Predicting the Blood-Brain Barrier (BBB) Penetrability and LogBB of PET Tracers Using Neural Networks

04.29.2025
QBIO 465 Final Project

Andrew Levy, Erika Li,
Tushar Zhade

Background on PET Imaging

- PET = Non-invasive imaging of biochemical processes in the brain.
- Requires tracers labeled with **positron-emitting isotopes**
- Gamma rays emitted upon positron-electron annihilation → image reconstruction.
- Blood-brain barrier (BBB) penetration is essential for brain imaging.

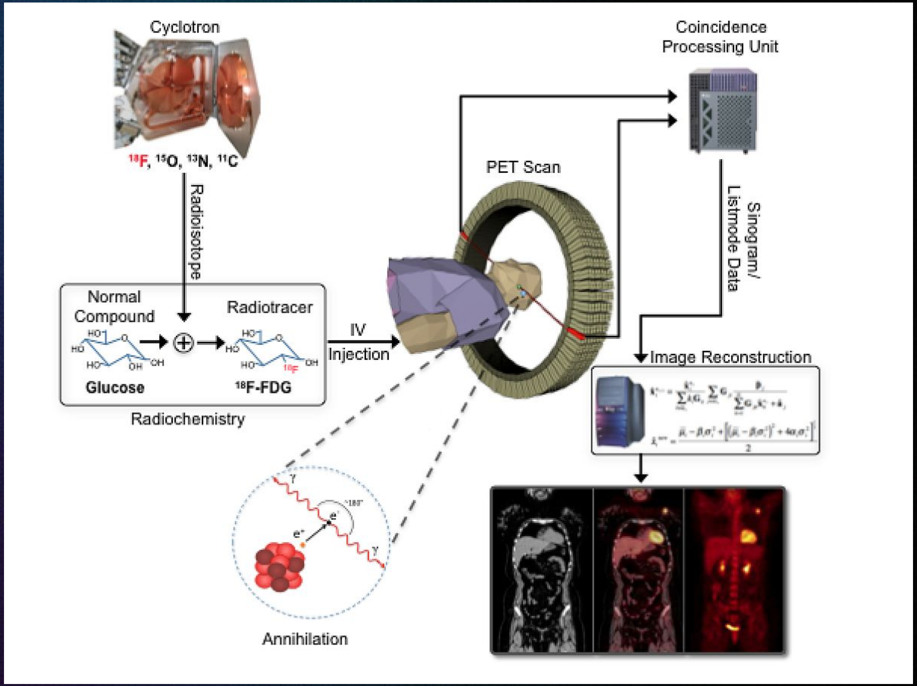


Image: University of Utah School of Medicine

The Problem

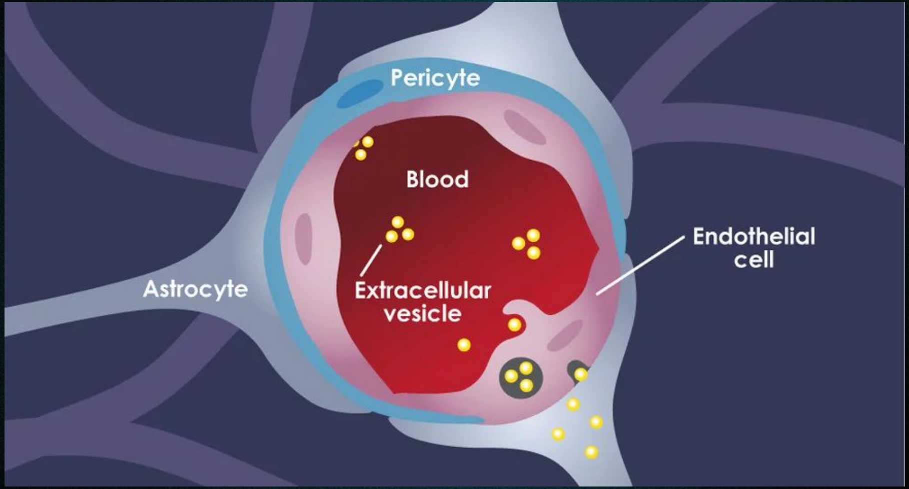


Image: Morad et al., 2021

- Tracers must cross the BBB to be effective.
- BBB permeability not always predictable from traditional drug properties.
- Existing models trained on general drug data may not generalize to PET tracers.

Project Goals

1. Classify whether a PET tracer is BBB-permeable.
2. Predict the logBB value (log[brain]/[blood] concentration).
3. Compare two neural network approaches:
    a. Descriptor-based MLP (fully-connected neural network)
    b. Graph-based convolutional neural network (GCN)
4. Evaluate model generalizability to PET tracers.

Datasets Used

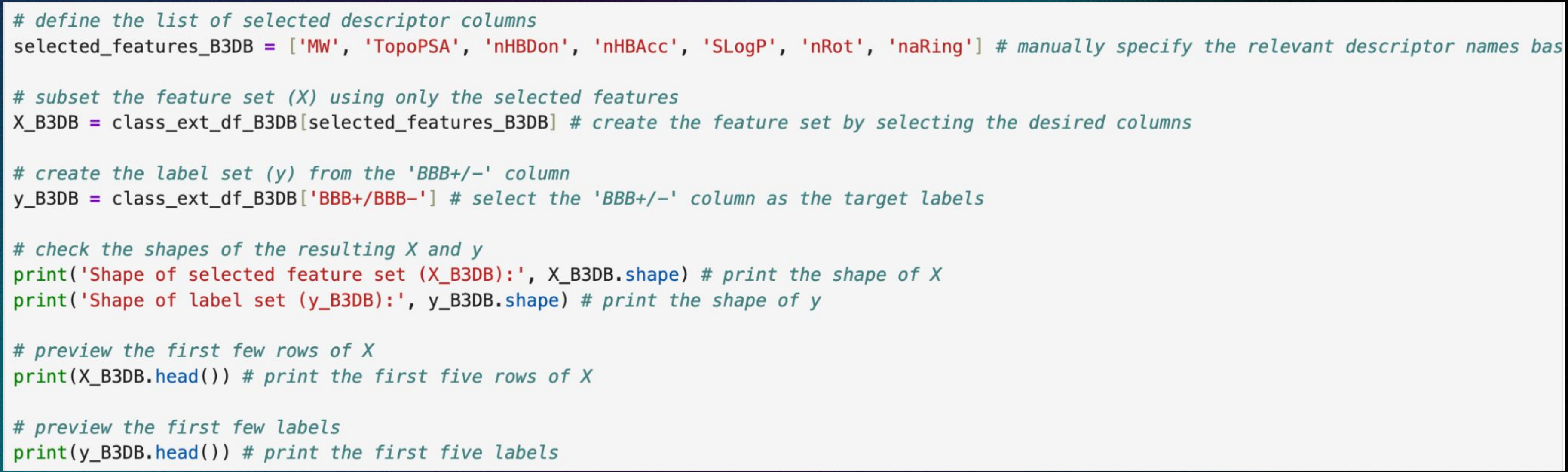
- Training Dataset: B3DB (Meng et al., 2021)
    - 7,807 compounds with BBB labels
    - 1,058 compounds with logBB values
- Test Set: Curated CNS PET tracers (Steen, 2022)
    - 130 compounds with BBB labels + logBB values
    - Emphasis on ^{11}C, ^{18}F, and ^{123}I radiolabeled compounds

Method: Feature Representations

- FCNN (Descriptor-Based):
  - Molecular weight, TPSA, H-bond donors/acceptors, number of rotatable bonds, number of aromatic rings, logP
  - Provided within B3DB dataset
- GNN (Graph-Based):
  - Molecule as a graph: atoms = nodes, bonds = edges
  - Uses graph convolutions to learn structure

