GraphImmuno:

Graph Convolutional Neural Networks with Physicochemical Properties for Adaptive Immune Response

Mallikharjuna Rao Sakhamuri

M.Sc. in Big Data
Analytics and
Artificial Intelligence

2022

Ollscoil Teicneolaíochta an Atlantaigh Dún na nGall

Atlantic Technological University Donegal

Department of Computing, ATU Donegal, Port Road, Letterkenny, Co. Donegal, Ireland.

GraphImmuno:

Graph Convolutional Neural Networks with Physicochemical Properties for Adaptive Immune Response

Author: Mallikharjuna Rao Sakhamuri

Supervised by: Dr. Shagufta Henna

A thesis submitted in partial fulfilment of the requirements for the

Master of Science in Computing in Big Data Analytics and Artificial Intelligence

Submitted to Atlantic Technological University

Arna chur isteach chuig Ollscoil Teicneolaiochta an Atlantaigh September 2022

Declaration

I hereby certify that the material, which l now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in Big Data Analytics and Artificial Intelligence, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to ATU's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been psudonymised and the data set and identifiers are not held by ATU. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of education assistance to future students and will be shared on the ATU Donegal (Computing) website (www.lyitcomputing.com) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in the Ireland. Consent can be withdrawn for the publishing of material online by emailing Thomas Dowling; Head of Department at thomas.dowling@lyit.ie to remove items from the ATU Donegal Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@lyit.ie to remove items from the Research THEA website. Material will continue to appear in printed formats once published and as websites are public medium, ATU cannot guarantee that the material has not been saved or downloaded.

Signature of Candidate	Date
Mallikharjuna Rao Sakhamuri	24-08-2022

Acknowledgements

It gives me immense pleasure to thank my dissertation mentor, Dr. Shagufta Henna, for her guidance and encouragement throughout the process. When I was writing my thesis, she was very supportive and cleared all my doubts. Her positive attitude and efforts inspired me to finish the dissertation.

I greatly benefited from the machine learning and artificial intelligence knowledge that Drs. Kevin Meehan and Shagufta Henna taught me as I worked on my dissertation. I'd like to convey my gratitude to them for their contributions to my development and success.

I would also like to thank Mr. Guangyuan Li, one of the authors of the research paper "DeepImmuno." When I contacted him for clarification on the DeepImmuno model, he answered all my questions shiftily.

This would not have been possible without the exciting contributions of molecular biology researchers and scientists. I am grateful to each of them for sharing their knowledge and opening up new avenues for me to explore, resulting in a memorable journey of a lifetime.

Abstract

The emergence of the Covid-19 pandemic has brought much-needed attention to the adaptive human immune system. Adaptive immunity entails specific immune cells and antibodies that target and eliminate invading pathogens while also being capable of preventing illness in the future by developing a new immune method. To comprehend about the immune system, gaining knowledge about T-cell response, peptides, and HLA interactions is mandatory. Acquiring such valuable knowledge will significantly aid in the development of new drugs and antibiotics that activate T-cells to combat the malicious pathogen and improve human immunity. There are currently few models and tools based on traditional ML methods traditional AI's convolutional networks to predict and explain the immunogenicity values-based peptide and HLA's reciprocal action. Although these methods perform adequately, they fail to represent network structure and do not explain the amino acid correlations of peptide and HLA. In this thesis, I propose "GraphImmuno," a graph-based neural network endowed with physicochemical properties. The model has been validated on Covid and dengue data with high accuracies. Further, the proposed model also performed graph learning tasks on the immunogenicity data, such as supervised and unsupervised learning.

Key words: immunity; artificial intelligence; graph convolutional neural network; peptides, HLA, pathogens; COVID-19; anti-biotics.

Acronyms

Acronym	Definition
HLA	Human Leukocyte Antigens
MHC	Major Histocompatibility Complexes
Pep	Peptide
NN	Neural Network
ML	Machine Learning
AI	Artificial Intelligence
CNN	Convolutional Neural Network
GCN	Graph Convolutional Network
PCA	Principal Component Analysis
MSE	Mean Squared Error
ROC	Rate of Change
AUC	Area Under the Curve
IEDB	Immune Epitope Database and Analysis Resource
APC	Antigen-Presenting Cells
AMP	AntiMicrobial Peptides
DNN	Deep Neural Network
GPU	Graphics Processing Unit
FN	False Negative
FP	False Positive
TP	True Positive
TN	True Negative
GWAS	Genome-Wide Association Studies
SMILES	Simplified Molecular Input Line Entry System
ANN	Artificial Neural Network
PPI	Protein - Protein Interaction

TCR	T Cell Receptor
T-Cell	T lymphocyte (White blood cell type)
RNA	Ribonucleic acid
DNA	Deoxyribonucleic acid
MAPPs	MHC-associated peptide proteomics
SARS	Severe acute respiratory syndrome

Table of Contents

Declaration 3
Acknowledgements 4
Abstract 5
Acronyms 6
Table of Contents 8
Table of Figures 10
Table of Tables 11
Table of Equations 12
Table of Algorithms 13
Table of Data values 14
Table of Code Lists 15
Chapter 1 1
1. Introduction 1
1.1. Purpose and Research Questions 4
1.2. Approach and Methodology 6
1.3. Contribution 6
1.4. Scope and Limitation 7
1.5. Report Outline 8
Chapter 2 9
2. Related Works 9
Chapter 3 19
3. GraphImmuno - Graph Convolutional Neural Networks with Physicochemical Properties for Adaptive Immune Response 19
3.1 Data Collection 19
3.2 Graph Model 21
3.3 Feedforward Neural Networks 23
3.4 GraphImmuno 25
3.5 DeepGraphImmuno 28
3.6 Embedded – GraphImmuno 30
Chapter 4 32
4. Performance Evaluation of GraphImmuno for Immune Response 32
4.1 Hardware 32

4.2 Software 32
4.3 Performance Metrics 33
4.3.1 Accuracy 33
4.3.2. Precision 33
4.3.3. Recall 33
4.3.4. Receivers operating characteristics (ROC) 34
4.3.5 Mean squared error (MSE) 34
4.3.6 Log Loss 34
4.3.7 Optimizer 35
4.4 Model Parameters (Hyperparameters) 35
4.5 DeepImmuno as Baseline 36
4.6 Results and Analysis of FFNImmuno 38
4.7 Results and Analysis of GraphImmuno 39
4.8 Results and Analysis of GraphImmuno over disease datasets 41
4.9 Results and Analysis of DeepGCN – GraphImmuno 43
4.10 Results and Analysis of Embedded – GraphImmuno 44
4.11 Discussion of the Results 46
Chapter 5 47
5. Conclusion and Future Works 47
5.1 Conclusion 47
5.2 Future Works 48
References i
Appendices viii
A. Data Samples viii
B. Code Listing x

Table of Figures

FIGURE 3.1 BLOCK DIAGRAM FOR GRAPH GENERATION USING HLA, PEPTIDE, AND PHYSICOCHEMICAL PROPERTIES OF AMINOS..... 22
FIGURE 3.2 GRAPHS FOR TWO DIFFERENT HLA AND PEPTIDES INTERACTIONS WITH INTER AND INTRA EDGES. 23
FIGURE 3.3 GRAPHIMMUNO - GRAPH CLASSIFICATION MODEL. 28
FIGURE 3.4 DEEPGRAPHIMMUNO - GRAPH CLASSIFICATION MODEL SORT POOLING LAYER. 29
FIGURE 4.1 ROC OF DEEPIMMUNO MODEL CNN WITH K10 FOLD VALIDATION..... 37
FIGURE 4.2 PRECISION VS RECALL OF DEEPIMMUNO-CNN WITH 10-FOLD VALIDATION. 37
FIGURE 4.3 LOSS VALUE VS EPOCHS FOR FFNIMMUNO. 38
FIGURE 4.4 ACCURACY VALUE VS EPOCHS FOR FFNIMMUNO. 39
FIGURE 4.5 LOSS VALUE VS EPOCHS FOR GRAPHIMMUNO. 40
FIGURE 4.6 ACCURACY VALUE VS EPOCHS FOR GRAPHIMMUNO..... 41
FIGURE 4.7 LOSS VALUE VS FOLDS FOR GRAPHIMMUNO..... 42
FIGURE 4.8 LOSS VALUE VS EPOCHS FOR DEEPGCN – GRAPHIMMUNO..... 43
FIGURE 4.9 ACCURACY VALUE VS EPOCHS FOR DEEPGCN – GRAPHIMMUNO..... 44
FIGURE 4.10 LOSS VALUE VS EPOCHS FOR EMBEDDED - GRAPHIMMUNO. 45
FIGURE 4.11 VISUALISATION OF THE EMBEDDINGS DISPLAYING THE QUALITATIVE MEASURE OF THE EMBEDDINGS..... 45

Table of Tables

TABLE 4.1 HYPERPARAMETERS OF GCN. 35
TABLE 4.2 GCNSUPERVISEDGRAPHCLASSIFICATION LAYER`S HYPER PARAMETER VALUES..... 40
TABLE 4.3 GRAPHIMMUNO MEASUREMENT VALUES OVER DISEASE DATASETS. 42

Table of Equations

EUQATION (3.1)…….. 26
EUQATION (4.1)……33
EUQATION (4.2)……33
EUQATION (4.3)……34

Table of Algorithms

ALGORITHM 3.1 : PHYSICOCHEMICAL-AWARE ENCODING STRATEGY 20
ALGORITHM 3.2 : GRAPHMODEL 21
ALGORITHM 3.3 : FFNIMMUNO 25
ALGORITHM 3.4 : GRAPHIMMUNO 27

Table of Data values

DATA VALUES 1 : HLA AND PSEUDO VALUES. VIII
DATA VALUES 2 : DATA SAMPLES FROM AAINDEX. VIII
DATA VALUES 3 : IMMUNOGENICITY VALUES FOR PEPTIDE AND HLA. IX

Table of Code Lists

Code List 1 HLA data frame to dictionary conversion. x
Code List 2 Dictionary Inventory. x
Code List 3 Graph Converter Combinator, numerical and unweighted edge methods. xi
Code List 4 Graph Converter weighted anchor edge and inter -intra methods. xi
Code List 5 Graph converter entrance method. xii
Code List 6 Method to modify unknown HLA values. xii
Code List 7 GCN preparation and training and validation. xiii
Code List 8 Validating the graph model on Dengue and Cell databases. xiii
Code List 9 Validating the graph model on Covid dataset. xiv
Code List 10 Graph Classification model. xiv
Code List 11 Graph generation and validating the graph models. xv
Code List 12 DGCN layer design. xv
Code List 13 DGCN model preparation and training. xvi
Code List 14 Unsupervised graph classification distance measure. xvi
Code List 15 Unsupervised model validation. xvii

Chapter 1

Introduction

With the breakdown of covid 19 pandemic there is an increase in the awareness of the Immunogenicity and immune system of human. Immunogenicity explains how an external agent such as vaccine works well in the human body to improve the immune strength of the humans. Thus, predicting the agents which increase the Immunity and their interaction with the cell's agents gained much significance in the recent days. This helps in producing the new antigens and new drugs which can effectively counter the disease-causing pathogens.

T-cells are responsible for fighting against the forging substances that enter a human body. T-cells are major components of white blood cells and mostly present in the bone marrow. Similarly, major histocompatibility complex (MHC) is a group of gens whose main functionality is to encode the peptide present on the surface of the cells. The resultant reaction will be terminating the disease-causing pathogens and eliminating them from the system to increase the immunity of the human system. An immune pathogen may escape the MHC detection and not letting it`s fragments appear to the corresponding T-cell`s recognition.

The MHC that is usually found in humans is referred as human leukocyte antigens (HLA) and functionality of the HLA is same as that of MHC which can be found in the other vertebrate species. MHC has different classes among themselves and interactions and bonding among the HLA classes and the peptides happens at various ranges making them polymorphic. Peptide is chain of amino acids, and the amino acid shape determines the shape. The polymorphic functionality is also known as protein folding. The shape of the amino acids string directs the functionality of that cell is the well-known definition of protein folding.

The first impression of the word "Protein" is as a nutrition in the food that we consume. But protein is a microscopic molecule inside a cell which performs various important jobs. the etymology of the word "protein" is "protas", which means "primary importance" (Dobson,

2003). Protein us a basic structure of life also known as molecule too. A molecule is a smaller chemical component that remains it`s chemical properties. Protein is made of smaller components called amino acids. A protein can be understood as a string of different colored beads. Each bead represents an amino acid. These amino acids contain smaller molecules contain carbon, oxygen Hydrogen and sometimes sulfur atoms. As protein is typically a string made of amino acids, there are 22 different types of amino acids that can combine. Depending on the order how these amino acids interact together the protein folds up into a particular shape. That shape influences what the protein does in human body.

Proteins can be seen as the workhorses of the cell and the perform various tasks related to structure and strengthening the cells. These actions allow the cells to grow and divide. The wealth and diversity and specificity of the functions a protein because of the property called “protein folding”. Hence a protein function depends on its shape. Failure in these “protein folding” is the cause of several diseases. “pathogens” is compound of a protein structure that makes a human sick. Inhibiting the function of such wrongly folded proteins will the diseases development in a host environment can be restricted. These reasons highlight the reasons to study the protein structure shape to learn the RNA, DNA, metabolites, molecular mechanisms, and cellular processing of under lying life. By exploring the specific topic of computational biology, molecular interaction, protein function, genome assembly, computational neuroscience and electronic health record, protein – protein interaction and mutagenicity prediction can be solved. These predictions can significantly help in new drug development.

Hence, figuring out the interaction between HLA and the peptides of the foreign agents in the cell system helps to identify and make predictions about the new antigens that can boost the immunity of the humans and to identify the extern agents which can trigger HLA of the cells to fight against the virus. Over the last few years with the advent of quality dataset of HLA helped in developing methods and tools which can identify the bonding between HLA types and the peptides. But most of these tools are developed on the conventional models. Most of the prediction model achieved decent accuracy but failed establish high standards with the prediction. Sometimes, even though the score predicted by the tools are high for few

sequences of TCR and peptide the does not infer immunity and are not able to explain the internal interaction and within interactions between the HLAs and peptides.

Most of the existing immunogenicity prediction models are built using scored function-based methods, consensus-based methods, and machine learning models. The first two methods relay on scoring functions of HLA combinations and integrating and binding methods. Further the models proposed in the previous works including neural networks perform well on the HLA and peptide datasets. They are successful in discovering the nonlinear relationships between the given combinations. The conventional neural network approximates the numerical values of the amino acids structure and properties. One of the successful models developed uses physicochemical properties of peptides. Physicochemical properties are directly derived from the peptides explain the behavioural nature of a peptide. These properties explain the behaviour and interaction expectations with biological and non-bio environmental surfaces. These properties are composed of molecular weight, melting point, vapour point, distribution etc.

Recently DeepImmuno (Li et al., 2021) is proposed that is variation of CNN model is a convolutional neural network model which successfully predicts the immunogenicity score for a given peptide and HLA combination. This model utilizes the physicochemical properties of the amino acids. The data is extracted from the AAindex database maintains the multiple kinds of physicochemical and biochemical properties of amino acids and the combination of the amino acids. Any given protein shape, it`s functionality and behaviour can be explained by the combinations of the above-mentioned properties. Out of all the available properties the most frequent 20 properties are selected and maintained in AAindex database.

DeepImmuno processes physicochemical properties for a selected list of HLA and peptide combinations. PCA is applied on 566 HLA and peptide properties data values for computational easiness. Encoded HLA and peptide values are passed as inputs to the fully connected CNN network to predict the immunogenicity values. Initially the model is trained on training dataset and later verified on the test dataset. The model is applied on various disease datasets such as dengue and covid. Overall, the model performed efficiently and attained high accuracy score.

Existing literature review shows that the DeepImmuno and other CNN model do not capture the connections importance of the interactions between the amino acids. The CNN model is unable to model the dependencies or relationships that can exist among Cytolytic T-cells to play an essential role in the adaptive immune system. Furthermore, current approaches do not account for the relationships and dependencies that exist in a molecular network. Understanding the forms and models of proteins can help us better understand how they work inside the human body, as well as develop new medications and proteins.

Further existing works face a scarcity of knowledge about approaches for solving protein and molecule structure problems, as well as learning assignments on protein structures. For modelling complicated protein and molecular networks, many computational approaches, such as existing traditional neural network models, are insufficient. Exploration of a strategy based on link representation learning for accurately predicting molecular associations to execute tasks such as molecular network graph representation, node classification, link prediction, graph classification, and graph embeddings remains a problem.

1.1. Purpose and Research Questions

Understanding these molecular networks and protein interaction complex networks may be best served by graph neural networks. Graphs are a type of data structure that is made up of two components: nodes (or vertices) and edges that connect two nodes. As a result, a graph can be described as a collection of nodes connected by edges that are only weakly. An adjacency matrix is frequently used to represent a graph, and its dimension is made up of the number of nodes in the graph. Graph Convolutional Neural Networks (GCNs) are a deep learning approach that performs analysis and reasoning on graph-described data. They are neural networks that can be used directly on graphs and offer a quick way to predict events at the node and edge levels.

