11-21-2019

Attributed Graph Classification via Deep Graph Convolutional Neural Networks

Susha Pozhampallan Suresh
University of Windsor

Attributed Graph Classification via Deep Graph Convolutional Neural Networks

By

Susha Pozhampallan Suresh

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Attributed Graph Classification via Deep Graph Convolutional Neural Networks

by

Susha Pozhampallan Suresh

APPROVED BY:

E. Abdel-Raheem
Department of Electrical and Computer Engineering

J. Lu
School of Computer Science

D. Wu, Advisor
School of Computer Science

L. Rueda, Co-Advisor
School of Computer Science

November 20, 2019

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon any copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

From social networks to biological networks, graphs are a natural way to represent a diverse set of real-world data. This research presents attributed graph convolutional neural network with a pooling layer (AGCP for short), a novel end-to-end deep neural network model which captures the higher-order latent attributes of weighted, labeled, undirected, attributed graphs of arbitrary size. The architecture of AGCP is an efficient variant of convolutional neural network (CNN) and has a linear filter function that convolves over the fixed topological structure of a graph to learn local and global attributes of the graph. Convolution is followed by a pooling layer that coarsens the graph while preserving the global structure of the original input graph using information gain. On the other hand, advances in high throughput technologies for next-generation sequencing have enabled machine learning research to acquire and extract knowledge from biological networks. We apply AGCP on three bioinformatics networks, ENZYMES, D&D, and GINA a graph dataset of gene interaction networks with genomic mutation attributes as the attributes of the vertices. In several experiments on these datasets, we demonstrate that AGCP yields better results in terms of classification accuracy relative to the previously proposed models by a considerable margin.

Keywords: Graph Convolutional Neural Networks, Deep Neural Network, Gene Interaction Network, Gleason Score prediction.

DEDICATION

I want to dedicate this thesis to my parents Mr. Suresh Pozhampallan Gopalan and Mrs. Usha Suresh, and my brother Mr. Sandhesh Pozhampallan Suresh.

ACKNOWLEDGMENTS

Special thanks to my supervisor Dr. Dan Wu for his patience, support, and professional guidance during my graduate studies. I want to take this opportunity to thank Dr. Luis Rueda for his words of inspiration and his immense support throughout my graduate studies. Without Dr. Rueda's grace, this thesis will not have become a reality. I want to thank my internal reader, Dr. Jiango Lu, and my external reader, Dr. Esam Abdel-Raheem, for their time, effort, and valuable opinions. I want to thank Dr. Ngom, who taught me machine learning and inspired me to indulge in the world of machine intelligence. I am greatly indebted to my parents for their words of inspiration throughout my education. I want to thank Mr. Jonathan Ketel for reviewing my thesis document and continually motivating me to follow my dreams. I want to thank my close friends Ms. Chandini Ravindernath and Mr. Jerry George Thomas for their constant support. I am thankful to all my friends, professors, colleagues, and the staff members of the School of Computer Science at the University of Windsor for their kind support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Pairwise connections among entities play a crucial role in a wide variety of computational applications. Many examples of real-life data have entities that share a relationship associated with a weight. In most of the cases, these entities have attributes associated with them. A graph or network represents these relationships efficiently. Examples of real-life data described in the form of a network include social networks, biological networks, chemical networks, citation networks, and research networks, among others. The analysis of data represented in the form of graphs is recent and is gaining a lot of traction due to the composite representation of graph structure and associated attributes (Gross & Yellen 2004). It is thus essential to focus on applying highly powerful machine learning algorithms to understand the complex structure of the graphs. Deep learning, which is a subfield of machine learning, has revolutionized fields such as video processing, speech recognition, and natural language understanding (Deng & Yu 2014). The complexity of graph data has imposed significant challenges on existing deep learning algorithms. Recently, many studies have emerged on extending deep learning approaches for graph data (Wu et al. 2019). We consider the problem of classifying graphs in this thesis, framed as a graph-based supervised classification. This research presents attributed graph convolutional neural network with a pooling layer (AGCP for short), a novel end-to-end deep neural network model which takes as input a weighted, attributed graph and predicts the class label of that graph.

In this chapter, we define graphs, weighted graphs, and attributed graphs followed by descriptions and examples of graph data generated by real-time industrial appli-

cations. Further, we describe different ways in which we can analyze the graph data using machine learning. We then describe the objective of our research, followed by problem definition. We then introduce the graph convolutional network (GCN) and formulate the research objective. The later sections of this chapter present the thesis motivation, thesis statement, followed by thesis contributions and thesis organization.

1.1 Graphs

Graphs are a universal language for describing a set of complex systems (Zhang et al. 2018). There are complex systems all around us; society is a collection of over seven billion individuals, communication systems link electronic devices, information and knowledge are organized and linked, whereas the interaction among thousands of genes and proteins regulate life, human thoughts are hidden in the connections between billions of neurons in our brain (Gross & Yellen 2004). All of these complex systems have a graph structure. The rapidly decreasing costs of high throughput sequencing and mass spectrometry, development of massively parallel technologies, and new sensor technologies have enabled the generation of data that describe these complex systems on multiple dimensions. Understanding and modeling these complex systems will have a massive impact on the betterment of society (Gross & Yellen 2004). Our research focuses on analyzing these complex graph-structured systems. We work on weighted, attributed, undirected graphs without self-loops.

A graph contains a set of vertices and a collection of edges that each connect a pair of vertices. Figure 1 shows an example of a graph with four vertices and four edges connecting the vertices, where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_{12}, e_{24}, e_{34}, e_{13}\}$.

Definition 1 (Graph) *A graph is defined as G= (V, E), where V is the finite set of vertices and E is the finite set of edges (Zhang et al. 2018). Also, $|V| = n$, where n is the number of vertices. $|E| = m$, where m is the number of edges.*

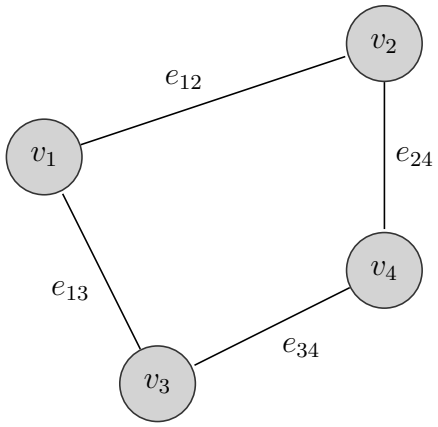


Figure 1: Example of a graph with four vertices and four edges connecting the vertices.

Definition 2 (Adjacency matrix) *The adjacency matrix of graph G is defined as $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ (Li, Han & Wu 2018), and represent the relationship between edges connecting the n vertices of a finite graph (Zhang et al. 2018).*

In this thesis, we consider a non-negative, symmetric adjacency matrix. The elements in the adjacency matrix are 1 or 0 in position according to whether two vertices are adjacent or not. In a graph $G = (V, E)$, let $v_i, v_j \in V$ denote two distinct vertices, and $e_{ij} \in E$ denote an edge between the vertices v_i and v_j. Each element a_{ij} of the adjacency matrix is denoted by a variable with two subscripts i and j. For example, a_{ij}, represents element at the i^{th} row and j^{th} column of the matrix. We define $a_{ij} = 1$ if $e_{ij} \in E$ and $a_{ij} = 0$ if $e_{ij} \notin E$. The adjacency matrix $\mathbf{A}$ of the graph shown in Figure 1 is given below:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Element $a_{ii} = 0$ as there is no self-loop in the graph. Likewise, $a_{24} = 1$ as there exists an edge between the vertices v_2 and v_4. As our research focuses on undirected graphs

without self-loops, the adjacency matrix will always be symmetric as is the case for all undirected graphs (Elspas & Turner 1970) (Fiedler 1973).

Definition 3 (k-order proximity) *Given a vertex $v \in V$, the k-order proximity of v is defined as the set of q vertices at an edge distance less than or equal to k from v and is denoted by $N_k(v)$ (Ying et al. 2018).*

It is also known as neighborhood of radius k or k-hop neighborhood or or k-order neighborhood.

Definition 4 (Subgraph) *Given a set of vertices $S \subseteq V$, the subgraph created by S is a graph that has S as its set of vertices, and it contains every edge of a graph G whose endpoints are in S.*

Definition 5 (Neighborhood subgraph) *The neighborhood subgraph of radius k of the target vertex $v \in V$ is the subgraph induced by the neighborhood of radius k of v, and v itself and is denoted by S_v^k.*

Figure 2 shows the target vertex marked in red and the neighbors of the target vertex in the k-hop neighborhood within $k = 1, 2$, and 3. As shown in the Figure 2, the subgraph S_v^1 is the graph with the target node shown in red color and its 1-hop neighbors shown in grey color; and the edges connecting them.

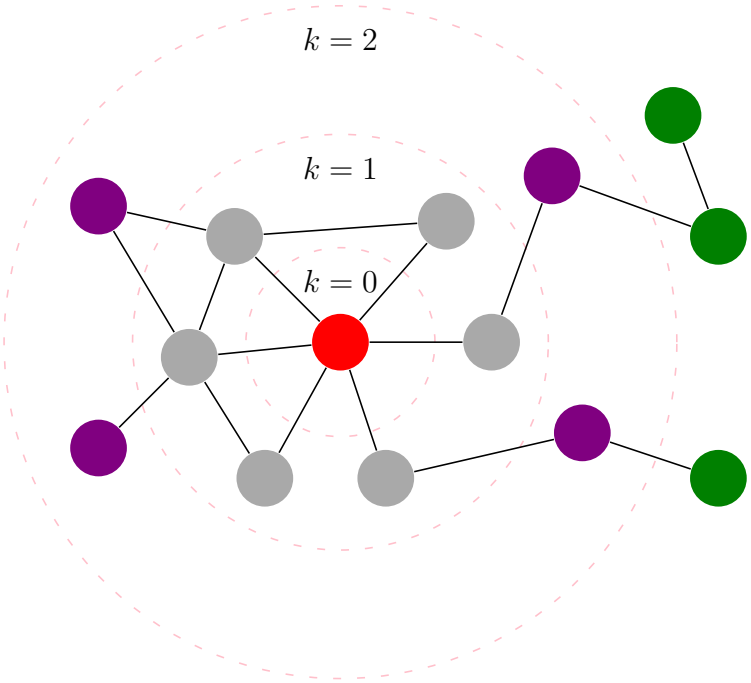


Figure 2: k-order proximity of target vertex marked in red and the neighbors of the target vertex in the k-hop neighborhood within $k = 1, 2$, and 3.

1.1.1 Weighted Graph

In a weighted graph, each edge is assigned a numeric label referred to as a weight (Porfiri & Stilwell 2007). The weight represents how strongly the two vertices are connected. An example of a weighted graph with four vertices and five edges is shown in Figure 3, where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_{12} = 2, e_{24} = 2.6, e_{34} = 3, e_{13} = 1, e_{32} = 1.9\}$, vertices v_1 and v_2 are connected with edge $e_{12} = 2$.

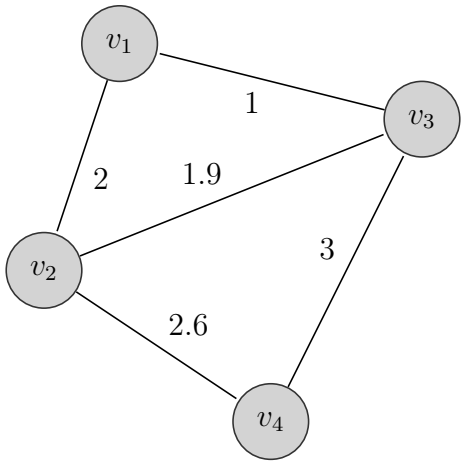


Figure 3: Example of a weighted graph with four vertices and five edges connecting the vertices. The edges have weights associated with them; these weights represent how strongly the vertices are connected.

Definition 6 (Weighted graph) *A weighted graph is defined as* $G = (V, E, \boldsymbol{W})$*, where* $\boldsymbol{W} = [w_{ij}] \in \mathbb{R}^{n \times n}$ *(Porfiri & Stilwell 2007) a graph with weights assocciated with each edge of the graph.*

Given a graph G, let there be a real number weight associated with each edge e_{ij} represented by w_{ij}. Then the graph together with the weights on its edges is called a weighted graph.

Definition 7 (Adjacency matrix of a weighted graph) *Given a weighted graph* G*, the adjacency matrix of a weighted graph is defined as* $\boldsymbol{W} = [w_{ij}]$*, and represents the weights of the edges connecting the* n *vertices of a finite graph (Fiedler 1973).*

In a weighted graph $G = (V, E, \mathbf{W})$, let $v_i, v_j \in V$ denote two distinct vertices, and $e_{ij} \in E$ denote an edge between the vertices v_i and v_j. In this thesis we consider positive weights only, thus $w_{ij} > 0$ if $e_{ij} \in E$ and $w_{ij} = 0$ if $e_{ij} \notin E$. The adjacency matrix $\mathbf{W}$ of the graph shown in Figure 3 is:

$$\mathbf{W}= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1.9 & 2.6 \\ 1 & 1.9 & 0 & 3 \\ 0 & 2.6 & 3 & 0 \end{bmatrix}$$

1.1.2 Attributed graph

Real-life graphs have attributes associated with the vertices in the form of an attribute matrix given by $\mathbf{X}$. These attributes describe the properties of the vertices and are generally represented by a vector $\mathbf{x}_v$. Recent work on graph-structured data has primarily focused on modeling the structure of graphs without attributes (Zhang et al. 2018). There are no significant works done on graph-structured data with real-world structural properties and correlated attributes. This research in deep learning focuses on an approach to exploit correlations among the attributes of linked vertices to predict graph characteristics with greater accuracy. Figure 4 shows an example of an attributed, weighted graph. The figure shows four vertices and five edges. Each vertex is associated with an attribute vector. For example, the second vertex of the graph v_2 has the attributes shown in the second row of the attribute matrix $[x_{21}, x_{22}, x_{23}, \cdots, x_{2d}]$.

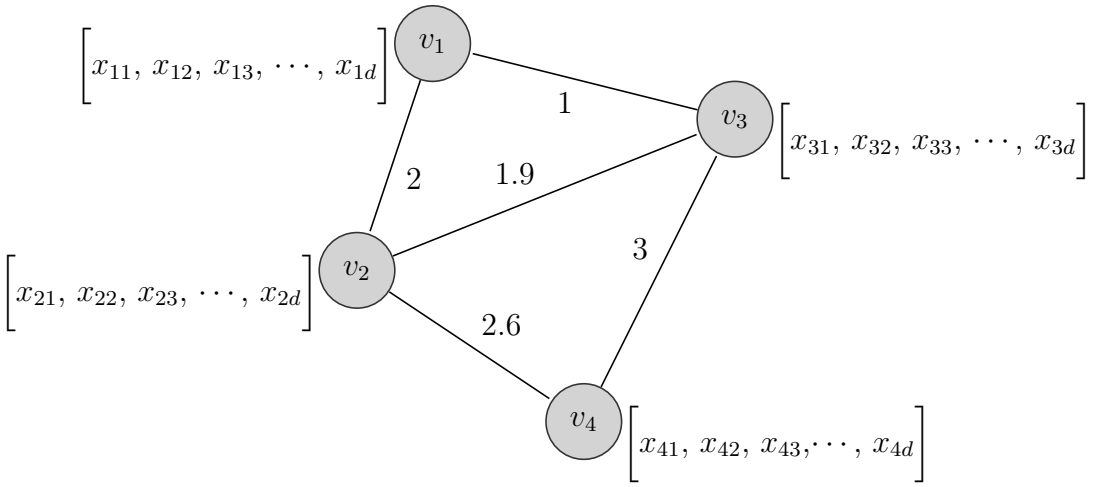


Figure 4: Example of an attributed graph with four vertices and five edges connecting the vertices. Each vertex of the graph contains d dimensional attribute vector.

Definition 8 (Attributed Graph) *An attributed, weighted graph is defined as* $G = (V, E, \boldsymbol{W}, \boldsymbol{X})$*, where* $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ *is the attribute matrix associated with the vertices of the graph, where* d *is the total number of attributes.*

An example of an attribute matrix $\mathbf{X}$ is shown below. Each row in the attribute matrix represents the corresponding attribute vector of a graph's vertex. For example, third row of the attribute matrix $[x_{31}, x_{32}, x_{33}, \cdots, x_{3d}]$ is the attribute vector of the third vertex v_3 in the attributed graph.

$$\mathbf{X}= \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2d} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \cdots & x_{nd} \end{bmatrix}$$

1.2 Graph Data

Graphs represent real-life data from different domains, such as social networks, research networks, and biological networks. To illustrate the diversity of graph data

domains, we begin by introducing several examples of graph data in this section. We can use machine learning algorithms on these graph-structured to do graph classification, prediction, and recommendation.

1.2.1 Social Networks

In a social network, the entity is the users, and users build their network by connecting with other users who are friends or followers. The users act as the vertices, and the type of connection between two users is the edge of the social network. Each user has their own set of properties associated with them, such as the images they have uploaded and their user profile information. An example of a social network is Facebook (Ugander et al. 2011). Figure 5 shows an example of a social network with users acting as the vertices, and the connection between them represents the edge of the network. Using machine learning algorithms, we can potentially recommend new friends to a user based on their connections and properties such as age, interests, and school or work.

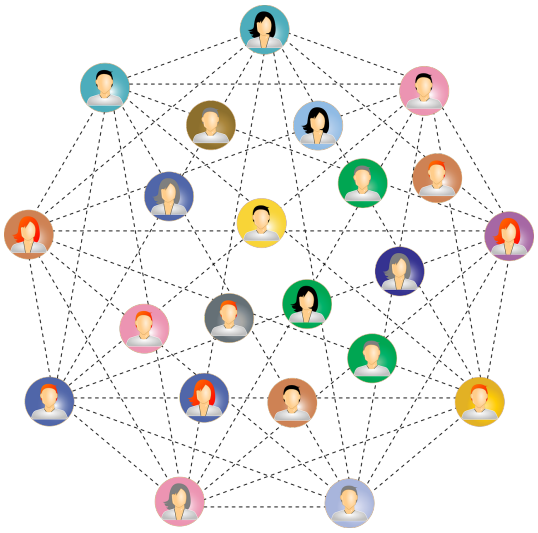


Figure 5: Example of a social network with users acting as the vertices and the connection among them representing the edges of the network.

1.2.2 Research Networks

In a research network, the entity is the researchers, and researchers build their network by connecting with other researchers who are colleagues, research students, collaborators, and followers. The researchers act as the vertices, and the type of research connection between two researchers is the edge of the research network. Each researcher has their own set of properties associated such as the project, research papers, articles that they share, datasets, patents, research proposals, and their user profile information. An example of a research network is ResearchGate (Yu et al. 2016). Figure 6 contains the research network and connections of a professor, students working in the professor's lab, and collaborators. Here professor, students, and collaborators act as the vertices of the network and connections among them represents the edges of the network. Using machine learning algorithms, we can potentially recommend a new research student to a professor, based on the research and collaboration information of the professor and student's research interests.

