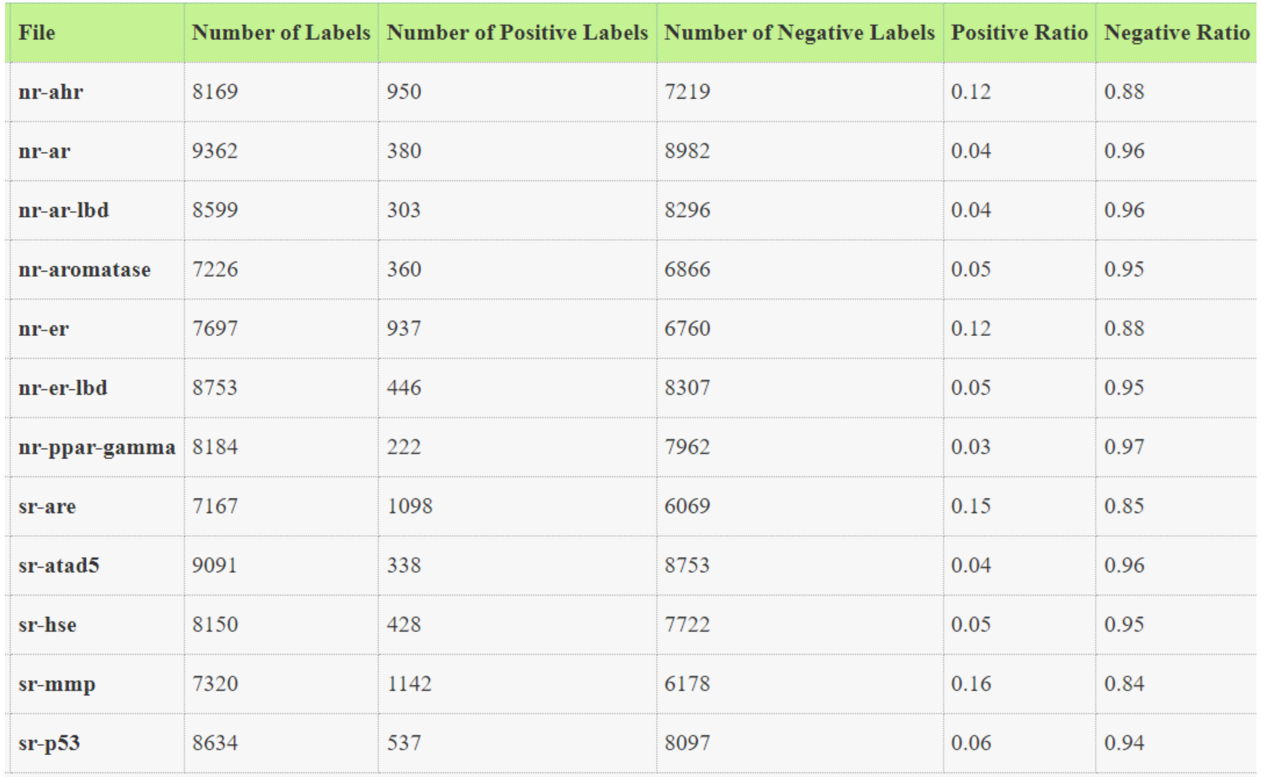
- Correlation Matrix:
- Observation:
  - Not closely related, except (***NR-AR, NR-AR-LBD***) & (***NR-ER, NR-ER-LBD***)
  - Try for 12 models, one for each label



Primary Dataset

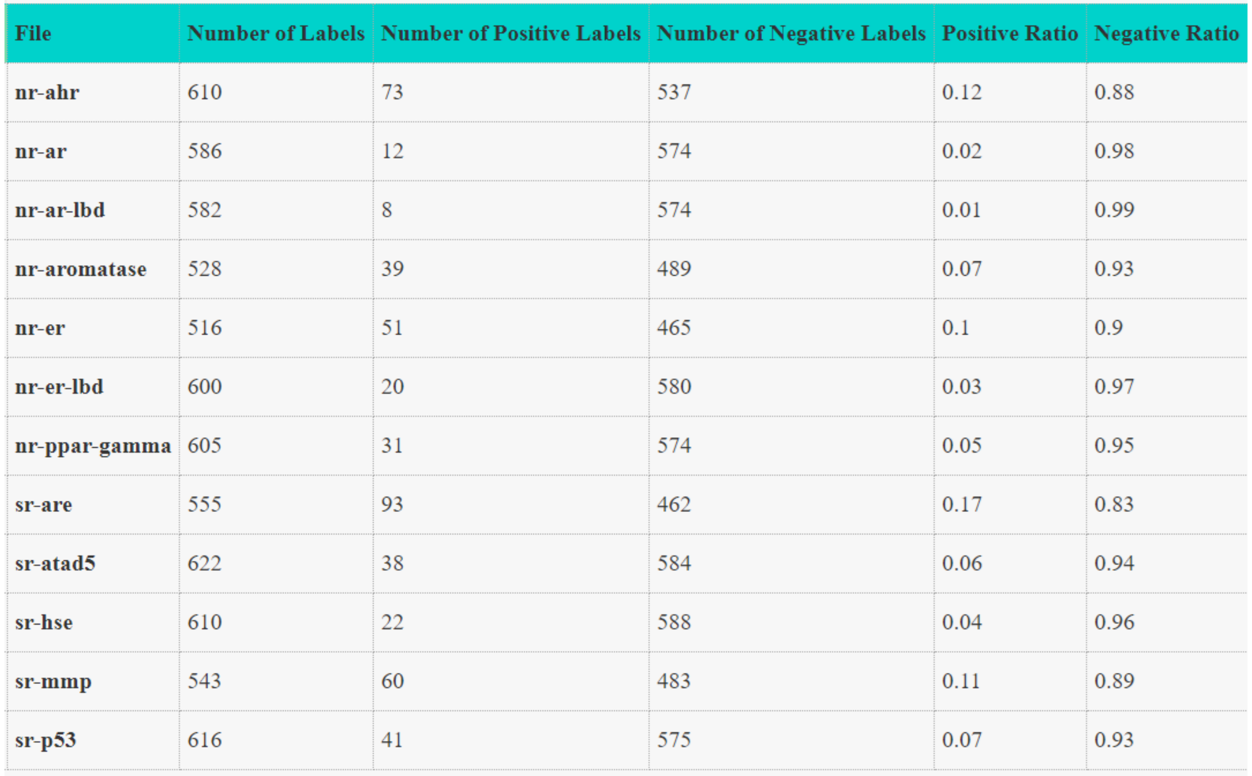
- Separate Datasets for each Label
- Separate Test sets as well.
- VERY Imbalanced -> UnderSampled
- Link: Separate Datasets

