Project Title: HiCu-ICD (Project ID 17)

- Project Team ID: 26
- Team Members: Ratul Saha (ratuls2 / ratuls2@illinois.edu)
- Github Repo Link: https://github.com/ratulsaha778/CS598-HiCu-Team26/

Introduction

The project, titled **"HiCu-ICD", is based on the MLHC 2022 paper "HiCu: Leveraging Hierarchy for Curriculum Learning in Automated ICD Coding"**. HiCu, or Hierarchical Curriculum Learning, improves ICD coding accuracy by leveraging the hierarchy of ICD codes, which groups diagnosis codes based on various organ systems in the human body.

Proceedings of Machine Learning Research 182:1–25, 2022 HiCu: Leveraging hierarchy for curriculum learning in automated ICD coding. W Ren, R Zeng, T Wu, T Zhu, RG Krishnan Machine Learning for Healthcare Conference, 198-223 Google Scholar Link: here

Weiming Ren wren@cs.toronto.edu, Ruijing Zeng jackzeng@cs.toronto.edu, Tongzi Wu tongziwu@cs.toronto.edu, Tianshu Zhu tianshu@cs.toronto.edu, Rahul G. Krishnan rahulgk@cs.toronto.edu (Department of Computer Science, University of Toronto & the Vector Institute, Toronto, Ontario, Canada)

Original Code repository: https://github.com/wren93/HiCu-ICD.

Improving clinician throughput is an important technological opportunity for supporting improved healthcare services. When clinicians write notes, a smart process would be beneficial which can document correct diagnosis codes against the human notes. Mapping the long and detailed clinical notes and discharge summaries buried under patient profiles (Electronic Health Records / EHR) to specific ICD (International Classification of Diseases) coding - is a time-consuming, error-prone, and challenging task due to many possible codes and the complex relationships between them. The original paper aims to address the problem of automated coding of medical diagnoses and procedures using the International Classification of Diseases (ICD) system. The paper proposes a novel hierarchical curriculum learning approach that leverages the hierarchical structure of the ICD codes to improve the accuracy and efficiency of automated ICD coding.

The paper uses a hierarchical structure of ICD codes to train the model in a curriculum learning framework. The approach involves learning simpler codes before more complex codes, designed based on the hierarchical structure of the ICD codes. It takes advantage of the hierarchical structure of the ICD codes to improve the model's ability to learn complex relationships between the codes, which is difficult to achieve using traditional flat learning approaches.

Curriculum learning is the design of curricula i.e., in the sequential design of tasks that gradually increase in difficulty. HiCu is an innovation over this process that can predict ICD codes from the natural language descriptions of the patients. It leverages the hierarchy of ICD codes which is grouped based on various organ systems of the human body. The HiCu algorithm uses the graph structure in the space of outputs to design curricula for multi-label classification.

The algorithm is based on:

- The Tree structure: The decision boundaries for different ICD codes are not independent. The ICD codes are organized in a tree structure which defines a notion of similarity between codes. This means that dissimilar labels will have different ancestors in the tree and vice versa. As we go deeper into each sub-tree, the specificity of the codes increases. This means that HiCu can provide wider and non-overlapping decision boundaries for multi-label classification as the diseases and organs are grouped under defined boundaries.
- HiCu explicitly incorporates techniques to handle label imbalance. This is essential to ensure parity of performance of predictive models on both rare and frequent labels. HiCu can predict rare and frequent labels with equal accuracy.

HiCu is an improvement over curriculum learning which is used in medical code prediction using graph structure for solving multi-label classification problem.

Scope of Reproducibility

ICD codes follow a logical grouping by following a hierarchy of disease, symptoms and body parts. The grouping helps form an ordered tree structure. The HiCu algorithm is inspired from this graph structure which is used to design curricula for multi-label classification. The hierarchical curricula learning claims to provide wider and non-overlapping decision boundary for multi-label classification as the disease and organs are grouped under defined boundaries. ICD codes are organized in a tree structure which establishes similarity, in other words dissimilar labels will have different ancestors in the tree.