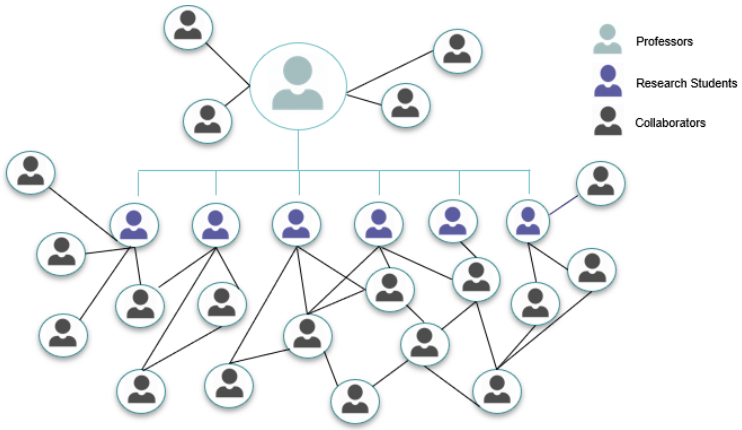


Figure 6: Example of a research network with the researchers acting as the vertices of the network and the connection among them representing the edges of the network.

1.2.3 Citation Networks

In a citation network, the research papers and the authors of research papers are the entities. Authors build their network by following other researchers and also by citing other research papers. Authors and research papers act as the vertices, and

the citation, authorship, and co-authorship of that research paper act as the edges of the citation network. Each research paper is associated with its own set of attributes, such as the text, date of publication, and keywords. An example of a citation network is the CiteSeerX network (Kodakateri Pudhiyaveetil et al. 2009). Figure 7 shows an example of a citation network. The researchers and research papers act as the vertices of the network, and the connections among the researchers and their publications serve as the edges of the network. Using machine learning, we can potentially categorize the research papers into different communities based on the content of the research papers.

Figure 7: Example of a citation network with the research papers and users represented as the vertices of the network and the connections among the authors and research papers represent the edges of the network.

1.2.4 Chemical Networks

In a chemical network, the molecules and the atoms are the entities. Atoms and molecules interact with one another to form the chemical network. Atoms and molecules, along with their properties such as chemical formulas, act as the vertices, and interactions and bonds among them represent the edges of the network. Figure 8 shows an example of a chemical network, which represents L-Lysin, a chemical molecule with corresponding SMILES string representations of molecules (Gómez-Bombarelli et al. 2018). The vertices represent atoms such as Amino radical and

Hydroxide. The edges represent the chemical bonds among them. Deep learning can be potentially used to generate novel chemical compounds with desirable chemical and pharmacological properties from scratch (Li, Vinyals, Dyer, Pascanu & Battaglia 2018).

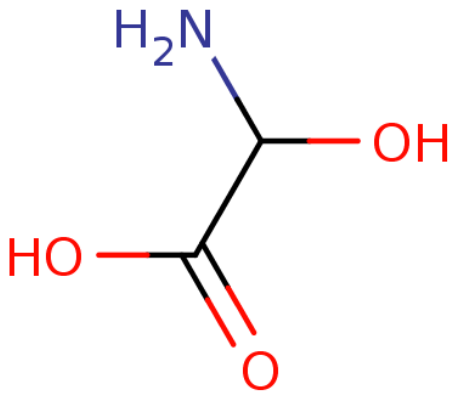


Figure 8: Example of a chemical network representing L-Lysin with the chemical element represented as the vertices and the chemical reactions among the chemical elements represent the edges of the network. The figure has been created using Brenda the comprehensive enzyme information system (Brenda 2019) and (Schomburg et al. 2004).

1.2.5 Biological Networks

In a biological network, genes, and proteins are the entities. Biotic interactions at many levels of detail, from the atomic interactions in a folded protein to the relationship of organisms in a population or ecosystem, can be modeled as networks. Examples of biological networks are protein-protein interaction networks, gene interaction networks, genomic regulatory networks, metabolic networks, signaling networks, neuronal networks, food webs, and species interaction networks. In this thesis, we focus on the study of biological networks such as gene interaction networks, protein structures, and tertiary protein structures. We define a gene interaction network as a set of vertices representing the genes and a set of edges representing the interaction between the genes (Warde-Farley et al. 2010). The co-expression created by the two genes or other functional associations are the interactions between them. The weight of the edge between two genes represents how strongly two genes are connected. We

are especially interested in the physical interactions between two gene products. The attributes of the gene interaction network are the genomic attributes of genetic mutations.

Figure 9 shows an example of gene interaction network. The vertices of this figure are genes of patients diagnosed with prostate cancer such as SYNE3, SUN1, SUN2, and SYNE1. The purple edges show the physical interactions between the genes (Szklarczyk et al. 2014). Using machine learning, we can analyze the topological features of a gene interaction network and perform gene ontology enrichment depending on their functional characteristics. For example, gene SYNE1 is categorized as being a receptor involved in chemical signaling between nerve cells and located on the membrane of Purkinje cells to the actin cytoskeleton. Seven genetic mutations were found in the SYNE1 gene that causes acinar adenocarcinoma, a type of prostate cancer. The mutations are caused by a premature stop signal in the instructions for making the SYNE-1 protein, resulting in a short protein with impaired functions. This type of mutation is commonly known as a nonsense mutation (Özgür et al. 2008).

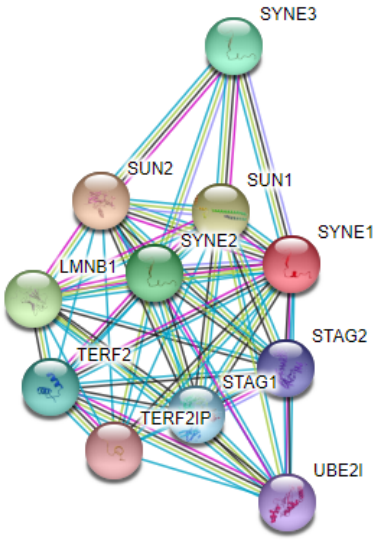


Figure 9: Example of a gene interaction network. The vertices of gene intetaction network are genes and the edges are the interactions between them. The figure has been created using the String protein-protein interaction database (*String* 2018) and (Szklarczyk et al. 2014).

1.3 Ways to Analyze Graphs using Machine Learning

Machine Learning is a scientific way of using computers and algorithms to build analytical models to help computers learn from data without using explicit instructions (Langley 1996). Machine learning has a wide variety of applications such as prediction, classification, and recommendation, clustering, language understanding, language generation, image analysis. Machine learning algorithms build a mathematical model based on sample data, known as "training data" to make predictions of "class labels" without being explicitly programmed to perform such a task. The machine learning task can be classified as supervised, unsupervised, or semi-supervised (Langley 1996).

In supervised machine learning tasks, the machine tries to learn a function that maps the input data to output labels by analyzing the input-label pairs from the training data. A good supervised machine learning algorithm will allow determining the class labels for unseen input data accurately (Langley 1996). In unsupervised machine learning tasks, the machine tries to learn a function that maps the input data to output labels using the training dataset without pre-existing labels (Langley 1996). The semi-supervised machine learning task is a hybrid approach where only a few input-label data are available in the training dataset (Langley 1996).

In this thesis, we present a supervised learning approach, where labeled data is available. On the other hand, advancement in technology has allowed us access to large datasets in the form of graphs. There are many ways in which we can use machine learning on graph datasets. We discuss vertex classification, edge prediction, community detection, and graph classification below. In this thesis, we focus on graph classification of weighted attributed graphs. In graph classification, the objective is to train a model on the training dataset with class labels and to classify unseen graphs to the corresponding graph labels.

1. Vertex Classification: Given a graph for which the labels of some vertices are known, and others are unknown, machine learning can be used to predict the

unknown labels ((Kipf & Welling 2016) and (Bruna et al. 2013)). Figure 10 shows an input graph with two known class labels: red and yellow. The machine learning task is to predict the unknown labels of the vertices marked in grey.

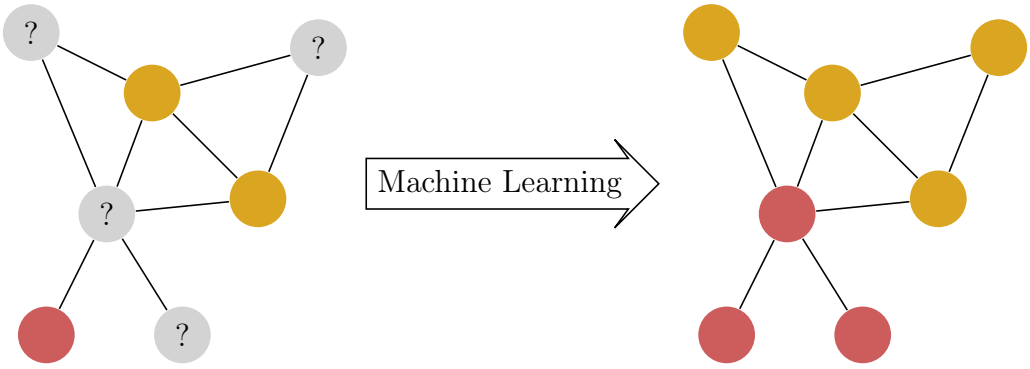


Figure 10: Example of vertex classification. Left: an input graph with two known class labels red and yellow, and unknown vertices marked in grey. Right: using machine learning, the unknown vertices are classified as either red or yellow.

2. Edge Prediction: Given a graph for which the edges between some vertices are known, machine learning can be used to predict whether an edge exists between two given vertices ((Duvenaud et al. 2015) and (Monti et al. 2017)). Figure 11 shows an input graph with some edges known and others unknown. The unknown edges in Figure 11 are marked with question marks. The machine learning task is to predict if an edge exists between the vertices or not.

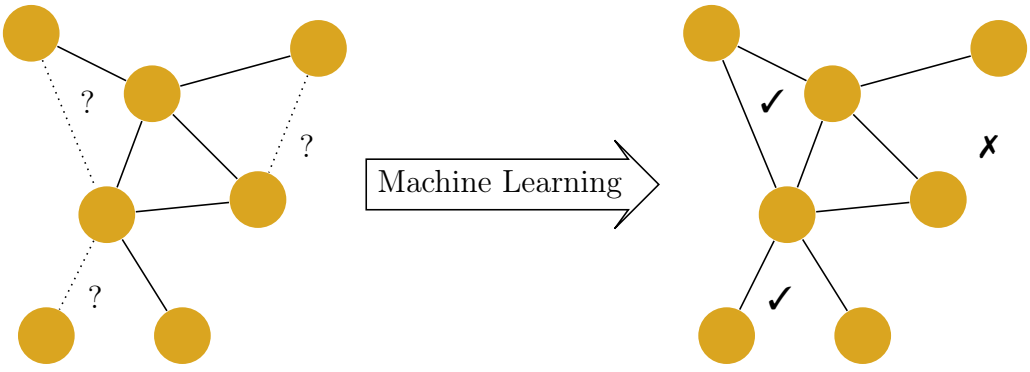


Figure 11: Example of edge prediction. Left: an input graph with some of the edges known and some unknown; unknown edges are marked with question marks. Right: machine learning is used to predict the existence of an edge between two vertices.

3. Community Detection: Community detection is a fundamental and widely stud-

ied problem that finds all densely-connected groups of vertices that separate them well from others in a graph. Given a graph that represents a dense community of vertices, machine learning can be used to predict the community of unknown vertices ((Hamilton et al. 2017), (Grover & Leskovec 2016), and (Fortunato & Hric 2016)). Figure 12 shows a graph with three communities: red, green, grey, and yellow. For example, A_3, B_3, C_3, D_3, and E_3 represents the distinct vertices of a dense community colored green.

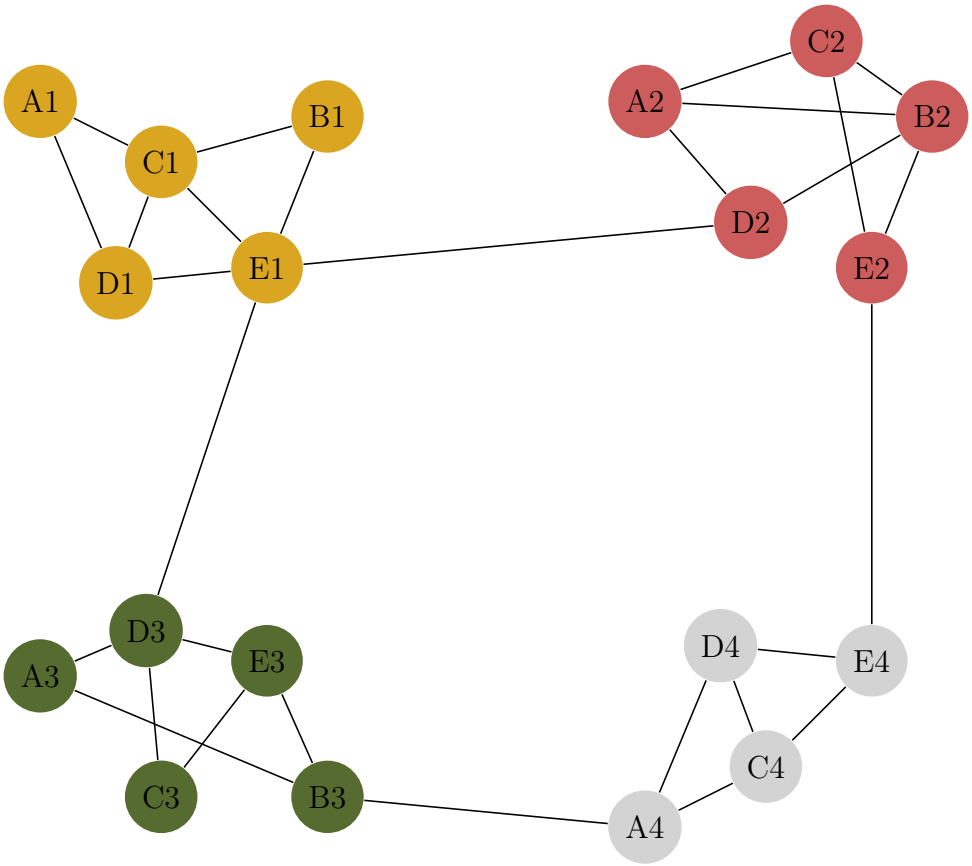


Figure 12: Example of community detection.

4. Graph Classification: Consider a labeled dataset of undirected graphs that may or may not be weighted or attributed. The machine learning task is to predict the label of the graph for which the labels are unknown (Zhang et al. 2018). Figure 13 shows an input dataset of graphs for which the graph labels are known. We use this input to learn a function that can classify the graphs for which the labels are unknown.

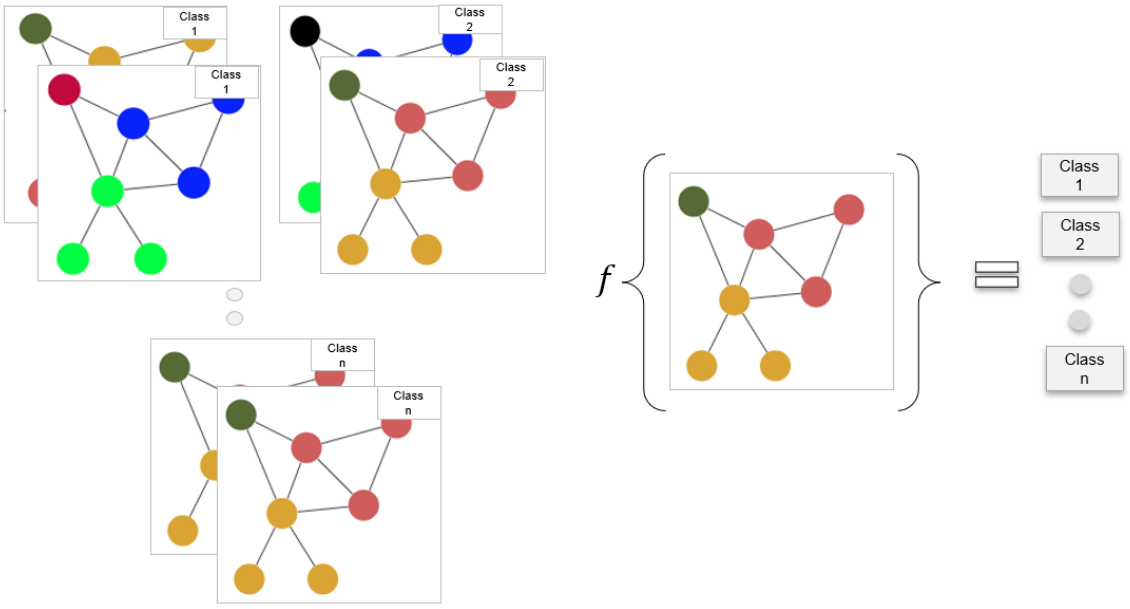


Figure 13: Given a dataset of a weighted, attributed graph with labels. The machine learning task to learn a function f that takes in as input a graph with unknown labels and predicts the class label of the graph.

We are interested in the supervised graph classification problem on weighted attributed graphs as most of the real-world graph data is weighted and attributed. Research in this field has a lot of applications in healthcare and life science, social networks, aerospace, communications and media, and other industries.

1.4 Problem Formulation

In this research, we consider the problem of classifying an input dataset of weighted, attributed graphs where labels are only available for a small subset of graphs. Our objective is to assign the input graph data with unknown labels to the corresponding class labels. We frame this problem as graph-based supervised learning. The objective of our method is to reduce the classification error. Formally, we can define this problem as follows. Given a weighted, attributed graph $G = (V, E, \mathbf{W}, \mathbf{X})$ and $\mathbf{x}_{v_i}$ is the attribute vector associated with the vertex $v_i \in V$, where V is a finite set of vertices in the given graph G. Let $Y = \{y_1, y_2, y_3 \cdots y_l\}$ be the set of l labels. The training dataset is defined by $S_t = \{(G_1, y_i), (G_2, y_i), \cdots (G_t, y_t)\}$, where t is the

total number of training samples. The objective is to learn a function $f : G \mapsto y_i$, which predicts the class label y_i of a given graph G. The classification function f is computed by minimizing the cross-entropy error (Creswell et al. 2017). The error between the predicted score and the expected label for binary classification is given by Equation (1):

$$L(y_i, \hat{y}_i) = -y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\,, \tag{1}$$

where $\hat{y}_i$ is the probability score of predicted label and y_i is the true label. As per Equation (1), the cross-entropy error increases as the predicted label probability diverges from the actual label. The error between the predicted score and the expected label for l labels in multi-class classification is given by Equation (2):

$$L(y_i, \hat{y}_i) = -\sum_{l=1}^{l} y_{i,l} \log(\hat{y}_{i,l})\,. \tag{2}$$

1.5 Applications

In this thesis, we apply the proposed method to solve three labe; classification problems in the weighted, attributed graph-structured dataset. We use three bioinformatics datasets, which are ENZYMES, GINA, and D&D. All these datasets have complex graph structures, which are difficult to analyze using traditional machine learning algorithms. The classification problems are described as follows:

1. We use attributed ENZYME dataset (Schomburg et al. 2004) generated using (Brenda 2019) to classify the tertiary protein graph structures to one of the six enzyme commission numbers.

