UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUCAS LIMA DE MELO

A study on graph neural networks for classification tasks and model interpretability on genomic datasets

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Profa. Dra. Mariana
Recamonde-Mendoza
Coadvisor: MSc. Thomas Vaitses Fontanari

Porto Alegre
February 2024

AGRADECIMENTOS

Desde que iniciei minha graduação, muitos anos se passaram e, durante esse tempo, cruzei com inúmeras pessoas especiais. Sem dúvida, seria impossível mencionar todas elas. Portanto, desejo iniciar expressando minha imensa gratidão aos meus pais, que desde o início me ofereceram todo o suporte e apoio necessários nesta jornada desafiadora, e à minha irmã, que é a minha fonte de luz e inspiração diária.

Agradeço também em especial à minha orientadora, Prof. Mariana Recamonde-Mendoza, e ao meu co-orientador Thomas Vaitses Fontanar, que me ajudaram imensamente no processo de construção deste trabalho.

Por fim, também gostaria de agradecer à todos meu amigos que apoiaram nesta jornada e a fizeram de alguma forma mais leve e também ao o Instituto de Informática (INF) e ao Parque Computacional de Alto Desempenho (PCAD) por disponibilizar seus recursos e sua estrutura durante esta etapa.

ABSTRACT

Recently, a few works have started proposing the use of graph neural networks (GNNs) to embed knowledge of gene interactions in machine learning models and thus produce more robust classifiers for genomic classification tasks. GNNs, however, produce embeddings for each gene in the biological network, and these embeddings must then be summarized into a single representation that can be used to produce a classification for the entire network - a process commonly referred to as pooling. Although previous works have achieved encouraging results, there is a lack of studies that aim to understand the effects of the choice of GNN architecture, biological network, and, in particular, the pooling approach. Therefore, this work aims to explore the impact that these alternatives have on the performance and interpretability of the resulting models. Our findings highlight SAGPool's superior predictive power and GraphSAGE's robustness across most pooling methods. We also showed that a preprocessing technique could offer enhanced performance for certain pooling methods, providing favorable trade-offs in predictive performance and computational resources. Despite challenges in identifying biomarker genes through saliency maps, we were able to identify genes like ADAM33 and DNASE1L3 that correlate with breast cancer. We conclude that the choice of the right architecture significantly impacts model performance and resource utilization, underscoring its importance in GNNs studies.

Keywords: Graph neural networks. Pooling. Interpretability. Genomics.

Um estudo sobre redes neurais em grafos para tarefas de classificação e interpretabilidade de modelos em conjuntos de dados genômicos

RESUMO

Recentemente, alguns trabalhos começaram a propor o uso de redes neurais de grafo (GNNs) para incorporar o conhecimento das interações genéticas nos modelos de aprendizado de máquina e, assim, produzir classificadores mais robustos para tarefas de classificação genômica. No entanto, as GNNs produzem *embeddings* para cada gene na rede biológica, e esses *embeddings* devem então ser resumidas em uma única representação que possa ser usada para produzir uma classificação para toda a rede - um processo comumente referido como *pooling*. Embora trabalhos anteriores tenham alcançado resultados encorajadores, há uma falta de estudos que visem entender os efeitos da escolha da arquitetura da GNN, da rede biológica e, em particular, da abordagem de pooling. Portanto, este trabalho tem como objetivo explorar o impacto que essas alternativas têm no desempenho e na interpretabilidade dos modelos resultantes. Nossos resultados destacam o poder preditivo superior do SAGPool e a robustez do GraphSAGE em relação à maioria dos métodos de *pooling*. Também demonstramos que uma técnica de pré-processamento pode oferecer um melhor desempenho para certos métodos de *pooling*, proporcionando compensações favoráveis em poder preditivo e recursos computacionais. Apesar dos desafios em identificar genes marcadores através de mapas de saliência, fomos capazes de identificar genes como ADAM33 e DNASE1L3 que se correlacionam com o câncer de mama. Concluímos que a escolha da arquitetura correta impacta significativamente o desempenho do modelo e a utilização de recursos, destacando sua importância em estudos de GNNs.

Palavras-chave: *graph neural networks*. *pooling*. interpretabilidade. genômica.

LIST OF FIGURES

Figure 2.1 Typical RNA-Seq pipeline 15
Figure 2.2 Overview of TopKPool Method 25
Figure 2.3 Overview of SAGPool Method 26

Figure 4.1 Global Pooling Architecture 34
Figure 4.2 Hierarchical Pooling Architecture 35
Figure 4.3 Sample of the dataset. The rows represent samples from the TCGA database, and the columns represent the gene expression values 37
Figure 4.4 Model Architecture for Experiment Two 38

Figure 5.1 Training Time and Memory Usage Experiment 42
Figure 5.2 Average peak memory usage by the number of nodes on the input graph across three executions 43
Figure 5.3 Average training time in seconds by the number of nodes on the input graph across three executions 45
Figure 5.4 Zoomed average training time in seconds by the number of nodes for SAG, Set2Set, Sort, TopK across three executions 46
Figure 5.5 Experiment performed on the influence of the GNN architecture on the results 47
Figure 5.6 Boxplot of F1 Test Score with different convolutional layers 49
Figure 5.7 Experiment performed on the influence of the preprocessing step on the results 51
Figure 5.8 Boxplot of F1 Test Score with different graph sizes 53
Figure 5.9 Model interpretability using saliency maps experiment 56
Figure 5.10 Normalized saliencies over samples for each gene 58
Figure 5.11 t-SNE visualization of the embeddings saliencies produced by the model 60

LIST OF TABLES

Table 2.1 Taxonomy of the chosen pooling methods. T: Trainability, nT: Non-trainable. D: Density, S: Sparse. F: Fixed, A: Adaptable. H: Hierarchical, G: Global.....23

Table 5.1 Average Test F1 score (Standard Deviation).....50
Table 5.2 Selected genes and their previous literature correspondence in BRCA studies57

LIST OF ABBREVIATIONS AND ACRONYMS

TCGA	The Cancer Genome Atlas
GNNs	Graph Neural Networks
SVM	Support Vector Machines
PPIs	Protein-Protein Interactions
DNA	Deoxyribonucleic Acid
RNA	Ribonucleic Acid
mRNA	Messenger RNA
RNA-Seq	RNA Sequencing
GDC	Genomic Data Commons
FPKM	Fragments Per Kilobase of transcript per Million mapped reads
FPKM-UQ	Fragments Per Kilobase of transcript per Million mapped reads upper quartile
GCN	Graph Convolutional Networks
CNN	Convolutional Neural Networks
GraphSAGE	Graph Sample and Aggregate
GAT	Graph Attention Networks
LSTM	Long Short-Term Memory
WL	Weisfeiler-Lehman
STRING	Search Tool for the Retrieval of Interacting Genes/Proteins
FC	Fully Connected (in the context of neural networks, referring to fully connected layer)
PSN	Patient Similarity Network
KEGG	Kyoto Encyclopedia of Genes and Genomes
SHAP	SHapley Additive exPlanations
GLRP	Graph Layer-wise Relevance Propagation

scRNA-seq	Single-cell RNA sequencing
BRCA	Breast Invasive Carcinoma

CONTENTS

1 INTRODUCTION....11
2 BACKGROUND....14
2.1 Biological Background....14
2.1.1 The Genome....14
2.1.2 RNA-Seq data....15
2.2 Graph Neural Networks....16
2.3 GCN - Graph Convolutional Networks....17
2.4 GraphSAGE - Graph Sample and Aggregate Networks....17
2.5 GAT - Graph Attention Networks....18
2.6 Training a Graph Neural Network....19
2.7 Graph Pooling....21
2.7.1 Pooling Layers....22
2.7.1.1 DiffPool....23
2.7.1.2 MinCutPool....23
2.7.1.3 TopKPool....24
2.7.1.4 SAGPool....25
2.7.1.5 Graclus....25
2.7.1.6 Set2Set....26
2.7.1.7 SortPool....27
2.8 Saliency Maps....27
3 RELATED WORK....29
4 METHODOLOGY....32
4.1 Experiment One - Time and Memory Usage Analysis....32
4.1.1 Synthetic Data Generation....32
4.1.2 Network Architecture....33
4.1.3 Training and Metrics....36
4.2 Experiment Two - Predictive Power on Genomic Data....36
4.2.1 Data....36
4.2.2 Network Architecture....37
4.2.3 Training and Metrics....38
4.3 Experiment Three - Interpretability....40
5 RESULTS....41
5.1 Experiment One - Time and Memory Usage Analysis....41
5.2 Experiment Two - Predictive Power on Genomic Data....46
5.3 Experiment Three - Interpretability....55
6 CONCLUSION....61
REFERENCES....63

1 INTRODUCTION

Cancers, a diverse group of diseases, pose a major global health challenge. They are characterized by the uncontrolled growth and spread of abnormal cells, which can infiltrate and destroy healthy tissues, eventually disrupting vital bodily processes. In 2022, cancer was the second leading cause of death in the United States, with 607,790 deaths, just behind heart disease (AHMAD et al., 2023). In 2023, the expected number of new cases is projected to exceed 1.9 million, with an anticipated 609,000 fatalities (SIEGEL et al., 2023). Breast cancer, a principal contributor to this factor, represents the most common cancer diagnosis among women. Recent statistics indicated that in 2023, there would be more than 297,00 new cases of invasive breast cancer, which is expected to result in over 43,000 deaths (SIEGEL et al., 2023). These statistics not only reveal the widespread impact of breast cancer but also highlight the need for progress in its detection and in understanding the disease's underlying mechanisms.

The sharing and analysis of omics data has shown promising results in the fight against breast cancer. By examining the genomic, transcriptomic, and proteomic profiles of the disease, researchers can obtain a comprehensive understanding of the disease's patterns and characteristics. The TCGA (standing for The Cancer Genome Atlas) project[1] has been a key player in this scenario, providing an extensive repository of over 2.5 petabytes of publicly available omics data. However, the complex nature of omics data sets a challenge, requiring advanced computational techniques and strategies for analysis and interpretation. Machine learning, a field of artificial intelligence, plays a key role in tackling this problem by providing computers with the ability to learn from the data, identify patterns, and then make predictions without being directly programmed, empowering researchers to analyze massive amounts of omics data rapidly.

In the past, traditional machine learning models like linear regression and support Vector Machines (SVM) were used to approach this problem. Nevertheless, these models struggled with the high dimensionality and complexity inherent to omics data, requiring advanced feature selection and engineering techniques to minimize those problems, and yet could result in information loss and limited generalizability (ALBARADEI et al., 2021). With the advances in computing and computers getting more powerful to accommodate vast amounts of data and processing, neural networks have risen in popularity to tackle these challenges. In the genomic field, it's no different, researchers started to ap-

[1] https://www.cancer.gov/ccg/research/genome-sequencing/tcga

ply neural networks to omics data for classification tasks as in (WU et al., 1995). More recently, a particular type of neural network, called graph neural network, has started to get the attention of researchers. This attention is driven by its unique ability to handle data that are naturally represented as graphs, like molecular structures and biological pathways.

A lot of effort has been put into modeling omics data. One way to do it is using Protein-Protein Interactions (PPIs), i.e., the physical or functional interactions between two or more proteins that embody complex biological functions and can elucidate the molecular basis of diseases (GONZALEZ; KANN, 2012). One can integrate PPI networks with various omics data types at DNA and RNA levels to get a better understanding of gene expression to protein interactions.

A GNN mechanism called pooling is particularly important when working with omics data. That is because pooling allows the node embedding information to be summarized into a single final embedding for the genomic network, which is essential since most of the genomic tasks involve some type of graph-level classification, such as the one in this study.

Traditional pooling methods are generally global, i.e., they aggregate information from the entire graph to form a single representation and can capture the overall structure and properties of the network. However, hierarchical methods, such as DiffPool(YING et al., 2019) and Self Attention Graph Pooling (LEE; LEE; KANG, 2019), are growing in popularity because of their capability of capturing both local and global features of the network, and because the possibility of gaining cluster insights at each network level.

Incorporating hierarchical pooling into graph neural networks for genomic analyses enables a more granular examination of the network interactions. By examining the outputs of these methods, one can gain possible comprehension of the gene expressions and also identify local patterns in their interactions.

In this work, our objectives are structured as follows: (i) Analyze the efficiency of pooling methods using a synthetic dataset; (ii) evaluate the performance of various GNN architectures in classifying cancer-related genetic data, specifically focusing on analyzing breast cancer PPI data from the TCGA database; (iii) provide an interpretability analysis of the most effective GNN architecture identified in our study.

The work is organized as follows. Chapter 2 contains the necessary background for understanding our work. Chapter 3 contains the reviewed literature that is most similar to ours. Chapter 4 presents our adopted methodology for all of our experiments. Chapter

5 is the place where we discuss our results. Chapter 6 concludes the study.

2 BACKGROUND

2.1 Biological Background

This section is dedicated to reviewing the essential concepts of genomics and gene expression data, especially the RNA-seq data.

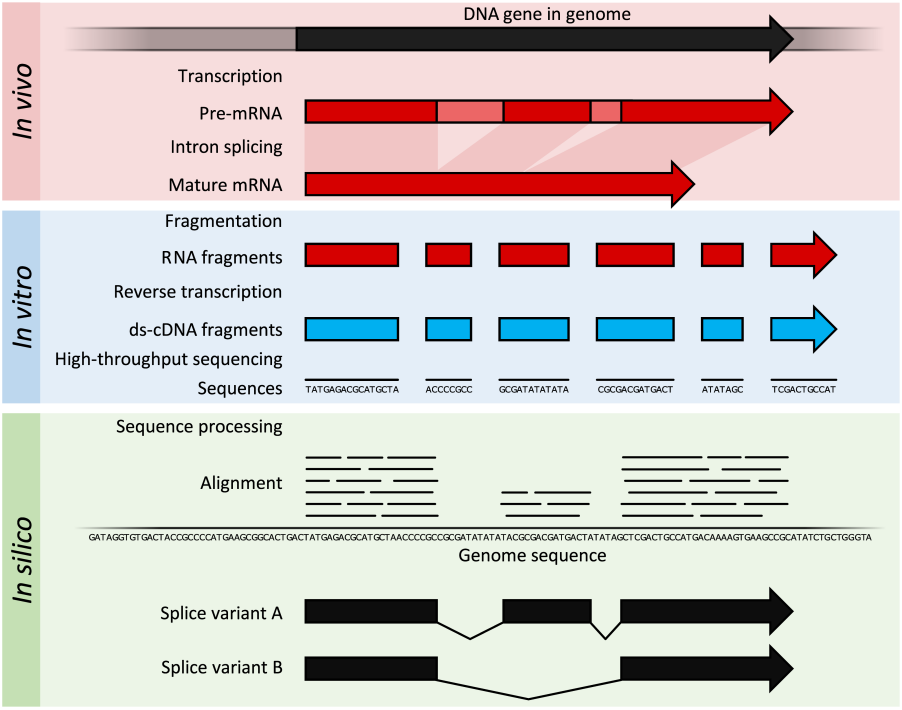
2.1.1 The Genome

A genome refers to the entirety of an organism's DNA. It encompasses all the details, the structure, function, development, and reproduction of that organism, where every cell within an organism's body possesses a replica of its genome (National Human Genome Research Institute, 2023). DNA, the molecule that carries genetic information in living organisms, is composed of nucleotides, each consisting of a phosphate group, a sugar group, and one of four nitrogen bases: adenine (A), thymine (T), guanine (G), and cytosine (C). The different sequences can determine traits like eye color and height but can also determine deficiencies. The DNA's instructions are transformed into RNA. RNA plays a role in converting these instructions into proteins, which are necessary for various cell functions.

The synthesis of proteins occurs in two primary steps: transcription and translation. Transcription is the first step in gene expression, where cells interpret sections of DNA referred to as genes to generate RNA molecules, most specifically messenger RNA (mRNA) molecules. The second step in gene expression is translation. In this step, the mRNA molecule is read according to the genetic code, which translates the sequence of DNA bases into the corresponding amino acid sequence in proteins (BROWN; CLANCY, 2008).

However, not all genes are active at all times, their level of expression can vary depending on factors such as the type of cell the stage of development, or even external signals. This change in expression is what allows different types of cells in an organism to perform different functions even though all cells contain the same set of genes. Also, this change allows scientists to study the expression patterns at different times in the same individual to help identify diseases and potentially produce new treatments since the dysregulation of gene expression is an indicator of many diseases, such as cancer.

Figure 2.1: Typical RNA-Seq pipeline



Source: Lowe et al. (2017)

2.1.2 RNA-Seq data

The transcriptome of a cell is the set of all its RNA molecules at a given time and is the subject of transcriptomics. Currently, two techniques of analyzing the transcripts are dominant: RNA-Seq and Microarrays. Microarrays are a technique that involves multiple pre-determined microscopic slides, also known as gene chips. These chips are then introduced with complementary nucleic acids in a process known as hybridization, where the sample DNA or RNA binds with the chip sequences to show gene expression. RNA-Seq is a newer technique that overcomes the limitations of microarrays relying on the existing pre-determined sequences, isolates the RNA, and breaks it into small fragments, known as reads.