GCN's prime objective for each amino acid node is to learn an embedding that includes details about its surroundings. Numerous applications, including node labelling, node prediction, edge prediction, and others, can make use of this embedding technique. Therefore, edges can

be transformed by including feed-forward neural network layers and combine graphs with neural networks after assigning embeddings to each node. The properties of each amino acid combined with neighbour amino acids properties give better understanding in the context of peptide and HLA reactions for triggering the T-Cells.

In contrast to DeepImmuno, to exploit the dependencies, interactions and relationships among Cytolytic T-cells, in this work, this thesis proposes a novel approach to immunogenic peptides for T-cell immunity prediction using Graph-based convolutional neural networks. The proposed graph-based approach utilizes psychometric-aware encoding strategy to balance the performance across diverse test datasets. Further, it proposes a graph based neural network coupled with psychometric-aware encoding strategy that can work well with a graph model and is stable under various dataset sizes and perform graph-based tasks such as supervised classification and unsupervised classification.

As a result, I'll use GCN models to solve the graph representation of protein molecules, and I have performed try learning tasks like classification of nodes and graphs, prediction of links (edges), and graph embeddings on the graph representation of molecular networks. The thesis aims to answer the following research questions:

1. Can feed-forward neural networks (FNNs) with physicochemical-aware encoding strategy accelerates the learning in contrast to DeepImmuno?
2. Can a graph model adequately describe the correlations and interactions that may exist in immunogenic peptides of covid disease data efficiently?
3. Is it feasible to conduct learning tasks on the graph representation of molecular networks, such as graph classification and graph embeddings?
4. Can physicochemical-aware encoding strategy improve the performance of the graph-based model to predict non-native peptides to elicit a T-cell response of covid disease data in contrast to DeepImmuno?

1.2. Approach and Methodology

The goal of this thesis is to create a graph-based convolution neural network that uses the physicochemical properties of amino acids to predict the adaptive immune response to a given HLA and peptide combination. This thesis attempts to solve two major machine learning problems: prediction and classification. The immunogenicity values are predicted using the graph model, and the HLA-peptide combination graphs are classified.

The following is a description of the proposed methodology:

"DeepImmuno" data is processed to answer research questions using existing graph model libraries and evaluation metrics. The primary goal of this research is to see if a graph-based model can learn the relationship between HLA, peptides, and immunogenicity response. Furthermore, to see if it is possible to perform graph-related tasks on the given data, such as supervised and unsupervised classification and graph embedding. The model also experiments with a feed forward network to test the model's learning ability.

The physicochemical-aware encoding strategy is used to convert the physicochemical properties data extracted from the AAindex database. For a given HLA and peptide combination, the encoded data values are converted to a graph adjacency matrix. The graph convolutional neural network (GCN) is used to predict immunity values on the covid and dengue data sets. Later, GCN is used to perform supervised and unsupervised classification tasks on the graph data. Finally, the process and results are compared to one another and to the baseline model. The limitations of the baseline model are highlighted, and a prospect for future developments and studies on this thesis is provided.

1.3. Contribution

The following are the primary contributions of this thesis:

- This thesis investigates the work on classification and prediction tasks for immunogenicity with critical insights and analysis.

- Further, I have prepared the "DeepImmuno" dataset by applying various techniques, such as dimensionality reduction.

- In the first instance, this work has applied physicochemical-aware encoding strategy on feed forward neural networks to accelerate the learning in contrast to the state-of-the-art model DeepImmuno.

- Secondly, this is the first work that models the “DeepImmuno" dataset to a graph model using graph-based neural networks to show interactions and relationships among amino acid of peptide and HLA.

- Thirdly, in contrast to previous works on Graph Convolutional Neural Networks, this thesis proposes a novel GCN model with physicochemical properties for adaptive immune response called as GraphImmuno to predict immunogenicity for given HLA and peptide combinations.

- Further, the work applies both the supervised and unsupervised graph classification tasks based on the GraphImmuno model.

- Finally, the thesis evaluates and compares the proposed FNN with physicochemical-aware encoding strategy and (GraphImmuno) to predict non-native peptides to elicit a T-cell response of covid disease data.

1.4. Scope and Limitation

Due to the limited time frame, some constraints must be imposed on this research to ensure that the work is completed on time.

- The data collected and prepared by the 'Deepimmuno' model is used in this work. There is no new public data added to the datasets.

- The edge weights are explicitly assigned due to a lack of accurate weight information among the edges. If the edge type is intra, a weight value of 2 is assigned, and a weight value of 1 is assigned to the inter edges.

- The output value for predicted immunogenicity is a binary value. The dataset contained four immunogenicity value classes, but they were converted into binary classes for ease of processing.

- Very few antigens with proven experimental immunogenic values may have a low confidence value due to incorrect T-Cell and HLA interaction values.

- Due to a lack of accurate HLA and Peptide node data, graph prediction tasks such as node prediction and graph prediction are performed.

- The structural coordinates and shape of the amino acids are ignored because the model's primary focus is on physicochemical properties.

- The prediction is model is not published to any interface and not deployed to any portal using docker as the scope of the work is to validate the proposed model.

1.5. Report Outline

The following chapter 2 details few of the recent works applying ML and AI methodologies in Immunotherapy, briefly discussing the novelties of the models and their shortcomings. This chapter is a literate survey study on Immunotherapy and related anti-biotic research. The various industrial implementations and products designed for immunogenicity prediction classification are discussed. Finally, the computational techniques available for developing the corresponding solutions to design the immunogenicity prediction model are evaluated. Chapter 3 explains the theoretical background for the proposed models, focusing on data preparation, development methodologies, neural network approaches, and model`s pseudo codes and block diagrams. next chapter, chapter 4 explains the experimental details and specifications, as well as the results of the experiments. Finally, chapter 5 summarizes the work and provides an outlook for possible future research and expansion on this topic, as well as makes some recommendations to overcome the shortcomings in immunogenicity prediction research and development.

Chapter 2

Related Works

The discovery of the molecules' anti-biotic behaviour specific to t-cell immunity piqued the interest of the bio and medical research communities particularly in the recent years. To overcome computational difficulties, researchers are employing advanced machine learning models and neural network models. In this chapter, I will review few of the AI and ML models and research papers that have been proposed in recent years to predict and classify peptide and HLA data related to human immunity. In the following sections of this chapter, the goal and methodology of the research work are briefly described, as well as the work's novelty and contributions.

(Malone, B. et al 2020) proposes a novel approach to Covid vaccine development that targets pathogens associated with the SARS-CoV-2 virus. To achieve the goal, this model combines two distinct methods. The majorly focused HLAs and their interactions with epitopes are 100 HLA-A, HLA-B, and HLA-DR. The immunogenic regions for the selected HLA types are identified using epitope hotspots stimulated by "Monte Carlo." Another stimulus is to identify TCR hotspots in humans that respond to viruses in order to boost immunity. This study provides information about epitopes, which will be very useful in developing vaccines. Though this model was successful in providing a blueprint for the interaction, it did not explain the crucial amino acids involved in the interaction or their structure.

(Barra, C. et al 2020) created the NNAlign MAC ANN to predict the correlation of MHC-associated peptide proteomics (MAPPs) and CD4 T-cells. This model is primarily concerned with MHC of class 2 and their interactions with specific MAPPs. Despite the fact that this model operates on limited available data, its core strength is in reducing noise and utilizing the MAPP's data. The model is explained using the protein drugs infliximab and rituximab, which are currently available. This model successfully explains the behaviour of MHC class 2 and the role of antigens in the development of protein-related drugs. However, the capability of this model is limited to a small number of data points and cannot be applied to large data sets.

(Doneva, Doytchinova, and Dimitrov, (2021)) discusses immunogenicity silico methods used in the biopharmaceutical industry. Biological silico experiments are those that are conducted with computer stimulation. ANNs and molecular docking are two popular techniques for this. Molecular docking is a structure-based model that makes use of score functions and molecule structure. Based on their binding strength, these models focus on peptide binding with HLA. Though these models recognize 90% of B cell epitopes, the challenge comes when dealing with smaller molecules with flexibility because the number of bonds is greater.

(Dimitrov, I. et al 2020) experimented with six machine learning algorithms for identifying protective immunogens, which will greatly aid in vaccine development and research. Machine algorithms, discriminant analysis, K-nearest neighbour, random forest, support vector machine, and gradient boosting are used in the proposed methods. The experiments used supervised learning approaches to classify a given protein as immunogen or non-immunogen based on the bacteria's origin protein. The models achieved an accuracy of 84%, and gradient boosting performed better than the other models. It is possible to say that the accuracy score can be improved by implementing neural network models on the limited number of bacterial databases.

(Cotugno, N. et al 2020) created a machine learning-based predictive model to predict the immunogenicity score of influenza trivalent inactivated vaccine (TIV). Samples of peripheral blood mononuclear cells (PBMCs) are collected from HIV-infected children, and vaccine peptides are applied in vitro. RT-PCR fluidigm is used to process the sorted cell samples. FACS cell sorting strategies are used on T-cell and B-cell samples that have not been stimulated in vitro. The gene expressions from the various subset values are fed into the prediction model as input data. Gene selection methods are used on gene expression data to select features, which are then passed on to ML models composed of SVM, random forest, and elastic net. The prediction score and gene importance ranking for the selected genes as outputs of the adaptive boosting model. The model performed admirably, with an accuracy score of 95.6% and only one false positive value. The only disadvantage of this prediction model is that it was trained on a small dataset and multiple folds validation was not performed on the data, which would have prevented overfitting.

(Lee, E.K. et al 2016) created the "DAMIP" gene predicting model to test the vaccine's immunity and efficiency. DAMIP (discriminant analysis via mixed-integer programming) is a non-linear machine learning model that produces multiclass output values. The model was fed values from two sources: external images and clinical data. External image data consists of global gene expressions and MRI images, whereas clinical data is based on real-time patient portfolio data. The data is extracted from the models using pattern recognition and feature selection methods. The ML models used are ensembled SVM, naive Bayesian, and random forest models. This prediction model produced satisfactory results with an error rate of less than 20%. Though this model outperformed other gene classification models, the data set used to train the model is extremely limited, and better results can be obtained using advanced neural network prediction models.

(Bezu, L. et al 2018) proposed a machine learning model based on the physicochemical properties of anti-cancer agents. Cancer-causing pathogens cause cell damage, which is also known as cell stress. As a result, the drugs develop a property known as immunogenic cell death (ICD). The proposed model computes the ICD score using the physicochemical properties of the given cell. The dimension reduction technique principal component analysis is used, and the dimensions are reduced in linear order based on the weight of the cells.

Following the experiments, it was determined that physicochemical properties alone are insufficient to determine a cell's ICD, but they can provide a certain probability with large chemical databases with ICD.

(Humeau, J. et al 2020) created the aforementioned ICD score model. The ICD model has been trained on 50,000 agents that can boost immunity by reducing cell stress. These agents have the potential to increase the cell's ICD. This proposed model, too, is based on the agents' physicochemical properties. This model takes advantage of the capacity to collect substantial amount of information, which the previous model did not. The previous model's properties serve as the foundation for this model, but the experiments are tailored to a specific cancer application. Despite the fact that this model is quite successful in identifying the agent "actinomycin D" using the ICD prediction score and by comparing positive and negative values, a portion of the experiments were conducted in virto stimulation.

(Marciniak, A., Tarczewska, M., and Kloska, S., 2020) used a random forest classifier algorithm on peptide and HLA(MHC) data. The crux of this model is in the model's data preparation. The peptide and MHC data is extracted and processed from the public database "Biohack." Initially, the peptide data is encoded according to its length, and the MHC values are encoded using one hot encoding. The missing amino acids are represented by the value zero as the final step in data preparation. The prepared data is fed into the random classifier model, and the experiments are run with the estimator number 416. The model achieved an accuracy of 84%, but there is still room for improvement. Other parameters, such as the structure or properties of the molecules, were not taken into account by the model. This model can be enhanced by incorporating advanced AI computational methods and expanding the data parameters.

(Schaap-Johansen et al 2021) review the tools available for peptides that cause cancer and their role in developing anti-cancer immunity. T-cells are in charge of identifying the epitopes that cause disease. This task is more efficiently completed if the t-cell can distinguish between peptides from the self and peptides from pathogens. One of the tools examined is "POPI," which employs local and global physical and chemical properties as well as an SVM-based method. "POPISK," another model, combines SVM with kernel methods. These models can account for the non-linear relationship between peptides and HLA. The models scored 0.68 for accuracy and 0.74 for area under the curve. These tools rely heavily on HLA binding properties. In terms of efficiency, these tools continue to have limitations.

(Smith, C.C. et al 2019) focuses on answering questions about cancer-causing tumors, their relationship with the immune system, and the use of minor HLA values to assess immunity. Random forest, SVM, and gradient boosting were used in the model. This study effectively explained the correlations between HLA class types and leaves few questions unanswered about the bio and chemical properties of antigens and their role in tumor immunogenic epitopes. This study was unable to determine whether the MHC haplotype influences the immune system in humans.

(Aranha, M.P., Jewel et al 2020) proposes a novel model for identifying the similarity between peptide and HLA combinations by combining sequence-based ANN with structure-based modeling. The study's objective is to produce a model that will aid in the development of drugs and medicines that target T-cells. The study concludes that structure-based models, when combined with stimulated values, accurately predict MHC group binding values. Peptide binding with HLA is a necessary precondition for peptide reorganization in T cells. As a result of considering structure-based models, this model focuses on HLA and peptide binding. The ensemble model is used in this study to identify the correct bindings. However, because some pathogens act quickly, an improvised model is required to perform at a larger scale.

(Ribeiro da Cunha, et al 2021) proposes an ensemble method for spectroscopy using the ML model SVM. The advantage of using spectroscopy is that it is simple to use and free of labels. The model's output predicts the behavior of the peptides as well as the possibility of antibiotics. Furthermore, spectroscopy requires less data to function. The antibiotic's metabolic footprints and functionality are investigated. Spectroscopy is used to observe and record the changes that occur when cells interact with antibiotics. For prediction, the spectra details were processed using the SVM and K nearest neighbors ML methods. Though this model is quite successful with a limited amount of data, there is an additional complex step of recording the data using spectroscopy, and the ML models used are quite primitive in comparison to the advancement that occurred.

(Zoffmann, S. et al 2019) proposes a novel ML-based drug for dealing with observable physical properties of the components. The information is derived from the Roche library. The Roche library contains data on 1.5 million antibacterial components. The random forest algorithm is applied to the extracted data to find the similarities of multiparametric phenotypes among the selected anti-biotic data set. The experiments are carried out iteratively to calculate the similarities and differences using the random forest. Each iteration adds a new component to the set, and the average similarity score is used for valuation. The potential of this prediction methodology can be used in the initial screening of new antibiotics. For data preparation, the model took into account countable parameters of the component structure; thus, all aspects of the components are reflected in the model's results.

(Stokes, J.M. et al 2020) trains a neural network model to perform prediction of the antibacterial molecule. Molecules data is extracted from the Drug Repurposing Hub and fed into the model as input. To identify the growth in "E. col," the network is trained using a sample data set. The trained model is then tested against existing chemical libraries. These libraries contain over a hundred million molecules. The molecule values are sorted and tested based on the accuracy score. The molecules with a different structure than regular antibiotics are considered. The study concludes that "Halicin" meets these requirements. The halicin molecule has been tested against a few common bacteria peptides, and the results are promising. This model was successful in eliminating time-consuming manual chemical lab

work, but it required a large amount of data to produce results because it was tested on millions of molecules.

(Parvaiz et al., 2021) created a machine learning model to search for β-lactam in a large database of components. This is a well-known antibacterial agent that is widely used in the development of various types of resistant drugs. A data library of 700,000 records is analyzed using the machine learning algorithm random forest. 74 components are chosen from the result set and processed further for testing in laboratory and virtual environments. The main role of machine learning in this model is to find similarities in component structure, search for desired components in large data sets, and reduce manual labor. The model successfully identified 22 components with antibacterial activity against -lactams. This model has a flaw in that it uses consensus compound selection.

(Hamid and Friedberg, 2018) represents the protein sequence as embeddings using the work embedding technique of natural language processing. The embedding trigrams are used instead of sequence similarity methods. The model's goal is to distinguish between harmful and non-harmful peptides. In the application, the model uses recurrent neural networks as AI methods. The overlapping trigrams' embedding vectors are used as input for the RNN model. The RNN implemented in this model is bidirectional in nature. The previous time step's input value is combined with the current time step's input value to increase generalization and reduce the number of parameters. Furthermore, this model performs well with long sequences. Although this model outperforms SVM and other ML models, it has a high memory requirement and ignores cell genetic information.

