Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Personalized Cancer Vaccine: An Improved Graph Neural Network Model for pMHC Binding Affinity Prediction

Author: ChiaYu Lin (VU:2729198, UvA:13692577)

1st supervisor: Adam Belloum
daily supervisor: Giulia Crocioni (Netherlands eScience Center)
2nd reader: Rob van Nieuwpoort

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

July 25, 2023

Abstract

Prediction of binding affinity for pMHC (protein Major Histocompatibility Complex) is a crucial step in formulating personalized cancer vaccines for advanced-stage cancer patients. The current state-of-art pMHC binding affinity prediction tool, MHCflurry, is trained on a sequence-based model using amino acid sequences data. However, MHCflurry is lack testing on rare peptides and alleles, which may lead to an overestimation of results. Instead of a sequence-based model, our work focuses on implementing a structure-based Graph Neural Network (GNN) model which takes into account the 3D structure of Protein-protein Interaction (PPI) data. The work started from the naive GNN model in the Deeprank-core package and enhanced its performance by applying a series of experiments on modifying the network architecture, training configuration settings, and pre-processing input PPI data. When compared to the retrained MHCflurry, my improved GNN model has demonstrated a 12.3%, 15.2%, and 6.1% of improvement in prediction performance on a shuffled, peptide-clustered, and allele-clustered data set, respectively. The result has suggested that the structure-based GNN model from the study excels over the sequence-based model MHCflurry 2.0 on pMHC binding affinity prediction in both shuffled and clustered data training on a total of 10088 PPI dataset.

Keywords: MHC, Graph Neural Network, Sequence-based model, Structure-based model.

Acknowledgements

I would like to express my sincere gratitude to everyone who has supported me throughout the duration of my Master's thesis.

Firstly, I would like to extend my deepest appreciation to my mentor at the eScience Center Amsterdam, Giulia Crocioni, for her unwavering support and mentorship. Her expertise in machine learning was invaluable to me, from initial project familiarization to meticulous progress planning. Her thoughtful advice and guidance in shaping the research question have been a constant source of inspiration to me.

Next, my heartfelt thanks to Dani Bodor and Team Flow. Their relentless help and support in helping me debug codes and providing timely advice have played a substantial role in this project. Their contribution has given me the strength to overcome obstacles and has made the process more fulfilling.

My appreciation also goes to Radboud University Medical Center for providing the binding affinity data essential to the completion of my research.

I would like to extend my sincere gratitude to Daniil Lepikhov and Dario Marzella for enriching my understanding of cluster-related knowledge. Their insightful explanations to share their expertise have greatly enhanced the quality of my work.

Finally, my deepest appreciation goes to Professor Adam Belloum. His diligent supervision and feedback on my progress have been instrumental in my journey. His invaluable inputs have helped shape my work into its current form and have inspired me to strive for academic excellence.

Once again, my deepest thanks to all of you. The completion of this thesis would not have been possible without your dedicated support and assistance.

Contents

1

Introduction

A personalized cancer vaccine is a product of immunotherapy, which is a new and effective way to treat cancer by utilizing the patient's own immune system to defeat cancer. A personalized cancer vaccine is formulated based on the selected mutated tumor peptides. These mutated peptides can activate the human body's immune system and effectively address tumor growth progression, without affecting with the body's normal cellular functions (1). The criteria for selecting appropriate mutated peptides is that they must have high binding affinities to MHC proteins. MHC protein is a group of genes found on the surface of the cells, providing information to the immune system related to health. To assess the binding affinity between mutated peptides and MHC proteins, they can form together into a complex, known as pMHC, to be selected for vaccine production.

There are already several cutting-edge tools that use machine learning techniques to predict pMHC binding affinity. The current state-of-the-art approach to predicting the binding affinities is a software tool called MHCflurry 2.0 (2). which achieved an area under the curve (AUC) prediction score of up to 0.90. However, MHCflurry 2.0 uses a sequence-based machine learning model that takes the amino acid sequence of pMHC as input. Since MHCflurry 2.0 uses a sequence-based model, it required a fixed length of inputs, which only supports variable-length of pMHCs up to 15-mers, containing limited information on the amino acid. Secondly, MHCflurry 2.0 were only trained and tested on the most common published datasets of peptides (2) which could result in poor predicting performance at rare alleles and peptides (3). Additionally, MHCflurry 2.0 did not apply data clustering configurations such as allele-clustering and peptide-clustering to the datasets, which could have led to an overestimation of accuracy, since it does not consider the real-life scenario of the appearance of rare allele and peptides.

1. INTRODUCTION

To solve the problem of the sequence-based MHCflurry 2.0 model, the thesis study adopts GNN, which is a structure-based machine learning model that takes the 3D structure of pMHCs as input for binding affinity prediction. The advantage of adopting a structure-based model is that it includes the complete information of a pMHC considering its space and structure. Furthermore, in order to better simulate pMHC data in real-world scenarios, the study employs clustered dataset configurations for model training to provide more accurate prediction results.

The tool that currently adopted the GNN model for pMHC binding affinity prediction is a package named Deeprank-core (4), developed by the eScience Center Amsterdam. However, the original GNN model from the Deeprank-core package (4), which performed an AUC score of 0.8285 and an MCC score of 0.5032 in testing, does not demonstrate good predictive performance. Therefore, further improvement is necessary for the GNN model.

The research aims to develop a neural network model that utilizes the Geometric Deep Learning (GDL) mechanism to predict the binding affinities of pMHC based on its 3D structure to select the suitable mutation tumor peptides for formulating a personalized cancer vaccine. In this research, I started from the original naive GNN model in the Deeprank-core package (4) and developed an improved GNN model for pMHC binding affinity prediction by (1) adjusting the configuration settings for GNN model training, (2) modifying the design of GNN architecture, and (3) input data pre-processing before GNN model training, to identify how much the performance of structure-based GNN model can reach compared to the performance with MHCflurry 2.0's sequence-based model. The research question of this paper is as followings:

> *RQ1: Can the structure-based model perform better than a sequence-based model on pMHC prediction?*

The performance comparison between my improved GNN model and MHCflurry will be analyzed based on both the shuffled and clustered (allele and peptides) dataset. Therefore, I extended my research question to the following subquestions:

> *RQ1.1: How much the performance of an improved structure-based model can reach than a sequence-based model on pMHC prediction based on a shuffled dataset configuration?*

> *RQ1.2: How much the performance of an improved structure-based model can reach than a sequence-based model on pMHC prediction based on clustered(allele and peptides) dataset configuration?*

The paper will be divided into 7 sections. First, an *Introduction Section* on the goal and motivation of the research which carried out the research questions of the topic. Secondly, the *Background Section* provides more knowledge regarding the research. Followed by *Related Work* discussing relevant experiments conducted in my research. The *Design Section*, gives a clear overview of the GNN model and experiment environment. Afterward, the *Evaluation Section* focuses on the various experiments conducted to explore the better-performed GNN model. The *Discussion Section* provides an interpretation of the research results and recommendations for future research. Finally, the *Conclusion Section* summarizes the contribution of the research.

2

Background

Over the years, the medical community has devoted itself to researching ways to treat cancer in patients with advanced-stage cancer. Common cancer treatments include surgery, chemotherapy, radiation therapy, and immunotherapy. Although there are several treatment options for cancer, each of them often has limitations, such as surgical removal of the cancer site, which is only effective for early-stage cancer. Radiation therapy is associated with damage to local peripheral tissues, while chemotherapy causes severe side effects as a result of the drugs administered. Among them, cancer immunotherapy is a relatively new treatment but has already established itself as a majorstay of mainstream cancer treatment (5).

The principle of immunotherapy is to treat cancer by utilizing the patient's own immune system. The treatment is accomplished by selecting targeted tumor mutation (5) and using these mutated tumor peptides to activate the immune system's ability to combat the disease (6) and thereby eliminate the tumor. Immunotherapy has shown excellent results in the treatment of a variety of malignancies and is more efficient, better tolerated, and reduces adverse reactions compared to other conventional treatments (7).

Personalized cancer vaccines, as a product of the immunotherapy approaches, are formulated on the basis of selective mutated tumor peptides. However, a patient's cancer can contain thousands of mutations (8). Therefore, selecting the appropriate tumor mutation peptides to utilize as vaccine candidates represents the major challenge for cancer immunotherapy. To serve as an effective vaccine candidate, mutation peptides must have high binding affinities to Major Histocompatibility Complex (MHC), together known as a pMHC complex. MHC protein is a group of genes found on the surface of the cells, which is closely related to immune response, such as assisting the immune system to recognize foreign substances (9). The term binding affinity in molecular science refers to the strength

of binding interactions between two molecules, which is the mutated peptide and MHC protein in the formed pMHC complex. Therefore, a tumor mutation peptide's suitability as an ideal vaccine candidate increases with the mutation peptide's binding affinity for the MHC.

With approximately four million binding affinity data available from Binding MOAD (Mother Of All Databases), Community Structure-Activity Resource (CSAR) dataset, ChEMBL, and other molecular databases (10), state-of-the-art approaches utilized machine learning (ML) techniques by taking these binding affinity data as input for model training. However, these common datasets are formed by sequences of amino acid genetic code and therefore the state-of-art pMHC binding affinity prediction tools' ML models were typically sequence-based (2) (11), which contains limited information for prediction. Especially in the case of pMHCs, which contain highly variable peptides (12), it is difficult to represent the complete information of a pMHC with a sequence of amino acid genetic code. The 3D modeling approach is an ideal way to predict pMHC binding affinities, which is not limited by the peptide variability length and can also identify more rare MHC alleles through 3D structures, which have not appeared before in the common molecular databases.

GDL is a newly developed machine learning technology for 3D modeling and is optimized for molecular sciences. The concept of GDL is that its neural models are generated from non-Euclidean domains, for example, graphs. This technique allows the representation of pMHC to be rendered graphically from 3D structures for model training. The study thus involves the combination of data-driven GDL with physics-based 3D modeling. The PANDORA modeling protocol (13) is used to build the 3D structure of pMHC, which is then used to label whether a mutation peptide binds with pMHC. From the use of these labeled data sets, the GNN model for predicting binding affinity is trained. Figure 2.1 demonstrated a 3D structure of a pMHC, the green-tangled gene refers to the MHC protein and the shorter red gene is the mutation peptide.

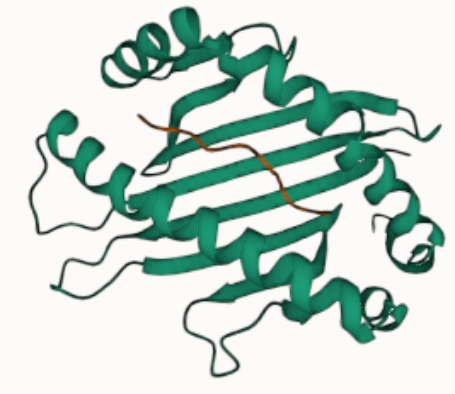

Figure 2.1: The 3D Structure of a pMHC

3

Related Work

In this section, papers relevant to my master's thesis study will be discussed. The section is further divided into four subsections: *Graph Deep Learning in Healthcare*, *State-of-art pMHC Prediction Tool, Hyperparameter Tuning for Neural Network, and Data Pre-processing for Neural Network*.

For the *Graph Deep Learning in Healthcare*, I will mention two types of applications that adopt the graph deep learning technique in the field of healthcare, the first application is related to the internal structure of the human body, regarding the diagnosis of brain activity. The second application is related to the effects of the external environment on the human body, regarding anomaly detection in air quality. Secondly, in the *State-of-art pMHC Prediction Tool*, I will give a deeper introduction to two state-of-art tools used for pMHC binding affinity prediction, which are the sequence-based model MHCflurry 2.0, and the CNN-trained model DeepRank. Followed by, in *Hyperparameter Tuning for Neural Network*, I will discuss a few methodologies papers regarding similar experiments conducted for my thesis to perform hyperparameter tuning in neural networks. Lastly, in *Data Pre-processing for Neural Network*, methodology related to data pre-processing for the neural network will be mentioned and compared to my study.

3.1 Graph Deep Learning in Healthcare

Analysis of Brain Activity

X Li et.al (14) suggested that the human body's complex brain regions have a connection to a neurological disorder or cognitive stimuli. Li's team proposed a GNN architecture, BrainGNN (15), which analyzes and converts the functional magnetic resonance images (fMRI) to a graph-based model to discover the neurological abnormalities in human brain

regions. To model the brain regions as a graph, brain regions of interest (ROIs) are defined as graph nodes, and the functional connections between those ROIs are defined as graph edges. BrainGNN can perform classification tasks on classifying neuro-disorder and healthy brain regions.

BrainGNN has some similarities compared to the work in the study, which also adopts the GNN architecture to perform classification on the binding affinity between mutation peptides and MHC proteins.

Anomaly Detection in Air Quality

X. Lin et.al (16) proposed a GNN model that combines spatial and temporal correlation to detect abnormal events in air quality data and protect human health. Existing anomaly detection models for air quality data often overlook the spatial aspects of regional air pollutants, relying on single-station approaches only. In Lin's anomaly detection GNN model, they characterize the relationship between the different air quality monitoring stations from both spatial and temporal aspects and construct the information into a graph structure dataset.

To represent spatial correlation, a weighted adjacency matrix is established to calculate the interconnections between the monitoring stations including the methodology of pollution data measurement.

On the other hand, to represent temporal correlation, a feature matrix is constructed to record the variants of pollutants at each monitoring station. The graph structure data are built upon these matrices' data from the continuous changes of node and edge information and further trained using the GNN model to predict whether the air quality has a serious effect on human health.

Similar to the air quality anomaly detection project in this paragraph, in which the graph structure data are constructed based on the calculated matrix information. For the PPI data generation of our study, the sequences of molecules are also calculated using Position-Specific Scoring Matrices (PSSMs), which count the positions of each amino acid from the molecular sequence. The PSSMs results are used to formulate a complete structure of the PPI data and further used in the training of the pMHC GNN model of the study.

3.2 State-of-art pMHC Prediction Tool

MHCflurry

O'Donnell et.al (17) proposed an open-source software tool *MHCflurry* and its improved

version MHCflurry 2.0 (2) to predict the binding of MHC-I peptides. Before the development of MHCflurry, the training process of MHC-I binding tools training process, could only be operated by developers, and its use was primarily restricted to private research purposes. In response to the surge in the discovery of tumor neoantigens, O'Donnell's team has released a Python-based package, MHCflurry, which is open-sourced and simple-to-install (18). MHC-flurry features a configurable interface, and its machine-learning model can be modified and re-trained according to the user's needs. MHCflurry is made up of two predictors, the BA predictor and the AP (antigen process) predictor, and thus, the two predictor model is built separately.

For the BA predictor, MHCflurry trained the model based on the binding affinity data of MHC peptides. It supports variable peptide lengths between 8 and 15 using a fixed-length encoding algorithm. The first four and last four alleles of the peptide are regarded as the "anchor location" that is most related to MHC. Following that, the inputted peptides are translated into a 15-length sequence with the anchor position alleles fixed and the residue alleles filled in with an X character. MHCflurry has the ability to train up to 14993 MHC alleles on a single neural network model.

For the AP predictor, it is trained based on the hits and decoys from the BA predictor. In MHCflurry 2.0 (2), O'Donnell's team defined a binding affinity less than 0.5744 as a "hit". MHCflurry took advantage of adopting training based on the combination of models using the BA and AP predictors, which feature 140 training models in total. The data set MHCflurry used was gathered from 11 studies that predict pMHC, which comes to 493,473 Mass Spectrum (MS) data and 219596 binding affinity data in total (2). Additionally, MHCflurry achieved a high prediction efficiency with up to 7000 identification per second with an AUC score of 0.90.

The study of the thesis, like MHCflurry, is an open-source software package named Deeprank-core freely available on GitHub (4) that uses the binding affinity value between the MHC protein and mutation peptide for model training. Similar to MHCflurry, Deeprank-core's model outputs the binding affinity results and scales them from 0 to 1 according to their distance value. However, instead of using the fixed-length encoding algorithm in MHCflurry that only takes into account the anchor location of peptides, the package focuses on the analysis of the 3D intersection area of two protein chains, which is the core area that best reflects the binding situation of the data. Additionally, the data used for model training were clustered and shuffled, which better classified the distribution of data sets and makes the prediction outcome more reliable during the actual pMHC binding affinity prediction.

DeepRank

DeepRank(19) is a software tool developed by the eScience Center Netherlands for predicting the binding affinity of pMHC based on the CNN deep learning framework. DeepRank adopts the widely used Pytorch(20) package to implement its neural network model(11). The training procedure begins with the model receiving HDF5 format files containing multiple PPI data with features and labels. Users can simply filter the PPIs from the HDF5 files that they intend to input into the model based on the features and labels of the PPIs (11). Those PPI data that match the values set by the user are fed into the neural network model, translated into a grid, and trained through a series of CNN layers, such as convolutional layers, pooling layers, and finally fully connected layers.

The Deeprank-core package (4) of my study uses the same data sets of DeepRank for model training. Additionally, both projects use the HDF5 files that store the PPI data as inputs for model training. The major difference is, our package is based on the GNN deep learning framework which translated the PPI information into graphs instead of grids. Moreover, the package is built upon the Pytorch package (20), the Geometric Pytorch (21), which is specially designed for the implementation of GNN architecture. My study modified the design of the neural network architecture from the Deeprank-core package and experimented by employing additional neural network techniques, such as batch normalization and standardization, and expanding the number of convolutional layers to enhance prediction performance.