To get the gene expression data with the RNA-Seq technique, the common pipeline is shown in Figure 2.1. It starts by generating the mature mRNA within the organisms by transcribing and splicing the genes. Then, the mRNA is fragmented and converted into double-stranded DNA to facilitate the sequencing, as the cDNA is more stable for such technologies. Finally, the cDNA is sequenced using high-throughput sequencing techniques and is aligned to a reference genome. The result of this pipeline is raw read counts.

To normalize these raw read counts, we have used the Genomic Data Commons (GDC) pipeline. Specifically, the Fragments Per Kilobase of transcript per Million mapped reads upper quartile (FPKM-UQ) technique was used to normalize the gene expression. The formula for the expression of gene g is given by:

$$FPKM_g = \frac{RM_g}{RM_{75}L_g} \times 10^9 \tag{2.1}$$

Here, $FPKM_g$ is the expression level of gene g, RM_g represents the number of reads mapped to gene g, indicating the raw count of sequencing fragments that align with the gene. The term RM_{75} is the 75th percentile of read counts mapped to all genes, which is used as a normalization factor to account for the depth of sequencing data. The gene length in base pairs is denoted by L_g and is essential for also normalizing the values within the sample, which helps to reduce the effect of longer genes having more reads and small genes having fewer reads. The whole process can be found in more detail on the GDC website [1].

2.2 Graph Neural Networks

Graph neural networks (GNN) consist of an adaptation of deep neural network models but are designed for working with graph-structured data (HAMILTON, 2020). As of this moment, it appears that the message passing framework (GILMER et al., 2017) is the most common approach for analyzing GNNs. The message passing framework can be decomposed as two operations at each iteration: an aggregate function and an update function (HAMILTON, 2020).

The embedding of each node u is updated according to its previous value and to the previous values of its neighbors $\mathcal{N}(u)$ through the AGGREGATE$^{(k)}$ function, as expressed in the equation:

$$h_v^{(k+1)} = \text{UPDATE}^{(k)}\left(h_v^{(k)}, \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k)}, \forall u \in \mathcal{N}(v)\right\}\right)\right) \tag{2.2}$$

where $h_u^{(k)}$ denotes the embedding of node u at step k and $h_u^{(0)}$ is defined as the input features of node u.

Many GNN architectures are elaborated from this framework, such as the three

[1] 4https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/Expression_mRNA_Pipeline/

selected for this study: GCN, GraphSAGE, and GAT. These architectures were selected because they provide both a sparse and dense, which will be explained in more detail in Section 2.7, implementation, which allows the comparison of the different pooling methods.

2.3 GCN - Graph Convolutional Networks

Following the principles of graph neural networks and message passing, Graph Convolutional Networks (GCN), proposed by Kipf and Welling (2017), have proved to be one of the most popular and effective baseline GNN architectures (HAMILTON, 2020). As the name suggests, GCN models represent an adaption of Convolutional neural networks(CNN) specifically designed for graph-structured data.

In GCNs, the heart of the message-passing framework is implemented through its convolutional layers, designed to aggregate and process information from a node's immediate neighbors. Mathematically, the convolutional layer is expressed as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.3}$$

where, $H^{(l)}$ represents the node features at layer l, $\tilde{A}$ is the adjacency matrix with self-connections, $\tilde{D}$ is its degree matrix, $W^{(l)}$ denotes the layer-specific weight matrix, and σ is a non-linear activation function.

Equation 2.3 can be seen as a composition of three core operations: normalization, feature transformation, and non-linear function. The normalization factor $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is reponsible for scaling node features. The feature transformation is represented by $H^{(l)}W^{(l)}$, analogous to convolution in CNNs. It aggregates features from the local neighborhood and enables the network to develop complex feature representations. Lastly, the output matrix is passed to a non-linear activation function σ, which helps capture patterns in the graph data.

2.4 GraphSAGE - Graph Sample and Aggregate Networks

Another popular GNN architecture is GraphSAGE (HAMILTON; YING; LESKOVEC, 2018). Unlike GCN, which uses the entire node neighborhood to calculate the node's feature vector, GraphSAGE introduces a novel approach based on sampling the neighbor-

hood to propagate information. In addition to that, GraphSAGE also proposes a generalized neighborhood aggregation function instead of averaging the information as in the GCN architecture.

The GraphSAGE algorithm can be mathematically seen as a combination of two steps: The aggregation of neighbor features and the subsequent update of node features. The aggregation step in the GraphSAGE algorithm is expressed as:

$$h_k^N(v) = \text{AGGREGATE}_k \left\{ h_{k-1}^u \,|\, \forall u \in N(v) \right\} \tag{2.4}$$

In this formula, $h_{\mathcal{N}(v)}^k$ is simply the aggregated features of the neighbors of the node v at layer k. The function AGGREGATE$_k$ takes the features h_u^{k-1} of each neighbor u in the neighborhood $\mathcal{N}(v)$ from the previous layer $k-1$. AGGREGATE$_k$ can be different functions such as a mean, LSTM, pooling, or any other custom method.

The second step of GraphSAGE updates the features of each node, concatenating the computed aggregated features of layer k with the features of layer $k-1$, which is mathematically represented as:

$$h_v^k = \sigma \left(W^k \cdot \text{CONCAT} \left(h_v^{k-1}, h_{\mathcal{N}(v)}^k \right) \right) \tag{2.5}$$

A weight matrix W^k is then used to transform the concatenated output by multiplying it. Similar to the GCN, a non-linear activation function is applied to this transformed output, making it capable of learning the graph patterns.

2.5 GAT - Graph Attention Networks

The third GNN architecture explored in this work was the Graph Attention Networks (VELIčKOVIć et al., 2018). The key component introduced in GAT was the attention mechanism, which computes attention coefficients for each neighbor, indicating the importance of that neighbor's features for the update operation of the node. The coefficient that represents the attention given to the edge connecting nodes i and j is designated as α_{ij} and is defined by the following equation:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\mathrm{T}} \left[\mathbf{W}h_i \,\|\, \mathbf{W}h_j\right]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\mathrm{T}} \left[\mathbf{W}h_i \,\|\, \mathbf{W}h_k\right]\right)\right)} \tag{2.6}$$

Here, $\mathcal{N}_i$ represents the set of nodes adjacent to node i, a refers to a learnable weight vector that is useful for the model's adaptive learning, while W denotes the shared linear transformation matrix. The node's output features are then computed as follows:

$$h_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j\right)$$

Where, as in the other architectures, the non-linear activation function is applied to the transformed feature output, which in this case is the summation weighted by attention coefficients α_{ij}.

The GAT layer can be extended for k separated heads, which means that the graph attention mechanism can be applied to k different subgraphs of the input graph, each of which is assigned a separate attention head. This allows the model to learn different representations of the graph and can be useful for graph representation learning. The final GAT layer can be expressed as:

$$h_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N(i)} \alpha_{kij} W_k h_j\right) \tag{2.7}$$

2.6 Training a Graph Neural Network

Training a Graph Neural Network (GNN) is similar to training neural networks. However, adjustments need to be made to account for the graph data, as within networks, the goal is to optimize a set of parameters (w) in order to minimize a loss function. The loss function measures the difference between the network's predictions and the actual data. While the Euclidean structure of data facilitates effective learning in classical deep learning models, the inherent variability in node connectivity and graph size within graph-structured data presents some subtleties during the training process of GNNs.

In a scenario where we are classifying graphs, the dataset consists of a collection of graphs, such as G_1 G_2... G_N. Each graph, represented by G_i is assigned a label denoted as t_i. These graphs are composed of nodes that contain feature vectors. The objective is to train the Graph Neural Network (GNN) to associate these graph structures with their classes.

When it comes to the loss function, we often use cross-entropy, which is calculated

over graphs. It can be represented by the equation:

$$E(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} t_{ik} \ln y_k(G_i, w), \tag{2.8}$$

Here N represents the number of graphs K represents the number of classes t_ik represents the class of graph i and $y_k(G_i, w)$ symbolizes the predicted probability that graph i belongs to class k. This prediction is based on factors such as the structure of the graph, node features, and network parameters represented by w.

Typically, when optimizing the network parameters w, we use gradient descent methods. This involves updating the parameters in a step-by-step manner to minimize the loss. The update rule can be expressed as follows;

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \tag{2.9}$$

In this equation, η represents the learning rate, and $\nabla E(w^{(\tau)})$ represents the gradient of the loss function with respect to the parameters. Backpropagation enhances this procedure by efficiently computing the gradient $\nabla E(w^{(\tau)})$. It functions by propagating the error in reverse from the output layer to the input layer, enabling parameter adjustment using the descent method. This iterative process plays a role in refining the network performance.

Another common approach is to use a batch of graph examples in the training process instead of one at a time. The batch size is a crucial hyperparameter throughout the training process. While a larger batch size can provide more accurate estimations of the error function, it can also be computationally demanding and slow down the training. Conversely, a smaller batch size may lead to suboptimal error function estimates and convergence issues. The training process is typically iterative, containing multiple epochs, where an epoch is defined as one pass through the entire training dataset.

However, a significant challenge in this learning process is overfitting, where the model performs well on training data but poorly on unseen data. Many Techniques were designed to mitigate this effect, like dropout and early stopping, collectively known as regularization (GOODFELLOW; BENGIO; COURVILLE, 2016). Dropout (SRIVASTAVA et al., 2014) is a simple technique that changes the input signals of nodes to zero, removing the connections with this node, given a dropout probability of p. Early stopping is a regularization technique that returns to the best parameters of the network at the point in time with the lowest validation set error after running the training for p epochs, where p

is the patience, i.e., the number of times to observe worsening validation set error before stopping the training.

2.7 Graph Pooling

In the context of GNNs designed for graph classification tasks, the process referred to as graph pooling is crucial to summarize the embeddings produced by the GNN, thereby obtaining a single representation of the whole graph. As shown by Grattarola et al. (2022), graph pooling can be seen as the union of 3 operations: selection, reduction, and connection (SRC).

In the SRC framework, each of those three operations plays a different role in the process. The selection operation selects specific nodes from the input graph to form what is known as *supernodes* in the pooled graph. Essentially, a supernode, represented as S_k, is a subset consisting of input nodes (x_i, s_i) assigned to sets $S_1, S_2, \ldots, S_K$ in the pooled graph, where s_i represents how much each node x_i contributes to the *supernode*. The reduce operation aggregates the node attributes of a graph G selected in each *supernode* S_k. Lastly, the connection operation is responsible for determining the presence or absence of an edge between each pair of supernodes.

Another contribution of Grattarola et al. (2022) was the definition of a pooling taxonomy. They propose that graph pooling methods were based on four categories: Trainability, Density of the supernodes, Adaptability of K, and Hierarchy.

Trainability refers to whether or not the method has parameters that are learned by optimizing a task-driven loss function. Trainable methods are a novel research area and were specifically designed for GNN with the intuition of not relying on prior assumptions of the network. In contrast, non-trainable methods are generally used for coarsening the graph based on certain assumptions, such as sorting nodes by their structural roles and clustering by the graph's community structures.

The density of the supernodes is related to the size of the supernodes after the selection operation. A method is called dense when the selection operation yields supernodes S_k with a cardinality of $O(N)$, meaning the size of each supernode is proportional to the total number of nodes in the graph N. Sparse methods, on the other hand, are termed this way when the supernodes have a constant cardinality of $O(1)$, meaning the selection operation consistently picks a fixed number of nodes to form each supernode, irrespective of the overall size of the graph.

Adaptability of K is concerned with the number of nodes K of the pooled graph. A method is classified as fixed when K is a hyperparameter of the pooling operator, is always constant, and is unrelated to the graph size. Conversely, the method is classified as adaptative if the number of supernodes is a function $K(G)$ of the input graph G.

Hierarchy is a crucial aspect of graph pooling methods. A method is called hierarchical if it can iteratively coarsen the graph into a smaller-sized one while aiming to preserve the hierarchical graph's structural information. On the other hand, a method is called global if it generates graph-level representations in a single step, reducing the graph to a single representation. Hierarchical methods yield multi-resolution graph representations, allowing GNNs to extract high-level properties, whereas global methods directly compute graph embeddings for integration with traditional vector-based layers. It is also important to highlight that both types of methods can participate in the same GNN architecture.

Furthermore, Liu et al. (2022) proposed to divide the hierarchical pooling methods into two sub-categories: Node Drop Pooling and Node Clustering Pooling. Methods that fit in the node clustering category treat graph pooling as a node clustering issue, where nodes are mapped into clusters that form the new nodes of a coarsened graph. In contrast, node drop pooling methods delete nodes with lower significance scores.

2.7.1 Pooling Layers

In this section, we will introduce the idea behind each pooling layer used for comparison in this work.

We made an effort to select methods that belong to different categories. The methods we chose were DiffPool (YING et al., 2019), SAGPool (LEE; LEE; KANG, 2019), MincutPool (BIANCHI; GRATTAROLA; ALIPPI, 2020), TopKPool (GAO; JI, 2019), Graclus (DHILLON; GUAN; KULIS, 2007), SortPool (ZHANG et al., 2018), and Set2Set (VINYALS; BENGIO; KUDLUR, 2016). Additionally, we also tested LaPool, EdgePool, and MVPool but found that the former was very slow for our settings, while EdgePool and MVPool exceeded our GPU's memory capacity.

Table 2.1 presents a breakdown of the selected methods with their respective categories, using abbreviations for clarity. The table is separated by trainability (T for trainable, nT for non-trainable), density (D for dense, S for sparse), adaptability (F for fixed, A for adaptable), and hierarchy (H for hierarchical, G for global).

Table 2.1: Taxonomy of the chosen pooling methods. T: Trainability, nT: Non-trainable. D: Density, S: Sparse. F: Fixed, A: Adaptable. H: Hierarchical, G: Global.

Method	T	nT	D	S	F	A	H	G
DiffPool, MincutPool	✓		✓		✓		✓	
TopKPool, SAGPool	✓			✓		✓	✓	
Graclus		✓		✓		✓	✓	
SortPool		✓	✓		✓			✓
Set2Set	✓		✓		✓			✓

Source: Adapted from Grattarola et al. (2022)

2.7.1.1 DiffPool

Proposed by Ying et al. (2019), DiffPool was the first algorithm that offers a learnable hierarchical representation of graphs. DiffPool addresses the limitations of traditional GNNs explained earlier by providing a general framework to construct deep multi-layer learning models with a differentiable module that can hierarchically pool graph nodes.

Given a graph represented as $G = (A, F)$, where A is the adjacency matrix and F is the node feature matrix, DiffPool aims to generate a hierarchical representation of the graph, which is done by stacking multiple GNN modules and learning to assign nodes to clusters at each layer. The main idea behind DiffPool is to use a learnable assignment matrix in each layer to coarsen the graph representation with a smaller number of nodes. The assignment matrix $S^{(l)}$ is learned using a separate GNN, which uses the node embeddings generated by the first GNN and the adjacency matrix for computing this assignment matrix.

2.7.1.2 MinCutPool

The Mincut Pooling algorithm (BIANCHI; GRATTAROLA; ALIPPI, 2020) is a graph pooling method inspired by the minCUT problem in graph theory.Given a graph $G = \{V, E\}$, the goal is to partition V into K disjoint subsets. This partitioning is achieved by minimizing the volume of edges removed, which can be viewed as:

$$\sum_{k=1}^{K} \frac{\sum_{i,j \in V_k} E_{i,j}}{\sum_{i \in V_k, j \notin V_k} E_{i,j}}$$

Mathematically, the mincut problem is expressed as:

$$\text{maximize} \frac{1}{K} \sum_{k=1}^{K} \frac{C_k^T A C_k}{C_k^T D C_k}$$

subject to $C \in \{0,1\}^{N\times K}$ and $C1_K = 1_N$, where C represents the cluster assignment matrix, D is the degree matrix, and A is the adjacency matrix.

MinCutPool employs a continuous relaxation of this problem where the critical element being optimized is the cluster assignment matrix C. This optimization aims to capture the solution typically identified by Spectral Clustering (SC) and incorporates node features to discern clusters. Its unsupervised loss function is composed of two terms: the cut loss, which drives strongly connected nodes to cluster together, and the orthogonality loss, which promotes cluster orthogonality and uniform cluster sizes and is essential to make the clusters as distinct from each other as possible. The cut loss achieves its maximum value of 0 when cluster assignments are orthogonal with the goal of achieving optimal partitioning. After these losses are computed, they are then combined with a specific-task loss, such as graph classification in our case. This approach helps the pooling operation to preserve the graph community structures and achieve the task-specific objectives.