This is an important distinction from the curricula learning by using NLP, where the learning algorithm does not assume any relationship.

As addressed in the original paper, following claims would be validated:

- When there is no hierarchy or fewer than 5 ICD Codes hierarchy, the model should perform poorly than original paper. This validates the

importance of the hierarchical structure in improving the multi-label classification performance of the HiCu algorithm.
- Reducing the ICD Code Hierarchy should affect the model training time. This is because the hierarchical curriculum learning approach requires the model to learn from more examples in the early stages of training, which can be computationally expensive. Reducing the hierarchy may cost to have different classification performance.

As part of the draft, the model was trained on MultiResCNN over 'train_full.csv' data with embed file 'processed_full_100.embed', Asymmetric loss function, and 'HierarchicalHyperbolic' decoder.

The final project report will focus on ablation studies and highlights in original project claims.

Methodology

This project aims at improving the ICD classification process. It leverages a hierarchical structure for curriculum learning to improve the accuracy and efficiency of automated ICD coding. It uses the MIMIC-III dataset for model training and evaluation.

The primary objective of this project is to design curricula for multi-label classification models that predict ICD diagnosis and procedure codes from natural language descriptions of patients. The project uses the MIMIC-III dataset for model training and evaluation. The data preprocessing code from MultiResCNN is used to set up the dataset. The project proposes Hierarchical Curriculum Learning (HiCu), an algorithm that uses graph structure in the space of outputs to design curricula for multi-label classification.

Data

The dataset MIMIC-III v1.4 (mimic 3) was downloaded from https://physionet.org/content/mimiciii/1.4/ which was first published 2nd Sept 2016 with the intent of enhancing data quality and providing a large amount of additional data for Metavision patients.

Resource citations: Johnson, A., Pollard, T., & Mark, R. (2016). MIMIC-III Clinical Database (version 1.4). PhysioNet. https://doi.org/10.13026/C2XW26.

Original publication: Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., & Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. Scientific Data, 3, 160035.

Citation for PhysioNet: Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

The dataset is downloaded from the protected website after the necessary training was obtained and the data usage agreement was signed.

This dataset provides a collection of comma-separated value (CSV) files. Each data file has its own identifiers, suffixed with 'ID'. E.g., SUBJECT_ID is assigned to a unique patient, HADM_ID refers to a unique admission, ICUSTAY_ID relates to a unique visit to ICU. Events such as notes, laboratory tests, and fluid balance are stored in a series of 'events' data files. E.g., OUTPUTEVENTS contains all measurements related to output for a given patient, while LABEVENTS contains laboratory test results for a patient. Data files prefixed with 'D_' are dictionaries and contain definitions for identifiers. Rows in CHARTEVENTS linked to a single ITEMID represent the measured concept, but actual name of the measurement is not present in this file. When CHARTEVENTS and D_ITEMS are joined by ITEMID, the details emerge.

The data files used are placed under 'data' folder in project root folder in Google Drive.

data

- D_ICD_DIAGNOSES.csv
- D_ICD_PROCEDURES.csv
- mimic3/PROCEDURES_ICD.csv
- mimic3/DIAGNOSES_ICD.csv
- mimic3/NOTEEVENTS.csv *italicized text*

HADM files were downloaded from https://github.com/jamesmullenbach/caml-mimic/tree/master/mimicdata/mimic3. They are also loaded into Google Drive under mimic3 folder.

- dev_50_hadm_ids.csv
- dev_full_hadm_ids.csv
- test_50_hadm_ids.csv
- test_full_hadm_ids.csv
- train_50_hadm_ids.csv
- train_full_hadm_ids.csv

Some data statistics:

- 50K+ patients, including their clinical notes and their corresponding ICD codes.
- 52,722 summaries.
- 8929 unique codes.