3.3 Hyperparameter Tuning for Neural Network

Increase Batch Size

S.L.Smith et.al (22) indicate that decreasing the learning rate and increasing the batch size present the same learning curve in the neural network. Furthermore, by using a larger batch size, the total number of parameter updates reduces, which leads to a more efficient training process. However, in contrast, a larger batch size will cause a slight decline in performance. Smith compared the learning curves of decaying the learning rates and increasing the batch sizes while recording the number of parameter updates. Their experiment result showed that the learning curves of decaying the learning rates and increasing the batch sizes showed no differences, but the parameter updates required reduced by nearly half, confirming that large batch size training is a good approach to implement an efficient training model.

With reference to Smith's result, my study carried out an experiment on analyzing the relationship between different batch sizes, the amount of time taken on model training, and their performance accuracy to explore the trade-offs. Unlike Smith's experiment, we tried more different batch sizes and focused more on the overall performance accuracy and training time rather than on the number of parameter updates.

Expand Convolution Layers

Romanuke et.al (23) proposed the finding of appropriate convolutional layers required for the CNN architecture. Romanuke suggested that for a neural network designed for image recognition, the number of convolution layers required depends on the complexity of the image's composition. If the image contains multiple categories, features, colors, and chrominances, it may require a greater number of convolutional layers. He used different complexity of image data sets as experimental objects and apply a different number of convolutional layers to analyze the error rate of CNN training results. Romanuke obtained a general rule from his experiments that for a simple image data set at least four convolutional layers are needed, and for more complicated image data sets, the convolutional layers should start from 5 or even more.

My study also conducted an experiment on finding the appropriate number of convolutional layers. Since Romanuke experimented on image data sets, while the study of my thesis analyzes 3D pMHC structure, the content and complexity of data sets may differ a lot, so the general rule from their experiment result can not be applied to my project directly. Thus, I started the experiment by using basic two convolutional layers and eventually increase the number of layers to investigate the overall performance.

3.4 Data Pre-processing for Neural Network

Input Data Transformation

Thang N. Ha et.al (24) suggested the importance of applying data transformation and standardization before it is involved in the training of the neural network. This is due to most machine-learning applications being based on Gaussian statistics which demand a Gaussian-like distribution of the data (24). Thang conducted an experiment to compare the classification performance in machine learning with inputted data without transformation and data with logarithmic transformation. The experiment results showed that applying a logarithmic transformation to each input attribute helps reshape the data distribution

to better replicate a Gaussian distribution and thus improves the network's classification performance (24).

My study adopted the data normalization technique for each feature in the data set. Different from Thang's experiment, which only applies a logarithmic transformation to the inputted data, we refer to other paper's experiments and added square-root and cubic-root transformation (25) (26) to our data set in order to find the most suitable transformation method to make each protein feature present more normal-distributed. In addition, along with data transformation, a standardization algorithm is applied.

4

Design

This section includes the original architecture of the GNN model, the experiments conducted to achieve the goal of improving the prediction performance of pMHC binding affinity, the performing measurement metrics, the content of the PPI data set, and the environment used to implement these experiments.

4.1 Data source

The data sets used for the training of the neural network model were offered by our cooperation partner, Radboud University Medical Center (Radboud UMC). In the project, the smallest data set unit is called a PPI. PPI represents the interface between two proteins, which are the tumor mutation protein and the MHC protein. A schematic diagram of PPI is displayed in Figure 4.1, where the blue block refers to the interface area the two protein chains actually interact with each other. The Deeprank-core project (19) chose to select only the PPI instead of the entire structure from both proteins because the interface area is where the chemical reaction really takes place and further contains more useful information for predicting bindings. In total, 100088 PPIs data were generated from the 3D structure of pMHCs and provided for model training. Each PPI data contains information regarding the feature type of graph nodes and edges and a target value used for classification that can be used for further evaluation.

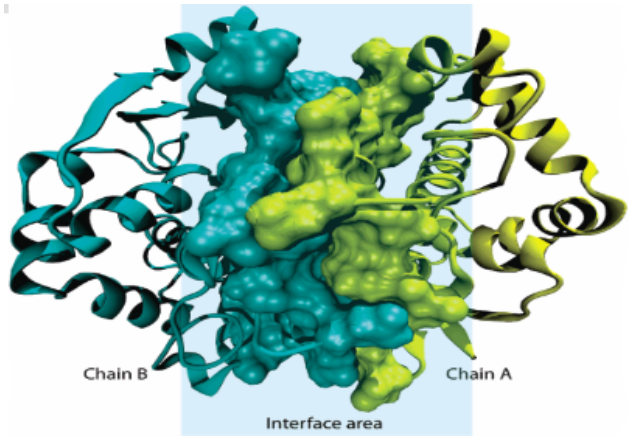

Figure 4.1: Sample of a Protein-protein Interface (27)

4.1.1 Target values

The target value, BA, from the PPI is a binding affinity value representing the strength of binding between the two protein chains. The greater the BA value, the more bonded the two proteins are with one another. According to the BA, a target value called binary (0 or 1) was labeled for the PPI. A zero is given when the BA value is less than 500nM(nanomoles), meaning that this PPI is marked as not bounded, while a one is given when the BA value is greater than 500nM, marking this PPI as bounded. The criteria of distance value 500nM to decide whether the pMHC is bound in my study follows the same criteria used in MHCflurry 2.0 (17). The GNN model from my master's thesis study is trained based on the binary value from the PPIs.

4.1.2 Features

Each PPI was also given multiple features to better provide information about the proteins. The features can be divided into node features and edge features, each describing the characteristics of this PPI after the conversion into a graph. These features were built and pre-defined by the Netherlands eScience Center, this including residue type, residue size, the number of donors, acceptor atoms, and up to 33 kinds of features. In the study, one-hot encoded features (polarity, res_type, covalent, same_chain, and pssm) are excluded from the analysis of the experiments since the mainstream opinion from the ML community suggests that one-hot encoded variables may result in poor performance or unexpected results in classification (28). Therefore, the remaining 28 types of features are used in the

analysis of the study. The detailed names of features and their descriptions can be found in the project's documentation.

4.2 Overview of Graph Neural Network Model

The study aims to design an improved GNN model based on the naive GNN model in the Deeprank-core package. Thus, an introduction of the original design of the GNN model in Deeprank-core along with a complete training procedure of how the performance of pMHC binding affinity is predicted will be described in this section.

4.2.1 Graph Neural Network Model

The original GNN architecture in the Deeprank-core package applied Graph-MLP to the network which combined the advantage of GNN architecture and multi-layer perceptrons (MLPs). GNN shows great performance in handling non-Euclidean structural data and MLPs eliminate the time-consuming problem caused by message passing (29). The GNN structure consists of two graph convolutional layers, with a graph MLP layer followed. In each graph convolutional layer, a message-passing function is defined which takes the number of node features, the number of edge features, and the edge indices as input. The graph convolutional layer further generates, aggregates, and calculates the surrounding information for each node and edge from the PPI graphs. The edge and node MLP layer is where each specific node and edge's calculation is done, in which a linear transformation is performed based on the information from the graph's node and edge, following an activation function, ReLU (Rectified Linear Unit)(Equation 4.1).

$$Relu(z) = max(0, z) \tag{4.1}$$

Followed by the two graph convolutional layers, is the graph MLP layer. The graph MLP layer further does the calculation by applying two linear transformations with an activation function between them with the node and edge information inputted and a hidden size set as 128. A detailed design of GNN architecture is shown in Figure 4.2.

4.2.2 Training Process

The training Figure 4.3 of pMHC binding affinity starts from reading the pMHC complexes from the 100088 inputted PPI data sets. The complexes are then transformed into graph interfaces, which contain node and edge feature information of the graphs. The GNN model is built upon these node and edge information. Afterward, a classification prediction of

binding affinity can be concluded from the output of the GNN model. Finally, the AUC and MCC scores, indicating the performance of the prediction are computed.

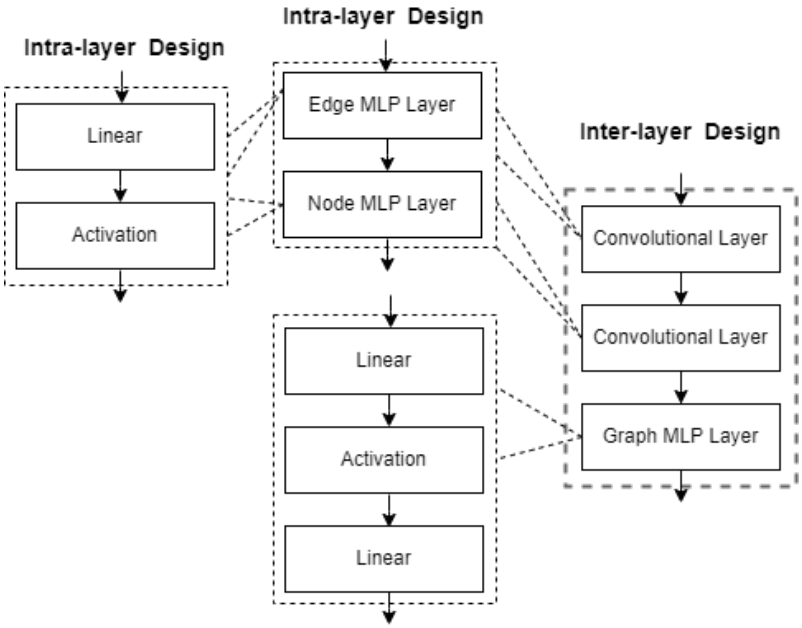

Figure 4.2: Original GNN Architecture Design

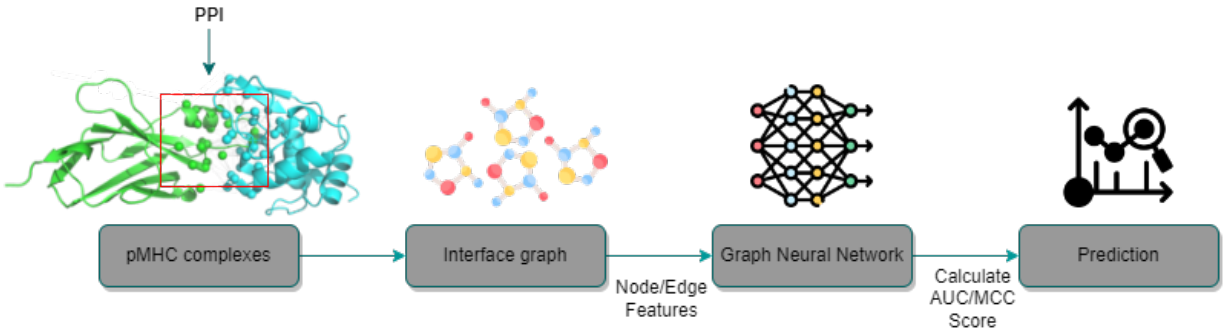

Figure 4.3: Training Process of pMHC Binding Affinity Prediction

4.3 Experiments Overview

4.3.1 Experiments Plans

The experiments in the work focus on how to implement an improved GNN model for the deeprank-core package (19), this includes increasing the accuracy of predicting the binding

affinity of pMHC complexes from the PPI data sets and the overall prediction efficiency. The types of experiments can be divided into three main categories:

1. **Adjustment of the configuration settings for GNN model training.**

   Training configurations may have a direct impact on the training performance of the model. For example, an insufficient number of training epochs will lead to an excessive training loss. Therefore, the experiments conducted in this category will revolve around trying different combinations of configurations to improve the predicting performance, for example, the experiment on different batch sizes.

2. **Modification on the design of GNN architecture**

   The architecture for the original GNN is described in subsection 4.2.1. However, from the research papers regarding the design of GNN architecture (23) (30) (31), there are a lot of machine-learning-related techniques to apply to the model to improve the prediction performance. Therefore, the experiments conducted in this category will revolve around implementing different GNN architectures for model training, such as adding batch normalization, and convolutional layers.

3. **Input data preprocessing**

   Data preprocessing is a common technique used in machine learning, it can help the model training achieve better performance by formatting the input data into a proper distribution that best matches the network architecture. The experiments conducted in this category will revolve around implementing different data pre-processing methods before applying model training, such as data standardization and feature transformation.

4.3.2 Performance Metrics

All the experiments conducted in the study employ two commonly used metrics to evaluate the training performance of my GNN model, which are the Area Under the Receiver Operating Characteristic Curve (AUC) and Matthews Correlation Coefficient (MCC).

Area Under the Receiver Operating Characteristic Curve

AUC is an excellent metric when it comes to measuring classification performance. AUC indicates how well the model can distinguish between classes (32). The neural network model from my thesis study is developed based on classification. Its prediction result is

classified through a binary value, where 0s stands for not binding and 1s stands for having a good binding affinity. Moreover, in the paper published for MHCflurry 2.0 (2), they chose AUC as their primary performance metric. To better compare the prediction performance of my work with MHCflurry 2.0, I selected AUC as the main performance metric, the higher the AUC, the better the model is at distinguishing whether the PPIs are binded or not.

Matthews Correlation Coefficient

The second performance metric used in the work is MCC score (Equation 4.2). The MCC score is a reliable statistical indicator that takes into account all of the four confusion matrix categories, so it offers a truthful score in evaluating binary classifications (33). MHCflurry 2.0 also adopts the MCC score as one of its performance metrics. Considering a more accurate comparison to MHCflurry 2.0, I applied the MCC score as the study's performance metric and select the threshold corresponding to the highest MCC score for each experiment.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{4.2}$$

4.3.3 Training Configurations

In the GNN architecture, common training configuration settings include batch size, maximum epoch size, minimum epoch size, learning rate, and optimizer (34). The project's training model also applies more configurations to prevent the over-fitting of model training such as early stopping patience and early stopping maximum gap, which decides the timing to stop the training process when the training shows no improvement after certain epochs or the training and validation loss decreases to a stable point with a certain gap. Besides, a class weight function has been utilized which gives different weights to the binary target value to prevent the imbalance of the data set. The original training configuration was set as the following parameters shown in Table 4.1.

Batch size	**Epoch size**	**Min epoch size**	**Learning rate**
16	40	20	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Optimizer**
15	0.06	True	Adam

Table 4.1: Original Configuration Setting

4.3.4 Data Set Configuration

The GNN models are trained on two types of data set configurations: shuffling and clustering.

Data Shuffling

The experiments of the work were all first done on a shuffled data set with different random seeds. This helps randomize the order of the data that appeared in the training set each time and therefore provides a more reliable performance outcome. During my work, I performed each experiment five times on a shuffled data set with distinct seeds.
The data split of the training, testing, and validation data set for shuffling is shown in Table 4.2, in which the study takes 90% of PPI data as the training set and leaves the rest 10% of data as testing. Besides, 20% of data from the training set will be used as validation.

Train	**Validation**	**Test**
90% of data	20% of training data	10% of data

Table 4.2: Data Split in GNN Model Training for Shuffled Data Set

Data Clustering

The work can be further divided into two clustering methods, peptide clustering, and allele clustering. The idea of clustering is to obtain similar peptide or allele sequences, which have smaller distances between each other into the same cluster. Radboud UMC computes the distances using a PAM scoring matrix, which calculates the value from the matrices based on the amino acid sequences of peptides and alleles (35). During the model training process, the cluster which has the furthest distance from other clusters will be left only for the testing set, while clusters having a smaller distance between each other will be used as the training and validation set. This approach simulates the real-life scenario in which there will always be rare alleles and peptides not been observed before and therefore not trained by the training set. During the work, I also performed the improved GNN model on a peptide-clustered and allele-clustered data set to increase the confidentiality of performance outcomes and ensure it is not over-dependency on the training data.

For allele clustering, the work created two clusters based on the calculated distance from three different types of MHC-I alleles (HLA-A, HLA-B and HLA-C). Radboud UMC takes the bigger cluster between the two clusters as the training and validation set, and the smaller cluster, which is the most distant one as the testing set. The data split of the

training, testing, and validation data set for allele-clustered is shown in Table 4.3, in which we take 72%, 18%, and the remaining 10% of PPI data as the training set, validation, and testing set, respectively.

For peptide clustering, the work created ten clusters based on the calculated distance from the sequences of peptides using the GibbsCluster algorithm (36). Radboud UMC takes the fourth cluster as the testing set since it's the most distant one compared to the rest of the clusters. The remaining nine clusters are then used for the training and validation sets. The data split for peptide-clustered shown in Table 4.4 also differs, which take 66%, 16%, and the remaining 18% of PPI data as the training set, validation, and testing set, respectively.

Train	**Validation**	**Test**
72% of data	18% of data	10% of data

Table 4.3: Data Split in GNN Model Training for Allele-clustered Data Set

Train	**Validation**	**Test**
66% of data	16% of data	18% of data

Table 4.4: Data Split in GNN Model Training for Peptide-clustered Data Set

4.3.5 Training Environment

The experiments conducted in the study were run on Snellius, which is a Dutch supercomputer established by SURF (37) for IT in Dutch education and research providing fast processors and GPUs. The experimenting environment uses gpu partitioning, where each node contains 72 cores and 480 GiB available memory size. Each experiment is submitted onto Snellius as a job, and assigned 1/4 of the node, which is 18 cores with 120 GiB memory. One thing to note is that instead of using all 18 cores of CPU, I used 16 cores to load the training data into the trainer model. The detailed environment configuration can be checked on Snellius's official page.