2. We use attributed D&D dataset (Kersting et al. 2016) to classify the protein graph structures as enzyme or non-enzyme type.

3. To classify the cancer types to either aggressive or non-aggressive based on the Gleason score of the patient, we generated a graph dataset called GINA. GINA

has been created using the GeneMANIA Cytoscape plugin(Warde-Farley et al. 2010) and cBioPortal (*cBioPortal* 2012). GINA is a dataset of gene interaction networks of patients diagnosed with prostate cancer. The genetic attributes in terms of mutations are the vertices of this network.

The objective is to train an algorithm such that the model learns to encode a new representation of the input graph using attributes and label information present on the graph. We use these representations to determine a function that takes an unseen weighted, attributed graph as input and predicts its class label accurately. With traditional machine learning algorithms accomplishing this task is difficult (Wu et al. 2019); we use robust deep learning algorithms to achieve the desired results. The deep learning algorithms that work well with graph data are called graph neural networks (Zhang et al. 2018).

1.5.1 Deep Learning on Graphs

Deep Learning algorithms are strong machine learning algorithms capable of understanding the complex latent attributes of data from any domain without the need to perform explicit attribute engineering (Grover & Leskovec 2016). Deep learning has revolutionized challenging tasks from natural language generation ((Dai et al. 2015), (Le & Mikolov 2014), (Devlin et al. 2018) and (Jackson & Moulinier 2007)), to image recognition (Hjelmås & Low 2001). The input data of these tasks have a fixed grid-like structure and are generated from the Euclidean domain. For example, images have a two-dimensional grid structure, while text data has a one-dimensional structure. Due to the significant amount of data being generated from non-Euclidean domains (Zhang et al. 2018), researchers have focused on understanding data with a complex structure such as three-dimensional images and data networks. The extant algorithms are not powerful enough to handle these complex data structures. As a result, researchers are borrowing ideas from CNN (Krizhevsky et al. 2012), recurrent neural networks (Schuster & Paliwal 1997), and deep autoencoders (Vincent et al. 2010) to design the architecture of graph neural networks. This thesis focuses on ex-

tending deep learning approaches, especially CNNs, to weighted, attributed graphs. This class of algorithms is generally known as graph convolutional neural networks (GCN).

1.5.2 Graph Convolutional Neural Networks

In deep learning, Graph Convolutional Neural Networks (GCNs) are a class of neural networks, explicitly designed for in-depth analysis of graph-structured data (Bruna et al. 2013). As the name indicates, the architecture of a GCN is an efficient variation of the architecture of CNN, which works efficiently for analyzing visual imagery. The convolution and pooling operations in CNN that work well on grid-structured data are generalized in GCN that has graph-structure. Existing GCNs follow two approaches for convolution operation: spectral-based and spatial-based approaches (Wu et al. 2019). Spectral-based GCNs has a convolution operation defined by filters inspired by graph signal processing. These models are capable of learning spectral filters that are defined by eigendecomposition of the Laplacian matrix (Kipf & Welling 2016), (Defferrard et al. 2016), (Bruna et al. 2013). In spatial-based approaches, the attribute representation of a vertex is updated in each layer of neural network architecture by aggregating the attributes of neighboring vertices (Hamilton et al. 2017). While GCNs operate on the vertex-level, graph pooling modules can be interleaved with the GCN layer to coarsen graphs into high-level sub-structures. Most of the existing graph neural network architectures work well on vertex classification. Little research has been done on graph classification on weighted, attributed graphs.

In this thesis, we present a novel deep neural network architecture inspired by CNN, called AGCP. AGCP takes as input highly complex weighted, attributed graph-structured data and predicts the label for unseen graphs. We are interested in biological networks.

1.6 Thesis Motivation

With a more in-depth analysis of graphs, we discovered that vertices that are connected strongly by edges have similar attributes and have the same class label (Zhang et al. 2018). Rather than studying the whole structure of the graph, it is crucial to find a new graph substructure that captures the high-level attributes by analyzing the attributes of a vertex and its neighbors. Powerful machine learning algorithms such as CNNs are good at capturing and representing the data by aggregating the attributes of its neighbors. CNNs work effectively on fixed grid-structured data such as images, which have a well-defined spatial ordering. Graphs, on the other hand, have an arbitrary size and complex topological structure, no fixed node ordering, or point of reference. Graphs are often dynamic and have multimodal attributes (Hamilton et al. 2017). It is a big challenge to replicate the architecture of a CNN that works well with fixed grid-structured data to graph-structured data because of the complex nature of graph-structured data. Reproducing a significant graph pooling algorithm, which is as efficient as max-pooling or average-pooling in CNNs, is also a considerable challenge. Our motivation for this thesis is to tackle the difficult problem of treating graph-structured data like fixed grid-structured data.

We also consider the problem of grading the tumor by predicting the aggressiveness of cancer to either aggressive or non-aggressive cancer type. Cancer is a genetic disease caused by mutations of genes. Cancer is characterized by the development of abnormal cells that divide uncontrollably and can infiltrate and destroy healthy body tissue (Futreal et al. 2004). A mutation is any change that occurs in DNA, either due to mistakes when DNA is replicated or as the result of environmental factors such as UV light or cigarette smoke. Prostate cancer is a leading cause of cancer-related mortality worldwide. Among men in Canada, prostate cancer is the most commonly occurring cancer and is a leading cause of cancer-related deaths (Alkhateeb et al. 2019) and (Hamzeh et al. 2017). The Gleason grading system remains the most potent prognostic predictor for patients with prostate cancer since the 1960s (Alkhateeb et al. 2019). Its application requires highly-trained pathologists, is tedious and, suffers from

limited inter-pathologist reproducibility, especially for the intermediate Gleason score 7. Automated annotation models in machine learning constitute a viable solution to remedy these limitations (Albertsen et al. 1998). We empirically found that our proposed model, AGCP, could be applied to GINA to predict the aggressiveness of cancer to help pathologists in the diagnosis of prostate cancer.

Types of Mutations: In this work, we consider the following types of mutations missense, nonsense, insertion, deletion, duplication, frameshift, and intron. A missense mutation is a change in a DNA base pair, which results in the substitution of amino acid for another one in the protein made by a gene interaction. A nonsense mutation is a change in one DNA base pair; in the altered DNA sequence, signals the cell to stop synthesizing the protein prematurely. A nonsense mutation results in a shortened protein that functions improperly. An insertion mutation is a change in the number of DNA bases in a gene by inserting a piece of DNA. As a result, the protein(s) made by the affected gene may not function properly. A deletion mutation changes the number of DNA bases by removing a few base pairs within a gene or several genes in the neighborhood. The deleted DNA alters the function of the resulting protein(s). A duplication mutation occurs when a piece of DNA is copied one or several times, resulting in improper functioning of protein(s). A frameshift mutation occurs whenever there is an addition or loss of DNA bases, which in turn changes a gene's reading frame. A reading frame consists of groups of three bases, each encoding for one amino acid. The resulting protein is usually nonfunctional.

1.7 Thesis Statement

In this thesis, we present:

1. A deep neural network architecture, AGCP, that:
    - Takes arbitrary sized, weighted, undirected, labeled, and attributed graphs as input.

- Learns a function to encode the label and attribute information of input graphs to solve classification and prediction problems on unseen graphs.

2. We propose a statistically significant pooling layer based on the principles of attribute selection approaches for classification. The pooling layer eliminates vertices to coarsen the graph based on the information gain of the attributes present on each vertex.

3. The proposed model is applied to GINA, a dataset of gene interaction network with genomic attributes in terms of mutations as attributes of the vertices to classify the aggressiveness of prostate cancer and two other benchmark bioinformatics datasets. We applied AGCP on ENZYME dataset to correctly assign each enzyme to one of the 6 EC top-level labels and D&D dataset to classify if a protein is an enzyme or a non-enzyme.

4. We compare the performance in terms of classification accuracy and AUC/ROC with two other state-of-the-art models for graph classification. We experimentally demonstrate that our model outperforms previously proposed methods by a considerable margin.

1.8 Thesis Contributions

1. **A new paradigm for graph classification.** We present AGCP, an end-to-end deep attributed graph convolutional neural network with a pooling layer which statistically downsamples the graph. AGCP is a novel supervised learning model that works on attributed, weighted, graph-structured data for graph classification based on the aggregation of attribute and label information of neighboring vertices.

2. **Better pooling strategy.** We propose a pooling layer thath uses information gain as a strategy to coarsen the graph while preserving the global structure of the original input graph.

3. **Application of a novel strategy for dimensionality reduction in graph-structured data.** We applied AGCP on:

   (a) GINA dataset: We generated a dataset of attributed, weighted graphs called GINA, which is a dataset of gene interaction networks with genomic attributes in terms of mutations as attributes of the vertices to classify the aggressiveness of the prostate cancer.

   (b) ENZYME dataset: ENZYME is a dataset of tertiary protein structures. The classification problem is to assign each enzyme to one of the six EC top-level classes accurately.

   (c) D&D dataset: D&D is a dataset of protein structures. The classification problem is to classify the graph structures to either enzyme or non-enzyme.

1.9 Thesis Organization

The organization of the thesis is as follows. In Chapter 2, we discuss the background of artificial neural networks, deep neural networks, and the motivation behind the technique used in our thesis, which is CNNs. In Chapter 3, we provide a literature review of some existing works in graph classification using deep learning. We discuss spatial-based convolutions, spectral-based convolutions, and graph representation learning. In Chapter 4, we describe our proposed methodology in detail. In Chapter 5, we present several experiments on benchmark bioinformatics datasets along with their results. We provide a comprehensive result analysis and some insights we acquired from the experiments performed in the later sections of this chapter. Finally, in Chapter 6, we discuss some future research directions and the conclusions of our work.

Chapter 2

Convolutional Neural Networks

This chapter presents the motivation of technique used in our thesis, CNNs. Before we describe the working of the complex architecture of the CNN model, we first introduce an artificial neural network (ANN), the elements of ANN followed by a deep neural network (DNN).

2.1 Artificial Neural Networks

An artificial neural network (ANN) or simply neural network is an algorithmic model run in software on a digital computer that is designed to mimic the way in which the human nervous system performs a particular task or function (LeCun et al. 2015). Both the human nervous system and neural networks can be viewed as three-stage systems, as shown in Figures 14 and 15, respectively. In the human nervous system, the receptors receive a stimulus from the external environment and convert it into electrical impulses that pass information to the brain. The effectors convert the electrical impulses generated by the brain into a response to the stimulus. There is a double-headed arrow in Figure 14 to show forward pass, forward passing of information-carrying signals through the brain and backward pass, indicating the presence of the back-propagation of the feedback. The feedback helps the brain to learn the response to stimulus more accurately.

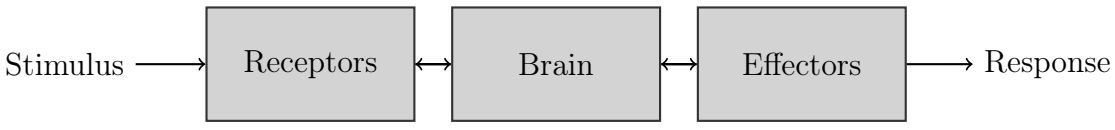


Figure 14: Block diagram of the brain.

A neural network takes an input, passes it through multiple layers of hidden neurons, and then outputs a prediction representing the combined input of all neurons in the network. Figure 15 shows a schematic view of an artificial neural network with an input layer, a hidden layer, and an output layer. The input to the neural network is a dataset with images of either cats or dogs. Each neuron in the output layer predicts the probability of the image belonging to its respective class label (cat or dog).

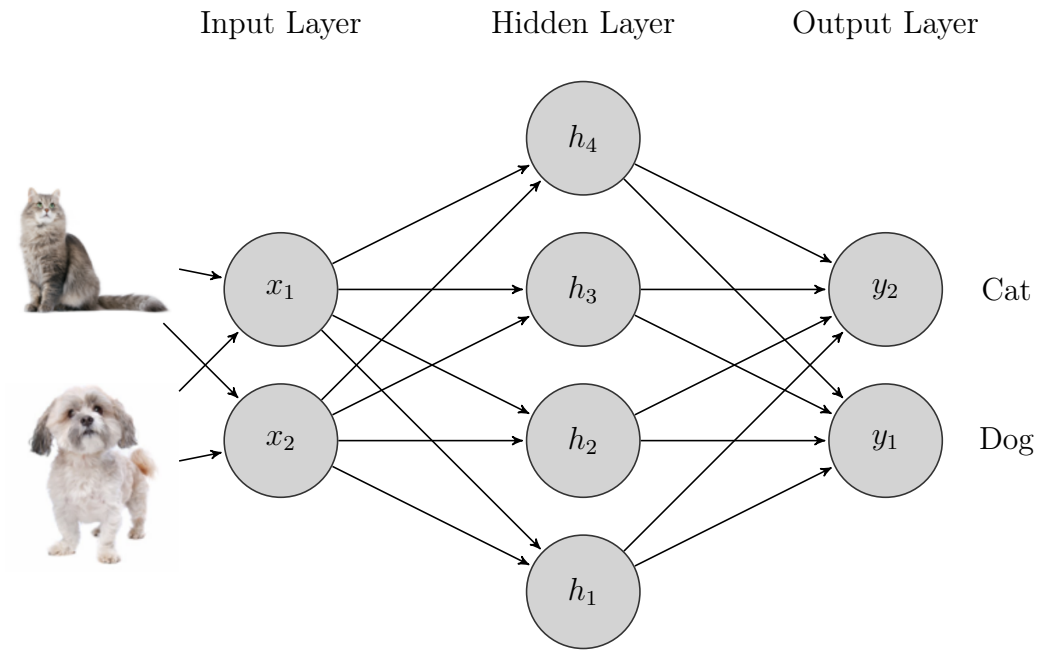


Figure 15: Example of an artificial neural network with an input layer, a hidden layer, and output layer. The input to the neural network is a dataset with images of either cats or dogs. The output layer is the probability of each of the possible labels.

Similar to the receptors in the human nervous system, the input layer shown in Figure 15 provides information from the dataset to the hidden layer of the neural network. The input layer consists of one or more input neurons, which are information processing units that are fundamental to the operation of a neural network.

The brain and hidden layers are central to their respective systems. Both the brain and the hidden layer continually receive information, process it, and make appropriate decisions. The hidden layer is shown in Figure 15 receives the input from the input layer, processes the information, performs the computation, and finally passes the

prediction to the output layer. This process is called forward propagation. The hidden layer consists of one or more hidden neurons (Goodfellow et al. 2016).

The output layer, as shown in Figure 15 is responsible for analyzing the weighted sum of all neurons in the hidden layer and then calculating the probability of the image belonging to each of the possible class labels similar to the effectors in the brain. The number of output neurons is equivalent to the number of known classes in the input data. For a binary classification problem, there are two output neurons. In a supervised learning approach, the predicted probability of a class label is compared to the actual class label to calculate the error in classification. This error is then backpropagated to adjust the calculations and values in subsequent forward propagations.

2.1.1 Neural Network Element - Artificial Neuron

An artificial neuron is a mathematical function that takes a group of weighted inputs, applies an adder function, followed by the activation function, and then returns their activated weighted sum as an output (Goodfellow et al. 2016). Figure 16 shows a model of an artificial neuron k with a set of inputs $\{x_1, x_2 \cdots x_m\}$ and their corresponding weights $\{w_1, w_2 \cdots w_m\}$, an adder function σ, an activation function $\phi(.)$, and the output of the neuron k given by y_k.

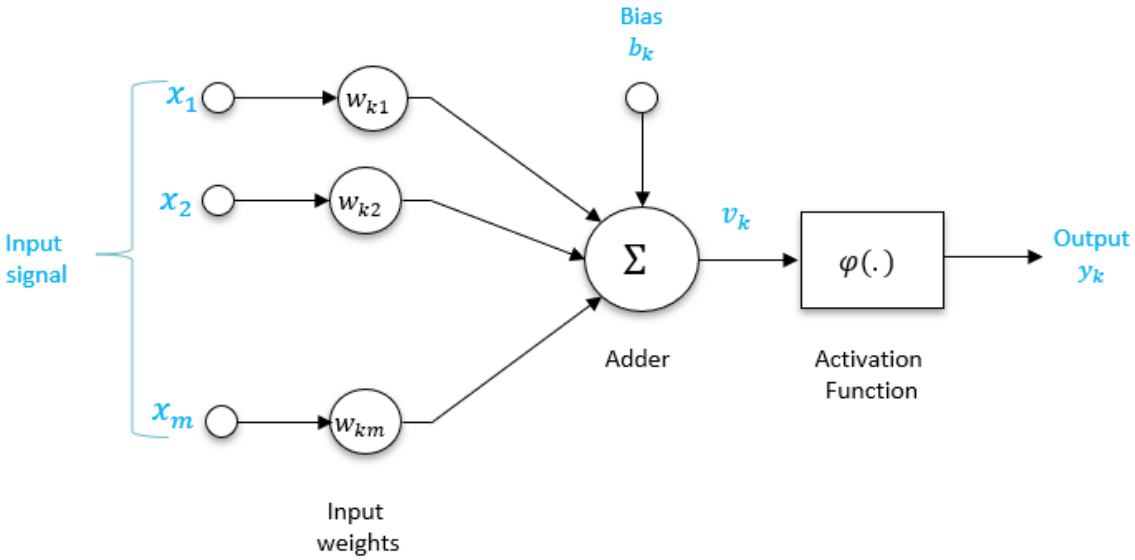


Figure 16: A model of an artificial neuron k with a set of inputs $\{x_1, x_2 \cdots x_m\}$ and their corresponding weights $\{w_1, w_2 \cdots w_m\}$, an adder function $\sum$, an activation function $\phi(.)$, and the output y_k.