- The experiment used top 50 codes with a subset of 11,317 summaries for training, validation and testing.
- The full dataset has 47,719 training summaries, 1,631 validation summaries and 3,372 testing summaries.

HiCu algorithm has 2 processes.

- Pre-processing: raw files are read, and intermediate files are created.
- Post-processing: the model is trained on the intermediate files.

All these data files were preprocessed through a python script 'preprocess_mimic3.py'. The script is executed by running the below command:

```
python preprocess_mimic3.py
```

Pre-processing step needed many Python libraries: genism, nltk, numpy, pandas, scikit_learn, scipy, torch, tqdm, transformers, utils. The inputs csv files were read from the 'data' folder and 'mimic3' sub-folder under 'data', under the project root. The output files were saved under 'mimic3' sub-folder.

Model

The Curriculum Learning algorithm is described below.

The original paper uses the Poincare model using hyperbolic embeddings. Below is the high-level encoder decoder architecture and the hierarchical curriculum learning algorithm of the ICD coding model.

The HiCu architecture consists of an encoder-decoder framework combined with a hierarchical curriculum learning algorithm. The numbers in the above figure indicate the sequential execution order of our training algorithm. The model is first trained on labels at the first level of the label tree using level one decoder, and then proceeds to level two using the knowledge transfer mechanism. This process is repeated until the model reaches the final level in the label tree.

During training at each level, the hyperbolic embeddings of the ICD codes are used to guide the attention computation in the decoder. The hyperbolic embeddings allow the model to learn a representation of the ICD code hierarchy that is more structured and aligned with the hierarchical structure of the codes. The model was run the original HiCu algorithm with high-order grouping of ICD code blocks to create a two-level hierarchy and is trained on the mimic3/train_full.csv dataset to verify its performance.

HiCu Algorithm

Training

The MultiResCNN model was first trained locally on a computer with 16GB of RAM and 128MB of GPU memory. Additionally the data was loaded into Google Drive and model training was done on the cloud using this Google Colab notebook "Team26_Colab_Notebook_1.ipynb".

The training program is run through script 'main.py'. Model hyperparameters were maintained in 'options.py'.

- Depth: 5
- Epochs: 2, 3, 5, 10, 500
- Model: MultiResCNN
- Decoder: Hierarchical Hyperbolic
- Batch Size: 8, 16
- Workers: 1, 8, 16
- Drop Out: 0.2
- Loss Function: ASL (Asymmetric Loss)

Evaluation

The new model performance was compared with the original hierarchical model build on CAML, DR-CAML, HyperCore etc.

For the draft version, the MultiResCNN model was used. Comparison of the results were done respectively on full-code and on top-50-code from MIMIC-III data files. The HiCu algorithm was also run on multiple different encoder architectures.

In [ ]:

```
# Code to mount Google drive:
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# To Copy Google Drive to Google Colab local drive:
!cp -r '/content/drive/MyDrive/HiCu-ICD-Team26-Spring24' '/content/'

# Model training:
# Different training scripts were run with the desired model configuration
!python /content/HiCu-ICD-Team26-Spring24/runs/run_multirescnn_50.py
```

Sample output is shown below:

Results

As per the research paper, the HiCu model and its implementation on different encoders showed model performance improvements. AUC, F1 and Precion@K attributes were compared between performance outcomes from various runs on different models. It was observed, when the model was run deep (higher depth) into the hierarchy, and the epoch count increased, it resulted in lesser loss function.

A sample output of the model performance:

```
epoch finish in 189.40s, loss: 0.1643
file for evaluation: ./data/mimic3/dev_50.csv
[MACRO] accuracy, precision, recall, f-measure, AUC
0.4113, 0.6294, 0.5163, 0.5673, 0.8816
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4590, 0.7112, 0.5641, 0.6292, 0.9161
rec_at_5: 0.5904
prec_at_5: 0.6037
rec_at_8: 0.7229
prec_at_8: 0.4864
rec_at_15: 0.8768
prec_at_15: 0.3299
```

Analyses

Due to large quantity of data and low availability of higher-power GPUs, model was run with a single GPU, and only for few epochs. The team's intention would be to run all the encoder variations.