(Fields et al., 2019) created a sequence-free model that takes advantage of the biological and physical properties of peptide toxins. The model investigates amino acids for evaluation and processing in the development of anti-bacterial drugs. The word2vec model in the ML pipeline is used to select amino acids from the collection. From the list, the pipelines select 20 amino and peptide combinations using a sliding window. These chosen combinations are combined with the biological and physical properties of the acids to be investigated in the synthesized environment. The model outperforms other models by utilizing sequence structures. The only flaw in this model is that only a few combinations with low confidence demonstrated strong antibacterial properties.

(Badura et al., 2020) created classification and regression models to discover the salts' anti-biotic properties. To process the molecular descriptors, this model employs an artificial neural network (ANN). Chemical properties are transformed into three-dimensional structures. To determine the molecular descriptors, these structures are processed using computational chemistry methods. The ANN model is a multilayer, unidirectional network. The regression model predicts MIC values, while the classification model predicts binary out regarding the active status of the component's anti-biotic status. Both layers' networks have 20 input units. The model trained using a prebuilt ANN tool and did not hyper tune the model.

(Nagpal et al. ,2018) proposed a model for predicting peptides that can activate antigen-presenting cells (APCs). The majority of vaccines interact with APC either directly or indirectly. Many existing methods concentrate on vaccine interactions with T- and B-cells. The proposed model uses the Support vector machine ML model to process A-cell experimental data from various sources. The model predicts A-cell interactions with epitopes. The model works with high-dimensional data with fewer data points. If more data is used to train the model, its performance can improve.

(Wang et al., 2021) predicts the immunogenicity of single sugar molecules referred to as Glycans. Unlike the other few molecules, the glycan chain structure is non-linear. As a result, a graph neural network (GNN) is used to predict the immunogenicity of this high-dimensional data set. The massage passing neural network was used to create this GNN model. In this proposed model, the node's values are considered by taking the values of all its surrounding neighbours. This is known as connected value aggregation. The model's initial training data is extracted from the CSDB database and then from the superbase database. The model performed exceptionally well, with a 96% accuracy value. Implementing the model with unlabelled data and using embedding layers are two examples of model improvements.

(Su et al. ,2019) created a neural network to analyze and recognize antimicrobial peptides. The DNN model is built by importing the kears library. As a stop loss function, the model includes an RMS optimizer and binary cross entropy. The model's hyper parameters are set to 0.0002, and the batch size is set to 32. The network is a multilayer convolutional network. When tested on benchmark datasets, the model achieved 73% accuracy. In other experiments, the model performed better when it was integrated into fusion models with multi-type feature values.

(Grafskaia et al., 2018) conducted transcriptomic research on marine animals to identify peptides and anti-bacterial behavior. For analysis, peptide information that has been synthesized is used. SVM and random forest machine learning algorithms are used to identify AMPs in the data. AMP can be thought of as a shorter version of the peptide. Pathogens are combated by them. The structural characteristics of the amino acid are considered for analysis. From the synthesised data, the model successfully identified three peptides. Though the model was accurate, the data used for the analysis did not apply to the larger picture and missed the real-world connection.

(Macesic et al., 2020) created a machine learning model to predict Polymyxin resistance in a genome dataset. The genome is the complete set of DNA instructions found in a cell. To represent the features, both a reference-based and a reference-free approach is used. The Genome-wide association studies (GWAS) filter aids in the identification of various DNA characteristics. Machine learning algorithms were applied to the data with and without GWAS filters. To validate the test data set, the final model with the highest accuracy was chosen. The algorithms achieved AUROC values ranging from 0.885 to.933. Though the model

outperformed rule-based approaches, it can be improved by incorporating a wider range of data. The current data set excludes single reference chromizes and certain types of genomes.

(Mohapatra, An, and Gómez-Bombarelli, n.d.) proposes "Macromolecule," a graph-based neural network, to understand immunogenicity in glycans. The chemical structures of the processed data are saved in text file format. These text files are then converted to SMILES format. SMILES is an abbreviation for Simplified Molecular Input Line Entry System. The SMILES format can be used to represent chemical structures that computers can understand. NetworkX, a graph library, is used to convert these string representations into graphs. Nodes represent the molecules, and edges represent the connections between them. On this prepared graph data, a graph neural network is used to perform classification and regression tasks. The classification model classified the labels based on the immunity label in the graph. The prediction model predicted the graph's biological activity. All of these are supervised learning models. This model used a graph model to represent the relationship between molecules, which other models did not do, but the data did not take the properties of the molecules into account.

(Burkholz, Quackenbush, and Bojar, 2021) develops a graph-based model to aid in their understanding of glycan structure and properties. "SweetNet" was the name given to the model. The architecture of the model is comprises of node embedding layers, graph convolutional layers, and fully connected layers. Galyens are passed as input in the form of a node list containing nodes and information about the bonds between them. The graphs transform this data into graphs that the graph convolution layers can process. The SweetNet model's learned representation is used to predict virus receptors and identify glycomic similarity. Ecological and biological data were used to validate the model. The model's MSE is 0.784. Better glycan data quality, it is agreed, will improve the model's performance.

(Weber et al., 2020) develops a model to predict immunogenicity values for peptides and HLA combinations based on the structural and dynamical properties of the molecules. The graph model architecture is made up of an input layer, a convolutional layer, a pooling layer, a dense layer, and a SoftMax layer as the activation layer, in that order. Peptide and Hal data are converted to matrices and subjected to a convolutional filter. To prepare categorical data for neural network input, one hot encoding is used. Data values for immunogenic and non-immunogenic data are extracted from the IEDB database. The average AUC value remained at 0.70 after ten folds of validation. The model successfully represented the structure of the molecules as graphs. The model did not produce exceptional results due to a lack of accurate and high-quality input data.

(Jabbari and Rezaei, 2019) and (Feng et al., 2021) explain various AI applications in cellular immunotherapy. According to these research papers, there are currently fewer AI applications used in clinical medicine because there are insufficient databases related to cell knowledge and a lack of general rules for cell information processing. The unpredictable nature of nanomaterials and human immune systems is one of the major roadblocks to

processing immunity data. These studies show that using machine learning to convert the properties of components to predict the immune activation path between external agents and human immune cells has numerous advantages. Among them are the development of new drugs, the identification of pathogens, and long-term treatments for terminal diseases.

In addition to traditional machine learning methods, implementing computer vision-based applications to study and analyse molecular characteristics related to damaged cells of cancel and other illnesses will pave the way for new medical discoveries. Visual examination of damaged organs using computer vision has grown in popularity in recent years. However, due to a lack of accurate knowledge databases, computer vision has yet to be applied at the cellular level. Traditional microscopic observations can be empowered with AI methods if an adequate amount of cell information is available for computing.

(Hildebrand et al., 2021) explains the critical role AI is playing in predicting the response to cancer immunotherapy. This study explains how convolutional neural networks are used to process tumour-specific images. Cancer is caused by a lack of repair of damaged DNA, which is discovered through molecular testing. In this study, a convolutional neural network (CNN) is used to understand the immune system's response to the affected cells. The CNN is made up of convolutional layers, pooling layers, fully connected layers, and an output layer. To predict the various tissue damages, the model is applied to various cancer datasets. The CNNN model is modified by leveraging existing image processing knowledge using transfer learning methods. Transfer learning methods such as ResNet, Inception, and AlexNet performed better, with AUC values closer to 0.84. Though image processing works well for cancer and covid detection using X-rays and other medical scanned images, it may not adequately explain cell behaviour for immunity because it fails to take cell properties into account for predictions.

(Martins et al., 2019) creates a machine learning pipeline to predict peptide bonding levels and HLA values. IEDB is a public database that houses information on various experimental epitopes. By manipulating the DNA values, the data from the database is processed to generate various peptides. Multiple methods for determining HLA class 1 and 2 bonding are used on this forged data. The model's results are prepared in the form of metrics and presented to the users in the form of interactive visuals. Though this pipeline is successful, it does so by combining existing HLA class models. Furthermore, the pipeline was validated against any benchmark dataset in this paper.

(O'Donnell et al., 2018) proposes an AI-based method for predicting immunogenicity based on HLA class 1 and peptide bonding. HLA 1 molecules play an important role in activating T-cells to fight against externally harmful agents. MHCflurry, the model proposed in this research paper, outperforms existing immunity predicting models. This model works especially well with HLA values of length 9. The data value gathered from public samples and users is fed into neural networks. This is a collection of ten different network models. When validated on IEDB and other benchmark datasets, this model had an AUC of more than 80%. Though the model accuracy values appear to be good, the model's shortcoming is that very

little data is considered for training the model, and by deploying multiple neural networks together, the model is overfitted with the data.

(Baranwal et al., 2020) proposes a novel method focusing on protein-protein interaction with the goal of gaining better insights into protein functions to aid in immunity research. In contrast to the existing model, which uses protein sequence information, the new model proposed in this paper uses graph-based information extracted from the protein structure to analyze PPI interaction details. This model embedded two graphs and predicted the likelihood of their interaction using their chain ID. The proposed novel model accepts two different graphs and embeds them to form a graph attention model. After that, the graph values are routed through a feed forward network and a SoftMax activation layer. The model achieved an impressive 98.89% accuracy. This model has very few enhancements. This model does not take any molecule properties into account when predicting interactions.

(Banerjee and Preissner, 2018) proposes an ML model based on the structure of the chemical components to determine the nature of the components. The goal of this study is to determine one of the chemical properties of the component's "taste." The random forest model is used in this study to introduce a workflow. The data was extracted from the public databases SuperSweet and BitterDB in order to keep the data organized by taste property. Water molecules are excluded as part of the data preparation process, and hydrogen atoms are explicitly added because hydrogen atoms are not mentioned by default. The binary fingerprints represent the structural features of the components, and the double and triple bonds between the molecules represent the positive and negative charges. The use of a binary fingerprint to represent molecules is common in AI-related prediction and classification tasks. The model has an accuracy of 95% and an AUC of 98. The proposed model achieved high accuracy, but the data operations performed on molecule data in this model are very simple.

(Yang et al., 2020) creates a novel ML pipeline to determine the interaction between virus proteins and human immune proteins by combining the random forest method with doc2vec. Understanding the PIP between viruses and human T-cells is extremely beneficial in understanding human immune system response and drug development. In natural language processing, doc2vec is commonly used to represent documents as vectors. In this study, the Doc2vec technique is used to perform unsurprised sequence embedding. The data from positive and negative samples is combined to form a corpus, and features are extracted from it using the doc2vec method. The ensemble method is used to predict using the reserved 80% of the training data. During validation, the model achieved 80% recall value, indicating that its performance is satisfactory. Despite having high accuracy values on test data, the model failed to meet expectations on new data sets.

Though all of the models mentioned above make significant contributions to the prediction of immunogenicity values using HLA and peptide details, as well as understanding the trigger response of T-cells to activate the human immune system, they have a few flaws. Many of them couldn't explain or use the internal structure of molecules and acids. Physical, chemical,

and biological properties of atoms all play a role in determining the outcome of any chemical reaction. Very few existing AI models take these properties into account when performing prediction and classification tasks. As a result, I propose in my thesis a graph model embedded with physicochemical properties for predicting immunogenicity values for given peptide and HLA combinations, as well as performing graph-related tasks. The ensemble method random forest is used for

Chapter 3

GraphImmuno - Graph Convolutional Neural Networks with Physicochemical Properties for Adaptive Immune Response

This chapter focuses on the methods used in developing graph models to represent HLA and peptide interactions as graphs. The approach includes data collection, data analysis, and data preparation for graph representation. The physicochemical properties of amino acids are applied to the peptide and HLA data collected to create adjacency matrices that represent the graph forms of the HLA and Peptide interactions.

Furthermore, this chapter also presents the methods used to identify the nodes, edges, and edge weights. The graphs models are implemented with variations in the model's architecture by incorporating different pool techniques. The following subsections explain the specifics of the graph models variations. The architecture of the models is designed with the goal of performing supervised and unsupervised graph tasks.

3.1 Data Collection

Peptides are collected from the Immune Epitope Database after disease-related molecular data has been analysed. This data set contains only convalescent and unexposed Cell, Dengue, and Covid datasets. As part of the data preparation process, only MHC values with lengths greater than 4 are considered for analysis. The selected HLA and peptides are encoded to a number string using the physicochemical properties of the amino acid. The AAindex1 database contains amino acid information based on their physicochemical properties.

Dimensionality reduction is used because the initial list of properties chosen from the database is too large for calculation. To reduce the size of the property list, the principal component method is used. Further, to normalize the outlier among the list of robust property values, I have used the RobustScaler library function. Only relevant features from the normalized feature matrix are retained after the noisy features are discorded. I have added a placeholder (-) value, peptide values of size 9 are adjusted to the same size.

Based on the peptide and T-cell response, the immunogenicity strength value is calculated using the beta distribution. Prior beta values are encoded using immunogenic class values. Totally, four immunogenic values that are taken into account (Negative, Low Positive,

Intermediate Positive, and High Positive). The average values for the given combination of the HLA(MHC) and the peptide are considered after bootstrapped iterations on the distributed.

ALGORITHM 3.1 : PHYSICOCHEMICAL-AWARE ENCODING STRATEGY

Input: *HLA paratopes, Amino acid associated physicochemical properties.*
Output: *Graph representation of HLA and Peptide data using the amino acid properties of T-cells.*
Input: *HLA paratopes, Amino acid associated physicochemical properties.*
Output: Physicochemical-aware encoded properties of AAIndex
    For *each indices* ***in*** AAIndex properties data
        Discord indices of the missing values
        If *len(HLA paratope sequences or peptides) >10* ***then***
            Add a placeholder amino acid "-" for padding
Input: *normalized feature matrix of physicochemical properties.*
Output: Workable feature matrix of Physicochemical properties
    For *all values of properties*
        Apply *principal component analysis (PCA) to the normalized feature matrix*
        If *variance >95%* ***then***
            Chose top 12 principal components
Input: *MHC allele.*
Output: Encoded HLA paratope values with length 46
    For *each MHC allele*
        Encode each MHC allele based on its paratopes' sequence
End

Algorithm 3.1 is the list of the steps for Physicochemical aware encoding strategy for the data preparation. The AAindex1 database supplies 566 physicochemical attributes related to amino acids. 13 indices out of the 566 characteristics are eliminated because the data for some amino acids is lacking. A placeholder amino acid "-" is added for filling in the gaps in the 9-mer peptides and HLA paratope sequences. The average of the other 20 canonical amino acids was used to determine the corresponding AAindex values. This approach brings the total number of amino acids to 21. Principal component analysis (PCA) is used to normalize the resultant 21 x 553 numeric matrix, removing noisy features, and leaving just the pertinent components. The selection of 12 primary components, accounts for 95% of the variance. The HLA and peptide staring are formatted as per length by padding if needed and the HLA staring

are encoded based on their paratope's sequences. Then the corresponding adjacency matrices are prepared for the HLA pseudo strings and peptide string values.

3.2 Graph Model

The combination of HLA and peptide is represented by an acyclic undirected graph. There are two kinds of edges in graphs. The connection between and within peptides and HLA is represented by intra edges, while the attraction between peptide and HLA is represented by inter edges. The weights are explicitly assigned by assigning a weight of 2 to the connection within and between HLA and a weight of 1 to the attraction between the HLA and peptide, emphasizing connections between the HLA and peptide. The node and edge data are converted into graphs for further analysis.

ALGORITHM 3.2 : GRAPHMODEL

Input: *ori dataframe, after_pca, hla_dic, dic_inventory*
Output : *graphs of hla and pep*

```
Source = p + str(i+1) for i in range(len(pep))
target = h + str(i+1) for i in range(len(hla))
Weight = itertools.repeat(1,len(source)*len(target))
Edges_intra = DataFrame(souce, target, weight)
feature_array = concatenate(feature_array_pep,feature_array_hla)
intra_pep = combinations(source)
intra_hla = combinations(target)
intra = intra_pep + intra_hla
Edges_inter = DataFrame(souce, target, weight)
graph = StellarGraph(nodes, edges)
return graph
```

End

Algorithm 3.2 is the Graphmodel`s list of the steps. The model is fed HLA and peptide dictionary values, as well as the immunity data frame and physicochemical data. Peptide values are converted into strings by appending the alphabet "P" based on peptide length. This peptide string is used to create intra-peptide edges. Similarly, HLA values are converted to strings by appending the letter 'H,' and intra-HLA edges are formed. Intra edges represent interactions with HLA and peptides. The term "inter edges" refers to the relationships between the HLA and the peptide. Weights are assigned based on the edge connection type. The Graph model retunes the graph format with nodes and edges for the given input data.