Dataset Statistics



File	Number of Labels	Number of Positive Labels	Number of Negative Labels	Positive Ratio	Negative Ratio
nr-ahr	8169	950	7219	0.12	0.88
nr-ar	9362	380	8982	0.04	0.96
nr-ar-lbd	8599	303	8296	0.04	0.96
nr-aromatase	7226	360	6866	0.05	0.95
nr-er	7697	937	6760	0.12	0.88
nr-er-lbd	8753	446	8307	0.05	0.95
nr-ppar-gamma	8184	222	7962	0.03	0.97
sr-are	7167	1098	6069	0.15	0.85
sr-atad5	9091	338	8753	0.04	0.96
sr-hse	8150	428	7722	0.05	0.95
sr-mmp	7320	1142	6178	0.16	0.84
sr-p53	8634	537	8097	0.06	0.94

Dataset Statistics (Test)

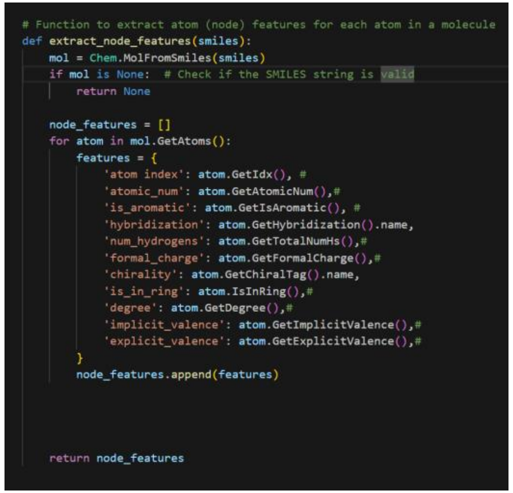


File	Number of Labels	Number of Positive Labels	Number of Negative Labels	Positive Ratio	Negative Ratio
nr-ahr	610	73	537	0.12	0.88
nr-ar	586	12	574	0.02	0.98
nr-ar-lbd	582	8	574	0.01	0.99
nr-aromatase	528	39	489	0.07	0.93
nr-er	516	51	465	0.1	0.9
nr-er-lbd	600	20	580	0.03	0.97
nr-ppar-gamma	605	31	574	0.05	0.95
sr-are	555	93	462	0.17	0.83
sr-atad5	622	38	584	0.06	0.94
sr-hse	610	22	588	0.04	0.96
sr-mmp	543	60	483	0.11	0.89
sr-p53	616	41	575	0.07	0.93

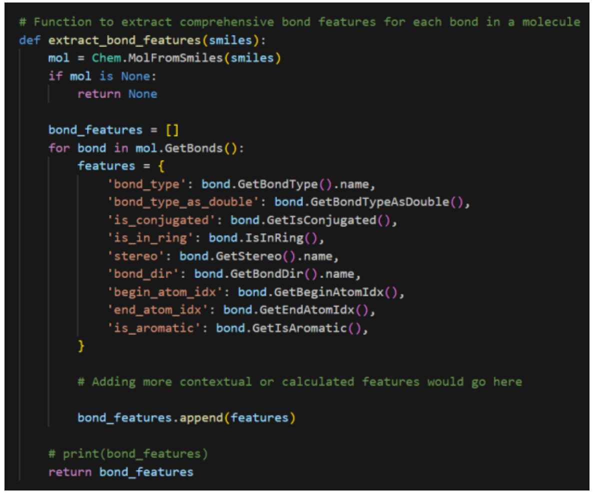
Proposed Solution (Architecture)

Main Idea

- Represent each molecule/ compound as a graph where
  - Each node == an Atom
  - Each edge == a bond between two atoms



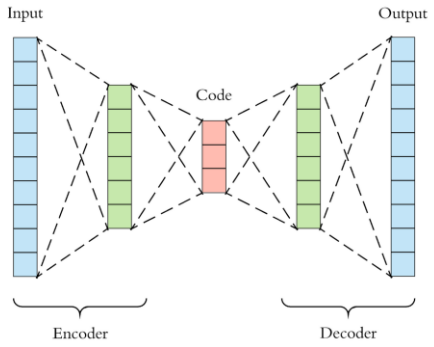
```python
# Function to extract atom (node) features for each atom in a molecule
def extract_node_features(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:  # Check if the SMILES string is valid
        return None

    node_features = []
    for atom in mol.GetAtoms():
        features = {
            'atom index': atom.GetIdx(), #
            'atomic_num': atom.GetAtomicNum(),#
            'is_aromatic': atom.GetIsAromatic(), #
            'hybridization': atom.GetHybridization().name,
            'num_hydrogens': atom.GetTotalNumHs(),#
            'formal_charge': atom.GetFormalCharge(),#
            'chirality': atom.GetChiralTag().name,
            'is_in_ring': atom.IsInRing(),#
            'degree': atom.GetDegree(),#
            'implicit_valence': atom.GetImplicitValence(),#
            'explicit_valence': atom.GetExplicitValence(),#
        }
        node_features.append(features)



    return node_features
```