Here, we identify the four basic elements of an artificial neuron:

1. A set of **inputs**, each of which is associated with a weight, w. As shown in Figure 16, a signal x_m at the input m connected to neuron k is multiplied by the weight w_{km}. The first subscript in w_{km} refers to the neuron, and the second subscript refers to the input end to which the weight refers. The weight of an artificial neuron belongs to the set of real numbers and can have a negative or positive value. A higher weight indicates a stronger activation of the neuron; likewise, a lower weight indicates that the neuron is not very active.

2. An **adder** is used for summing the weighted input signals.

3. **Bias** is an additional parameter in the neural network that is used to adjust the weighted sum of the neuron inputs to specific desired values allowing the model to achieve a better fit for the given data.

4. An **activation function** is applied to the output of the adder for limiting the amplitude of the output of a neuron ((Goodfellow et al. 2016) and (Karlik & Olgac 2011)). The activation function limits the acceptable amplitude range of the output signal to some finite value.

In mathematical terms, we can describe the neuron k depicted in Figure 16 by the following equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \,, \tag{1}$$

and

$$y_k = \varphi(u_k + b_k) \,, \tag{2}$$

where $x_1, x_2, \cdots x_m$ are m input signals, $w_{k1}, w_{k2}, \cdots w_{km}$ are the corresponding weights of the neuron k, u_k (not shown in Figure 16) is the linear adder, b_k is the bias, $\varphi(.)$ is the activation function, and y_k is the output of the neuron. The effect of the bias on the linear adder function can be seen as follows:

$$v_k = (u_k + b_k) \,. \tag{3}$$

During the training phase of the ANN, the neural network learns the weights and bias, also known as the state of the neural network that can produce optimal output probability. The predicted probability is then compared to the actual probabilities to calculate the error or loss or cost. The error is then backpropagated through the network to update the current state of the neural network.

2.1.2 Loss Function

The loss function, also known as cost function or error function, is a measure of the capacity of a neural network to approximate the true function that produces the desired output from input data. In supervised learning, the desired output is the ground truth labels or classes y_i of each sample in the dataset. In a single forward pass, the neural network takes as inputs the states of the neural network given by weights, biases, and examples from the training set and predicts the labels $\hat{y}_i$ for each sample. The difference between the predicted labels and the ground truth labels y_i are calculated, which is the total loss of the network. Although many loss functions exist, all of them essentially penalize on the difference or error between the predicted $\hat{y}_i$ for a given sample and its actual label y_i. In this thesis, we use cross-entropy error,

given by:

$$L(y_i, \hat{y}_i) = (\hat{y}_i \log(y_i) + (1 - \hat{y}_i) \log(1 - y_i)) \,. \tag{4}$$

From Equation (4), we can decipher that cross-entropy loss increases as the predicted probability diverges from the actual label. The objective of the neural network is to reduce the loss function to a minimum. With the help of optimization techniques such as stochastic gradient descent, the loss function learns to reduce the error in prediction gradually.

2.1.3 Types of Activation Functions

The activation function is crucial in the training of an ANN. Their primary purpose is to convert an input signal of a node in an ANN to an output signal. There are three commonly used activation functions sigmoid, tanh, and ReLU. We briefly discuss each of these functions below.

Sigmoid: The sigmoid function, whose graph is "S"-shaped as shown in Figure 17, is the most common activation function used in the construction of neural networks. The sigmoid function is a strictly increasing function and exhibits a graceful balance between linear and non-linear behavior. The sigmoid non-linearity function has the following mathematical form (Han & Moraga 1995):

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \,. \tag{5}$$

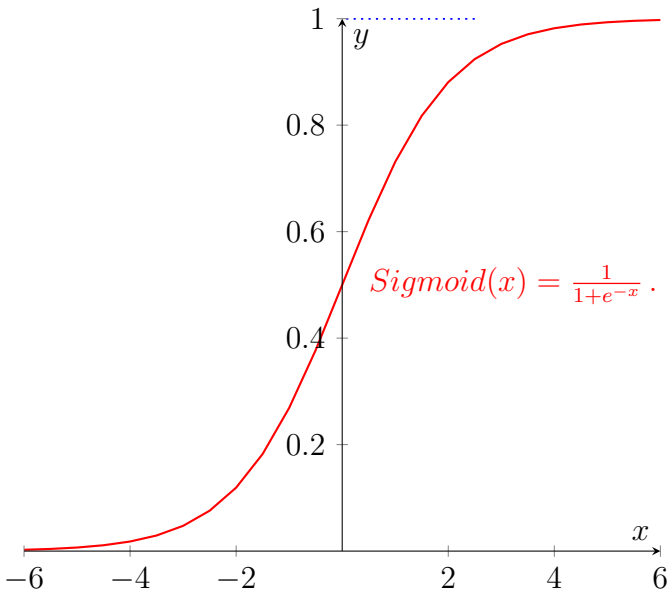


Figure 17: Sigmoid activation function.

The sigmoid function takes as input a real value and limits the amplitude range to between 0.0 and 1.0 so that the range of values do not become too large or too low. However, the downside of the sigmoid activation function is the vanishing gradient (Karlik & Olgac 2011). If the activation of the neuron is too large at either of the two axes, the network becomes saturated at higher values. Thus, the gradient at these regions approaches zero. The second issue with the sigmoid activation function is that it is not zero centered, which makes the gradient updates go too far in all possible directions. With the output having a value between zero and one, it makes optimization more difficult to achieve. The hyperbolic tangent function overcomes this downside.

Tanh: Mathematically, a hyperbolic tangent function is defined as follows:

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} . \tag{6}$$

The tanh function takes a real value as input and limits it between -1 and 1 (Xiao et al. 2005). Its output is zero-centered, and hence the optimization is easier. However, even this activation function suffers from the "vanishing gradient" problem in the positive and negative domains similar to the sigmoid function. Figure 18 shows a

graph of the hyperbolic tangent activation function.

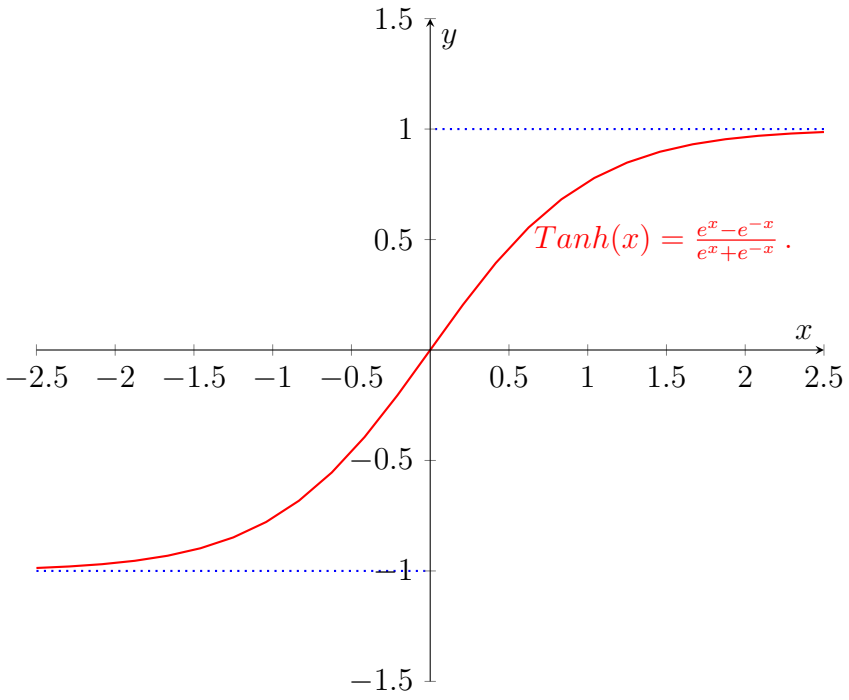


Figure 18: Hyperbolic Tangent activation function.

The issue with a vanishing gradient can be overcome using a ReLU activation function.

ReLU: The rectified linear unit activation function has gained a lot of popularity these days because it avoids the vanishing gradient problem in the positive axis (Li & Yuan 2017). The ReLU function takes a real value as input and limits it between 0 and $+\infty$. The ReLU has the following mathematical form:

$$ReLU(x) = max(0, x)\,, \tag{7}$$

where x is the input to the neuron. Figure 19 shows a graph of the rectified linear unit activation function.

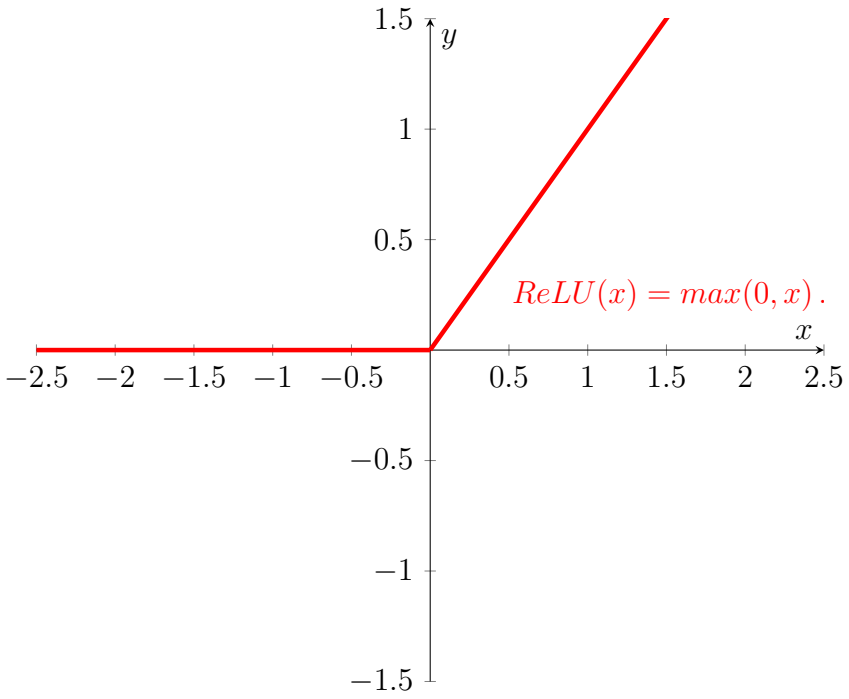


Figure 19: Rectified Linear Unit activation function.

Softmax: The softmax activation function is an efficient variant of the sigmoid activation function for multi-class classification. Softmax is typically used as the output of classifier, to represent probability distribution over n classes. Softmax compresses values to positive values between 0.0 and 1.0. The softmax function has the following mathematical form:

$$Softmax(x) = \frac{exp(x_i)}{\sum_j exp(x_j)} \,. \tag{8}$$

Figure 20 shows a graph of the softmax activation function.

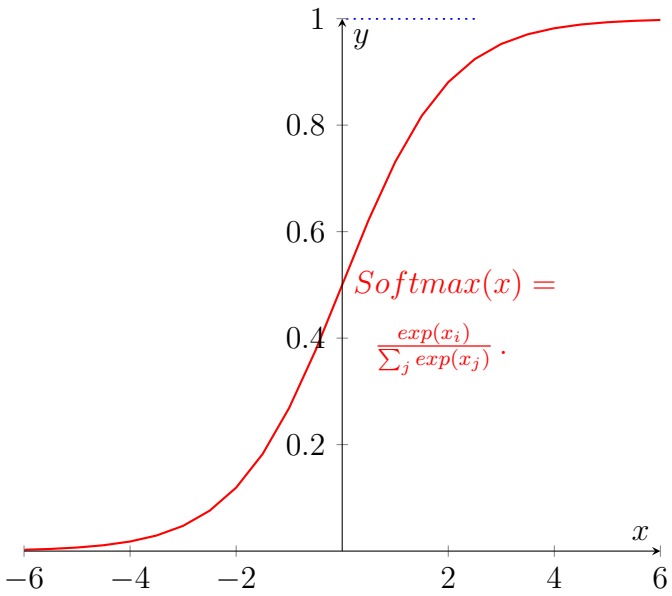


Figure 20: Softmax activation function.

2.1.4 Deep Neural Networks

A deep neural network or DNN is an artificial neural network with multiple hidden layers (Goodfellow et al. 2016). Figure 21 shows an example of deep neural network architecture with two hidden layers for the ease of representation. It has two hidden layers: h_1 and h_2, each hidden layer has four neurons, which is represented by h_{ij}, indicating the j^{th} neuron in the i^{th} hidden layer. For example, in Figure 21, h_{23} represents the third neuron in the second hidden layer. The input data is passed to each of the four neurons in the hidden layer h_1 by the input layer x_1. The neurons do the mathematical calculations to identify the low-level attributes of the input data. Each neuron in the hidden layer has an activation after computation, and the neurons pass these activations to the next layer h_2. The neurons in h_2 perform mathematical calculations to identify the high-level attributes of the input data and then outputs a value y_i, which is the probability of the input data belonging to one of the two or more class labels. Figure 21 takes, as input, a dataset with images of either dogs or cats. The output layer gives the probability of the labels cat and dog.

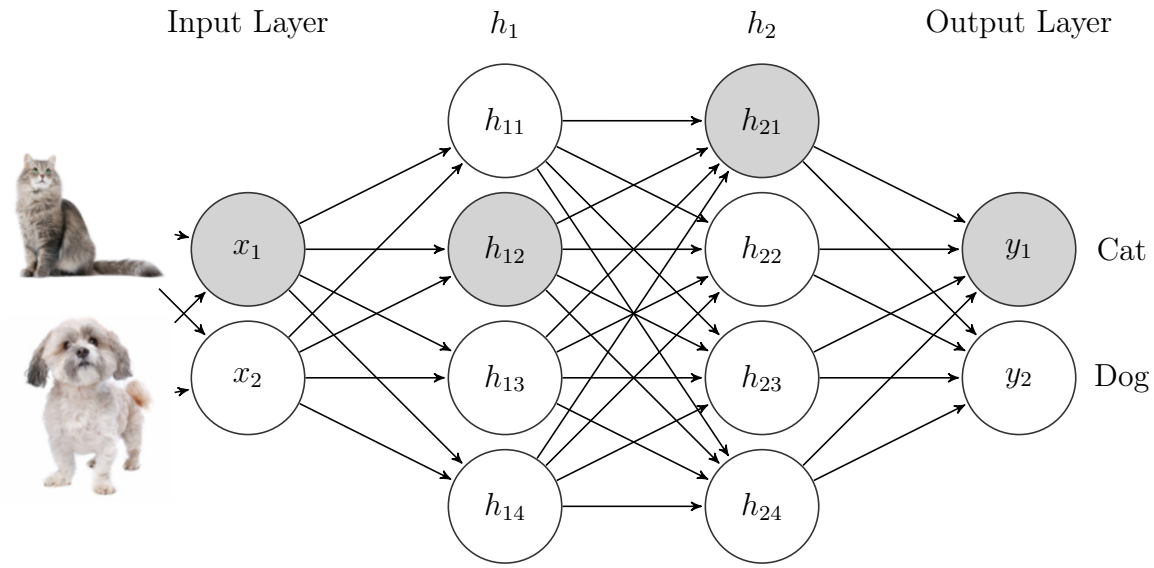


Figure 21: Example of a deep neural network with an input layer, multiple hidden layers, and an output layer. The input to the neural network is a dataset with images of either cats or dogs. The output layer gives the probability of the labels cat and dog.

Deep learning is the study and application of deep neural networks to solve a particular task. These tasks include classification, prediction, representation learning, recommendation, and dimensionality reduction. Deep neural networks are mainly used to analyze complex data such as text and images.

2.2 Convolutional Neural Networks

CNN (LeCun et al. 1989) is very similar to an ANN and is designed to perform on fixed grid-structured data. They contain neurons that have learnable weights and biases. The CNN architecture is inspired by the biology of human vision and is most commonly applied to analyzing visual imagery. Each neuron receives an input, performs a dot product, and optionally applies non-linearity, also known as an activation function. There are three main types of layers used to build CNN architectures: the convolutional layer, the pooling layer, and the fully-connected layer. These layers are interleaved to form the full CNN architecture (Krizhevsky et al. 2012), as shown in Figure 22.

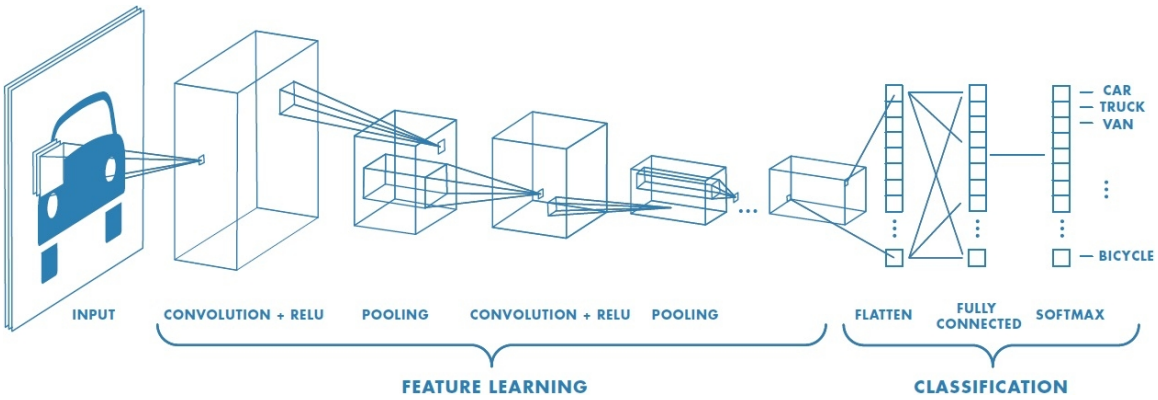


Figure 22: Architecture of a convolutional neural network (*Mathworks* 2019).

Each neuron in the input layer takes an image pixel as the input and passes it to the corresponding neurons in the next layer. The image region covered by these neurons is called the activation map or feature map. Each neuron in the hidden layer takes an activation map from the previous layer as input and transforms it into many different representations by applying convolution and pooling filters. Filters are two-dimensional matrices of weights, which are analogous to the weight parameters of the traditional DNN described in the previous section.

2.2.1 Convolutional Layer

The convolutional layer is the most significant layer in the CNN model and performs most of the computations. The parameter of the convolutional layer is a $(w \times h \times d)$ learnable filter vector, where w is the filter pixel width, h is the height, and d is the filter depth. For example, a typical filter is a $(5 \times 5 \times 3)$, representing five-pixel width and height and three-pixel depth for three color channels. Typically, a CNN model uses multiple filters to capture multiple features of the input image. During forward propagation, each filter slides over the width and height of the input volume and calculates the sum of the dot products between the entries of the filter and the input at each position. As the filter slides over the width and height of the input volume, a two-dimensional activation map is produced. The activation map encapsulates the representation of low-level features present in the input volume, such as an edge or blotch of color. This activation map is passed on to the subsequent layers to capture

the high-level features of the input volume, such as the wheels or headlights of a car. The convolutional layer is followed by a non-linear activation function ReLU. Figure 23 shows an example of a convolutional layer in action. The 3×3 filter convolves over the 5×5 input volume. The convolutions start from the top left corner of the input volume and end at the bottom right corner and cover the whole image. The top-left pixel is called the target pixel, and by performing convolution operation, we generate new representations of that target pixel. The top-left pixel of the activation map is calculated by element-wise matrix multiplication and summation of overlapping pixels from both the input and the filter.