5

Evaluation & Results

In this section, a detailed explanation of the design of the experiments, the results I obtained, and how much they improved the overall performance compared to the original GNN model will be provided. The experiments will be introduced in three categories: experiments about (1)*Adjusting the configuration settings for GNN model training*, experiments about (2)*Modifying the design of GNN architecture*, and lastly, experiments on (3)*Input data pre-processing*. Due to the time and resource constraints on Snellius, all the experiments done in the study will be trained with a shuffled data set and run five times each with a random seed. After discovering the best-performed GNN model from the experiments, the improved GNN model will then be trained on other data set configurations, allele-clustering, and peptide-clustering. Lastly, a prediction performance comparison of the retrained sequence-based MHCflurry 2.0 to the improved structure-based GNN model of the study on all types of data set configuration will be provided.

5.1 Experiments on Adjustment of Configuration Settings

5.1.1 Batch Size Experiment

Experiment Description

S.L.Smith et.al (22) stated that raising the batch size will increase parallelism and reduce training times. However, the trade-off is, the increased batch size may cause a slight drop in performance accuracy (22). Therefore, this experiment aims to explore different batch sizes and records the time required for the model to train along with the prediction results obtained to decide the best batch size for the project. The batch sizes which the experiment conducted were 16, 64, 128, 256, 512, and 1024, where the batch size of 16 was the original

setting. Other configuration settings for model training are shown in Table 5.1, which is the same as the original GNN model.

Batch size	**Epoch size**	**Min epoch size**	**Learning rate**
16/64/128/256/512/1024	20	40	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Standardization**
15	0.06	True	True
Optimizer			
Adam			

Table 5.1: Configuration Setting for Batch Size Experiment

Experiment Result

Throughout the experiment, all training for different batch sizes ended successfully, with the exception of batch size 1024, which fails due to memory exhaustion. Considering the project budget, allocating more memory is not a viable option. As a result, we decided to abandon the attempt to conduct experiments with batch sizes of 1024 and instead focus on the comparison among other batch sizes.

Table 5.2 showed the average AUC score results for different batch sizes each ran five times with random data seeds. From the comparison, the experiments using batch size 64 showed the best-predicting performance with an average of AUC 0.8817, 0.8588, and 0.8551 on the training, validation, and testing sets, respectively. Batch size 512 shows the worst-predicting performance with an average of AUC 0.8682, 0.8524, and 0.8496 on the training, validation, and testing sets, respectively. Other batch sizes, 64, 128, and 256 showed a moderate performance with similar results.

	AUC Score (Training)	AUC Score (Validation)	AUC Score (Testing)
Batch Size 16	0.8787 ± 0.0031	0.8585 ± 0.0020	0.8523 ± 0.0054
Batch Size 64	0.8817 ± 0.0063	0.8588 ± 0.0054	0.8551 ± 0.0044
Batch Size 128	0.8778 ± 0.0040	0.8554 ± 0.0029	0.8518 ± 0.0109
Batch Size 256	0.8750 ± 0.0044	0.8562 ± 0.0025	0.8530 ± 0.0037
Batch Size 512	0.8682 ± 0.0033	0.8524 ± 0.0015	0.8496 ± 0.0027

Table 5.2: Average AUC Scores for Batch Size Experiments

With our second performance metric, MCC score (Table 5.3), the experiments using batch size 64, also achieve the highest scoring compared to other batch sizes, with an average MCC score of 0.5994, 0.5587, and 0.5472 on the training, validation, and testing sets, respectively. Batch size 512 also receives the worst-predicting performance with an average MCC score of 0.5723, 0.5411, and 0.5394 on the training, validation, and testing sets, respectively.

	MCC Score (Training)	MCC Score (Validation)	MCC Score (Testing)
Batch Size 16	0.5972 ± 0.0058	0.5571 ± 0.0040	0.5442 ± 0.0056
Batch Size 64	0.5994 ± 0.0110	0.5587 ± 0.0134	0.5472 ± 0.0078
Batch Size 128	0.5915 ± 0.0104	0.5543 ± 0.0067	0.5470 ± 0.0211
Batch Size 256	0.5864 ± 0.0100	0.5522 ± 0.0036	0.5489 ± 0.0038
Batch Size 512	0.5723 ± 0.0055	0.5411 ± 0.0045	0.5394 ± 0.0077

Table 5.3: Average MCC Scores for Batch Size Experiments

Figure 5.1: Timings for Batch Size Experiments

In order to test whether increasing the batch size improves the total timing of model training, we did a comparison of the time (minutes) taken for each batch size experiment to be finished training (Figure 5.1). It is obvious that the total timing declined rapidly from more than 200 minutes for batch size 16 to around 170 minutes for batch sizes 64 and 128. The time taken for the batch-size experiments to train partly corresponds to Smith's statement that increasing the batch size will reduce training times when the size increases from 16 to 128. However, unlike Smith's result, as the batch size rises from 128 to 512, the total time increases as well. This could be due to the larger memory space required for the experiment, which burdens Snellius' load and influences its processing speed.

Experiment Conclusion

In conclusion, we adopted batch size 64 since both the AUC and MCC metrics display a slight improvement compared to the original batch size 16 (Figure 5.2, Figure 5.3). Moreover, adopting batch size 64 is a good trade-off between timing and metrics performances. Besides, we are also aware that the timings to train the model do not decline that dramatically as the batch size increases, as we thought at the beginning. This may be due to the uncertainty of resource allocation in the cloud computing environment. Thus, the timings can only be used as a reference indicator in my experiment, but it is already enough to

understand which batch size uses the least time. In the following experiments, batch size 64 will be applied replacing batch size 16 in the original GNN training configurations.

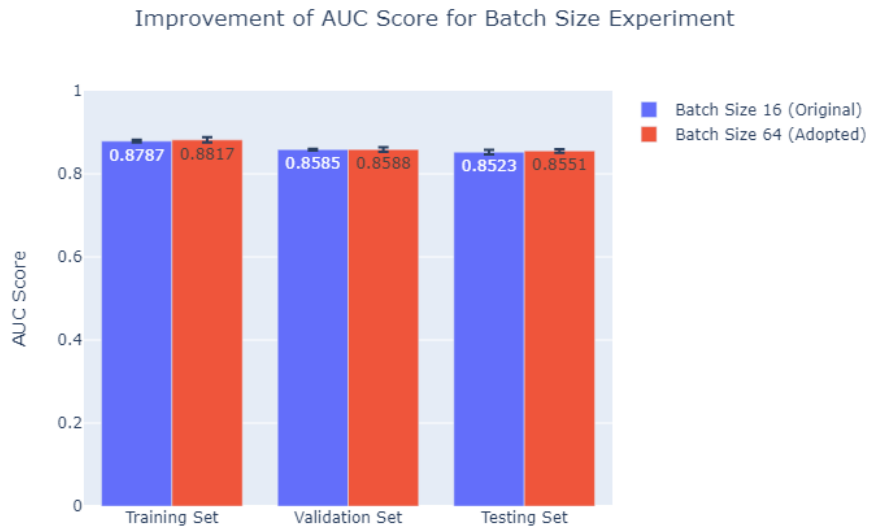

Figure 5.2: Improvement of AUC Score for Batch Size Experiment

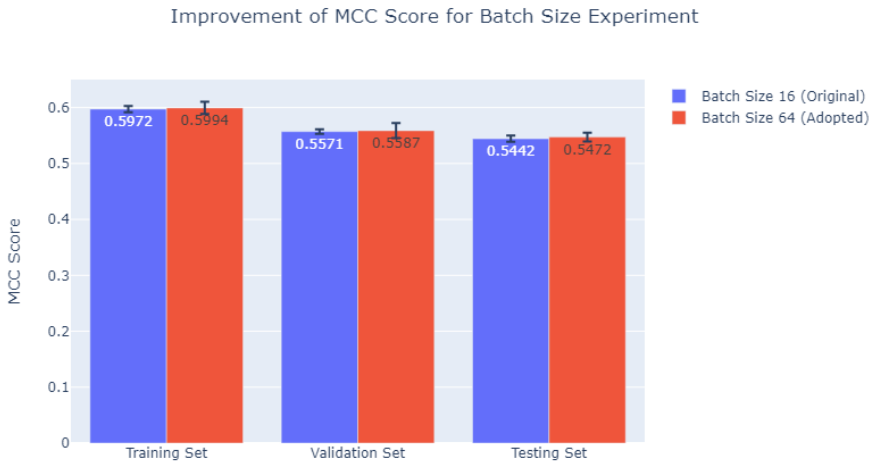

Figure 5.3: Improvement of MCC Score for Batch Size Experiment

5.2 Experiments on Modification of GNN architecture

5.2.1 Batch Normalization Experiment

Experiment Description

Batch Normalization (BN) is a technique added to the intermediate neural network layers to standardize layer outputs before passing them to the next layer. S.Iofee et.al (30) stated that BN can prevent internal covariate shift which makes the neural network more stable. However, Tianle Cai et.al (31) proposed their findings that BN may not show effective performance on graph classification tasks. In addition, various studies have been arguing about the exact location of the neural network's BN should be placed. Therefore, in the batch normalization experiments, I tried to place BN *before the activation function*, *after the activation function*, *between the two convolutional layers*, and *without batch norm* to investigate whether BN is appropriate to apply in a GNN model. Furthermore, to see which BN architecture obtains the best prediction performance. Since from the definition of BN, BN already has the effect of standardizing the data, I also tried to *remove the data standardization function* I applied in the data standardization experiment (section 5.3) to compare the network's performance. The configuration settings for model training are presented in Table 5.4 and the experimented GNN architectures, which show the exact locations where BNs are placed, are illustrated in Figure 5.4, 5.5, 5.6 and 5.7.

Batch size	**Epoch size**	**Min epoch size**	**Learning rate**
64	40	20	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Standardization**
15	0.06	True	True/False
Optimizer			
Adam			

Table 5.4: Configuration Setting for Batch Normalization Experiment

Experiment Result

Throughout the BN experiments, all training ended successfully, the matching of experiment names and their combinations of BNs and standardization is presented in Table 5.5. We give different BN architectures a short name (BN1 to BN4) to better describe the experiments.

Experiment Name	Batch Norm Placed	Standardization
Without BN	None	True
BN1 Without Standardize	Batch Norm before activation function (Figure 5.4)	False
BN1	Batch Norm before activation function (Figure 5.4)	True
BN2	Batch Norm after activation function (Figure 5.5)	True
BN3	Batch Norm between two convolutional layers (Figure 5.6)	True
BN4	Batch Norm between two convolutional layers adding activation functions (Figure 5.7)	True

Table 5.5: Matching Table for Batch Normalization Experiments

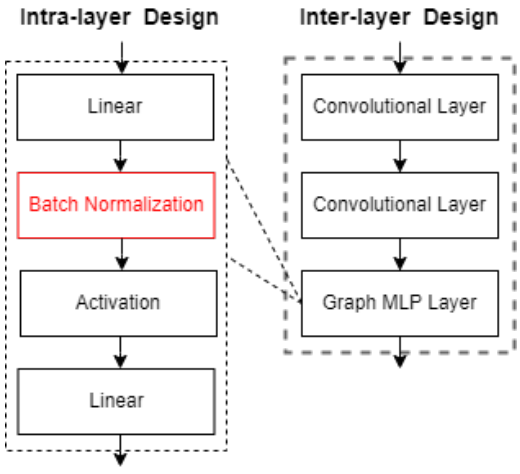

Figure 5.4: GNN Architecture of Batch Norm Method 1 (BN1)

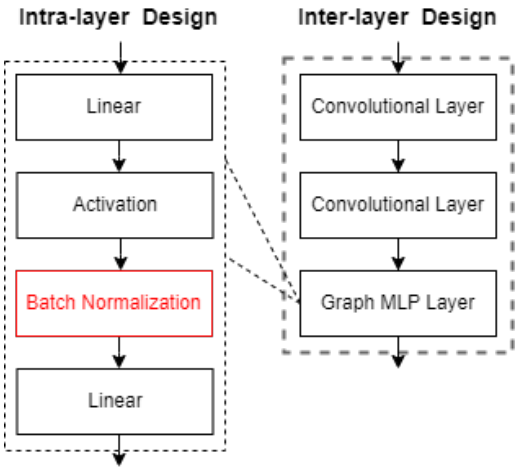

Figure 5.5: GNN Architecture of Batch Norm Method 2 (BN2)

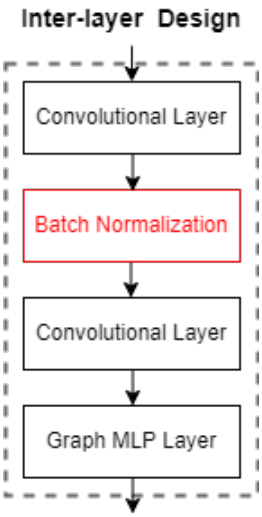

Figure 5.6: GNN Architecture of Batch Norm Method 3 (BN3)

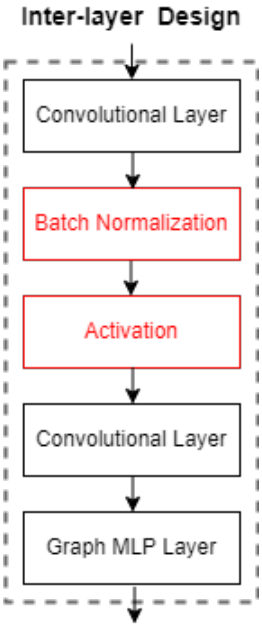

Figure 5.7: GNN Architecture of Batch Norm Method 4 (BN4)

Figure 5.8 showed the AUC score results for running each BN method once. In the beginning, we want to know if we can apply BNs only, without standardizing the network. Therefore, we conducted two experiments for BN1, one with standardization and the other without. The results showed that BN1 with standardization achieved an AUC score of 0.8243 as opposed to 0.6764 for BN1 without standardization, which is nearly a 20% improvement in performance in training. This result demonstrated that standardization is still a crucial technique for the network, and therefore, we will continue to use the standardization method during training for the subsequent BN architectures (BN2 to BN4). Additionally, it is obvious to see only the experiments BN1 with standardization and BN2 obtained an AUC score above 0.8 in comparison to other BN architectures, indicating that other BN architecture does not fit well in the network. Moreover, the learning curves for the BN experiments in Figure 5.9 confirmed that only the experiments BN1 with standardization and BN2 have a stable convergence curve over the duration of training. The training process is ineffective for other BN architectures because of the sharp up-and-down movements in their learning curve.

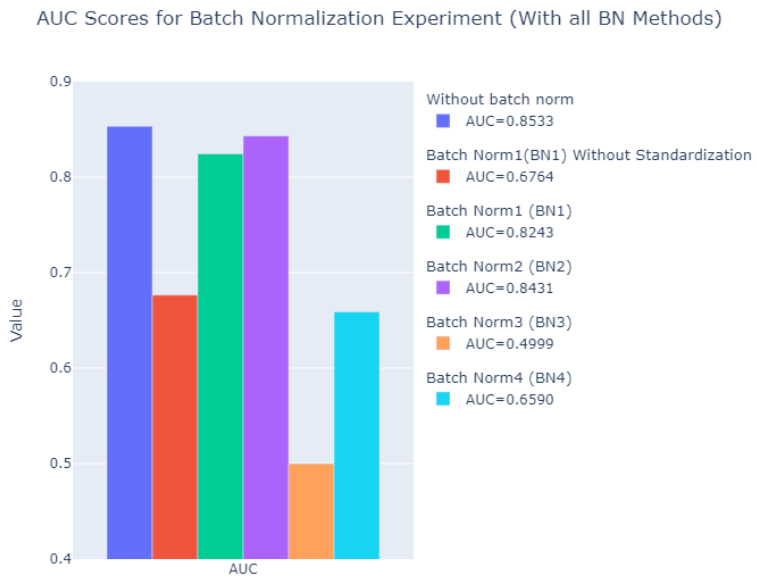

Figure 5.8: AUC Score for all Batch Normalization Methods (Running Once)

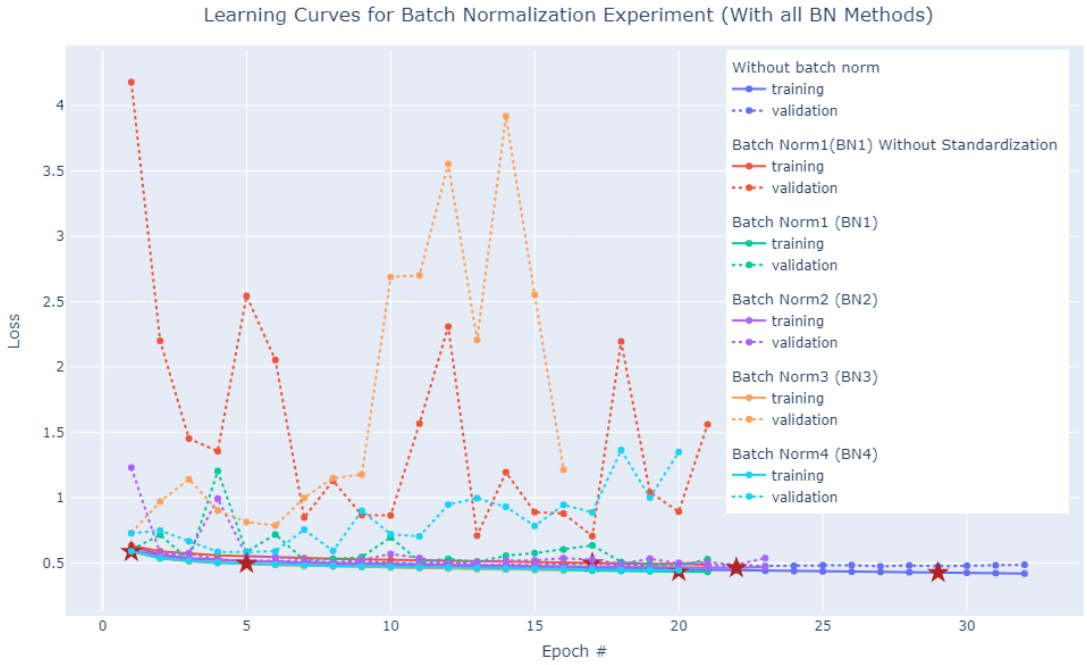

Figure 5.9: Learning Curves for all Batch Normalization Methods (Running Once)

Through the above observation, we know that only the model without applying BN, applying BN1, and applying BN2 has a relatively good performance. To verify which BN methods suit our GNN model the best, we ran each BN method (without BN, BN1, and BN2) five times each using shuffled data sets with random seeds. The average AUC and MCC scores for these three batch normalization result is shown in Table 5.6 and Table 5.7. According to the results, the neural network achieves the best performance in both metrics without the addition of batch normalization. In terms of AUC scores, BN1 and BN2 both lost 1 to 2% of the testing set's prediction accuracy, while MCC scores for BN1 and BN2 were on average 0.52 and 0.53 compared to the original architecture's 0.54.