Plans

For the final project, the project team's plan is to extensively run the training scripts with different encoders with higher depth and higher epochs for broader period of time to get more comparable data points.

Below are the training scripts that are planned to be executed and compared against the original performance numbers.

• **MultiResCNN**:

o MultiResCNN_50, MultiResCNN_full

o MultiResCNN_HiCu0_full

o MultiResCNN_HiCuA_50, MultiResCNN_HiCuA_full

o MultiResCNN_HiCuA_asl_50, MultiResCNN_HiCuA_asl_full

o MultiResCNN_HiCuC_50, MultiResCNN_HiCuC_full

o MultiResCNN_HiCuC_asl_50, MultiResCNN_HiCuC_asl_full

• **RAC**:

o RAC_50, RAC_full

o RAC_HiCuA_50, RAC_HiCuA_full

o RAC_HiCuC_50, RAC_HiCuC_full

• **LAAT**:

o LAAT_50, LAAT_full

Additionally, the model would be tested with code changes for ablation.

References

• https://physionet.org/content/mimiciii/1.4/

• https://github.com/wren93/HiCu-ICD/blob/main/README.md

• https://github.com/gkajale2/HiCu-ICD-main

• https://github.com/blackhat-93/HiCu-Reproduce

• https://github.com/foxlf823/Multi-Filter-Residual-Convolutional-Neural-Network

- https://github.com/foxfi823/Multi-Filter-Residual-Convolutional-Neural-Network
- https://paperswithcode.com/paper/hicu-leveraging-hierarchy-for-curriculum
- https://arxiv.org/abs/2208.02301
- https://proceedings.mlr.press/v182/ren22a/ren22a.pdf
- https://github.com/jamesmullenbach/caml-mimic/tree/master/mimicdata/mimic3
- https://github.com/dbiswas0605/MCS-DS-CS598-DLH-HiCu

-------- Executable code starts from here --------

Pip libraries setup

In [ ]:
```
# # No need to run this cell
# # Saved here as a record of package versions that worked off-the-shelf from Colab on April 2024

# # Colab's python version was 3.10.12
# !pip install gensim==4.3.2
# !pip install nltk==3.8.1
# !pip install numpy==1.25.2
# !pip install pandas==2.0.3
# !pip install scikit-learn==1.2.2
# !pip install scipy==1.11.4
# !pip install tqdm==4.66.2
# !pip install transformers==4.38.2
# !pip install packaging==24.0
# !pip install torch==2.2.1+cu121
```

Check package versions

In [ ]:
```
import sys
import gensim
import nltk
import numpy
import pandas
import sklearn
import scipy
import tqdm
import transformers
import packaging
import torch

print("python:", sys.version)
print("gensim:", gensim.__version__)
print("nltk:", nltk.__version__)
print("numpy:", numpy.__version__)
print("pandas:", pandas.__version__)
print("scikit-learn:", sklearn.__version__)
print("scipy:", scipy.__version__)
print("tqdm:", tqdm.__version__)
print("transformers:", transformers.__version__)
print("packaging:", packaging.__version__)
print("torch:", torch.__version__)
```

```
python: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
gensim: 4.3.2
nltk: 3.8.1
numpy: 1.25.2
pandas: 2.0.3
scikit-learn: 1.2.2
scipy: 1.11.4
tqdm: 4.66.2
transformers: 4.38.2
packaging: 24.0
torch: 2.2.1+cu121
```

Check CUDA & RAM availability

In [ ]:
```
if torch.cuda.is_available():
    print("CUDA is available. GPU: " + torch.cuda.get_device_name(0))
else:
    print("CUDA is not available.")
```

```
    print('CUDA is not available.')

!nvidia-smi
```

```
CUDA is not available.
/bin/bash: line 1: nvidia-smi: command not found
```