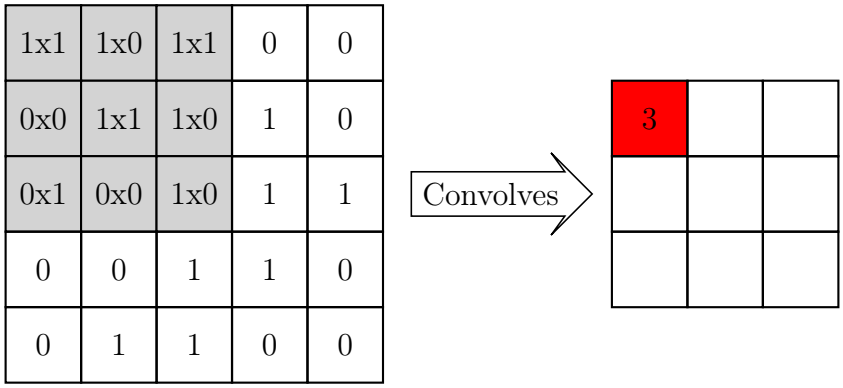


Figure 23: Example of the convolutional layer: filter/kernel convolves through the entire depth of the input grid, an activation map is created by element-wise matrix multiplication and summation of the results. Left: the input image volume with a 3×3 filter convolving over the image. Right: An activation map formed as a result of convolution.

2.2.2 Pooling Layer

The pooling layer performs a downsampling operation of the input volume along the spatial dimensions width and height, resulting in a lower-dimensional representation of the features in the activation map. There are multiple pooling strategies, such as max-pooling and average-pooling. Figure 24 shows an example of max-pooling applied on CNN. In max-pooling, the activation map is formed by taking the maximum value from each pooling window. Pooling window is an integer or tuple/list of 2 integers: pool height and pool width, and stride is an integer, specifying the strides of the pooling operation. For example, in Figure 24, after max-pooling, the 4×4 input

activation map is downsampled to a 2×2 activation map. The pooling operation takes as input the parameters such as the window size 2×2 and stride of size 2. The algorithm takes the maximum value from the specified window size and slides 2 pixels over to find the next value. In the first pooling window, 6 is the maximum pixel value. We eliminate all other values and form an activation map with just the maximum pixel values from the image volume.

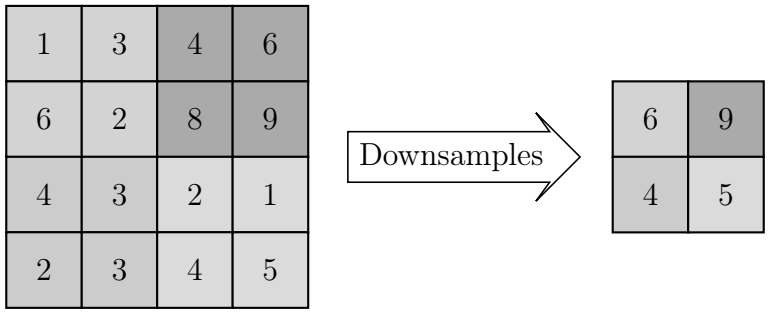


Figure 24: Example of the max-pooling layer, which downsamples the activation map by taking the maximum value in each pooling window. Left: Input volume with a pooling window size of 2×2, the maximum pixel value of input volume within the pooling window is used to form the activation map. Right: Downsampled activation map after pooling.

2.2.3 Fully-Connected Layer

The fully-connected layer is a feedforward neural network that receives the activation map from the pooling layer as input and converts the three-dimensional volume to one-dimensional vectors to be passed to the softmax activation function, which gives the probability of the possible class labels.

The CNN is a powerful model that has been proven to work well on image datasets (Krizhevsky et al. 2012). Randomly initialized CNN models have given high-performance on benchmark datasets such as MNIST (Van der Maaten 2009). CNNs are capable of extracting highly meaningful statistical patterns in large-scale and high-dimensional datasets, which has led to breakthroughs in image, video, text, and sound recognition tasks (Krizhevsky et al. 2012). We intend to provide a generalization of the convolution operation that works on data from regular domains such as images, to data originating from an irregular domain such as graphs. The

next section describes the challenge of applying the convolution operation on graph datasets.

2.2.4 Why the Convolution Operation on Graphs is Challenging

GCNs are a class of graph neural networks that use the convolution operation, which is the core operation in the CNN model, to extract meaningful statistical patterns from graph-structured data. The convolution operation allows a model to leverage the structural information of graph data and capture the attribute representations to make accurate class label predictions. The convolution operation is easier on grid-structured image data. Applying this operation on graph-structured data is challenging, because graphs have highly complex topological structure, arbitrary size, and are also dynamic (Hamilton et al. 2017). We have overcome this challenge by proposing a GCN that uses a robust variant of the convolution operation.

The convolution operation tries to generate new attribute representations of each pixel in an input image by aggregating the attributes of neighboring pixels by convolving a filter across the height, width, and depth of the image. Here attributes mean the intensity of the pixels in the images. The pixel for which we try to find new representation by aggregating attributes of neighbors is called target pixel. The convolution is possible because the pixels of an image has a grid-like structure. The pixels are located adjacent to each other.

GCNs, in general target, to find a new representation of each vertex of the graph by aggregating the attributes of its neighbors. Our method considers each node in a graph as a pixel in an image. We aggregate the attributes of the target and neighboring vertices by aggregating the vertices in k-order proximity. For example, if $k = 2$, we aggregate the attributes of nodes in the second-order proximity of the target node. The right side of Figure 25 shows the target vertex marked in red. If $k = 1$, we aggregate the attributes in first-order proximity or immediate neighbors of the target vertex.

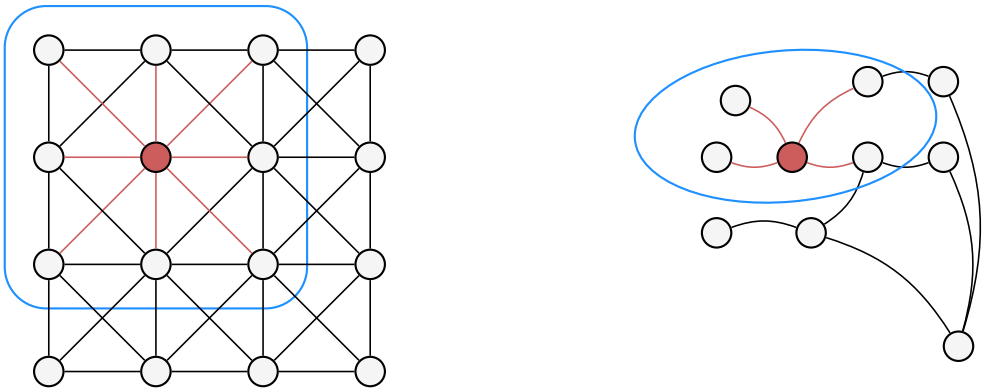


Figure 25: Analogous to CNN for images, AGCP considers each vertex as a pixel and aggregates the features of the target vertex and the neighbors of the target vertex.

The aggregation technique mentioned above is the central idea of the convolution operation in AGCP. We present a deep neural network architecture that leverages this idea to perform convolutions on graph data. We discuss the architecture of our proposed deep neural network in Chapter 4.

Chapter 3

Related Work

With the recent advancements in deep learning, many researchers are interested in exploring the application of deep learning on graph-structured data. Graph-structured data are generated from the non-Euclidean domain and have highly complex structural properties (Zhang et al. 2018). With an increase in real-world graph-structured data, it has become crucial to apply machine learning models that can take graph-structured data as input and then understand both the underlying features and the complex relationships represented in the graph in order to generate outputs such as predictions, classifications, regression, representation learning, clustering or recommendations.

Steered by the success of deep learning, researchers are trying to apply efficient neural network architectures such as convolutional networks, recurrent networks, and deep autoencoders in order to design a suitable neural network architecture that works well on graph-structured data. A significant amount of research is being done on node classification, edge prediction, graph classification, and label prediction for graphs using these deep learning architectures (Zhang et al. 2018).

In this research, we are interested in applying convolutional networks to graph-structured data. This class of neural networks is commonly known as a graph convolutional network or GCN. GCNs are inspired by the architecture of CNNs, which can exploit the shift-invariance, local connectivity, and compositionality of image data (Krizhevsky et al. 2012). As a result, CNNs can extract meaningful local attributes. The convolution operation is a critical operation in CNN, imitating this operation in a GCN is challenging owing to the complex structural representation of a graph.

In this chapter, we discuss some of the most recent works done by researchers in graph-structured data. The design of the convolution operation in GCNs falls into two major categories: spectral-based and spatial-based. Section 3.1 discusses the spectral-based, followed by the spatial-based approaches in Section 3.2, which is then followed by a brief introduction into structure-aware convolutional neural networks in Section 3.3. This chapter ends with a table that briefs all major researches done on GCNs.

3.1 Spectral-based Convolutions

The spectral-based approaches define graph convolutions by using filters from graph signal processing, where the function of the graph convolution operation is to remove noise from graph signals (Bruna et al. 2013). In spectral-based approaches, concepts from spectral graph theory are used to define convolutions in the context of graphs with Fourier analysis (Wu et al. 2019). Spectral graph theory defines the properties of a graph in relationship to the eigenvectors and eigenvalues of matrices associated with graphs. Here, we consider Laplacian and adjacency matrix. The adjacency matrix for both weighted and unweighted graphs has been defined in Chapter 1. The Laplacian matrix of graph is determined by $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of a graph G. Graph theory also provides graph clustering techniques to formulate the downsampling of graphs.

Spectral CNN exploits the concepts in spectral graph theory to define the convolution operation by taking the normalized graph Laplacian matrix, defined as, $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}$ is the identity matrix, $\mathbf{D}$ is a diagonal matrix of degrees (constructed of the degrees of each node), $\mathbf{A}$ is the adjacency matrix of undirected graph, and $\mathbf{D}_{ii} = \sum_j (a_{ij})$ where $\mathbf{D}_{ii}$ is the value at the i^{th} column of the i^{th} row of the degree matrix. As graph Laplacians are real positive semidefinite (Hermitian matrix whose Eigenvalues are non-negative) (Bruna et al. 2013), we can refactor $\mathbf{L}$ as $\mathbf{L} = \mathbf{U}\lambda\mathbf{U}^{\mathbf{T}}$, where $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1 \dots \mathbf{u}_n]$ is the matrix of eigenvectors ordered by eigenvalues and λ is the diagonal matrix of eigenvalues.

In graph signal processing, the signal or attribute or features on the vertex of an attributed graph is given by vector $\mathbf{x} \in \mathbb{R}^d$ where d is the number of attributes. The Fourier transform of a signal $\mathbf{x}$ on a graph $G = (V, E)$ is defined as $\mathscr{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$. The inverse of the graph Fourier transform is defined as $\mathscr{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ represents the resulting vertex signal after applying the Fourier transform. Thus, the graph convolution of the input signal $\mathbf{x}$ with a filter $\mathbf{g} \in \mathbb{R}^d$ is defined as:

$$\mathbf{x} *_G \mathbf{g} = \mathbf{U}\Big((\mathbf{U}^T\mathbf{x}) \odot (\mathbf{U}^T\mathbf{g})\Big), \tag{1}$$

where $\odot$ denotes the Hadamard product, and $*_G$ represents the convolution operation on the graph. Equation (1) represents the Hadamard product of the signal and the filter (Defferrard et al. 2016). Spectral-based graph convolutions all follow the definition where a graph convolution is simplified as $\mathbf{g}_\theta = diag(\mathbf{U}^T\mathbf{g})$. The key difference is in the choice of the filter.

The first spectral CNN was proposed by (Bruna et al. 2013). They assumed that the filter is a learnable parameter. Their definition of a graph convolution layer is given by the following equation:

$$\mathbf{X}^{l+1} = \theta * \mathbf{X}^l = \mathbf{U}diag(\theta)\mathbf{U}^T\mathbf{X}^l, \tag{2}$$

where $\mathbf{X}^{l+1}$ and $\mathbf{X}^l$ are the node attributes at layers $l+1$ and l, respectively.

We present some important research done in spectral convolutions in the following sections. We discuss the aim, contributions, techniques proposed, experiments, results, and conclusions for each work.

3.1.1 Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Aim: The main aim of research done by (Defferrard et al. 2016) is to generalize the convolution and pooling operations in grid-like data to graph-like data.

Contribution: The authors proposed strictly localized spectral filters in their re-

search paper. They also presented an effective pooling strategy that arranges the vertices as a binary tree based on a Graclus multilevel clustering algorithm in the graph coarsening phase. In this research, the authors tried to overcome two significant shortcomings of filters defined in the spectral domain, which are:

1. It is not naturally localized, and
2. Convolutions are costly with a time complexity of $O(n^2)$, where n is the number of nodes in the input graph for the multiplication operation.

Proposed Technique: The authors proposed the following technique in their work:

1. Spectral filtering of graph signals: The definition of the convolution operation in the attribute domain is difficult. This research defined a convolution operation in the Fourier domain, denoted by $*_G$ as per Equation (1).
2. Polynomial parametrization for localized filters: The authors proposed a parameterized polynomial filter to make convolutions easier on a graph.
3. Graph Coarsening using fast pooling: Graph coarsening is a technique that consists of clustering the vertices of a graph that has similar attributes. The authors solved the multi-layer coarsening algorithm, where each layer of the neural network produces a coarsened graph using the Graclus multilevel clustering algorithm. The Graclus' algorithm is a greedy rule which consists, at each coarsening level, in marking an unmarked vertex i and matching it with one of its unmarked neighbors j that maximizes the local normalized cut $W_{ij}(1/d_i+1/d_j)$, where W_{ij} is the edge weight between the vertices v_i and v_j and $1/d_i$ and $1/d_j$ is the degree of v_i and v_j respectively.

Experiments and results: The authors applied the proposed graph neural network to data from both Euclidean and non-Euclidean spaces. They applied the model on the following datasets:

1. MNIST dataset: MNIST dataset has 70,000 digits represented on a 2D grid of size $28*28$. They compared the proposed model with classic CNN and observed

that the performance of the proposed model is par with classic CNN, and the slight accuracy drop can be accounted for by the isotropic nature of the spectral filters.

2. 20NEWS dataset: 20NEWS dataset is used for the text categorization task. The authors constructed a graph with $n = |V| = 10,000$ nodes and $|E| = 132,834$ edges from the unstructured text dataset. The authors observed that while the experiment results did not outperform the multinomial naive Bayes classifier on this small dataset, it does defeat fully connected networks.

Conclusions: The authors were able to reduce the computational complexity to $O(n)$ based on the number of nodes in the graph. They also found that the quality of the graph data is vital for the classification task. The authors stated that as part of future work, they would like to explore the possibility of applying their model to graph-structured data rather than grid-structured data.

3.1.2 Semi-Supervised Classification With Graph Convolutional Networks (GCN)

Aim: In the research done by (Kipf & Welling 2016), the authors tried to accomplish vertex classification on graphs with some vertex labels known and some unknown using spectral approaches. This problem is a semi-supervised learning task on a graph with labeled nodes. The motivation of the convolution operation in this research is a first-order approximation of spectral graph convolutions.

Contribution: This research contributed to graph convolutional neural networks by proposing a first-order approximation in the Fourier-domain to obtain an efficient linear-time graph. The contributions made by authors are given below:

1. Firstly, the authors introduced a simple layer-wise propagation rule for the graph neural network models, which is motivated from a first-order approximation of spectral graph convolutions.

2. Secondly, the authors demonstrate how the graph-based neural network model is

used for faster semi-supervised classification of nodes with a much more scalable proposed model.

Proposed Technique: The authors proposed the following technique in their research paper.

1. Attributed graph: The input to this model is a dataset of attributed graphs defined in Section 1.1.2.

2. Propagation rule: The propagation rule computes the attribute representation of a node as an aggregate of the attribute representations of its neighbors. Every neural network layer can then be written as a non-linear function, $H^{l+1} = f(H^l, \mathbf{A})$ and $f(H^l, \mathbf{A}) = \sigma(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}H^{(l)}\mathbf{W}^{(l)})$, where $f(H^l, \mathbf{A})$ represents a hidden layer at level one, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the symmetric adjacency matrix $\mathbf{A}$, σ is the activation function (ReLU in this case), $\mathbf{W}$ is the weight matrix, and $\mathbf{I}$ is the identity matrix. The degree matrix $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$ and is normalized to prevent exploding or vanishing gradients. The output $\mathbf{Z}$ of the model (forward propagation) is thus given by:

$$\mathbf{Z} = f(X, \mathbf{A}) = softmax\Big(\check{\mathbf{A}}\, ReLU(\check{\mathbf{A}}\, \mathbf{X}\, \mathbf{W}^{(l)})\, \mathbf{W}^{(l+1)}\Big), \tag{3}$$

where $\check{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ and $\mathbf{W}^l$ and $\mathbf{W}^{(l+1)}$ are the weights of a neural network at the l^{th} and $(l+1)^{th}$ levels respectively.

Experiments: The authors applied the proposed GCN to graph data from different domains. They applied the model on Citeseer, Cora, Pubmed (Sen et al. 2008), and NELL (Carlson et al. 2010).

1. Network architecture: The authors trained a two-layer as well as ten-layer GCN on a graph data set of 1,000 examples, out of which 500 were labeled. They trained the neural network using a dropout rate for all layers, L2 regularization factor for the first GCN layer, and a number of hidden layers. The model was trained for 200 epochs using the Adam optimizer (Kingma & Ba 2014) with a

learning rate of 0.01, and early stopping with a window size of ten units. The model was applied to the data to obtain results and was compared with the state-of-the-art baseline models.

2. Baseline: The authors compared the proposed model with six other previously proposed methods in graph neural networks: ManiReg (Manifold Regularization) (Belkin et al. 2006), SemiEmb (Semi-supervised embedding) (Weston et al. 2012), LP (Label Propagation) (Zhu et al. 2003), DeepWalk (Perozzi et al. 2014), ICA (iterative classification algorithm) (Lu & Getoor 2003), and Planetoid (Yang et al. 2016). The classification accuracy was compared, and the results showed that the proposed model (Kipf & Welling 2016) performed better.

Conclusions: The authors experimentally proved that the propagation of feature information from neighboring nodes in every layer improves classification performance. We leverage this finding in our approach.

