Open in Colab

CS598 DL4H: Project Notebook - Draft

Paper Title - CNN-DDI: a learning-based method for predicting drug–drug interactions using convolution neural networks

Members:

- Avinash Baldeo (abaldeo2@illinois.edu)
- Jinfeng Wu (jinfeng4@illinois.edu)
- Hao Zhang (haoz8@illinois.edu)

Introduction

Background of the Problem

1. Type of Problem

The problem addressed in the paper is the prediction of drug-drug interactions (DDIs). Specifically, the research focuses on developing a computational method to predict whether two drugs will interact and the type of interaction they may have, which can be synergistic, antagonistic, or neutral (no reaction). This is a problem of feature engineering and predictive modeling within the domain of bioinformatics and pharmacology.

2. Importance of the Problem

Predicting DDIs is crucial for pharmaceuticals development for several reasons:

- Antagonistic interactions can lead to reduced drug efficacy or increased toxicity, which can harm patients. Predicting such interactions can prevent adverse drug events.
- Early prediction of DDIs can save time and resources in the drug development process by identifying potential issues before clinical trials.
- Understanding DDIs is key to customizing drug regimens for individual patients, especially those on multiple medications.
- By avoiding adverse drug reactions, we can reduce hospital readmissions and other healthcare costs associated with managing drug complications.
- Knowing if two drugs interact is also useful since drugs similar to either of the two are more likely to interact and cause the same effect.

3. Difficulty of the Problem

The prediction of DDIs is challenging due to:

- The way drugs interact can be influenced by numerous factors, including genetics, lifestyle, and the presence of other drugs.
- Drug data is typically heterogenous and comes from various sources and in different formats, making it difficult to integrate and analyze.
- Drugs can be described by many features, such as chemical properties, targets, and pathways, leading to high-dimensional data that is hard to process.
- Many potential drug combinations have not been observed or recorded, leading to sparse data and making it difficult to predict interactions for new or less-studied drugs.

4. State of the Art Methods and Effectiveness

The paper reviews mentions traditional and state-of-the-art methods for predicting DDIs:

- Traditional methods include text mining and statistical methods, which have been the foundation for early DDI prediction efforts.
- Logistic regression and other classical machine learning models have been used to predict DDIs by analyzing drug features.
- More recent studies have applied deep learning methods, such as Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs), to predict DDIs. These methods can automatically learn complex representations of drug features.

Paper Explanation

1. What did the paper propose?

The paper proposed a novel algorithm named CNN-DDI, which utilizes a convolutional neural network (CNN) architecture to predict drug-

The paper proposed a novel algorithm named CNN-DDI, which utilizes a convolutional neural network (CNN) architecture to predict drug drug interactions (DDIs). The method involves two main components:

- This feature selection part of the algorithm calculates the similarity between drug feature vectors using measures like Jaccard similarity. It then optimizes the data from high dimension to low dimension to generate feature vectors as input for the prediction module.
- The prediction module uses the feature vectors from the selection module as inputs to predict DDI-associated events. The CNN model consists of five convolutional layers, two fully connected layers, a residual block and a softmax layer.

2. What are the innovations of the method?

The main innovations of the CNN-DDI paper include:

- The method uses a combination of drug categories, targets, pathways, and enzymes as feature vectors, which was unique in the context of DDI prediction.
- While deep neural networks (DNNs) have been used in the past for similar tasks, the paper highlights the advantages of CNNs in feature learning and reducing overfitting and dealing with noise in input.
- The paper demonstrates that using a combination of multiple drug features leads to more informative and effective predictions than using a single feature type.
- The CNN-DDI algorithm is works well for different similarity measures (Jaccard, cosine, and Gaussian), with Jaccard similarity being used in the experiments.

3. How well the proposed method work (in its own metrics)

The proposed CNN-DDI method performed well according to the metrics used in the study. The CNN-DDI algorithm outperforms other methods like Random Forest (RF), Gradient Boosting Decision Tree (GBDT), Logistic Regression (LR), and K-Nearest Neighbor (KNN) across several evaluation metrics. Specifically, CNN-DDI achieved the highest scores in accuracy (ACC), area under the precision-recall curve (AUPR), area under the ROC curve (AUC), F1-score, precision, and recall. The paper also compared CNN-DDI with prior DNN model (DDIMDL) and claimed that CNN-DDI still performed better even when using the same features.

4. What is the contribution to the research regime?

- The paper tries to advance DDI prediction by introducing a novel method that leverages the strengths of CNNs for feature learning and prediction.
- By providing a more accurate method for predicting DDIs, the paper contributes to the drug development process, potentially reducing the time and resources spent on identifying adverse drug interactions.
- The method could be used to improve patient safety by better predicting potential DDIs, thus preventing adverse effects that could arise from drug combinations.
- The CNN-DDI algorithm could be instrumental in the field of personalized medicine, where accurate DDI predictions are essential for tailoring drug regimens to individual patients.

The CNN-DDI paper demonstrates the importance of feature learning and the potential of deep learning in improving the prediction of DDIs.

Scope of Reproducibility:

The primary hypothesis of the paper is that the CNN-DDI model, which utilizes a feature selection framework and a novel CNN architecture, can accurately predict drug-drug interactions and outperform other the models mentioned in the paper (Random forest, Logistic Regression, K-nearest neighbor, Gradient boosted Decision Tree, & DDIMDL). This hypothesis is tested by implementing the CNN-DDI model according to the details mentioned in the paper and training on the the collected dataset with the same (inferred) hyperparameters settings. Afterwards, we compare the results with table 3 and 4 from the paper to our results.

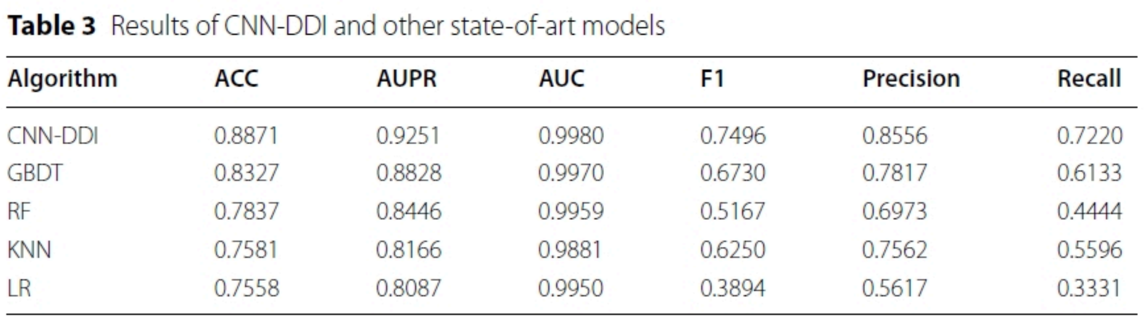


Table 3 Results of CNN-DDI and other state-of-art models

Algorithm	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8871	0.9251	0.9980	0.7496	0.8556	0.7220
GBDT	0.8327	0.8828	0.9970	0.6730	0.7817	0.6133
RF	0.7837	0.8446	0.9959	0.5167	0.6973	0.4444
KNN	0.7581	0.8166	0.9881	0.6250	0.7562	0.5596
LR	0.7558	0.8087	0.9950	0.3894	0.5617	0.3331

Table 4 Comparison of CNN-DDI with DDIMDL

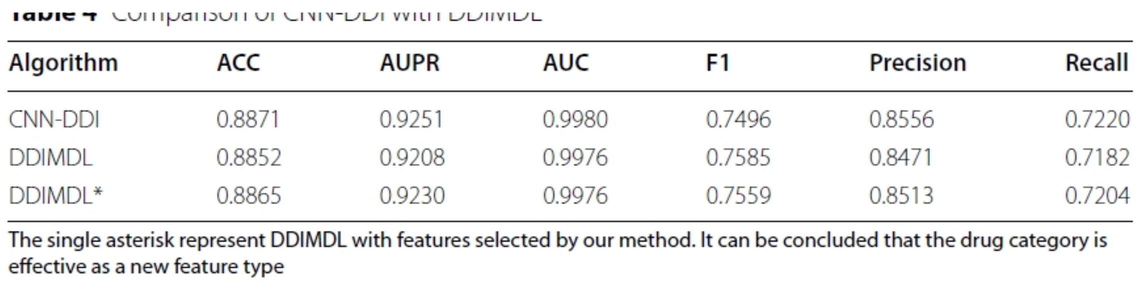


Table 4 Comparison of CNN-DDI with DDIMDL

Algorithm	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8871	0.9251	0.9980	0.7496	0.8556	0.7220
DDIMDL	0.8852	0.9208	0.9976	0.7585	0.8471	0.7182
DDIMDL*	0.8865	0.9230	0.9976	0.7559	0.8513	0.7204

The single asterisk represent DDIMDL with features selected by our method. It can be concluded that the drug category is effective as a new feature type

Methodology

Environment

Most of the code in this notebook is taken from the github repo for the paper "A multimodal deep learning framework for predicting drug-drug interaction events" [2]. This code was written for Python 3.7 and had the following dependencies.

- numpy (==1.18.1)
- Keras (==2.2.4)
- pandas (==1.0.1)
- scikit-learn (==0.21.2)
- tensorflow (==1.15)

The code for this notebook is updated to be run in google colab environment (Python 3.10) and is tested with the following package versions:

- numpy (==1.25.2)
- pandas (==2.0.3)
- scikit-learn (==1.2.2)
- tensorflow (==2.15.0)
- tqdm (==4.66.2)
- psutil (==5.9.5)
- gdown (==4.7.3)

In [1]:
```
!python --version
```

```
Python 3.10.12
```

If not running in Google colab environment, uncomment & run the following commands to install all dependencies.

In [2]:
```
# !pip install numpy==1.25.2
# !pip install pandas==2.0.3
# !pip install scikit-learn==1.2.2
# !pip install tensorflow==2.15.0
# !pip install tqdm==4.66.2
# !pip install psutil==5.9.5
# !pip install gdown==4.7.3
```

In [3]:
```
import sys
import os
import random
import csv
import sqlite3
import time
import numpy as np
import pandas as pd
import tensorflow as tf
from pandas import DataFrame
from tqdm import tqdm

# set seed
seed = 1
# random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
os.environ["PYTHONHASHSEED"] = str(seed)

print("TensorFlow version:", tf.__version__)
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
TensorFlow version: 2.15.0
Num GPUs Available:  1
```

In [4]:
```
def set_max_gpu_mem(size=10,unit=1024):
  limit = size * unit
  gpus = tf.config.experimental.list_physical_devices('GPU')
  if gpus:
      try:
          # Restrict TensorFlow to only allocate specified memory on the first GPU
          print(f"Setting GPU memory limit to {size}GB.")
          tf.config.experimental.set_virtual_device_configuration(
              gpus[0],
              [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=limit)])
          logical_gpus = tf.config.experimental.list_logical_devices('GPU')
          print(len(gpus), "Physical GPU,", len(logical_gpus), f"Logical GPU(s) with memory limit:{limit}")
      except RuntimeError as e:
          # Virtual devices must be set before GPUs have been initialized
          print(e)
```

In [5]:
```
# SET MEMORY LIMIT TO 10GB AVOID CRASHING DURING TRAINING
# ADJUST AS NEEEDED
set_max_gpu_mem(10)
```

```
Setting GPU memory limit to 10GB.
1 Physical GPU, 1 Logical GPU(s) with memory limit:10240
```

In [6]:
```
import psutil
PROCESS = psutil.Process(os.getpid())

def print_memory_usage(unit_size=10 ** 6):
    """Prints current memory usage stats.
    See: https://stackoverflow.com/a/15495136

    :return: None
    """
    total, available, percent, used, free, *_ = psutil.virtual_memory()
    total, available, used, free = total / unit_size, available / unit_size, used / unit_size, free / unit_size
    proc = PROCESS.memory_info()[1] / unit_size
    print('process = %s total = %s available = %s used = %s free = %s percent = %s'
          % (proc, total, available, used, free, percent))
```

Data

The dataset used by CNN-DDI is from the DDIMDL Github repository (https://github.com/YifanDengWHU/DDIMDL). The DDIMDL paper classifies DDIs' events into 65 types and includes 572 drugs with more than 70,000 associated events. The data was originally collected from the DrugBank website (https://go.drugbank.com/) using a web scraper and then processed and stored into a SQLite database (event.db). To utilize this dataset for CNN-DDI, we had to extract the 1622 category types for the drugs in the database from the Drugbank and store it. The database is stored in the public google drive folder and should be automatically downloaded from there. The updated scraper code to include category is our github repo and is purposely left out of this notebook to save time as its not needed to be run again.