2.7.1.3 TopKPool

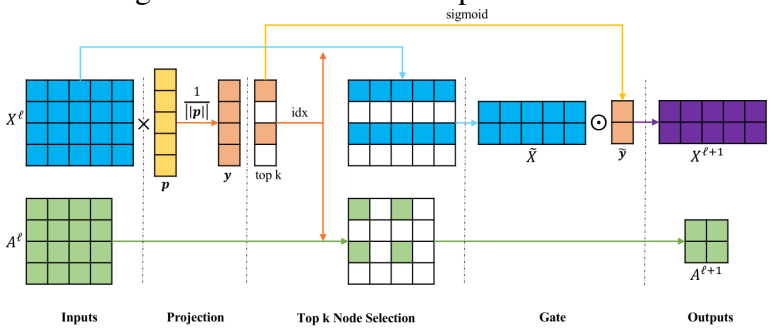
TopK pool (GAO; JI, 2019) is a learnable node-drop algorithm that uses a projection vector to transform nodes into scores. The idea is to select just the TopK scores nodes in the current layer along with their edges to get the results and use them at the next layer.

Additionally, TopK pooling uses a trainable projection vector p to compute the scores of the nodes as $y_i = \frac{x_i^\top p}{\|p\|}$ where yi measures how much information of node i can be retained when projected onto the direction of p. After the computation, a ranking function is employed and retrieves the indices of the top-scoring nodes. Finally, these scores $\hat{y}$ serving as a gate defined by $\hat{y} = \text{sigmoid}(y(\text{idx}))$, will be used to obtain the node feature in the next layer $X^{\ell+1} = \tilde{X}^\ell \odot (\hat{y}\mathbf{1}^\top)$. This gate mechanism is an essential part of the process, enabling differentiability and compatibility with existing GNN architectures.

As a sorting method, TopK Pool doesn't have an unsupervised loss to train on. That's because there's no obvious unsupervised loss for the ranking function. This means that TopKPool uses the task-specific loss to train the model.

The whole process can be seen in the figure 2.2 extracted from the original paper (GAO; JI, 2019)

Figure 2.2: Overview of TopKPool Method



Source: Gao and Ji (2019).

2.7.1.4 SAGPool

The self-attention graph pooling(SAGPool) (LEE; LEE; KANG, 2019) algorithm is also a learnable node-drop and sort-based method. The main idea is to employ a GNN to compute the self-attention scores, focusing on the entire graph topology as well as node features instead of independent node features such as TopK Pool.

Given a graph represented as $G = (A, X)$, where A is the adjacency matrix and X is the node feature matrix, the self-attention score Z is calculated by the GCN as $Z = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{\text{att}}\right)$, where σ is the activation function, $\tilde{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-connections, $\tilde{D} \in \mathbb{R}^{N\times N}$ is the degree matrix of $\tilde{A}$, $X \in \mathbb{R}^{N\times F}$ is the input features of the graph with N nodes and F-dimensional features, and $\Theta_{\text{att}} \in \mathbb{R}^{F\times 1}$ is the parameter of the SAGPool layer. In the same way that TopKPool, the scores are used to select the indices of the k best-scoring nodes, where k is a hyperparameter that determines the number of nodes to keep.

$$\text{idx} = \text{top-rank}\left(Z, \lceil kN \rceil\right)$$

.

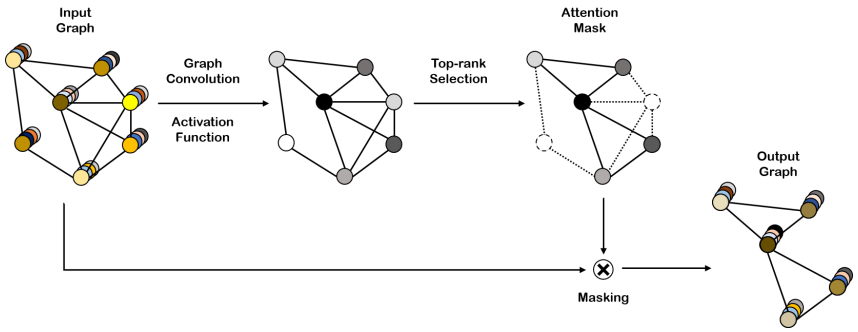
The authors also provide a comprehensive schema of the architecture, shown in Figure 2.3

2.7.1.5 Graclus

An older approach to hierarchical clustering is Graclus (DHILLON; GUAN; KULIS, 2007). This method introduces a multilevel algorithm for graph clustering, focusing on optimizing weighted graph clustering objectives.

Given a graph represented as G=(V,E,A), where V represents vertices, E edges,

Figure 2.3: Overview of SAGPool Method



Source: Lee, Lee and Kang (2019)

and A the adjacency matrix, Graclus assigns a weight w_i for each node i and each cluster $V_c \subseteq V$, the total weight of the cluster is calculated by the formula $w(V_c) = \sum_{i \in V_c} w_i$. This formula sums the weights w_i of all nodes i that belong to the cluster V_c.

By coarsening the graph at each level without the need for eigenvector computation, Graclus shows that the algorithm can be efficient for large-scale graphs and yet provides good clustering quality.

2.7.1.6 Set2Set

Based on the concept of global representation in graph neural networks, Set2Set (VINYALS; BENGIO; KUDLUR, 2016) makes use of a sequential mechanism that aggregates the feature representation of each node into a single graph representation. Set2Set is a technique that effectively captures the overall context of the graph by generating an order invariant embedding for its nodes.

In other words, the algorithm starts with an initial query vector and updates it using an LSTM at every step, according to graph nodes. Softmax product of node features xi and the query vector qt results in an attention score that leads to a context vector rt that represents a weighted sum of node features. The attention score is defined by

$$\alpha_{i,t} = \text{softmax}(x_i \cdot q_t)$$

and the context vector $\mathbf{r}_t$ by the weighted sum of the node features:

$$\mathbf{r}_t = \sum_{i=1}^{N} \alpha_{i,t} x_i$$

The final embedding for the graph is the concatenation of the series of query and con-

text vectors across the steps, which captures the essential graph features. This process is iteratively refined through training, resulting in the production of interpretable graph embeddings suitable for the given task.

2.7.1.7 SortPool

Another global pooling algorithm is SortPool (ZHANG et al., 2018). The main idea of this method is to sort the feature descriptors, each of which represents a vertex, to later pass them into traditional 1-D convolutional and dense layers. SortPool employs the Weisfeiler-Lehman (WL) graph labeling method as a preprocessing step to sort the graphs based on their structural roles within the graph.

Mathematically, the SortPool method process can be represented as:

$$Z_{\text{sorted}} = \text{Sort}(Z_{1:h}, Z_h)$$

Where Z_{sorted} is the reordered tensor after SortPooling, with $Z_{1:h}$ being the input tensor of vertex feature descriptors and Z_h the sorting key-based on the last graph convolution layer's output, which the authors also indicate that signifies the most refined continuous Weisfeiler-Lehman (WL) colors. After that, the method adjusts the sizes of the output tensors from the original size n to a kernel size k, with the intention of unifying graph vertice sizes.

2.8 Saliency Maps

Proposed by Simonyan, Vedaldi and Zisserman (2014), saliency maps based on gradients are a baseline approach for computing input attribution, thereby offering a way to model explainability. Saliency examines the outputs of the model, which, in classification models, are the scores for each target class, specifying the probability of the input belonging to each class. It uses the first-order approximation of the score S_{c}, where S_{c} is the the score for a class c. Mathematically, the score S_{c} given an input x_0 can be viewed as:

$$S_c(\mathbf{x_0}) \approx \mathbf{w}^T\mathbf{x_0} + b \tag{2.10}$$

where **w** is given by :

$$\mathbf{w} = \left.\frac{\partial S_c}{\partial \mathbf{x}}\right|_{\mathbf{x_0}} \tag{2.11}$$

We can then use these scores to consider what are the most important features of the model since the gradients are the coefficients of each feature, and the absolute value of these coefficients can be taken to represent feature importance and seen as a measure of how little certain features need to change to alter the output of the network significantly.

3 RELATED WORK

In this chapter, we analyze the related works that approached graph neural networks for genomic classification tasks. The goal here is to overview the methodology and results found by other works in the same field. Besides that, we contrast the lack of focus on the choice of the pooling algorithm.

Recently, there has been a significant increase in the application of graph neural network models for various cancer-related classification tasks. Ramirez et al. (2020) investigate graph convolutional neural networks for expression-based cancer type classification. The authors used two types of graphs for the GCNN models: co-expression graphs based on gene expression correlations and Protein-Protein Interaction (PPI) graphs derived from the STRING database, with both models having a version with singletons, i.e., nodes that have no neighboring nodes, and a version removing singletons. Their approach was relatively close to ours: They applied a convolutional layer followed by a pooling layer that was then passed to a single dense, fully connected neural network. However, they restricted the model to one layer, and the chosen pooling method was a greedy algorithm that chose an unselected node to be paired with another unpaired neighbor node. Additionally, they applied an *in silico* gene perturbation, where they evaluated the genes that most changed the accuracy when perturbed as the most discriminative genes.

Later, the same authors integrated a clinical layer for cancer survival prediction (RAMIREZ et al., 2021). A similar approach was implemented, using a Cox layer instead of a fully connected neural network to predict patient risk scores. To interpret the model this time, the authors utilized their GCNN model to get significant nodes in the hidden layer that affect the risk score, which is the final prediction of the Cox layer, and examined the relationship between gene expression levels and these nodes.

Li, Wang and Nabavi (2021) and Li et al. (2022) proposed graph convolutional networks for multi-omics data. The former applied a GNN model based on a convolutional layer, in which they used a ChebNet layer. Following the results of the GNN layer, they apply a pooling mechanism that is then concatenated with a two-layer FC network that is trained in parallel with the GNN since the authors argue that this approach helps the framework better extract the overall sample feature representation. The concatenated vector is then passed to a classification layer to get the final prediction. The latter used a similarity network fusion to construct a patient similarity network (PSN) that was then used as input to a convolutional layer, in which they used the classic GCN proposed by Kipf

and Welling (2017) followed by a classification layer, without the intermediary pooling layer. Subsequently, the same authors in Li, Wang and Nabavi (2021) did a comparative analysis of graph-attention-based models using the GAT layer and the well-established GCN models in Li and Nabavi (2023) wherein they concluded that the GAT models work better for smaller graphs while GCN models for larger graphs.

Another work that employed graph convolutional networks for genomic classification tasks was proposed by Hayakawa et al. (2022). The authors produced a framework using KEGG pathways to classify subtypes of diffuse large B-cell lymphoma. The GCN model consisted of two convolutional layers followed by one pooling layer, which was an average pooling, and similar to the other works, the resultant vector was passed to a fully connected network to make the final prediction. The authors also did a feature importance analysis using the SHapley Additive exPlanations (SHAP) (LUNDBERG; LEE, 2017), where they summarized the shapley values across samples to get the pathways' importance.

Chereda et al. (2021) also proposed a graph convolutional network applied to breast cancer data for metastasis prediction using a ChebNet convolutional layer followed by a max-pooling and a two-layer fully connected network. The innovative approach comes in how the post-training explainability is done using the Graph Layer-wise Relevance Propagation (GLRP) algorithm, which, in the end, generates explanations in the form of relevant subgraphs for each data point and allows an interpretation of the molecular subnetworks that are individual for each patient.

Distinct from the previous works, the studies in Wang, Bai and Nabavi (2021) and Yin et al. (2022) explored a related but yet different area, which is the classification of cell types. Using single-cell RNA sequencing (scRNA-seq), the authors in Yin et al. (2022) craft a framework consisting of a GRAPHSAGE convolutional layer to generate the embeddings that are then concatenated and passed to a two-layer neural network to predict the cell type. Wang, Bai and Nabavi (2021) work is quite similar to their previous work in Li, Wang and Nabavi (2021), where they also make use of a parallel neural network together with a GCN, but this time for classifying cell types with scRNA-seq.

Furthermore, Grattarola et al. (2022) proposed a formal characterization for pooling methods based on three operations: selection, reduction, and connection (SRC). The authors categorize more than thirty existing pooling methods under this framework and introduce a taxonomy based on specific properties of the SRC functions. Additionally, they also provided benchmarks on established graph classification datasets such as Protein and

Mutagen. Liu et al. (2022) also studied pooling in graph neural networks, proposing a taxonomy including flat and hierarchical pooling, with hierarchical pooling further divided into node clustering and node drop pooling.

4 METHODOLOGY

This work is divided into three different kinds of experiments. Experiment one relates to an investigation of the scalability of pooling methods due to the number of nodes in the graph, specifically examining how increased node counts affect computational demands and algorithmic efficiency. This experiment explores how larger graphs can result in higher computational costs, challenging the scalability of different pooling methods approaches. Experiment two analyzes the predictive power of various GNN architectures by varying pooling methods and convolutional layers and interpreting results. The goal here is to discern how these variations affect the overall performance of the models in a genomic environment, aiming to optimize the GNN design. In experiment three, we analyze the saliencies of the graph neural network to try to understand which features were most important to the model making the prediction. In the next sections, we will describe in more detail the data and models used for each experiment.

4.1 Experiment One - Time and Memory Usage Analysis

As commented in Section 2, pooling methods can be divided into categories, where the form that each of these methods handles the graph changes significantly. To analyze the impact of these changes, We conducted an experiment to measure the impact of increasing graph nodes in terms of memory usage and time consumption. An algorithm was used to generate the synthetic data with the same number of nodes at each graph, and a fixed GNN model architecture was used to maintain consistency in the results.

4.1.1 Synthetic Data Generation

In this work, the synthetic data is obtained through a fixed graph topology, employing the Erdős-Rényi (ERDöS; RéNYI, 2006) model as the foundational framework to mimic the characteristics of real-world PPI network data, similar to the data used for experiment two. The Erdős–Rényi model is a classic approach for creating random graphs. It starts with a set of isolated nodes and then connects each pair of nodes with a predetermined probability. This process results in a graph where the presence of an edge between any two nodes is equally likely, reflecting a purely random network structure.

The model is instantiated with a predefined number of graphs set at 500 and an edge creation probability of 0.05. Furthermore, we create graphs with 100, 300, 500, 700, 1000, 3000, 5000, 7000, 8000, and 9000 nodes using this framework to test the scaling of pooling methods.

As our objective with this experiment was to see the impact on memory and time rather than predictive power, we generated a one-dimensional feature vector with random values ranging from zero to one, again mimicking the real data. Besides that, we made it a binary graph-level classification task by assigning a label to each graph based on the mean value of its random feature vector.

4.1.2 Network Architecture

To ensure consistency, we used a fixed GNN architecture and only modified the pooling method for each comparison. The architectural configuration is adjusted to align with the categories of the pooling method employed, global or hierarchical. Global pooling methods are designed to aggregate information on the entire graph, focusing on capturing a representation of the graph's overall structure, while hierarchical methods allow the graph to be coarsened multiple times.

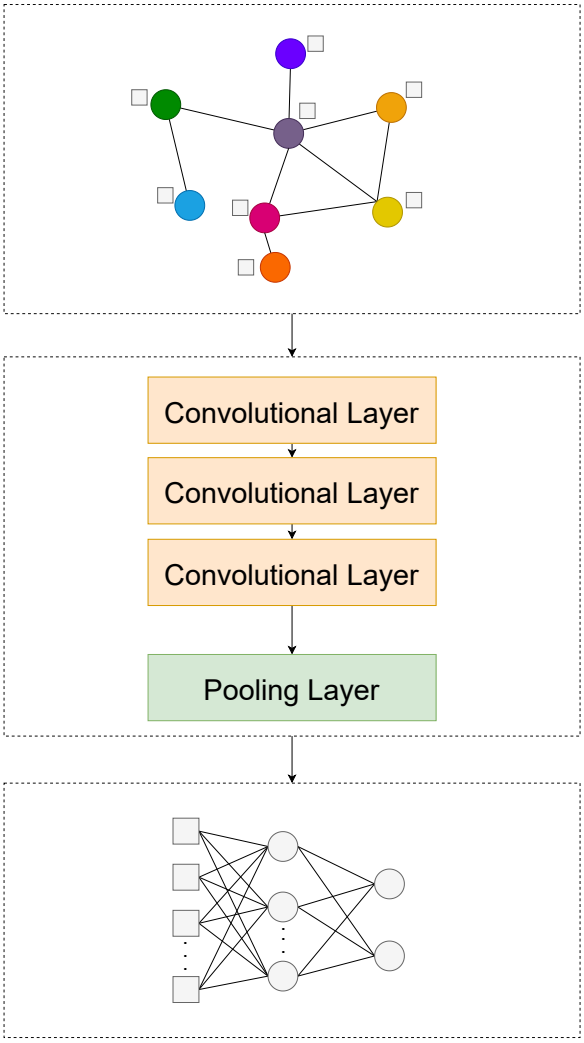
The architecture with global pooling methods was set to three convolutional layers, each followed by a non-linear function. After these layers, a single global pooling operation selects the nodes, which are then flattened and fed into a single-layer fully connected network with 256 hidden units, batch normalization, and dropout rate of 0.1. The whole process can be seen in Figure 4.1

In hierarchical pooling architectures, unlike the straightforward three-layer convolutional structure followed by global pooling, the architecture integrates three coarsening layers. The coarsened layer is composed of a convolutional layer followed by a non-linear function and a hierarchical pooling operation to extract features at each level. The hierarchical model architecture can be seen in 4.2.

As a convolutional layer, we used GCN, which is a well-established baseline model (HAMILTON, 2020). The number of hidden units on the convolutional layer was set to 32, and the fully connected network is the same as the global pooling.