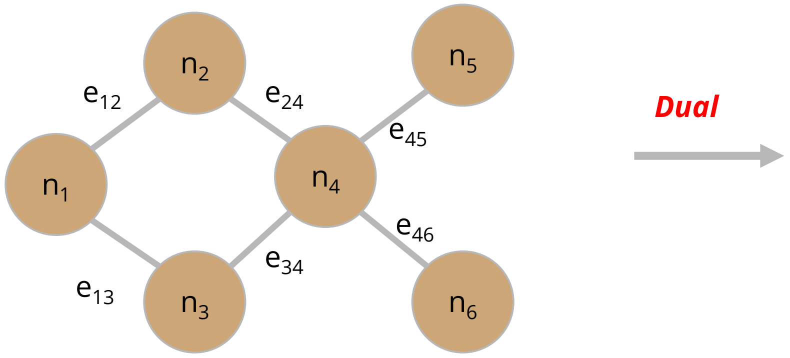


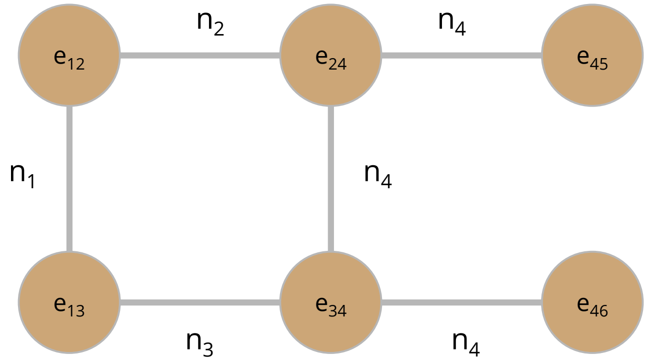
```python
# Function to extract comprehensive bond features for each bond in a molecule
def extract_bond_features(smiles):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return None

    bond_features = []
    for bond in mol.GetBonds():
        features = {
            'bond_type': bond.GetBondType().name,
            'bond_type_as_double': bond.GetBondTypeAsDouble(),
            'is_conjugated': bond.GetIsConjugated(),
            'is_in_ring': bond.IsInRing(),
            'stereo': bond.GetStereo().name,
            'bond_dir': bond.GetBondDir().name,
            'begin_atom_idx': bond.GetBeginAtomIdx(),
            'end_atom_idx': bond.GetEndAtomIdx(),
            'is_aromatic': bond.GetIsAromatic(),
        }

        # Adding more contextual or calculated features would go here

        bond_features.append(features)

    # print(bond_features)
    return bond_features
```

Main Idea

- Use ***Graph-Auto-Encoders*** and
- Use ***middle hidden layer*** as ***"feature Vector"***
- Original molecular graphs -> ***Node features*** represented
- Dual of each molecular graph -> ***Edge Features*** represented