Data Loading

In [7]:
```
# Function to check if running on Google Colab
def is_running_on_colab():
    return 'google.colab' in sys.modules
```

In [8]:
```
if is_running_on_colab() and 0: # enable/disable using my drive folder
    from google.colab import drive
    print('using my drive folder')
    # Mount Google Drive
    DRIVE_PATH = '/content/drive'
    drive.mount(DRIVE_PATH)
    BASE_PATH = f"{DRIVE_PATH}/My Drive/CNN_DDI/"
else:
    # use gdown to download from public drive folder
    import gdown
    print('downloading from public drive folder')
    url = "https://drive.google.com/drive/folders/1ln1ga9J7XzwAnAikKXS-ejXcPLe27jgI?usp=sharing"
    output_folder = 'CNN_DDI'
    gdown.download_folder(url, output=output_folder, quiet=False)
    BASE_PATH = f"./{output_folder}/"
```

```
downloading from public drive folder
```

```
Retrieving folder contents
Retrieving folder 1-IKo1Uf3DPJp6br-bKR1V47ZYKbOLEY4 .ipynb_checkpoints
Processing file 1XJY2IvPXVUr2mnNSjARSh8YUPzIsAtnk Ablation Plan.png
Processing file 1VhwNiJHXzmdARWhGrWx_JX88MaIBYka6 cnn_ddi_model.h5
Processing file 1oRINnkMG0ay3myhE2bkbS2-lo148DwYU cp.ckpt.index
Processing file 1Jd9gZlXapv8wHyhkJYMEngw6rZPC7yAr DrugList.txt
Processing file 1PVEf5gtFZdWDerqgThdXlhSBUajtrXHy event.db
Processing file 17OwHb_qH209fo-DkawQs8R7E3OtTmCbr Reproduction Results Comparison.xlsx
Processing file 1KVYOLkXTHgUw38q08FyOsKAOL3LElHzY smile+target+enzyme+category_all_CNN_DDI.csv
Processing file 1ctCNfuNeSk5j5rI9I_-F2BVWr808QEEE smile+target+enzyme+category_each_CNN_DDI.csv
Processing file 16rToBuMrW_eDXfOA2EHqpP8ojPWnbGY5 Table 2 R.png
Processing file 1cD-GkXTKiNEwVONa5kzUf5jpVly2oJhx Table 2.png
Processing file 11SNLsWjDe9RqzCc90UUiDOieNMprl4AZ Table 3 R.png
Processing file 1aUzLZji-e5m8w1fwoFG_2ZB5fsBeMVtk Table 3.png
Processing file 13zFOSsoLenVKDi7mRLMJA-7_VwZqc2Ul Table 4 R.png
Processing file 1vTVHcCusQn0WMRqyI3E4_s_Fz6yW2CaO Table 4.png
Processing file 1n3Ls-m1vIuZqEyUs-1m5wZzQKVsBAGs2 Table 5.PNG
Building directory structure completed
```

```
Retrieving folder contents completed
Building directory structure
Downloading...
From: https://drive.google.com/uc?id=1XJY2IvPXVUr2mnNSjARSh8YUPzIsAtnk
To: /content/CNN_DDI/Ablation Plan.png
100%|██████████| 80.9k/80.9k [00:00<00:00, 19.0MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=1VhwNiJHXzmdARWhGrWx_JX88MaIBYka6
From (redirected): https://drive.google.com/uc?id=1VhwNiJHXzmdARWhGrWx_JX88MaIBYka6&confirm=t&uuid=cda01291-c04f-49bd-9a3a-0d2
b90c7c1e4
To: /content/CNN_DDI/cnn_ddi_model.h5
100%|██████████| 472M/472M [00:05<00:00, 92.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=1oRINnkMG0ay3myhE2bkbS2-lo148DwYU
To: /content/CNN_DDI/cp.ckpt.index
100%|██████████| 3.35k/3.35k [00:00<00:00, 10.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Jd9gZlXapv8wHyhkJYMEngw6rZPC7yAr
To: /content/CNN_DDI/DrugList.txt
100%|██████████| 7.61k/7.61k [00:00<00:00, 14.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=1PVEf5gtFZdWDerqgThdXlhSBUajtrXHy
To: /content/CNN_DDI/event.db
100%|██████████| 30.6M/30.6M [00:00<00:00, 111MB/s]
Downloading...
From: https://drive.google.com/uc?id=17OwHb_qH209fo-DkawQs8R7E3OtTmCbr
To: /content/CNN_DDI/Reproduction Results Comparison.xlsx
100%|██████████| 180k/180k [00:00<00:00, 82.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=1KVYOLkXTHgUw38q08FyOsKAOL3LElHzY
To: /content/CNN_DDI/smile+target+enzyme+category_all_CNN_DDI.csv
100%|██████████| 220/220 [00:00<00:00, 732kB/s]
Downloading...
From: https://drive.google.com/uc?id=1ctCNfuNeSk5j5rI9I_-F2BVWr808QEEE
To: /content/CNN_DDI/smile+target+enzyme+category_each_CNN_DDI.csv
100%|██████████| 6.57k/6.57k [00:00<00:00, 16.8MB/s]
Downloading...
From: https://drive.google.com/uc?id=16rToBuMrW_eDXfOA2EHqpP8ojPWnbGY5
To: /content/CNN_DDI/Table 2 R.png
100%|██████████| 96.7k/96.7k [00:00<00:00, 76.4MB/s]
Downloading...
From: https://drive.google.com/uc?id=1cD-GkXTKiNEwVONa5kzUf5jpVly2oJhx
To: /content/CNN_DDI/Table 2.png
100%|██████████| 234k/234k [00:00<00:00, 90.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=11SNLsWjDe9RqzCc90UUiDOieNMprl4AZ
To: /content/CNN_DDI/Table 3 R.png
100%|██████████| 82.3k/82.3k [00:00<00:00, 80.5MB/s]
Downloading...
From: https://drive.google.com/uc?id=1aUzLZji-e5m8w1fwoFG_2ZB5fsBeMVtk
To: /content/CNN_DDI/Table 3.png
100%|██████████| 88.2k/88.2k [00:00<00:00, 81.7MB/s]
Downloading...
From: https://drive.google.com/uc?id=13zFOSsoLenVKDi7mRLMJA-7_VwZqc2Ul
To: /content/CNN_DDI/Table 4 R.png
100%|██████████| 99.5k/99.5k [00:00<00:00, 65.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=1vTVHcCusQn0WMRqyI3E4_s_Fz6yW2CaO
To: /content/CNN_DDI/Table 4.png
100%|██████████| 95.0k/95.0k [00:00<00:00, 28.0MB/s]
Downloading...
From: https://drive.google.com/uc?id=1n3Ls-m1vIuZqEyUs-1m5wZzQKVsBAGs2
To: /content/CNN_DDI/Table 5.PNG
```

```
100%|██████████| 53.6k/53.6k [00:00<00:00, 67.8MB/s]
Download completed
```

In [9]:
```python
# Define data and model path
EVENT_DATA_PATH = BASE_PATH + "event.db"
# DRUG_LIST_PATH =  BASE_PATH + "DrugList.txt"
# WEIGHT_PATH = "/content/drive/My Drive/CNN_DDI/checkpoints/cp.ckpt.index"
```

In [10]:
```python
VECTOR_SIZE = 572                         # num drugs (model input size)
EVENT_NUM = 65                            # num unique event types
DROP_RATE = 0.3                           # Default dropout rate
conn = sqlite3.connect(EVENT_DATA_PATH)
```