3.2 Spatial-based Convolutions

Borrowing the idea of the convolution operation of a CNN on an image, spatial-based methods define graph convolutions based on a node's spatial relations. In the convolution operation, a node's attributes are represented by aggregating the attributes of that node as well as it's neighbors. Graph pooling modules can be stacked with the GCN layer, to coarsen the graphs into high-level sub-graphs which capture the attributes of the original input graph. These models are generally embedding the node attributes to a higher-dimensional space where the nodes with similar attributes are projected to close proximity; this is called *node embedding*.

3.2.1 Inductive Representation Learning on Large Graphs

Aim: In the research done by (Hamilton et al. 2017), the authors tried to generate the representation of nodes attributes by aggregating the features of neighboring nodes.

They proposed GraphSAGE, which is an inductive framework that generates the node embeddings for graphs that are unseen during the time of training.

Contribution: The research done by the authors is an important contribution in spatial-based graph convolutional neural networks as the research proposed a generic inductive framework. The significant contributions made by the authors are the following:

1. Firstly, the authors introduced GraphSAGE, a learning algorithm that incorporates the node attributes as well as the topology of the nodes (e.g., node degrees) to form the embedding of the nodes.

2. Secondly, the authors introduced a new mechanism for graph training; instead of training the network to find the embedding of the node, the model is trained to find the aggregator function. An aggregator function samples the features of the node's k-hop neighborhood. The model uses the aggregated node feature to make the final predictions and to calculate the backpropagation error.

Proposed Technique: The authors proposed the following techniques in their research.

1. Embedding generation (forward propagation) algorithm: The inputs to these models are attributed graphs, which were mentioned in Section 1.1.2. Suppose that there is a target node v_i, the aim here is to either predict the node's class label or to find the embedding of the target node. To achieve this aim, the authors are not just using the features of the target node, but also the features of the neighboring nodes. This is achieved by aggregating the node features and transforming the aggregated features to the target node. The obtained target representation of each node is then passed on to the next level, as follows:

$$h_P^{k+1} = ReLU\left(W^k\, h_P^k, \sum_{n \in N(P)} \left(ReLU\left(Q^k\, h_n^{(k)}\right)\right)\right), \tag{4}$$

where h_P^{k+1} is the $(k+1)^{th}$ level feature of node P. ReLU is the non-linear activation function, explained in Section 2.1.3, $W^k h_P^k$ represents the transform of P's

own features from level k. $\sum(.)$ is the aggregator function and $(ReLU\,(Q^k\,h_n^{(k)}))$ represents the transform and aggregate functions of the n neighbors of node P. The authors used three aggregator functions, namely mean aggregator, LSTM aggregator, and pooling aggregator. The mean aggregator function computes the elementwise mean of the vector in each level. The LSTM aggregator uses an LSTM architecture to find the aggregation of the node attributes. In the pooling aggregator, each neighbor's attribute vector is independently fed through a fully-connected neural network; following this, an elementwise max-pooling operation is applied to aggregate information across the neighbor set.

2. Learning the parameters of GraphSAGE: The authors use stochastic gradient descent (Bottou 2010) to learn predictive representations. The weight matrix W is tuned by applying the graph-based loss functions such as unsupervised loss, neighborhood sampling, and minibatch optimization (Li et al. 2014).

Experiments:

1. Network architecture: The authors did a fair implementation of the baseline models; all models share an identical implementation of minibatch iterators, a loss function, and a neighborhood sampler.

2. Baseline: The authors proposed three variants of GraphSAGE: a mean aggregator, an LSTM aggregator, and a pooling aggregator. They also proposed an extended inductive version of GraphSAGE with a semi-supervised GCN (Kipf & Welling 2016) termed as GraphSAGE-GCN. These models were compared to DeepWalk (Perozzi et al. 2014), a logistic regression feature-based classifier that ignores graph structure, and a random classifier. The authors used evolving graphs, which constantly add unseen data to the growing graphs. They implemented the models on Reddit to perform community detection on and citation data derived from Thomson Reuters Web of Science Core Collection.

Conclusions: The authors introduced a novel inductive approach to generate the embedding of a graph efficiently. This method outperforms the baseline models by a significant margin.

3.2.2 An End-to-End Deep Learning Architecture for Graph Classification (DGCN)

Aim: In the research done by (Zhang et al. 2018), the authors proposed a GCN that imitates the classic end-to-end CNN architecture for fixed graph-structured data. The input is graph-structured data that is used to perform graph classification.
Contribution: This research proposed a novel neural network architecture that works very well on graph-structured data. The contributions are given below:

1. Firstly, the authors are trying to solve the difficult problem of extracting the hidden feature information from the graph for classification problems by introducing the localized graph convolutions.

2. Secondly, the authors introduced a novel SortPooling layer which sorts graph vertices in a logical order so that traditional neural networks can be trained on the graphs.

Proposed Technique: The authors proposed the following technique in this research.

1. Graph convolution layers: The inputs to these models are undirected graphs. The graph convolution operations are similar to those given in Equation (1) (Kipf & Welling 2016) and are divided into four major steps:

   (a) First, a linear feature transformation is applied to the node information matrix by mapping the c feature channels to $\acute{c}$ channels in the next layer by applying spectral filters with first-order approximations.

   (b) In the second step, the propagation of node information of the neighboring nodes and the node itself.

   (c) The third step is to multiply the propagated features with the normalized degree matrix of the adjacency matrix.

   (d) The fourth step is to apply a non-linear activation function to obtain the output.

2. Graph SortPooling: The primary function of the SortPooling layer is to sort the feature descriptors, each of which represents a vertex, in a logical order before feeding them into the traditional 1-D convolutional and dense layers.

Experiments:

1. Dataset: The datasets are: MUTAG, PTC, NCI1, PROTEINS, and D&D from (Kersting et al. 2016).

2. Network architecture: The architecture of DGCN has 16 output channels followed by a pooling layer. The second 1-D convolutional layer has 32 output channels. The hidden layer has 128 hidden units with a dropout rate of .50 and is followed by the softmax output layer. The activation function used is the Tanh function for the convolution layer and ReLU for all other layers. Stochastic gradient descent (SGD) with the Adam updating rule is used for the optimization.

3. Baseline: The authors compared DGCN with four other graph kernel algorithms. The graphlet kernel (GK) (Shervashidze et al. 2009), the random walk kernel (RW) (Vishwanathan et al. 2010), the propagation kernel (PK) (Neumann et al. 2016), and the Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al. 2011).

   The authors were able to achieve highly competitive results in comparison to graph kernels, achieving the highest accuracies on the MUTAG, PROTEINS, and D&D datasets indicting that DGCN can utilize the node and structural information efficiently. For the NCI1 dataset, DGCN falls behind WL-kernel, and for the PTC dataset, DGCN falls behind the propagation kernel.

Conclusions:

1. DGCN accepts graph-structured data as input without the need to first transform the data to tensors, making gradient-based training efficient.

2. DGCN proposed a novel SortPooling layer which enables learning from input graph topology by sorting vertex features instead of summing them.

3. This model achieves better performance than existing methods on datasets as discussed above.

3.3 Structure-Aware Convolutional Neural Networks

Structure-Aware convolutional neural networks are a class of neural network architecture that works on both Euclidean or grid-structured (e.g., images), and with non-Euclidean or graph-structured (e.g., social networks) data. The convolution operation of these neural networks is designed such that it can work with data with diverse topological structures. In (Chang et al. 2018), the authors modeled the local structure information into the generalized filters to achieve the structure-aware convolutions.

In Table 1, we summarize the contributions made by other researchers in the application of deep learning to graph-structured data.

Table 1: Summary of previous works

Year	**Paper Title**	**Authors**	**Major Contribution**
2011	Weisfeiler-Lehman graph kernels. Journal of Machine Learning Research (Shervashidze et al. 2011)	Nino Shervashidze, Pascal Schweitzer, Erik Jan van, Leeuwen Kurt, Mehlhorn Karsten M. Borgwardt	This paper makes kernel machines such as SVMs feasible for graph classification by computing graph similarity measures. Dataset used: MUTAG, PTC, NCI1, PROTEIN, and D&D.

2018	Graph Capsule Convolutional Neural Networks (Verma & Zhang 2018)	Saurabh Verma & Zhi-Li Zhang	This paper proposed graph capsules, which encapsulate more information about nodes in a local neighborhood in a small vector in place of scalar output. Datasets used: PTC, PROTEINS, NCI1, NCI109, D&D, and ENZYMES.
2018	An End-to-End Deep Learning Architecture for Graph Classification (Zhang et al. 2018)	Muhan Zhang, Zhicheng Cui, Marion Neumann & Yixin Chen	This research proposed: 1. Localized Graph kernels to perform convolutions. 2. A SortPooling layer which sorts graph vertices in a consistent order so that traditional neural networks can be trained on the graphs. Datasets used: MUTAG, PTC, NCI1, PROTEINS, and D&D.

2016	Learning convolutional neural networks for graphs (Niepert et al. 2016)	Mathias Niepert, Mohamed Ahmed, & Konstantin Kutzkov	This research proposed: 1. A technique to generate a graph neighborhood by creating a sequence of nodes. 2. A unique mapping from the graph representation to a vector representation such that nodes with similar structural roles in the neighborhood graphs are positioned similarly in the vector representation. Datasets used: MUTAG, PTC, NCI1, PROTEIN, and D & D.
2014	DeepWalk: Online Learning of Social Representations (Perozzi et al. 2014)	Bryan Perozzi, Rami Al-Rfou & Steven Skiena	Proposed node search using uniform random walks. The obvious limitation of such a strategy is that it gives us no control over the explored neighborhoods. Datasets used: BlogCatalog, Flickr, and YouTube.

2015	LINE: Large-scale Information Network Embedding (Tang et al. 2015)	Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan & Qiaozhu Mei	Proposed a breadth-first strategy, sampling nodes primarily and optimizing the likelihood independently over only 1-hop and 2-hop neighbors. The effect of such an exploration is easier to characterize, but it is restrictive and provides no flexibility in exploring nodes at further depths.
2016	node2vec: Scalable Feature Learning for Networks (Grover & Leskovec 2016)	Aditya Grover & Jure Leskovec	They proposed a framework for continuous feature representations for the nodes in networks by a flexible and controllable way to explore the neighborhood through parameters p and q.
2017	HARP: Hierarchical Representation Learning for Networks (Chen et al. 2018)	Haochen Chen, Bryan Perozzi, Yifan Hu & Steven Skiena	In this approach, a graph coarsening procedure is used to collapse related nodes in the graph together into "supernodes," and then DeepWalk, node2vec, and LINE is run on this coarsened graph.

2018	Towards Gene Expression Convolutions using Gene Interaction Graphs (Dutil et al. 2018)	Francis Dutil, Joseph Paul Cohen, Martin Weiss, Georgy Derevyanko & Yoshua Bengio	This paper experimentally found that gene expression predictions can be done using graph convolutional network with gene interaction graph as input and perform better than multi-layer perceptron or logistic regression models.

3.4 Conclusion

We discussed some of the major contributions made by other researchers in the application of deep neural networks in graph-structured data. Most of the literature discussed above ((Kipf & Welling 2016), (Bruna et al. 2013) and (Defferrard et al. 2016)) is designed for vertex classification. The research done by (Zhang et al. 2018) is a graph classification approach, but does not work well on attributed, weighted graphs. The spectral-based methods ((Kipf & Welling 2016), (Bruna et al. 2013) and (Defferrard et al. 2016)) use matrix factorization and Eigen decomposition in their convolution strategy, which are costly operations and requires additional computation. The spatial-based approaches are computationally in-expensive compared to spectral-based approaches but do not support any pooling strategy. Thus, these models are memory inefficient, slow, and non-parallelizable approaches. The key difference from the approaches discussed above and our approach is that we treat graphs as images with fixed topology and apply convolution operation with learnable filters to understand the statistical patterns hidden in the graph data. Another key innovation is that we introduce a pooling layer that coarsens the graph by statistically measuring the information gain of a vertex.

Chapter 4

Proposed Graph Convolutional Neural Network

In this chapter, we present a novel graph convolutional neural network architecture with a pooling layer called attributed graph convolutional neural network with pooling (AGCP). This architecture takes a weighted, labeled, attributed graph with arbitrary size and fixed topology as input and predicts the "class labels" as output. AGCP has an efficient convolution layer followed by a pooling layer that coarsens the graph by eliminating less important vertices while preserving the global structure of the graph. In this chapter, we first discuss the overall architecture of the proposed model, followed by a detailed description of the convolution, pooling, and the prediction generation layers. We further analyze the complexity of the proposed model, followed by evaluation techniques.

4.1 Pipeline of the Proposed Model

The pipeline of the proposed model is depicted in Figure 26, there are two main modules which make up the architecture of AGCP. The first module is a data handler that collates input data from different sources and pre-processes the data to be passed on to the AGCP module. The AGCP module takes the input graphs and assigns them to the corresponding class labels. ENZYME and D&D are pre-processed weighted datasets that we acquired from (Kersting et al. 2016). We designed the data handler module to generate three variations of GINA.

Graph data is the data of vertex and edge information of the graph, and vertex attributes usually contain the attributes of each vertex. We then embed the attribute vector to the vertices of the graph. We pre-process both graph data, and vertex attributes to convert the data to real values instead of categorical and text variables. We pass the weighted attributed graph dataset to the AGCP module. The AGCP module receives the weighted, attributed graph dataset as input and learns the labeled graph data and vertex attribute distributions to classify the labels of unseen graphs. The AGCP module contains a series of convolution layer followed by a pooling layer based on the depth of the proposed neural network specified, followed by a prediction layer. The prediction layer predicts the probability of input belonging to the corresponding class labels based on the classification problem.

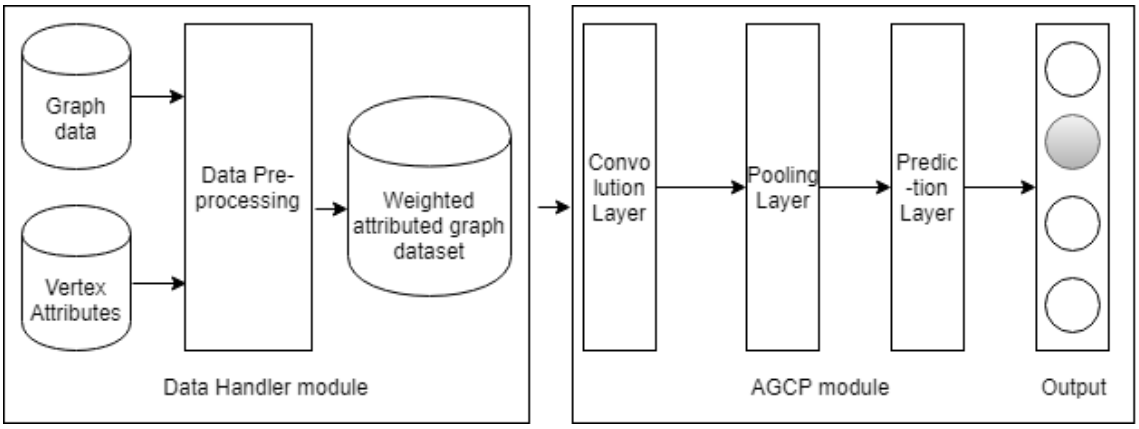


Figure 26: Overall pipeline of the proposed model.

4.1.1 Data Handler Module

The data handler module was designed to generate three variations of GINA. This module collates the data from different sources and pre-processes the data to prepare it for further analysis by the AGCP model. As shown in Figure 26, the data handler module generates the weighted, attributed graph dataset from the data sources. Further, this module assigns weights to the edges connecting the vertices of the graph and embeds the attributes to the vertices of the graph. We identified two primary data sources in the data handler module:

1. Graph data: The graph data is a dataset of graphs that have a finite set of

vertices and edges connecting the vertices. The edges connecting the vertices have weights associated with them, which determine how strongly two vertices are connected. In this thesis, to generate GINA, we consider the gene interaction network of patients diagnosed with prostate cancer. We acquired the graph data from GeneMANIA Cytoscape plugin (Warde-Farley et al. 2010).

2. Vertex attributes: The vertex attributes are a finite set of correlated attributes that describe the attributes of the data and help us predict the class labels better. Modern-day graphs have attributes embedded in the vertices of the graph. Each graph represents a patient gene interaction network. The vertices of the gene interaction network are genes. The attributes of genes are clinical features of patients. We acquired the mutation data of patients from cBioPortal (*cBioPortal* 2012). We consider different types of mutations, for example, nonsense and missense. The attributes in the vertices are clinical features such as Reference_Allele, Tumor_Seq_Allele1, and Tumor_Seq_Allele2. For each variation of GINA, we tried a different set of attributes to compare the performance.

4.2 AGCP Module

The AGCP module contains the AGCP model, which is the core of our proposed architecture. This module takes a dataset of weighted, attributed graphs with labels as input. The graphs are of arbitrary sized with a fixed topology. We convolve a randomly initialized filter vector over the entire graph topology to define the new attribute representation of the graph. Then, we update the attribute representation of a vertex by aggregating the attributes of its k-order neighbors. We then take the average of all neighboring attributes and their corresponding weights. The new attribute representation of the graph contains the same number of vertices, but the attributes present on the vertices are the aggregated average of the k-order neighbors. The graph with new representation is then passed to an activation function to normalize the graph. Here we use the ReLU activation function. The output graph after ReLU activation contains the same number of vertices and edges as the original input

graph, but the attribute vectors on the vertices are different. The output graph might contain edges and vertices that are not relevant to the classification task. Thus, we pass the resulting output graph to the pooling layer. To downsample the graph, we introduce a pooling layer that eliminates the vertices and edges of the graph that are not statistically relevant to the classification task. The pooling layer preserves the global structural roles of the graph while reducing the dimensionality. We interleave multiple pooling layers and convolution layers until optimal performance is achieved. The number of layers is a design choice. Finally, we apply a prediction generation layer followed by a softmax activation function to classify the input graphs to corresponding class labels. We discuss the detailed working of the AGCP model in the upcoming section.