Descriptor-Based FCNN

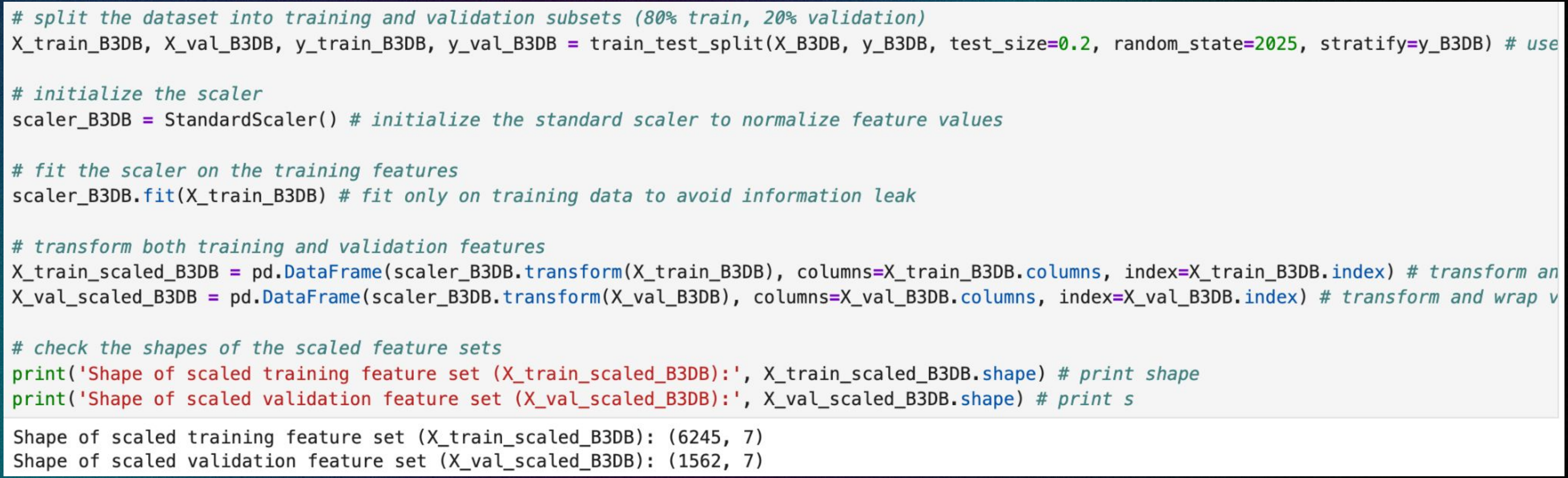
- Step 1: Select 7 physicochemical descriptors from B3DB dataset: MW, TPSA, HBD, HBA, logP, rotatable bonds, aromatic rings
- Step 2: Extract the Labels (BBB+ or BBB-)



```python
# define the list of selected descriptor columns
selected_features_B3DB = ['MW', 'TopoPSA', 'nHBDon', 'nHBAcc', 'SLogP', 'nRot', 'naRing'] # manually specify the relevant descriptor names bas

# subset the feature set (X) using only the selected features
X_B3DB = class_ext_df_B3DB[selected_features_B3DB] # create the feature set by selecting the desired columns

# create the label set (y) from the 'BBB+/-' column
y_B3DB = class_ext_df_B3DB['BBB+/BBB-'] # select the 'BBB+/-' column as the target labels

# check the shapes of the resulting X and y
print('Shape of selected feature set (X_B3DB):', X_B3DB.shape) # print the shape of X
print('Shape of label set (y_B3DB):', y_B3DB.shape) # print the shape of y

# preview the first few rows of X
print(X_B3DB.head()) # print the first five rows of X

# preview the first few labels
print(y_B3DB.head()) # print the first five labels
```

Descriptor-Based FCNN

- Step 3: Split data into training and validation sets (80%/20%), stratified by BBB+/BBB– labels
- Step 4: Standardize feature values using StandardScaler fitted only on training set

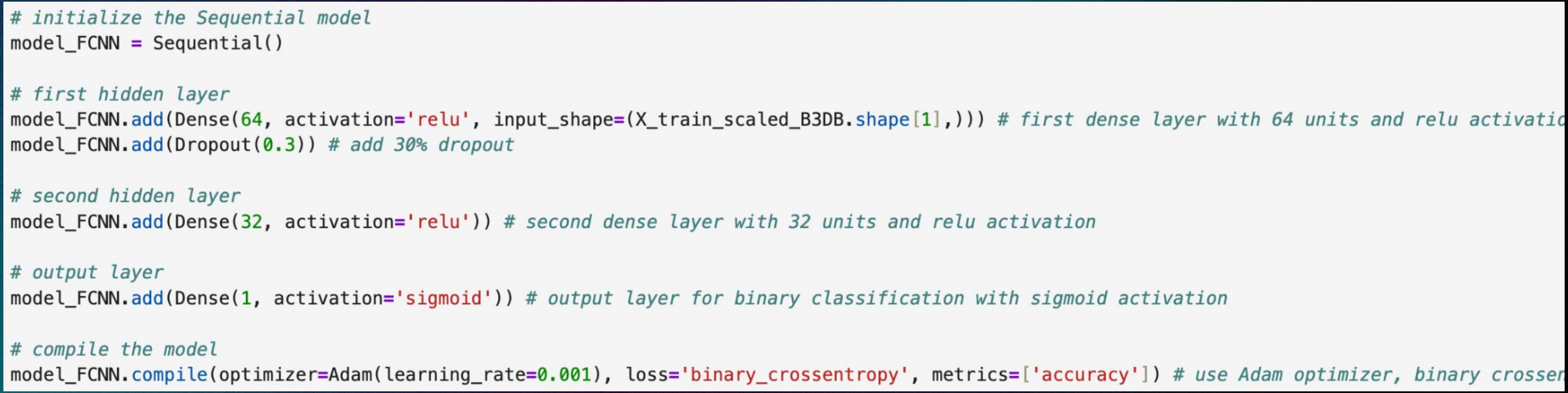


```
# split the dataset into training and validation subsets (80% train, 20% validation)
X_train_B3DB, X_val_B3DB, y_train_B3DB, y_val_B3DB = train_test_split(X_B3DB, y_B3DB, test_size=0.2, random_state=2025, stratify=y_B3DB) # use

# initialize the scaler
scaler_B3DB = StandardScaler() # initialize the standard scaler to normalize feature values

# fit the scaler on the training features
scaler_B3DB.fit(X_train_B3DB) # fit only on training data to avoid information leak

# transform both training and validation features
X_train_scaled_B3DB = pd.DataFrame(scaler_B3DB.transform(X_train_B3DB), columns=X_train_B3DB.columns, index=X_train_B3DB.index) # transform an
X_val_scaled_B3DB = pd.DataFrame(scaler_B3DB.transform(X_val_B3DB), columns=X_val_B3DB.columns, index=X_val_B3DB.index) # transform and wrap v

# check the shapes of the scaled feature sets
print('Shape of scaled training feature set (X_train_scaled_B3DB):', X_train_scaled_B3DB.shape) # print shape
print('Shape of scaled validation feature set (X_val_scaled_B3DB):', X_val_scaled_B3DB.shape) # print s
```

```
Shape of scaled training feature set (X_train_scaled_B3DB): (6245, 7)
Shape of scaled validation feature set (X_val_scaled_B3DB): (1562, 7)
```

Descriptor-Based FCNN

- Step 5: Build a Sequential Fully Connected Neural Network (FCNN) with:
  - Two hidden layers (64 and 32 neurons) using ReLU activation
  - Dropout regularization (30%) after first hidden layer
  - Output layer with sigmoid activation for binary classification

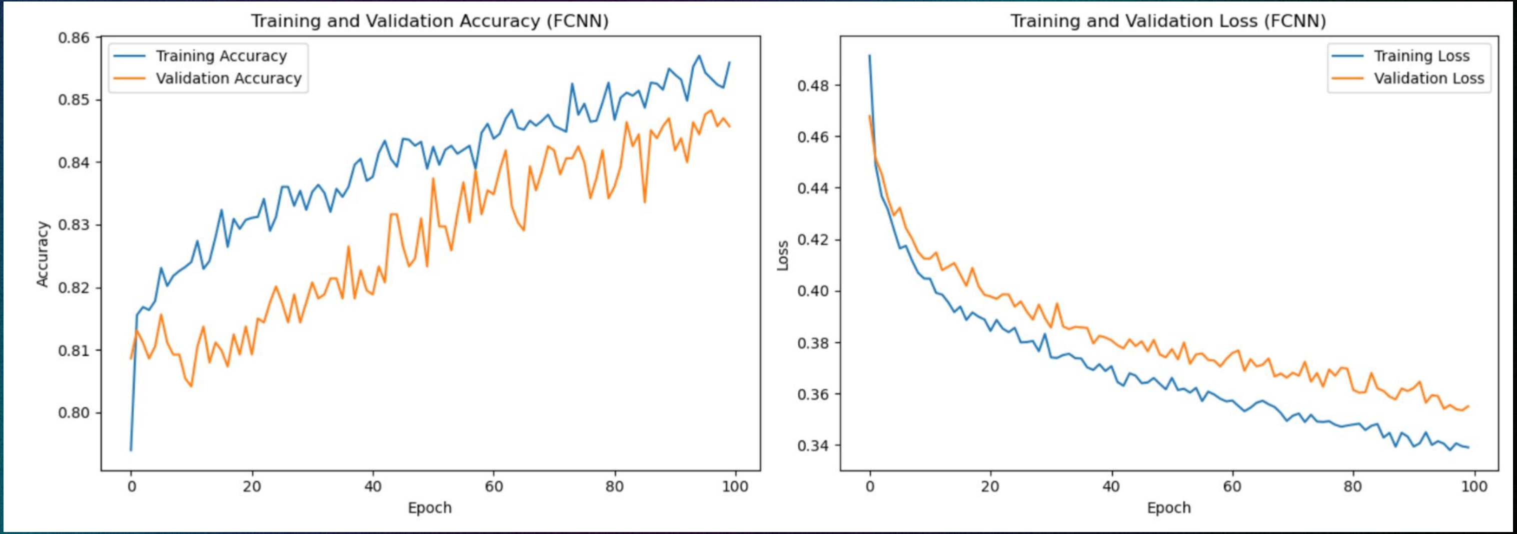


```python
# initialize the Sequential model
model_FCNN = Sequential()

# first hidden layer
model_FCNN.add(Dense(64, activation='relu', input_shape=(X_train_scaled_B3DB.shape[1],))) # first dense layer with 64 units and relu activatio
model_FCNN.add(Dropout(0.3)) # add 30% dropout

# second hidden layer
model_FCNN.add(Dense(32, activation='relu')) # second dense layer with 32 units and relu activation

# output layer
model_FCNN.add(Dense(1, activation='sigmoid')) # output layer for binary classification with sigmoid activation

# compile the model
model_FCNN.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy']) # use Adam optimizer, binary crossen
```