- **df_drug** contains 572 kinds of drugs and their features

In [11]:
```python
df_drug = pd.read_sql('select * from drug;', conn)
print("df_drug shape", df_drug.shape)
print("-" * 40)
print("Columns:", df_drug.columns.tolist())
print("-" * 40)
df_drug.info()
print("-" * 40)
print("Sample:")
for index, row in df_drug.head().iterrows():
    print(row.to_dict())
```

```
df_drug shape (572, 8)
----------------------------------------
Columns: ['index', 'id', 'target', 'enzyme', 'pathway', 'smile', 'name', 'category']
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 572 entries, 0 to 571
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   index     572 non-null    int64 
 1   id        572 non-null    object
 2   target    572 non-null    object
 3   enzyme    572 non-null    object
 4   pathway   572 non-null    object
 5   smile     572 non-null    object
 6   name      572 non-null    object
 7   category  572 non-null    object
dtypes: int64(1), object(7)
memory usage: 35.9+ KB
----------------------------------------
Sample:
{'index': 0, 'id': 'DB01296', 'target': 'P14780|Q00653|P01375|P01579|P33673', 'enzyme': 'P33261|P05181', 'pathway': 'hsa:4318|
hsa:4791|hsa:7124|hsa:3458', 'smile': '9|10|14|18|19|20|178|181|283|284|285|286|299|308|332|338|339|340|341|344|345|346|347|35
1|352|365|366|367|380|393|405|406|528|563|566|567|571|582|592|614|615|617|637|638|639|643|661|662|663|679|680|681|682|683|689|
690|691|701|703', 'name': 'Glucosamine', 'category': 'DBCAT000338|DBCAT002312|DBCAT002311|DBCAT000085|DBCAT000337|DBCAT00212
2'}
{'index': 1, 'id': 'DB09230', 'target': 'Q02641', 'enzyme': 'P08684', 'pathway': 'hsa:782', 'smile': '9|10|11|12|13|14|15|16|1
8|19|20|129|131|132|178|182|183|184|185|189|192|196|199|283|284|285|286|299|301|332|333|335|338|339|340|341|344|345|346|351|35
2|355|356|365|366|370|371|374|375|376|377|380|384|390|391|392|393|395|401|405|416|420|423|430|434|437|440|441|443|446|449|452|
454|455|464|470|490|502|514|516|520|524|535|540|545|546|549|552|553|556|558|560|564|566|570|573|578|579|582|584|592|594|595|59
9|600|603|607|608|613|614|615|618|619|628|633|634|637|640|643|654|656|659|660|664|665|666|668|671|673|677|678|679|680|683|684|
688|689|690|692|693|694|696|697|698|700|704|708|709|710|712|737|800', 'name': 'Azelnidipine', 'category': 'DBCAT000328|DBCAT00
3960|DBCAT000243|DBCAT000244|DBCAT000021|DBCAT003297|DBCAT001211|DBCAT001212|DBCAT002703|DBCAT000574|DBCAT003919|DBCAT002646|D
BCAT005101|DBCAT000389|DBCAT000388|DBCAT000227|DBCAT003676'}
{'index': 2, 'id': 'DB05812', 'target': 'P05093', 'enzyme': 'P08684|Q06520|P10635|P10632|P05177|P33261|P11712', 'pathway': 'hs
a:1586', 'smile': '9|10|11|12|14|18|143|147|178|179|182|183|184|185|186|192|199|283|284|285|286|308|332|333|334|335|339|341|34
4|345|346|351|352|355|356|358|365|366|370|371|372|373|374|376|384|387|390|396|403|406|416|418|430|434|435|441|442|445|446|447|
449|453|464|470|472|482|490|491|495|502|506|516|520|521|523|524|530|538|539|540|545|546|549|552|555|556|564|570|571|576|577|57
8|582|584|585|592|595|599|600|603|607|608|613|617|618|628|633|634|637|640|641|656|657|660|664|665|668|677|678|679|680|683|688|
689|696|697|698|699|708|709|710|711|712|776|777|797|818|839|860', 'name': 'Abiraterone', 'category': 'DBCAT000981|DBCAT000980|
DBCAT000024|DBCAT002086|DBCAT000402|DBCAT004503|DBCAT002610|DBCAT000403|DBCAT002640|DBCAT004482|DBCAT000868|DBCAT002644|DBCAT0
03042|DBCAT000489|DBCAT002636|DBCAT004528|DBCAT000911|DBCAT002625|DBCAT000934|DBCAT003919|DBCAT003232|DBCAT002648|DBCAT004049|
DBCAT002646|DBCAT003893|DBCAT004487|DBCAT000394|DBCAT005101|DBCAT003461|DBCAT002090|DBCAT000003|DBCAT004974|DBCAT000144|DBCAT0
02137|DBCAT000057|DBCAT003931|DBCAT002667|DBCAT000487|DBCAT000309'}
{'index': 3, 'id': 'DB01195', 'target': 'Q14524|P35499|Q12809', 'enzyme': 'P10635|P11712', 'pathway': 'hsa:6331|hsa:6329|hsa:3
757', 'smile': '9|10|11|12|14|15|18|19|23|24|25|178|180|181|182|185|283|284|285|286|287|299|332|333|338|340|341|344|345|346|35
1|352|355|356|363|365|366|370|371|381|382|384|390|392|393|405|416|420|430|434|439|441|443|446|451|464|470|476|490|493|498|516|
520|524|528|535|540|541|542|548|549|552|553|556|564|565|569|570|573|574|578|579|581|582|584|589|592|594|595|597|599|603|604|60
6|607|608|611|613|614|618|619|620|623|625|626|628|632|633|634|637|638|640|641|642|643|645|646|651|655|656|660|664|666|668|671|
672|677|678|679|680|681|682|683|684|686|688|689|692|698|699|704|708|709|710|719|735|756|782|798|819', 'name': 'Flecainide', 'c
ategory': 'DBCAT003297|DBCAT003865|DBCAT002518|DBCAT002210|DBCAT000010|DBCAT002609|DBCAT004029|DBCAT000489|DBCAT004528|DBCAT00
0911|DBCAT004505|DBCAT002623|DBCAT004031|DBCAT000394|DBCAT005101|DBCAT003950|DBCAT003951|DBCAT004525|DBCAT003957|DBCAT003956|D
BCAT000128|DBCAT002690|DBCAT003972|DBCAT005659|DBCAT002668|DBCAT004027|DBCAT000695|DBCAT003823|DBCAT000600|DBCAT000615'}
{'index': 4, 'id': 'DB00201', 'target': 'P30542|P29274|O07343|P21817|BE0004922|P78527|O00329|P42336|P42338|BE0004914|O13315'
```

```
{'index': 4, 'id': 'DB00201', 'target': 'P30542|P29274|Q07343|P21817|BE0004922|P78527|O00329|P42336|P42338|BE0004914|Q13315',
'enzyme': 'P20815|P05177|P24462|P08684|P05181|P10632|P11712|P04798|Q16678|P10635', 'pathway': 'hsa:134|hsa:135|hsa:5142|hsa:62
61|hsa:5591|hsa:5293|hsa:5290|hsa:5291|hsa:472', 'smile': '9|10|11|14|15|16|18|19|143|148|149|178|183|184|283|284|285|286|332|
340|351|352|355|357|358|359|365|373|374|375|376|377|378|379|381|384|386|387|388|389|390|391|396|397|403|416|418|420|431|437|43
8|439|441|442|443|447|449|450|451|453|464|472|482|484|485|487|491|493|494|495|499|504|506|519|521|523|530|535|536|538|540|545|
547|549|553|555|560|569|572|580|585|593|596|601|602|611|613|621|624|628|636|645|646|647|650|654|657|673|674', 'name': 'Caffein
e', 'category': 'DBCAT000443|DBCAT003635|DBCAT002662|DBCAT002675|DBCAT000044|DBCAT000437|DBCAT004212|DBCAT000402|DBCAT002611|D
BCAT004503|DBCAT002609|DBCAT002642|DBCAT002634|DBCAT002623|DBCAT002628|DBCAT003919|DBCAT002646|DBCAT000394|DBCAT005101|DBCAT00
2135|DBCAT002094|DBCAT002112|DBCAT000003|DBCAT004973|DBCAT002148|DBCAT000127|DBCAT000509|DBCAT002156|DBCAT002180|DBCAT000507|D
BCAT000506|DBCAT000253|DBCAT000504|DBCAT003636|DBCAT002181|DBCAT001579|DBCAT001034'}
```

- **df_event** contains 65 unique event types

In [12]:
```
df_event = pd.read_sql('select * from event_number;', conn)
print("df_event shape", df_event.shape)
print("-" * 40)
print("Columns:", df_event.columns.tolist())
print("-" * 40)
df_event.info()
print("-" * 40)
print(df_event.describe(include='all'))
print("-" * 40)
print("Sample:")
for index, row in df_event.head().iterrows():
    print(row.to_dict())
```

```
df_event shape (65, 2)
----------------------------------------
Columns: ['event', 'number']
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   event   65 non-null     object
 1   number  65 non-null     object
dtypes: object(2)
memory usage: 1.1+ KB
----------------------------------------
                                                    event number
count                                                  65     65
unique                                                 65     52
top     The metabolism of name can be decreased when c...     10
freq                                                    1      5
----------------------------------------
Sample:
{'event': 'The metabolism of name can be decreased when combined with name.', 'number': '19620'}
{'event': 'The risk or severity of adverse effects can be increased when name is combined with name.', 'number': '18992'}
{'event': 'The serum concentration of name can be increased when it is combined with name.', 'number': '11292'}
{'event': 'The serum concentration of name can be decreased when it is combined with name.', 'number': '4772'}
{'event': 'The therapeutic efficacy of name can be decreased when used in combination with name.', 'number': '2624'}
```

- **df_interaction** contains the 37,264 DDIs between the 572 kinds of drugs

In [13]:
```
df_interaction = pd.read_sql('select * from event;', conn)
print("df_interaction shape", df_interaction.shape)
print("-" * 40)
print("Columns:", df_interaction.columns.tolist())
print("-" * 40)
df_interaction.info()
print("-" * 40)
print(df_interaction.describe(include='all').head(3))
print("-" * 40)
print("Sample:")
for index, row in df_interaction.head().iterrows():
    print(row.to_dict())
```

```
df_interaction shape (37264, 6)
----------------------------------------
Columns: ['index', 'id1', 'name1', 'id2', 'name2', 'interaction']
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37264 entries, 0 to 37263
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   index        37264 non-null  int64 
 1   id1          37264 non-null  object
 2   name1        37264 non-null  object
 3   id2          37264 non-null  object
 3   id2          37264 non-null  object
 4   name2        37264 non-null  object
 5   interaction  37264 non-null  object
dtypes: int64(1), object(5)
memory usage: 1.7+ MB
----------------------------------------
          index      id1       name1      id2       name2  \
count   37264.0    37264       37264    37264       37264
unique      NaN      560         560      558         558
top         NaN  DB01118  Amiodarone  DB01149  Nefazodone

                                              interaction
count                                               37264
unique                                              37264
top     The risk or severity of adverse effects can be...
----------------------------------------
Sample:
{'index': 0, 'id1': 'DB12001', 'name1': 'Abemaciclib', 'id2': 'DB01118', 'name2': 'Amiodarone', 'interaction': 'The risk or se
verity of adverse effects can be increased when Abemaciclib is combined with Amiodarone'}
{'index': 1, 'id1': 'DB12001', 'name1': 'Abemaciclib', 'id2': 'DB11901', 'name2': 'Apalutamide', 'interaction': 'The serum con
centration of Abemaciclib can be decreased when it is combined with Apalutamide'}
{'index': 2, 'id1': 'DB12001', 'name1': 'Abemaciclib', 'id2': 'DB00673', 'name2': 'Aprepitant', 'interaction': 'The serum conc
entration of Abemaciclib can be increased when it is combined with Aprepitant'}
{'index': 3, 'id1': 'DB12001', 'name1': 'Abemaciclib', 'id2': 'DB00289', 'name2': 'Atomoxetine', 'interaction': 'The metabolis
m of Abemaciclib can be decreased when combined with Atomoxetine'}
{'index': 4, 'id1': 'DB12001', 'name1': 'Abemaciclib', 'id2': 'DB00188', 'name2': 'Bortezomib', 'interaction': 'The metabolism
of Abemaciclib can be decreased when combined with Bortezomib'}
```

- **df_extraction** contains the interactions after NLP process was run on the list of drugs to parse extract key information about the interactions. Note, this was already done as data preprocessing step in original DDIMDL paper and was not performed CNN-DDI paper. Hence, we simply load from the event.db database.
  - Mechanism: The biological effect of the drugs, such as effects on metabolism, serum concentration, or therapeutic efficacy.
  - Action: The change that occurs as a result of the interaction, typically an increase or decrease in the effect of one or both drugs.
  - Drug A: The drug whose efficacy is affected by the interaction.
  - Drug B: The other drug involved in the interaction.