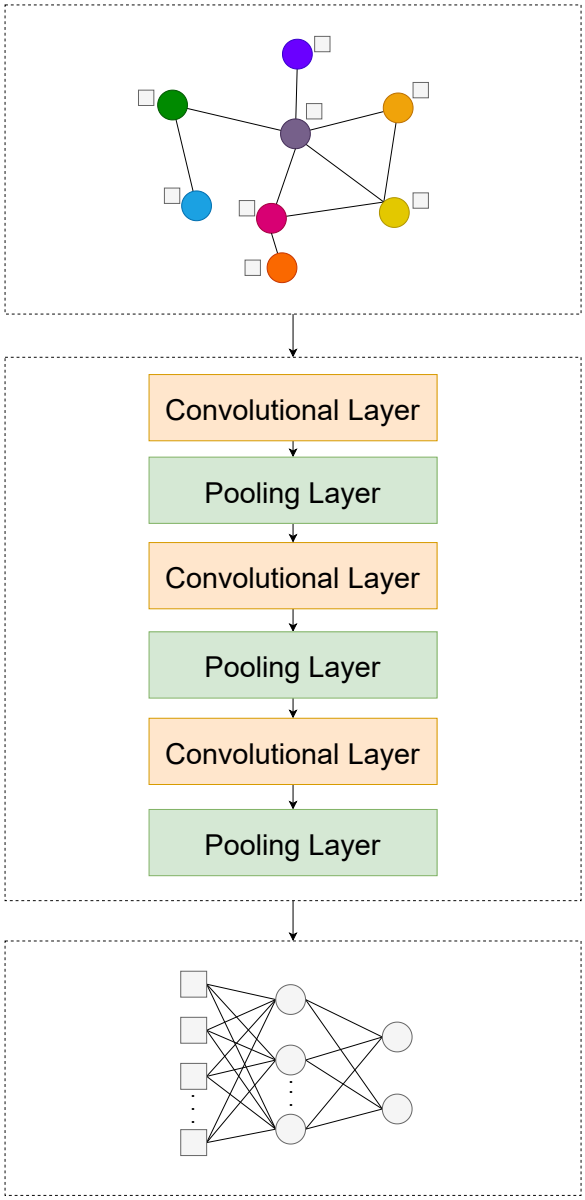
We tested this architecture using various pooling methods, including DiffPool, MincutPool, TopkPool, SAGPool, Graclus, SortPool, and Set2Set, as discussed in Section 2.7.1.

Figure 4.1: Global Pooling Architecture



Source: The Authors

Figure 4.2: Hierarchical Pooling Architecture



Source: The Authors

4.1.3 Training and Metrics

For each graph size in the range of 100 to 9000 nodes, we conducted three independent runs to ensure that small computational fluctuation would not affect the correct evaluation. We trained the model for 30 epochs. Training time and peak GPU memory usage were recorded for each run and graph size using Python's Time and Pytorch libraries, respectively.

4.2 Experiment Two - Predictive Power on Genomic Data

In this experiment, our focus shifted to evaluating the impact of different pooling methods on the performance of graph neural networks in the context of genomic data. Specifically, we examined RNA-seq data for Breast Invasive Carcinoma (BRCA) from The Cancer Genome Atlas (TCGA). The main objective was to investigate how different pooling methods and convolutional layer choices affect the model's performance. We aimed to enhance our comprehension of the relationship between GNN pooling methods, convolutional layers, and their effectiveness in genomic data to advance GNN applications in biological research.

4.2.1 Data

The data used for this experiment was RNA-seq data for Breast invasive carcinoma (BRCA) from The Cancer Genome Atlas (TCGA), choosing specifically the upper-quartile FPKM (UQ-FPKM) data provided via Xena (GOLDMAN et al., 2020). The data were normalized with the GDC pipeline, explained in Section 2.1.2, and then we eliminated any genes and samples that had over 20% missing values. We also normalized our features to follow a Gaussian distribution using a log transformation to the FPKM values.

After the transformation, the dataset has graphs containing 14133 genes, and a sample of the data can be seen in Figure 4.3, where the rows are the samples from the TCGA database and the columns represent each gene expression value. Furthermore, we used one more preprocessing step to make the comparison tests for all pooling methods possible regarding time and memory. A heavy-edge matching algorithm was used to coarsen the graph by half two times, thus making the final graph containing 3534 genes.

Figure 4.3: Sample of the dataset. The rows represent samples from the TCGA database, and the columns represent the gene expression values

	Gene 1 Expr	Gene 2 Expr	⋮	Gene 14132 Expr	Gene 14133 Expr
TCGA-3C-AAAU-01A	0.220	-1.467	⋮	0.979	-1.035
TCGA-5L-AAT0-01A	0.200	-0.940	⋮	-0.246	0.980
TCGA-A2-A0CQ-01A	1.644	-0.377	⋮	0.118	-0.0693

Source: The Authors

The heavy-edge matching algorithm works by taking the edges of a graph and forming pairs of nodes based on the weight of these edges. It identifies the heaviest edges that connect nodes not already part of a pair and merges their connected nodes into single nodes, generating a coarsened graph with 50% of the original size.

However, we also tested how this preprocessing step, which acts as a fixed pooling layer previous to the model, impacts the performance of the model. For methods that we could test, which were TopK, SAG, Sort, and Set2Set2, different numbers of times that we applied the preprocessing step, resulting in graphs with 14133 nodes with no preprocessing step, 7067 nodes with one step, 3534 nodes, which was the number that we used for all methods, 1767 nodes with three steps, and 884 nodes with four steps of the heavy-edge matching algorithm.

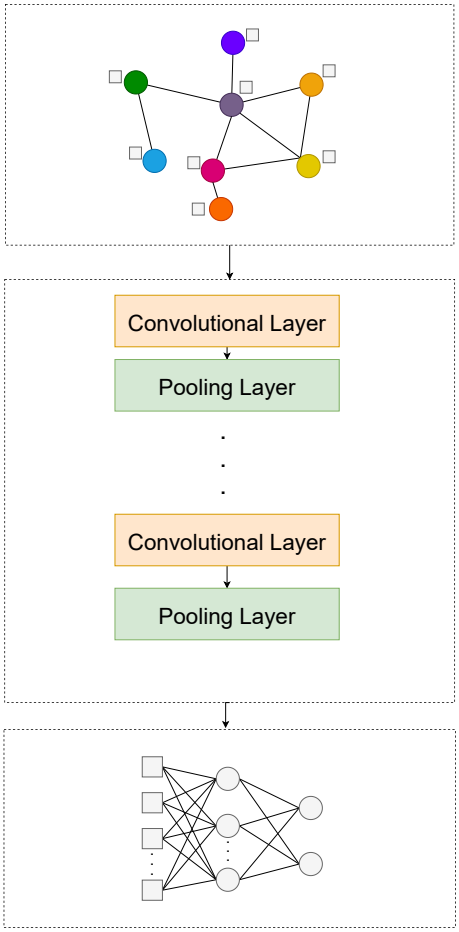
4.2.2 Network Architecture

Experiment two's network architecture is almost identical to experiment one network architecture, except for the variation in the convolutional layer, which was fixed in the previous experiment Section (4.1) and the number of hidden units on the convolutional layer that was set to 16 due to the larger number of edges in the TCGA data when compared to the synthetic data, which caused an increasing in memory and thus we reduced the number of hidden units. In this experiment, we used GraphSAGE, GCN, and GAT as convolutional layers and combined them with our selected pooling methods so we could evaluate their impact on the overall performance.

The network was repeated for each level of the coarsening layer to assess the impact of varying coarseness on our model's performance. In this context, levels represent distinct degrees of data coarsen within the coarsening layer, each providing a different

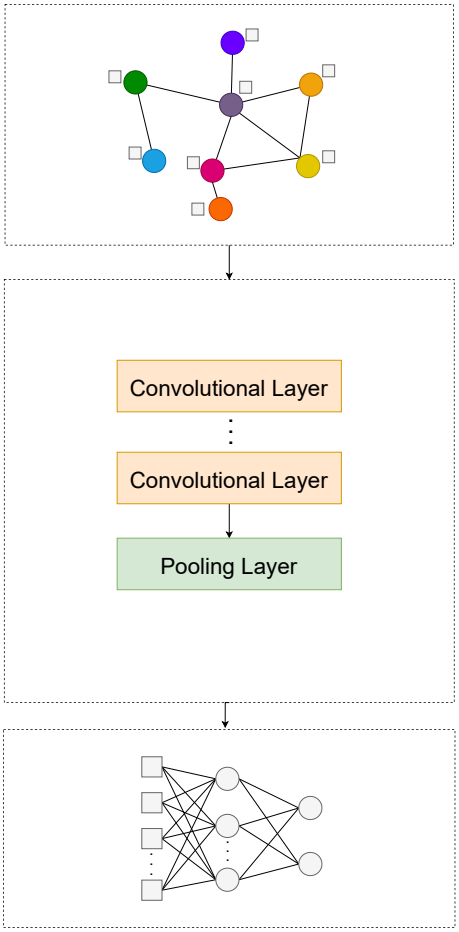
resolution or granularity of the genomic data. By experimenting with these various levels, we aimed to understand how changes in data coarseness influence the model's ability to learn and predict. The final architecture can be seen in Figure 4.4.

Figure 4.4: Model Architecture for Experiment Two



(a) Model Architecture for Hierarchical Pooling Methods

Source: The Authors



(b) Model Architecture for Global Pooling Methods

Source: The Authors

4.2.3 Training and Metrics

The training process was also similar to 4.1 but a bit more complex due to the type of analysis we conducted and to ensure that we could get the best performance for each model we trained.

The chosen loss function was the well-known cross-entropy loss but with a slight modification to adjust to our problem. A weighted version of the cross-entropy loss was implemented for the experiment due to the imbalance between the classes, which is com-

mon in genomic datasets. This can be seen as a form of regularization, as the mistakes in the minority classes have a higher penalty that is proportional to the inverse of their sizes. The weight for a class $c \in 1, ..., C$ is $w_c = \frac{N}{CN_c}$, where N represents the total count of samples in the dataset, and N_c indicates the number of samples in the specific class c. The weighted cross-entropy loss formula is given by:

$$L_{\text{ce}} = -\sum_{n=1}^{N} w_{y_n} \cdot \left(y_n \log \left(\frac{\exp(h_{n,y_n})}{\sum_{c=1}^{C} \exp(h_{n,c})} \right) \right) \tag{4.1}$$

Where y_n represents the true class label for sample n, with $y_n \in \{1, \ldots, C\}$.

It is important to note that for pooling methods with secondary losses, we added them to the weighted cross-entropy loss to train the model.

The selected optimizer was AdamW (LOSHCHILOV; HUTTER, 2019) and the scheduler for updating the learning rate dynamically while training was done with cosine annealing without warm restarts (LOSHCHILOV; HUTTER, 2017). The AdamW optimizer was selected for its approach concerning weight decay regularization, implementing a variation from the traditional Adam optimizer by decoupling weight decay from gradient updates with the promise of improving training performance in deep learning models. Complementing this, the learning rate was dynamically adjusted using a cosine annealing scheduler without warm restarts, which decreases the learning rate following a cosine curve at each epoch, enhancing the convergence by avoiding an abrupt change in the learning rate.

As in Section 4.1, we have used 30 epochs to train the model. Furthermore, we have used 3-times stratified repeated holdouts for evaluating performance. Stratified holdout is a method of cross-validation used in machine learning where the dataset is split into training and testing sets multiple times in a way that maintains the same proportion of classes or characteristics in each split as in the entire dataset. This approach ensures that each class or characteristic is represented in both the training and testing sets, making the evaluation more robust and reliable. The split between the train and test set at each holdout was settled as 80% for training and 20% for testing.

On top of that, 20% of the training set was designated for the validation set, which was used for tuning the hyperparameters. The hyperparameters selected for tuning the model were just the learning rate and the weight decay, given that the cost of tuning the hyperparameters was very high for some pooling methods, so we decided to tune the essential hyperparameters. Note that we just tuned the hyperparameters in the first

holdout run out of the three holdout runs and used them for the other two because of this cost issue.

When evaluating the performance of our models, we focused on the F1 score. We opted for the Macro version since we are dealing with a multi-class scenario. The Macro F1 score calculates the individual F1 scores for each class. Then, it averages them to provide a balanced measure that takes into account all classes equally. The formula for the Macro F1 score is:

$$\text{Macro F1 Score} = \frac{1}{C}\sum_{i=1}^{C} 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4.2}$$

Here Precision_i is calculated as the ratio of positives (TP_i) to the sum of positives and false positives $(TP_i + FP_i)$ while Recall_i is calculated as the ratio of true positives to the sum of true positives and false negatives $(TP_i + FN_i)$.

Furthermore, we compare the results with a baseline model, which is the same single-layer fully connected network model with 256 hidden units but without any convolutional or pooling layer in the middle.

4.3 Experiment Three - Interpretability

Experiment three is an implementation of the saliency algorithm explained in Section 2.8 on the best model architecture seen in experiment two. We used the final model in the first run, where we tuned the hyperparameters to perform the saliency analysis.

We then get the saliency values for each target class and proceed to identify the 15 most significant features. This is achieved by calculating the median of the normalized saliency values across all samples for each feature. The median, rather than the mean, is chosen as the measure of the central tendency to minimize the impact of outliers, thereby providing a more robust representation of feature importance.

Following the identification of the top 15 features based on median saliency values, we proceed to map back these features to their corresponding genes and then review the literature to understand the role these genes have played in previous BRCA research.

5 RESULTS

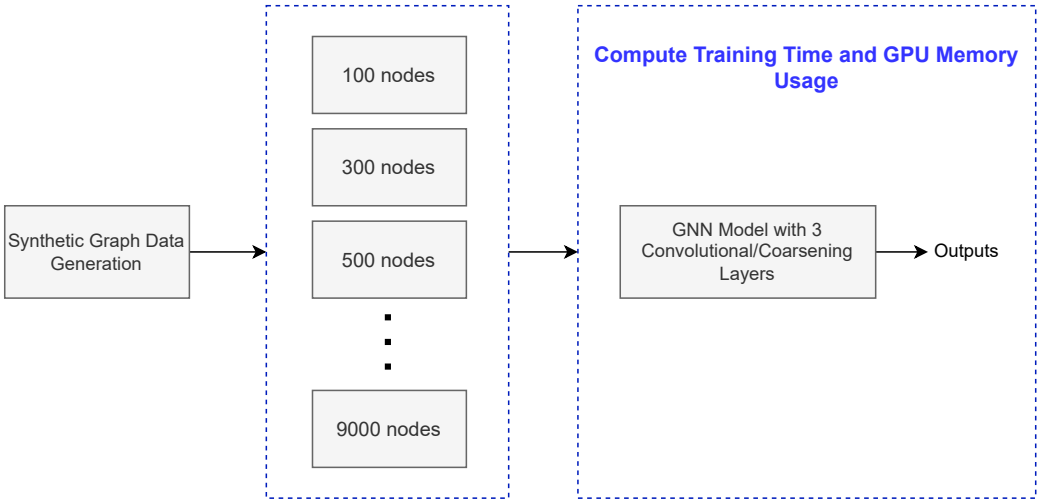
In this chapter, we present our findings, addressing three key questions raised in this work: How is the scalability of the current pooling methods, how do different pooling strategies and the choice of convolutional layers affect the performance of the GNN on genomic-related data, and how is explainability incorporated and evaluated within these models. Section 5.1 details the results for the first question and analyzes how the different categories impact the scalability of the selected pooling methods. Section 5.2 describes the results of the impact of modifying the GNN architecture, most specifically, the pooling method and convolutional layer, in the Breast Cancer dataset constructed over the STRING network. Additionally, in Section 5.3, we provide the analysis of the most important genes found within the model outputs and their previous correspondence in the literature.

5.1 Experiment One - Time and Memory Usage Analysis

As seen in Section 2.7, pooling is a fundamental component of GNN models, and studies as Grattarola et al. (2022) and Liu et al. (2022) have proposed different criteria for categorizing these methods. Thus, this experiment focuses on the scalability of those methods concerning time and memory usage by incrementing the number of nodes of the input graph. As explained in chapter 4.1 the data was artificially generated using the erdős-rényi method. The training was done with a 24GB GPU server hosted by the PCAD[1] infrastructure at INF/UFRGS, and with the pytorch geometric version[2] of the methods. Figure 5.1 summarizes the experiment.