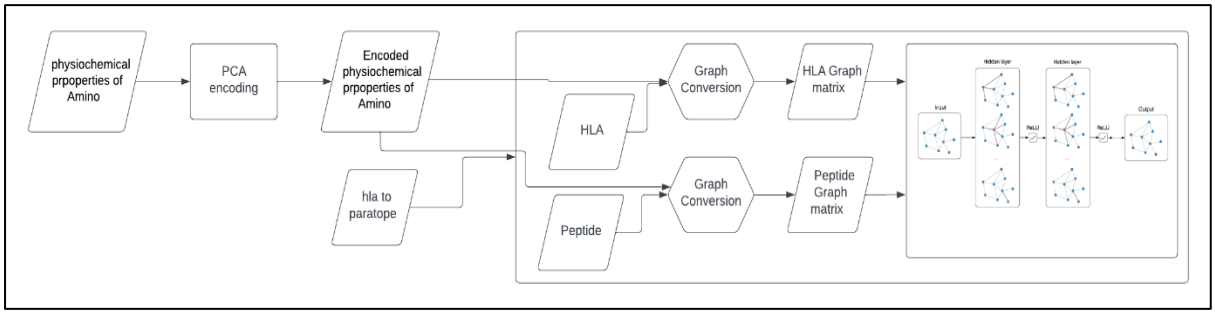

Figure 3.1 Block diagram for graph generation using HLA, Peptide, and physicochemical properties of aminos.

Figure 3.1 depicts a block diagram for graph generation with HLA, Peptide, and encoded values of amino physicochemical properties. HLA paratopes (HLA-antigen interacting residues) are used as a proxy for various HLA alleles, because these sequences contain the most important information to describe peptide-HLA spatial interactions during encoding. These sequences are used as input for each evaluated deep learning algorithm to represent each HLA allele and encoded peptide sequence in a numerical matrix. The extensive physicochemical characteristics of amino acids are taken into account by the AAindex encoding method. The input data is shared with a graph-based neural network to model the immunogenic peptides for T-cell immunity dataset, and a GCN coupled with psychometric-aware encoding technique that can function with a graph model and is stable across dataset sizes has been created.

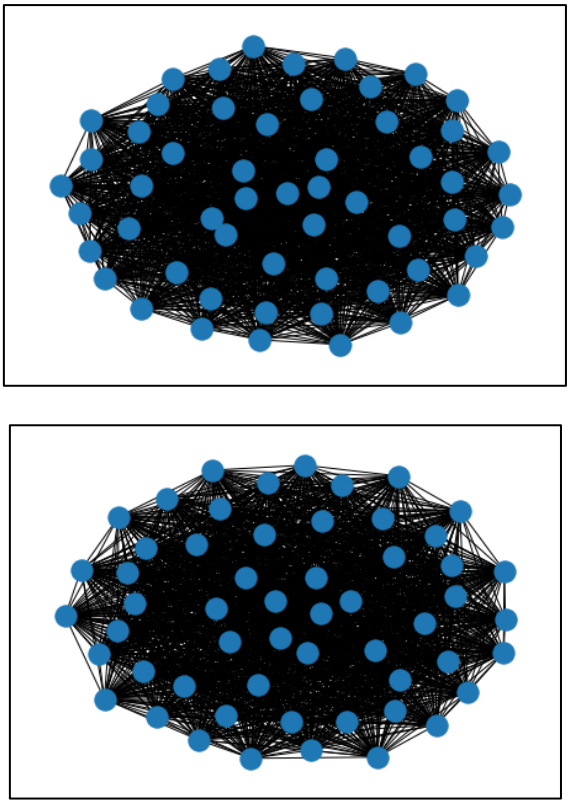

Figure 3.2 Graphs for two different HLA and peptides interactions with inter and intra edges.

Figure 3.2 demonstrates graph representations of interactions between two different HLA and peptide groups. Intra edges represent peptide and HLA connections, whereas enter edges represent peptide and HLA connections. All of the amino acid string characters that have been chosen are represented as nodes. Edges between nodes are explicitly weighted based on their inter and intra nature.

3.3 Feedforward Neural Networks

I have prepared a numerical matrix by converting the physicochemical properties of the amino acids indices database AAindex, the processed HLA and peptide values. The default amino acid string 'ARNDCQEGHILKMFPSTWYV-' is used for matrix value conversion to generate the 12*21 matrix that is passed as the model's input feature. The model predicts the immunogenicity value for the given HLA and peptide combination.

Nodes are permitted between layers n and n+1 in a feedforward network, and no back propagation is permitted between the layers. The nodes of layer n are linked to the nodes of

layer n+1, resulting in a fully connected network. The first layer is the input layer, and its shape is determined by the number of features in the input (Hecht-Nielsen, R., 1990). Each network is made up of multiple hidden layers, each with its own activation function. Kernels and bias exist in the connections between these layers. These values become learnable and can be adjusted during model training. Kernel is also known as weight, and each node usually has a unique value that scales the input and hidden node values. The purpose of bias in the network is to adjust the scaled-up values before they are propagated into the next layer of the network.

In feed-forward neural network for the given HLA and peptide physicochemical properties, the network in the diagram produces binary output. The hidden layers are located between the input and output layers. The network connections are unidirectional and forward connected. Data normalization occurs in the network's hidden layers.

The first hidden layer of the proposed model is used to perform batch normalization on the input data received from the input layer. The weights and bias values of a regular back propagated neural network are updated using back propagation by adjusting the output values and comparing them to the expected output values using gradient descent functions. In this model, batch normalization is typically used to solve the problem of updating hyperparameter values across multiple layers (Ioffe, S. and Szegedy, C., 2015). Normalization of parameters improves network performance by reducing the number of epochs required to train the model to achieve better accuracy and loss values.

Drop rate is introduced in the network in the second hidden layer to address overfit problems that occur when the model is trained on smaller datasets and to avoid statistical noise and generalization errors. This is accomplished by regularizing the nodes in the model and dropping nodes during training. This allows the network to correct the incorrect values propagated from previous layers (Srivastava et al 2014). Because the proposed model is a fully connected network, there is a high likelihood that the network will overfit the data, resulting in inaccurate accuracy and prediction values. With the implementation of the dropout layers, the decision to drop nodes in the network is completely random.

Algorithm 3.3 is the list of the steps for FFNImmuno model. This model constructs a feed forward neural network, takes the physicochemical properties of HLA and Pepptide as input, and predicts immunogenicity values. The input layer contains the same number of features as the input value. Hidden layers are created by combining hidden values and dropout rate values. Every hidden layer is a feed forward layer with an output shape of [None, 32]. Before the final layer, an activation function is added to introduce non-linearity into the output layer. Using the 'adam' optimizer (Zhang, 2018), the model is complied with the input and output values. The experiments are carried out using loss and accuracy as evaluation metrics to assess performance.

ALGORITHM 3.3 : FFNIMMUNO

Input: *HLA and Pepptide`s physicochemical propteries matrix*
Output: *immunogenicity value*

```
Input layer = layers.input(shape = number_features), name = input_fetures
output layer output shape = [None, 2]
For in range (no.of hidden layers)
    Create Feedforward layer (hidden_units, dropout_rate)
    Hidden layers output shape = [None, 32]
Add dense layer with activation function (name = logits)
final layers output shape = [None, 2]
Return model

def run_experiment(model, x input, Y output)
    Compile model (optimizer = Adam)
    loss = Categorical Entropy
    metric = accuracy
    History = model.fit(x,y)
End
```

3.4 GraphImmuno

Graph neural networks perform well on data structures and data sets represented in the form of graphs. These models work by learning the graph's nodes as well as the degree-specific weights of the nodes and the edges between the nodes. The features are learned by the model during training, and the model applies the knowledge gained to the test and validation data for graph prediction and classification, as well as other graph-related tasks.

This supervised graph classification model is made up of the graph classification classes and the dense layer. The adjacency matrices contain the Physicochemical properties of the HLA

and peptides, which are fed into the model. The Physicochemical properties are graph node features and explicit weights assigned to edges based on interactions.

The imported layers from graph classification classes with ‘relu’ activation functions and 64 units. The mean pooling layer efficiently sums up the node representation from the graph to represent the graph. The network has two dense layers with unit sizes of 32 and 16, respectively, as well as 'relu' activation layers. The output layer has one unit, and the activation is 'sigmoid.' The sigmoid function is a non-linear function that produces an output between 0 and 1 by adding all the input weights.

$$\sigma(x) = 1/(1+\exp^{(-x)}) \qquad (3.1)$$

Equation 3.1 gives the equation for sigmoid activation function (logistic sigmoid activation function). Activation functions are also known add transfer functions. The above equation is a continuous function with domain range (-∞, +∞). The functions range values are between 0 and +1 and σ(0) is 0.5. The functional exponential in nature. As the x value increases “e^-x” value becomes zero making the value of σ(x) as 1. Likewise, as the x value decreases the denominator becomes infinity and the value of σ(x) as 0.

Algorithm 3.4 is the list of the steps for GraphImmuno model. The model receives data in graph format, including nodes, edges, and weights. The model outputs a vector representation of the input data. In an iteration, the weights are aggregated and assigned to the edges. Convolution layers and pooling layers are combined as hidden layers to form a graph convolution model. The optimizer is used to compile the model. Loss and accuracy are mentioned as model evaluation metrics. As the test data are being fitted to the model, log values are streamed into the history parameter. The prediction function of the model predicts the value for the test input value.

ALGORITHM 3.4 : GRAPHIMMUNO

Input: *Graph G(V, E); input features {xv, ∀v ∈ V}; depth K; weight matrices Wk, ∀k ∈{1, ..., K};*
Output : *Vector representations zv for all v ∈ V*

```
    h ← xv, ∀v ∈ V
1       for k = 1...K
2           for v ∈ V
3               h ← AGGREGATEk({h (k−1, u) , ∀u ∈ N (v)})
4               h k v ← σ  Wk · CONCAT(h k−1 v , h k N(v) )
5           End
6           H(k, v) ← h(k,v)/||h (k,v)||2, ∀v ∈ V
7       End
8   Z(v) ← h (K v) , ∀v ∈ V

9   gc_model = GCNSupervisedGraphClassification(layer_size, activations, generator, dropout)
10  Input,output = x_inp, x_out = gc_model.in_out_tensors()
11  for in range (no.of hidden layers)
12      Add dense layer(units, activation function)(layer)
13  End
14  Model = .model(input, output)
15  Model.compile(optimizer, loss, metrics)
16  for i in no.of sets
17      train_gen = generator.flow(fold[i],targets, batch size)
18      test_gen = generator.flow(fold[i],targets, batch size)
19  History = Model.fit(train, test , epochs, class weights)
20  Predction = Model.predict(test data set)
End
```

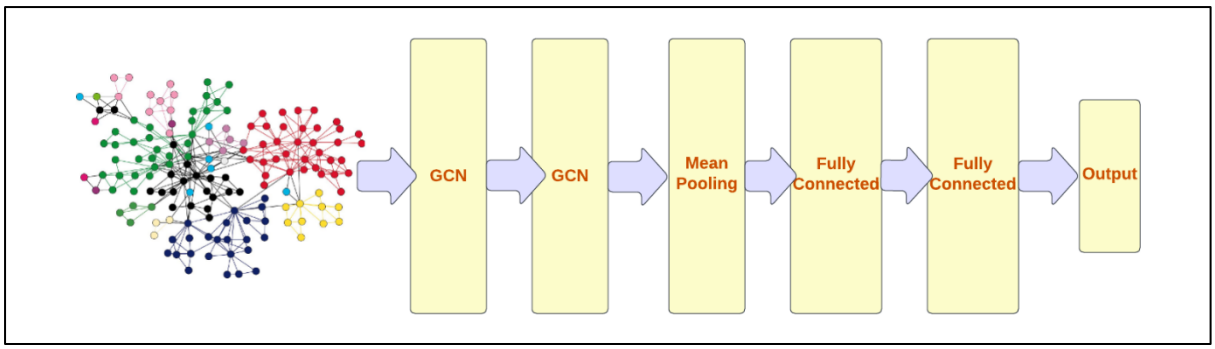

Figure 3.3 GraphImmuno - graph classification model.

Figure 3.3 is the block diagram for GraphImmuno model. The model is fed graph data in the form of an adjacency matrix. The first few layers are convolutional graph layers that decode the graph data before processing it. The information is routed through a mean pooling layer. The data in the pooling layer is represented as a graph by learning node representation. The data is then passed through fully connected layers to perform the supervised classification. The output layer is the final layer in the network and is responsible for binary classification.

3.5 DeepGraphImmuno

DeepGraphImmuno is a GraphImmuno model that has been enhanced with additional layers to perform sort pooling and embedding to produce a representation of the graph data. The extraction of feature information from structured datasets is critical for graph-based neural networks. Sort pooling is a generic pooling method for aggregating graphs' given structured feature data into fixed dimensional representations (Naderializadeh, N. 2021). Because high-dimensional vectors are translated into low-dimensional space, this is also known as embedding.

The adjacency matrix and node features of the graph serve as input to the model, which will be processed through the first four convolutional layers. The difference between these convolutional layers and the convolutional layers in the previous model is that, these layers use adjency normalized form. These layers are embedded with the "tanh" activation function and have units of 32, 32, 32, and 1 respectively. Except for the last layer, the dense layers in the network have "relu" as the activation function. The final output layer has a "sigmoid" function. The drop layer in the model's purpose is to prevent overfitting.

ALGORITHM 3.5: DEEPGRAPHIMMUNO

Input: *Graph G(V, E); input features {xv, ∀v ∈ V}; depth K; weight matrices Wk, ∀k ∈{1, ..., K};*
Output : *Vector representations zv for all v ∈ V*

```
dgc_model = GCNSupervisedGraphClassification(layer_size, activations, generator, dropout)
    add conv1D (filters, kernel_size)
    add maxpool1D (pool size)
    add conv1D (filters, kernel_size)
    add flatten(x_out)
    add dense layer (units, activation)
    add dropout layer (dropout rate)
Prediction = dense(units = 1, activation = 'sigmoid')
```

End

Algorithm 3.5 lists the steps for DeepGraphImmuno model. The model accepts graph data in the form of adjacency matrices and node feature matrices and returns the vector representation of the given graph data. The supervised classification model builds the deep neural network layer by layer. The first layers are graph convolution layers. They are used in the network to perform normalization. The representation is made by adding a layer for sort pooling. This pooling layer is followed by a conventional layer, flatten layers, dense layer, and dropout layer. The final layer is the output layer, which has an activation function for non-linearity in the output value.

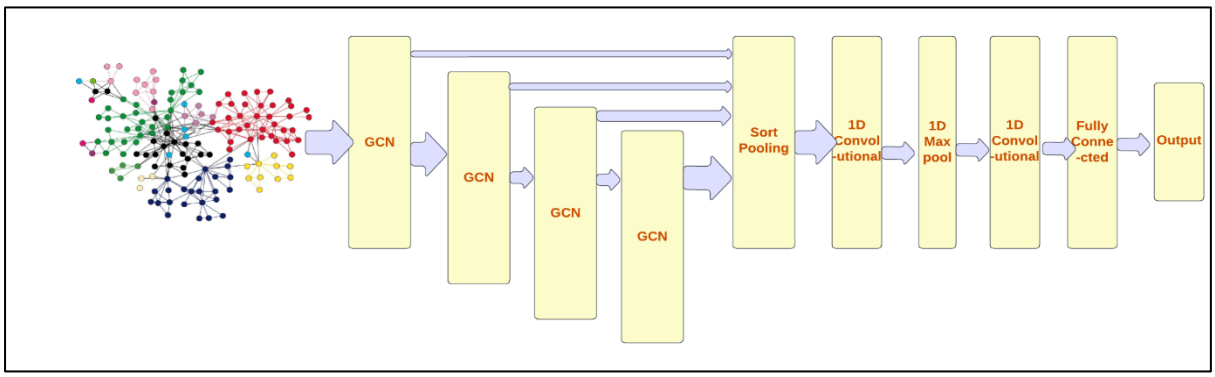

Figure 3.4 DeepGraphImmuno - graph classification model sort pooling layer.

Figure 3.4 illustrates the block diagram for DeepGraphImmuno. This model is empowered with sort pooling layer. The first four layers are convolutional layers that take the adjacency and feature matrices as input. The values from these layers are fed into the sorting layer,

which embeds them. The normalized data is received from this layer and passed to the sequence's single convolution layers. The network's dense layers perform binary classification. Non-linear activation function for classification is integrated into the output layer.

3.6 Embedded – GraphImmuno

The previous classification models are supervised classification models, which means that the model learned from the training dataset and then performed classification tasks on the validation and test datasets. Embedded - The GraphImmuno model performs graph classification tasks autonomously (Bai, Y., Ding, H., Qiao, 2019). This model can also be used to compute embedding vectors because it uses ground truth distance between graphs to classify the vectors. Finding the minimum subgraphs is another method for implementing unsupervised graph classification, but it is a computationally expensive method.

Laplacian spectrum distance is used to determine the similarity between graphs and to calculate the distance between graphs. For a connected graph G, Laplacian distance L = Diag(Tr) D, where D is the distance matrix and Diag(Tr) is the diagonal matrix. Embedded - GraphImmuno encodes graph data into embedding vectors and uses a generator to compute the distance and similarities between graphs.