	AUC Score (Training)	**AUC Score (Validation)**	**AUC Score (Testing)**
Without BN	0.8817 ± 0.0063	0.8588 ± 0.0054	0.8551 ± 0.0044
BN1	0.8741 ± 0.0056	0.8547 ± 0.0035	0.8450 ± 0.0132
BN2	0.8568 ± 0.0070	0.8483 ± 0.0063	0.8476 ± 0.0028

Table 5.6: Average AUC Scores for Batch Normalization Experiments

	MCC Score (Training)	**MCC Score (Validation)**	**MCC Score (Testing)**
Without BN	0.5994 ± 0.0110	0.5587 ± 0.0134	0.5472 ± 0.0078
BN1	0.5873 ± 0.0090	0.5514 ± 0.0051	0.5262 ± 0.0162
BN2	0.5518 ± 0.0139	0.5354 ± 0.0086	0.5395 ± 0.0110

Table 5.7: Average MCC Scores for Batch Normalization Experiments

Experiment Conclusion

In conclusion, we decided to stay on the existing network architecture without the adoption of a batch normalization algorithm, since it results in the best prediction on both performance metrics. The blue bars in Figure 5.10 and Figure 5.11 showed the performance without applying batch normalization, which remains the same as the original GNN model from the Deeprank-core package. However, the experiments showed the importance of applying the standardization function to the data set although BN claims to have some effect on standardizing data. Secondly, the experiments proved the research conducted by Tianle Cai (31), that batch normalization has ineffective performance for graph classification models.

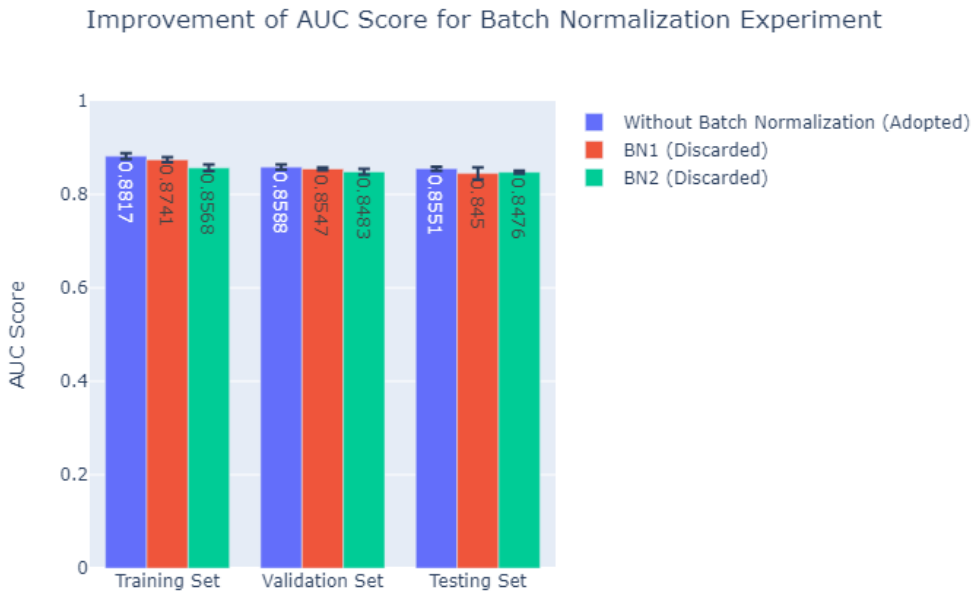

Figure 5.10: Improvement of AUC Score for Batch Normalization Experiment

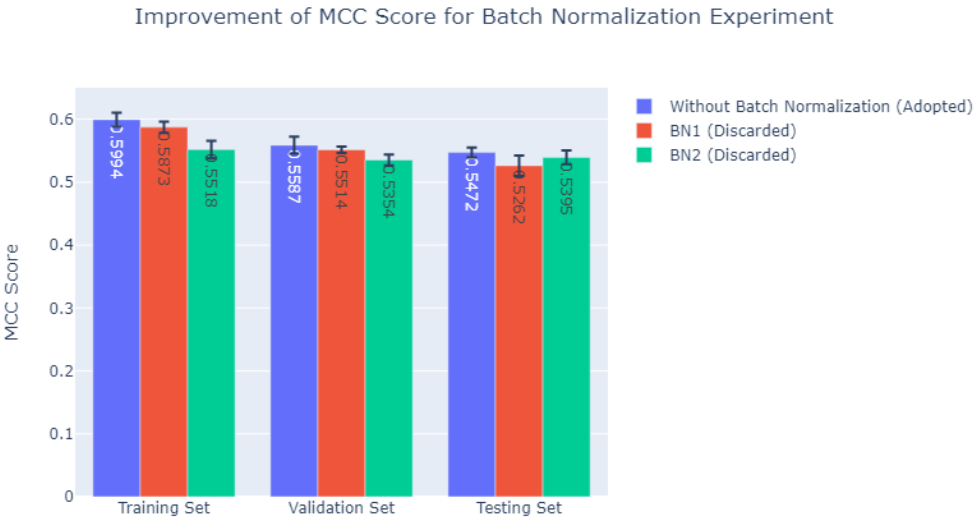

Figure 5.11: Improvement of MCC Score for Batch Normalization Experiment

5.2.2 Expanding Neural Network Experiment

Experiment Description

It has been suggested that the number of convolutional layers is correlated with the complexity of an image recognition problem (IRP) (23). In the case of my study, the IRP is primarily caused by the number of features contained in each PPI data set. The original GNN architecture constructs from two convolutional layers, finding an appropriate convolutional number may improve the performance of pMHC prediction. As a result, I attempted to increase the GNN architecture's dimensions by adding one and two more convolutional layers (Figure 5.12 and Figure 5.13) in an effort to see if it would enhance the overall performance. During the experiments, I realized that my initial epoch size setting of 40 and my minimum epoch size of 20 set for network model training is not adequate due to the growth of the neural network, the learning curves of the network with convolutional layers expanded is not showing convergence with a maximum epoch size of 40 and minimum epoch size of 20. Therefore, I tried the expanded networks on larger epochs and minimum epoch sizes to allow the network to train more to achieve an optimal learning curve. The training configuration settings for the expanding network architecture experiment are presented in Table 5.8.

Batch size	**Epoch size**	**Min epoch size**	**Learning rate**
64	40/60	20/40	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Standardization**
15	0.06	True	True/False
Optimizer			
Adam			

Table 5.8: Configuration Setting for Expanding Neural Network Experiment

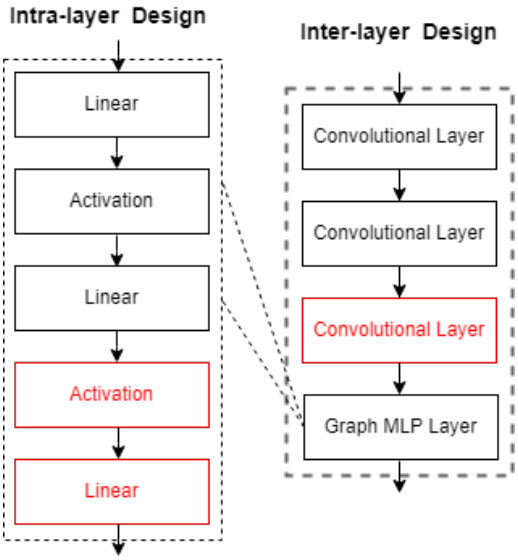

Figure 5.12: GNN Architecture of Expand Neural Network 1 (Expand to Three Neural Network)

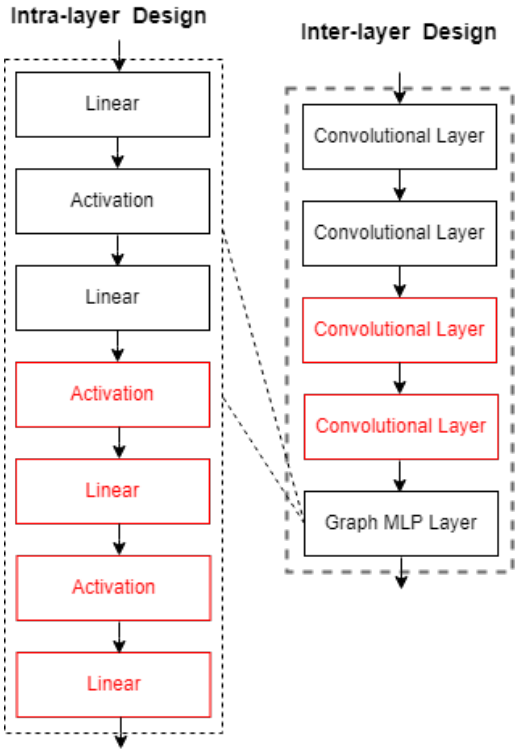

Figure 5.13: GNN Architecture of Expand Neural Network 2(Expand to Four Neural Network)

Experiment Result

Throughout the expansion of neural network experiments, all training ended successfully, the matching of experiment names and their combinations of the number of convolutional layers and epoch sizes settings are shown in Table 5.9. At first, we set the epoch size of all experiments with the original epoch size of 40 and the minimum epoch size of 20. However, we noticed that the learning curves of the experiment have not yet converged. This makes sense because an expanded neural network may require more training to reach a stable convergence result. Therefore, we increased the number of epoch sizes and minimum epoch sizes to 60 and 40 for training the network with more than one convolutional network.

Experiment Name	Convolutional Layers	Epoch Size	Min Epoch
Without Expansion	2 (Original)	40	20
Expand to Two Convolutional Layers	3 (One layer added)	60	40
Expand to Three Convolutional Layers	4 (Two layers added)	60	04

Table 5.9: Matching Table for Expanding Neural Network Experiment

The results of the expanded neural network experiments using AUC and MCC performance metrics are demonstrated in Table 5.10 and Table 5.11. In the testing set, the network architecture with three convolutional layers had an average AUC score of 0.8553 and an average MCC score of 0.5518, which were substantially higher than the original architecture without expansion's 0.8553 and 0.5472 scores, respectively. Additionally, the MCC score for the network model reached an average of 0.6 for the first time in the training set.

	AUC Score (Training)	AUC Score (Validation)	AUC Score (Testing)
Without Expansion	0.8817 ± 0.0063	0.8588 ± 0.0054	0.8551 ± 0.0044
Expand to Three Convolutional Layers	0.8843 ± 0.0046	0.8594 ± 0.0014	0.8553 ± 0.0040
Expand to Four Convolutional Layers	0.8810 ± 0.0134	0.8579 ± 0.0028	0.8529 ± 0.0073

Table 5.10: Average AUC Scores for Neural Network Expansion Experiments

	MCC Score (Training)	MCC Score (Validation)	MCC Score (Testing)
Without Expansion	0.5994 ± 0.0110	0.5587 ± 0.0134	0.5472 ± 0.0078
Expand to Three Convolutional Layers	0.6010 ± 0.0125	0.5562 ± 0.0080	0.5518 ± 0.0053
Expand to Four Convolutional Layers	0.5962 ± 0.0309	0.5570 ± 0.0076	0.5489 ± 0.0116

Table 5.11: Average MCC Scores for Neural Network Expansion Experiments

Experiment Conclusion

In conclusion, I adopted the Expand Neural Network1 architecture in Figure 5.12, where the number of convolution layers is expanded to three, along with a larger epoch size of 60 and a minimum epoch size of 40. From the experiment, I found the appropriate convolutional number and improve the overall performance for our GNN model (Figure 5.14 and Figure 5.15). As a result, in the experiments that follow, the new number of convolutional layers of three and the configuration settings for epoch sizes will replace the original GNN architecture in the project.

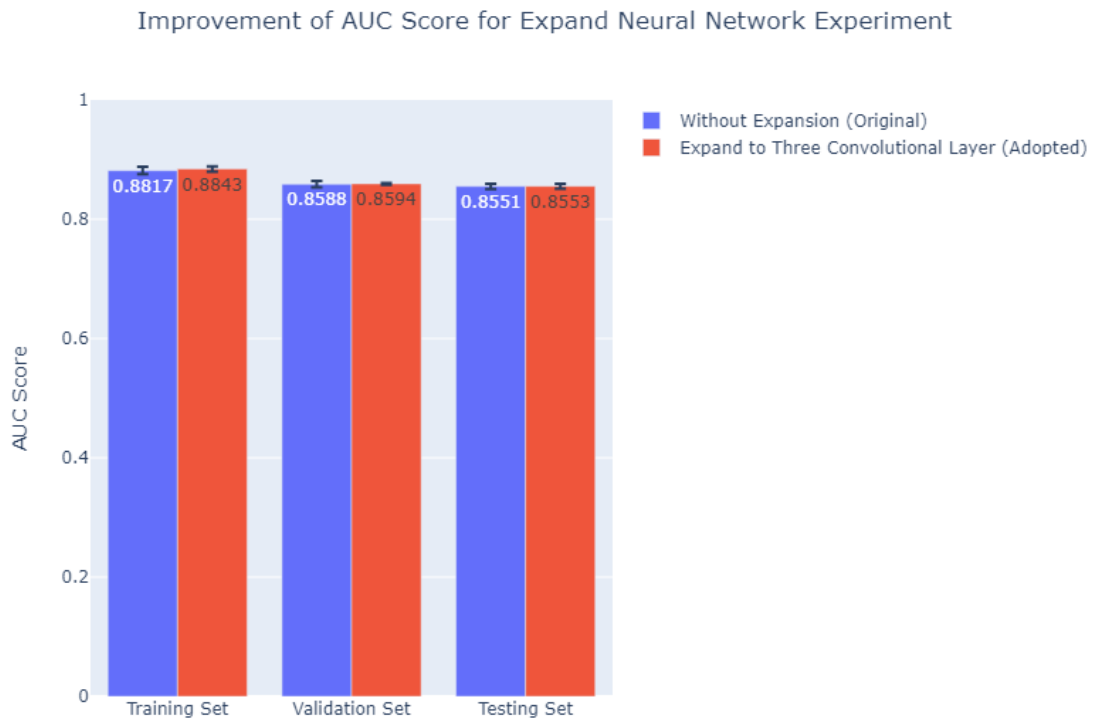

Figure 5.14: Improvement of AUC Score for Expand Neural Network Experiment

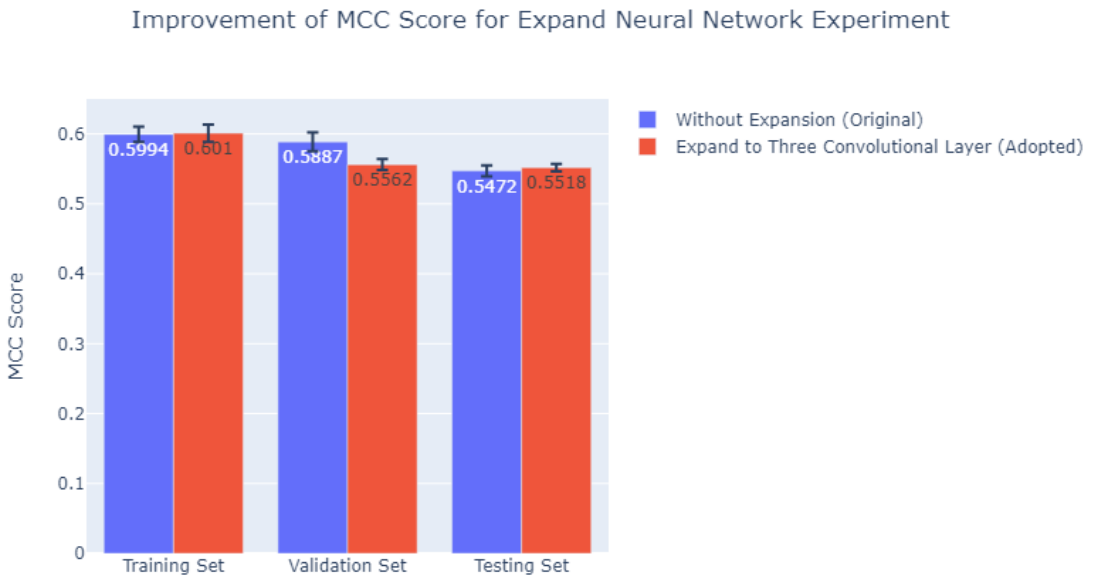

Figure 5.15: Improvement of MCC Score for Expand Neural Network Experiment

5.3 Experiments on Input Data Preprocessing

5.3.1 Data Standardization Experiment

Experiment Description

Data standardization is a common technique used in building machine-learning models, it could improve the model performance by re-scaling the data values and minimizing the influence of out-liners. In the data standardization experiment, a standardization function is applied to the data set, which scales each input feature value by subtracting the mean and dividing by the standard deviation of the feature values (Equation 5.1). Then the outcome of applying data standardization is compared to the one without standardization to investigate whether data standardization could really improve the predicting performance for my GNN model. The configuration setting for the data standardization experiment model training is shown in Table 5.12.

$$Val' = \frac{Val - \mu(Val)}{\sigma(Val)} \tag{5.1}$$

Batch size	Epoch size	Min epoch size	Learning rate
16	40	20	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Standardization**
15	0.06	True	True/False
Optimizer			
Adam			

Table 5.12: Configuration Setting for Data Standardization Experiment

Experiment Result

Throughout the data standardization experiments, all training ended successfully. From Table 5.13 and Table 5.14, we can see the average AUC and MCC scores of applying standardization and without applying standardization to the data set before training based on a five-time-execution with a shuffled data set on random seeds. With data standardization pre-processed, the AUC score in the testing set showed a significant improvement from an average of 0.8285 to 0.08523, which is a nearly 3% growth. The MCC scores, on the other hand, also demonstrated an impressive rise, going from an average of 0.5032 to 0.5442 in the testing set. Additionally, the scores for the training and validation sets have also increased significantly in a similar manner.