[1] http://gppd-hpc.inf.ufrgs.br
[2] https://github.com/pyg-team/pytorch_geometric

Figure 5.1: Training Time and Memory Usage Experiment



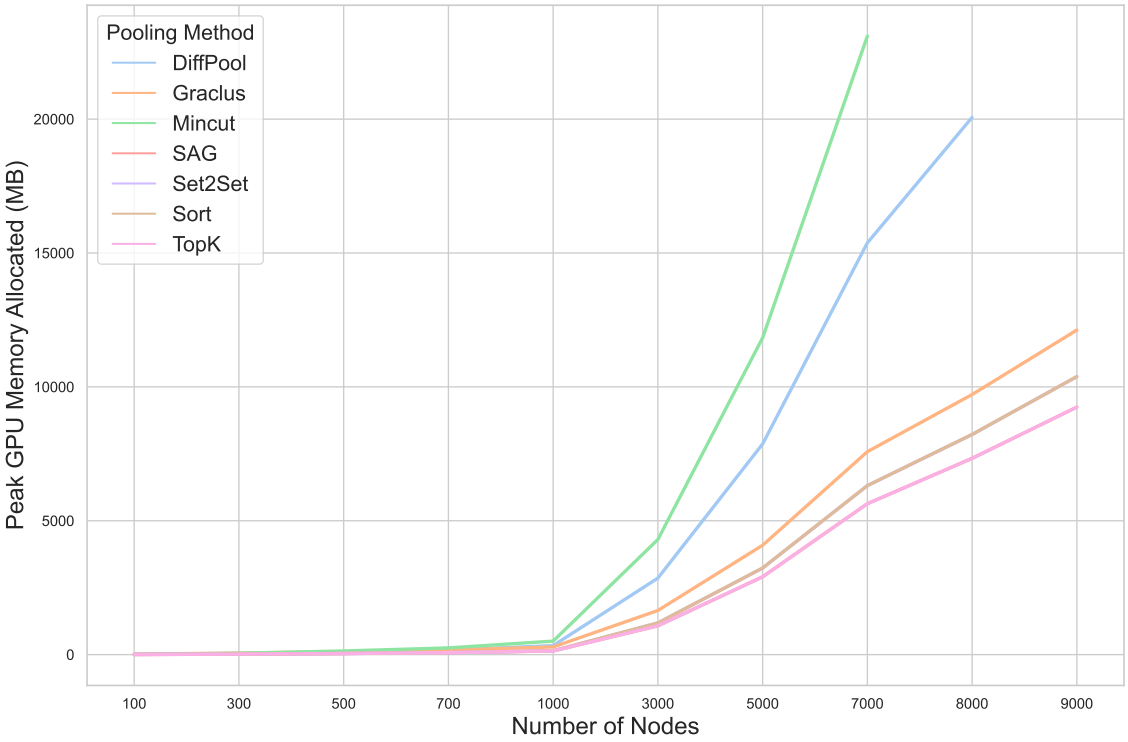
Source: The Authors

Figure 5.2 shows the memory usage of all seven selected methods. Up to about 1000 nodes, the memory allocation for all methods is relatively similar. The most consuming method at this point was Mincut, with almost 0.5 GB of memory consumed, while SAG and TopK consumed the least memory, using just 0.13 GB, indicating that at smaller scales, the choice between those may not significantly impact memory usage. However, both DiffPool and MinCut diverge from the rest by showing a much more abrupt increase in memory usage, getting up to 15 GB and 23 GB, respectively, at 7000 nodes, while the other methods keep the memory usage around 6 GB.

Referring back to Table 2.1, we can see that both DiffPool and MinCut are dense methods, along with Set2Set. Set2Set and Sort are a special case where it is classified as dense but does not need the input data to be dense as they only use dense operations internally, this way not consuming as much memory as DiffPool and Mincut. This suggests that while methods that require the input data in a dense format might be manageable for small to medium-sized graphs, their memory consumption becomes very massive as the graphs get larger. On the other hand, the sparse methods kept their memory usage increasing at a much slower rate, which makes them more suitable for computational biology and other domains where networks can grow to thousands of nodes.

The higher memory usage of dense methods like DiffPool and MinCut is largely due to their memory complexity, which comes from maintaining large dense matrices in the GPU's memory. In DiffPool, the assignment matrix is used for clustering nodes into supernodes scales with O(N×M), where N is the number of nodes and M is the number of supernodes, which is huge in large graphs. Similarly, MinCut also constructs a form of

Figure 5.2: Average peak memory usage by the number of nodes on the input graph across three executions



Source: The Authors

an assignment matrix but consumes more memory due to the calculation of regularization losses, which necessitate storing extra parameters related to the graph's connectivity and cluster assignments. In contrast, sparse methods maintain a more consistent and lower memory complexity, typically O(1) per supernode, making them more manageable and efficient for large graph structures.

Specifically, the SAG and TopK, which are both node-dropping methods according to Liu et al. (2022) classification, demonstrate the most efficient use of memory among the sparse methods, highlighting their potential utility in scenarios where the input data is large. Besides that, we see that Mincut exploded our GPU memory limit at 7000 nodes, while DiffPool with 8000 nodes.

The results were consistent across all pooling methods with almost zero difference in execution results.

Figure 5.3 provides the results of the average training time of various pooling methods concerning the number of nodes within the graph data. Up to the threshold of 1000 nodes, the pooling methods are again very close and maintain a relatively low and stable training time.

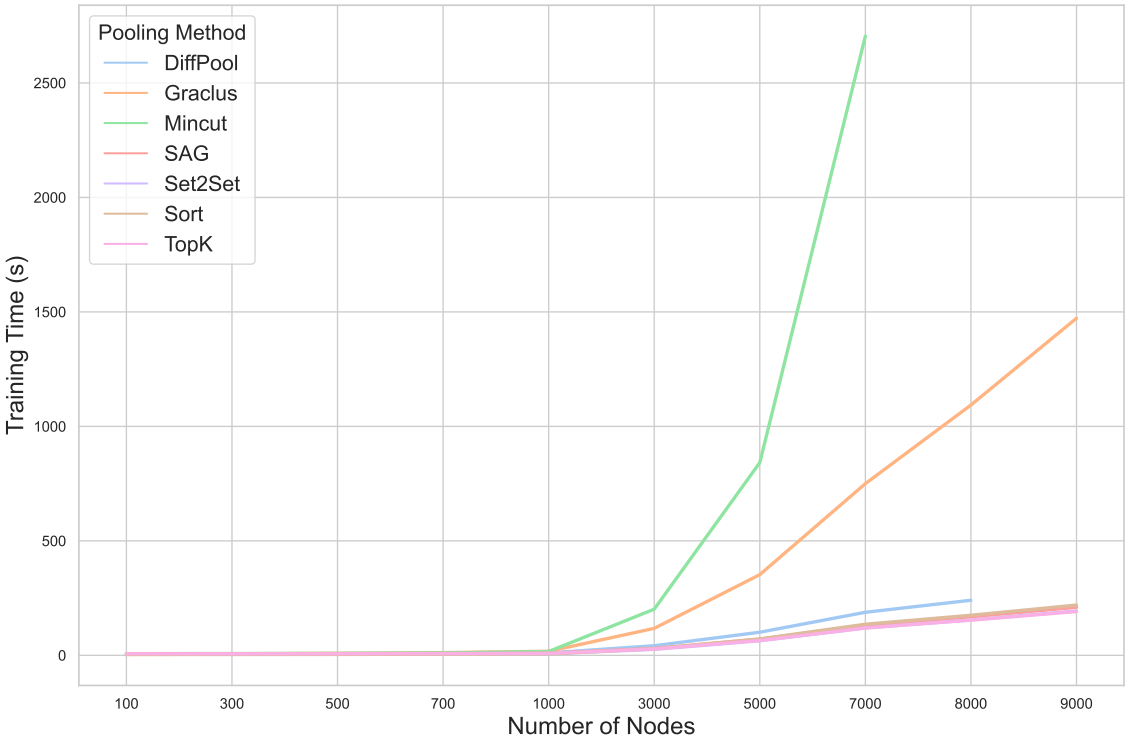
Mincut, represented by the green line, shows a dramatic increase in training time,

outpacing the other methods as the number of nodes grows, going from 16 seconds at 1000 nodes to 200 seconds at 3000 nodes and 2700 seconds at 7000 nodes, displaying very limited scalability in this aspect. Another method that stands out is Graclus, which is an older pooling method compared to others, and although it has a lower increase rate than Mincut, it can get to almost 1500 seconds of training time in scenarios with graphs of 9000 nodes. The remaining methods showcase a more gradual increase in training time, especially SAG, Set2Set, Sort, and TopK, with most methods having a maximum training time of about 200 seconds. Zoomed Figure 5.3 displays the granular difference between them, with TopK in particular, showing a little increase in training time, making it the most scalable method in this comparison, followed by Set2Set, SAG, and Sort. Nonetheless, all four of them are suitable for large graphs concerning the time taken to train the model. Furthermore, there are no apparent categories that influence training time, which indicates that the intrinsic characteristics of each method are the most determinant factor.

The difference in results between the executions was, again, negligible, as it was very close to zero for all pooling methods.

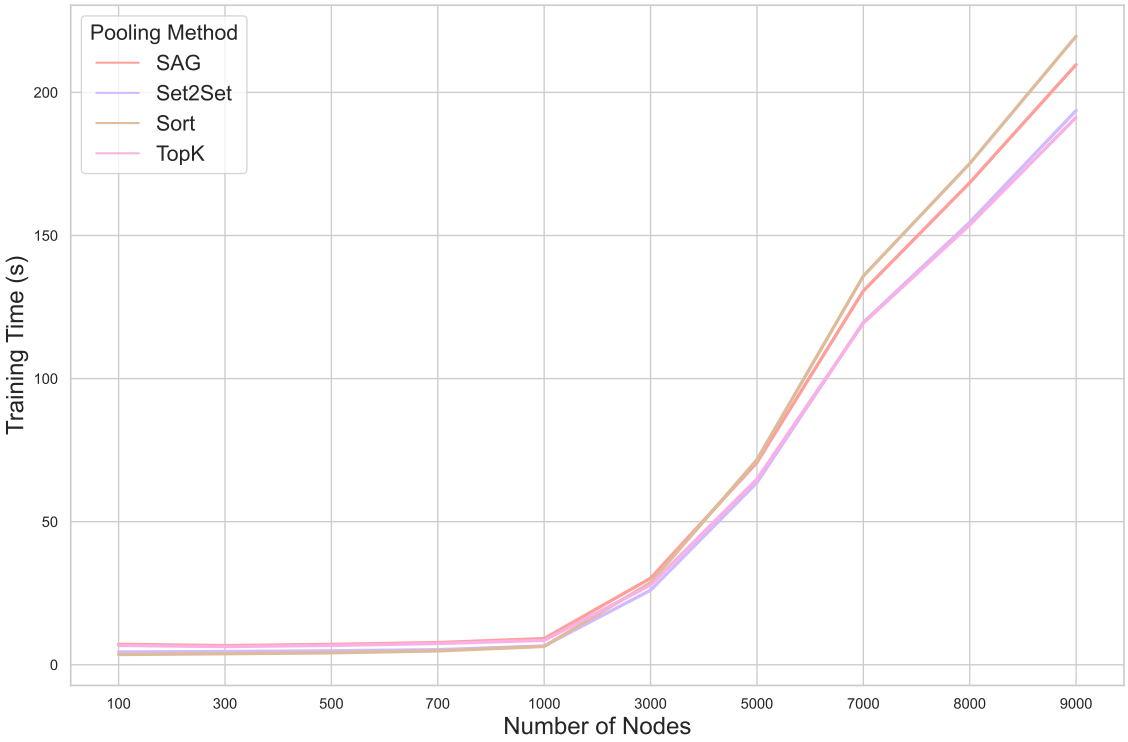
In the next section, we will present the results of using those methods on GNN architectures with breast cancer data and discuss whether the scalability characteristics and limitations of these approaches yield a significant impact on the model performance.

Figure 5.3: Average training time in seconds by the number of nodes on the input graph across three executions



Source: The Authors

Figure 5.4: Zoomed average training time in seconds by the number of nodes for SAG, Set2Set, Sort, TopK across three executions

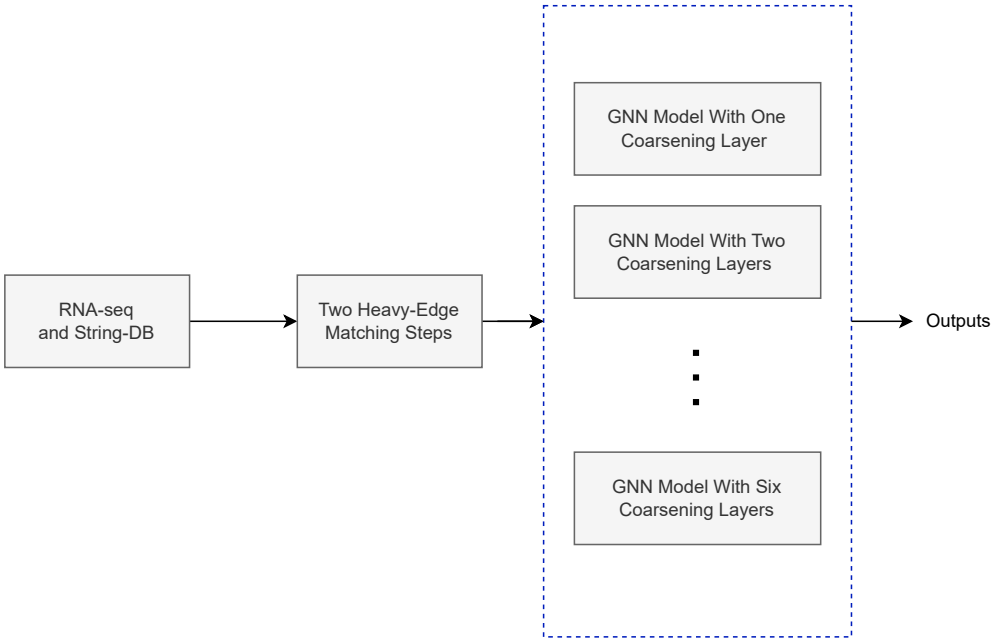


Source: The Authors

5.2 Experiment Two - Predictive Power on Genomic Data

In this section, we discuss the model's performance assessment concerning the different GNN architectures on the BRCA dataset. We focus on how the pooling methods and convolutional layers affect the model's performance. Specifically, we focus on the hierarchical pooling methods evaluating the performance of the models using 1 to 6 coarsening layers and comparing them with global pooling methods using 1 to 6 convolutional layers. Figure 5.5 summarizes the experiment.

Figure 5.5: Experiment performed on the influence of the GNN architecture on the results



Source: The Authors

Figure 5.6 shows the F1 score for each pooling layer and convolutional layer on the three holdouts runs at each level. At first glance, we can see that the SAG method outperforms the other methods by a considerable margin with the best F1 score of 0.834 with SAGE architecture and two levels, being the only method that comes close to the single-layer fully connected neural network used as the baseline that has a value of 0.859. Table 5.1 displays in more detail the average f1 macro score as well as the standard deviation of the three holdout runs with the best F1 score at each level highlighted in green. The results reinforce the advantage of SAG, especially with the SAGE architecture being the best combo in 5 out of the six levels and the best overall. In the case of the GCN convolution, the SAG pooling method again showed superior performance at most levels, with average F1 scores like 0.833 at level 2 and 0.810 at level 1, but drastically falling in performance with four or more levels. The GAT architecture had significantly lower performance and more variability at the best pooling method in the experiments, which could be due to the fact that we used just one attention head since increasing the number of heads made the model out of memory. For instance, at level 1, the SAG pooling method had the highest average F1 score of 0.582, at level 2 the Mincut pooling method led with a F1 score of 0.476, and at level 3, the Graclus method was the best with a F1 score of 0.336.

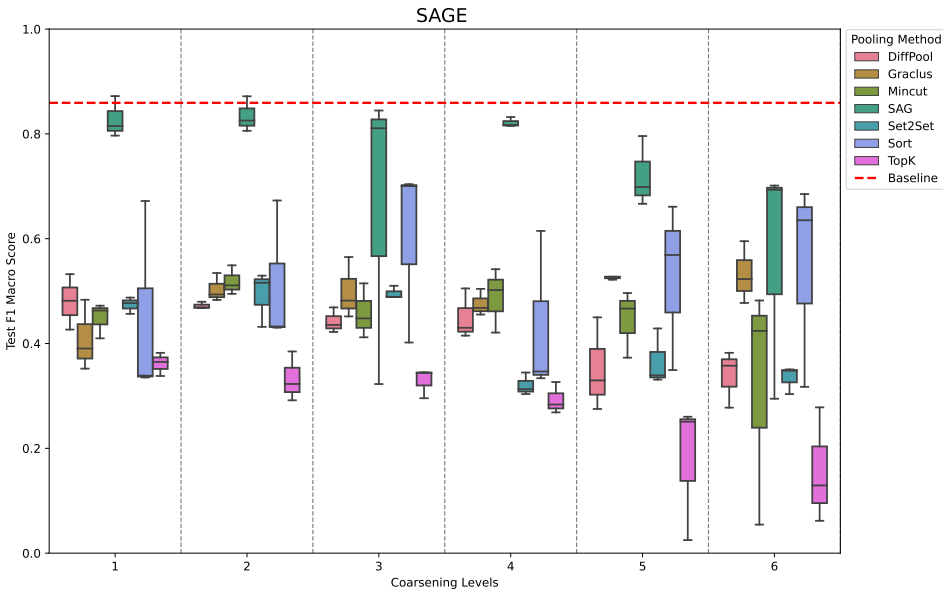
The results with SAGE showed that adding a second coarsening layer performed better than with just one layer. However, a downtrend is observed when we increase the

number of coarsening layers after two for almost all pooling methods, except Graclus and Sort, the last, nonetheless, has a big variance in the performance across the holdout runs. Going back to table 2.1, we see that both Graclus and Sort are non-trainable methods, and both take advantage of increasing the number of coarsening layers, while trainable methods' performance starts to decrease after the first or second coarsening layer.