Algorithm 1: AGCP

We consider the problem of supervised learning on a dataset of the weighted, attributed graph. Our model has two phases, known as the training phase and the inference phase. During the training phase, the AGCP model finds the best parameters with the training dataset. The parameters to AGCP are the filter vector $\mathbf{F}$, number of filters f and the weight matrix $\mathbf{W}$. The best parameters are found by forward propagation step of the AGCP algorithm. The error in prediction is then propagated back to adjust the parameters until convergence. During the inference phase, the classification task of our model consists of predicting the labels of the graph for which the labels are unknown. Let the training dataset of graphs be $S_t = \{(G_1, y_1), (G_2, y_2), (G_3, y_3) \cdots (G_t, y_t)\}$, where t is the number of the training samples, G is weighted attributed graph, and $Y = \{y_1, y_2, y_3 \cdots y_l\}$ is the set of of l labels and the testing dataset is dataset of graphs without the labels. Let $G = (V, E, \mathbf{W}, \mathbf{X})$ be a graph for which the labels are unknown, the goal of AGCP is to derive a mapping function $f : G \mapsto Y$; which predicts the class label y_i for a given graph G. We accomplish this task by devising a convolution layer, represented by Convolution(), which updates the attribute representations of each vertex by aggregating the attributes of its k-order neighbors. The output of the convolutional layer

is a graph $G_{\mathbf{z}_v}$ with new vertex representations. The output from the convolutional layer is then passed on to the pooling layer, represented by Pooling(). In the pooling layer, the objective is to find a coarse graph $G_C = (V_C, E_C)$, where $V_C \subseteq V$ is the set of vertices of the coarser graph and $E_C \subseteq E$ is the set of edges in the coarser graph and $|V_C| << |V|$ and $|E_C| << |E|$. The prediction generation layer, represented by Predict() is a typical fully-connected layer with only a feed-forward neural network. The detailed working of the convolution layer is given in the upcoming section, followed by the pooling layer and prediction generation layer. Algorithm 1 depicts the working of the AGCP model.

Algorithm 1: AGCP(G)

Input : graph, G; vertex attribute matrix, $\mathbf{X} \in \mathbb{R}^{n \times d}$; number of layers, l

Output : label, y_i of the graph

Parameters: filter, $\mathbf{F}$; number of filters, f; weight matrices, $\mathbf{W}^l$ for layer l

```
for i = 1 to l do
    G^i_{z_v} = Convolution(G);
    G^i_C = Pooling(G^i_{z_v});
end
y_i ← Predict(G^l_C);
return y_i
```

4.2.1 Graph Convolution Layer

Graph convolution is the core operation of the AGCP and is executed in the convolution layers. The primary purpose of the convolution operation is to analyze the latent attributes of highly- complex graph data, and to find useful attribute representations to make accurate predictions efficiently (Henaff et al. 2015). In-depth analysis of graph data has shown that the vertices in close proximity have similar attributes and, thus, have the same class labels (Grover & Leskovec 2016). Our proposed graph convolutional neural network learns the best attribute representation of each vertex of the graph by aggregating the attributes of the neighboring vertices within the k-order.

The proposed AGCP model has four main steps in the convolution operation:

1. Linear attribute transformation ($Trans()$): The linear attribute transformation is an element-wise multiplication of randomly initialized weighted filter vector with the attribute vector of each vertex of the entire graph. This operation helps assign higher weights to more prevalent attribute vector present on the vertices of the graph and lower weights to the less prevalent vertices. Figure 27 shows an example of linear attribute transformation. The attribute vector representing transcript exon and reference allele of SYNE1 gene is $(28, 1)$ and $(3, 1)$ for SYNE2. When multiplying the attribute vector with the filter vector of $(3, 0)$, the attribute vector of gene SYNE2 is updated to $(9, 0)$ and $(84, 0)$ for SYNE1.

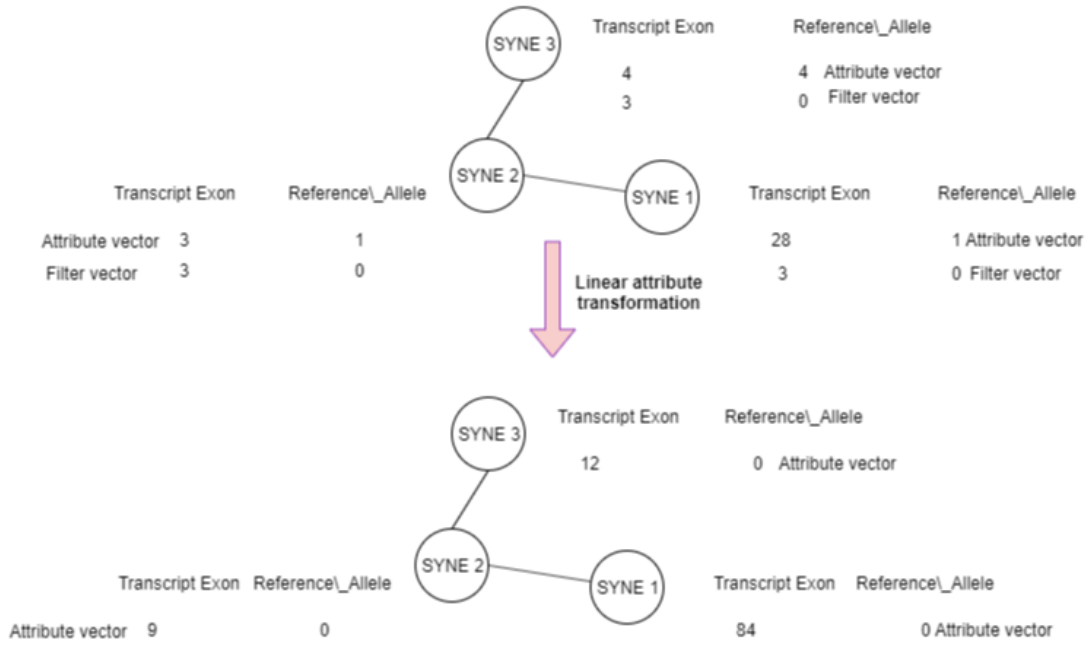


Figure 27: The new attribute representation of vertices after linear attribute transformation.

2. Node neighbor aggregation ($Aggregate()$): AGCP, updates the attribute representations of each vertex in a graph, by aggregating the attribute information of its neighbors in k-order proximity at each convolution layer. We define a variable k, also known as search depth. AGCP then aggregates the attributes

of the vertices in the k-order proximity of each vertex by taking the average of all the attributes descriptors of the neighboring vertices. For example, if $k = 1$, we consider the neighbors in first-order proximity of the vertices as explained in Definition 3.

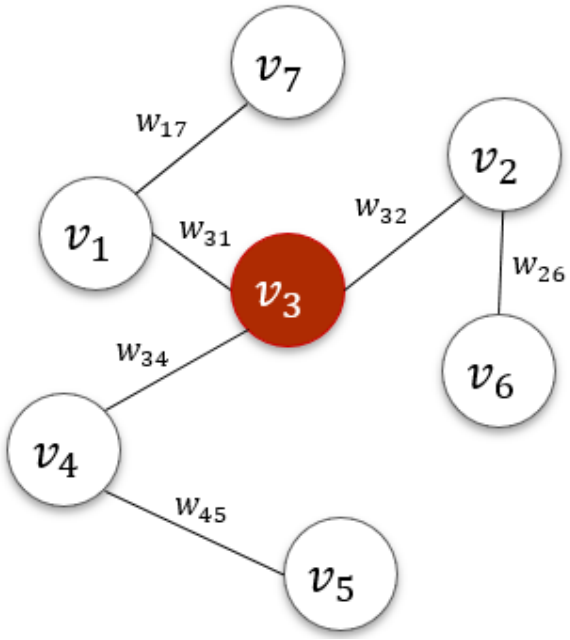


Figure 28: The new attribute representation of target vertex v_3 marked in red is obtained by aggregating the attributes of its 1-hop neighbors.

The aggregate function can be mathematically explained with the help of Figure 28 and Equation 1 as follows: As shown in Figure 28, let the target vertex be v_3. To aggregate the attribute vector of the vertex v_3; we consider the attributes of vertices within $k = 1$ proximity of v_3. Let the vertices within the 1-order proximity of vertex v_3 represented by $N_1(v_3)$ and $\mathbf{H}_{N_1(v_3)}$ represent the average aggregated attribute vector of target vertex v_3 and is given by:

$$\mathbf{H}_{N_1(v_3)} = \frac{\big((\mathbf{x}_{v_2} \times w_{32}) + (\mathbf{x}_{v_1} \times w_{31}) + (\mathbf{x}_{v_4} \times w_{34})\big)}{|N_1(v_3)|}, \tag{1}$$

3. Vertex information propagation ($Propagate()$): AGCP propagates the new attribute representations to the next layer of the neural network. These attribute representations then acts as the input graph to the next layer of AGCP.

4. Non-linear activation function ($\sigma()$): AGCP then applies a non-linear activation function to the graph propagated from the previous layer. In AGCP, we use the ReLU activation function.

Algorithm 2: Convolution

Algorithm 2 depicts the working of our convolution layer. The input to our convolutional layer is a weighted, attributed graph G; the input attribute vector of all vertices $(\mathbf{x}_v, \forall v \in V)$; the number of layers l; and the search depth k. The output is a graph with new vertex representations $(G_{\mathbf{z}_v}, \forall v \in V)$. The outer loop of the algorithm shows the current convolution layer. For k, find the neighboring vertices, $u \in N_k(v), \forall v \in V$. At each step in the convolution layer, H^i represents the attribute vector of vertices at the current layer l. At each layer, for each vertex $(v, \forall v \in V)$ in the graph G, the algorithm aggregates the attribute representations of the vertices in its immediate neighborhood, $\{\mathbf{H}_u^{i-1}, \forall u \in N_k(v)\}$, into a single vector $\mathbf{H}_{N_k(v)}^i$. Note that this aggregation step depends on the representations generated at the previous iteration of the outer loop, $i-1$, and base case with $i = 0$. At $i = 0$, the input vertex features are represented by vector $\mathbf{H}_v^0$. After aggregating the neighboring feature vectors, AGCP then concatenates the vertices's current representation, $\mathbf{H}_v^{i-1}$, with the aggregated neighborhood vector, $\mathbf{H}_{N_k(v)}^i$, given by the function Concatenate(). The concatenated vector is then fed through a nonlinear activation function σ. In the convolution layer, AGCP propagates the output representations to the next step of the algorithm $\mathbf{H}_v^i$. We denote the final representations output at layer l as $\mathbf{z}_v = \mathbf{H}_v^l, \forall v \in V$. The graph with new representations $G_{\mathbf{z}_v}$ is the output of the convolution layer. In order to learn useful parameters such as weight matrices $\mathbf{W}^l$, $\forall \in 1, ..., l$, filter vector $\mathbf{F}$, and, number of filters f, we apply a graph-based loss function to the output representations. We tune AGCP via stochastic gradient descent to find the best parameters. The graph-based loss function encourages nearby vertices to have similar representations.

Algorithm 2: Convolution(G)

Input : graph, G; number of layers, l; search depth, k

Output : graph with new vector representations, $(G_{\mathbf{z}_v}, \forall v \in V)$

Parameters: filter, $\mathbf{F}$; number of filters, f; weight matrices, $\mathbf{W}^l$ for layer l

$\boldsymbol{H}_v^0 \leftarrow Trans(\boldsymbol{x}_v), \forall v \in V$;
for $i = 1$ *to* l **do**
  for $v \in V$ **do**
    $\mathbf{H}^i_{N_k(v)} \leftarrow Aggregate(\{\mathbf{H}_u^{i-1}, \forall u \in N_k(v)\})$
    $\mathbf{H}_v^i \leftarrow \sigma\left(\mathbf{W}^i.Concatenate(\mathbf{H}_v^{i-1}, \mathbf{H}^i_{N_k(v)})\right)$
  end
end
$\mathbf{z}_v \leftarrow \mathbf{H}_v^l, \forall v \in V$;
return $G_{\mathbf{z}_v}$

4.2.2 Graph Pooling Layer

Downsampling is very important in graph analysis ((Vaishnav & Tatu 2016) and (Nguyen & Do 2014)). We devised a pooling layer in AGCP, which consists of statistically selecting the genes that aid in the classification of cancer type. The main function of this layer is to coarsen the graph to produce a subgraph which preserves the global graph structural information at a different scale. One of the new features of the AGCP architecture is the pooling layer that consists of statistically eliminating the vertices. We calculate the information gain of the attribute vectors of each vertex in a graph, and then set a threshold. If the information gain of a vertex is less than the threshold, we eliminate that vertex. The threshold information gain is set by taking the average information gain of all vertices of an input graph.

The inspiration for our pooling layer comes from information theory. In information theory, there are two measures of information called entropy and information gain (Asim et al. 2018). Entropy is a common way to measure impurities in a given set of samples. Here, impurities refer to attribute vectors that do not contribute to

the prediction of class labels.

The focus is to find the vertices whose attributes in a given set of training graphs are the most useful for discriminating between class labels. The information gain of a vertex helps us determine the most relevant vertices that contribute to the prediction of class labels, as information gain is the information contained in a vertex. Vertices that can accurately classify the graph to either of the possible class labels have maximal information. Given a graph $G = (V, E)$, let $V = \{v_1, v_2, \cdots, v_n\}$ be the set of vertices in graph G and $Y = \{y_1, y_2, \cdots, y_l\}$ be the set of class labels. Then, the information gain $IG(v)$ of a vertex is given by:

$$IG(v) = -\sum_{Y} P(y_i) \log P(y_i) + \sum_{Y} P(y_i|v) \log P(y_i|v)\,, \tag{2}$$

where $P(y_i)$ is the probability of label y_i (Azhagusundari & Thanamani 2013).

Algorithm 3: Pooling

Algorithm 3 depicts the working of our pooling layer. The objective of the pooling layer is to produce a coarser graph $G_C = (V_C, E_C)$ from $G_{\mathbf{z}_v} = (V_{\mathbf{z}_v}, E)$, where $V_C \subseteq V_{\mathbf{z}_v}$ is the set of vertices of the coarser graph and $E_C \subseteq E$ is the set of edges in the coarser graph. Also, the number of vertices and edges in the coarsened graph is less than the number of vertices and edges in the original input graph. That is, $|V_C| << |V_{\mathbf{z}_v}|$ and $|E_C| << |E|$. $G_{\mathbf{z}_v}$ is the input graph to the pooling layer with the new representations from the convolution layer.

Algorithm 3: Pooling($G_{\mathbf{z}_v}$)

Input : input graph, $G_{\mathbf{z}_v} = (V_{\mathbf{z}_v}, E)$ with new attribute representations from the graph convolution layer

Output: Coarser graph, $G_C = (V_C, E_C)$, where, $|V_C| << |V_{\mathbf{z}_v}|$ and $|E_C| << |E|$

```
Threshold = Average(IG of the input graph);
for v ∈ V_{z_v} do
    if IG(v) < Threshold then
        prune v and all the edges connected to v;
    else
        G_C ← G_{z_v}
    end
end
return G_C
```

4.2.3 Prediction Generation Layer

The prediction generation layer is a typical fully-connected layer with only a feed-forward neural network. The input to this layer is a graph G_C with a compact representation of the attributes from the previous layer. The prediction generation layer predicts the class label from the compact representation given by: $G_C \mapsto y_i$. This layer is followed by the Softmax activation function as AGCP allows both binary-class and multi-class classification.

4.2.4 Forwardpropagation and Backpropagation in AGCP

Figure 29 shows the architecture of the AGCP; we use this figure to describe forward propagation in AGCP. AGCP accepts a graph G and produces an output label $\hat{y}$, since AGCP learns the parameter weights of the filter function, which helps in labeling predictions by minimizing the prediction error or cost (Chen 1990). We call this forward propagation, "learning," or "training" in machine learning. The prediction error is then propagated back to adjust the parameters of the AGCP to minimize the

cost or error.

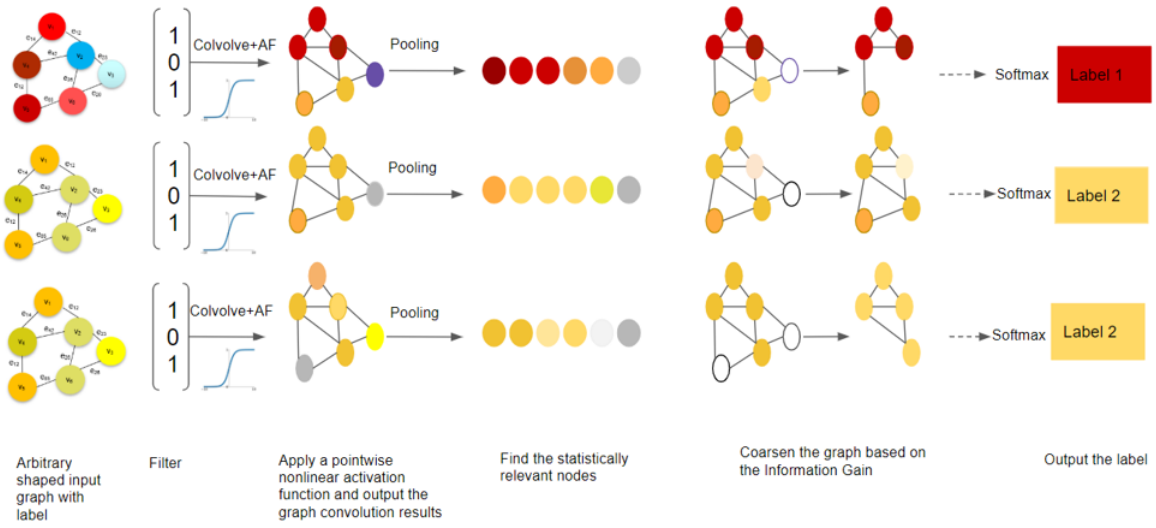


Figure 29: Forward propagation in AGCP.

Input Layer: The input layer in AGCP takes weighted, attributed graphs with fixed topology and arbitrary size as input. The data from the input layer are passed on to the convolution layer.

Convolution Layer: The convolution layer in AGCP receives the graphs from the input layer. The graphs are analyzed, and a randomly initialized filter function is applied to form a new attribute representation. Aggregating the attributes of the neighboring vertices of the original input graph helps the model to recognize and distinguish the low-level attributes. In Figure 29, we consider the first input graph, which has a class label 1. After the first convolutional layer, the vertices of the output graph have a new representation. AGCP has multiple convolution layers. As the network goes more in-depth, the convolution layers tries to learn the high-level attributes of the graph.

The model then applies an activation function to the output from the convolution layer. In this case, we use the ReLU activation function. The output graphs with new representations are passed to the pooling layer.

Pooling Layer: The pooling layer takes the graph with the new representations as input. The model finds the threshold information gain value for each graph and finds the information gain for each vertex. If the information gain of any given vertex is less than the threshold value, the model eliminates that vertex, which reduces the dimensionality of the graph to a different scale. The coarsened sub-graph represents the global attributes of the original input graph at a different scale.

We interleave multiple convolution layers and pooling layers to analyze both low-level and high-level graph attributes. The number of layers used is a parameter to the neural network design. AGCP performs well with three convolution and pooling layers. After three convolution and pooling layers, the coarse subgraph, which has high-level attribute representation, is passed on to the prediction generation layer.

Back Propagation: AGCP accepts an input graph G and produces the probability score for the output label $\hat{y}$. The predicted probability score is then used to calculate the error or cost of classification is calculated using Equation 1. The error is then backpropagated to update the model parameters such as filter vector, number of filters, and the weight matrix.