	AUC Score (Training)	AUC Score (Validation)	AUC Score (Testing)
Without Standardization	0.8408 ± 0.0038	0.8351 ± 0.0030	0.8285 ± 0.0059
With Standardization	0.8787 ± 0.0031	0.8585 ± 0.0020	0.8523 ± 0.0054

Table 5.13: Average AUC Scores for Data Standardization Experiments

	MCC Score (Training)	MCC Score (Validation)	MCC Score (Testing)
Without Standardization	0.5219 ± 0.0060	0.5134 ± 0.0066	0.5032 ± 0.0085
With Standardization	0.5972 ± 0.0058	0.5571 ± 0.0040	0.5442 ± 0.0056

Table 5.14: Average AUC Scores for Data Standardization Experiments

Experiment Conclusion

In conclusion, we adopted applying standardization in the step of data pre-processing since it greatly increases the prediction performance of the GNN model. An overall comparison of improvement of applying data standardization is demonstrated in Figure 5.16 and Figure 5.17. As this is the first experiment conducted for the study, data standardization will be applied in all subsequent experiments.

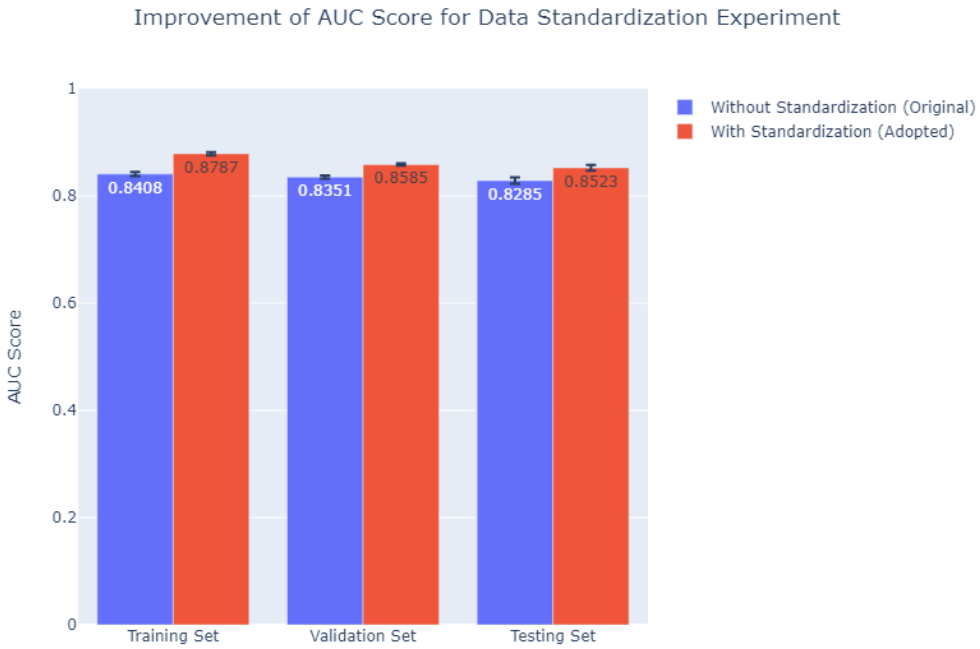

Figure 5.16: Improvement of AUC Score for Data Standardization Experiment

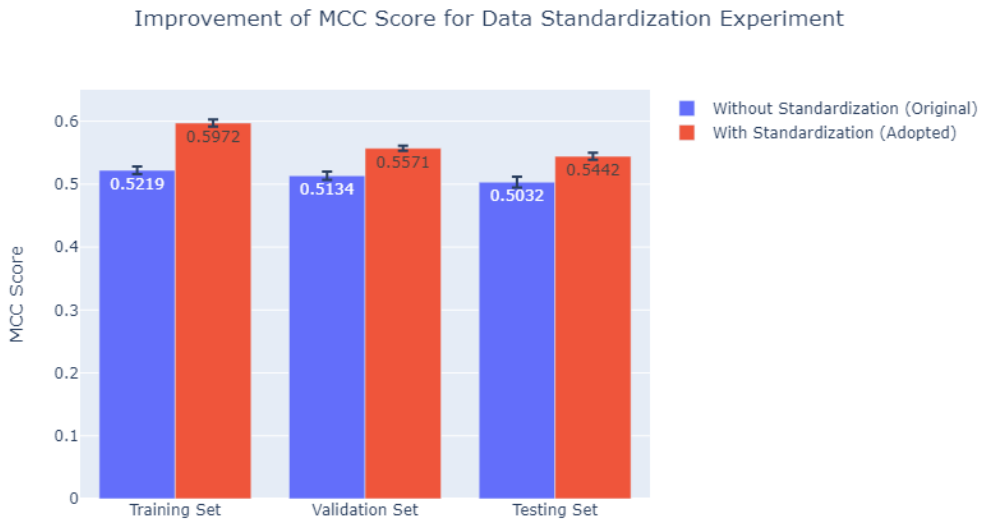

Figure 5.17: Improvement of MCC Score for Data Standardization Experiment

5.3.2 Feature Transformation Experiment

Experiment Description

According to Thang (24), applying data transformation to data before it is inputted into the neural network may expand the spread of data distribution and therefore improve subtly the performance of unsupervised classification. This is because the majority of machine learning applications are based on Gaussian statistics which require data with a Gaussian-like distribution (24), for example, standardization. In my data standardization experiment, a statistical calculation (Equation 5.1) is applied directly to each feature in the data set without considering how the data distribution looks like. The data distribution for some of the features doesn't appear to be normally distributed before standardization. Therefore, the goal of this feature transformation experiment is to identify the transformation approach that will best transform each individual feature's data set to ensure it is more normally distributed before standardization. Figure 7.20 illustrates the data distribution of various transformation methods investigated on the data set, using the feature *sasa* as an example. It is obvious that for feature sasa, the square root transformation helps the data turned most normally distributed. Thus, I will perform a square root transformation on all PPI images that contain the sasa feature.

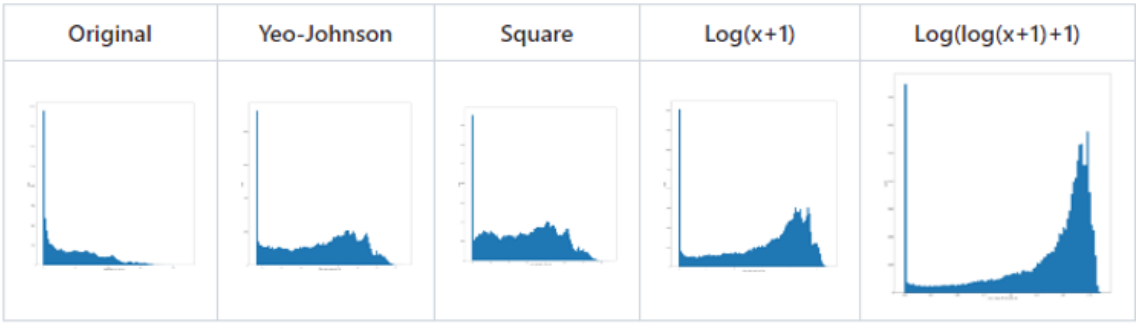

Figure 5.18: Data Distribution after Different Transformation Method for sasa

For other features, just like sasa, I also examined the data set with six transformation methods, which are Yeo-Johnson, square root, log(x), log(x+1), log(log(x+1)+1), and cubic root transformation. The summary of the desired transformation method for each feature is listed in Table 5.15, while the complete data distribution investigation for each feature attempting distinct transformation methods is provided in chapter 7. The experiment will do a comparison of the predicting performance between the model with applying feature transformation in data pre-processing and without applying feature transformation. The configuration settings for the feature transformation experiment are presented in Table 5.16.

Transformation Method	**Features**
Yeo-Johnson	None
Square root	sasa
Cubic root	electrostatic, vanderwaals
Log(x)	None
Log(x+1)	bsa, res_depth
Log(log(x+1)+1)	None
Original (no transformation needed)	res_size, res_charge, hse hb_donors, hb_acceptors, features starts with irc_

Table 5.15: Summarize of Transformation Method for Each Feature

Batch size	Epoch size	Min epoch size	Learning rate
64	60	40	0.001
Early stop patience	**Early stop gap**	**Weight class**	**Standardization**
15	0.06	True	True
Optimizer	**Feature Transformation**		
Adam	True/False		

Table 5.16: Configuration Setting for Feature Transformation Experiment

Experiment Result

The experiments, with and without feature transformation applied before standardizing the data set were conducted five times each shuffled with a random seed. The performance outcome is stated in Table 5.17 and Table 5.18. Comparing the experiments' AUC scores, the scores using feature transformation have a slight drop of around 0.002% in the training and validation set. While in contrast, the testing set shows a slight increase around 0.0002%. The outcome for MCC scores also shows no great difference, with a little fall in the training and validation set when applying feature transform, and a minor increase in the testing set.

	AUC Score (Training)	AUC Score (Validation)	AUC Score (Testing)
Without Feature Transformation	0.8843 ± 0.0046	0.8594 ± 0.0014	0.8553 ± 0.0040
With Feature Transformation	0.8822 ± 0.0043	0.8580 ± 0.0011	0.8555 ± 0.0030

Table 5.17: Average AUC Scores for Feature Transformation Experiments

	MCC Score (Training)	MCC Score (Validation)	MCC Score (Testing)
Without Feature Transformation	0.6010 ± 0.0125	0.5562 ± 0.0080	0.5518 ± 0.0053
With Feature Transformation	0.6007 ± 0.0097	0.5558 ± 0.0057	0.5552 ± 0.0102

Table 5.18: Average MCC Scores for Feature Transformation Experiments

The performance outcome appears to be surprising with no significant changes. However, from the data distribution for each feature, applying transformation indeed normalizes the distribution greatly. Recall from the research questions, the performance of the GNN model on peptide-clustering and allele-clustering is also an important concern in the study since it simulates the real scenario of rare alleles and peptides. Therefore, we decide to conduct another experiment verifying the performance with and without feature transformation on other data set configurations, which are peptide-clustered and allele-clustered instead of using shuffled data only. The comparison is demonstrated in Figure 5.19 and Figure 5.20. From the comparison, we can see that although the prediction accuracy on applying feature transformation based on a shuffled data set did not show much improvement, however, when based on a clustered data set, applying feature transformation presented a significant boost from 0.8340 to 0.8404 in AUC and 0.5004 to 0.5158 in MCC for peptide-clustered. Similarly, the allele-clustered technique also presented a great growth from 0.6579 to 0.6673 in AUC and 0.2411 to 0.2508 in MCC.

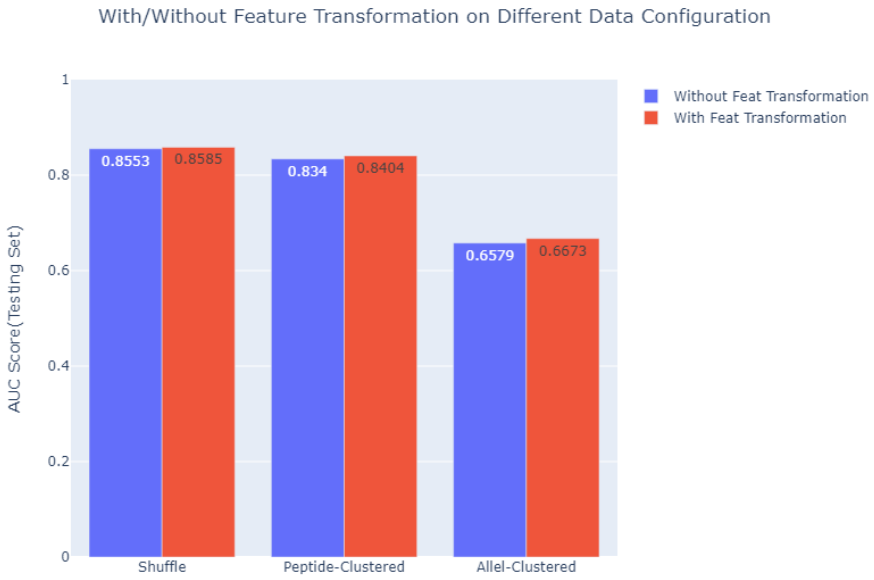

Figure 5.19: AUC Score for Model With and Without Feature Transformation on Different Data Classification

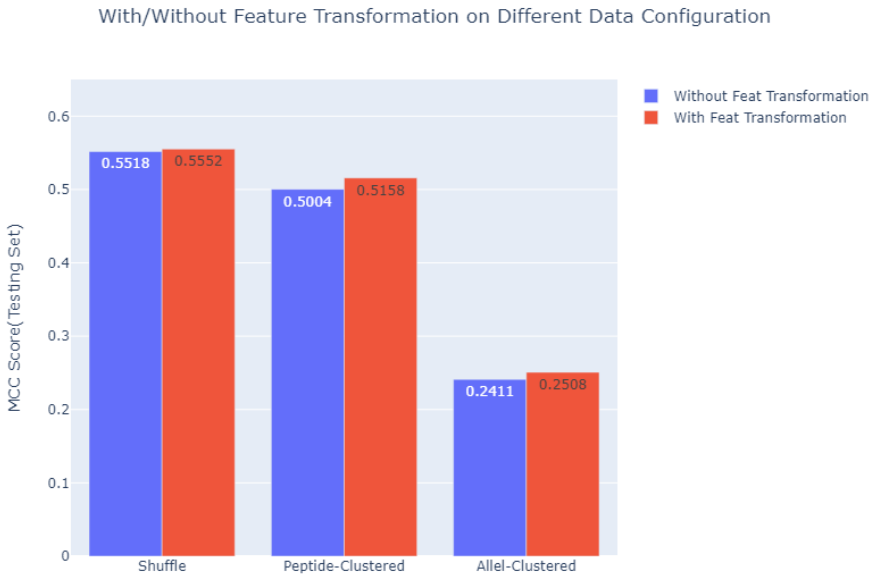

Figure 5.20: AUC Score for Model With and Without Feature Transformation on Different Data Classification

Experiment Conclusion

In conclusion, although applying feature transformation before standardizing in the data pre-processing stage does not show much increment using shuffled data set. However, the boost of prediction accuracy in clustered data sets proved the necessity of adopting feature transformation in the GNN model. Thus, the work will adopt feature transformation as an element in the composition of the best-performed GNN model.

5.4 Experiment on Data Set Configuration

Experiment Description

After exploring experiments on adjusting the configuration settings, modification of GNN architecture, and pre-processing input data, we found the best-performing GNN architecture for the package. This includes applying standardization and transformation on the input data for each feature, increasing the batch size parameter of the model training to 64, and finally expanding two more graph convolutional layers to the existing GNN architecture. However, these experiments were all done based on a shuffled data class with random seeds. To better simulate the real PPI data, Radboud UMC introduced two clustering methods based on the type of alleles and peptides in the data. Because of this, we also want to know how well the best-prediction GNN model, which we tested on shuffled data, performs on allele clustering and peptide clustering. Finally, our work, which focuses on a structure-based model, will then be compared to the retrained MHCflurry 2.0, which employs a sequence-based model based on the same shuffled, allele, and peptide clustered dataset using 10088 PPI data points.

The MHCflurry is retrained based on the same method from the paper (2), where a detailed description of the model training method and script (18) can be found. In brief, the training and validation sets are randomly divided into four folds. Each fold contains three splits for training and one split for validation. Moreover, in each training split, 35 models were trained differently in terms of layers, learning rate, and neurons per layer, in total 140 models are trained in a fold. Among the 35 models, the validation set for each fold is used to select the best model. If the performance of the validation set increases, then the best model will be added to the ensemble. The models will be added until the ensemble accuracy shows no more improvement (2).