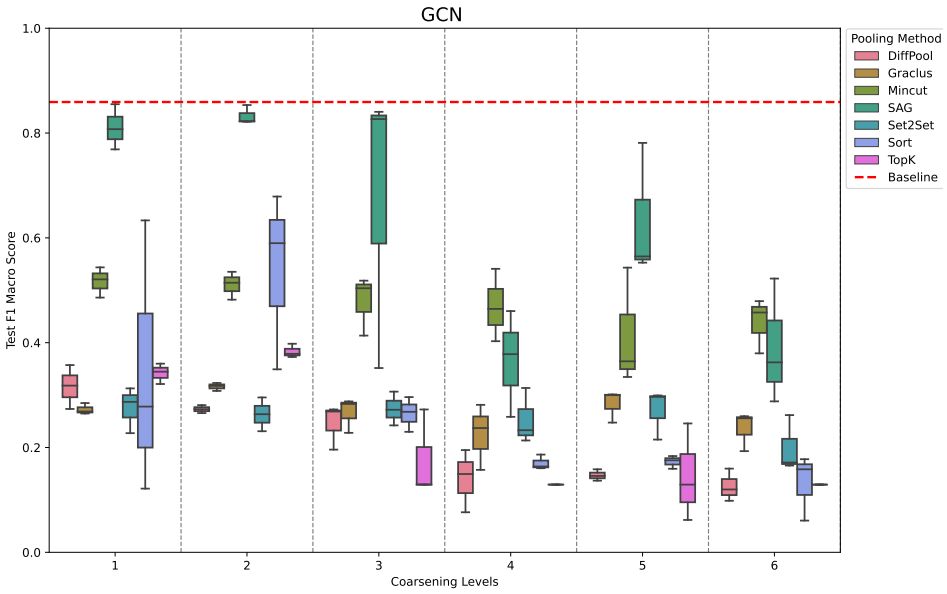
Another interesting aspect is that, despite having the same pooling taxonomy, Top-K did not perform as well as SAG, according to their F1 scores. This finding is significant because it shows that just because two methods are similar in theory, it doesn't mean they will work the same way in practice.

Comparing the results in this section with the results of Section 5.1, it is clear that the investment for both training time and memory usage does not result in a good performance. Especially, the mincut method, which is the higher memory and time-consuming method, yields a maximum average F1 score of 0.582, which puts it in the intermediary performance category together with Sort, but far below the SAG method that achieved at best a 0.834 average F1 score, and has a faster training and consumes less memory.

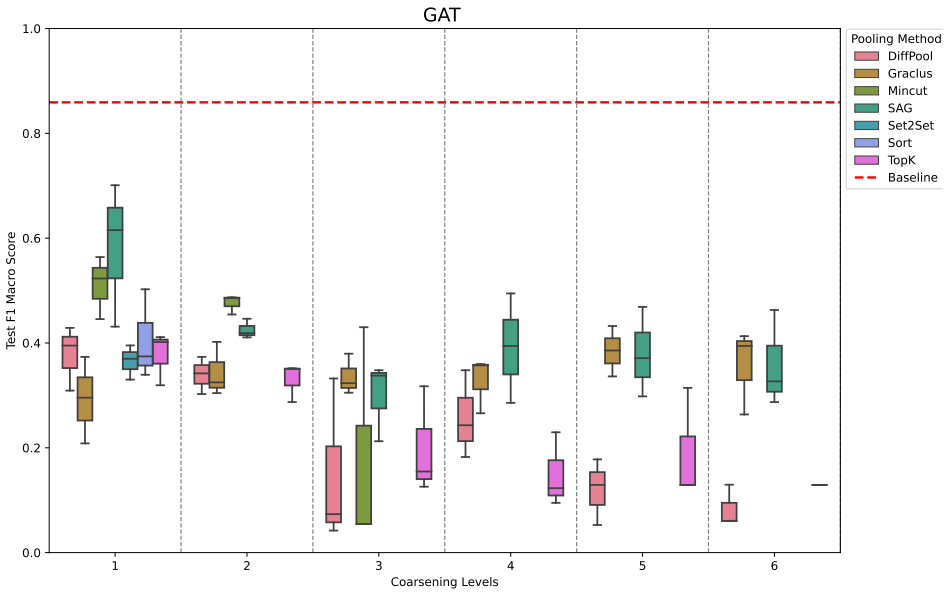
Figure 5.6: Boxplot of F1 Test Score with different convolutional layers



(a) Boxplot of Test F1 Macro Score for each pooling method with SAGE architecture, at each coarsening level



(b) Boxplot of Test F1 Macro Score for each pooling method with GCN architecture, at each coarsening level



(c) Boxplot of Test F1 Macro Score for each pooling method with GAT architecture, at each coarsening level

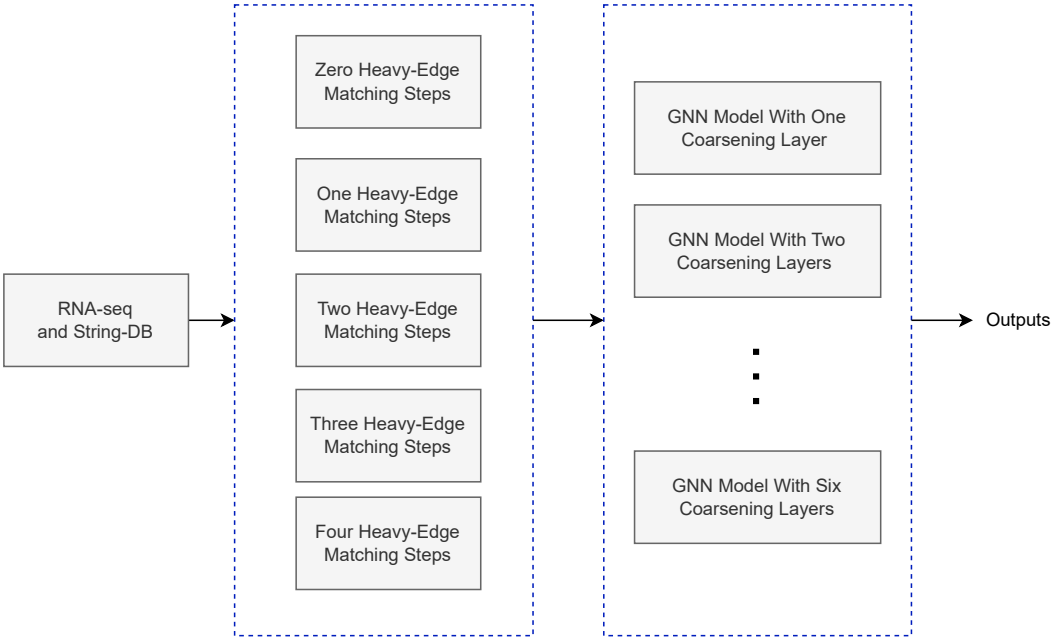
Source: The Authors

Table 5.1: Average Test F1 score (Standard Deviation)

Pooling Method	Coarsening Level / Convolution Layer	1	2	3	4	5	6
DiffPool	GAT	0.378 (0.062)	0.339 (0.036)	0.149 (0.159)	0.258 (0.084)	0.120 (0.063)	0.084 (0.040)
	GCN	0.316 (0.042)	0.273 (0.008)	0.246 (0.043)	0.140 (0.060)	0.147 (0.011)	0.126 (0.031)
	SAGE	0.480 (0.053)	0.472 (0.007)	0.442 (0.024)	0.450 (0.048)	0.352 (0.089)	0.339 (0.055)
Graclus	GAT	0.292 (0.083)	0.344 (0.052)	0.336 (0.039)	0.328 (0.053)	0.385 (0.048)	0.357 (0.081)
	GCN	0.273 (0.011)	0.316 (0.008)	0.266 (0.033)	0.225 (0.063)	0.283 (0.031)	0.236 (0.037)
	SAGE	0.409 (0.068)	0.504 (0.027)	0.500 (0.059)	0.476 (0.025)	0.526 (0.003)	0.532 (0.059)
Mincut	GAT	0.511 (0.060)	0.476 (0.018)	0.180 (0.217)	–	–	–
	GCN	0.517 (0.029)	0.511 (0.027)	0.479 (0.057)	0.469 (0.069)	0.414 (0.113)	0.439 (0.052)
	SAGE	0.448 (0.034)	0.518 (0.028)	0.458 (0.052)	0.488 (0.062)	0.445 (0.064)	0.320 (0.232)
SAG	GAT	0.582 (0.138)	0.425 (0.019)	0.299 (0.075)	0.392 (0.104)	0.379 (0.086)	0.359 (0.092)
	GCN	0.810 (0.043)	0.833 (0.018)	0.673 (0.278)	0.366 (0.101)	0.633 (0.129)	0.391 (0.120)
	SAGE	0.828 (0.039)	0.834 (0.034)	0.659 (0.292)	0.821 (0.009)	0.720 (0.067)	0.563 (0.233)
Set2Set	GAT	0.365 (0.033)	–	–	–	–	–
	GCN	0.276 (0.044)	0.263 (0.032)	0.274 (0.032)	0.253 (0.053)	0.270 (0.048)	0.200 (0.054)
	SAGE	0.474 (0.016)	0.492 (0.053)	0.496 (0.012)	0.320 (0.021)	0.366 (0.054)	0.334 (0.027)
Sort	GAT	0.405 (0.086)	–	–	–	–	–
	GCN	0.344 (0.262)	0.539 (0.171)	0.265 (0.033)	0.170 (0.014)	0.173 (0.012)	0.132 (0.063)
	SAGE	0.449 (0.193)	0.512 (0.139)	0.602 (0.173)	0.432 (0.159)	0.526 (0.160)	0.546 (0.199)
TopK	GAT	0.377 (0.050)	0.330 (0.037)	0.199 (0.103)	0.149 (0.071)	0.191 (0.107)	0.129 (0.000)
	GCN	0.342 (0.020)	0.383 (0.013)	0.177 (0.083)	0.129 (0.000)	0.146 (0.093)	0.129 (0.000)
	SAGE	0.362 (0.022)	0.333 (0.047)	0.328 (0.028)	0.293 (0.030)	0.179 (0.133)	0.156 (0.111)

We also wondered whether the previous preprocessing with heavy-edge matching affected the results. We then tested the pooling methods that can use a big input graph without the preprocessing step, with one more layer of this preprocessing step, resulting in graphs with 1767 nodes and two more layers, resulting in graphs with 884 nodes. The methods that were compared were SAG, TopK, Sort, and Set2Set, all four with the SAGE architecture that showed a better performance in the previous experiment. Figure 5.7 gives an outline of this second part of the experiment.

Figure 5.7: Experiment performed on the influence of the preprocessing step on the results

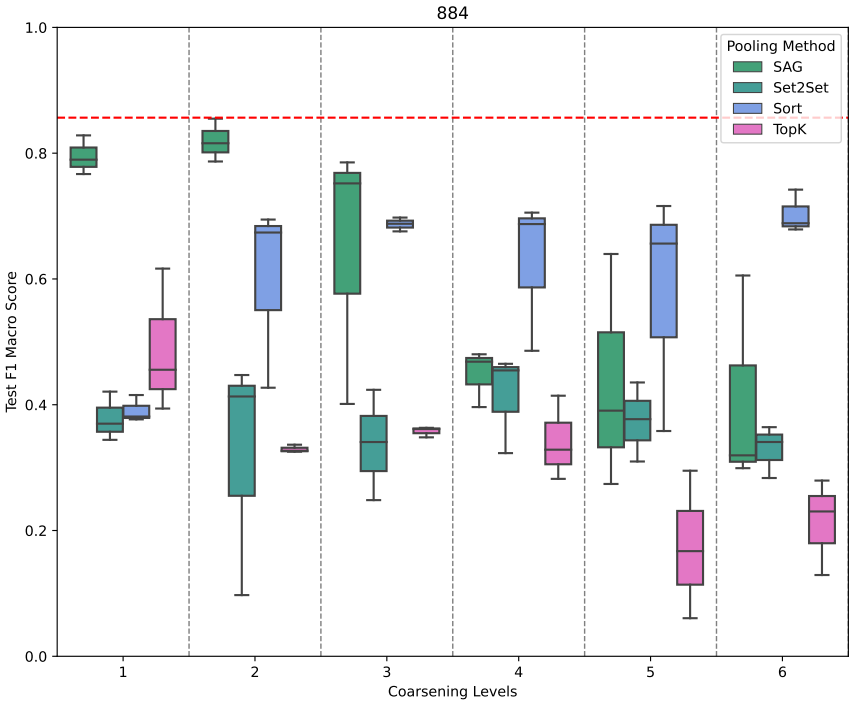


Source: The Authors

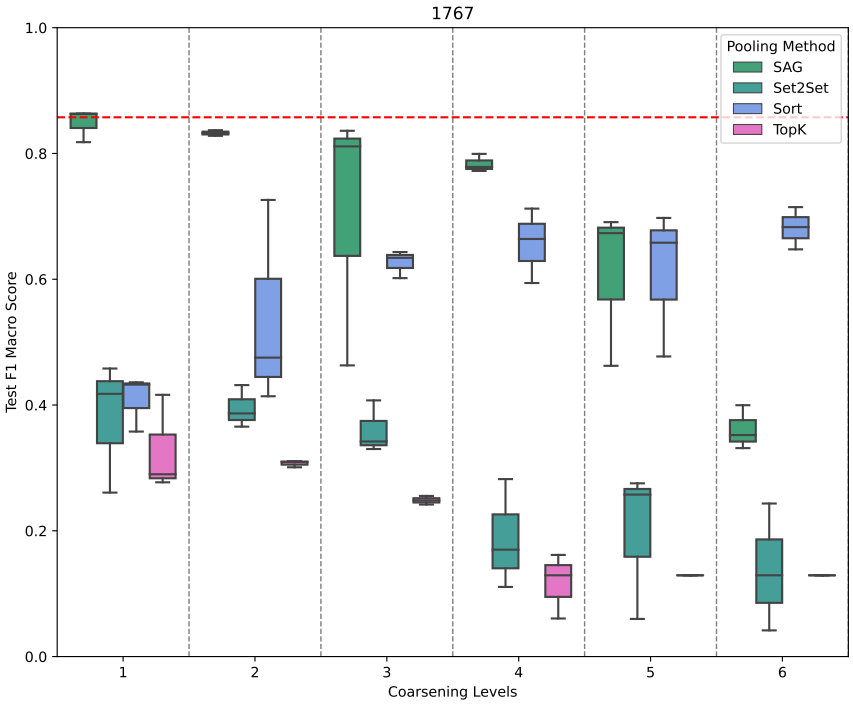
Figure 5.8 provides an overview of the experimental outcomes. We again saw that, regardless of whether or not a preprocessing step is applied, the SAG method outperforms the other methods by a considerable margin. The average F1 scores for SAG at level on the test set are as follows: 0.833 for graphs with 14,133 nodes, 0.822 for 7,067 nodes, 0.834 for 3,534 nodes, 0.8324 for 1,767 nodes, and 0.819 for 884 nodes, while the second best method being Sort 0.598, 0.538, 0.511, 0.661, 0.712, respectively at level 2. Set2Set, the other global pooling method together with Sort in this experiment, showed a similar trend, whereas the performance dropped in smaller graph sizes, going from 0.423 at 14133 nodes to 0.319 at 884 nodes, but yet an inferior performance overall. TopK, although the same taxonomy as SAG, again did not stand out at any graph size, being the worst method for almost all graph sizes, except at 884 nodes, where it had an intermediary performance with 0.488 at level one.

In summary, the result of this experiment stands out as two characteristics of the pooling methods. First, hierarchical methods tend to decrease the performance as we increase the number of coarsening layers. This could be due to the loss of information by coarsening the graph multiple times, while global pooling does not appear to have this tendency, depending on the intrinsics operations of the individual methods. The second refers to hierarchical methods that could benefit from a prior preprocessing, which can be seen as a fixed pooling operation, to increase the performance and also reduce the complexity of the problem, demanding less computational and time resources.