Algorithm 3.6 shows the procedure for Embedded - GraphImmuno model. This unsupervised classification model accepts graph adjacency matrices as inputs. The model employs any notion of distance between the graphs. The distance between eigenvalues in the preceding model is calculated using a Laplacian matrix. The pair model is built by combining the data from both graphs and the vector distance. Pair model trains on data using the given optimizer and loss values. The graph convolution classification layer's in and out tensors are embedded in a model. The embedding model predicts the classification of any two graphs. Graph distance computes the laplacian distance between two graphs by taking the shortest distance between the spectrums into account.

ALGORITHM 3.6: EMBEDDED – GRAPHIMMUNO

Input: *Graph G(V, E); input features {xv, ∀v ∈ V}; depth K; weight matrices Wk, ∀k ∈ {1, ..., K}*
Output : *Vector representations zv for all v ∈ V*

```
usgc_model = GCNSupervisedGraphClassification(layers, activation, generator)
pair_model = Model(tensor1 + tensor2, vec_distance)

inp, out = gc_model.in_out_tensors()

embedding_model = Model(inp, out)

pair_model.compile(optimizer, loss)

embeddings = embedding_model.predict(graphs)


def graph_distance (graph1, graph2)
    spec1 = laplacian_spectrum(graph1)
    Spec2 = laplacian_spectrum(graph2)
    distance = min(len(spec1), len(spec2))
return normalized(distance)


#Training the model

pair_model.compile(optimizer, loss)

history = pair_model.fit(train, epochs, verbose=0)
```

End

Chapter 4

Performance Evaluation of GraphImmuno for Immune Response

In this section, I assess the performance and the effectiveness of the proposed models: GCN supervised graph classification, feedforward network, DeepGCN supervised graph classification and unsupervised graph classification. This section includes information about the model results analysis as well as some background information about the main attributes that contributed to the model's creation. The loss and accuracy values for the models explains the efficiency of the models. Recall measure indicates the model`s ability for the data validation done over different disease datasets. Additionally, the discussion of the model's performance explains the values portrayed in graphs and tables.

4.1 Hardware

The hardware specifications for the experiment are as follows: Intel(R) Xeon(R) CPU with 2.30GHz and 13GB of RAM. The experiments run on Colab resources with a GPU Tesla T4 and a 2.30GHz CPU, as well as a 33GB HDD. GOOGLE COLAB's provides *NVIDIA-SMI 460.32.03*, *Driver Version: 460.32.03* and *CUDA Version: 11.2.* Colab`s RAM space is 12,6 GB and Disk space is 40GB. The number of available slots for physical processors are 1 socket and the number of cores for each processor is 1.

4.2 Software

To implement the models, I used Tensorflow, PyTorch, RDkit, NetworkX, DGL, and Molecule. These libraries help to display the data as graphs and to process the information. On top of already-existing DL frameworks, the DGL - Deep Graph Library Python module is created for simple implementation of the graph neural network model family. It currently supports TensorFlow, MXNet, and PyTorch (Jamasb et al., 2020). A Python library called Networkx aids in the visualization of networks and graphs. This library allows for the creation and manipulation of graphs as well as the study of dynamics and functions (Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008)). StellarGraph library helps in discovering the patterns about to solve the graph related problems such as classification and link or node prediction using the structured graph data (Data61, C. (2018)). StellarGraph is built with TensorFlow 2 and the Keras high-level API, as well as Pandas and NumPy, making it simple to embed and implement.

4.3 Performance Metrics

4.3.1 Accuracy

The number of accurate predictions obtained is the accuracy score. Loss values are those that show a deviation from the desired target state (s). gives insight into how well a machine does at making accurate value predictions.

Accuracy = (True positives +True Negatives) / (Positives +Negatives) *(4.1)*

Equation 4.1 is the formula for calculating the accuracy value. The number of correctly predicted data points out of all data points determines accuracy. The number of true positives and true negatives divided by the total number of true positives, true negatives, false positives, and false negatives would be a more precise definition. In biomedical data, a better understanding of the model's performance and information about false positives are critical.

4.3.2. Precision

Precision provides the impact of false positives results in the model. Lower precision shows the model performance is degraded due to false positives. Precision indicates the percentage of positive identifications are correct.

Precision = True positive / (True Positive + False Positive) *(4.2)*

Equation 4.2 is the formula to calculate precision value. The denominator is the sum of true and false positives, which is the total positives value, whereas the numerator is only true positives.

4.3.3. Recall

Recall shows the impact of false negatives on the model. Lower recall indicates high false negatives. Recall tells what proportion of the actual positive class got correctly classified.

Recall = True Positive / (True positive + False Negative) *(4.3)*

Equation 4.3 gives out the recall value. It contributes to determining the model's ability to classify positive samples. The sum of true positives and false negatives is the denominator.

4.3.4. Receivers operating characteristics (ROC)

AUC-ROC is calculated by graphing true positive versus false positive values at different discriminating thresholds. It is a metric widely used in binary classifiers. When AUC = 1, the classifier can correctly differentiate among all Positive and Negative class points. When the AUC is zero, the classifier will forecast all Negatives and all Positives completely incorrectly.

4.3.5 Mean squared error (MSE)

The mean squared error is a risk function that describes the deviation of data points from the regression line or regression plane. The lower the value of the error, the better the model performed, and absolute zero is the ideal value for this measure. Because the MSE value is calculated by squaring the sum of the deviations, it can never be negative. The only stipulation of this measurement is that it has no upper bound and thus can gauge the model's performance in all terms.

$$MSE = 1/n \Sigma (y - y\tilde{})^2 \qquad (4.3)$$

Equation 4.4 calculates the value of MSE. It can be defined the sum of squared difference between the actual and predicted values. In equation 4.4 n is the number of items and Σ denotes summation. y is the actual value and y~ is the predicted value.

4.3.6 Log Loss

Log loss is based on probability estimates and is also regarded as logistic regression loss or cross-entropy loss. It is frequently used in logistic regression of nature multinomial and neural networks. Different versions of expectation-maximization is utilized to assess classifier probability outputs instead of distinct projections (Pedregosa, F(2011)).

4.3.7 Optimizer

The Adam optimization technique is a stochastic gradient descent extension that updates network weights iteratively based on training data (Brownlee, J., 2022). The Adam optimizer has the advantages of being simple to use and computationally effective. Adam Optimizer consumes extremely low RAM. It is appropriate for problems with large amounts of data and/or parameters because the gradients are invariant to diagonal rescaling. It is also appropriate for non-stationary applications. This optimizer is appropriate for situations with very noisy or sparse gradients because hyper-parameters have a clear meaning and often require little modification.

The Adam optimizer combines stochastic gradient descent, momentum, and RMSprop. Similar to SGD with momentum by taking into account the gradients' exponentially weighted average, it modifies the learning rate with squared gradients, like RMSprop, and uses momentum by using the gradient average rather than the gradient itself, like SGD (Jais, I.K.M. et ai., 2019).

4.4 Model Parameters (Hyperparameters)

Below are the hyper parameters used to tune the model for optimized performance.

Hyper parameter	Value
layer_sizes	[64, 64]
Activations	Relu and Sigmoid
dropout_rate	0.2
num_epochs	100
batch_size	256

Table 4.1 Hyperparameters of GCN.

Table 4.1 describes the neural network model hyperparameters used in the implementation. The layer size indicates the number of neurons in a layer. In the network, activation functions are used to remove linearity in the output layer and to determine whether or not the neuron should be active. Drop rate helps to thin the network during training and prevents overfitting. The number of epochs determines the total number of iterations on the training data. Batch size determines the total number of values that can be processed at once through the network.

The GCNSupervisedGraphClassification's first two layers are made up of Graph Convolutional layers, each with 64 units and relu activations. The following layer, the mean pooling layer, compiles the learned node representation into a graph representation. The graph representation is fed into two completely linked layers with 32 and 16 units and relu activations, respectively. The output layer, with a single unit and sigmoid activation, is the final layer (Zhang, M. et al 2018). The model is trained over 100 epochs with a drop rate of 0.2 to avoid the over fitting of the model.

4.5 DeepImmuno as Baseline

DeepImmuno-CNN predicts the immunogenicity of MHC-peptide pairs using a traditional convolutional neural network (CNN). In contrast to existing techniques for predicting immunogenicity, the confidence of the experimental immunogenicity substantiation in the training dataset is used to preferentially measure the combinations of the peptide and the MHC complex used in the model as per the confidence level. To address the sparsity issues inherent in one-hot encoding, each amino acid sequence is encrypted using dimension reductionality method principal component analysis. Total of 566 features of amino acid`s physicochemical properties taken from the AAindex1 source DB are processed to create the dataset.

A 10-fold cross-validation conducted on the IEDB dataset to validate the effectiveness of the DeepImmuno-CNN model. Certain tested algorithms require an absolute threshold (0.5) to predict whether a peptide is immunogenic or non-immunogenic. DeepImmuno CNN was found to be extremely steady, with a good score of area under the ROC curve of 0.85 and area under the precision-recall curve value of 0.81 for each fold of the 10 folds. However, CNN fails to perform the classification and prediction tasks based on the HLA and peptide structure and amino properties.

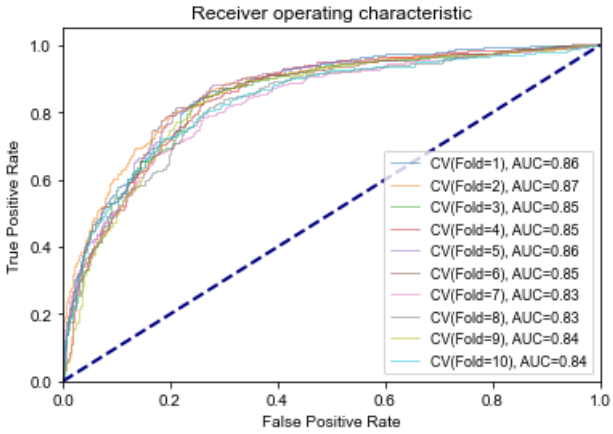

Figure 4.1 ROC of DeepImmuno model CNN with K10 fold validation.

Figure 4.1 depicts the ROC values for the DeepImmuno model with K10 fold validation over the IEDB database. The threshold value of 0.5 was chosen to determine whether a peptide is immunogenic or not. The graph shows that the average AUC for ten folds is 0.85. The values appear to be stable across all iterations, with only minor variations in AUC values for few folds.

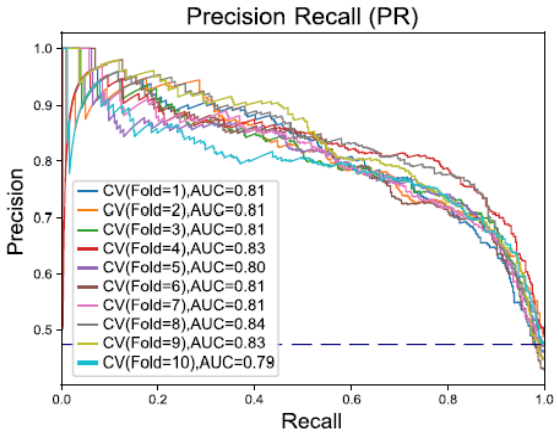

Figure 4.2 Precision vs Recall of DeepImmuno-CNN with 10-fold validation.

Figure 4.2 depicts the precision vs recall for the DeepImmuno model with CNN and 10-fold validation on the IEDB database. The model's average auPR value across 10 folds is 0.81. Precision indicates the percentage of related results, while recall indicates the percentage of total correctly classified related results.

4.6 Results and Analysis of FFNImmuno

The keras-based feed forward network is trained on the HLA and Peptide datasets. The "physicochemical-aware encoding technique" characteristics are encoded to the HLA and peptide strings. The encoded values data frame is split into train and test dataset to predict the immunogenicity values for the given HLA and peptide combination.

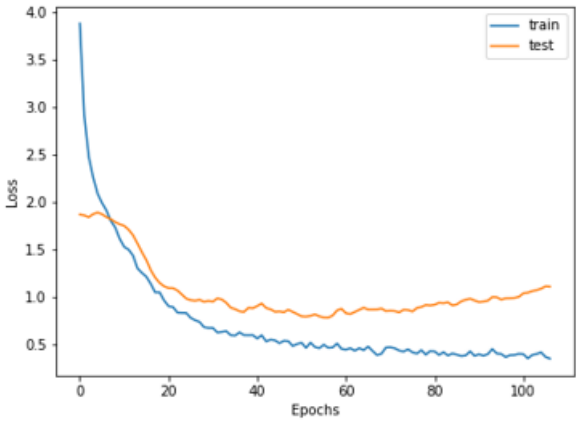

Figure 4.3 Loss value vs epochs for FFNImmuno.

Figure 4.3 shows the FFN loss values for both the train and test data sets over 100 epochs. Loss values clearly decrease during the 20th epoch and converge by the 40th epoch. The network's unidirectional nature explains the early convergence. In contrast to a traditional network, the network's connections do not form a circle (Ramasubramanian, K. and Singh, A., 2019). Because the data values are propagated forward rather than backward through multiple hidden nodes, the convergence occurs during the early epochs and the loss values converge as expected.

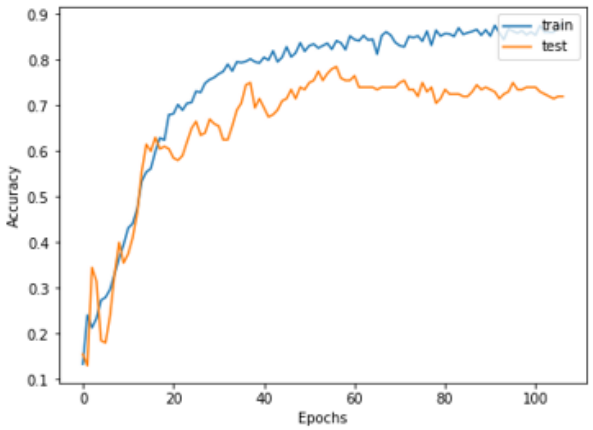

Figure 4.4 Accuracy value vs epochs for FFNImmuno.

In terms of convergence, the accuracy values in figure 4.4 follow the same pattern as the loss diagram. The convergence occurs early, and the values have stabilized by the 40th epoch. The model learns the trainable parameters more quickly because of the low drop rate and learning rate values.

4.7 Results and Analysis of GraphImmuno

The data converted into graph format by stellargraph's mapper "PaddedGraphGenerator" is processed by a fully connected graph convolution neural network. 70% of the graph data is set aside for training, with the remainder set aside for testing. The network's mean pooling layer pools the graph data for classification, and the results indicate that the model learns about the graph structures and features efficiently during the initial epochs and achieves a decent accuracy value during the first few rounds of training. The difference between train and test values represents the model's ability to learn on a graph-represented dataset.

With GCN supervised graph classification, the model performed well overall but with a significant variance after 10-fold cross validation. The small dataset size is most likely to be responsible for the high variance and the presence of noise in the data could be the other reason for variance. Overall, the performance is marginally lower than that of FNN. However, because I did not tune the model to achieve the best possible performance, some improvement over the current baseline is possible. This model finishes a classification task that previous models had fails to finish. This straightforward GCN with mean pooling graph classification model is advantageous in comparison to graph kernel-based methods.

Hyper parameter	Value
layer_sizes	[64, 64]
Activations	Relu and Sigmoid
dropout_rate	0.2
num_epochs	100 and 500
Generator	PaddedGraphGenerator
batch_size	50

Table 4.2 GCNSupervisedGraphClassification layer`s hyper parameter values.

Table 4.2 describes the hyper parameters used in GCNSupervisedGraphClassification layer of GraphImmuno model. The first GCN layers are [64, 64] in length, with Relu serving as the activation layer. The final output layer employs the Sigmoid activation layer. Unsupervised models are run for 500 epochs, while supervised models are run for 100. A minimum drop rate of 0.2 is assigned to the model to prevent data overfitting during data propagation from one layer to the next. The model has a batch size of 50.

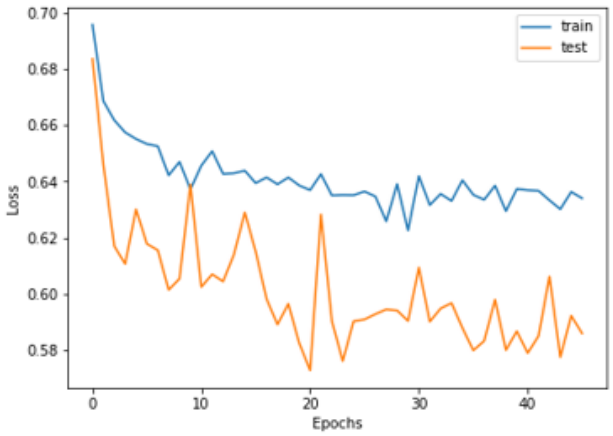

Figure 4.5 Loss value vs epochs for GraphImmuno.