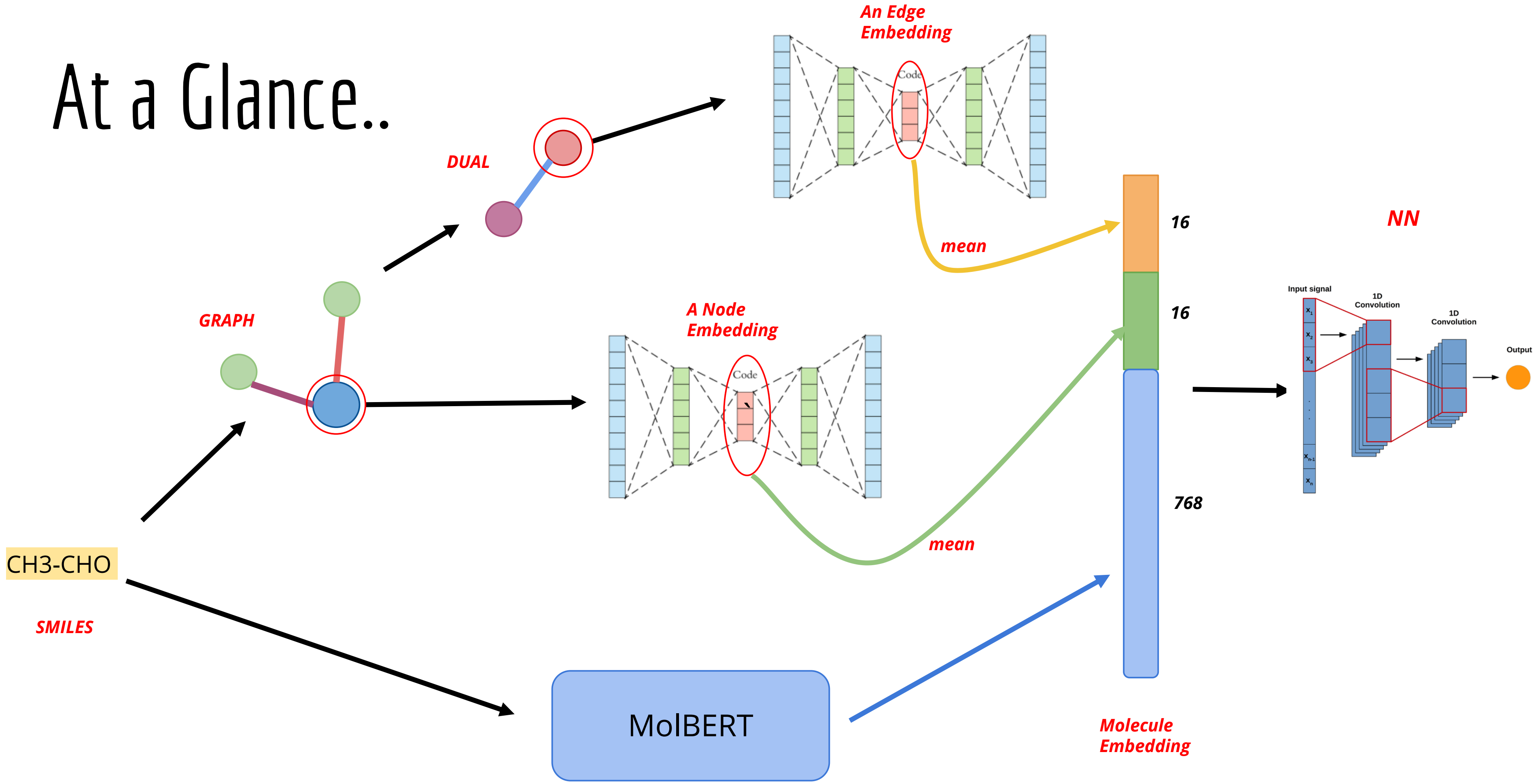




Main Idea

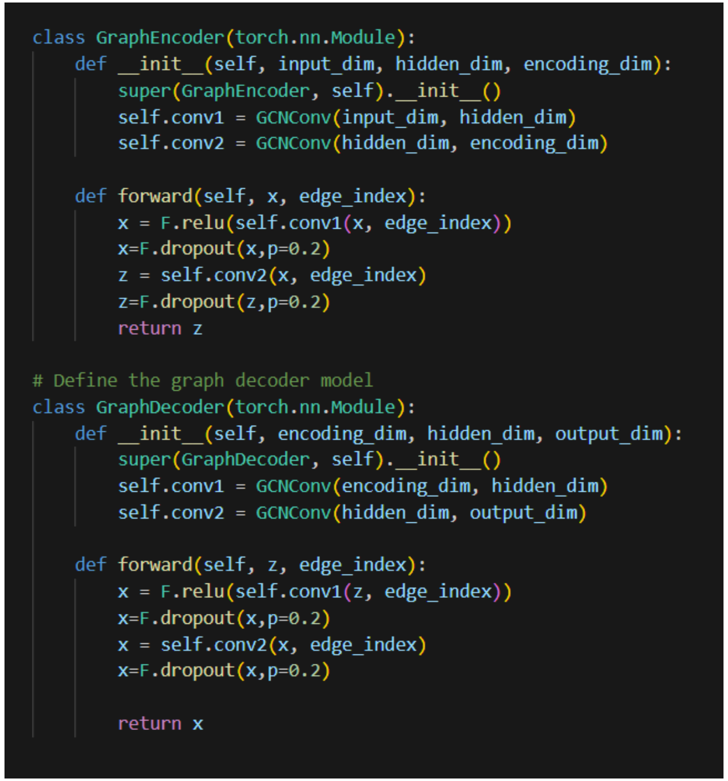
- Each node gets a feature vector; so take ***mean over all nodes*** to get feature vector for a ***graph/molecule***
- Use ***MolBert*** to get another set of features (embedding) for entire molecule
- Final Feature vector/Embedding for a molecule
  = (***Mean Node Embedding)*** + (***Mean Edge Embedding)*** + (***MolBert Embedding)***

At a Glance..

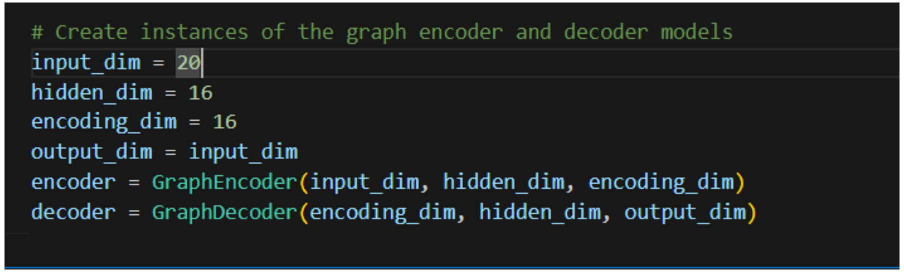


An Edge Embedding
DUAL
Code
mean
16
16
NN
GRAPH
A Node Embedding
Code
Input signal
1D Convolution
1D Convolution
Output
768
mean
CH3-CHO
SMILES
MolBERT
Molecule Embedding

Architecture (Encoder, Decoder)