In [14]:

```
df_extraction = pd.read_sql('select * from extraction;', conn)
print("df_extraction shape", df_extraction.shape)
print("-" * 40)
print("Columns:", df_extraction.columns.tolist())
print("-" * 40)
df_extraction.info()
print("-" * 40)
print(df_extraction.describe(include='all').head(3))
print("-" * 40)
print("Sample:")
for index, row in df_extraction.head().iterrows():
    print(row.to_dict())

mechanism = df_extraction['mechanism']
action = df_extraction['action']
drugA = df_extraction['drugA']
drugB = df_extraction['drugB']
```

```
df_extraction shape (37264, 5)
----------------------------------------
Columns: ['index', 'mechanism', 'action', 'drugA', 'drugB']
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37264 entries, 0 to 37263
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   index      37264 non-null  int64
 1   mechanism  37264 non-null  object
 2   action     37264 non-null  object
 3   drugA      37264 non-null  object
 4   drugB      37264 non-null  object
dtypes: int64(1), object(4)
memory usage: 1.4+ MB
----------------------------------------
          index      mechanism    action        drugA       drugB
count   37264.0          37264     37264        37264       37264
unique      NaN             57         4          561         558
top         NaN  The metabolism  increase  Atomoxetine  Amiodarone
----------------------------------------
Sample:
{'index': 0, 'mechanism': 'The risk or severity of adverse effects', 'action': 'increase', 'drugA': 'Abemaciclib', 'drugB': 'A
miodarone'}
```

```
{'index': 1, 'mechanism': 'The serum concentration', 'action': 'decrease', 'drugA': 'Abemaciclib', 'drugB': 'Apalutamide'}
{'index': 2, 'mechanism': 'The serum concentration', 'action': 'increase', 'drugA': 'Abemaciclib', 'drugB': 'Aprepitant'}
{'index': 3, 'mechanism': 'The metabolism', 'action': 'decrease', 'drugA': 'Abemaciclib', 'drugB': 'Atomoxetine'}
{'index': 4, 'mechanism': 'The metabolism', 'action': 'decrease', 'drugA': 'Abemaciclib', 'drugB': 'Bortezomib'}
```

Data Prepocessing

For each drug, a binary feature matrix is constructed based on the presence (1) or absence (0) of specific features. This is done by first extracting all unique features across all drugs for a given feature type and then populating a matrix where each row corresponds to a drug and each column to a feature. If a drug has a particular feature, the corresponding cell in the matrix is marked as 1, otherwise as 0. The Jaccard Similarity between the feature vectors of drugs is computed to measure the similarity between drugs based on their features. PCA is applied to reduce the dimensionality of the similiariy matrix. The reduced-dimensionality feature vectors are then used as input for the CNN model to predict DDIs. For each drug-drug pair, the feature vectors of the two drugs are combined to form a single input vector to the model.

In [15]:
```
def create_feature_set_name(feature_list):
    #TODO set_name not needed ,remove
    """
    Create a feature set name from a list of features.
    source code adapted from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L338

    Args:
        feature_list (list): A list of feature names.

    Returns:
        tuple: A tuple containing two elements:
            - feature_name (str): A string of concatenated feature names separated by "+".
            - set_name (str): A string of feature names separated by "+" with a trailing "+".
    """
    feature_name = "+".join(feature_list)
    set_name = ""
    for feature in feature_list:
        set_name = feature + "+"
    set_name = set_name[:-1]
    return feature_name, set_name
```

In [16]:
```
# Define the Jaccard Similarity function
def Jaccard(matrix):
    """
    Calculate the Jaccard similarity between rows of a given matrix.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L89

    Args:
        matrix (array-like): A 2D array (or matrix) where each row represents a set in binary form (1s and 0s),
                             with 1 indicating the presence of an element in the set, and 0 indicating absence.

    Returns:
        numpy.matrix: A matrix of Jaccard similarity scores between each pair of rows in the input matrix.
    """
    matrix = np.mat(matrix)
    numerator = matrix * matrix.T
    denominator = np.ones(np.shape(matrix)) * matrix.T + matrix * np.ones(np.shape(matrix.T)) - matrix * matrix.T
    return numerator / denominator
```

In [17]:
```
from sklearn.decomposition import PCA

def feature_vector(feature_name, df, vector_size):
    """
    Generates a feature vector for each drug based on the specified feature using Jaccard Similarity and PCA reduction.

    This function first constructs a feature matrix for drugs based on the presence or absence of specific features
    (e.g., targets, enzymes). It then computes the Jaccard Similarity matrix for these drugs and finally reduces the
    dimensionality of this matrix to the specified vector size using PCA.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L86

    Args:
        feature_name (str): The name of the feature column in the DataFrame `df` to be used for generating feature vectors.
        df (DataFrame): A pandas DataFrame containing drug data. Each row corresponds to a drug, and the specified
                        feature column contains feature identifiers separated by '|'.
        vector_size (int): The target number of dimensions for the feature vectors after PCA reduction.
        numpy.ndarray: A 2D array where each row represents the reduced-dimensionality feature vector for a drug.
    """
    all_feature = []
    drug_list = np.array(df[feature_name]).tolist()
    # Extract unique features from the feature column for all drugs
    for features in drug_list:
        for each_feature in features.split('|'):
            if each_feature not in all_feature:
                all_feature.append(each_feature)

    # Initialize a feature matrix with zeros
    feature_matrix = np.zeros((len(drug_list), len(all_feature)), dtype=float)
    # Construct a DataFrame for easier manipulation
    df_feature = DataFrame(feature_matrix, columns=all_feature)

    # Populate the feature matrix with 1s where a drug has a particular feature
    for i, features in enumerate(drug_list):
        for each_feature in features.split('|'):
            df_feature.at[i, each_feature] = 1

    # Compute the Jaccard Similarity matrix
    sim_matrix = Jaccard(np.array(df_feature))
    sim_matrix = np.asarray(sim_matrix)

    # Apply PCA to reduce the dimensionality of the similarity matrix
    pca = PCA(n_components=vector_size)
    pca.fit(sim_matrix)
    reduced_sim_matrix = pca.transform(sim_matrix)

    return reduced_sim_matrix
```

In [18]:

```
def prepare(df_drug, feature_list, vector_size, mechanism, action, drugA, drugB):
    """
    Prepares feature vectors and labels for drug interaction events.

    This function processes a list of drug interaction features to generate corresponding
    feature vectors and labels. It assigns a unique numerical label to each unique
    mechanism-action pair and constructs feature vectors for each drug based on the provided
    features.

    source code adapted from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L50

    Args:
        df_drug (DataFrame): DataFrame containing drug data, including names.
        feature_list (list): List of features to be included in the feature vector.
        vector_size (int): The size of the feature vector for each feature.
        mechanism (Series): Series of mechanisms involved in drug interactions.
        action (Series): Series of actions resulting from drug interactions.
        drugA (Series): Series of primary drugs involved in interactions.
        drugB (Series): Series of secondary drugs involved in interactions.

    Returns:
        tuple: A tuple containing:
            - new_feature (numpy.ndarray): Array of feature vectors for drug interactions.
            - new_label (numpy.ndarray): Array of labels for each drug interaction event.
            - event_num (int): The total number of unique interaction events.
    """
    d_label = {}
    d_feature = {}
    d_event = []

    # Concatenate mechanism and action to form unique interaction events
    for i in range(len(mechanism)):
        d_event.append(mechanism[i] + " " + action[i])

    # Count occurrences of each event and assign a unique label
    count = {}
    for event in d_event:
        count[event] = count.get(event, 0) + 1
    sorted_events = sorted(count.items(), key=lambda x: x[1], reverse=True)
    for i, (event, _) in enumerate(sorted_events):
        d_label[event] = i

    # Initialize a zero vector for feature aggregation
    vector = np.zeros((len(df_drug['name']), 0), dtype=float)

    # Aggregate feature vectors for each feature in the list
    for feature in feature_list:
        vector = np.hstack((vector, feature_vector(feature, df_drug, vector_size)))

    # Map drug names to their feature vectors
    for i, name in enumerate(df_drug['name']):
        d_feature[name] = vector[i]

    # Construct feature vectors and labels for each interaction event
    new_feature = []
    new_label = []
    for i in range(len(d_event)):
        combined_feature = np.hstack((d_feature[drugA[i]], d_feature[drugB[i]]))
        new_feature.append(combined_feature)
        new_label.append(d_label[d_event[i]])

    new_feature = np.array(new_feature)
    new_label = np.array(new_label)
    event_num = len(sorted_events)

    return (new_feature, new_label, event_num)
```

In [19]:

```
def construct_feature_matrix(feature_list, df_drug, vector_size, mechanism, action, drugA, drugB):
    """
    Processes each feature in the given feature list by preparing and accumulating their corresponding new features.

    source code adapted from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L358

    Args:
        feature_list (list): A list of features to be processed.
        df_drug (DataFrame): The DataFrame containing drug data.
        vector_size (int): The size of the vector to be used in preparation.
        mechanism, action, drugA, drugB: Additional parameters required by the `prepare` function.

    Returns:
        tuple: A tuple containing three elements:
            - all_matrix (list): A list of feature matrices, where each matrix corresponds to a feature in the feature_list.
            - new_label (numpy.ndarray): The label matrix corresponding to the feature matrices.
            - event_num (int): The total number of unique events or classes in the label matrix.
    """
    all_matrix = []
    for feature in feature_list:
        #print(feature)
        new_feature, new_label, event_num = prepare(df_drug, [feature], vector_size, mechanism, action, drugA, drugB)
        all_matrix.append(new_feature)
    return all_matrix, new_label, event_num
```

In [20]:

```
feature_list = ["smile", "target", "enzyme", "category"]
feature_name, set_name = create_feature_set_name(feature_list)
print(feature_name)
```

```
smile+target+enzyme+category
```

In [21]:

```
feature_matrix, new_label, event_num = construct_feature_matrix(feature_list, df_drug, VECTOR_SIZE, mechanism, action, drugA,
print(f"Shape of feature_matrix: ({len(feature_matrix)}, {len(feature_matrix[0])}, {len(feature_matrix[0][0])})")
print(f"Number of label: {new_label.shape[0]}")
print(f"Number of events: {event_num}")
```

```
Shape of feature_matrix: (4, 37264, 1144)
Number of label: 37264
Number of events: 65
```

Model

In [22]:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Input, Activation, BatchNormalization
from tensorflow.keras.layers import Conv1D, Flatten, Add
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
```

DDIMDL Model

The model consists of an input layer, two dense layers with ReLU activation and dropout for regularization, and an output layer with softmax activation for multi-class classification. It uses the Adam optimizer and Categorial Cross Entropy as the loss function.

In [23]:
```
def DNN(vector_size=VECTOR_SIZE, event_num=EVENT_NUM, drop_rate=DROP_RATE):
    """
    A deep neural network (DNN) model for predicting drug-drug interactions.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L35

    Args:
        vector_size (int): The size of the input feature vector.
        event_num (int): The number of unique interaction events (classes) to predict.
        drop_rate (float): The dropout rate for regularization.

    Returns:
        model: A compiled Keras model ready for training.
    """
    # Define the input layer
    train_input = Input(shape=(vector_size * 2,), name='Inputlayer')
     # First dense layer with 512 units and ReLU activation
    train_in = Dense(512, activation='relu')(train_input)
    train_in = BatchNormalization()(train_in)
    train_in = Dropout(drop_rate)(train_in)
    # Second dense layer with 256 units and ReLU activation
    train_in = Dense(256, activation='relu')(train_in)
    train_in = BatchNormalization()(train_in)
    train_in = Dropout(drop_rate)(train_in)
    # Output dense layer with 'event_num' units for classification
    train_in = Dense(event_num)(train_in)
    # Softmax activation to convert logits to probabilities for multi-class classification
    out = Activation('softmax')(train_in)
    # Create the model
    model = Model(inputs=train_input, outputs=out)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

In [24]:
```
DDIMDL_model = DNN()
DDIMDL_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 Inputlayer (InputLayer)     [(None, 1144)]            0         
                                                                 
 dense (Dense)               (None, 512)               586240    
                                                                 
 batch_normalization (Batch  (None, 512)               2048      
 Normalization)                                                  
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               131328    
                                                                 
 batch_normalization_1 (Bat  (None, 256)               1024      
 chNormalization)                                                
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 65)                16705     
                                                                 
 activation (Activation)     (None, 65)                0         
                                                                 
=================================================================
Total params: 737345 (2.81 MB)
Trainable params: 735809 (2.81 MB)
Non-trainable params: 1536 (6.00 KB)
_________________________________________________________________
```

CNN-DDI Model

The model architecture is based on the CNN-DDI method described in the paper It includes an input layer, multiple convolutional layers with LeakyReLU activation, a residual block, and fully connected layers with a softmax layer for multi-class classification. It uses the Adam optimizer and Categorial Cross Entropy as the loss function.