Figure 4.5 shows the loss value for the GraphImmuno model after 40 epochs. As seen in the graph, the test loss value dropped rapidly after the 10th epoch, and the values are low when compared to the train loss values. The loss values begin at a high of 0.70 and gradually decreases to 0.58 over the epochs. According to the graph, convergence for both train and test loss values occur after the 20th epoch.

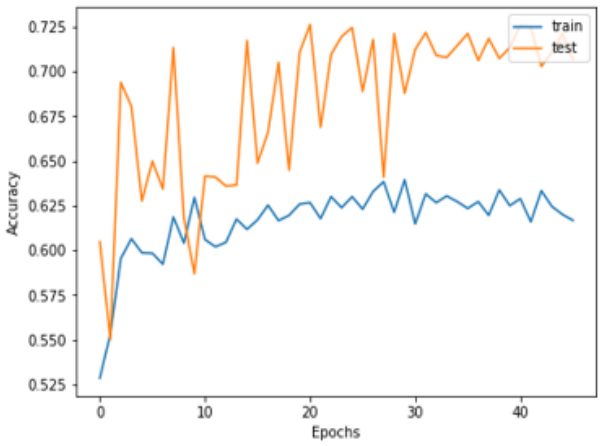

Figure 4.6 Accuracy value vs epochs for GraphImmuno.

Figure 4.6 shows the accuracy value for the GraphImmuno model after 40 epochs. The graph shows that the train accuracy value stabilized after the 10th epoch and that the values are relatively low in comparison to the test accuracy values. The accuracy values begin at 0.525 and gradually increase to 0.725 over the epochs. According to the graph, convergence for both train and test accuracy occur after the 20th epoch. The accuracy values of the train are more stable than the accuracy values of the test. The test accuracy is more unsteady, but the deviations are small. It can be assumed that as the number of epochs increases, the loss and accuracy values will improve.

4.8 Results and Analysis of GraphImmuno over disease datasets

The graphs generated by StellarGraph's "padded graph generator" are trained using the graph classification model created by StellarGraph's GCNSupervisedGraphClassification layer. The model is run for 100 epochs and 10 folds, with the loss values dropping significantly as the model learned the features. The loss value of the test sets is noticeably lower than the loss value of the train data.

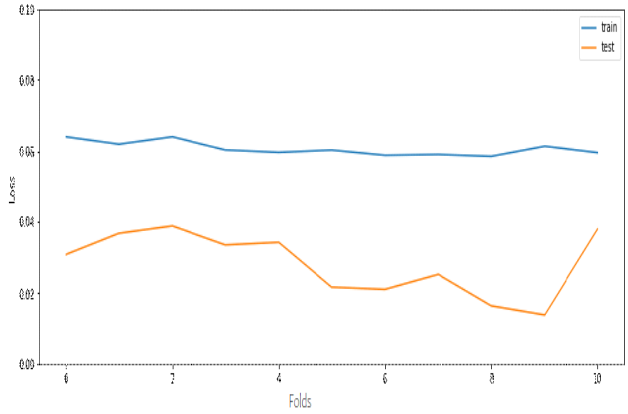

Figure 4.7 Loss value vs folds for GraphImmuno.

Figure 4.7 depicts the Loss value for GraphImmuno model k-fold validation. The model is trained over ten folds for validation to ensure less variation in the result and to reduce bias and overfitting. The graph shows that the average loss for the train dataset is 0.06, while the average loss for the test dataset is 0.03. Despite a few oscillations in the test loss, the train loss remains constant throughout the course. The model performed admirably, as evidenced by the extremely low loss values.

Dataset	**Measurement**	**Value**
Validation dataset	Mean Squared Error	0.195
Dengue dataset	Accuracy	0.975
Cell data set	Recall	0.857
Covid data set (Convalescent)	Recall	0.920
Covid data set (Unexposed)	Recall	0.875

Table 4.3 GraphImmuno measurement values over disease datasets.

Table 4.3 shows various measurement values for the GCNSupervisedGraphClassification model across various disease datasets. The accuracy value for the Dengue dataset is 0.97, indicating that the model learned the features and predicted immunogenicity very accurately. Even though the model's recall values are superior to those of Cell and Covid, the model experienced shortcut learning. The inability to perform well on different kind of data even though the model performs well on the known dataset is because of the shortcut learning (Geirhos et al. (2010)). Because of the shortcut learning, all of the predictive values are less than 0.5. This is because the graph's implied weight assignments may not fully reflect the true peptide-MHC interactions, resulting in ambiguous results.

4.9 Results and Analysis of DeepGCN – GraphImmuno

DeepGCN - GraphImmuno model updates the GCN - GraphImmuno model with sort pooling and performs a supervised graph classification task. The model is trained to predict the label of previously unseen graphs using a collection of graphs with a categorial label as input. Figure 4.8 depicts the model's loss value over 10 epochs, while Figure 4.9 depicts the accuracy values over the same number of epochs.

Because of the additional layers, the model required more epochs to converge the loss and accuracy values than the GCN - GraphImmuno model. However, the additional layers did improve the accuracy. The network's convolutional and max pools receive the output from the sort-pool layer and process the graph data before sending it to the fully connected layer. As a result, the loss and accuracy values gradually change. The improved accuracy of the Supervised classification model reached 0.76 over 100 epochs and can be improved with a larger dataset, additional training, and parameter tuning.

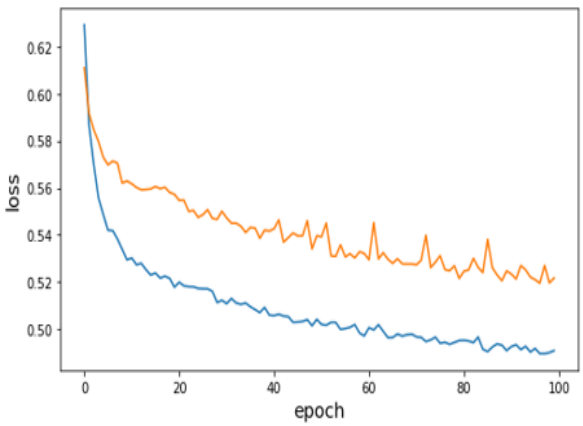

Figure 4.8 Loss value vs epochs for DeepGCN – GraphImmuno.

Figure 4.8 depicts the loss value for the DeepGCN - GraphImmuno model over 100 epochs. The blue line represents the loss for the train data set, while the orange line represents the loss for the validation dataset. For both sets, the loss value begins at a high of 0.62 and gradually decreases to 0.50. The training set performs better because its loss values were lower.

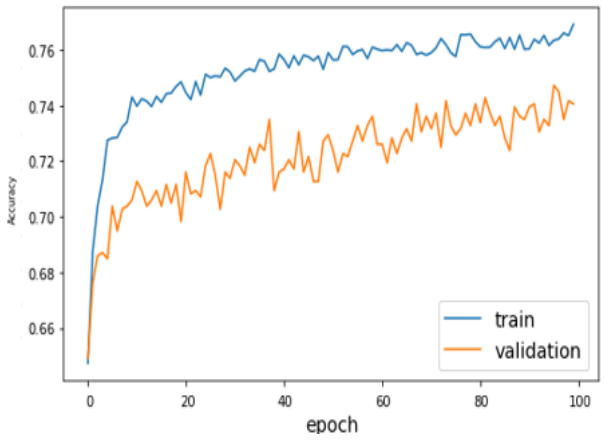

Figure 4.9 Accuracy value vs epochs for DeepGCN – GraphImmuno.

Figure 4.9 depicts the accuracy values for the DeepGCN - GraphImmuno model over 100 epochs. The accuracy of the supervised classification model was 0.74 for validation data and 0.76 for train data. During the 0^{th} epoch, both values had accuracy values of 0.66, and convergence occurred after the 40th epoch. The spike in accuracy occurs during the first 5^{th} epoch itself. Because of the sort-pooling methods and large number of epochs, the accuracy values are higher than in the GraphImmuno model.

4.10 Results and Analysis of Embedded – GraphImmuno

The unsupervised graph classification model "Embedded - GraphImmuno" is run on the StellerGraph-prepared graph data. This model computes embedding vectors as well as graph representations. The graphs are encoded into embedding vectors by the model. The graphs are classified by the model based on their ground truth distance.

The loss for the unsupervised graph classification model over 500 epochs is depicted in Figure 4.10 The model needs initial 100 epochs to converge the values and to learn about the data and features. Later, for the rest of the epochs, the loss value didn't change much because it was nearly zero. However, because the features are not labeled, the model requires only a few initial epochs. Figure 4.11 depicts a qualitative measure of the embeddings obtained using the dimensionality reduction model t-distributed stochastic neighbour embedding (tsne). Because of the reduced dimensions, it can be assumed that the embeddings made from the HLA and peptide using the "physicochemical-aware encoded properties" have very close embeddings and overlaps.

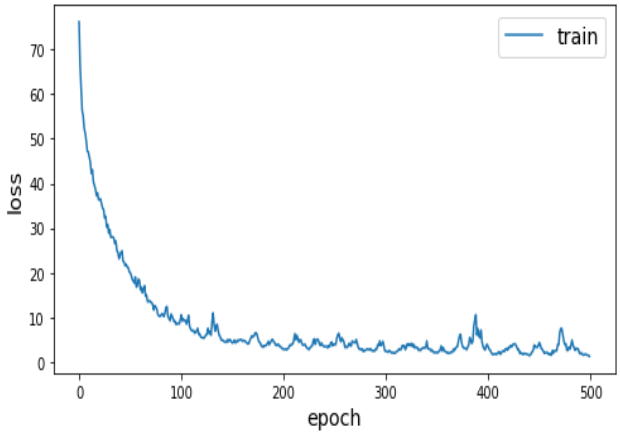

Figure 4.10 Loss value vs epochs for Embedded - GraphImmuno.

Figure 4.10 illustrates the loss value for the Embedded - GraphImmuno model over 500 epochs. The loss value of this unsupervised classification model converges after the first 100 epochs. The loss values began at 70 and eventually falls close to zero by the final epoch. With the exception of sporadic changes, the loss values remain unchanged after the 200th epoch. The graph's convergence indicates that the model learned the parameters quickly and performed well.

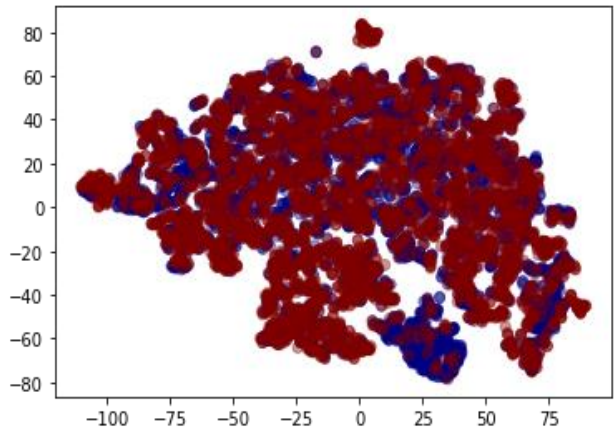

Figure 4.11 Visualisation of the embeddings displaying the qualitative measure of the embeddings.

Figure 4.11 visualises the embeddings in the qualitative measure achieved by the GCN unsupervised classification model. The overlapping of the embeddings is caused by the data's dimensionality reduction. The two distinct colors represent two distinct immunogenicity values for the combination of HLA and peptide values.

4.11 Discussion of the Results

Following a thorough analysis of the results of FFNImmuno, it can be said that FNNs speeds up the learning process. The early convergence of loss and accuracy values confirms that FNN models quick learning abilities. The nature of the network with forward propagation without back propagation helped the model to accelerate the learnings. Furthermore, it is clear that the GraphImmuno model outperformed the DeepImmuno model in terms of prediction values across disease data sets. The GraphImmuno model's accuracy of 97.5 over Dengue data and recall values of 92.0 and 87.5 over Covid's convalescent and unexposed datasets indicate that the graph convolutional neural network with physicochemical properties improves immunogenicity prediction.

Furthermore, the graphical representation of the peptides and HLA values enables higher accuracy graph learning tasks such as supervised graph classification and unsupervised graph classification. The GraphImmuno is performs supervised classifications with an accuracy value of 0.725. This exhibits the ability of performing graph tasks for the physicochemical properties' values. The DeepGraphImmuno model's accuracy and learning rate were improved by the addition of additional convolutional layers and sort pooling to 0.77. Embedded – GraphImmuno does unsupervised classification of the graphs using the embedding technique and achieves low loss values over a smaller number of epochs.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

Understanding T-cell immunity aids in the development of new antibiotics and vaccines that will aid in the strengthening of the human immune system. As a result, there is a greater interest in understanding the functionality of the peptides that will elicit t-cells, as well as the interaction between the peptides and HLA. Though there are a few existing ML and AI models using convolutional networks to predict immunogenicity for peptides, graph models are best at representing the molecules. Therefore, in this thesis, I addressed these questions using the proposed model GraphImmuno. GraphImmuno is a graph convolutional neural network with amino acid physicochemical properties.

Physicochemical properties are extracted from the AAindex database and PCA is used to reduce the dimensions. These peptide and amino acid properties are converted into an adjacency matrix, and the data is converted into graph format for processing with graph libraries. The peptide and HLA amino acids are represented as nodes, and the connections between the HLA and peptide, as well as the connections within, are represented as edges. Convolution, pooling, and fully connected hidden layers are followed by an output layer in a graph neural network. On the adjacency matrices of the feature values of the nodes, GraphImmuno predicts immunogenicity values and performs supervised and unsupervised classification tasks.

After reviewing the research results, it is clear that GraphImmuno learned about the correlations and interactions that exist between peptides and HLAs. GraphImmuno also performed graph learning tasks such as supervised and unsupervised classification. GraphImmuno was validated on a dengue dataset and achieved an accuracy of 0.975, with a recall value of 0.92 for Covid convalescent. Furthermore, after a few initial epochs, the GraphImmuno loss value plummeted dramatically. This indicates that the graph model quickly learned about the feature values. By adding a sort pooling layer and more convolutional layers to the GraphImmuno model, it can now perform unsupervised classification. DeepGraphImmuno outperformed GraphImmuno in the supervised classification task. GraphImmuno with embedding performed unsupervised classification. The conclusion could be that GraphImmuno answered all of this thesis' proposed research questions.

5.2 Future Works

The GraphImmuno model addresses the thesis's proposed question. Still, there are a few future extensions that could be done with this model. Some of these potential future developments are listed below.

- Due to time constraints, the GraphImmuno model is not used for node and edge prediction tasks. In the future, a node data collection will be created using HLA and peptide amino acid values, and the node data values will be used for node and edge prediction and classification tasks.

- The GraphImmuno model does not investigate the structural properties of peptides and HLA. These aspects can also be incorporated into the component graph designs.

- The weights of the graphs are currently explicitly assigned. These values can be calculated using the properties of the molecules' connections. The model may then be able to predict the outcomes more accurately.

- The model's performance on a previously unseen dataset can improve if it is trained with a variety of data values. There is currently a scarcity of high-quality analytical data in the medical and biological sectors.

- The output values of GraphImmuno immunogenicity prediction are currently binary. However, in the real world, immunogenicity values come in a variety of classes. As a result, the GraphImmuno model can be improved to produce multi-class output values.

- There is no user interface portal for accessing the GraphImmuno model. Enabling a UI portal by deploying the model on docker can improve the model's accessibility.

References

Dobson, C.M. (2003). Protein folding and misfolding. Nature, 426(6968), pp.884–890. doi:10.1038/nature02261.

Li, G., Iyer, B., Prasath, V.B.S., Ni, Y. and Salomonis, N. (2021). DeepImmuno: deep learning-empowered prediction and generation of immunogenic peptides for T-cell immunity. Briefings in Bioinformatics, 22(6). doi:10.1093/bib/bbab160.

Malone, B., Simovski, B., Moliné, C., Cheng, J., Gheorghe, M., Fontenelle, H., Vardaxis, I., Tennøe, S., Malmberg, J.-A., Stratford, R. and Clancy, T. (2020). Artificial intelligence predicts the immunogenic landscape of SARS-CoV-2 leading to universal blueprints for vaccine designs. Scientific Reports, 10(1). doi:10.1038/s41598-020-78758-5.

Barra, C., Ackaert, C., Reynisson, B., Schockaert, J., Jessen, L.E., Watson, M., Jang, A., Comtois-Marotte, S., Goulet, J.-P., Pattijn, S., Paramithiotis, E. and Nielsen, M. (2020). Immunopeptidomic Data Integration to Artificial Neural Networks Enhances Protein-Drug Immunogenicity Prediction. Frontiers in Immunology, 11. doi:10.3389/fimmu.2020.01304.

Doneva, N., Doytchinova, I. and Dimitrov, I. (2021). Predicting Immunogenicity Risk in Biopharmaceuticals. Symmetry, 13(3), p.388. doi:10.3390/sym13030388.