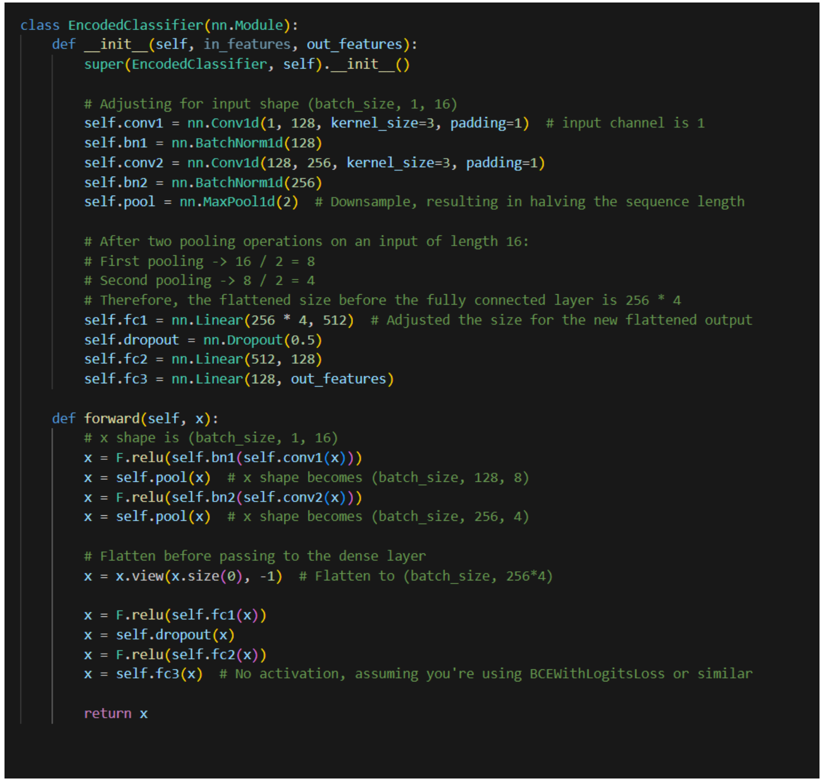


```python
class GraphEncoder(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, encoding_dim):
        super(GraphEncoder, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, encoding_dim)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x=F.dropout(x,p=0.2)
        z = self.conv2(x, edge_index)
        z=F.dropout(z,p=0.2)
        return z

# Define the graph decoder model
class GraphDecoder(torch.nn.Module):
    def __init__(self, encoding_dim, hidden_dim, output_dim):
        super(GraphDecoder, self).__init__()
        self.conv1 = GCNConv(encoding_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, output_dim)

    def forward(self, z, edge_index):
        x = F.relu(self.conv1(z, edge_index))
        x=F.dropout(x,p=0.2)
        x = self.conv2(x, edge_index)
        x=F.dropout(x,p=0.2)

        return x
```



```python
# Create instances of the graph encoder and decoder models
input_dim = 20
hidden_dim = 16
encoding_dim = 16
output_dim = input_dim
encoder = GraphEncoder(input_dim, hidden_dim, encoding_dim)
decoder = GraphDecoder(encoding_dim, hidden_dim, output_dim)
```

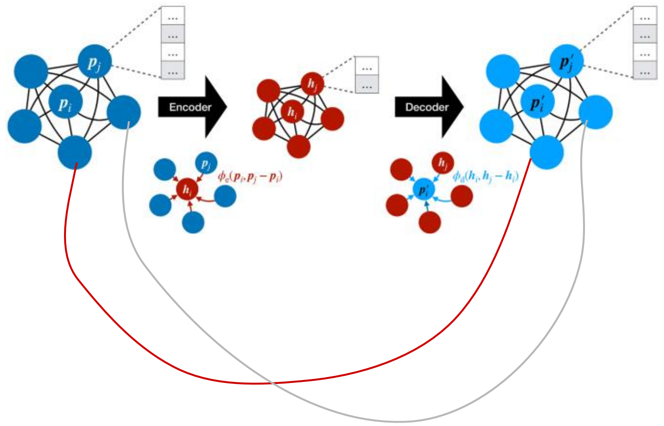
Architecture (DNN for classification)


```python
class EncodedClassifier(nn.Module):
    def __init__(self, in_features, out_features):
        super(EncodedClassifier, self).__init__()

        # Adjusting for input shape (batch_size, 1, 16)
        self.conv1 = nn.Conv1d(1, 128, kernel_size=3, padding=1)  # input channel is 1
        self.bn1 = nn.BatchNorm1d(128)
        self.conv2 = nn.Conv1d(128, 256, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm1d(256)
        self.pool = nn.MaxPool1d(2)  # Downsample, resulting in halving the sequence length

        # After two pooling operations on an input of length 16:
        # First pooling -> 16 / 2 = 8
        # Second pooling -> 8 / 2 = 4
        # Therefore, the flattened size before the fully connected layer is 256 * 4
        self.fc1 = nn.Linear(256 * 4, 512)  # Adjusted the size for the new flattened output
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, out_features)

    def forward(self, x):
        # x shape is (batch_size, 1, 16)
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool(x)  # x shape becomes (batch_size, 128, 8)
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool(x)  # x shape becomes (batch_size, 256, 4)

        # Flatten before passing to the dense layer
        x = x.view(x.size(0), -1)  # Flatten to (batch_size, 256*4)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)  # No activation, assuming you're using BCEWithLogitsLoss or similar

        return x
```