Descriptor-Based FCNN

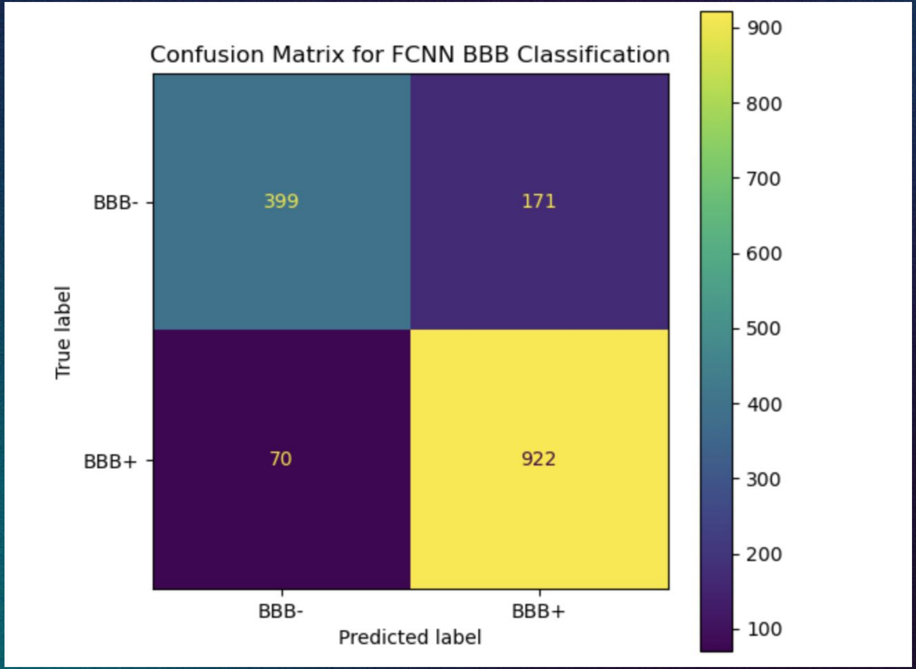
- Step 6: Train FCNN for 100 epochs using Adam optimizer and binary cross-entropy loss

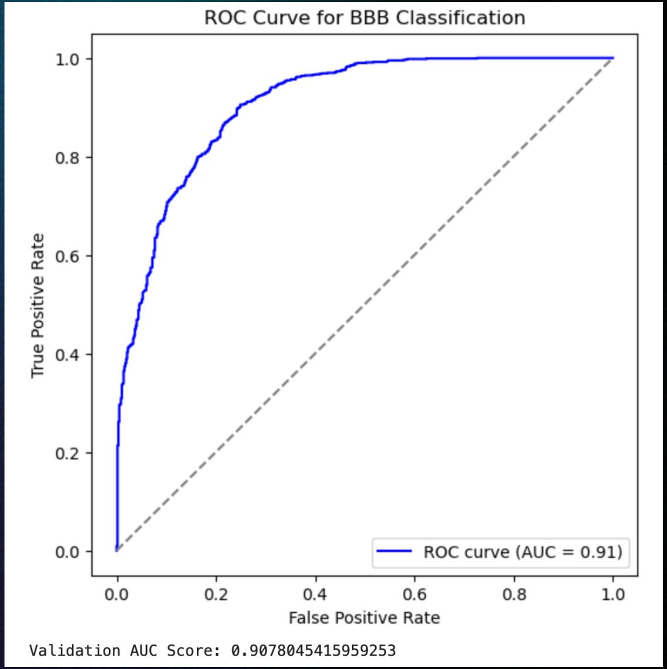
```
# train the model on the training data
history_FCNN = model_FCNN.fit(X_train_scaled_B3DB, y_train_B3DB.replace({'BBB-': 0, 'BBB+': 1}),
                               validation_data=(X_val_scaled_B3DB, y_val_B3DB.replace({'BBB-': 0, 'BBB+': 1})),
                               epochs=100, batch_size=32, verbose=1) # train for 100 epochs, batch size of 32
```



Descriptor-Based FCNN

- Step 7: Evaluate model using validation accuracy, confusion matrix, ROC curve, and classification report

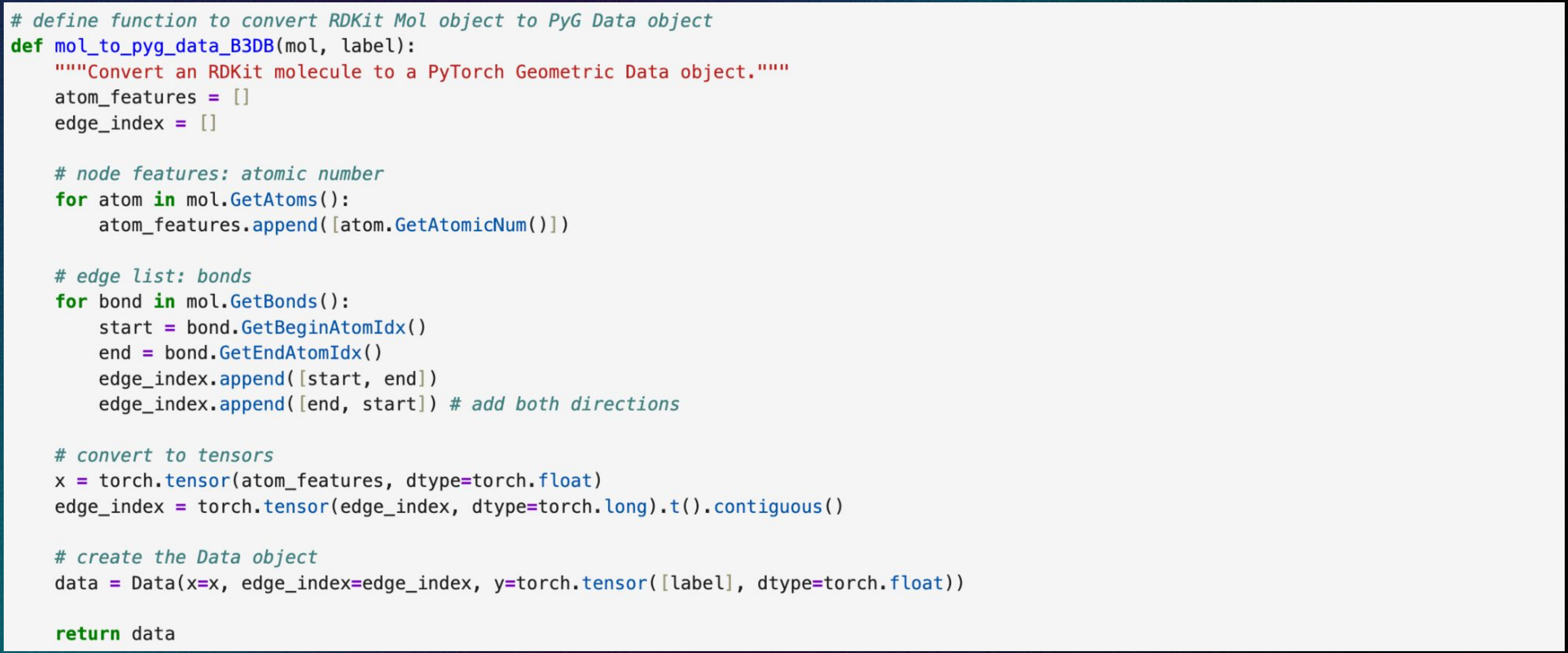




```
Validation AUC Score: 0.9078045415959253
```