Table 5 The convolution layers of CNN-DDI

Layer name	number of filters	Kernel size	Output shape
Conv1	64	3 × 1	(64, 572, 4)

Conv2	128	3 × 1	(128, 572, 4)
Conv3_1	128	3 × 1	(128, 572, 4)
Conv3_2	128	3 × 1	(128, 572, 4)
Conv4	256	3 × 1	(256, 572, 4)

In [25]:
```
def CNN_DDI(vector_size=VECTOR_SIZE, event_num=EVENT_NUM):
    """
    Convolutional Neural Network (CNN) model for predicting drug-drug interactions (DDIs).

    Implementation based on "CNN-DDI: a learning-based method for predicting drug-drug interactions using convolution neural
    https://doi.org/10.1186/s12859-022-04612-2

    Args:
        vector_size (int): The size of the input feature vector for each drug.
        event_num (int): The number of unique DDI event types to predict.

    Returns:
        model: A compiled Keras model ready for training.
    """
    # Define the input layer
    inputs = Input(shape=(vector_size, 2), name='InputLayer')

    # Convolutional layers as specified in the paper
    conv1 = Conv1D(filters=64, kernel_size=3, strides=1, padding='same')(inputs)
    conv1 = LeakyReLU(alpha=0.2)(conv1)

    conv2 = Conv1D(filters=128, kernel_size=3, strides=1, padding='same')(conv1)
    conv2 = LeakyReLU(alpha=0.2)(conv2)

    # Residual block starts
    conv3_1 = Conv1D(filters=128, kernel_size=3, strides=1, padding='same')(conv2)
    conv3_1 = LeakyReLU(alpha=0.2)(conv3_1)

    conv3_2 = Conv1D(filters=128, kernel_size=3, strides=1, padding='same')(conv3_1)
    conv3_2 = LeakyReLU(alpha=0.2)(conv3_2)

    # Add the input of the residual block (conv2) to its output (conv3_2)
    res_out = Add()([conv2, conv3_2])
    # Residual block ends

    conv4 = Conv1D(filters=256, kernel_size=3, strides=1, padding='same')(res_out)
    conv4 = LeakyReLU(alpha=0.2)(conv4)

    # Flatten the output of the last convolutional layer
    flatten = Flatten()(conv4)

    # Fully connected layers
    fc1 = Dense(267, activation='relu')(flatten)

    fc2 = Dense(event_num)(fc1)  # Assuming 'num_classes' is the number of DDI event types
    out = Activation('softmax')(fc2)

    # Create the model
    model = Model(inputs=inputs, outputs=out)

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

In [26]:
```
CNN_DDI_model = CNN_DDI()
CNN_DDI_model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 InputLayer (InputLayer)        [(None, 572, 2)]     0           []

 conv1d (Conv1D)                (None, 572, 64)      448         ['InputLayer[0][0]']

 leaky_re_lu (LeakyReLU)        (None, 572, 64)      0           ['conv1d[0][0]']
 conv1d_1 (Conv1D)           (None, 572, 128)             24704     ['leaky_re_lu[0][0]']

 leaky_re_lu_1 (LeakyReLU)   (None, 572, 128)             0         ['conv1d_1[0][0]']

 conv1d_2 (Conv1D)           (None, 572, 128)             49280     ['leaky_re_lu_1[0][0]']

 leaky_re_lu_2 (LeakyReLU)   (None, 572, 128)             0         ['conv1d_2[0][0]']

 conv1d_3 (Conv1D)           (None, 572, 128)             49280     ['leaky_re_lu_2[0][0]']

 leaky_re_lu_3 (LeakyReLU)   (None, 572, 128)             0         ['conv1d_3[0][0]']

 add (Add)                   (None, 572, 128)             0         ['leaky_re_lu_1[0][0]',
                                                                     'leaky_re_lu_3[0][0]']

 conv1d_4 (Conv1D)           (None, 572, 256)             98560     ['add[0][0]']

 leaky_re_lu_4 (LeakyReLU)   (None, 572, 256)             0         ['conv1d_4[0][0]']

 flatten (Flatten)           (None, 146432)               0         ['leaky_re_lu_4[0][0]']

 dense_3 (Dense)             (None, 267)                  3909761   ['flatten[0][0]']
                                                          1

 dense_4 (Dense)             (None, 65)                   17420     ['dense_3[0][0]']

 activation_1 (Activation)   (None, 65)                   0         ['dense_4[0][0]']

==================================================================================================
Total params: 39337303 (150.06 MB)
Trainable params: 39337303 (150.06 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Other ML Models

The other models compared against in the paper include random forest (RF), gradient boosting decision tree (GBDT), logistic regression (LR) and K-nearest neighbor (KNN).

In [27]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier

def logistic_regression_pred(X_train, Y_train, X_test):
    #Logistic Regression model
    # original source code from: https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L182
    model = LogisticRegression()
    model.fit(X_train, Y_train)
    pred = model.predict_proba(X_test)
    return pred

def random_forest_pred(X_train, Y_train, X_test):
    #Random Forest Classifier with 100 trees
    # original source code from: https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L172
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, Y_train)
    pred = model.predict_proba(X_test)
    return pred

def gbdt_pred(X_train, Y_train, X_test):
    #Gradient Boosting Decision Tree (GBDT) model
    # original source code from: https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L174
    model = GradientBoostingClassifier()
    model.fit(X_train, Y_train)
    pred = model.predict_proba(X_test)
    return pred

def svm_pred(X_train, Y_train, X_test):
    #Support Vector Machine (SVM) model with probability estimates
    # original source code from: https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L176
    model = SVC(probability=True)
    model.fit(X_train, Y_train)
    pred = model.predict_proba(X_test)
    return pred

def knn_pred(X_train, Y_train, X_test):
    #K-Nearest Neighbors (KNN) classifier with 4 neighbors
    # original source code from: https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L180
    model = KNeighborsClassifier(n_neighbors=4)
    model.fit(X_train, Y_train)
    pred = model.predict_proba(X_test)
    return pred
```

Metrics

CNN-DDI is multi-class classification problem. For evaluation, accuracy (ACC), area under the precision–recall-curve (AUPR), area under the ROC curve (AUC), F1 score and Precision are used as evaluation metrics. Please note, AUPR & AUC are micro-averaged while other metrics are macro-averaged. This is done since the data classes are imbalanced.

In [28]:

```
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import precision_recall_curve

def multiclass_precision_recall_curve(y_true, y_score):
    """
    Calculate the precision-recall curve for the first class in a multiclass classification problem.

    This function reshapes the true labels and predicted scores if necessary, and then computes
    the precision-recall curve for the first class. It is designed to work with one-vs-rest
    multiclass classification models where each class is treated independently.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L265

    Args:
      y_true: array-like of shape (n_samples,) or (n_samples, n_classes)
              True binary labels or binary label indicators for each class.
      y_score: array-like of shape (n_samples,) or (n_samples, n_classes)
               Target scores, can either be probability estimates of the positive class,
               confidence values, or non-thresholded measure of decisions.

    Returns:
      precision: array of shape (n_thresholds + 1,)
                 Precision values such that element i is the precision of predictions with
                 score >= thresholds[i] and the last element is 1.
      recall: array of shape (n_thresholds + 1,)
              Recall values such that element i is the recall of predictions with
              score >= thresholds[i] and the last element is 0.
      pr_thresholds: array of shape (n_thresholds,)
                     Decreasing thresholds on the decision function used to compute
                     precision and recall.
    """
    # Ensure the true labels and scores are 1D arrays, reshaping if necessary
    y_true = y_true.ravel()
    y_score = y_score.ravel()
    # Reshape y_true and y_score to 2D arrays if they are 1D
    if y_true.ndim == 1:
        y_true = y_true.reshape((-1, 1))
    if y_score.ndim == 1:
        y_score = y_score.reshape((-1, 1))
    # Extract the true labels and scores for the first class
    y_true_c = y_true.take([0], axis=1).ravel()
    y_score_c = y_score.take([0], axis=1).ravel()
    # Compute precision, recall, and thresholds for the first class
    precision, recall, pr_thresholds = precision_recall_curve(y_true_c, y_score_c)
    return (precision, recall, pr_thresholds)


def roc_aupr_score(y_true, y_score, average="macro"):
    """
    Calculate the Area Under the Precision-Recall Curve (AUPR) for binary or multiclass classification.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L278

    Args:
      y_true: array-like of shape (n_samples,) or (n_samples, n_classes)
              True binary labels or binary label indicators for multiclass classification.
      y_score: array-like of shape (n_samples,) or (n_samples, n_classes)
               Target scores, can either be probability estimates of the positive class,
               confidence values, or non-thresholded measure of decisions.
      average: string, ['micro', 'macro', 'binary'] (default='macro')
      average: string, [ 'micro', 'macro', 'binary' ] (default 'macro')
               If 'binary', calculate AUPR for binary classification problems.
               If 'micro', calculate metrics globally by considering each element of the label
               indicator matrix as a label.
               If 'macro', calculate metrics for each label, and find their unweighted mean.

    Returns:
      AUPR score: float
                  Area Under the Precision-Recall Curve (AUPR) score.
    """
    # Function to calculate AUPR for binary classification
    def _binary_roc_aupr_score(y_true, y_score):
        precision, recall, pr_thresholds = precision_recall_curve(y_true, y_score)
        return auc(recall, precision)

    # Function to handle averaging of AUPR scores for multiclass classification
    def _average_binary_score(binary_metric, y_true, y_score, average):  # y_true= y_one_hot
        if average == "binary":
            return binary_metric(y_true, y_score)
        # Handle micro averaging
        if average == "micro":
            y_true = y_true.ravel()
            y_score = y_score.ravel()
        # Ensure y_true and y_score are 2D arrays
        if y_true.ndim == 1:
            y_true = y_true.reshape((-1, 1))
        if y_score.ndim == 1:
            y_score = y_score.reshape((-1, 1))
        n_classes = y_score.shape[1]
        score = np.zeros((n_classes,))
        # Calculate AUPR for each class and average
        for c in range(n_classes):
            y_true_c = y_true.take([c], axis=1).ravel()
            y_score_c = y_score.take([c], axis=1).ravel()
            score[c] = binary_metric(y_true_c, y_score_c)
        return np.average(score)

    return _average_binary_score(_binary_roc_aupr_score, y_true, y_score, average)
```

In [29]:

```
from sklearn.preprocessing import label_binarize

def evaluate(pred_type, pred_score, y_test, event_num):
    """
    Evaluate the performance of predictions for multi-class classification.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L203

    Args:
        pred_type (array-like): Predicted labels for each sample.
        pred_score (array-like): Predicted scores or probabilities for each class for each sample.
        y_test (array-like): True labels for each sample.
        event_num (int): Number of distinct events or classes.