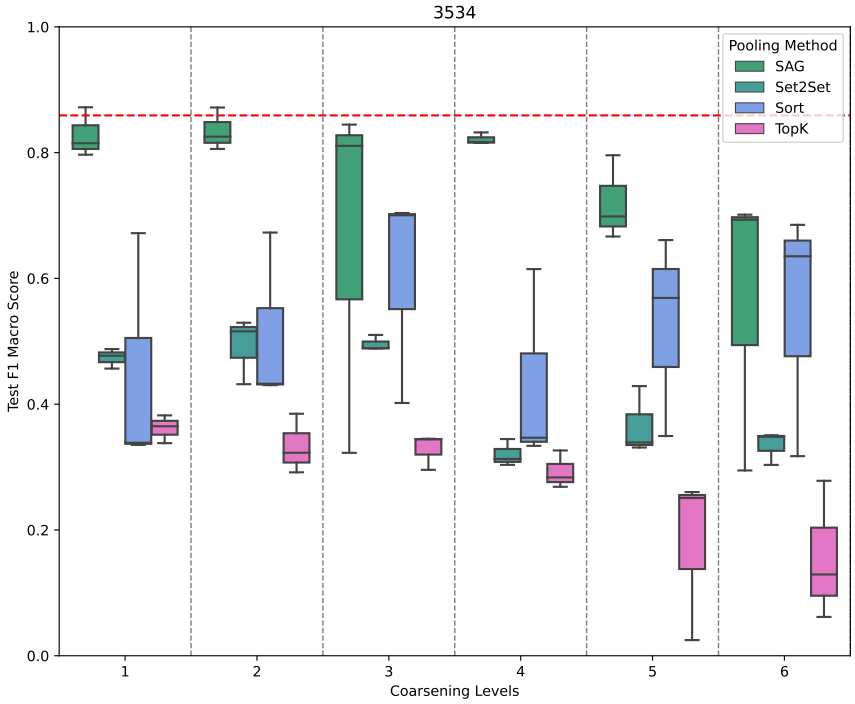
Figure 5.8: Boxplot of F1 Test Score with different graph sizes



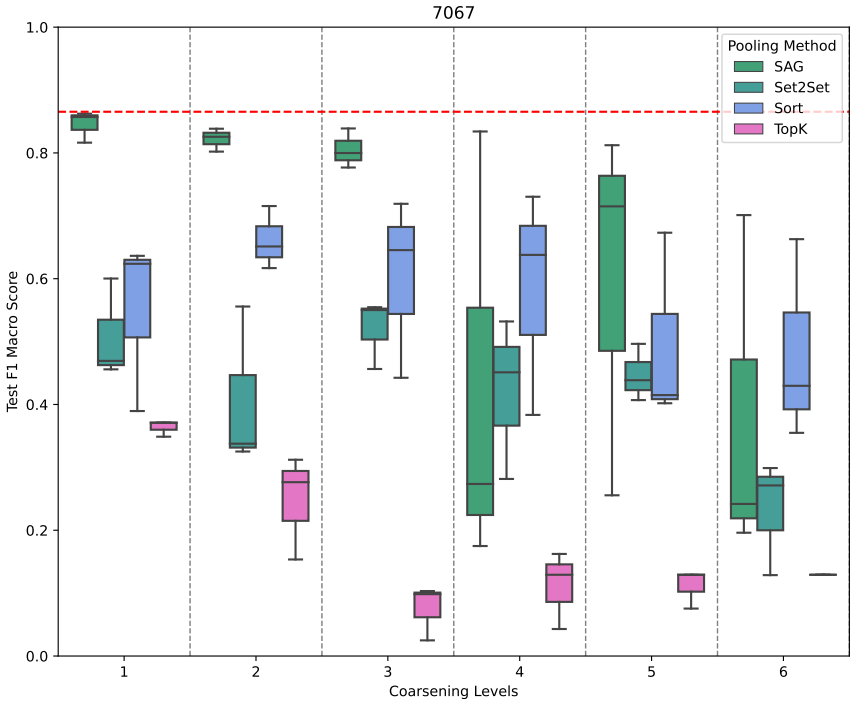
(a) Boxplot of Test F1 Macro Score for each pooling method with 884 nodes at each coarsening level



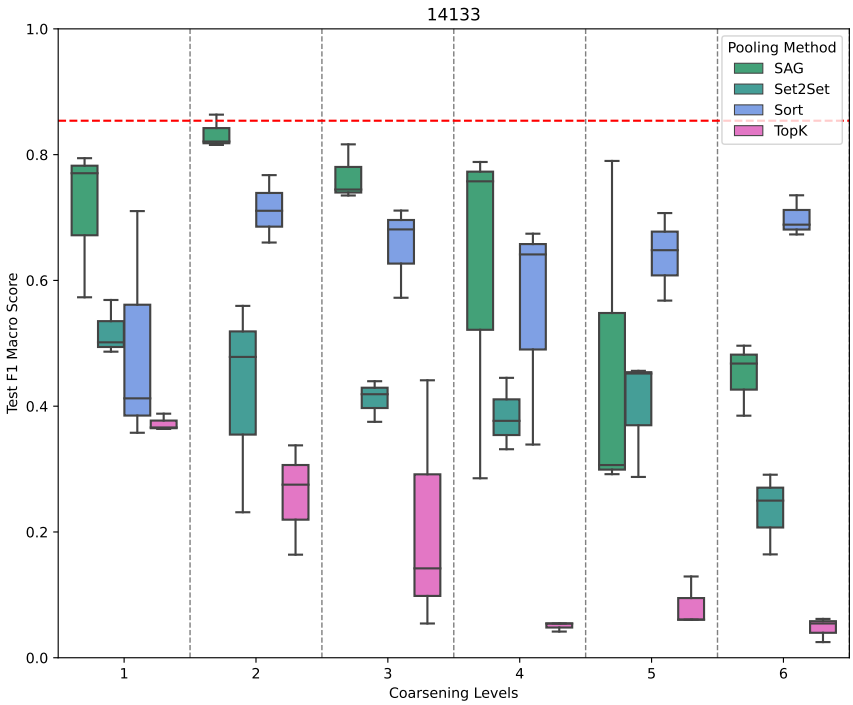
(b) Boxplot of Test F1 Macro Score for each pooling method with 1767 nodes at each coarsening level



(c) Boxplot of Test F1 Macro Score for each pooling method with 3534 nodes at each coarsening level



(d) Boxplot of Test F1 Macro Score for each pooling method with 7067 nodes at each coarsening level



(e) Boxplot of Test F1 Macro Score for each pooling method with 14133 nodes at each coarsening level

Source: The Authors

5.3 Experiment Three - Interpretability

Understanding the output of models in machine learning, particularly in the context of genomic data, is essential for the identification and validation of biomarkers. Model interpretability helps link genomic patterns to biomarkers, enabling informed decisions by researchers and clinicians. Hence, we analyzed the model's saliencies to understand which genes were more important for each target class. We decided to analyze the model architecture with SAGPool, SAGE at level two with graphs of 3534 nodes, as it was the best performer for this level and has a good overall balance of prior processing, nodes selected at the final output, and performance. Figure 5.9 gives an overview of experiment three.

Figure 5.9: Model interpretability using saliency maps experiment

Source: The Authors

Figure 5.10 shows the top 15 features based on our dataset's saliency score for each target label. The results demonstrate that the model's decision is very complex, as the medians of the feature saliencies are very close to each other, which makes the analysis of the outputs much harder. Figure 5.11 shows the t-SNE visualization of the embeddings of the saliencies, in which we can observe an intersection of the subtypes Her2, LumA, and LumB.

An explanation for that is that the model might rely on a complex interaction of features rather than individual features, meaning that no single feature stands out as particularly influential, but rather, the interaction of features is what might drive the model's decisions. Another plausible explanation is that we found that SAGPool was giving the maximum score for a good amount of features, which suggests that the model perceives a wide array of features as significantly relevant and also distributes the importance across them, making the saliency of each feature very close to each other.

We also noticed the complexity of the analysis when researching the top 2 features of each target label in the literature to check if those were previously studied as biomarkers in breast cancer. Remembering Section 4.2.1, the data for the model analyzed was preprocessed two times with a greedy cluster algorithm, thus one node in the processed graph represents a supernode of four nodes at the original graph. Table 5.2 displays the genes selected and their presence in the literature.

The results show that 15 out of 40 genes were previously associated with breast cancer in other studies. Apart from index 1692 in Basal, at least one gene in the other supernodes was researched and in some way related to breast cancer carcinoma. For Basal, we found that index 1116 had three genes researched: SLC25A32 was found to reduce median survival of breast cancer patients by 42 months when highly amplified (SANTORO et al., 2020), ISG20L2 was cited as a novel diagnostic and prognostic biomarker for breast cancer in Yin et al. (2021), and FLAD1 was found overexpressed in breast cancer (JIA et al., 2019). In Her2, the up-regulation and down-regulation of the ORMDL3 gene were correlated with bad prognosis (ZHANG et al., 2019), while ADAM33 was identified as a novel predictive biomarker in BRCA (MANICA et al., 2017). Additionally, RGS19 was discovered as an upregulation factor of the Nm23-H1 gene (LI et al., 2017),

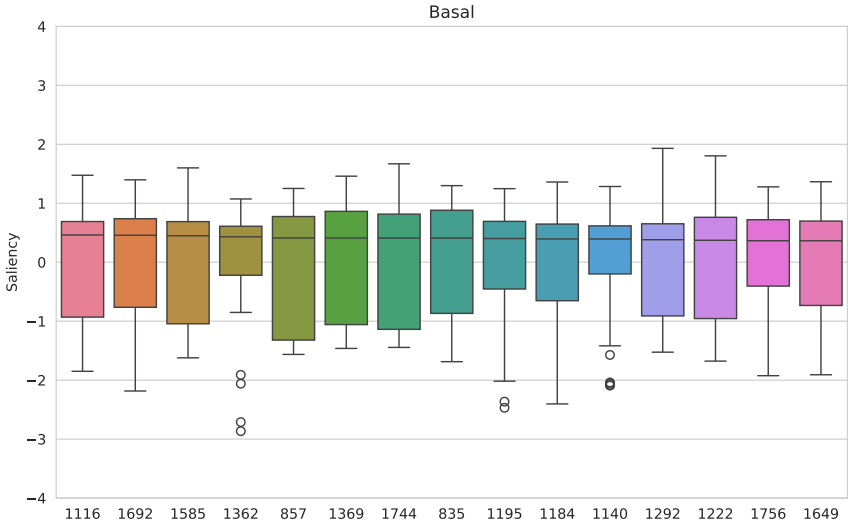
which was studied as a suppressor of breast cancer metastasis (KIM; LEE, 2021). The most important genes that we found correspondence for LumA was only GPS2, whose lower expression correlates with poorer survival (CHAN et al., 2021), while for LumB the gene FXYD2 was seen downregulated (ZHANG et al., 2022), TRPM6 was also downregulated in breast cancer (QIN et al., 2020) and DNASE1L3 was studied as a possible prognostic marker in breast cancer and other types of cancer (DENG et al., 2021). For the control target label, Normal, the genes previously researched were GRIN3B, NLRP12, PYCARD, and NAIP. GRIN3B was studied along GRIN2B; both were noticed as being highly expressed, but the former did not show any correlation with cancer patient survival, while the latter did (GALLO et al., 2022). NAIP was shown overexpressed in patients with unfavorable prognostic factors(CHOI et al., 2007), while the inhibited expression of NLRP12 affected the triple-negative breast cancer (KUANG et al., 2023). PYCARD was also conferred poor outcomes in multiple breast cancer data sets along with other genes (DAIRKEE et al., 2009).

Table 5.2: Selected genes and their previous literature correspondence in BRCA studies

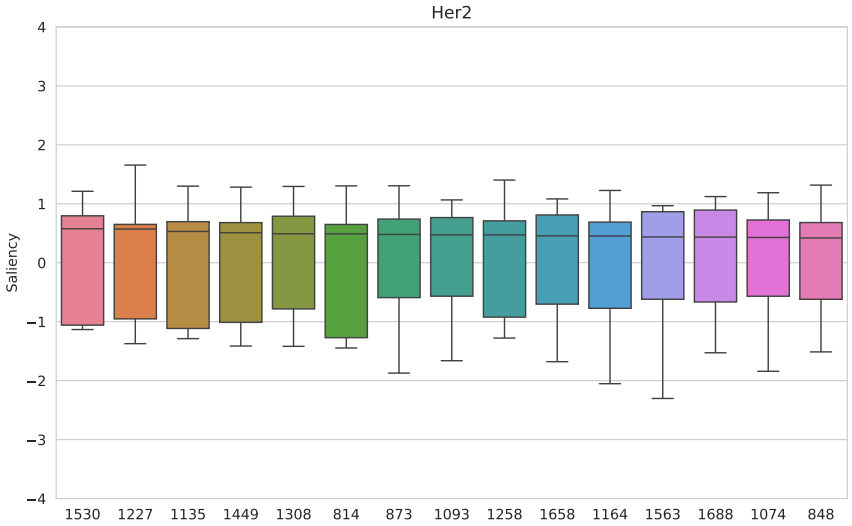
Label	Index	STRING Protein Id	Gene	Literature
Basal	1116	ENSP00000399979	SLC52A1	
		ENSP00000297578	SLC25A32	✓
		ENSP00000323424	ISG20L2	✓
		ENSP00000292180	FLAD1	✓
Basal	1692	ENSP00000221954	CEACAM4	
		ENSP00000245620	LILRB3	
		ENSP00000043402	RTN4R	
		ENSP00000247271	OMG	
Her2	1530	ENSP00000479816	DENND1B	
		ENSP00000287008	PCDH1	
		ENSP00000377724	ORMDL3	✓
		ENSP00000348912	ADAM33	✓
Her2	1227	ENSP00000319254	GIPC3	✓
		ENSP00000365637	KIAA1217	
		ENSP00000378483	RGS19	✓
		ENSP00000359795	GIPC2	
LumA	1061	ENSP00000360398	SLC25A27	
		ENSP00000406546	SLC23A3	
		ENSP00000260505	TTLL7	
		ENSP00000359892	SLC44A5	
LumA	1202	ENSP00000301039	PROCA1	
		ENSP00000370104	GPS2	✓
		ENSP00000272430	RTKN	
		ENSP00000221459	LIN7B	
LumB	871	ENSP00000326070	SLC41A3	
		ENSP00000292079	FXYD2	✓
		ENSP00000258538	SLC41A2	
		ENSP00000354006	TRPM6	✓
LumB	837	ENSP00000378053	DNASE1L3	✓
		ENSP00000249014	CDC42EP1	
		ENSP00000334665	FSCN2	
		ENSP00000297157	RP9	
Normal	1408	ENSP00000234389	GRIN3B	✓
		ENSP00000264833	OLFM2	
		ENSP00000272133	CNIH3	
		ENSP00000355155	GRIN3A	
Normal	1564	ENSP00000375653	NLRP12	✓
		ENSP00000247470	PYCARD	✓
		ENSP00000219596	MEFV	
		ENSP00000428657	NAIP	✓

Source: The Authors

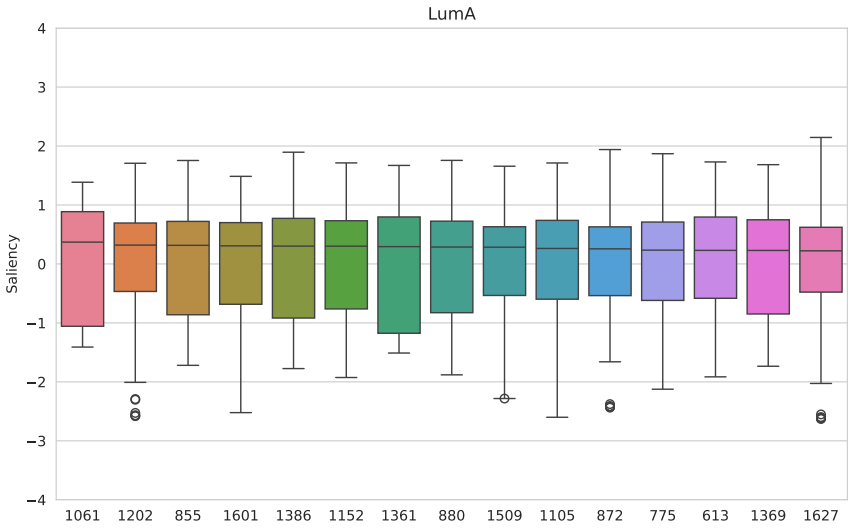
Figure 5.10: Normalized saliencies over samples for each gene



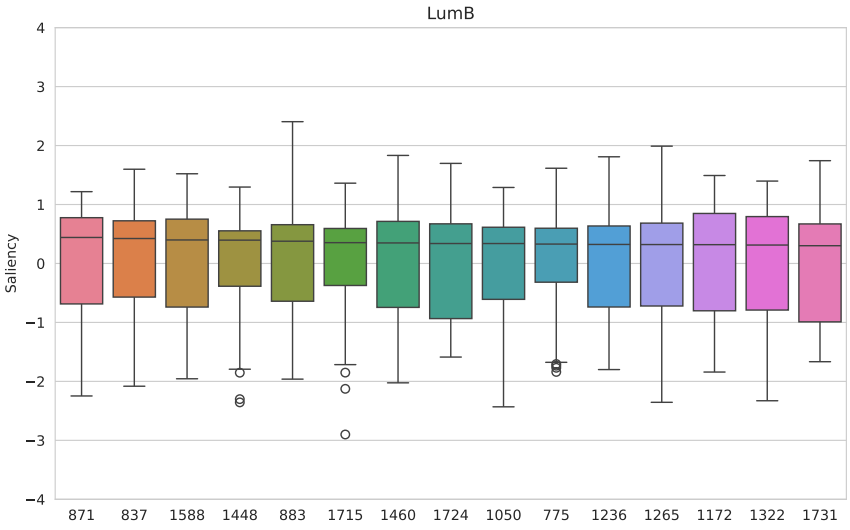
(a) Boxplot of saliencies for Basal subtype