Graph Convolutional Neural Network

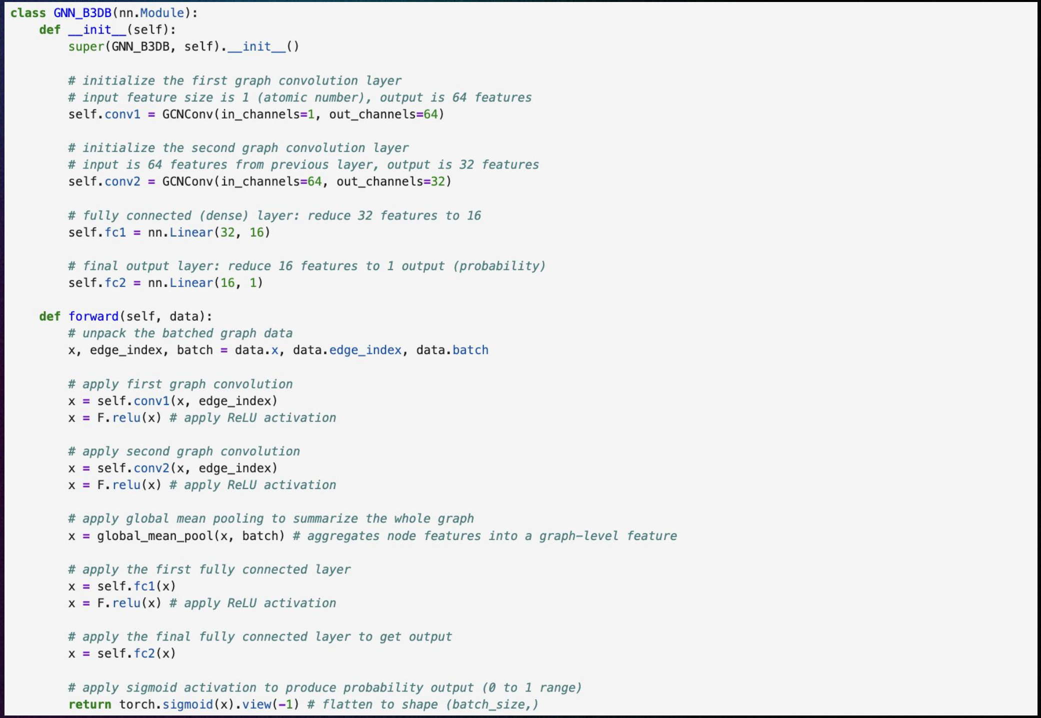
- Step 1: Convert SMILES strings from B3DB dataset into molecular graphs using RDKit



```python
# define function to convert RDKit Mol object to PyG Data object
def mol_to_pyg_data_B3DB(mol, label):
    """Convert an RDKit molecule to a PyTorch Geometric Data object."""
    atom_features = []
    edge_index = []

    # node features: atomic number
    for atom in mol.GetAtoms():
        atom_features.append([atom.GetAtomicNum()])

    # edge list: bonds
    for bond in mol.GetBonds():
        start = bond.GetBeginAtomIdx()
        end = bond.GetEndAtomIdx()
        edge_index.append([start, end])
        edge_index.append([end, start]) # add both directions

    # convert to tensors
    x = torch.tensor(atom_features, dtype=torch.float)
    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

    # create the Data object
    data = Data(x=x, edge_index=edge_index, y=torch.tensor([label], dtype=torch.float))

    return data
```

Graph Convolutional Neural Network

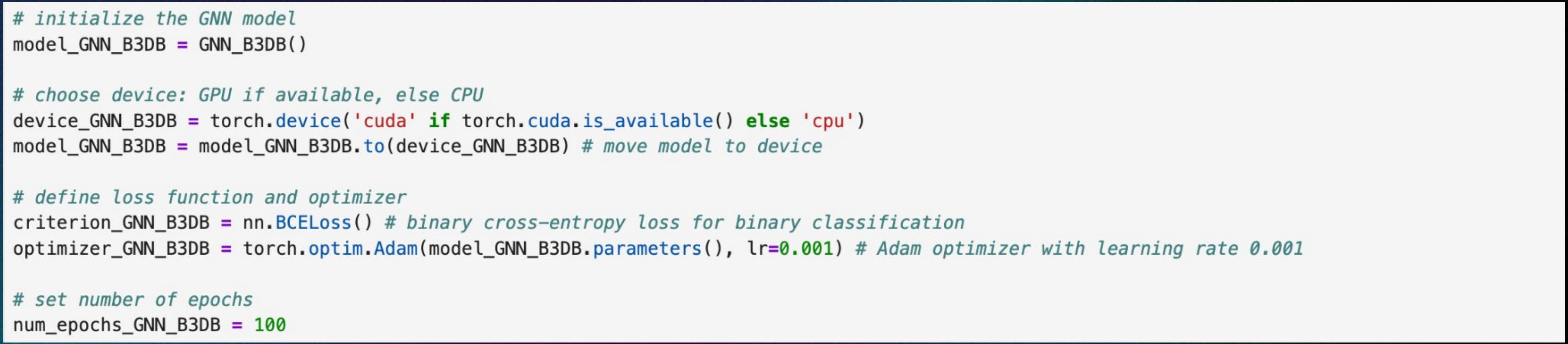
- Step 2: Build a Graph Convolutional Network (GCN) using PyTorch Geometric with:
  - Two GCNConv layers (64 and 32 hidden units) with ReLU activations
  - Global mean pooling to aggregate node features into a graph-level representation
  - Fully connected layers (32 → 16 → 1) with final sigmoid activation



```python
class GNN_B3DB(nn.Module):
    def __init__(self):
        super(GNN_B3DB, self).__init__()

        # initialize the first graph convolution layer
        # input feature size is 1 (atomic number), output is 64 features
        self.conv1 = GCNConv(in_channels=1, out_channels=64)

        # initialize the second graph convolution layer
        # input is 64 features from previous layer, output is 32 features
        self.conv2 = GCNConv(in_channels=64, out_channels=32)

        # fully connected (dense) layer: reduce 32 features to 16
        self.fc1 = nn.Linear(32, 16)

        # final output layer: reduce 16 features to 1 output (probability)
        self.fc2 = nn.Linear(16, 1)

    def forward(self, data):
        # unpack the batched graph data
        x, edge_index, batch = data.x, data.edge_index, data.batch

        # apply first graph convolution
        x = self.conv1(x, edge_index)
        x = F.relu(x) # apply ReLU activation

        # apply second graph convolution
        x = self.conv2(x, edge_index)
        x = F.relu(x) # apply ReLU activation

        # apply global mean pooling to summarize the whole graph
        x = global_mean_pool(x, batch) # aggregates node features into a graph-level feature

        # apply the first fully connected layer
        x = self.fc1(x)
        x = F.relu(x) # apply ReLU activation

        # apply the final fully connected layer to get output
        x = self.fc2(x)

        # apply sigmoid activation to produce probability output (0 to 1 range)
        return torch.sigmoid(x).view(-1) # flatten to shape (batch_size,)
```

Graph Convolutional Neural Network

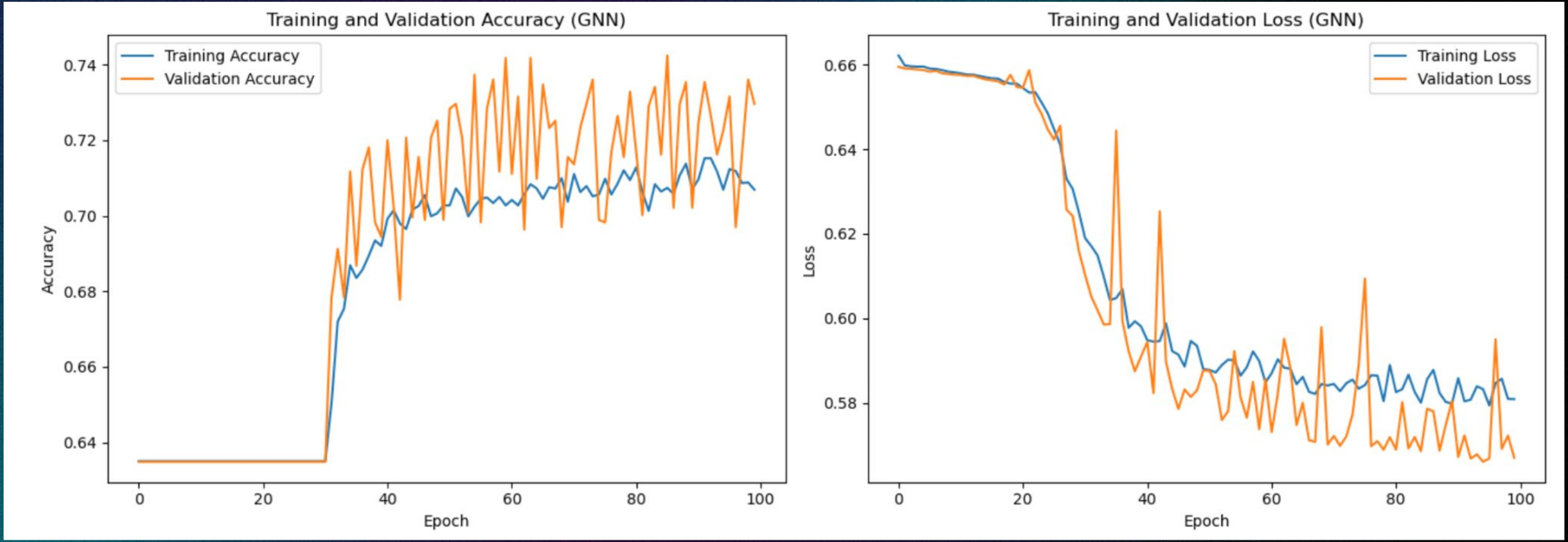
- Step 3: Train GCN on molecular graphs using binary cross-entropy loss and Adam optimizer



```python
# initialize the GNN model
model_GNN_B3DB = GNN_B3DB()

# choose device: GPU if available, else CPU
device_GNN_B3DB = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_GNN_B3DB = model_GNN_B3DB.to(device_GNN_B3DB) # move model to device

# define loss function and optimizer
criterion_GNN_B3DB = nn.BCELoss() # binary cross-entropy loss for binary classification
optimizer_GNN_B3DB = torch.optim.Adam(model_GNN_B3DB.parameters(), lr=0.001) # Adam optimizer with learning rate 0.001

# set number of epochs
num_epochs_GNN_B3DB = 100
```