    Returns:
        list: A list containing two numpy arrays. The first array contains overall evaluation metrics for the model,
              and the second array contains evaluation metrics for each class.
    """
    # Define the number of evaluation metrics for overall performance
    all_eval_type = 11
    # Initialize an array to store overall evaluation metrics
    result_all = np.zeros((all_eval_type, 1), dtype=float)
    # Define the number of evaluation metrics for each class
    each_eval_type = 6
    # Initialize an array to store evaluation metrics for each class
    result_eve = np.zeros((event_num, each_eval_type), dtype=float)
    # Convert true labels to one-hot encoding
    y_one_hot = label_binarize(y_test, classes=np.arange(event_num))
    # Convert predicted labels to one-hot encoding
    pred_one_hot = label_binarize(pred_type, classes=np.arange(event_num))

    # Calculate precision and recall for multi-class classification
    precision, recall, th = multiclass_precision_recall_curve(y_one_hot, pred_score)

    # Calculate overall evaluation metrics
    result_all[0] = accuracy_score(y_test, pred_type)
    result_all[1] = roc_aupr_score(y_one_hot, pred_score, average='micro')
    result_all[2] = roc_aupr_score(y_one_hot, pred_score, average='macro')
    result_all[3] = roc_auc_score(y_one_hot, pred_score, average='micro')
    result_all[4] = roc_auc_score(y_one_hot, pred_score, average='macro')
    result_all[5] = f1_score(y_test, pred_type, average='micro')
    result_all[6] = f1_score(y_test, pred_type, average='macro')
    result_all[7] = precision_score(y_test, pred_type, average='micro')
    result_all[8] = precision_score(y_test, pred_type, average='macro')
    result_all[9] = recall_score(y_test, pred_type, average='micro')
    result_all[10] = recall_score(y_test, pred_type, average='macro')

    # Calculate evaluation metrics for each event type
    for i in range(event_num):
        result_eve[i, 0] = accuracy_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel())
        result_eve[i, 1] = roc_aupr_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel(),
                                          average=None)
        result_eve[i, 2] = roc_auc_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel(),
                                         average=None)
        result_eve[i, 3] = f1_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel(),
                                    average='binary')
        result_eve[i, 4] = precision_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel(),
                                           average='binary')
        result_eve[i, 5] = recall_score(y_one_hot.take([i], axis=1).ravel(), pred_one_hot.take([i], axis=1).ravel(),
                                        average='binary')
    # Return the overall and per-class evaluation metrics

    return [result_all, result_eve]
```

Training & Evaluation

For the CNN-DDI reproduction, 5-fold cross validation was used to evaluate the model according to the paper. The data was randomly split into 5 subsets, with 4 used for training and 1 for testing in each fold. The final metrics reported are the average across the 5 folds.

In [30]:
```
from sklearn.model_selection import KFold

def get_index(label_matrix, event_num, seed, CV):
    """
    Generate indices for K-fold cross-validation for each class in the label matrix.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L117

    Args:
        label_matrix (array-like): A 1D array containing the class labels for each sample.
        event_num (int): The number of unique events or classes.
        seed (int): Random seed for reproducibility of the shuffle in KFold.
        CV (int): The number of folds for the K-fold cross-validation.

    Returns:
        numpy.ndarray: An array of indices indicating the fold number for each sample.
    """
    # Initialize an array to store the fold indices for all samples
    index_all_class = np.zeros(len(label_matrix))
    # generate fold indices for each class
    for j in range(event_num):
        # Find the indices of samples belonging to the current class
        index = np.where(label_matrix == j)
        # Initialize KFold with the specified number of splits, shuffling, and random seed
        kf = KFold(n_splits=CV, shuffle=True, random_state=seed)
        # Initialize a counter for the fold number
        k_num = 0
        # Get train and test indices for each fold
        for train_index, test_index in kf.split(range(len(index[0]))):
            # Assign the fold number to the corresponding samples in the overall index array
            index_all_class[index[0][test_index]] = k_num
            # Increment the fold number
            k_num += 1
    # Return the array of fold indices
    return index_all_class
```

In [31]:
```
def cross_validation(feature_matrix, label_matrix, clf_type, event_num, seed, CV, num_epochs, batch_size, patience=10):
    """
    Perform K-fold cross-validation to evaluate the performance of specified classifiers on a DDI prediction task.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L130

    Args:
        feature_matrix (array-like or list of array-like): Feature matrix or list of feature matrices for training and testir
        label_matrix (array-like): Label matrix corresponding to the true class labels.
        clf_type (str): Type of classifier to be evaluated. Supported types include 'DDIMDL', 'CNN_DDI', 'RF', 'GBDT', 'SVM',
        event_num (int): Number of unique events or classes.
        seed (int): Random seed for reproducibility of the shuffle in KFold.
        CV (int): Number of folds for the K-fold cross-validation.
        num_epochs (int): Number of training epochs for neural network models (DDIMDL and CNN_DDI).
        batch_size (int): Batch size used during training of neural network models (DDIMDL and CNN_DDI).
        patience (int, optional): Number of epochs with no improvement after which training will be stopped for early stoppir

    Returns:
        tuple: A tuple containing two numpy arrays. The first array contains overall evaluation metrics for the model,
               and the second array contains evaluation metrics for each class.
    """
    # Initialize arrays to store evaluation results

    all_eval_type = 11
    result_all = np.zeros((all_eval_type, 1), dtype=float)
    each_eval_type = 6
    result_eve = np.zeros((event_num, each_eval_type), dtype=float)
    y_true = np.array([])
    y_pred = np.array([])
    y_score = np.zeros((0, event_num), dtype=float)
    # Generate indices for K-fold cross-validation
    index_all_class = get_index(label_matrix, event_num, seed, CV)
    matrix = []
    if type(feature_matrix) != list:
        matrix.append(feature_matrix)
        feature_matrix = matrix
    for k in range(CV):
        # Split data into training and testing sets based on fold index
        train_index = np.where(index_all_class != k)
        test_index = np.where(index_all_class == k)
        pred = np.zeros((len(test_index[0]), event_num), dtype=float)
        # Train and predict with each feature matrix (in case of multiple feature matrices)
        for i in range(len(feature_matrix)):
            x_train = feature_matrix[i][train_index]
            x_test = feature_matrix[i][test_index]
            y_train = label_matrix[train_index]
            # one-hot encoding training labels
            y_train_one_hot = np.array(y_train)
            y_train_one_hot = (np.arange(event_num) == y_train[:, None]).astype(dtype='float32')
            y_test = label_matrix[test_index]
            # one-hot encoding of testing labels
            y_test_one_hot = np.array(y_test)
            y_test_one_hot = (np.arange(event_num) == y_test[:, None]).astype(dtype='float32')
            if clf_type == 'DDIMDL':
                dnn = DNN()
                # print_memory_usage()
                early_stopping = EarlyStopping(monitor='val_loss', patience=patience, verbose=0, mode='auto')
                dnn.fit(x_train, y_train_one_hot, batch_size=batch_size, epochs=batch_size, validation_data=(x_test, y_test_c
                        callbacks=[early_stopping])
                pred += dnn.predict(x_test)
            elif clf_type == 'CNN_DDI':
                x_train_reshaped = x_train.reshape(-1, VECTOR_SIZE, 2)
                x_test_reshaped = x_test.reshape(-1, VECTOR_SIZE, 2)
                cnn_ddi = CNN_DDI()
                # print_memory_usage()
                early_stopping = EarlyStopping(monitor='val_loss', patience=patience, verbose=0, mode='auto')
                cnn_ddi.fit(x_train_reshaped, y_train_one_hot, batch_size=batch_size, epochs=num_epochs, validation_data=(x_t
                            callbacks=[early_stopping])
                pred += cnn_ddi.predict(x_test_reshaped)
            elif clf_type == 'RF':
                pred += random_forest_pred(x_train, y_train, x_test)
            elif clf_type == 'GBDT':
                pred += gbdt_pred(x_train, y_train, x_test)
            elif clf_type == 'SVM':
                pred += svm_pred(x_train, y_train, x_test)
            elif clf_type == 'FM':
                pred += gbdt_pred(x_train, y_train, x_test)
            elif clf_type == 'KNN':
                pred += knn_pred(x_train, y_train, x_test)
            elif clf_type == 'LR':
                pred += logistic_regression_pred(x_train, y_train, x_test)
        # Aggregate predictions from all feature matrices and determine predicted class
        pred_score = pred / len(feature_matrix)
        pred_type = np.argmax(pred_score, axis=1)
        # Accumulate true labels, predicted labels, and predicted scores
        y_true = np.hstack((y_true, y_test))
        y_pred = np.hstack((y_pred, pred_type))
        y_score = np.row_stack((y_score, pred_score))
    # Evaluate the performance of the classifier
    result_all, result_eve = evaluate(y_pred, y_score, y_true, event_num)
    return result_all, result_eve
```

In [32]:
```
def save_result(feature_name, result_type, clf_type, result, base_path=BASE_PATH):
    """
    Save the evaluation results of a classifier into a CSV file.

    original source code from:
    https://github.com/YifanDengWHU/DDIMDL/blob/master/DDIMDL.py#L321

    Args:
        feature_name (str): Name of the feature set used for the classifier.
        result_type (str): Type of result being saved (e.g., 'accuracy', 'precision').
        clf_type (str): Type of classifier (e.g., 'CNN-DDI', 'RF').
        result (list): A list of evaluation results to be saved.
        base_path (str, optional): Base path for saving the result file. Defaults to BASE_PATH.

    Returns:
        int: 0 on successful execution.
    """
    # Construct the file path by combining base path, feature name, result type, and classifier type
    file_path = base_path + feature_name + '_' + result_type + '_' + clf_type + '.csv'
    with open(file_path, "w", newline='') as csvfile:
        writer = csv.writer(csvfile)
        for i in result:
            writer.writerow(i)
    return 0
```

In [33]:
```
def run_cross_validation(clf_list, featureName, featureMatrix, labelMatrix, event_num=EVENT_NUM, seed=0, num_folds=5, num_epo
    """
    run the cross-validation for the list of classifiers and save the results.

    Args:
        clf_list (list): list of classifier model names
        featureName (str): A descriptive name for the feature set used, which will be included in the result filenames.
        featureMatrix (array-like): The matrix of features used for training and testing the classifiers.
        labelMatrix (array-like): The matrix of labels corresponding to the featureMatrix.
        event_num (int): The number of unique event types to predict. Defaults to EVENT_NUM.
        seed (int): The random seed for reproducibility of cross-validation splits. Defaults to 0.
        num_folds (int): The number of folds for K-fold cross-validation. Defaults to 5.
        num_epochs (int): The number of epochs for training each classifier. Defaults to 100.
        batch_size (int): The batch size for training each classifier. Defaults to 128.