Experiment Result

Table 5.19 demonstrated an AUC performance outcome for testing sets with various types of data classifications for our GNN model training. In addition to the known method of

using shuffle data to scramble the data set, which produces an accuracy of up to 0.8585 in the test set. The peptide-clustering technique also attained a relatively high accuracy of 0.8404. The allele-clustered data set came in last with a score of 0.6673. On the other hand, the performance of the sequence-based model MHCflurry, which was retrained on the shuffle, peptide-clustered and allele-clustered does not show acceptable results. With a significant decline in AUC scores of 0.7355, 0.6885, and 0.6062, respectively. The MCC score for the testing sets also showed something similar, for our structured-based GNN model, we achieved 0.5552, 0.5158, and 0.2508 for shuffle, peptide-clustered and allele-clustered. While the results for the retrained sequence-based MHCflurry model are 0.4716, 0.3767, and 0.2354, which is a poor outcome in comparison to our work. The reason why only the prediction outcome from the testing sets are compared for data set configurations is that, for clustering, a totally different allele and peptide data cluster is used to simulate the scenario that the testing set is seen as rare alleles and peptides for the model. Therefore, we are only curious about the outcome of the testing sets which represent the ability of my improved GNN model perform on identifying rare alleles and peptides.

Data Set Classification	**Shuffled**		**Peptide-clustered**		**Allele-clustered**	
Performance Metrics	AUC	MCC	AUC	MCC	AUC	MCC
GNN (Structured-based)	0.8585	0.5552	0.8404	0.5158	0.6673	0.2508
Retrained MHCflurry (Sequenced-based)	0.7355	0.4716	0.6885	0.3767	0.6062	0.2354

Table 5.19: AUC and MCC Score in Testing Set for Data Classification Experiment

Experiment Conclusion

In conclusion, the experiments show that when applying both shuffled, peptide-clustered, and allele-clustered datasets configuration, the improved structured-based GNN model demonstrated an enhancement of 12.3%, 15.2%, and 6.1%, respectively, compared to the sequence-based retrained MHCflurry model trained on the 10088 PPI data. A comparison showing the growth between the retrained MHCflurry and the improved GNN model in the study is shown in Figure 5.21 and Figure 5.22.

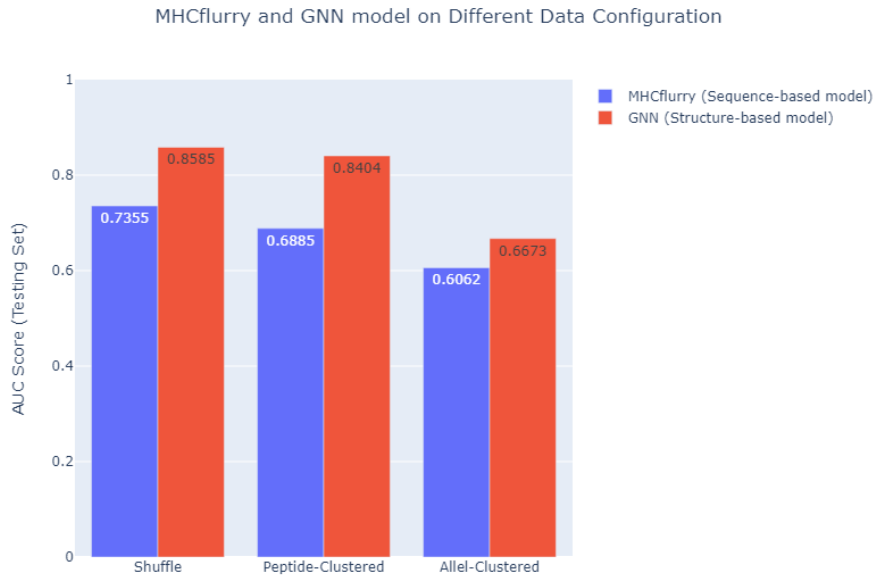

Figure 5.21: Improvement of AUC Score for Data Classification Experiment

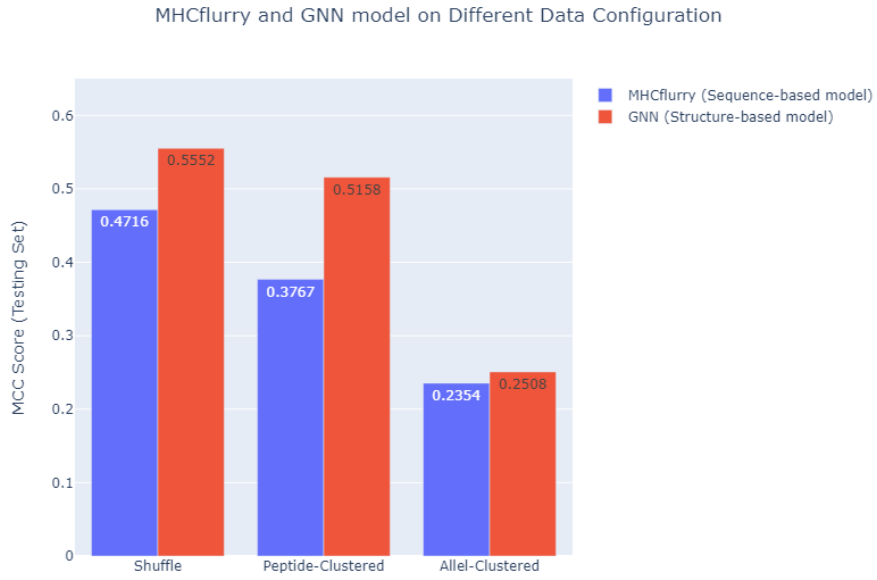

Figure 5.22: Improvement of MCC Score for Data Classification Experiment

6

Discussion

The research's findings have shed light on how a GNN model's performance can be improved and the performance differences between a structure-based model and a sequence-based model. However, a detailed explanation of the outcome should be provided due to the limitations of the current work. In this section, we will first provide a brief recap of our key results while answering the research questions of the work. Additionally, the limitations of the work's experiments are described. Finally, several suggestions are made on how future research can be improved and carried out.

6.1 Results for Research Questions

In the study, (1) we experimented with and adopted various types of machine learning, and data pre-processing methods in improving the prediction accuracy for the pMHC binding affinity prediction GNN model. From the best GNN (structured-based) model, (2) we further experimented on how well it performed on shuffled and clustered data set configurations. Finally, (3) we compared our results with the sequence-based MHCflurry model retrained by the same data classification methods and PPI data set to answer the sub-research questions of the paper:

> *RQ1.1: How much the performance of an improved structure-based model can reach than a sequence-based model on pMHC prediction based on a shuffled data set configuration?*

The result indicates that in shuffled data, our structured-based model performed 12% better in predicting the binding affinity between PPIs than the sequenced-based model, MHCflurry.

RQ1.2: How much the performance of an improved structure-based model can reach than a sequence-based model on pMHC prediction based on clustered(allele and peptides) data set configuration?

In both peptide-clustered and allele-clustered data, the improved structured-based model of the work performed 16% and 6% better in predicting the binding affinity between PPIs than the retrained sequenced-based model, MHCflurry, using the same 10088 PPI data points to train.

As a result, we can provide a conclusion to our main research question:

RQ1: Can a structure-based model perform better than a sequence-based model on pMHC prediction?

The study has demonstrated that our improved structure-based GNN model has a significant increment in prediction from both AUC and MCC performance metrics using different data set configurations compared to the retrained sequence-based model, MHCflurry, using the 100088 PPI data set generated from Radboud UMC. This shows by adopting a structured-based GNN model, we can obtain more genetic information and better predict the binding affinity between PPIs from its 3D structure rather than using only a sequence of genetic code. The original trained MHCflurry model conducted by O'Donnell (17) achieved an AUC score up to 0.9. The lack of data configuration techniques in their model and the common data sets they used which do not take rare alleles and peptides into account, however, could lead to an overestimated prediction accuracy. By employing the clustered data set configuration and generating more PPI data from the 3D structure of rare alleles and peptides, which more accurately simulates the real-life scenario, the performance of the neural network model will become more reliable.

6.2 Limitations

The limitations of the work are mainly related to the conducted experiments and are listed in the following points. The cause of these limitations is due to the lack of time and limited resources on the supercomputer, Snellius. Since there contains 100 thousand PPI data in our work, the time and resources required for training are considerable. Thus, we tried to enhance the accuracy of our experiments as much as possible in the limited time available under the selection of experiments.

- **No grid search for finding best-prediction model**
  During our work, we only experimented with finding the best-performance model without a grid search, which only experimented with and adopted a solution for each hyper-parameter tuning. For example, for the batch size experiments, only the different batch size parameters were experimented with and adopted according to the best one performed due to the time and resource limitation on Snellius.

- **Best GNN model only for the shuffled data set.**
  Every experiment operated in the study is on finding the best-performing GNN model based on a set of shuffled data along with given random seeds. However, it is unsure that the best-performed GNN model on a shuffled data set is also applicable when it comes to the use of an allele and peptide-clustered data set. The reason why the experiments are not tested on the clustered data set is also due to the limited time and resources on Snellius.

- **Limited amount and type of data used for training.**
  The study applied 10088 PPI binding affinity data generated from the 3D structure of mutation proteins and MHC proteins for model training. However, when compared to MHCflurry 2.0, which uses 493,473 MS (Mass Spectrometry) data and 219596 binding affinity data for model training, the amount of data in the study has a lot of room for improvement. Another limitation is, the study focuses on training the binding affinity data only, while MHCflurry also adopts MS data into the dataset for pMHC binding affinity prediction.

6.3 Future Research

Based on the above work limitations, the following points can be summarized for future research.

- **More hyper-parameter tuning and performing a grid search.**
  According to the limitation, the study experimented with finding the best-performance model individually for each hyper-parameter tuning (batch size, number of convolutional layers, and batch normalization) and input data pre-processing experiments (data standardization and transformation) on a shuffled data set only. For future research, more hyper-parameter tuning techniques can be experimented on the GNN

model to improve the overall prediction accuracy. For instance, applying layer normalization and drop-out layer to the network. Moreover, a grid search can be performed, which tries every possible combination of hyper-parameters and data preprocessing techniques to find the best-prediction GNN model. Grid search will not only be experimented on the shuffled data set only but also based on allele and peptide-clustered data sets to analyze the best-prediction GNN model for every data set configuration.

- **More data for model training.**
  According to the limitation on the amount and type of data, the study uses 10088 PPI binding affinity for model training. For future research, we can generate more PPI data regarding the binding affinity information from the 3D structure of pMHCs. In addition to binding affinity data, the MS-based method for measuring peptide quantification is becoming increasingly popular (38). The generated MS data contains pMHC-related information and can be used to predict pMHC binding affinity. Therefore, future research may consider including MS data in the data set for model training.

- **Cross-validation**
  Future research can apply cross-validation methods to verify whether the GNN model of the study is capable of generalizing new data and will not cause over-fitting results. K-fold cross-validation is a common approach for performing cross-validation in which the data set is divided into k small subsets and one of the subsets is used as the testing set (39). This ensures that for each training process, the subset selected for testing would be different and will not be included in the training and validation set (40). In this way, the experiment results for obtaining the best-predicting GNN model in the study will become more reliable.

- **New GNN Architecture.**
  Future research can experiment with an entirely new network architecture suitable for GNN. Scientific papers (41)(42) on researching new GNN architecture have become highly prevalent in the last few years. For example, the introduction of the E(n)-

Equivariant Graph Neural Networks (EGNNs), offers an efficient GNN model by eliminating higher-order representations in intermediate layers of the network (41).

7

Conclusion

The study aims to identify an effective GNN model for predicting the binding affinity between cancer mutations and MHC proteins in order to construct personalized cancer vaccines for patients suffering from end-stage cancers.

The study started from a simple GNN model from the Deeprank-core package (19) and experimented with techniques for modifying the GNN architecture, adjusting the training configuration settings, and pre-processing the input data to search for the best-performed GNN model. In addition, the study trained the best-performed GNN model obtained from the experiments on shuffled, peptide-clustered and allele-clustered data to simulate the distribution of PPI data in real scenarios and in the meanwhile ensures the results are not overestimated.

The results show my improved GNN model has demonstrated a 12.3%, 15.2%, and 6.1% of improvement in prediction performance on a shuffled, peptide-clustered, and allele-clustered dataset, respectively, over the retrained MHCflurry. The results have shown that the study's improved GNN structure-based model outperforms a retrained MHCflurry sequence-based model on both shuffled and clustered data with the 10088 PPI dataset generated from the 3D structure of pMHCs.

However, due to the limited time and resources in operating the experiments, the performance results of our current research may contain slightly erroneous which may affect the decision of adopting the best-performed GNN model. In spite of the limitations, the improved structure-based GNN model developed in the work has already presented a satisfactory prediction outcome over the sequence-based retrained MHCflurry model. In further research, the study will focus on overcoming the limitations by experimenting with more hyper-parameter tuning techniques, conducting a grid search on the combination of

hyper-parameters, applying cross-validation, and generating more data in model training to further improve the prediction performance.

References

[1] Jyothi Thundimadathil. **Cancer Treatment Using Peptides: Current Therapies and Future Prospects**. *Journal of amino acids*, **2012**:967347, 12 2012. 1

[2] Laserson U O'Donnell TJ, Rubinsteyn A. **MHCflurry 2.0: Improved Pan-Allele Prediction of MHC Class I-Presented Peptides by Incorporating Antigen Processing**. *Cell Systems*, **11**(1):42–48.e7, 2020. 1, 5, 9, 18, 44

[3] Barbara Bravi, Jérôme Tubiana, Simona Cocco, Rémi Monasson, Thierry Mora, and Aleksandra M. Walczak. **Flexible machine learning prediction of antigen presentation for rare and common HLA-I alleles**. *bioRxiv*, 2020. 1

[4] **Deeprank-core**. `https://github.com/DeepRank/deeprank-core`. 2, 9, 10, 60

[5] Lucero M. Cato C. Chang L. Geiger J. Henry D. Hernandez J. Hung F. Kaur P. Teskey G. Tran A. Koury, J. **Immunotherapies: Exploiting the Immune System for Cancer Treatment.** *Journal of immunology research*, **2018**:1–16, 2018. 4

[6] Panetta J. Kirschner, D. **Modeling immunotherapy of the tumor – immune interaction**. *Journal of Mathematical Biology*, **37**:235–252, 1998. 4

[7] Shuzhen Tan, Dongpei Li, and Xiao Zhu. **Cancer immunotherapy: Pros, cons and beyond**. *Biomedicine Pharmacotherapy*, **124**:109821, 2020. 4

[8] Sharma P. K. Peter Goedegebuure S. Gillanders W. E. Zhang, X. **Personalized cancer vaccines: Targeting the cancer mutanome.** *Vaccine*, **35**:1094–1100, 2017. 4

[9] **major histocompatibility complex**. `https://www.britannica.com/science/major-histocompatibility-complex`, 2023. 4, 59

[10] Rocco Meli, Garrett M. Morris, and Philip C. Biggin. **Scoring Functions for Protein-Ligand Binding Affinity Prediction Using Structure-based Deep Learning: A Review**. *Frontiers in Bioinformatics*, **2**, 2022. 5

[11] Georgievska S Ambrosetti F Ridder L Marzella DF Réau MF Bonvin AMJJ Xue LC. Renaud N, Geng C. **DeepRank: a deep learning framework for data mining 3D protein-protein interfaces.** *Nat Commun*, **12**(1):7068, 2021. 5, 10

[12] Peng Wang, John Sidney, Courtney Dow, Bianca Mothé, Alessandro Sette, and Bjoern Peters. **A Systematic Assessment of MHC Class II Peptide Binding Predictions and Evaluation of a Consensus Approach**. *PLOS Computational Biology*, **4**(4):1–10, 04 2008. 5

[13] Parizi F. M. van Tilborg D. Renaud N. Sybrandi D. Buzatu R. Rademaker D. T. 't Hoen P. A. C. Xue L. C. Marzella, D. F. **PANDORA: A Fast, Anchor-Restrained Modelling Protocol for Peptide: MHC Complexes.** *Frontiers in immunology*, **13**:878762, 2022. 5

[14] Xiaoxiao Li, Yuan Zhou, Nicha Dvornek, Muhan Zhang, Siyuan Gao, Juntang Zhuang, Dustin Scheinost, Lawrence H. Staib, Pamela Ventola, and James S. Duncan. **BrainGNN: Interpretable Brain Graph Neural Network for fMRI Analysis**. *Medical Image Analysis*, **74**:102233, 2021. 7

[15] **Graph Neural Network for Brain Network Analysis**. `https://github.com/xxlya/BrainGNN_Pytorch`. 7

[16] Xiaoling Lin, Hongzhang Wang, Jing Guo, and Gang Mei. **A Deep Learning Approach Using Graph Neural Networks for Anomaly Detection in Air Quality Data Considering Spatiotemporal Correlations**. *IEEE Access*, **10**:94074–94088, 2022. 8

[17] Bonsack M Riemer AB Laserson U Hammerbacher J O'Donnell TJ, Rubinsteyn A. **MHCflurry: Open-Source Class I MHC Binding Affinity Prediction**. *Cell Systems*, **7**(1):129–132.e4, 2018. 8, 14, 48

[18] **MHCFlurry**. `https://github.com/openvax/mhcflurry`. 9, 44

[19] **DeepRank**. `https://github.com/DeepRank/deeprank`. 10, 13, 16, 52

[20] **Pytorch**. https://pytorch.org/. 10

[21] **Pytorch Geometric**. https://pytorch-geometric.readthedocs.io/en/latest/. 10

[22] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. **Don't decay the learning rate, increase the batch size**. *arXiv preprint arXiv:1711.00489*, 2017. 10, 21

[23] Vadim Romanuke. **An attempt of finding an appropriate number of convolutional layers in cnns based on benchmarks of heterogeneous datasets**. *Electrical, Control and Communication Engineering*, **14**:51–57, 07 2018. 11, 17, 32