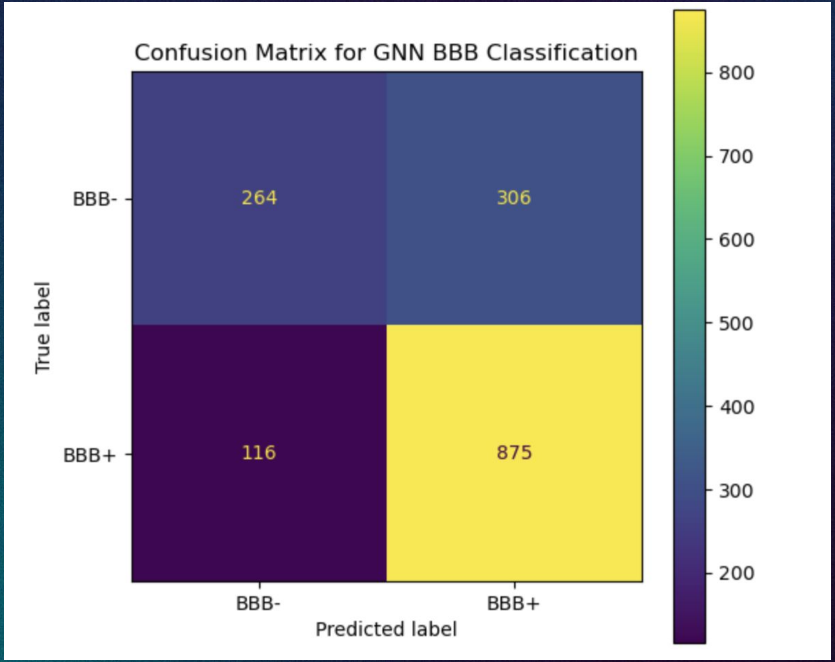
Graph Convolutional Neural Network

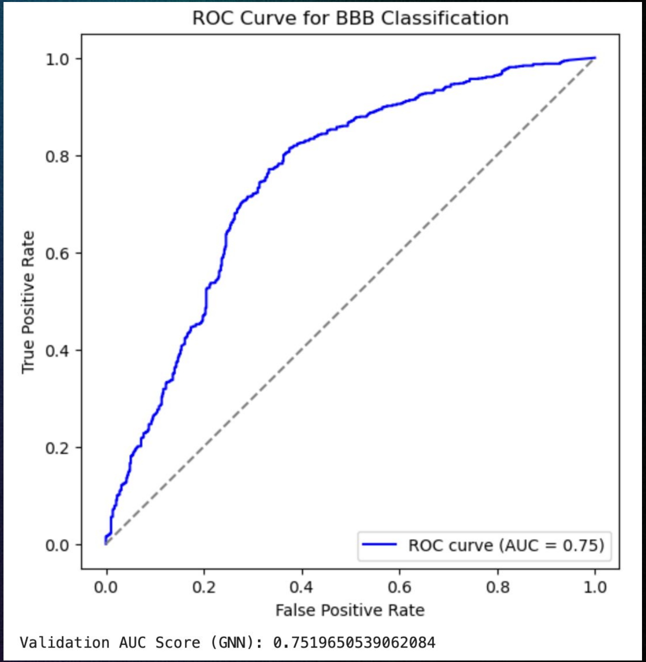
- Step 3: Train GCN on molecular graphs using binary cross-entropy loss and Adam optimizer



Graph Convolutional Neural Network

- Step 4: Evaluate model using validation accuracy, confusion matrix, ROC curve, and classification report

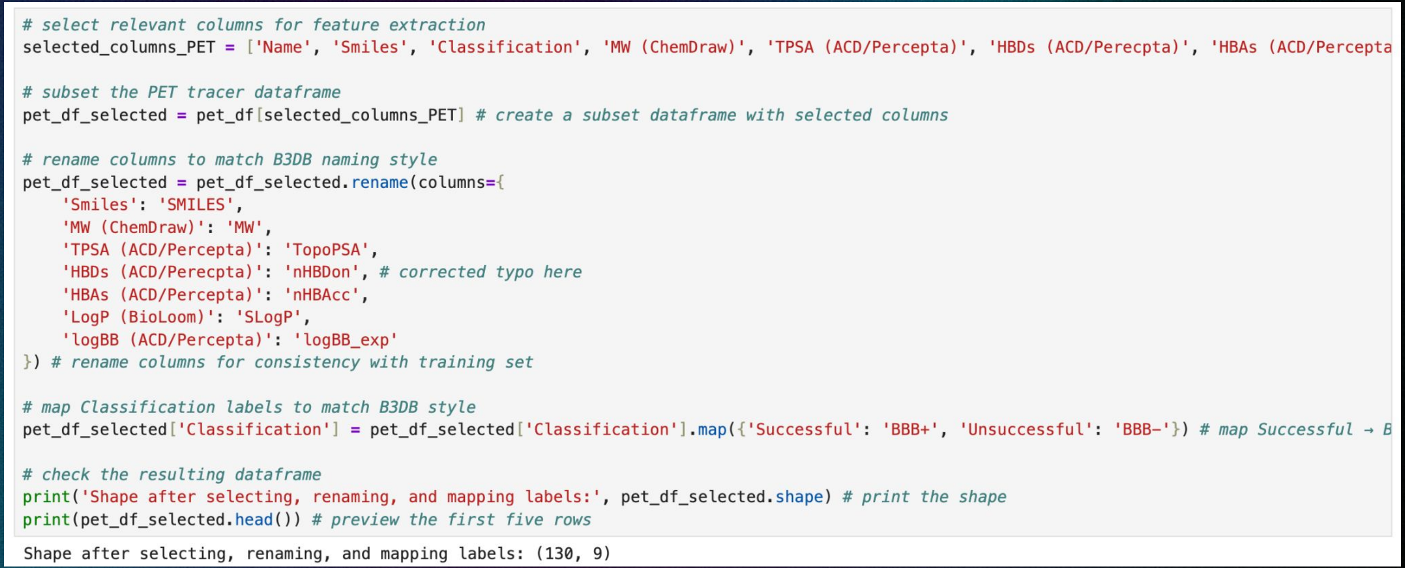




```
Validation AUC Score (GNN): 0.7519650539062084
```

Evaluate Models on PET Tracer Data

- Step 1: Preprocessed CNS PET tracer dataset:
  - Extracted relevant descriptors (MW, TPSA, HBD, HBA, logP)
  - Calculated additional features (rotatable bonds, aromatic rings) using RDKit
  - Mapped Classification labels ('Successful' → BBB+, 'Unsuccessful' → BBB−)
  - Standardized features using the same scaler fitted on B3DB training set

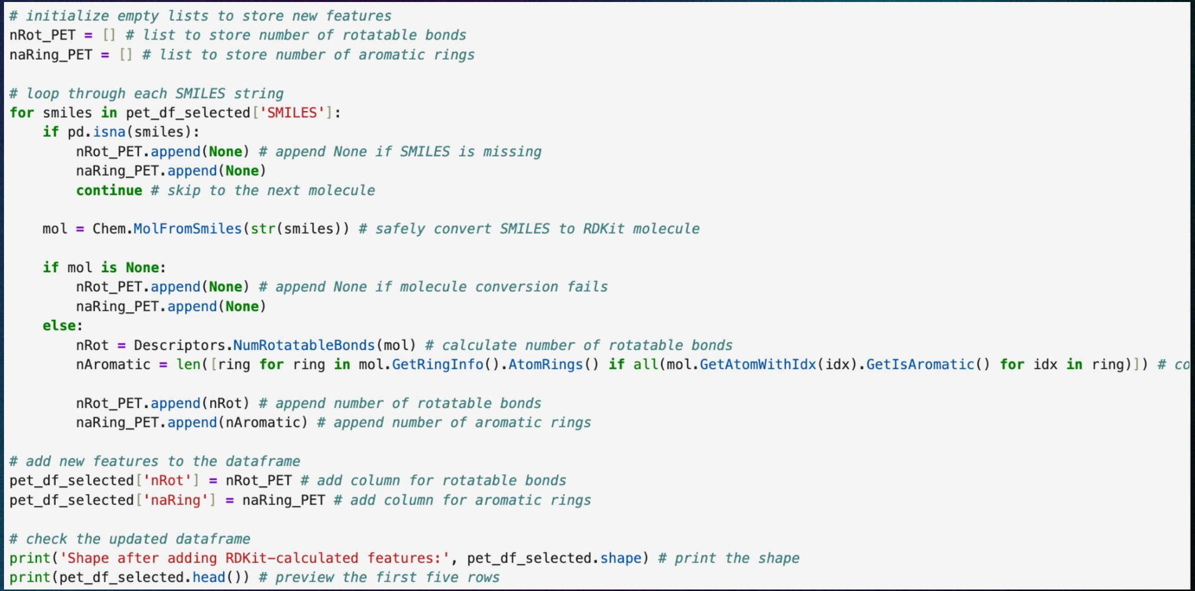


```
# select relevant columns for feature extraction
selected_columns_PET = ['Name', 'Smiles', 'Classification', 'MW (ChemDraw)', 'TPSA (ACD/Percepta)', 'HBDs (ACD/Perecpta)', 'HBAs (ACD/Percepta

# subset the PET tracer dataframe
pet_df_selected = pet_df[selected_columns_PET] # create a subset dataframe with selected columns

# rename columns to match B3DB naming style
pet_df_selected = pet_df_selected.rename(columns={
    'Smiles': 'SMILES',
    'MW (ChemDraw)': 'MW',
    'TPSA (ACD/Percepta)': 'TopoPSA',
    'HBDs (ACD/Perecpta)': 'nHBDon', # corrected typo here
    'HBAs (ACD/Percepta)': 'nHBAcc',
    'LogP (BioLoom)': 'SLogP',
    'logBB (ACD/Percepta)': 'logBB_exp'
}) # rename columns for consistency with training set

# map Classification labels to match B3DB style
pet_df_selected['Classification'] = pet_df_selected['Classification'].map({'Successful': 'BBB+', 'Unsuccessful': 'BBB-'}) # map Successful → B

# check the resulting dataframe
print('Shape after selecting, renaming, and mapping labels:', pet_df_selected.shape) # print the shape
print(pet_df_selected.head()) # preview the first five rows
```

```
Shape after selecting, renaming, and mapping labels: (130, 9)
```