    Returns:
        dict: Two dictionaries containing the overall and per-event evaluation results for each classifier.
    """
    result_all = {}
    result_eve = {}
    all_matrix = featureMatrix
    new_label = labelMatrix
    start = time.perf_counter()
    for clf in clf_list:
      clf_start = time.perf_counter()
      print(f"running cross validation for {clf}")
      # Perform cross-validation using the specified classifier
      all_result, each_result = cross_validation(all_matrix, new_label, clf, event_num, seed, num_folds, num_epochs, batch_si
      # Save the cross-validation results to CSV files
      save_result(featureName, 'all', clf, all_result)
      save_result(featureName, 'each', clf, each_result)
      result_all[clf] = all_result
      result_eve[clf] = each_result
      print(all_result)
      print(f"time used for {clf}:", time.perf_counter() - clf_start)
    print("Total time used:", time.perf_counter() - start)
```

Computational Requirements

The CNN-DDI model has approximately 39 million parameters. To run the full cross-validation training and evaluation, it requires modern, high performance GPU such as Google Colab T4 with at least 10 GBs of memory. It takes on average 60-70 minutes to run. For each fold it runs 100 epochs x number of feature matrices (max is 4).

Demonstration

For the purposes of running this notebook in under 8 minutes, the number of epochs has been set to 1 (vs. 100) and number of folds to 2 (vs. 5).

In [34]:
```
classifiers = ["CNN_DDI"]
num_folds = 2
num_epochs = 1
batch_size = 128
CV_seed = 0

run_cross_validation(clf_list=classifiers,featureName=feature_name,featureMatrix=feature_matrix,labelMatrix=new_label,
                     event_num=event_num, seed=CV_seed, num_folds=num_folds, num_epochs=num_epochs, batch_size=batch_size)
```