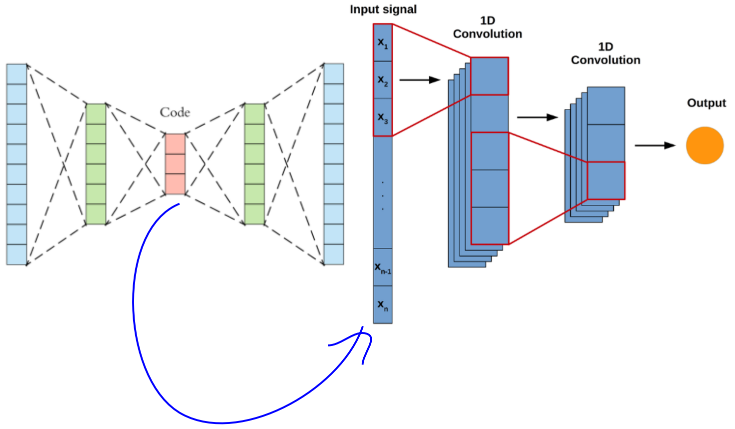
Loss Function and its intuition

Reconstruction Loss

Classification Loss



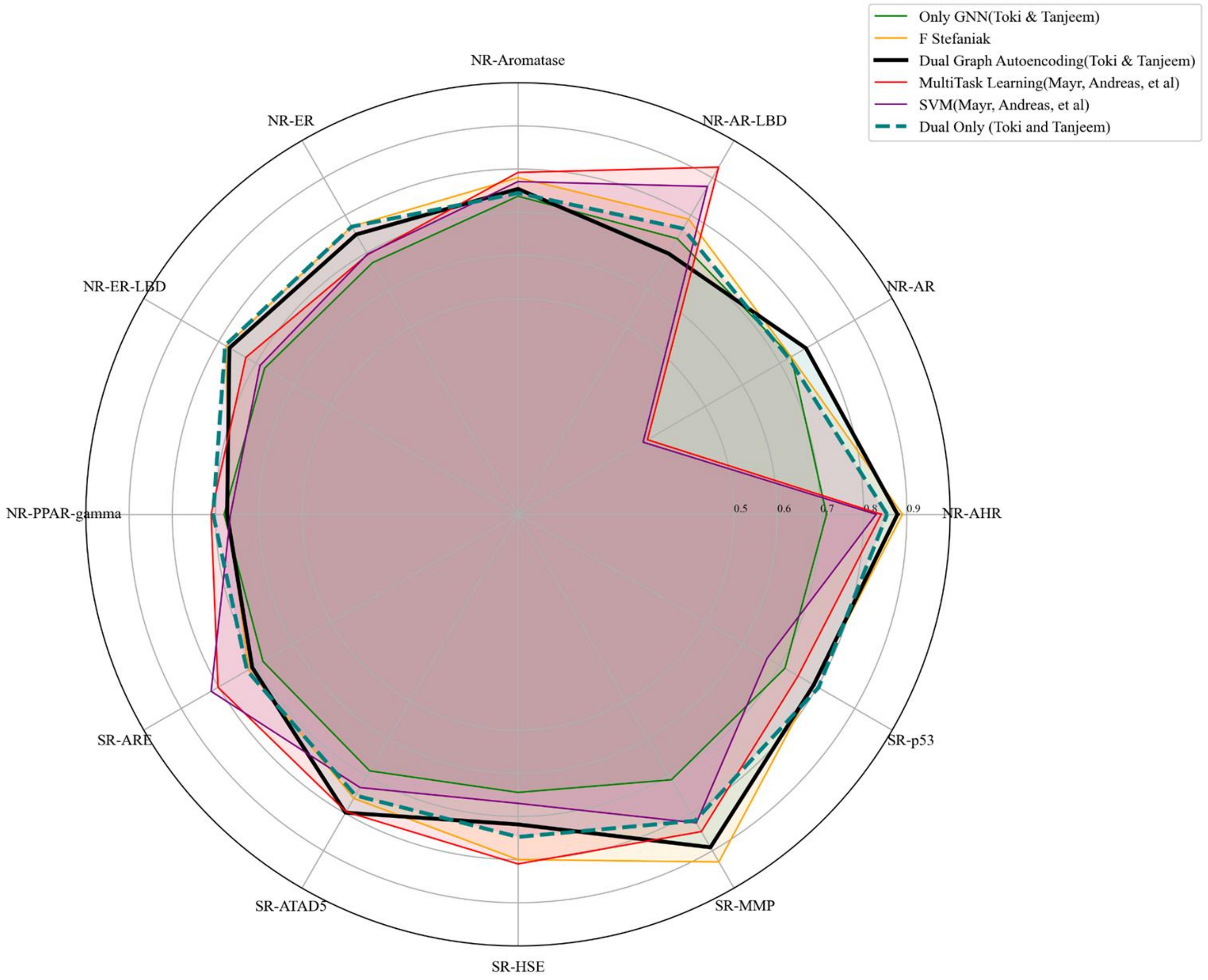
MSE LOSS



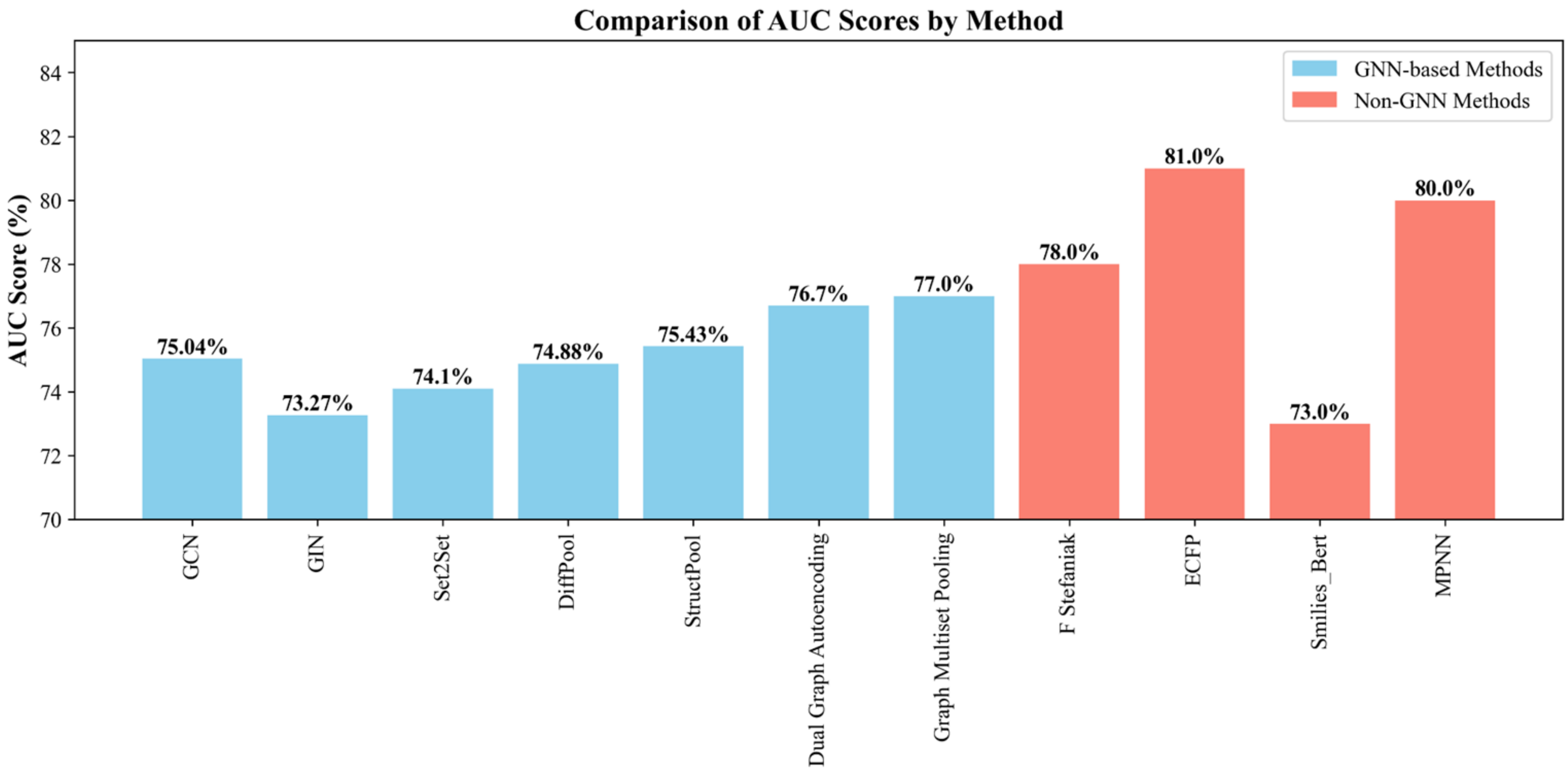
BCE with Logits Loss

Performance Report

Class-wise Results



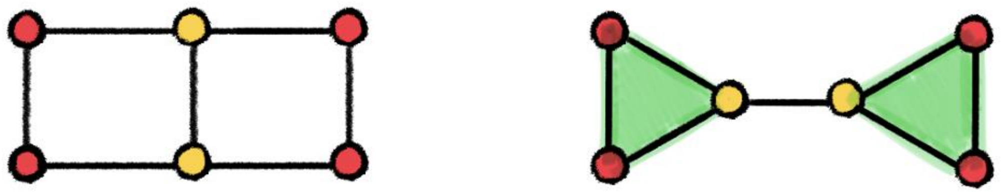
Comparison with SOTA methods