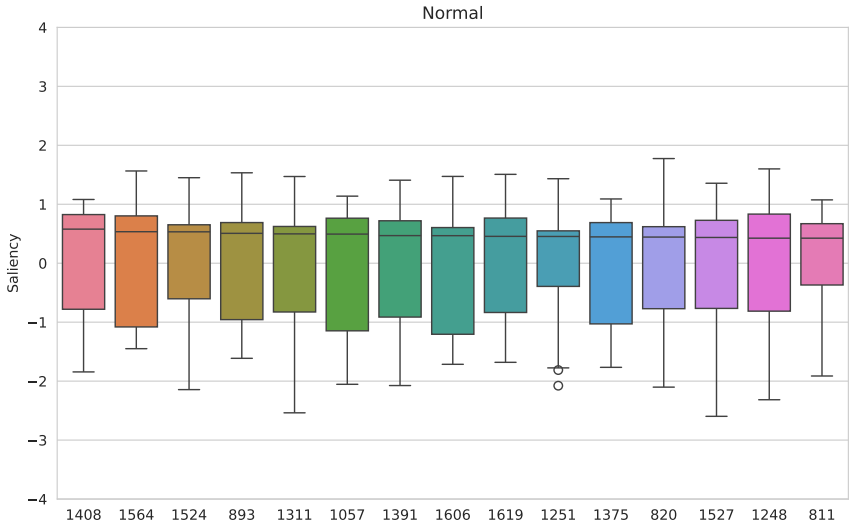
(b) Boxplot of saliencies for Her2 subtype



(c) Boxplot of saliencies for LumA subtype



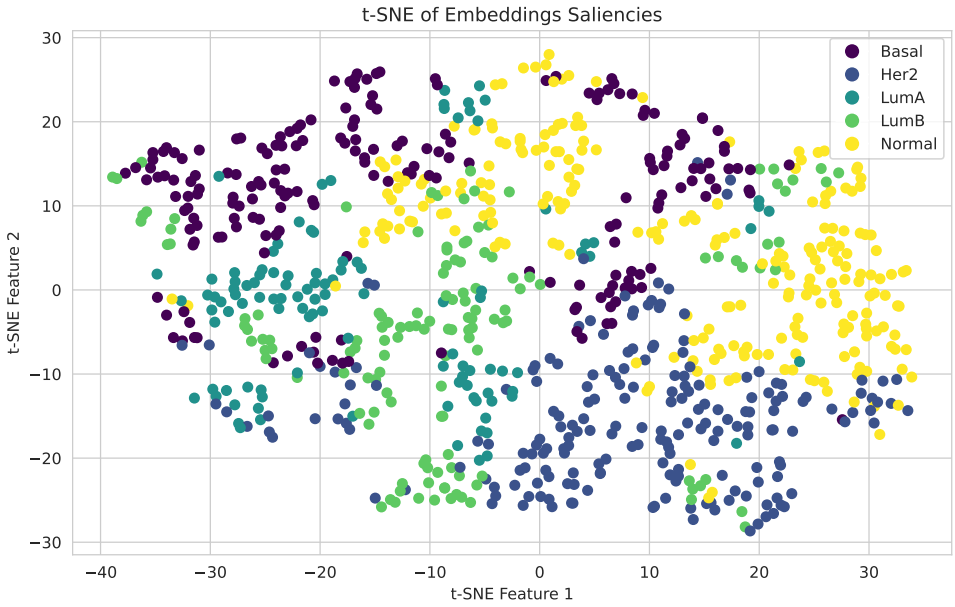
(d) Boxplot of saliencies for LumB subtype



(e) Boxplot of saliencies for Normal subtype

Source: The Authors

Figure 5.11: t-SNE visualization of the embeddings saliencies produced by the model



Source: The Authors

6 CONCLUSION

The integration of genomics and Graph Neural Networks (GNNs) has emerged as a significant development in the realm of computational biology, revolutionizing the way we approach and interpret complex biological data. In this work, we have explored how the choice of the GNN architecture affected the model efficacy in a multi-label classification scenario, with a particular emphasis on the convolutional layers and pooling methods for RNA-Seq data in breast cancer. Additionally, we present a comprehensive evaluation of the performance and resource utilization of various pooling methods. Finally, we enhance our study with an explicability analysis of the best GNN architecture, utilizing saliency maps.

In our experiment regarding the performance and resource utilization for various pooling methods, we have shown that methods that require a dense adjacency matrix as input instead of a sparse matrix have poor scalability and consume much more memory when the number of nodes in the graph is increased. When we analyzed the time taken to train the model, we could not find a relationship between the taxonomy proposed by (GRATTAROLA et al., 2022) and the results, which means that this factor is mainly affected by the internal characteristics of the chosen method.

In our second experiment, we examined various GNN (Graph Neural Network) architectures, leading to notable findings. Firstly, we have shown that SAGPool (LEE; LEE; KANG, 2019) performed significantly better than the other methods in terms of predictive power. Second, we showed that GraphSAGE (HAMILTON; YING; LESKOVEC, 2018) performed better for the majority of pooling methods, while the GCN (KIPF; WELLING, 2017) had a decent performance when compared to the GCN but seemed to be more affected by the increase in numbers of the coarsening layers than GraphSAGE and GAT (VELIčKOVIć et al., 2018), the latter, however, performed poorly for all pooling methods. Additionally, we explored the impact of preprocessing on model performance. Our results suggest that for certain methods, such as SAG and Sort (ZHANG et al., 2018), the balance between predictive performance and computational resource requirements could be advantageous.

The third experiment was designed to explain the model outputs using saliency maps. However, throughout the experiment, we found that the importance of the features was very close to each other, which made the task of analyzing the possible biomarkers' genes more difficult. Nevertheless, we researched the top two most important genes and

found some previous studies that related them to breast cancer, such as ADAM33 and DNASE1L3.

Therefore, our work has demonstrated the significant influence that the choice of Graph Neural Network (GNN) architecture can have on model performance, and we endorse researchers to meticulously test different architectures to advance understanding and develop robust models. However, the study also opens up several opportunities for future research. One area of interest is understanding how increased computational resources might affect model performance, especially with the potential application of dense matrix methods across the entire graph. Additionally, the possibility of adapting pooling methods that require dense matrix formats to sparse formats, which could address a potential constraint against these methods, would be another opportunity for future studies. Finally, exploring how different explainer methods impact the interpretability of pooling methods represents another promising research direction.

REFERENCES

AHMAD, F. B. et al. Provisional mortality data - united states, 2022. **MMWR Morb Mortal Wkly Rep**, Centers for Disease Control and Prevention, v. 72, n. 18, p. 488–492, May 2023.

ALBARADEI, S. et al. Machine learning and deep learning methods that use omics data for metastasis prediction. **Computational and Structural Biotechnology Journal**, v. 19, 09 2021.

BIANCHI, F. M.; GRATTAROLA, D.; ALIPPI, C. **Spectral Clustering with Graph Neural Networks for Graph Pooling**. 2020.

BROWN, W. A. C.; CLANCY, S. Translation: Dna to mrna to protein. In: . [s.n.], 2008. Available from Internet: <https://api.semanticscholar.org/CorpusID:89529269>.

CHAN, S. et al. Loss of g-protein pathway suppressor 2 promotes tumor growth through activation of akt signaling. **Frontiers in Cell and Developmental Biology**, v. 8, 01 2021.

CHEREDA, H. et al. **Explaining decisions of graph convolutional neural networks: patient-specific molecular subnetworks responsible for metastasis prediction in breast cancer**. 2021. 1-16 p. Available from Internet: <https://www.semanticscholar.org/paper/df73fca81d247a64075d893f256ec8dbf084078f>.

CHOI, J. et al. Neuronal apoptosis inhibitory protein is overexpressed in patients with unfavorable prognostic factors in breast cancer. **Journal of Korean Medical Science**, v. 22, p. S17 – S23, 2007. Available from Internet: <https://api.semanticscholar.org/CorpusID:59869>.

DAIRKEE, S. et al. Immutable functional attributes of histologic grade revealed by context-independent gene expression in primary breast cancer cells. **Cancer research**, v. 69, p. 7826–34, 10 2009.

DENG, Z. et al. Dnase1l3 as a prognostic biomarker associated with immune cell infiltration in cancer. **OncoTargets and Therapy**, Volume 14, p. 2003–2017, 03 2021.

DHILLON, I. S.; GUAN, Y.; KULIS, B. Weighted graph cuts without eigenvectors a multilevel approach. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 29, n. 11, p. 1944–1957, 2007.

ERDöS, P.; RéNYI, A. On the evolution of random graphs. In: ____. **The Structure and Dynamics of Networks**. Princeton: Princeton University Press, 2006. p. 38–82. ISBN 9781400841356. Available from Internet: <https://doi.org/10.1515/9781400841356.38>.

GALLO, S. et al. Met oncogene controls invasive growth by coupling with nmda receptor. **Cancers**, MDPI, v. 14, n. 18, p. 4408, 2022.

GAO, H.; JI, S. **Graph U-Nets**. 2019.

GILMER, J. et al. Neural message passing for quantum chemistry. In: PRECUP, D.; TEH, Y. W. (Ed.). **Proceedings of the 34th International Conference on Machine Learning**. PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 1263–1272. Available from Internet: <https://proceedings.mlr.press/v70/gilmer17a.html>.

GOLDMAN, M. J. et al. Visualizing and interpreting cancer genomics data via the xena platform. **Nature Biotechnology**, v. 38, p. 675 – 678, 2020. Available from Internet: <https://api.semanticscholar.org/CorpusID:218836480>.

GONZALEZ, M. W.; KANN, M. G. Chapter 4: Protein interactions and disease. **PLOS Computational Biology**, v. 8, n. 12, p. e1002819, 2012. Https://journals.plos.org/ploscompbiol/article/file?id=10.1371/journal.pcbi.1002819type=printable. Available from Internet: <https://app.dimensions.ai/details/publication/pub.1001404740>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GRATTAROLA, D. et al. Understanding pooling in graph neural networks. **IEEE Transactions on Neural Networks and Learning Systems**, Institute of Electrical and Electronics Engineers (IEEE), p. 1–11, 2022. Available from Internet: <https://doi.org/10.1109%2Ftnnls.2022.3190922>.

HAMILTON, W. **Graph Representation Learning**. Morgan & Claypool Publishers, 2020. (Synthesis lectures on artificial intelligence and machine learning). ISBN 9781681739632. Available from Internet: <https://books.google.com.br/books?id=V6HnzQEACAAJ>.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. **Inductive Representation Learning on Large Graphs**. 2018.

HAYAKAWA, J. et al. Pathway importance by graph convolutional network and shapley additive explanations in gene expression phenotype of diffuse large b-cell lymphoma. **PLoS ONE**, v. 17, 2022. Available from Internet: <https://api.semanticscholar.org/CorpusID:249988764>.

JIA, X. et al. Flad1 is overexpressed in breast cancer and is a potential predictor of prognosis and treatment. **International Journal of Clinical and Experimental Medicine**, E-Century Publishing Corporation, v. 12, n. 4, p. 3138–3152, 2019. ISSN: 1940-5901/IJCEM0081640. Available from Internet: <https://e-century.us/files/ijcem/12/4/ijcem0081640.pdf>.

KIM, B.; LEE, K.-J. Activation of nm23-h1 to suppress breast cancer metastasis via redox regulation. **Experimental Molecular Medicine**, v. 53, p. 1–12, 03 2021.

KIPF, T. N.; WELLING, M. **Semi-Supervised Classification with Graph Convolutional Networks**. 2017.

KUANG, W. et al. Inhibited expression of nlrp12 promotes the development of triple-negative breast cancer by activating the nf-b pathway. **Cell Biochemistry and Biophysics**, Springer, v. 81, n. 4, p. 727–735, 2023.

LEE, J.; LEE, I.; KANG, J. **Self-Attention Graph Pooling**. 2019.

LI, B.; NABAVI, S. **A Multimodal Graph Neural Network Framework for Cancer Molecular Subtype Classification**. 2023.

LI, B.; WANG, T.; NABAVI, S. Cancer molecular subtype classification by graph convolutional networks on multi-omics data. **Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics**, 2021. Available from Internet: <https://api.semanticscholar.org/CorpusID:236519361>.

LI, X. et al. Mogcn: A multi-omics integration method based on graph convolutional network for cancer subtype analysis. **Frontiers in genetics**, v. 13, p. 806842, 02 2022.

LI, Y. et al. Rgs19 upregulates nm23-h1/2 metastasis suppressors by transcriptional activation via the camp/pka/creb pathway. **Oncotarget**, v. 8, 07 2017.

LIU, C. et al. Graph pooling for graph neural networks: Progress, challenges, and opportunities. **arXiv preprint arXiv:2204.07321**, 2022.

LOSHCHILOV, I.; HUTTER, F. **SGDR: Stochastic Gradient Descent with Warm Restarts**. 2017.

LOSHCHILOV, I.; HUTTER, F. **Decoupled Weight Decay Regularization**. 2019.

LOWE, R. et al. Transcriptomics technologies. **PLOS Computational Biology**, v. 13, p. e1005457, 05 2017.

LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: GUYON, I. et al. (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, Inc., 2017. p. 4765–4774. Available from Internet: <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.

MANICA, G. et al. Down regulation of adam33 as a predictive biomarker of aggressive breast cancer. **Scientific Reports**, v. 7, p. 44414, 03 2017.

National Human Genome Research Institute. **A Brief Guide to Genomics**. 2023. <https://www.genome.gov/genetics-glossary/Genome>. Accessed: 2023-12-10.

QIN, F. et al. Evaluation of the trpm protein family as potential biomarkers for various types of human cancer using public database analyses. **Experimental and Therapeutic Medicine**, v. 20, 05 2020.

RAMIREZ, R. et al. Classification of cancer types using graph convolutional neural networks. In: **Frontiers of Physics**. [s.n.], 2020. Available from Internet: <https://api.semanticscholar.org/CorpusID:219709593>.

RAMIREZ, R. et al. Prediction and interpretation of cancer survival using graph convolution neural networks. **Methods**, v. 192, p. 120–130, 2021. ISSN 1046-2023. Deep networks and network representation in bioinformatics. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1046202321000165>.

SANTORO, V. et al. Slc25a32 sustains cancer cell proliferation by regulating flavin adenine nucleotide (fad) metabolism. **Oncotarget**, v. 11, 02 2020.

SIEGEL, R. L. et al. Cancer statistics, 2023. **CA Cancer J Clin**, v. 73, n. 1, p. 17–48, Jan 2023.

SIMONYAN, K.; VEDALDI, A.; ZISSERMAN, A. **Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps**. 2014.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 56, p. 1929–1958, 2014. Available from Internet: <http://jmlr.org/papers/v15/srivastava14a.html>.

VELIčKOVIć, P. et al. **Graph Attention Networks**. 2018.

VINYALS, O.; BENGIO, S.; KUDLUR, M. **Order Matters: Sequence to sequence for sets**. 2016.

WANG, T.; BAI, J.; NABAVI, S. Single-cell classification using graph convolutional networks. **BMC Bioinformatics**, BioMed Central, v. 22, n. 1, p. 364, 2021. Available from Internet: <https://doi.org/10.1186/s12859-021-04278-2>.

WU, C. et al. Neural networks for molecular sequence classification. **Mathematics and Computers in Simulation**, v. 40, n. 1, p. 23–33, 1995. ISSN 0378-4754. Available from Internet: <https://www.sciencedirect.com/science/article/pii/0378475495000164>.

YIN, J. et al. Cenpl, isg20l2, lsm4, mrpl3 are four novel hub genes and may serve as diagnostic and prognostic markers in breast cancer. **Scientific reports**, v. 11, n. 1, p. 15610, August 2021. ISSN 2045-2322. Available from Internet: <https://europepmc.org/articles/PMC8328991>.

YIN, Q. et al. scGraph: a graph neural network-based approach to automatically identify cell types. **Bioinformatics**, v. 38, n. 11, p. 2996–3003, 04 2022. ISSN 1367-4803. Available from Internet: <https://doi.org/10.1093/bioinformatics/btac199>.

YING, R. et al. **Hierarchical Graph Representation Learning with Differentiable Pooling**. 2019.

ZHANG, M. et al. An end-to-end deep learning architecture for graph classification. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 32, n. 1, Apr. 2018. Available from Internet: <https://ojs.aaai.org/index.php/AAAI/article/view/11782>.

ZHANG, M. et al. Identifying genes with tri-modal association with survival and tumor grade in cancer patients. **BMC Bioinformatics**, v. 20, 01 2019.

ZHANG, Z. et al. Downregulation of fxyd2 is associated with poor prognosis and increased regulatory t cell infiltration in clear cell renal cell carcinoma. **Journal of Immunology Research**, v. 2022, p. 1–19, 10 2022.