In [ ]:

```
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 13.6 gigabytes of available RAM

Not using a high-RAM runtime
```

In [ ]:

```
import os
num_cores = os.cpu_count()
print("Number of CPU cores:", num_cores)
```

```
Number of CPU cores: 2
```

Transfer data to Google Colab local drive (faster training)

In [1]:

```
# Give access to Google Drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# Copy Google Drive to Google Colab local drive
!cp -r '/content/drive/MyDrive/HiCu-ICD-Team26-Spring24' '/content/'

# Change directory to HiCu-Reproduce folder in Google Colab
import os
os.chdir('/content/HiCu-ICD-Team26-Spring24')
print("Current directory:", os.getcwd())
print("Content of current directory:", os.listdir())
```

```
Mounted at /content/drive
Current directory: /content/HiCu-ICD-Team26-Spring24
Content of current directory: ['Copy of HiCu.ipynb', 'Team26_Colab_Notebook_1.ipynb', 'preprocess_mimic3.py', 'requirements
t', 'README.md', 'utils', 'runs', 'data', '.ipynb_checkpoints', 'main.py']
```

In [ ]:

```
import os
os.chdir('/content/HiCu-ICD-Team26-Spring24')
print("Current directory:", os.getcwd())
print("Content of current directory:", os.listdir())
```

```
Current directory: /content/HiCu-ICD-Team26-Spring24
Content of current directory: ['requirements.txt', 'Copy of HiCu.ipynb', 'preprocess_mimic3.py', 'utils', 'runs', 'README.m
'db_main.py', 'main.py', 'Team26_Colab_Notebook_0.ipynb', 'main1.py', '.ipynb_checkpoints', 'data', 'Team26_Colab_Notebook_1
ynb']
```

Run!