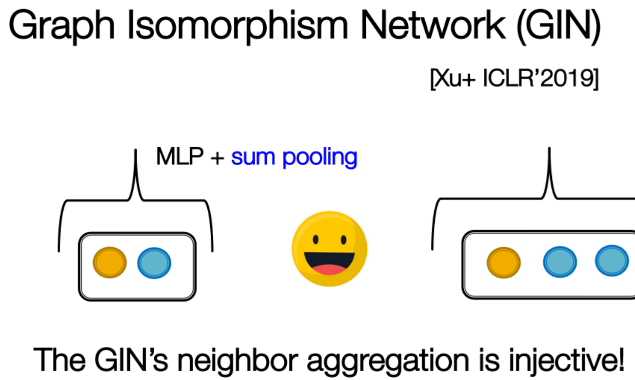


Comparison of AUC Scores by Method
AUC Score (%)
84
82
80
78
76
74
72
70
GNN-based Methods
Non-GNN Methods
75.04%
73.27%
74.1%
74.88%
75.43%
76.7%
77.0%
78.0%
81.0%
73.0%
80.0%
GCN
GIN
Set2Set
DiffPool
StructPool
Dual Graph Autoencoding
Graph Multiset Pooling
F Stefaniak
ECFP
Smilies_Bert
MPNN