[24] Thang N. Ha, David Lubo-Robles, Kurt J. Marfurt, and Bradley C. Wallet. **An in-depth analysis of logarithmic data transformation and per-class normalization in machine learning: Application to unsupervised classification of a turbidite system in the Canterbury Basin, New Zealand, and supervised classification of salt in the Eugene Island minibasin, Gulf of Mexico**. *Interpretation*, **9**(3):T685–T710, 06 2021. 11, 12, 39

[25] Fatemeh Mostofi, Vedat Toğan, and Hasan Başağa. **House price prediction: A data-centric aspect approach on performance of combined principal component analysis with deep neural network model**. *Journal of Construction Engineering, Management Innovation*, **4**:106–116, 06 2021. 12

[26] Agnese Marchesi, Alessandro Bria, Claudio Marrocco, Mario Molinara, Jan-Jurre Mordang, Francesco Tortorella, and Nico Karssemeijer. **The Effect of Mammogram Preprocessing on Microcalcification Detection with Convolutional Neural Networks**. In *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*, pages 207–212, 2017. 12

[27] **COMPUTATIONAL SYSTEMS BIOLOGY (COSBI) GROUP PAGE**. http://home.ku.edu.tr/~okeskin/research.htm. 14

[28] **Why One-Hot Encode Data in Machine Learning?** 14

[29] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh V. Chawla. **NOSMOG: Learning Noise-robust and Structure-aware MLPs on Graphs**, 2023. 15

[30] Sergey Ioffe and Christian Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**, 2015. 17, 26

[31] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. **GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training**, 2021. 17, 26, 30

[32] Sarang Narkhede. **Understanding auc-roc curve**. *Towards Data Science*, **26**(1):220–227, 2018. 17

[33] Davide Chicco and Giuseppe Jurman. **The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation**. *BMC Genomics*, **21**, 01 2020. 18

[34] Jiaxuan You, Rex Ying, and Jure Leskovec. **Design Space for Graph Neural Networks**, 2021. 18

[35] **Scoring matrix - Bioinformatics.Org Wiki**. 19

[36] Massimo Andreatta, Bruno Alvarez, and Morten Nielsen. **GibbsCluster: Unsupervised clustering and alignment of peptide sequences**. *Nucleic acids research*, **45**, 04 2017. 20

[37] **SURF**. `https://www.surf.nl/en`. 20

[38] L.E. Stopfer, A.D. D'Souza, and F.M. White. **1,2,3, MHC: a review of mass-spectrometry-based immunopeptidomics methods for relative and absolute quantification of pMHCs**. *Immuno-Oncology and Technology*, **11**:100042, 2021. 50

[39] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009. 50

[40] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. **A Fair Comparison of Graph Neural Networks for Graph Classification**, 2022. 50

[41] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. **E(n) Equivariant Graph Neural Networks**, 2022. 50, 51

[42] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. **Auto-GNN: Neural Architecture Search of Graph Neural Networks**, 2019. 50

[43] **allele**. https://www.nature.com/scitable/definition/allele-48/, 2014. 59

[44] **What is Mass Spectrometry?** https://www.broadinstitute.org/technology-areas/what-mass-spectrometry, 2023. 59

[45] Skolnick J. Tonddast-Navaei, S. **Are protein-protein interfaces special regions on a protein's surface?** *The Journal of chemical physics*, **143**(24), 2015. 59

[46] **Neoantigen**. https://www.cancer.gov/publications/dictionaries/cancer-terms/def/neoantigen, 2023. 59

Appendix

Terminology Table

Term	Definition
Allele	An allele is a variant of the same nucleotide sequence at the same location on a long DNA molecule (43).
Antigen Process	An immune process that expresses the trigger of immune cells.
Binding Affinity	The strength of binding interactions between two molecules.
Immunotherapy	Immunotherapy is a type of therapy that treats cancer by activating self-immune system.
Mass Spectrometry	An analytical approach for evaluating the mass-to-charge ratio (m/z) of one or more molecules within a sample(44).
Peptide MHC	The peptide-major histocompatibility complex (pMHC) is a group of closely related genes on vertebrate DNA that code for cell surface proteins required by the immune system (9).
PPI	Protein-protein Interactions(PPI) is the area of a protein's surface where it interacts with another(45).
Neoantigen	Neoantigen is a protein that forms on cancer cells when tumor mutations occur, and which can aid the body's immune defense against cancer cells (46).

Contribution on Deeprank-core Package

The full experiment description, discussion, and code for the improved GNN architecture are implemented on the 3D Vac repository, which is an private repository for experimenting with issues in the Deeprank-core package. The list of the issues I contributed to can be seen in the repository.

Besides the experiments on improving the GNN architecture on the deep rank-core package (4). I have contributed to seventeen issues and pull requests with more than 4000 code lines on developing the package. The list of issues and pull requests can be seen on the deep rank-core repository.

Feature Transformation

For features whose data distributions do not look normal-distributed, transformations are required during the input data pre-processing stage. In this section, data distributions utilizing different transformation methods for each node and edge feature in the study will be shown. Note that for features containing different value types, the transformation methods applied vary based on the following criteria:

- **Features containing values $>$ 0 (No zero and negative values)**
  Transformation methods applied: *Log(x), Log(x+1), Yeo-Johnson*, and *Square root* transformation.

- **Features containing values $>=$ 0 (Contain zero values but no negative values)**
  Transformation methods applied: *Log(x+1), Yeo-Johnson*, and *Square root* transformation.

- **Features containing values $><=$ 0 (Contain zero values, positive and negative values)**
  Transformation methods applied: *Cube root* and *Yeo-Johnson* transformation.

electrostatic

For feature electrostatic, applying *cube root transformation* makes the data distribution most normal-distributed.

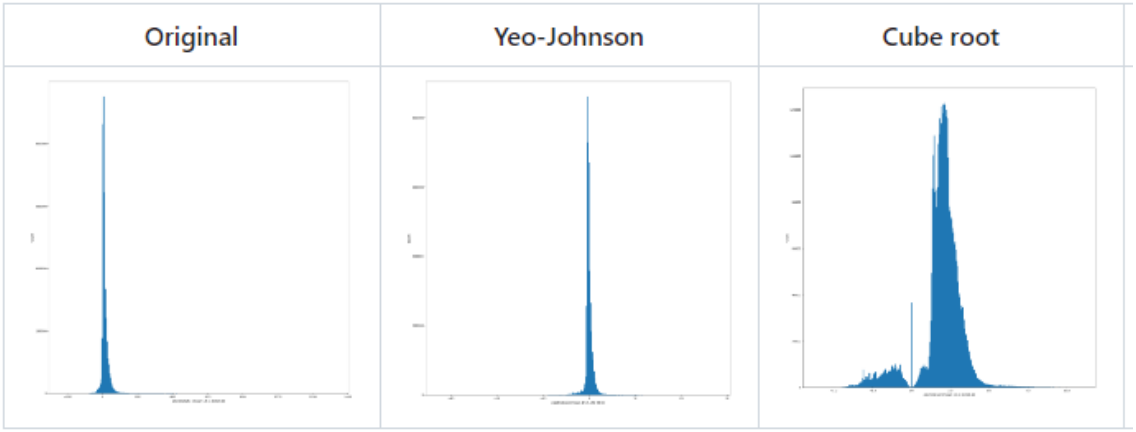

Figure 7.1: Data Distribution for electrostatic

bsa

For feature bsa, applying *log(x+1) transformation* makes the data distribution most normal-distributed.

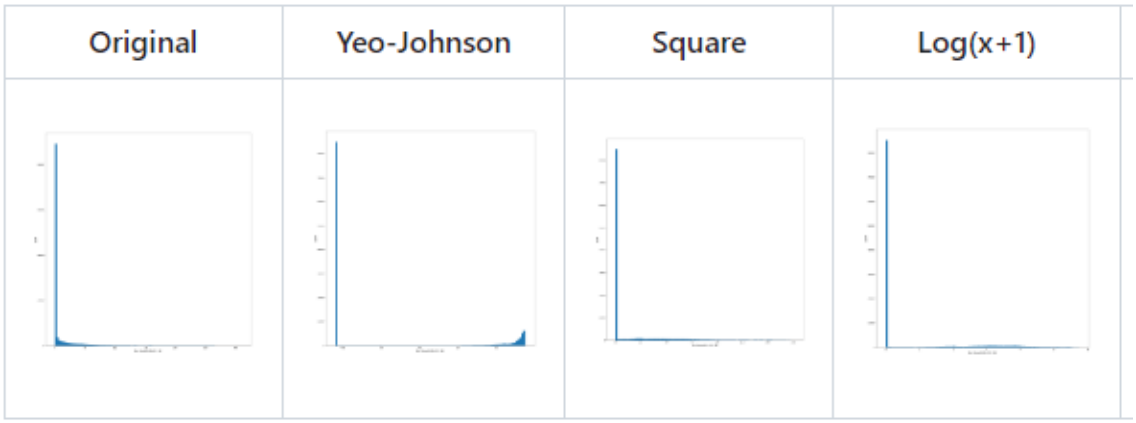

Figure 7.2: Data Distribution for bsa

hb_acceptors

For feature hb_acceptors, *no transformation* is needed since the original data distribution already looks normal-distributed.

Figure 7.3: Data Distribution for hb_acceptors

hb_donors

For feature hb_donors, *no transformation* is needed since the original data distribution already looks normal-distributed.

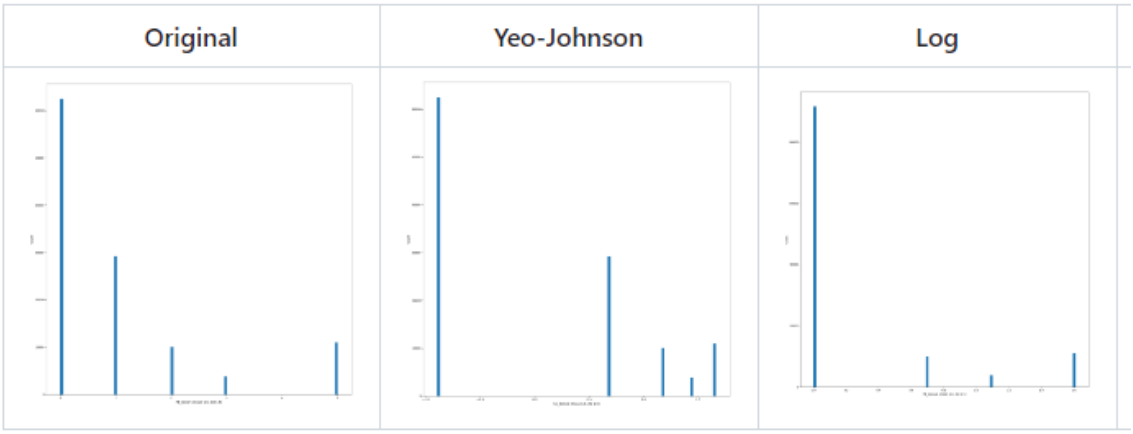

Figure 7.4: Data Distribution for hb_donors

hse

For feature hse, *no transformation* is needed since the original data distribution already looks normal-distributed.

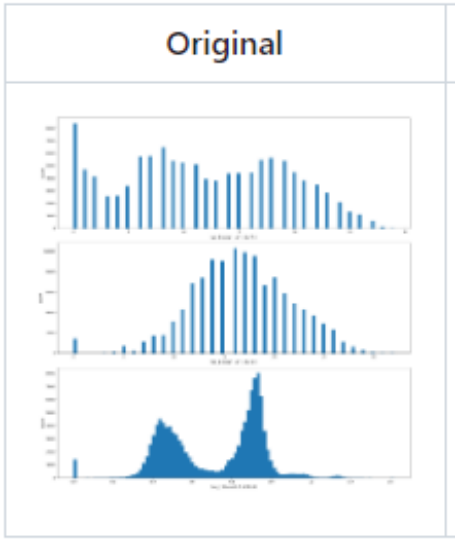

Figure 7.5: Data Distribution for hse

irc_negative_negative

For feature irc_negative_negative, *no transformation* is needed since the original data distribution already looks normal-distributed.

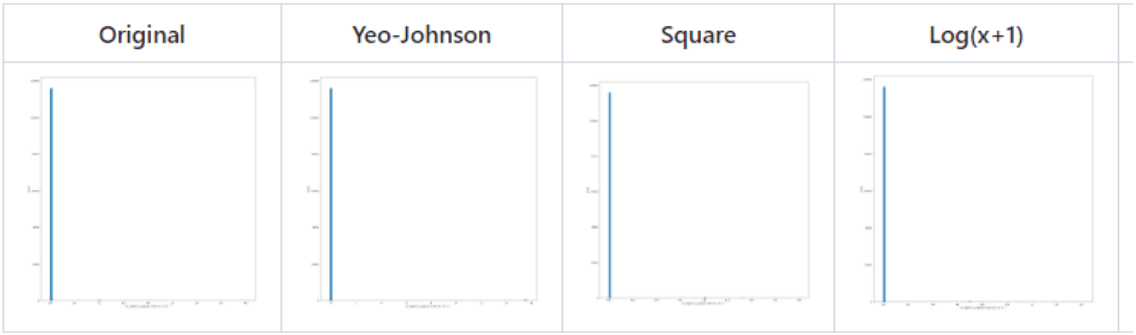

Figure 7.6: Data Distribution for irc_negative_negative

irc_negative_positive

For feature irc_negative_positive, *no transformation* is needed since the original data distribution already looks normal-distributed.

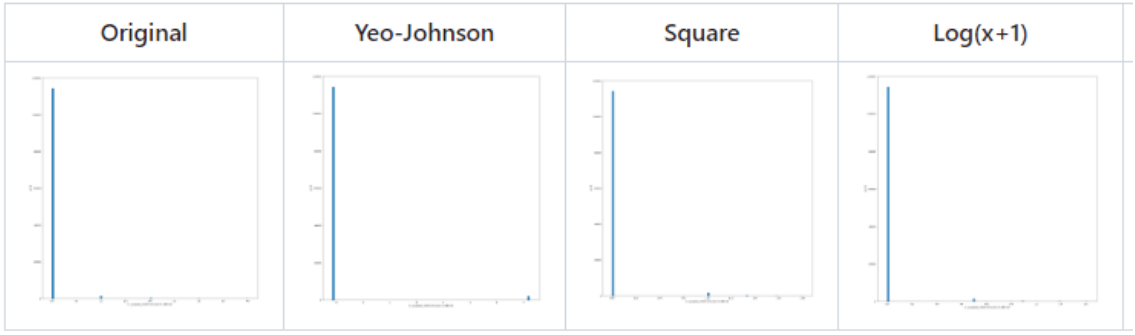

Figure 7.7: Data Distribution for irc_negative_positive

irc_nonpolar_negative

For feature irc_nonpolar_negative, *no transformation* is needed since the original data distribution already looks normal-distributed.

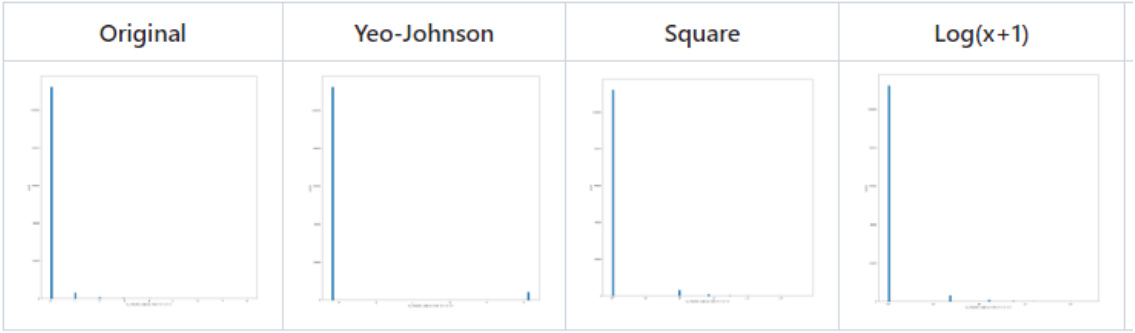

Figure 7.8: Data Distribution for irc_nonpolar_negative

irc_nonpolar_nonpolar

For feature irc_nonpolar_nonpolar, *no transformation* is needed since the original data distribution already looks normal-distributed.

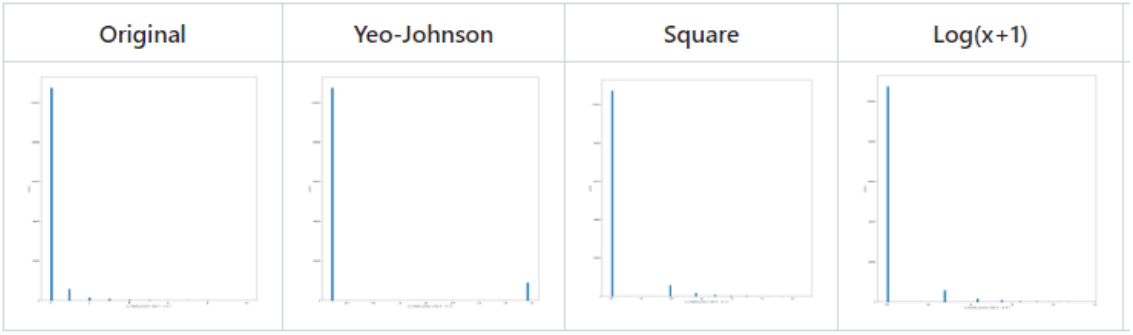

Figure 7.9: Data Distribution for irc_nonpolar_nonpolar

irc_nonpolar_polar

For feature irc_nonpolar_polar, *no transformation* is needed since the original data distribution already looks normal-distributed.