Dimitrov, I., Zaharieva, N. and Doytchinova, I. (2020). Bacterial Immunogenicity Prediction by Machine Learning Methods. Vaccines, 8(4), p.709. doi:10.3390/vaccines8040709.

Cotugno, N., Santilli, V., Pascucci, G.R., Manno, E.C., De Armas, L., Pallikkuth, S., Deodati, A., Amodio, D., Zangari, P., Zicari, S., Ruggiero, A., Fortin, M., Bromley, C., Pahwa, R., Rossi, P., Pahwa, S. and Palma, P. (2020). Artificial Intelligence Applied to in vitro Gene Expression Testing (IVIGET) to Predict Trivalent Inactivated Influenza Vaccine Immunogenicity in HIV Infected Children. Frontiers in Immunology, 11. doi:10.3389/fimmu.2020.559590.

Lee, E.K., Nakaya, H.I., Yuan, F., Querec, T.D., Burel, G., Pietz, F.H., Benecke, B.A. and Pulendran, B. (2016). Machine Learning for Predicting Vaccine Immunogenicity. Interfaces, 46(5), pp.368–390. doi:10.1287/inte.2016.0862.

Bezu, L., Sauvat, A., Humeau, J., Gomes-da-Silva, L.C., Iribarren, K., Forveille, S., Garcia, P., Zhao, L., Liu, P., Zitvogel, L., Senovilla, L., Kepp, O. and Kroemer, G. (2018). eIF2α phosphorylation is pathognomonic for immunogenic cell death. Cell Death & Differentiation, 25(8), pp.1375–1393. doi:10.1038/s41418-017-0044-9.

Humeau, J., Sauvat, A., Cerrato, G., Xie, W., Loos, F., Iannantuoni, F., Bezu, L., Lévesque, S., Paillet, J., Pol, J., Leduc, M., Zitvogel, L., de Thé, H., Kepp, O. and Kroemer, G. (2020). Inhibition of transcription by dactinomycin reveals a new characteristic of immunogenic cell stress. EMBO Molecular Medicine, 12(5). doi:10.15252/emmm.201911622.

Marciniak, A., Tarczewska, M. και Kloska, S. (2020) 'Predicting immunogenicity in murine hosts with use of Random Forest classifier', Zeszyty Naukowe. Telekomunikacja i Elektronika / Uniwersytet Technologiczno-Przyrodniczy w Bydgoszczy, nr 24, pp. 31–43(2020).

Schaap-Johansen, A.-L., Vujović, M., Borch, A., Hadrup, S.R. and Marcatili, P. (2021). T Cell Epitope Prediction and Its Application to Immunotherapy. Frontiers in Immunology, 12. doi:10.3389/fimmu.2021.712488.

Smith, C.C., Chai, S., Washington, A.R., Lee, S.J., Landoni, E., Field, K., Garness, J., Bixby, L.M., Selitsky, S.R., Parker, J.S., Savoldo, B., Serody, J.S. and Vincent, B.G. (2019). Machine-Learning Prediction of Tumor Antigen Immunogenicity in the Selection of Therapeutic Epitopes. Cancer Immunology Research, 7(10), pp.1591–1604. doi:10.1158/2326-6066.cir-19-0155.

Aranha, M.P., Jewel, Y.S.M., Beckman, R.A., Weiner, L.M., Mitchell, J.C., Parks, J.M. and Smith, J.C. (2020). Combining Three-Dimensional Modeling with Artificial Intelligence to Increase Specificity and Precision in Peptide–MHC Binding Predictions. The Journal of Immunology, 205(7), pp.1962–1977. doi:10.4049/jimmunol.1900918.

Ribeiro da Cunha, B., Fonseca, L.P. and Calado, C.R.C. (2021). Simultaneous elucidation of antibiotic mechanism of action and potency with high-throughput Fourier-transform infrared (FTIR) spectroscopy and machine learning. Applied Microbiology and Biotechnology, 105(3), pp.1269–1286. doi:10.1007/s00253-021-11102-7.

Zoffmann, S., Vercruysse, M., Benmansour, F., Maunz, A., Wolf, L., Blum Marti, R., Heckel, T., Ding, H., Truong, H.H., Prummer, M., Schmucki, R., Mason, C.S., Bradley, K., Jacob, A.I., Lerner, C., Araujo del Rosario, A., Burcin, M., Amrein, K.E. and Prunotto, M. (2019). Machine learning-powered antibiotics phenotypic drug discovery. Scientific Reports, 9(1). doi:10.1038/s41598-019-39387-9.

Stokes, J.M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N.M., MacNair, C.R., French, S., Carfrae, L.A., Bloom-Ackerman, Z., Tran, V.M., Chiappino-Pepe, A., Badran, A.H., Andrews, I.W., Chory, E.J., Church, G.M., Brown, E.D., Jaakkola, T.S., Barzilay, R. and Collins, J.J. (2020). A Deep Learning Approach to Antibiotic Discovery. Cell, [online] 180(4), pp.688-702.e13. doi:10.1016/j.cell.2020.01.021.

Parvaiz, N., Ahmad, F., Yu, W., MacKerell, A.D. and Azam, S.S. (2021). Discovery of beta-lactamase CMY-10 inhibitors for combination therapy against multi-drug resistant Enterobacteriaceae. PLoS ONE, [online] 16(1). doi:10.1371/journal.pone.0244967.

Hamid, M.-N. and Friedberg, I. (2018). Identifying antimicrobial peptides using word embedding with deep recurrent neural networks. Bioinformatics, 35(12), pp.2009–2016. doi:10.1093/bioinformatics/bty937.

Fields, F.R., Freed, S.D., Carothers, K.E., Hamid, M.N., Hammers, D.E., Ross, J.N., Kalwajtys, V.R., Gonzalez, A.J., Hildreth, A.D., Friedberg, I. and Lee, S.W. (2019). Novel antimicrobial peptide discovery using machine learning and biophysical selection of minimal bacteriocin domains. Drug Development Research, 81(1), pp.43–51. doi:10.1002/ddr.21601.

Badura, A., Krysiński, J., Nowaczyk, A. and Buciński, A. (2020). Application of artificial neural networks to prediction of new substances with antimicrobial activity against Escherichia coli. Journal of Applied Microbiology, 130(1), pp.40–49. doi:10.1111/jam.14763.

Nagpal, G., Chaudhary, K., Agrawal, P. and Raghava, G.P.S. (2018). Computer-aided prediction of antigen presenting cell modulators for designing peptide-based vaccine adjuvants. Journal of Translational Medicine, 16(1). doi:10.1186/s12967-018-1560-1.

Wang, Y., Wang, H., Hou, M., Wang, Y., Bu, D., Zhang, C., Huang, C. and Sun, S. (2021). Glycan immunogenicity prediction based on Graph neural network. 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). doi:10.1109/bibm52615.2021.9669605.

Su, X., Xu, J., Yin, Y., Quan, X. and Zhang, H. (2019). Antimicrobial peptide identification using multi-scale convolutional network. BMC Bioinformatics, 20(1). doi:10.1186/s12859-019-3327-y.

Grafskaia, E.N., Polina, N.F., Babenko, V.V., Kharlampieva, D.D., Bobrovsky, P.A., Manuvera, V.A., Farafonova, T.E., Anikanov, N.A. and Lazarev, V.N. (2018). Discovery of novel antimicrobial peptides: A transcriptomic study of the sea anemone Cnidopus japonicus. Journal of Bioinformatics and Computational Biology, 16(02), p.1840006. doi:10.1142/s0219720018400061.

Macesic, N., Bear Don’t Walk, O.J., Pe’er, I., Tatonetti, N.P., Peleg, A.Y. and Uhlemann, A.-C. (2020). Predicting Phenotypic Polymyxin Resistance in Klebsiella pneumoniae through Machine Learning Analysis of Genomic Data. mSystems, 5(3). doi:10.1128/msystems.00656-19.

Mohapatra, S., An, J. and Gómez-Bombarelli, R. (n.d.). Graph attribution methods applied to understanding immunogenicity in glycans. [online] Available at: https://icml-compbio.github.io/icml-website-2021/2021/papers/WCBICML2021_paper_54.pdf [Accessed 23 Aug. 2022].

Burkholz, R., Quackenbush, J. and Bojar, D. (2021). Using graph convolutional neural networks to learn a representation for glycans. Cell Reports, 35(11), p.109251. doi:10.1016/j.celrep.2021.109251.

Weber, J., Chowell, D., Krishna, C., Chan, T. and Zhou, R. (2020). Predicting HLA-I peptide immunogenicity with deep learning and molecular dynamics. doi:10.21203/rs.3.rs-104972/v1.

Hecht-Nielsen, R. (1990). ON THE ALGEBRAIC STRUCTURE OF FEEDFORWARD NETWORK WEIGHT SPACES. Advanced Neural Computers, pp.129–135. doi:10.1016/b978-0-444-88400-8.50019-4.

Ioffe, S., and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on Machine Learning (pp. 448–456). PMLR.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), p.1929–1958.

Naderializadeh, N., Comer, J., Andrews, R., Hoffmann, H., and Kolouri, S. 2021. Pooling by Sliced-Wasserstein Embedding. In Advances in Neural Information Processing Systems (pp. 3389–3400). Curran Associates, Inc..

Bai, Y., Ding, H., Qiao, Y., Marinovic, A., Gu, K., Chen, T., Sun, Y. and Wang, W., 2019. Unsupervised inductive graph-level representation learning via graph-graph proximity. arXiv preprint arXiv:1904.01098.

Zhang, Z. (2018). Improved Adam Optimizer for Deep Neural Networks. 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS). doi:10.1109/iwqos.2018.8624183.

Jamasb, A.R., Viñas, R., Ma, E.J., Harris, C., Huang, K., Hall, D., Lió, P. and Blundell, T.L. (2020). Graphein - a Python Library for Geometric Deep Learning and Network Analysis on Protein Structures and Interaction Networks. doi:10.1101/2020.07.15.204701.

Hagberg, A., Swart, P. and S Chult, D., 2008. Exploring network structure, dynamics, and function using NetworkX (No. LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Data61, C., 2018. Stellargraph machine learning library. Publication Title: GitHub Repository. GitHub.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, pp.2825-2830.

Brownlee, J., 2017. Gentle introduction to the adam optimization algorithm for deep learning. Machine Learning Mastery, 3.

Jais, I.K.M., Ismail, A.R. and Nisa, S.Q., 2019. Adam optimization algorithm for wide and deep neural network. Knowledge Engineering and Data Science, 2(1), pp.41-46.

Zhang, M., Cui, Z., Neumann, M. and Chen, Y. (2018) “An End-to-End Deep Learning Architecture for Graph Classification”, Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). doi: 10.1609/aaai.v32i1.11782

Geirhos, R., Jacobsen, J., Michaelis, C., Zemel, R., Brendel, W., Bethge, M. and Wichmann, F., 2020. Shortcut learning in deep neural networks. Nature Machine Intelligence, 2(11), pp.665-673.

Ramasubramanian, K. and Singh, A., 2019. Deep learning using keras and tensorflow. In Machine Learning Using R (pp. 667-688). Apress, Berkeley, CA.

Jabbari, P. and Rezaei, N. (2019). Artificial intelligence and immunotherapy. Expert Review of Clinical Immunology, 15(7), pp.689–691. doi:10.1080/1744666x.2019.1623670.

Feng, R., Yu, F., Xu, J. and Hu, X. (2021). Knowledge gaps in immune response and immunotherapy involving nanomaterials: Databases and artificial intelligence for material design. Biomaterials, 266, p.120469. doi:10.1016/j.biomaterials.2020.120469.

Hildebrand, L.A., Pierce, C.J., Dennis, M., Paracha, M. and Maoz, A. (2021). Artificial Intelligence for Histology-Based Detection of Microsatellite Instability and Prediction of Response to Immunotherapy in Colorectal Cancer. Cancers, 13(3), p.391. doi:10.3390/cancers13030391.

Martins, J., Magalhães, C., Rocha, M. and Osório, N.S. (2019). Machine Learning-Enhanced T Cell Neoepitope Discovery for Immunotherapy Design. Cancer Informatics, 18, p.117693511985208. doi:10.1177/1176935119852081.

O'Donnell, T.J., Rubinsteyn, A., Bonsack, M., Riemer, A.B., Laserson, U. and Hammerbacher, J. (2018). MHCflurry: Open-Source Class I MHC Binding Affinity Prediction. Cell Systems, 7(1), pp.129-132.e4. doi:10.1016/j.cels.2018.05.014.

Baranwal, M., Magner, A., Saldinger, J., Turali-Emre, E.S., Elvati, P., Kozarekar, S., VanEpps, J.S., Kotov, N.A., Violi, A. and Hero, A.O. (2020). Struct2Graph: A graph attention network for structure based predictions of protein-protein interactions. doi:10.1101/2020.09.17.301200.

Banerjee, P. and Preissner, R. (2018). BitterSweetForest: A Random Forest Based Binary Classifier to Predict Bitterness and Sweetness of Chemical Compounds. Frontiers in Chemistry, 6. doi:10.3389/fchem.2018.00093.

Yang, X., Yang, S., Li, Q., Wuchty, S. and Zhang, Z. (2020). Prediction of human-virus protein-protein interactions through a sequence embedding-based machine learning method. Computational and Structural Biotechnology Journal, 18, pp.153–161. doi:10.1016/j.csbj.2019.12.005.

Appendices

A. Data Samples

```
HLA    pseudo
HLA-A*0203   ------------YYEKVAH---TVDTLYRY-YYTKWEQWYTWY---
HLA-A*0205   ------Y-----YYEKVV---------YDYYRDTKWQWYWY-----
HLA-A*0206   -----MY-----YYEKVAH---TVDTLYRYHYYTKWVQLYTWY---
HLA-A*0207   ------M-----YYEKVAH---TVDTLYR-HYYTKWVQLYTWY---
HLA-A*3001   ------M-----YYENV-QT-----DTLYIYHYTKWWQLYWY----
HLA-A*6801   -----MY-----YYRNNAQT---V--DTLYYDYTKWAVQWYTWY--
HLA-A*9235   ------M-----YYEKV-HT-----DTYRYHYYTKWVQLYWY----
HLA-A*9253   ------M-----YYEKHT---------VDTRYYTKWYTWY------
HLA-B*0602   ----YDS------YEKYRQ-------ANKYWYYTKWEQWYWY----
HLA-B*1557   ---------------YQWN-----TLYRYEFYWTKWAAYW------
```

Data Values 1 : HLA and pseudo values.

```
H ANDN920101
D alpha-CH chemical shifts (Andersen et al., 1992)
R PMID:1575719
A Andersen, N.H., Cao, B. and Chen, C.
T Peptide/protein structure analysis using the chemical shift index method:
  upfield alpha-CH values reveal dynamic helices and aL sites
J Biochem. and Biophys. Res. Comm. 184, 1008-1014 (1992)
C BUNA790102    0.949
I    A/L     R/K     N/M     D/F     C/P     Q/S     E/T     G/W     H/Y     I/V
    4.35    4.38    4.75    4.76    4.65    4.37    4.29    3.97    4.63    3.95
    4.17    4.36    4.52    4.66    4.44    4.50    4.35    4.70    4.60    3.95
//
H ARGP820101
D Hydrophobicity index (Argos et al., 1982)
R PMID:7151796
A Argos, P., Rao, J.K.M. and Hargrave, P.A.
T Structural prediction of membrane-bound proteins
J Eur. J. Biochem. 128, 565-575 (1982)
C JOND750101    1.000  SIMZ760101    0.967  GOLD730101    0.936
  TAKK010101    0.906  MEEJ810101    0.891  ROSM880104    0.872
  CIDH920105    0.867  LEVM760106    0.865  CIDH920102    0.862
  MEEJ800102    0.855  MEEJ810102    0.853  ZHOH040101    0.841
  CIDH920103    0.827  PLIV810101    0.820  CIDH920104    0.819
  LEVM760107    0.806  NOZY710101    0.800  GUYH850103   -0.808
  PARJ860101   -0.835  WOLS870101   -0.838  BULH740101   -0.854
I    A/L     R/K     N/M     D/F     C/P     Q/S     E/T     G/W     H/Y     I/V
    0.61    0.60    0.06    0.46    1.07      0.    0.47    0.07    0.61    2.22
    1.53    1.15    1.18    2.02    1.95    0.05    0.05    2.65    1.88    1.32
//
```

Data Values 2 : Data samples from AAindex.

peptide	HLA	immunogenicity	test	respond	potential
AACIVGCE	HLA-A*0201	Negative	6	0	0.264411
AAFYHLPLI	HLA-A*1101	Negative	33	0	0.063459
AAKGRGAA	HLA-B*0801	Negative	4	0	0.288128
AAMDGGC	HLA-A*2902	Negative	7	0	0.256535
AANPHATF	HLA-A*0201	Negative	6	0	0.252341
AAPPPQRA	HLA-A*0201	Negative	4	0	0.294556
AATGIAAV	HLA-A*0201	Negative	4	0	0.28251
ACEDDKLN	HLA-A*0101	Negative	4	0	0.292527
ADAFILLNY	HLA-A*0101	Negative	4	0	0.295486
ADDDSFFK	HLA-A*0101	Negative	4	0	0.309335
ADVEFCLS	HLA-A*0201	Negative	15	0	0.133064

Data Values 3 : Immunogenicity values for Peptide and HLA.