In [ ]:

```
# NOTE: Change the name of the .py training script to your desired model configuration
print("Current directory:", os.getcwd())
!python /content/HiCu-ICD-Team26-Spring24/runs/run_multirescnn_50.py

# Save results back to google drive
#!cp -r '/content/HiCu-ICD-Team26-Spring24/models' '/content/drive/MyDrive/HiCu-ICD-Team26-Spring24'
```

```
Current directory: /content/HiCu-ICD-Team26-Spring24
Starting run No. 1 of 1
Namespace(MODEL_DIR='./models', DATA_DIR='./data', MIMIC_3_DIR='./data/mimic3', MIMIC_2_DIR='./data/mimic2', data_path='./da
mimic3/train_50.csv', vocab='./data/mimic3/vocab.csv', Y='50', version='mimic3', MAX_LENGTH=4096, model='MultiResCNN', deco
='RandomlyInitialized', filter_size='3,5,9,15,19,25', num_filter_maps=50, conv_layer=1, embed_file='./data/mimic3/processed_
l_100.embed', hyperbolic_dim=50, test_model=None, use_ext_emb=False, cat_hyperbolic=False, loss='BCE', asl_config='0,0,0', a
reduction='sum', n_epochs='2,2,3,5,50', depth=5, dropout=0.2, patience=10, batch_size=8, lr=5e-05, weight_decay=0, criterio
rec_at_8', gpu='-1', num_workers=8, tune_wordemb=True, random_seed=0, thres=0.5, longformer_dir='', reader_conv_num=2, reade
rans_num=4, trans_ff_dim=1024, num_code_title_tokens=36, code_title_filter_size=9, lstm_hidden_dim=512, attn_dim=512, sched
=0.9, scheduler_patience=5, command='python main.py --model MultiResCNN --vocab ./data/mimic3/vocab.csv --decoder RandomlyI
alized --Y 50 --data_path ./data/mimic3/train_50.csv --MAX_LENGTH 4096 --embed_file ./data/mimic3/processed_full_100.embed
ne_wordemb --batch_size 8 --lr 5e-5 --n_epochs 2,2,3,5,50 --criterion prec_at_8 --random_seed 0 --num_workers 8', gpu_list=
1])
loading lookups...
Depth 0: 14
Depth 1: 30
Depth 2: 39
Depth 3: 47
```

```
Depth 4: 50
loading pretrained embeddings from ./data/mimic3/processed_full_100.embed
adding unk embedding
MultiResCNN(
  (word_rep): WordRep(
    (embed): Embedding(51921, 100, padding_idx=0)
    (embed_drop): Dropout(p=0.2, inplace=False)
  )
  (conv): ModuleList(
    (0): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(3,), stride=(1,), padding=(1,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(3,), stride=(1,), padding=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(3,), stride=(1,), padding=(1,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (1): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(5,), stride=(1,), padding=(2,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(5,), stride=(1,), padding=(2,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(5,), stride=(1,), padding=(2,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (2): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(9,), stride=(1,), padding=(4,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(9,), stride=(1,), padding=(4,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(9,), stride=(1,), padding=(4,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (3): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(15,), stride=(1,), padding=(7,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(15,), stride=(1,), padding=(7,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(15,), stride=(1,), padding=(7,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (4): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(19,), stride=(1,), padding=(9,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(19,), stride=(1,), padding=(9,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(19,), stride=(1,), padding=(9,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
```

```
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (5): ModuleList(
      (0): Conv1d(100, 100, kernel_size=(25,), stride=(1,), padding=(12,))
      (1): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(25,), stride=(1,), padding=(12,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(25,), stride=(1,), padding=(12,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (decoder): RandomlyInitializedDecoder(
    (U): Linear(in_features=300, out_features=50, bias=True)
    (final): Linear(in_features=300, out_features=50, bias=True)
    (loss_function): BCEWithLogitsLoss()
  )
)
train_instances 8066
dev_instances 1573
test_instances 1729
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning: This DataLoader will create 8 worke
rocesses in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is
ng to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the wor
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
Total epochs at each level: [2, 2, 3, 5, 50]
Training model at depth 4:
EPOCH 0
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with m
ithreaded code, and JAX is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
epoch finish in 4937.82s, loss: 0.3305
file for evaluation: ./data/mimic3/dev_50.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0791, 0.2174, 0.0946, 0.1318, 0.7155
[MICRO] accuracy, precision, recall, f-measure, AUC
0.1438, 0.6436, 0.1562, 0.2514, 0.7640
rec_at_5: 0.3373
prec_at_5: 0.3807
rec_at_8: 0.4422
prec_at_8: 0.3170
rec_at_15: 0.6166
prec_at_15: 0.2362

evaluation finish in 216.65s
saved metrics, params, model to directory ./models/MultiResCNN_RandomlyInitialized_A_BCE_50_Apr_13_04_39_00

EPOCH 1
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning: This DataLoader will create 8 worke
rocesses in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is
ng to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the wor
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with m
ithreaded code, and JAX is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
epoch finish in 4905.53s, loss: 0.2737
file for evaluation: ./data/mimic3/dev_50.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1592, 0.3589, 0.1886, 0.2472, 0.7842
[MICRO] accuracy, precision, recall, f-measure, AUC
0.2510, 0.7022, 0.2809, 0.4013, 0.8328
rec_at_5: 0.4392
prec_at_5: 0.4745
rec_at_8: 0.5580
prec_at_8: 0.3893
rec_at_15: 0.7301
prec_at_15: 0.2763

evaluation finish in 211.44s
saved metrics, params, model to directory ./models/MultiResCNN_RandomlyInitialized_A_BCE_50_Apr_13_04_39_00

EPOCH 2
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning: This DataLoader will create 8 worke
rocesses in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is
ng to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the wor
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with m
ithreaded code, and JAX is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```