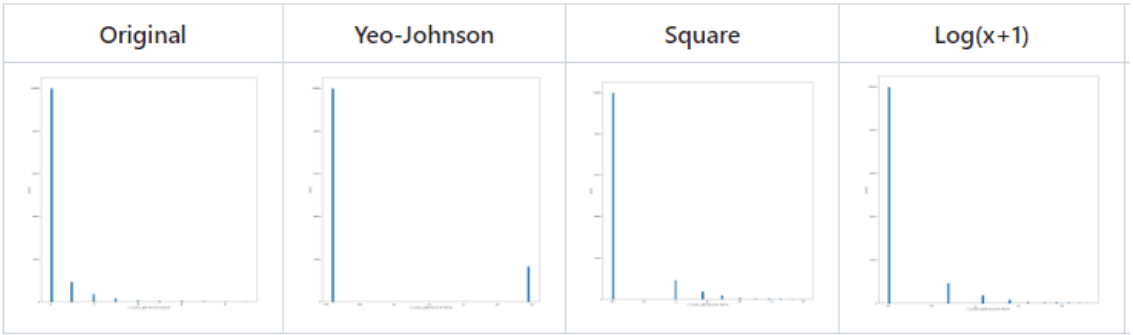

Figure 7.10: Data Distribution for irc_nonpolar_polar

irc_nonpolar_positive

For feature irc_nonpolar_positive, *no transformation* is needed since the original data distribution already looks normal-distributed.

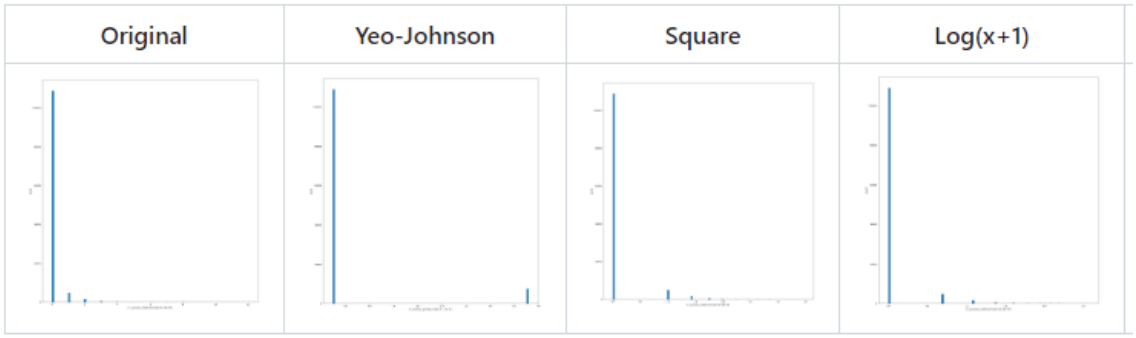

Figure 7.11: Data Distribution for irc_nonpolar_positive

irc_polar_negative

For feature irc_polar_negative, *no transformation* is needed since the original data distribution already looks normal-distributed.

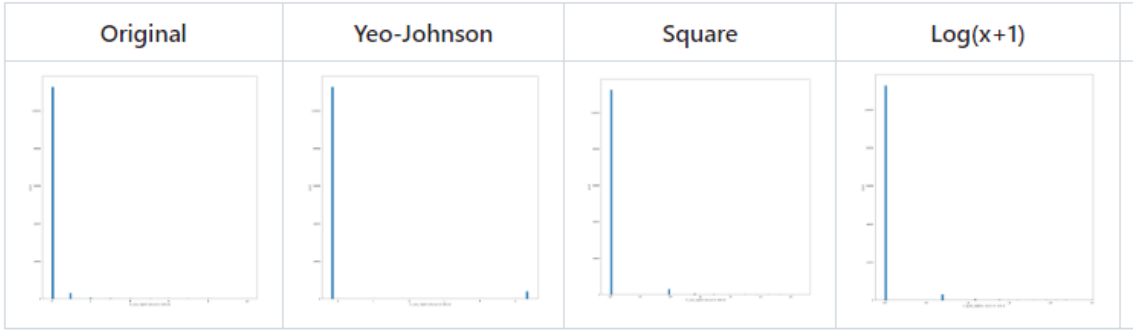

Figure 7.12: Data Distribution for irc_polar_negative

irc_polar_polar

For feature irc_polar_polar, *no transformation* is needed since the original data distribution already looks normal-distributed.

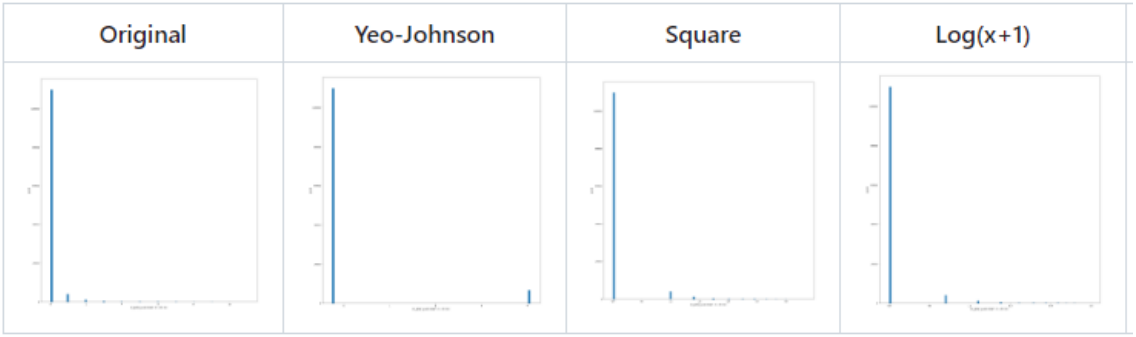

Figure 7.13: Data Distribution for irc_polar_polar

irc_polar_positive

For feature irc_polar_positive, *no transformation* is needed since the original data distribution already looks normal-distributed.

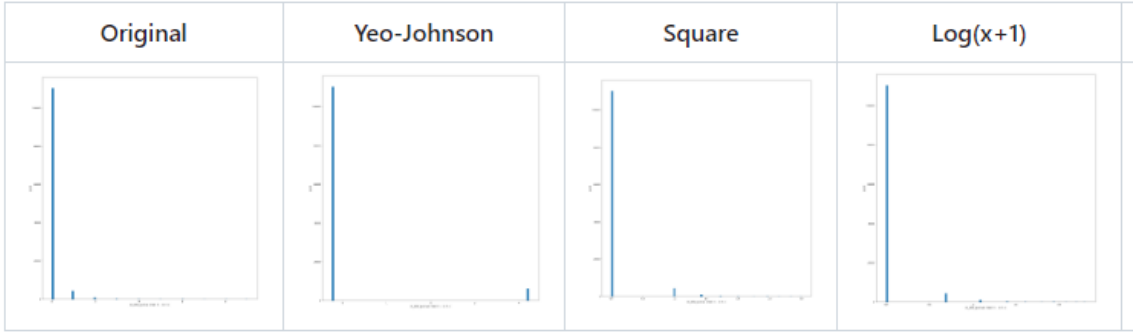

Figure 7.14: Data Distribution for irc_polar_positive

irc_positive_positive

For feature irc_positive_positive, *no transformation* is needed since the original data distribution already looks normal-distributed.

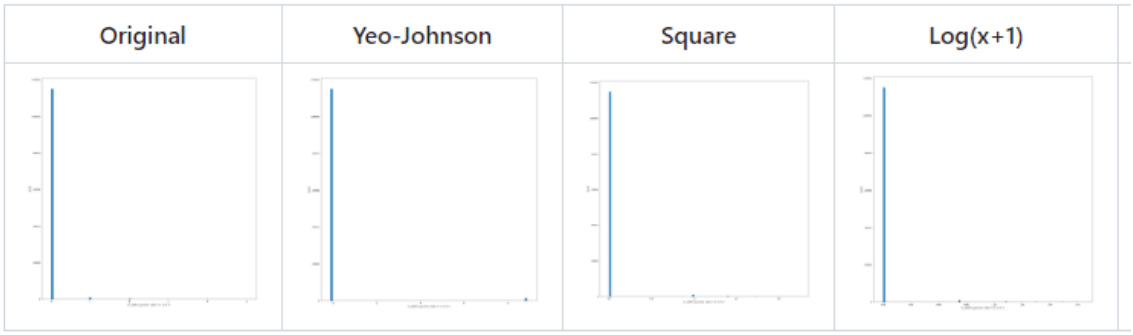

Figure 7.15: Data Distribution for irc_positive_positive

irc_total

For feature irc_total, *no transformation* is needed since the original data distribution already looks normal-distributed.

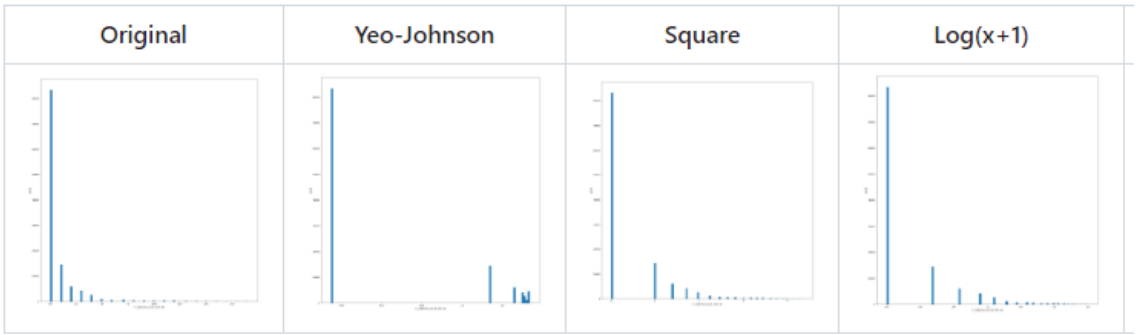

Figure 7.16: Data Distribution for irc_total

res_charge

For feature res_charge, *no transformation* is needed since the original data distribution already looks normal-distributed.

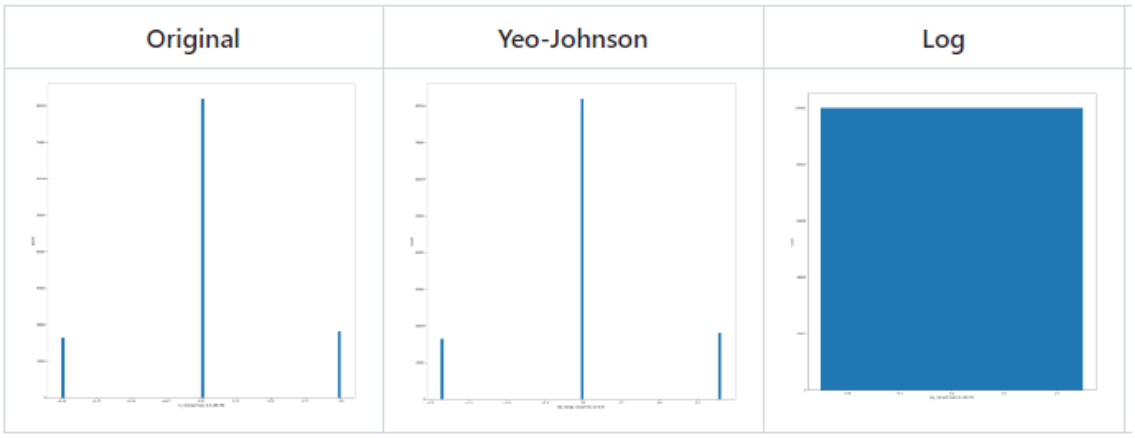

Figure 7.17: Data Distribution for res_charge

res_depth

For feature res_depth, applying *log(x+1) transformation* makes the data distribution most normal-distributed.

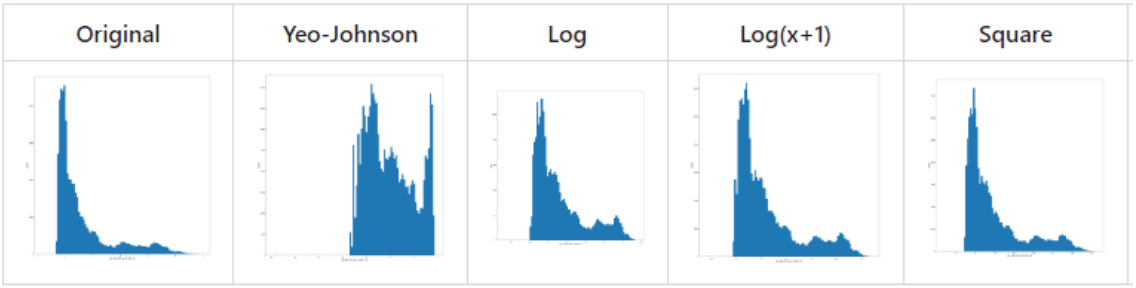

Figure 7.18: Data Distribution for res_depth

res_size

For feature res_size, *no transformation* is needed since the original data distribution already looks normal-distributed.

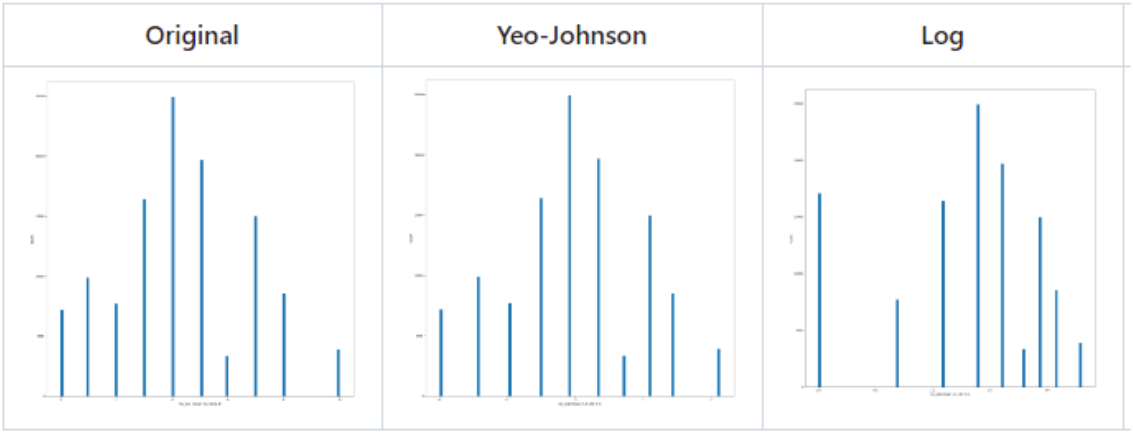

Figure 7.19: Data Distribution for res_size

sasa

For feature sasa, applying *square root transformation* makes the data distribution most normal-distributed.

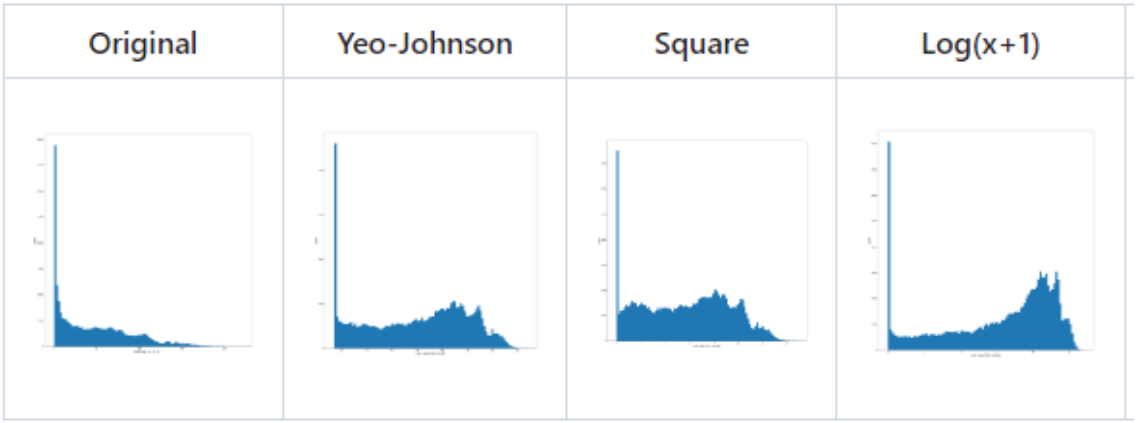

Figure 7.20: Data Distribution for sasa

vanderwaals

For feature vanderwaals, applying *cube root transformation* makes the data distribution most normal-distributed.

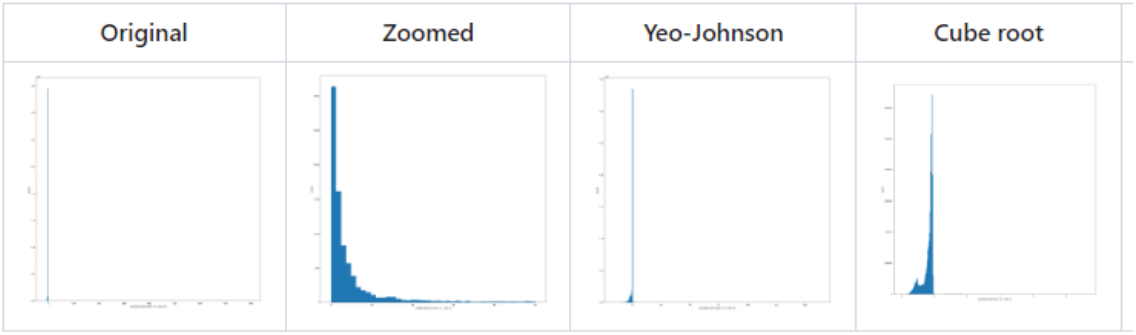

Figure 7.21: Data Distribution for vanderwaals