B. Code Listing

```
[ ]  def hla_df_to_dic(hla):
         dic = {}
         for i in range(hla.shape[0]):
             col1 = hla['HLA'].iloc[i]  # HLA allele
             col2 = hla['pseudo'].iloc[i]  # pseudo sequence
             dic[col1] = col2
         return dic
```

Code List 1 HLA data frame to dictionary conversion.

```
[ ]  def dict_inventory(inventory):
         dicA, dicB, dicC = {}, {}, {}
         dic = {'A': dicA, 'B': dicB, 'C': dicC}

         for hla in inventory:
             type_ = hla[4]  # A,B,C
             first2 = hla[6:8]  # 01
             last2 = hla[8:]  # 01
             try:
                 dic[type_][first2].append(last2)
             except KeyError:
                 dic[type_][first2] = []
                 dic[type_][first2].append(last2)

         return dic
```

Code List 2 Dictionary Inventory.

```python
@staticmethod
def combinator(pep,hla):
    source = ['p' + str(i+1) for i in range(len(pep))]
    target = ['h' + str(i+1) for i in range(len(hla))]
    return source,target

@staticmethod
def numerical(pep,hla,after_pca,embed=12):   # after_pca [21,12]
    pep = pep.replace('X','-').upper()
    hla = hla.replace('X','-').upper()
    feature_array_pep = np.empty([len(pep),embed])
    feature_array_hla = np.empty([len(hla),embed])
    amino = 'ARNDCQEGHILKMFPSTWYV-'
    for i in range(len(pep)):
        feature_array_pep[i,:] = after_pca[amino.index(pep[i]),:]
    for i in range(len(hla)):
        feature_array_hla[i,:] = after_pca[amino.index(hla[i]),:]
    feature_array = np.concatenate([feature_array_pep,feature_array_hla],axis=0)
    #print(feature_array_pep.shape,feature_array_hla.shape,feature_array.shape)
    return feature_array

@staticmethod
def unweight_edge(pep,hla,after_pca):
    source,target = Graph_Constructor.combinator(pep,hla)
    combine = list(itertools.product(source,target))
    weight = itertools.repeat(1,len(source)*len(target))
    edges = pd.DataFrame({'source':[item[0] for item in combine],'target':[item[1] for item in combine],'weight':weight})
    feature_array = Graph_Constructor.numerical(pep,hla,after_pca)
    try:nodes = IndexedArray(feature_array,index=source+target)
    except: print(pep,hla,feature_array.shape)
    graph = StellarGraph(nodes,edges,node_type_default='corner',edge_type_default='line')
    return graph
```

Code List 3 Graph Converter Combinator, numerical and unweighted edge methods.

```python
@staticmethod
def weight_anchor_edge(pep,hla,after_pca):
    source, target = Graph_Constructor.combinator(pep, hla)
    combine = list(itertools.product(source, target))
    weight = itertools.repeat(1, len(source) * len(target))
    edges = pd.DataFrame({'source': [item[0] for item in combine], 'target': [item[1] for item in combine], 'weight': weight})
    for i in range(edges.shape[0]):
        col1 = edges.iloc[i]['source']
        col2 = edges.iloc[i]['target']
        col3 = edges.iloc[i]['weight']
        if col1 == 'a2' or col1 == 'a9' or col1 ==  'a10':
            edges.iloc[i]['weight'] = 1.5
    feature_array = Graph_Constructor.numerical(pep, hla, after_pca)
    nodes = IndexedArray(feature_array, index=source + target)
    graph = StellarGraph(nodes, edges, node_type_default='corner', edge_type_default='line')
    return graph

@staticmethod
def intra_and_inter(pep,hla,after_pca):
    source, target = Graph_Constructor.combinator(pep, hla)
    combine = list(itertools.product(source, target))
    weight = itertools.repeat(2, len(source) * len(target))
    edges_inter = pd.DataFrame({'source': [item[0] for item in combine], 'target': [item[1] for item in combine], 'weight': weight})
    intra_pep = list(itertools.combinations(source,2))
    intra_hla = list(itertools.combinations(target,2))
    intra = intra_pep + intra_hla
    weight = itertools.repeat(1,len(intra))
    edges_intra = pd.DataFrame({'source':[item[0] for item in intra],'target':[item[1] for item in intra],'weight':weight})
    edges = pd.concat([edges_inter,edges_intra])
    edges = edges.set_index(pd.Index(np.arange(edges.shape[0])))
    feature_array = Graph_Constructor.numerical(pep, hla, after_pca)
    nodes = IndexedArray(feature_array, index=source + target)
    graph = StellarGraph(nodes, edges, node_type_default='corner', edge_type_default='line')
    return graph
```

Code List 4 Graph Converter weighted anchor edge and inter -intra methods.

```python
    @staticmethod
    def entrance(df,after_pca,hla_dic,dic_inventory):
        graphs = []
        graph_labels = []
        from tqdm import tqdm
        for i in tqdm(range(df.shape[0]),total=df.shape[0]):
            pep = df['peptide'].iloc[i]
            try:
                hla = hla_dic[df['HLA'].iloc[i]]
            except KeyError:
                hla = hla_dic[rescue_unknown_hla(df['HLA'].iloc[i],dic_inventory)]
            label = df['immunogenicity'].iloc[i]
            #if label != 'Negative': label = 0
            #else: label = 1
            #graph = Graph_Constructor.unweight_edge(pep,hla,after_pca)
            #graph = Graph_Constructor.unweight_edge(pep,hla,after_pca)
            graph = Graph_Constructor.intra_and_inter(pep,hla,after_pca)
            graphs.append(graph)
            graph_labels.append(label)
        graph_labels = pd.Series(graph_labels)
        return graphs,graph_labels
```

Code List 5 Graph converter entrance method.

```python
[ ]  def rescue_unknown_hla(hla, dic_inventory):
        type_ = hla[4]
        first2 = hla[6:8]
        last2 = hla[8:]
        big_category = dic_inventory[type_]
        #print(hla)
        if not big_category.get(first2) == None:
            small_category = big_category.get(first2)
            distance = [abs(int(last2) - int(i)) for i in small_category]
            optimal = min(zip(small_category, distance), key=lambda x: x[1])[0]
            return 'HLA-' + str(type_) + '*' + str(first2) + str(optimal)
        else:
            small_category = list(big_category.keys())
            distance = [abs(int(first2) - int(i)) for i in small_category]
            optimal = min(zip(small_category, distance), key=lambda x: x[1])[0]
            return 'HLA-' + str(type_) + '*' + str(optimal) + str(big_category[optimal][0])
```

Code List 6 Method to modify unknown HLA values.

```
for fold in fold_indices:
    i = 1
    graphs, graph_labels = Graph_Constructor.entrance(ori_train, after_pca, hla_dic, dic_inventory)
    generator = PaddedGraphGenerator(graphs=graphs)
    gc_model = GCNSupervisedGraphClassification(
        layer_sizes=[64, 64],
        activations=["relu", "relu"],
        generator=generator,
        dropout=0.2, )
    x_inp, x_out = gc_model.in_out_tensors()
    predictions = Dense(units=32, activation="relu")(x_out)
    predictions = Dense(units=16, activation="relu")(predictions)
    predictions = Dense(units=1, activation="sigmoid")(predictions)
    model = Model(inputs=x_inp, outputs=predictions)
    #model.compile(optimizer=Adam(0.001), loss=mean_squared_error)
    model.compile(optimizer=Adam(0.001), loss=mean_squared_error,metrics=["acc"])
    train_gen = generator.flow(
        fold[0],
        targets=graph_labels.iloc[fold[0]].values,
        batch_size=256, )
    test_gen = generator.flow(
        fold[1],
        targets=graph_labels.iloc[fold[1]].values,
        batch_size=1, )
    epochs = 100
    es1 = EarlyStopping(monitor='loss', patience=2, restore_best_weights=False)
    es2 = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=False)
    history = model.fit(
        train_gen, epochs=epochs, validation_data=test_gen, shuffle=True, callbacks=[es1, es2, ],
        class_weight={0: 0.5, 1: 0.5})

    # test in validation
    pred = model.predict(test_gen)
    from sklearn.metrics import mean_squared_error
    result = mean_squared_error(graph_labels.iloc[fold_indices[0][1]], pred, squared=False)
    holding['validation'].append(result)
```

Code List 7 GCN preparation and training and validation.

```
# test in dengue
#ori_test = pd.read_csv('data/dengue_test.csv')
ori_test = pd.read_csv('dengue_test.csv')
graphs_test, graph_labels_test = Graph_Constructor.entrance(ori_test, after_pca, hla_dic, dic_inventory)
generator_test = PaddedGraphGenerator(graphs=graphs_test)
input = generator_test.flow(graphs_test)
prediction = model.predict(input)
from sklearn.metrics import accuracy_score, recall_score, precision_score
hard = [1 if item >= 0.5 else 0 for item in prediction[:, 0]]
result = accuracy_score(graph_labels_test, hard)
holding['dengue'].append(result)

# test in cell
#ori_test_cell = pd.read_csv('data/reproduce_gcn_data/ori_test_cells.csv')
ori_test_cell = pd.read_csv('ori_test_cells.csv')
graphs_test, graph_labels_test = Graph_Constructor.entrance(ori_test_cell, after_pca, hla_dic, dic_inventory)
generator_test = PaddedGraphGenerator(graphs=graphs_test)
input = generator_test.flow(graphs_test)
prediction = model.predict(input)
hard = [1 if item >= 0.5 else 0 for item in prediction]
result1 = recall_score(graph_labels_test, hard)  # recall
ori_test_cell['result'] = prediction
ori_test_cell = ori_test_cell.sort_values(by='result', ascending=False).set_index(
    pd.Index(np.arange(ori_test_cell.shape[0])))
result2 = np.count_nonzero(ori_test_cell['immunogenicity'].values[:20] == 1)  # top20
result3 = np.count_nonzero(ori_test_cell['immunogenicity'].values[:50] == 1)  # top50
holding['cell'].append((result1, result2, result3))
```

Code List 8 Validating the graph model on Dengue and Cell databases.

```
    # test in covid
    #ori = pd.read_csv('data/reproduce_gcn_data/sars_cov_2_result.csv')
    ori = pd.read_csv('sars_cov_2_result.csv')
    ori = ori.sample(frac=1, replace=False).set_index(pd.Index(np.arange(ori.shape[0])))
    ori_test_covid = retain_910(ori)
    graphs_test, graph_labels_test = Graph_Constructor.entrance(ori_test_covid, after_pca, hla_dic, dic_inventory)
    generator_test = PaddedGraphGenerator(graphs=graphs_test)
    input = generator_test.flow(graphs_test)
    prediction = model.predict(input)
    hard = [1 if item >= 0.5 else 0 for item in prediction]
    result1 = recall_score(ori_test_covid['immunogenicity-con'], hard)  # convalescent recall
    result2 = recall_score(ori_test_covid['immunogenicity'], hard)  # unexposed recall
    result3 = precision_score(ori_test_covid['immunogenicity-con'], hard)  # convalescent recall
    result4 = precision_score(ori_test_covid['immunogenicity'], hard)  # unexposed recall
    holding['covid'].append((result1, result2, result3, result4))
    print('round {}, finished covid'.format(i))
    break
```

Code List 9 Validating the graph model on Covid dataset.

```
[ ]  def create_graph_classification_model(generator):
         gc_model = GCNSupervisedGraphClassification(
             layer_sizes=[64, 64],
             activations=["relu", "relu"],
             generator=generator,
             dropout=0.5,
         )
         x_inp, x_out = gc_model.in_out_tensors()
         predictions = Dense(units=32, activation="relu")(x_out)
         predictions = Dense(units=16, activation="relu")(predictions)
         predictions = Dense(units=1, activation="sigmoid")(predictions)

         # Let's create the Keras model and prepare it for training
         model = Model(inputs=x_inp, outputs=predictions)
         model.compile(optimizer=Adam(0.005), loss=binary_crossentropy, metrics=["acc"])

         return model
```

Code List 10 Graph Classification model.

```
[ ]  def get_generators(train_index, test_index, graph_labels, batch_size):
         train_gen = generator.flow(
             train_index, targets=graph_labels.iloc[train_index].values, batch_size=batch_size
         )
         test_gen = generator.flow(
             test_index, targets=graph_labels.iloc[test_index].values, batch_size=batch_size
         )

         return train_gen, test_gen
```

```
[ ]  test_accs = []

     stratified_folds = model_selection.RepeatedStratifiedKFold(
         n_splits=folds, n_repeats=n_repeats
     ).split(graph_labels1, graph_labels1)

     for i, (train_index, test_index) in enumerate(stratified_folds):
         print(f"Training and evaluating on fold {i+1} out of {folds * n_repeats}...")
         train_gen, test_gen = get_generators(
             train_index, test_index, graph_labels1, batch_size=30
         )

         model = create_graph_classification_model(generator)

         history, acc = train_fold(model, train_gen, test_gen, es, epochs)

         test_accs.append(acc)
```

Code List 11 Graph generation and validating the graph models.

```
[ ]  k = 35  # the number of rows for the output tensor
     layer_sizes = [32, 32, 32, 1]

     dgcnn_model = DeepGraphCNN(
         layer_sizes=layer_sizes,
         activations=["tanh", "tanh", "tanh", "tanh"],
         k=k,
         bias=False,
         generator=generator,
     )
     x_inp, x_out = dgcnn_model.in_out_tensors()
```

Code List 12 DGCN layer design.

```
[ ] x_out = Conv1D(filters=16, kernel_size=sum(layer_sizes), strides=sum(layer_sizes))(x_out)
    x_out = MaxPool1D(pool_size=2)(x_out)

    x_out = Conv1D(filters=32, kernel_size=5, strides=1)(x_out)

    x_out = Flatten()(x_out)

    x_out = Dense(units=128, activation="relu")(x_out)
    x_out = Dropout(rate=0.5)(x_out)

    predictions = Dense(units=1, activation="sigmoid")(x_out)
```

```
[ ] model = Model(inputs=x_inp, outputs=predictions)

    model.compile(
        optimizer=Adam(lr=0.0001), loss=binary_crossentropy, metrics=["acc"],
    )
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

```
[ ] train_graphs, test_graphs = model_selection.train_test_split(
        graph_labels, train_size=0.9, test_size=None, stratify=graph_labels,
    )
```

```
[ ] gen = PaddedGraphGenerator(graphs=graphs)

    train_gen = gen.flow(
        list(train_graphs.index - 1),
        targets=train_graphs.values,
        batch_size=50,
        symmetric_normalization=False,
    )
```

Code List 13 DGCN model preparation and training.

```
[ ] gc_model = sg.layer.GCNSupervisedGraphClassification(
        [64, 32], ["relu", "relu"], generator, pool_all_layers=True
    )
```

```
[ ] inp1, out1 = gc_model.in_out_tensors()
    inp2, out2 = gc_model.in_out_tensors()

    vec_distance = tf.norm(out1 - out2, axis=1)
```

```
[ ] pair_model = keras.Model(inp1 + inp2, vec_distance)
    embedding_model = keras.Model(inp1, out1)
```

```
[ ] def graph_distance(graph1, graph2):
        spec1 = nx.laplacian_spectrum(graph1.to_networkx(feature_attr=None))
        spec2 = nx.laplacian_spectrum(graph2.to_networkx(feature_attr=None))
        k = min(len(spec1), len(spec2))
        return np.linalg.norm(spec1[:k] - spec2[:k])
```

```
[ ] graph_idx = np.random.RandomState(0).randint(len(graphs), size=(100, 2))
```

```
[ ] targets = [graph_distance(graphs[left], graphs[right]) for left, right in graph_idx]
```

```
    train_gen = generator.flow(graph_idx, batch_size=10, targets=targets)
```

```
[ ] pair_model.compile(keras.optimizers.Adam(1e-2), loss="mse")
```

Code List 14 Unsupervised graph classification distance measure.

```
[ ] embeddings = embedding_model.predict(generator.flow(graphs))
```

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn import model_selection
```

```
[ ] train_labels, test_labels = model_selection.train_test_split(
        graph_labels, train_size=0.1, test_size=None, stratify=graph_labels
    )

    test_embeddings = embeddings[test_labels.index - 1]
    train_embeddings = embeddings[train_labels.index - 1]

    lr = LogisticRegression(multi_class="auto", solver="lbfgs")
    lr.fit(train_embeddings, train_labels)

    y_pred = lr.predict(test_embeddings)
    gcn_acc = (y_pred == test_labels).mean()
    print(f"Test classification accuracy: {gcn_acc}")
```

Code List 15 Unsupervised model validation.