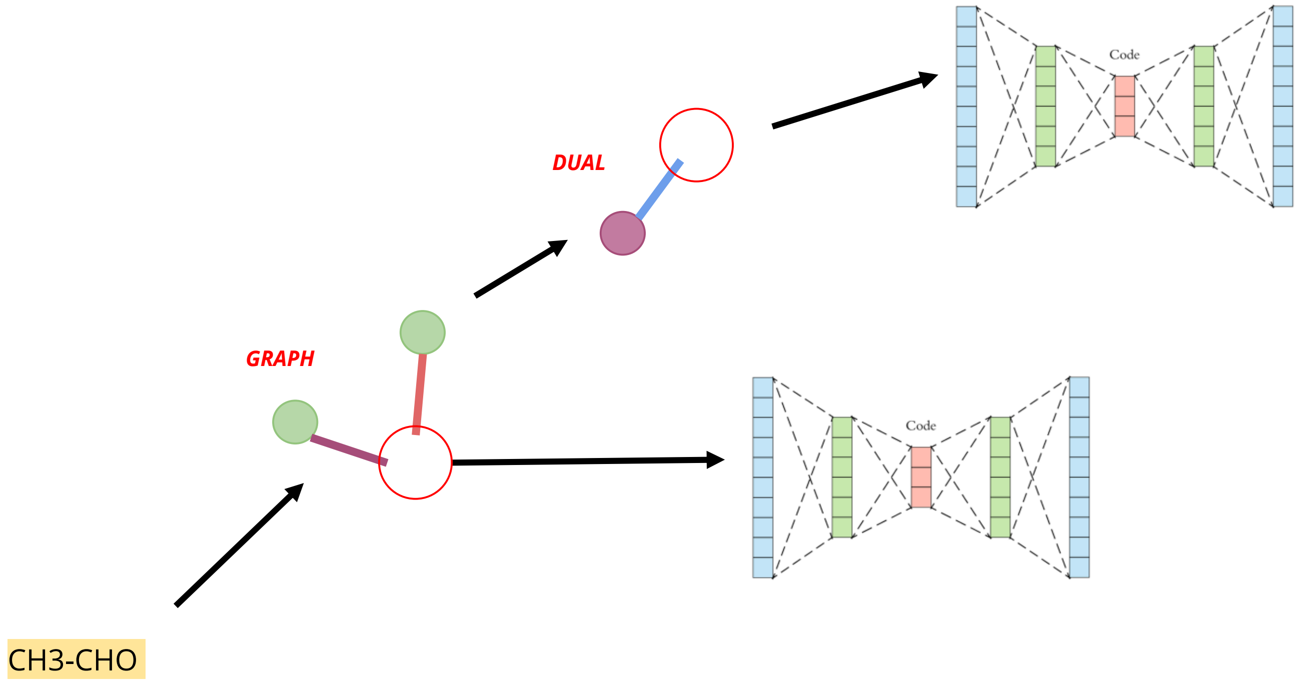
Challenges/Discussion

Graph Isomorphism
(Weisfeiler-Lehman graph isomorphism test)




Graph Isomorphism Network (GIN)
[Xu+ ICLR'2019]
MLP + sum pooling
The GIN's neighbor aggregation is injective!

GIN is theoretically the most expressive GNN

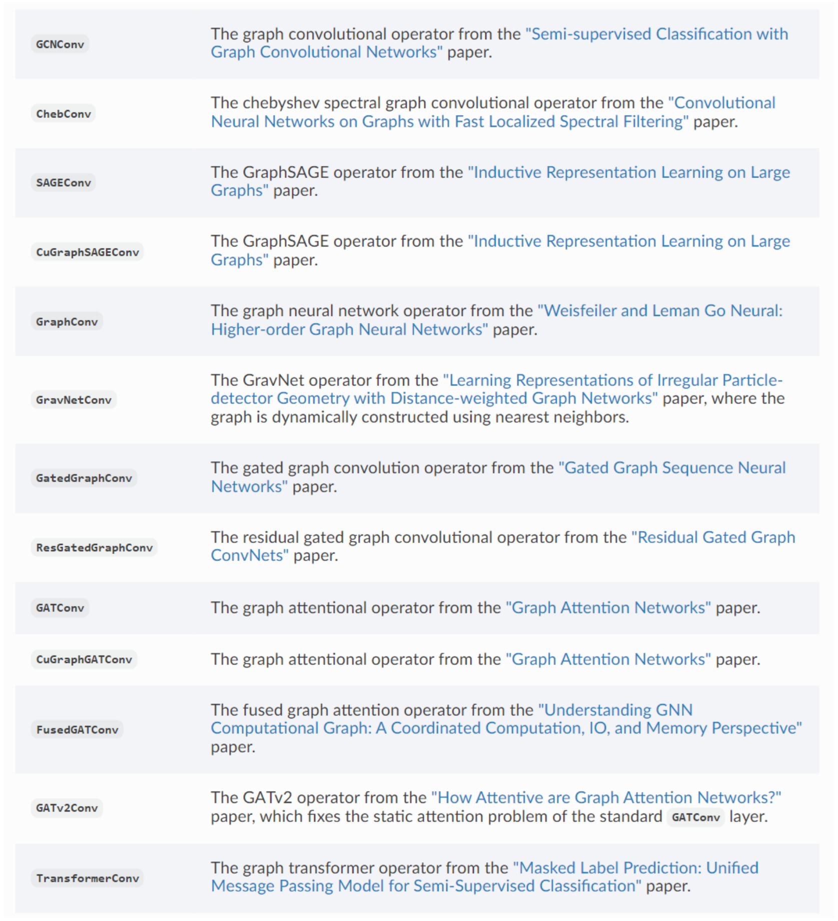


Dual Graph Autoencoder “Might” pass the WL-Isomorphism Test

Future Directions

Prospects

- Try it on different Datasets
- Try with GNNs other than GCNs
- Doing Preprocessing



`GCNConv`	The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper.
`ChebConv`	The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper.
`SAGEConv`	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper.
`CuGraphSAGEConv`	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper.
`GraphConv`	The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper.
`GravNetConv`	The GravNet operator from the "Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks" paper, where the graph is dynamically constructed using nearest neighbors.
`GatedGraphConv`	The gated graph convolution operator from the "Gated Graph Sequence Neural Networks" paper.
`ResGatedGraphConv`	The residual gated graph convolutional operator from the "Residual Gated Graph ConvNets" paper.
`GATConv`	The graph attentional operator from the "Graph Attention Networks" paper.
`CuGraphGATConv`	The graph attentional operator from the "Graph Attention Networks" paper.
`FusedGATConv`	The fused graph attention operator from the "Understanding GNN Computational Graph: A Coordinated Computation, IO, and Memory Perspective" paper.
`GATv2Conv`	The GATv2 operator from the "How Attentive are Graph Attention Networks?" paper, which fixes the static attention problem of the standard `GATConv` layer.
`TransformerConv`	The graph transformer operator from the "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification" paper.