Evaluate Models on PET Tracer Data

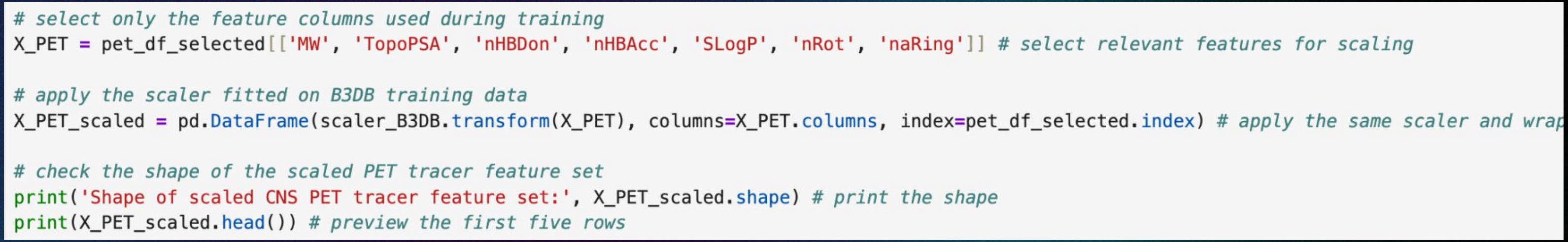
- Step 1: Preprocessed CNS PET tracer dataset:
  - Extracted relevant descriptors (MW, TPSA, HBD, HBA, logP)
  - Calculated additional features (rotatable bonds, aromatic rings) using RDKit
  - Mapped Classification labels ('Successful' → BBB+, 'Unsuccessful' → BBB–)
  - Standardized features using the same scaler fitted on B3DB training set



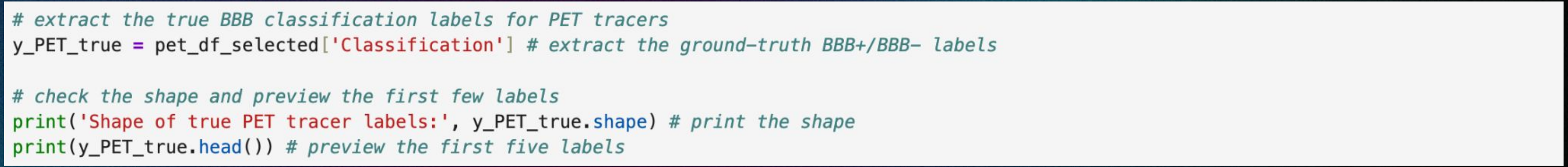
```python
# initialize empty lists to store new features
nRot_PET = [] # list to store number of rotatable bonds
naRing_PET = [] # list to store number of aromatic rings

# loop through each SMILES string
for smiles in pet_df_selected['SMILES']:
    if pd.isna(smiles):
        nRot_PET.append(None) # append None if SMILES is missing
        naRing_PET.append(None)
        continue # skip to the next molecule

    mol = Chem.MolFromSmiles(str(smiles)) # safely convert SMILES to RDKit molecule

    if mol is None:
        nRot_PET.append(None) # append None if molecule conversion fails
        naRing_PET.append(None)
    else:
        nRot = Descriptors.NumRotatableBonds(mol) # calculate number of rotatable bonds
        nAromatic = len([ring for ring in mol.GetRingInfo().AtomRings() if all(mol.GetAtomWithIdx(idx).GetIsAromatic() for idx in ring)]) # co

        nRot_PET.append(nRot) # append number of rotatable bonds
        naRing_PET.append(nAromatic) # append number of aromatic rings

# add new features to the dataframe
pet_df_selected['nRot'] = nRot_PET # add column for rotatable bonds
pet_df_selected['naRing'] = naRing_PET # add column for aromatic rings

# check the updated dataframe
print('Shape after adding RDKit-calculated features:', pet_df_selected.shape) # print the shape
print(pet_df_selected.head()) # preview the first five rows
```

Evaluate Models on PET Tracer Data

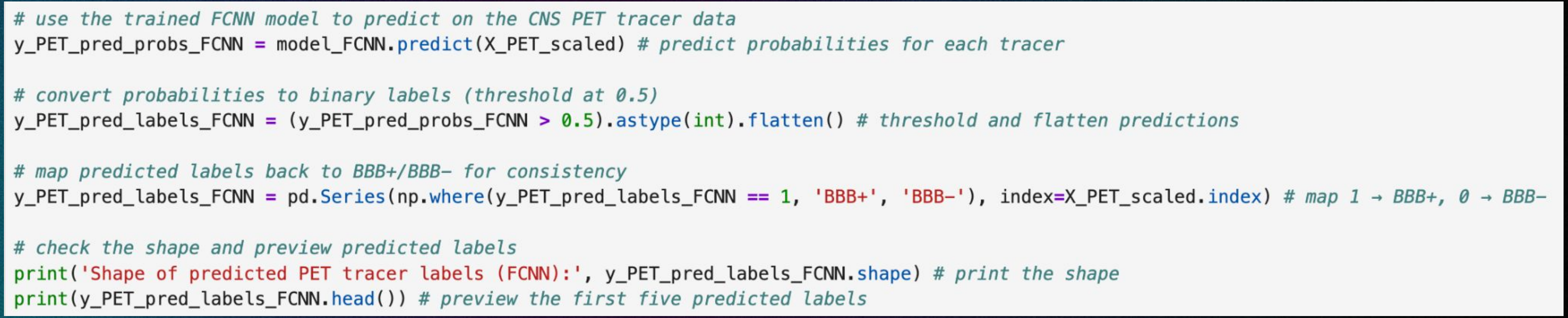
- Step 2: Apply trained FCNN to predict BBB penetrability from descriptor features



```python
# select only the feature columns used during training
X_PET = pet_df_selected[['MW', 'TopoPSA', 'nHBDon', 'nHBAcc', 'SLogP', 'nRot', 'naRing']] # select relevant features for scaling

# apply the scaler fitted on B3DB training data
X_PET_scaled = pd.DataFrame(scaler_B3DB.transform(X_PET), columns=X_PET.columns, index=pet_df_selected.index) # apply the same scaler and wrap

# check the shape of the scaled PET tracer feature set
print('Shape of scaled CNS PET tracer feature set:', X_PET_scaled.shape) # print the shape
print(X_PET_scaled.head()) # preview the first five rows
```



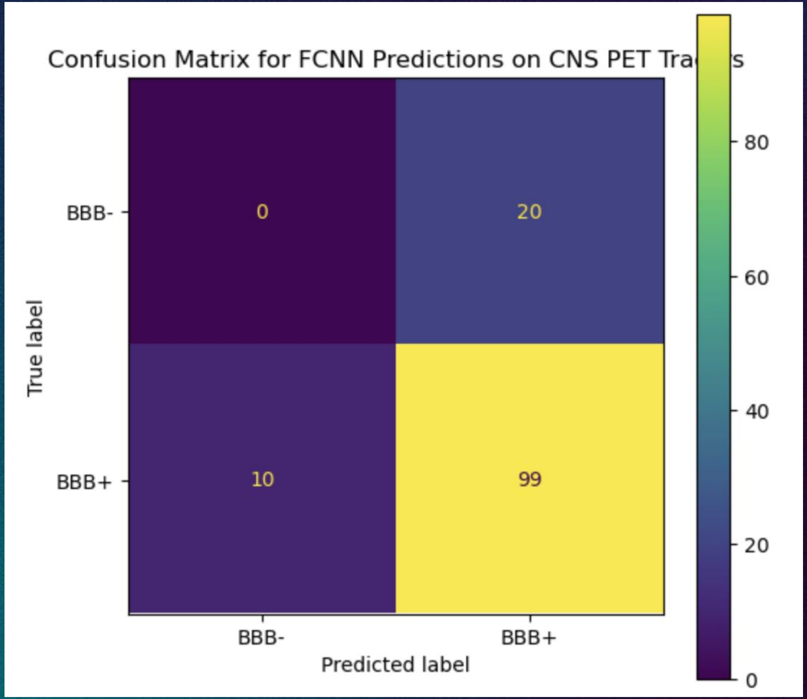
```python
# extract the true BBB classification labels for PET tracers
y_PET_true = pet_df_selected['Classification'] # extract the ground-truth BBB+/BBB- labels

# check the shape and preview the first few labels
print('Shape of true PET tracer labels:', y_PET_true.shape) # print the shape
print(y_PET_true.head()) # preview the first five labels
```