```
running cross validation for CNN_DDI
146/146 [==============================] - 23s 91ms/step - loss: 1.6078 - accuracy: 0.5426 - val_loss: 0.9443 - val_accuracy:
0.7127
583/583 [==============================] - 4s 6ms/step
146/146 [==============================] - 15s 87ms/step - loss: 1.4443 - accuracy: 0.5897 - val_loss: 0.8755 - val_accuracy:
0.7214
583/583 [==============================] - 3s 6ms/step
146/146 [==============================] - 15s 88ms/step - loss: 1.7953 - accuracy: 0.4901 - val_loss: 1.3737 - val_accuracy:
0.5818
583/583 [==============================] - 4s 6ms/step
146/146 [==============================] - 15s 89ms/step - loss: 1.2169 - accuracy: 0.6581 - val_loss: 0.6843 - val_accuracy:
0.7849
583/583 [==============================] - 4s 6ms/step
146/146 [==============================] - 18s 108ms/step - loss: 1.6497 - accuracy: 0.5236 - val_loss: 0.9424 - val_accuracy:
0.7180
582/582 [==============================] - 6s 10ms/step
146/146 [==============================] - 18s 106ms/step - loss: 1.4265 - accuracy: 0.5912 - val_loss: 0.8505 - val_accuracy:
0.7293
582/582 [==============================] - 4s 6ms/step
146/146 [==============================] - 16s 91ms/step - loss: 1.8596 - accuracy: 0.4757 - val_loss: 1.4138 - val_accuracy:
0.5759
582/582 [==============================] - 3s 6ms/step
146/146 [==============================] - 17s 107ms/step - loss: 1.3300 - accuracy: 0.6284 - val_loss: 0.7094 - val_accuracy:
0.7814
582/582 [==============================] - 3s 6ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defi
ned and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
[[0.77436668]
 [0.8333017 ]
 [0.59029952]
 [0.99392502]
 [0.96260746]
 [0.77436668]
 [0.39520954]
 [0.77436668]
 [0.57176779]
 [0.77436668]
 [0.34944318]]
time used for CNN_DDI: 215.075280704
Total time used: 215.075329419
```

Results

Overview

The two fundamental hypothesis of the paper is CNN-DDI outperforms other models for predicting DDIs and that category is effective predictive feature for DDIs. In the paper Table 2, Table 3 and Table 4 demonstrate the performance about CNN-DDI model for the set of evaluation metrics (ACC, AURP, AUC, F1, Precision, and Recall). We generate the same three tables and name them Table 2 Reproduction, Table 3 Reproduction, and Table 4 Reproduction in our experiment. We then try to compare them with the paper and verify author's arguments accordingly.

Table 2 Reproduction

The goal of Table 2 is to test the effectiveness of adding certain features, especially for the drug category. Features T, P, E, and C (target, pathway, enzyme and drug category) are put into model training according to different combinations. Table 2 demonstrates that the performance is improved for CNN-DDI with four feature types. Adding the "category" feature does provide noticeable improvement.

Due to time constraints, we reproduced last 5 combinations and report the results in Table 2 Reproduction. In general, most results in Table 2 Reproduction have about 0~5% difference to the values of Table 2. But the results in the two tables follow similar patterns, demonstrating the successful reproduction for those metrics. The results in Table 2 Reproduction verify that adding "category" feature into feature list improves the model performance.

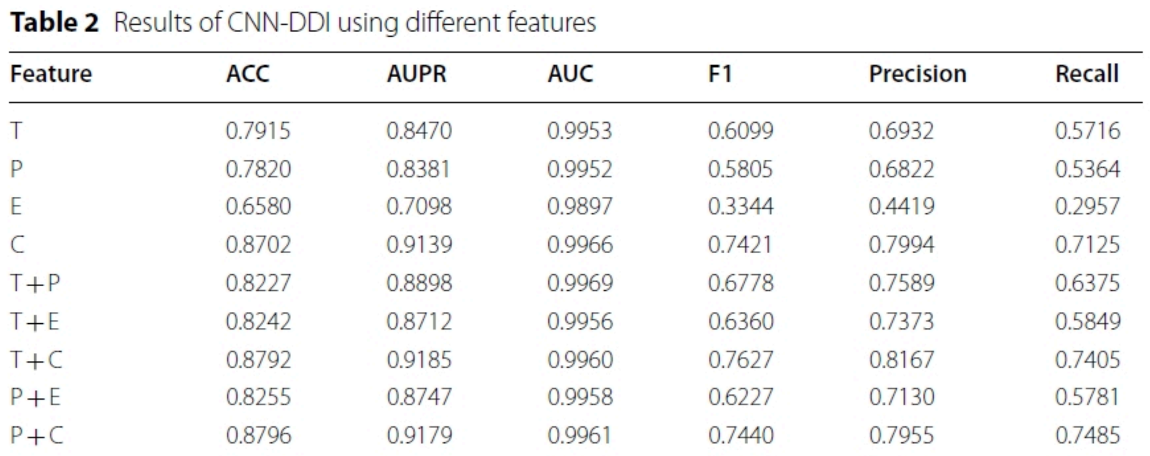
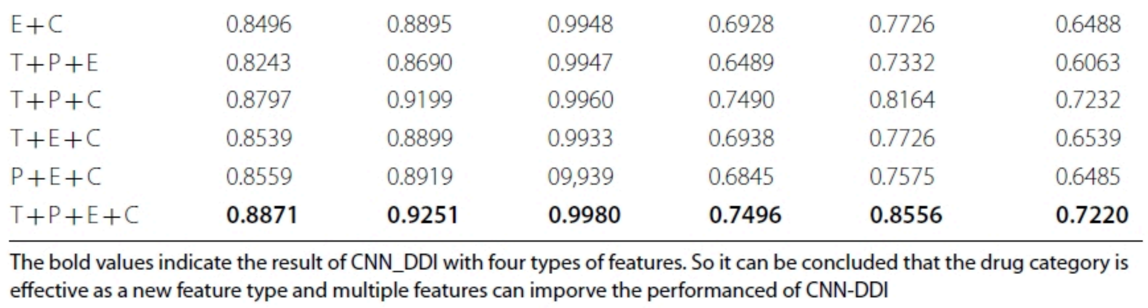


Table 2 Results of CNN-DDI using different features

Feature	ACC	AUPR	AUC	F1	Precision	Recall
T	0.7915	0.8470	0.9953	0.6099	0.6932	0.5716
P	0.7820	0.8381	0.9952	0.5805	0.6822	0.5364
E	0.6580	0.7098	0.9897	0.3344	0.4419	0.2957
C	0.8702	0.9139	0.9966	0.7421	0.7994	0.7125
T+P	0.8227	0.8898	0.9969	0.6778	0.7589	0.6375
T+E	0.8242	0.8712	0.9956	0.6360	0.7373	0.5849
T+C	0.8792	0.9185	0.9960	0.7627	0.8167	0.7405
P+E	0.8255	0.8747	0.9958	0.6227	0.7130	0.5781
P+C	0.8796	0.9179	0.9961	0.7440	0.7955	0.7485



E+C	0.8496	0.8895	0.9948	0.6928	0.7726	0.6488
T+P+E	0.8243	0.8690	0.9947	0.6489	0.7332	0.6063
T+P+C	0.8797	0.9199	0.9960	0.7490	0.8164	0.7232
T+E+C	0.8539	0.8899	0.9933	0.6938	0.7726	0.6539
P+E+C	0.8559	0.8919	09,939	0.6845	0.7575	0.6485
T+P+E+C	**0.8871**	**0.9251**	**0.9980**	**0.7496**	**0.8556**	**0.7220**

The bold values indicate the result of CNN_DDI with four types of features. So it can be concluded that the drug category is effective as a new feature type and multiple features can imporve the performanced of CNN-DDI

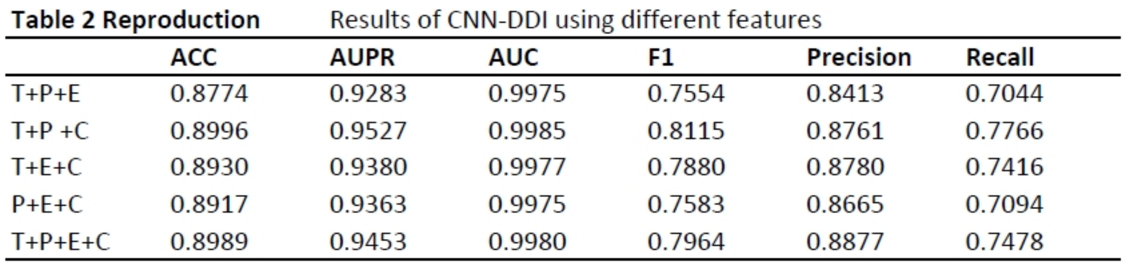


Table 2 Reproduction Results of CNN-DDI using different features

	ACC	AUPR	AUC	F1	Precision	Recall
T+P+E	0.8774	0.9283	0.9975	0.7554	0.8413	0.7044
T+P +C	0.8996	0.9527	0.9985	0.8115	0.8761	0.7766
T+E+C	0.8930	0.9380	0.9977	0.7880	0.8780	0.7416
P+E+C	0.8917	0.9363	0.9975	0.7583	0.8665	0.7094
T+P+E+C	0.8989	0.9453	0.9980	0.7964	0.8877	0.7478

Table 4 Reproduction:

Table 4 compares the outputs of CNN-DDI and DDIMDL with and without "category" feature included. Both models use the same source of data. Based on Table 4, the paper concludes that CNN-DDI outperforms DDIMDL.

To verify this argument, the reproduced results are documented in Table 4 Reproduction. Again, most data are within 5% difference and follow similar patterns. However we cannot fully support claim that the CNN-DDI model outperforms the DDIMDL model when using the same 4 features. We do not observe that to be the case as shown in last row of table 4 reproduction, the DDIMDL performance in all metrics is slighty better than CNN-DDI result.

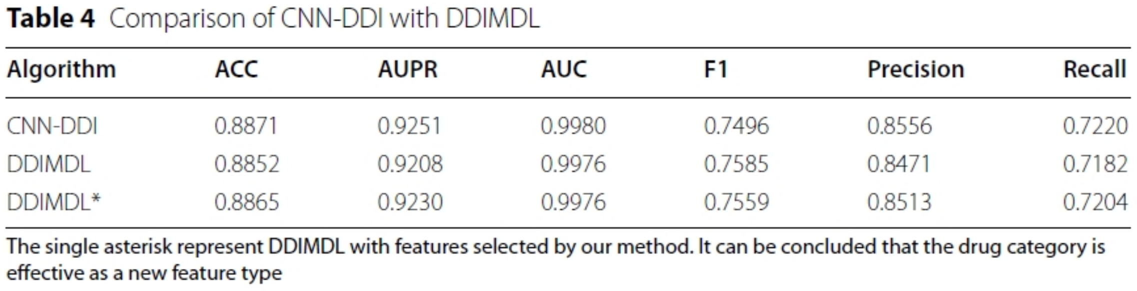


Table 4 Comparison of CNN-DDI with DDIMDL

Algorithm	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8871	0.9251	0.9980	0.7496	0.8556	0.7220
DDIMDL	0.8852	0.9208	0.9976	0.7585	0.8471	0.7182
DDIMDL*	0.8865	0.9230	0.9976	0.7559	0.8513	0.7204

The single asterisk represent DDIMDL with features selected by our method. It can be concluded that the drug category is effective as a new feature type

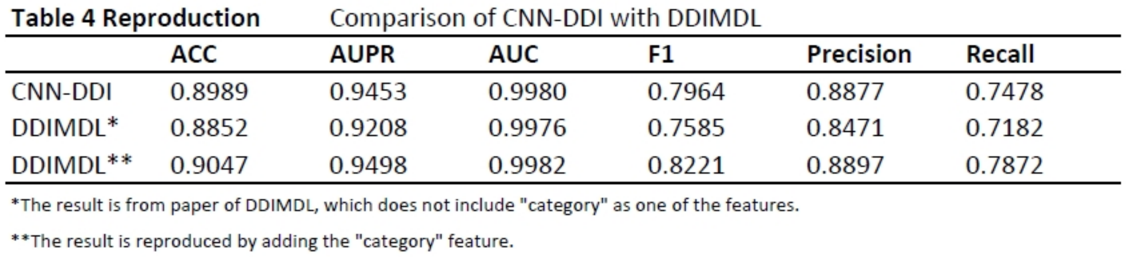


Table 4 Reproduction Comparison of CNN-DDI with DDIMDL

	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8989	0.9453	0.9980	0.7964	0.8877	0.7478
DDIMDL*	0.8852	0.9208	0.9976	0.7585	0.8471	0.7182
DDIMDL**	0.9047	0.9498	0.9982	0.8221	0.8897	0.7872

*The result is from paper of DDIMDL, which does not include "category" as one of the features.

**The result is reproduced by adding the "category" feature.

Model comparison

Table 3 and Table 3 Reproduction:

In the paper, Table 3 compares the performance of CNN-DDI with model GBDT, RF, KNN, and LR. The data shows CNN-DDI has stronger performance over those other ML models.

We reproduced all the results as Table 3 Reproduction except GBDT due to running issues. The results match the papers in the way that CNN-

DDI model performs better than the rest of the models in all metrics.

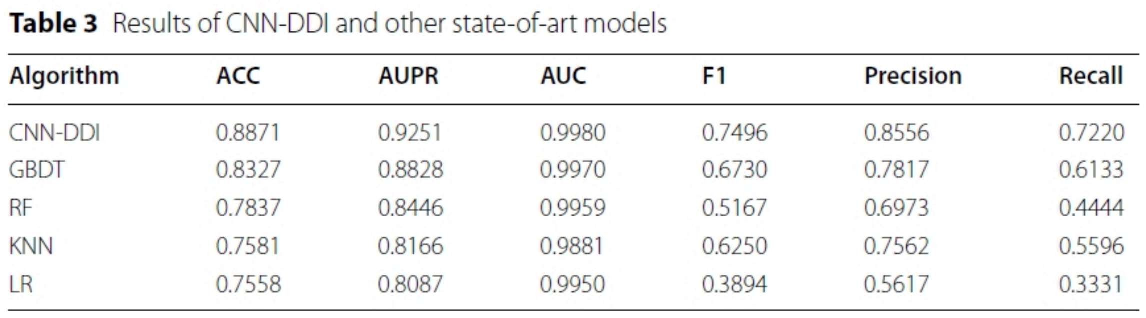


Table 3 Results of CNN-DDI and other state-of-art models

Algorithm	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8871	0.9251	0.9980	0.7496	0.8556	0.7220
GBDT	0.8327	0.8828	0.9970	0.6730	0.7817	0.6133
RF	0.7837	0.8446	0.9959	0.5167	0.6973	0.4444
KNN	0.7581	0.8166	0.9881	0.6250	0.7562	0.5596
LR	0.7558	0.8087	0.9950	0.3894	0.5617	0.3331

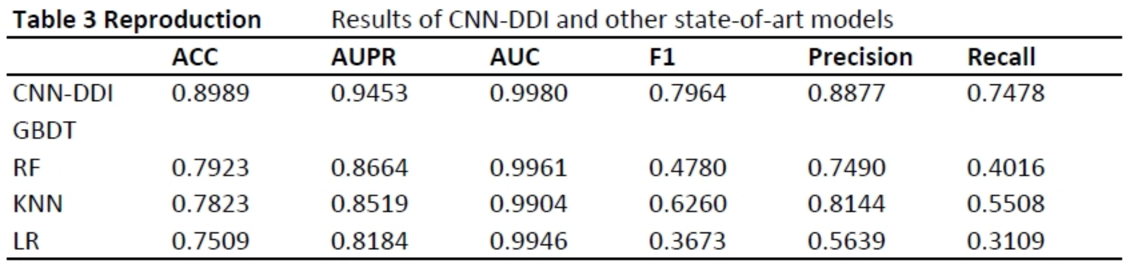


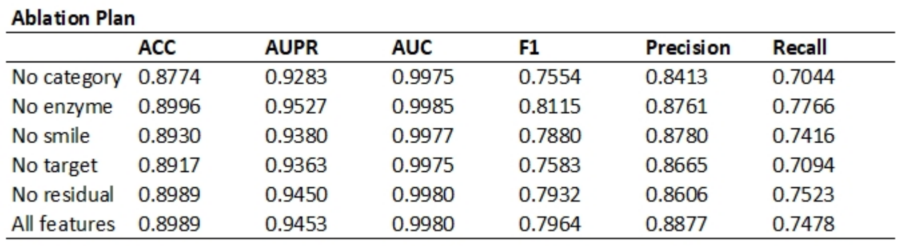
Table 3 Reproduction Results of CNN-DDI and other state-of-art models

	ACC	AUPR	AUC	F1	Precision	Recall
CNN-DDI	0.8989	0.9453	0.9980	0.7964	0.8877	0.7478
GBDT						
RF	0.7923	0.8664	0.9961	0.4780	0.7490	0.4016
KNN	0.7823	0.8519	0.9904	0.6260	0.8144	0.5508
LR	0.7509	0.8184	0.9946	0.3673	0.5639	0.3109

Ablation Plan:

For ablation, we will evaluate the model's performance by individually removing drug categories, targets, pathways, and enzyme features. For model ablation, we plan to assess the impact of removing the residual block on the model performance and try different loss functions.

Currently, we have tested the impact of removing features and residual block. The results indicate that the residual block has limited impact on the model's performance. Moreover, training on all four features does have the highest performance, adding category feature is valid. We will test on different loss function in the next phases for Ablation.



Ablation Plan

	ACC	AUPR	AUC	F1	Precision	Recall
No category	0.8774	0.9283	0.9975	0.7554	0.8413	0.7044
No enzyme	0.8996	0.9527	0.9985	0.8115	0.8761	0.7766
No smile	0.8930	0.9380	0.9977	0.7880	0.8780	0.7416
No target	0.8917	0.9363	0.9975	0.7583	0.8665	0.7094
No residual	0.8989	0.9450	0.9980	0.7932	0.8606	0.7523
All features	0.8989	0.9453	0.9980	0.7964	0.8877	0.7478

Discussion

Reproducibility Assessment

This paper is reproducible with certain efforts. The pattern of reproduced results matches well with the results in the paper. Most numbers are within reasonable value differences if not being the same, indicating the success of reproduction.

Negative Results

Even though we assert that this paper is reproducible, it is worth mentioning that the we cannot produce the evaluation results without running the entire cross-validation process. The data provided both in the reproduction tables and papers' tables is generated from the results of the training & evaluation process. If we try to save the model at the end of the cross-validation method and then load model and evaluate with the whole dataset, the performance drops significantly for some metrics.

Ease in Reproduction

The easy parts in reproduction come from those facts:

1. The structure of this CNN model is relatively simple and clear given the information from the paper.

2. Most of the data was available and data gathering methods are indicated.
3. This model is able to be run in the local machine with RXT 2080ti and complete the cross-validation in approximately 1 hour.

Difficulties in Reproduction

1. No access to the source code. This paper is about CNN-DDI while the GitHub repo provided is for an earlier paper for the DDIMDL model. While the available code from DDIMDL did provide the base for our team to work on, it is not guaranteed to be the same as the codebase from the authors of CNN-DDI paper. Even though the overall structure of this CNN model is described in the paper, we were still missing some critical information, such as the number of epochs, whether dropout was used, and the settngs of other hyperparameters that may impact the model performance. The lack of access to the full source code and these crucial details can be among the possible causes of any differences between reproduction data and paper data.
2. The provided dataset is incomplete. Adding drug category to the feature list can improve the model performance is the key statement of CNN-DDI paper. However, the category data was not available in the database of the DDIMDL repo. Thus, we gather the missing data and merge it with the existing dataset. However this data may have changed since the authors of the CNN-DDI paper accessed it.
3. Computing resources are limited. It may sound contradictory since we just mentioned it is easy for this project to be able to run on the local machine. Initially, running on local machine fails due to speed and out-of-memory issue. Our team spent some effort switching between different versions of PyTorch and TensorFlow with various settings to make the running successful eventually. Now it takes 1 hour to train the model.

Suggestions for Enhancing Reproducibility

For authors of this paper, it would be appreciated if they can release the source code or respond to the technical questions in email about the settings of this CNN-DDI model. Only having source code of other paper can be challenging. For future reproducers, we recommend going through the paper carefully, paying attention to the CNN model structure and understanding the authors' main goal so you are clear what to do. Also suggest using the DDIMDL's resource wisely since you can get hints about overall structure of your coding for CNN-DDI. There are many reusable parts including data processing, result evaluation, etc.

Next Phase

1. Adjusting model evaluation approaches. As stated above, currently when saving and evaluating the model with the whole dataset, prediction outcome gets lower values compared with the ones from the training process. Current assumptions are either we are not saving the model correctly and evaluated with the appropriate data input, or the trained model is not generalized well. Our team will investigate this problem.
2. Add more data visualizations. Since most of the input data is grid-like text data we found it difficult to have fancy visualizations. We will try to add some for the similiarity matrix.
3. Completing the reproduction. There are some results missing compared to the paper, including GBDT model comparision and doing all the different feature combinations for table 2. Our team will make the results section complete for final notebook submission.
4. Finishing the ablation plan. Our team has tested out different feature combinations and residual blocks for model performance. There are more items to be tested in the list such as different loss functions.

Public Github Repo

All source codes and data are available in our projects repo linked below

https://github.com/abaldeo/CS598_DLH_Project/tree/CNN_DDI

Important Please make sure your on CNN-DDI branch

References

1. Zhang, C., Lu, Y. & Zang, T. CNN-DDI: a learning-based method for predicting drug–drug interactions using convolution neural networks. BMC Bioinformatics 23 (Suppl 1), 88 (2022). https://doi.org/10.1186/s12859-022-04612-2

2. Deng, Y., Xu, X., Qiu, Y., Xia, J., Zhang, W., & Liu, S. (2020). A multimodal deep learning framework for predicting drug-drug interaction events. Bioinformatics (Oxford, England), 36(15), 4316–4322. https://doi.org/10.1093/bioinformatics/btaa501