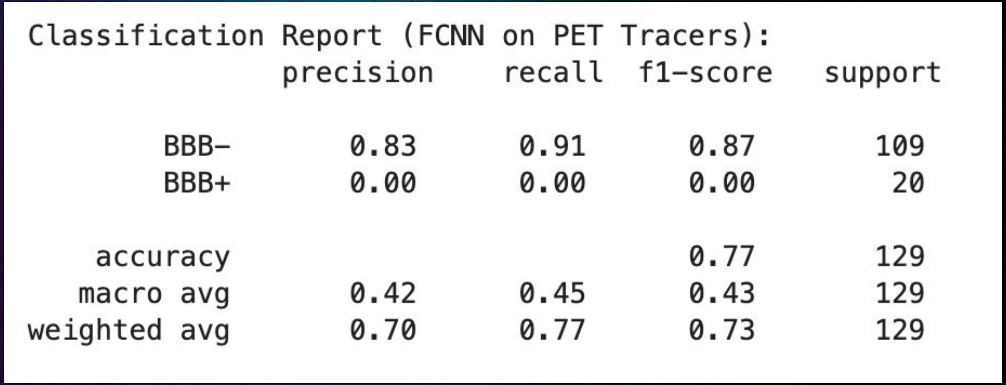


```python
# use the trained FCNN model to predict on the CNS PET tracer data
y_PET_pred_probs_FCNN = model_FCNN.predict(X_PET_scaled) # predict probabilities for each tracer

# convert probabilities to binary labels (threshold at 0.5)
y_PET_pred_labels_FCNN = (y_PET_pred_probs_FCNN > 0.5).astype(int).flatten() # threshold and flatten predictions

# map predicted labels back to BBB+/BBB- for consistency
y_PET_pred_labels_FCNN = pd.Series(np.where(y_PET_pred_labels_FCNN == 1, 'BBB+', 'BBB-'), index=X_PET_scaled.index) # map 1 → BBB+, 0 → BBB-

# check the shape and preview predicted labels
print('Shape of predicted PET tracer labels (FCNN):', y_PET_pred_labels_FCNN.shape) # print the shape
print(y_PET_pred_labels_FCNN.head()) # preview the first five predicted labels
```

Evaluate Models on PET Tracer Data

- Step 3: Evaluate performance of FCNN

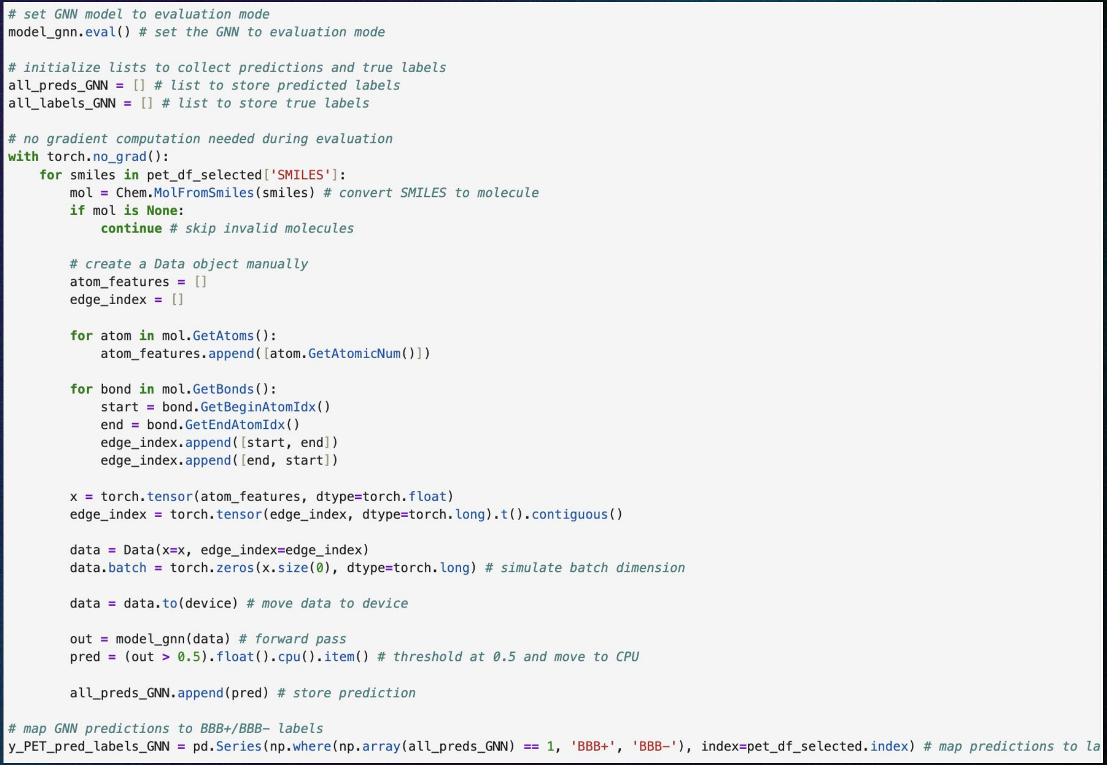




```
Classification Report (FCNN on PET Tracers):
              precision    recall  f1-score   support

        BBB−       0.83      0.91      0.87       109
        BBB+       0.00      0.00      0.00        20

    accuracy                           0.77       129
   macro avg       0.42      0.45      0.43       129
weighted avg       0.70      0.77      0.73       129
```

Evaluate Models on PET Tracer Data

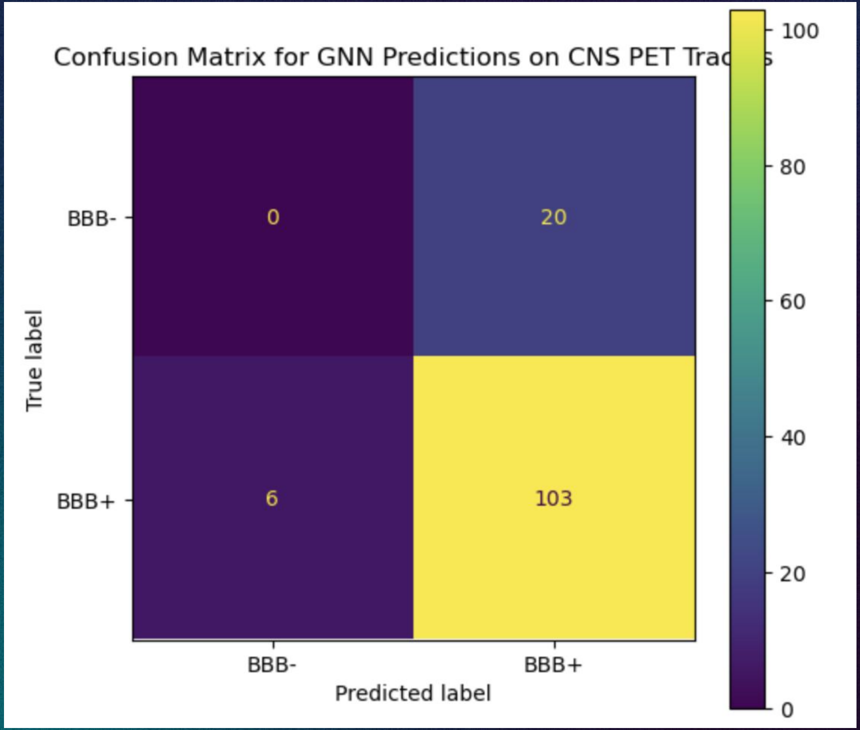
- Step 4: Convert CNS PET tracer SMILES to molecular graphs and apply trained GCN for prediction

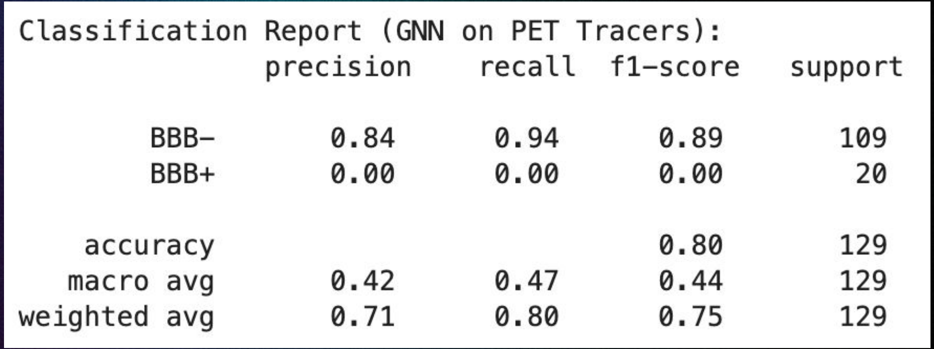


```
# set GNN model to evaluation mode
model_gnn.eval() # set the GNN to evaluation mode

# initialize lists to collect predictions and true labels
all_preds_GNN = [] # list to store predicted labels
all_labels_GNN = [] # list to store true labels

# no gradient computation needed during evaluation
with torch.no_grad():
    for smiles in pet_df_selected['SMILES']:
        mol = Chem.MolFromSmiles(smiles) # convert SMILES to molecule
        if mol is None:
            continue # skip invalid molecules

        # create a Data object manually
        atom_features = []
        edge_index = []

        for atom in mol.GetAtoms():
            atom_features.append([atom.GetAtomicNum()])

        for bond in mol.GetBonds():
            start = bond.GetBeginAtomIdx()
            end = bond.GetEndAtomIdx()
            edge_index.append([start, end])
            edge_index.append([end, start])

        x = torch.tensor(atom_features, dtype=torch.float)
        edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

        data = Data(x=x, edge_index=edge_index)
        data.batch = torch.zeros(x.size(0), dtype=torch.long) # simulate batch dimension

        data = data.to(device) # move data to device

        out = model_gnn(data) # forward pass
        pred = (out > 0.5).float().cpu().item() # threshold at 0.5 and move to CPU

        all_preds_GNN.append(pred) # store prediction

# map GNN predictions to BBB+/BBB- labels
y_PET_pred_labels_GNN = pd.Series(np.where(np.array(all_preds_GNN) == 1, 'BBB+', 'BBB-'), index=pet_df_selected.index) # map predictions to la
```

Evaluate Models on PET Tracer Data

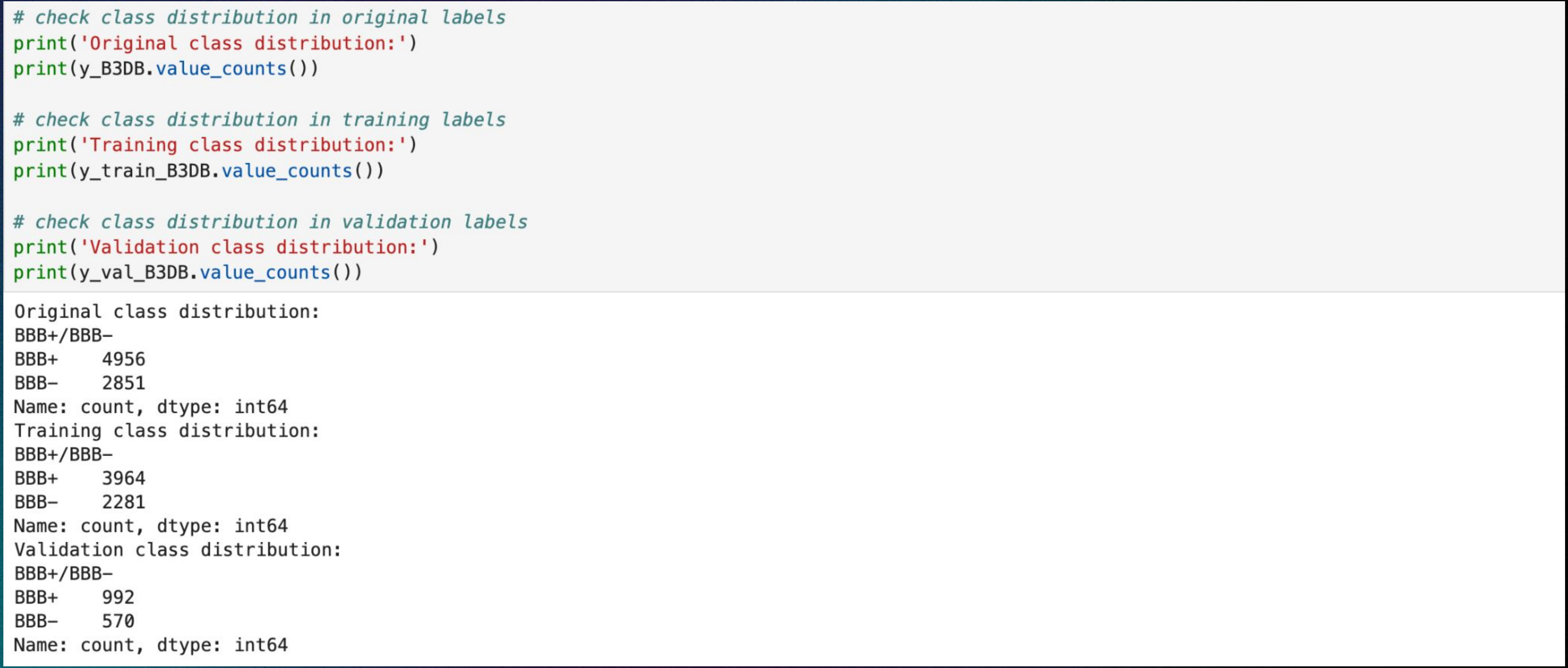
- Step 5: Evaluate performance of GCN





```
Classification Report (GNN on PET Tracers):
              precision    recall  f1-score   support

        BBB–       0.84      0.94      0.89       109
        BBB+       0.00      0.00      0.00        20

    accuracy                           0.80       129
   macro avg       0.42      0.47      0.44       129
weighted avg       0.71      0.80      0.75       129
```

Main Issue: Class Imbalance



```python
# check class distribution in original labels
print('Original class distribution:')
print(y_B3DB.value_counts())

# check class distribution in training labels
print('Training class distribution:')
print(y_train_B3DB.value_counts())

# check class distribution in validation labels
print('Validation class distribution:')
print(y_val_B3DB.value_counts())
```

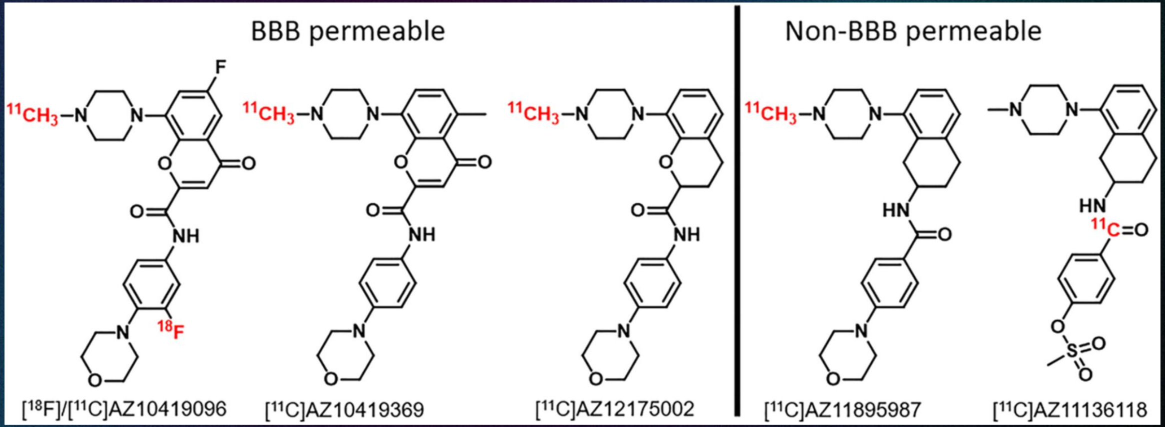
```
Original class distribution:
BBB+/BBB-
BBB+    4956
BBB-    2851
Name: count, dtype: int64
Training class distribution:
BBB+/BBB-
BBB+    3964
BBB-    2281
Name: count, dtype: int64
Validation class distribution:
BBB+/BBB-
BBB+    992
BBB-    570
Name: count, dtype: int64
```

Next Steps

1. Address the class imbalance observed in the training set:
   a. Either undersample the abundant BBB+ data
   b. Or incorporate additional BBB– molecules from external sources
2. Retrain the FCNN and GCN models on the balanced dataset and aim to obtain more reliable results
3. Compare model performance using precision, recall, and F1-score to determine which architecture better predicts BBB penetrability
4. Using the better-performing model, slightly modify the architecture to support logBB regression instead of classification
5. Train the adjusted model to predict logBB values
6. Apply the trained logBB regression model to the CNS PET tracer dataset and evaluate predictive performance

Expected Results and Limitations

- General drug-trained models are expected to **generalize well to PET tracers**
- Descriptor-based MLP (FCNN) is expected to outperform the GCN
- BBB permeability correlates more strongly with molecular descriptors (e.g., logP, hydrogen bonding) than with detailed structure
- Limitations
  - Lack of pharmacokinetic data (e.g., metabolism, protein binding) may constrain model performance
  - Structural models (GCNs) may underperform without CNS-specific fine-tuning



References

Meng et al. (2021). A curated diverse molecular database of blood-brain barrier permeability with chemical descriptors. *Scientific Data*. https://doi-org.libproxy2.usc.edu/10.1038/s41597-021-01069-5

Morad, G., Carman, C. V., Hagedorn, E. J., Perlin, J. R., Zon, L. I., Mustafaoglu, N., Park, T.-E., Ingber, D. E., Daisy, C. C., & Moses, M. A. (2019). Tumor-Derived Extracellular Vesicles Breach the Intact Blood–Brain Barrier *via* Transcytosis. *ACS Nano*, *13*(12), 13853–13865. https://doi.org/10.1021/acsnano.9b04397

PET | Radiology | U of U School of Medicine. (2021, November 10). Medicine.utah.edu. https://medicine.utah.edu/radiology/research/learn/pet

Steen. (2022). The Application of in silico Methods for Prediction of Blood-Brain Barrier Permeability of Small Molecule PET Tracers. *Frontiers in Nuclear Medicine*. https://doi.org/10.3389/fnume.2022.853475

Thank you!