4.3 Complexity Analysis of AGCP

The complexity of an algorithm is the measure of the amount of time and/or space needed to produce the desired output, as a funciton of the size of the input. AGCP has two essential phases: training and inference. Additionally, AGCP has back-propagation using SGD, batch normalization, dropout, and classification using the sigmoid function. In this section, we explain the complexity of the training and inference phases of AGCP. We report the worst-case complexity by order of input size and model parameters using the big-O notation. Traditional deep learning models perform millions of matrix multiplications, convolutions, multiplications, and summations, which are computationally very expensive. AGCP has a set of parameters that affect the complexity of training:

1. The number of the input graphs, t.
2. The number of nodes in the input graph, n.
3. The number of edges in the input graph, m.
4. The number of attributes, d.
5. The number of layers in AGCP; l is a constant.
6. The number of neurons in each layer of AGCP; p is a constant.
7. The number of filters used in the convolution layer of AGCP; f is a constant.
8. The number of epochs for AGCP; e which can be considered as a constant.
9. The activation function used after the convolution layer; a is a constant.

The performance of the AGCP algorithm grows linearly with the number of input graphs and the nodes and edges of that graph. The other parameters can be considered as constant and thus not affect the complexity vastly. In big-O terms, the worst-case running time of AGCP can be notated as $\mathcal{O}(t \times n \times m)$. We discuss the training time of our model on GPU and CPU in Section 5.4.

Experimental studies have shown that additional computations such as batch normalization, dropout, and classification using the sigmoid function take up only 5-10% of the total training time (Zhang et al. 2018). Once the model is trained, and all the weights are learned, the complexity of inference is $\mathcal{O}(t \times n \times m)$. The space complexity of the model is linear to the number and size of the input samples to be stored, $\mathcal{O}(t)$.

4.4 Evaluation of the Proposed Model

We evaluate the proposed model using different evaluation methods and compared the results with previously published benchmark results. We used three metrics for assessing the accuracy of disease state prediction: ROC, AUC, and Accuracy ((Bishop

1995) and (Goodfellow et al. 2016)). The model evaluation methods used are discussed below with the examples using GINA.

4.4.1 Classification Accuracy

Accuracy is the most crucial metric of model evaluation in this thesis. AGCP supports both binary and multi-class classification problems. In the case of binary-classification, we can generally categorize the classification into four groups. A true positive (TP) represents the predicted class label where the actual class label is predicted correctly as an aggressive cancer type. A false-positive (FP) represents a prediction which incorrectly predicted an aggressive cancer type instead of non-aggressive. A false negative (FN) represents a prediction of an aggressive cancer type as non-aggressive. A true negative (TN) represents a non-aggressive cancer type predicted as non-aggressive.

Table 2: Categories of prediction for GINA.

		Actual disease state	
		Aggressive	Non-Aggressive
Predicted disease state	Aggressive	True-Positive (TP)	False-Positive (FP)
	Non-Aggressive	False-Negative (FN)	True-Negative (TN)

1. Classification Accuracy: Classification accuracy is the ratio of correct predictions to the total number of input samples (Wallach et al. 2009):

$$Accuracy = \frac{TP + TN}{t} . \tag{3}$$

2. Precision: Precision is the number of items correctly predicted, divided by the number of all predictions Equation (4). Precision shows the percentage of

predictions that are done correctly (Wallach et al. 2009):

$$Precision = \frac{TP}{TP + FP}\,. \tag{4}$$

3. Recall or Sensitivity: The recall is the number of items correctly predicted, divided by the total number of aggressive cancer type Equation (5) (Wallach et al. 2009). Recall shows the percentage of predictions correctly made by the classifier as aggressive to the number of all aggressive cancer type:

$$Recall = \frac{TP}{TP + FN}\,. \tag{5}$$

4. Specificity: Specificity is the true negative rate or the proportion of negatives that are correctly identified (Chitra & Seenivasagam 2013):

$$Specificity = \frac{TN}{FP + TN}\,. \tag{6}$$

5. F1-Score: The F1-score is the average of the precision and recall, where the best score is 1, and the worst score is 0. F1-score combines both precision and recall so it can be used as the overall utility of the model (Wallach et al. 2009):

$$F1 = 2 * \frac{Precision.Recall}{Precision + Recall}\,. \tag{7}$$

4.4.2 Area Under the Curve

Area Under the Curve (AUC) is one of the most commonly used metrics for evaluation of deep learning models (Fawcett 2006). AUC is one of the mainly used metric for binary classification problem performance evaluation. The AUC of a classifier gives the probability that the classifier ranks a randomly chosen aggressive cancer type example higher than a randomly chosen non-aggressive cancer type example. Before defining AUC, let us understand two important terms:

1. True Positive Rate: We can define True Positive Rate as the number of items correctly predicted as aggressive, divided by the total number of the sample with an aggressive cancer type. True Positive Rate corresponds to the proportion of aggressive cancer type data that are correctly predicted as aggressive, to all aggressive data points (Fawcett 2006).

2. False-Positive Rate: We can define the False-Positive Rate as the number of FP / (FP+TN). The false-positive rate represents the proportion of non-aggressive cancer types that are incorrectly predicted as aggressive, to all non-aggressive data points (Fawcett 2006).

Both the False-Positive Rate and true positive rate take values in the range $[0, 1]$. The ROC curve is a plot between the false-positive rate on the x-axis and true positive rate on the y-axis. The area under the curve is in the range $[0, 1]$. A higher AUC indicates a better performance of the classifier (Fawcett 2006).

In this chapter, we discuss the architecture of AGCP, followed by the working of our proposed model, complexity, and evaluation of the model. We discuss the experimental setup and the results of the application of our model on the three benchmark datasets in the following chapter. We also give a comparison of our model with two other state-of-the-arts graph classification models.

Chapter 5

Results and Comparative Analysis

In this chapter, we present the classification tasks we defined with a detailed discussion of the datasets, followed by the performance evaluation of the proposed model in terms of classification accuracy and AUC. We further report the wall-clock training time in minutes until convergence of our method. In several experiments on the datasets, we demonstrate the performance of our model on a set of hyperparameters.

5.1 Experiments and Results

In this section, we explain different applications of our proposed model on different datasets, following which we introduce two baseline models that we are using to compare the performance of our proposed model.

5.1.1 Datasets

Here, we discuss different datasets that we use in our research. We begin by introducing ENZYME, followed by D&D and GINA which is a dataset of weighted attributed graphs that we generated to classify cancer types to either aggressive or non-aggressive.

1. ENZYME: ENZYME is a dataset of tertiary protein graph structures obtained from (Borgwardt et al. 2005), which consists of 600 enzymes from the BRENDA enzyme database (Sen et al. 2008). In this case, the task is to correctly assign each enzyme to one of the six Enzyme Commission Numbers (EC number), top-level classes, summarized in Table 3. Out of 16 attributes, the ones that help us

distinguish EC numbers are catalyzed reaction rate, kinetics, substrates/products, inhibitors, cofactors, activators, structure, and stability.

Table 3: Top-level EC Numbers.

Class	**Reaction catalyzed**
EC_1 Oxidoreductases	To catalyze oxidation/reduction reactions; transfer of H and O atoms or electrons from one substance to another.
EC_2 Transferases	To transfer a functional group from one substance to another. The group may be methyl-, acyl-, amino- or phosphate group.
EC_3 Hydrolases	To help the formation of two products from a substrate by hydrolysis.
EC_4 Lyases	To help in non-hydrolytic addition or removal of groups from substrates.
EC_5 Isomerases	To help in intramolecular rearrangement, such as isomerization changes, within a single molecule.
EC_6 Ligases	Join together two molecules by the synthesis of new C-O, C-S, C-N or C-C bonds with the simultaneous breakdown of ATP.

2. D&D: D&D dataset consists of 1,178 protein structure graphs, which are either enzymes or non-enzymes. The task is to classify a protein as enzyme or non-enzyme, which is essentially a two-class classification problem. We embedded

36 attributes to the vertices of the graph. Some of the most useful attributes for distinguishing enzymes from non-enzymes are secondary-structure content, amino acid frequencies, number of disulfide bonds, and size of the largest cleft.

3. GINA: GINA is a dataset consisting of 498 graphs that we have generated to study the interactions of genes in a patient diagnosed with prostate cancer, with or without mutations. We aim to predict the aggressiveness of the disease, in a given patient. We are leveraging the gene expressions as well as the patient's clinical attributes to make more accurate predictions.

   We obtained the genetic mutation data of prostate cancer patients, along with their clinical attributes and gene expression from cBioPortal (*cBioPortal* 2012). We then loaded the genetic mutation data into the GeneMANIA Cytoscape plugin (Smoot et al. 2010) to generate the graph-structured data with weighted edges. Each graph, thus, represents the gene interaction network of a patient. After obtaining the graph-structured data, we embeded the clinical attributes of the patient to the vertices of their respective gene interaction network.

   A gene interaction network is a set of vertices representing genes connected by edges representing functional relationships amongst them. The edges are physical interactions, meaning the two given genes are thought to interact with their gene products such as RNA or proteins. In this thesis, we are interested in physical interactions between two genes in terms of their protein-protein interaction study. These data are collected from primary studies found in protein interaction databases, including BioGRID (Stark et al. 2006) and PathwayCommons (Cerami et al. 2010).

   Each vertex of the gene interaction network has a real-valued attribute vector with n dimensions. In this thesis, we generated three variations of GINA: GINA I with 24 attributes, GINA II with 25 attributes, and GINA III with 1 attribute in the vertices of the gene interaction network. The attributes of different variations of GINA is a combination of clinical attributes and gene expression data. Each graph has an associated label, which is the aggressiveness

of the cancer type of a particular patient. The classification task is essentially a binary classification problem to classify the disease state of the tumor to an aggressive or non-aggressive type. Patients who have a Gleason score of seven or above are considered to have an aggressive cancer type, and a Gleason score below seven is considered to be a non-aggressive cancer type. Figure 30 shows a gene interaction network of the proposed labeled, attributed GINA, with n attributes embedded to each of the vertices. The label is the state of the disease and can be either aggressive or non-aggressive cancer type.

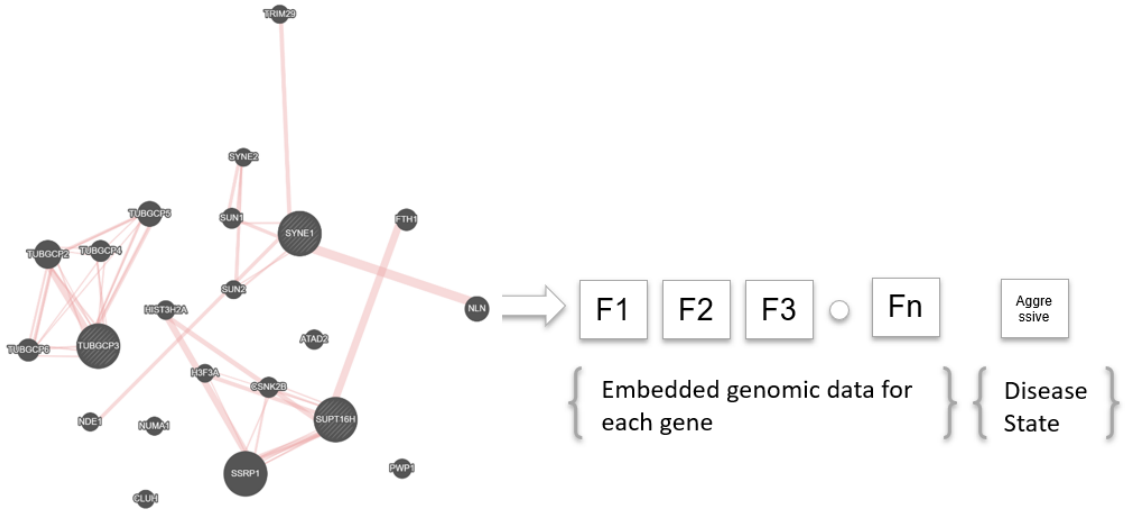


Figure 30: Schematic view of an attributed gene interaction network with label. The gene interaction network has been created using GeneMANIA Cytoscape plugin (Warde-Farley et al. 2010).

Data pre-processing: Data pre-processing is an essential step in any deep learning task; this step prepares raw data for further processing. The clinical attributes are categorical, and we converted them to real-valued vectors using one-hot encoding (Bishop 1995) for GINA I, GINA II, and GINA III. ENZYME and D&D dataset were already pre-processed and were obtained from (Zhang et al. 2018).

In this thesis, we use one-hot encoding (Bishop 1995) to convert the raw categorical attributes into real-value vectors on GINA I, GINA II, and GINA III. ENZYME and D&D dataset were already preprocessed and are obtained from (Zhang et al. 2018).

We created the following datasets with GINA, to study the potential variations in performance:

1. GINA I: This dataset consists of a weighted gene interaction network with 24 real-valued clinical attributes embedded as the vertex attributes.

2. GINA II: This dataset consists of a weighted gene interaction network with 24 real-valued clinical attributes and additionally, one gene expression of each gene embedded as the vertex attributes. Thus, this dataset contains 25 attributes in total.

3. GINA III: This dataset consists of a weighted gene interaction network with only one gene expression value embedded as the vertex attribute.

The statistics of the datasets are given in Table 5.

Table 5: Dataset statistics.

Dataset	**Number of graphs**	**Avg number of edges**	**Avg number of vertices**	**Number of classes**	**Number of Attributes**
GINA I	498	204.60	302.40	2	24
GINA II	498	204.60	302.40	2	25
GINA III	498	204.60	302.40	2	1
D&D	1,178	1921.60	284.40	2	36
ENZYME	600	124.30	32.60	6	16

5.1.2 Data Splitting

We have divided all five datasets: D&D, ENZYME, and the three variations of GINA into training and testing datasets. Following conventional settings, we performed a 10-fold cross-validation (Browne 2000). In the 10-fold cross-validation, we divide the dataset into 10-equal folds or parts. We used one of the fold as the testing set and the union of remaining folds as the training set and repeated the experiments for 2 times; thus, 20 runs per dataset. We repeat the process of choosing a training and testing data from different folds and calculating the testing accuracy for 20 runs per dataset. The final classification accuracy is the average testing accuracy of all 20 runs.

5.1.3 Baselines

In this work, we compare AGCP with two other baseline methods: graph convolutional networks (GCN) (Kipf & Welling 2016) discussed in Section 3.1.2, and deep graph convolutional neural network (DGCN) (Zhang et al. 2018) discussed in Section 3.2.2. Both models work well with binary-class and multi-class classification problems.

5.2 Performance Evaluation in terms of Classification Accuracy

We implemented the proposed model and baseline models using PyTorch (Fey et al. 2018) and Tensorflow (Abadi, Martín and Barham, Paul and Chen, Jianmin and Chen, Zhifeng and Davis, Andy and Dean, Jeffrey and Devin, Matthieu and Ghemawat, Sanjay and Irving, Geoffrey and Isard, Michael 2016). Table 6 shows a summary of the mean classification accuracy and the standard deviations after 20 runs. In each run, we train AGCP for 400 epochs, GCN, and DGCN for 300 epochs. The results indicate that the proposed model performs marginally better over the baseline models. As shown in Table 6, for GINA I, our model performs slightly better than GCN and considerably better than DGCN. For GINA II, AGCP outperforms GCN

and DGCN. On D&D, AGCP outperforms GCN by a considerable margin and DGCN by a slight margin. On ENZYME, AGCP gives better classification accuracy than DGCN and GCN by a slight margin. On the other hand, on GINA III, GCN outperforms AGCP by a slight margin, and AGCP outperforms DGCN by a considerable margin.

We anticipate that the improved performance of our model is due to the proposed convolution layer better at capturing the correlated attributes present on the vertices and its neighbors. The pooling layer is also able to select the graph vertices that constitute for better classification.

Table 6: Result summary in terms of classification accuracy.

Method	**AGCP**	**GCN**	**DGCN**
GINA I	75.30 ± 2.54	74.40 ± 1.90	70.67 ± 1.20
GINA II	79.60 ± 1.34	73.20 ± 2.30	68.00 ± 1.40
GINA III	$70.20 \pm .70$	72.60 ± 0.64	62.90 ± 0.56
D&D	81.98 ± 0.87	72.40 ± 0.60	79.30 ± 0.90
ENZYME	72.98 ± 0.87	70.40 ± 0.60	72.37 ± 0.94

Performance Evaluation on GINA: Here, we discuss the performance of AGCP, GCN, and DGCN on all three variations of GINA. We provided different variations of GINA as input to the model. We trained the proposed model and the baseline models with the best hyperparameters for each model after repeated experiments. We provide more details of the experiments in Section 5.5.1. GINA II contains gene interaction networks of prostate cancer patients, along with the twenty-five clinical attributes of genetic mutations. Compared to the other two variations of GINA,

GINA II has richer attributes present in the vertices of the gene interaction networks. AGCP yields the highest average classification accuracy of 79.60% in classifying GINA II, compared to GCN and DGCN. In comparison with the three variations of GINA, AGCP on GINA II gives the highest accuracy. The increase in classification accuracy in GINA II is accounted for by the rich attributes present in graph vertices. We can also infer that the clinical attributes of mutations along with gene expression data aids in the classification of aggressiveness of cancer. AGCP yields higher accuracy than DGCN by 11.60% (79.60% - 68.00%) and GCN by 6.40% (79.60% - 73.20%).

AGCP on GINA I give the second-best results in terms of accuracy followed closely by GINA III. GINA I contains all twenty-four genomic mutation attributes of the patients diagnosed with prostate cancer. AGCP classifies the aggressiveness of the tumor with an average accuracy of 75.30%.

GINA III contains only one attribute, which is gene expression values of a gene in the network as vertex attributes. Analyzing the results ascertain that the clinical attributes, along with the gene interaction network, can aid in the classification of the aggressiveness of cancer. AGCP on GINA III achieves 70.20% accuracy with GCN, which is the best in comparison to AGCP and DGCN. The results on all three variations of GINA indicate that the accuracy of our proposed model increases with an increased number of attributes, which is ideal for analyzing real-life graph data since the graphs used in day-to-day life have a high number of attributes.

Performance Evaluation on D&D: D&D is the largest dataset with which we conducted our experiments. AGCP shows the highest accuracy for this dataset, which indicates that the model performs better with more data samples as more data samples help the model to learn the attributes better. To give more quantitative insight into the performance trends, AGCP performs 9.58% (81.98% - 72.40%) better in terms of classification accuracy than GCN and 2.65% (81.98% - 79.30%) better than DGCN.

Performance Evaluation on ENZYME: On ENZYME, the classification accuracy of AGCP is considerably better than the other two models; DGCN and GCN. This indicates that our proposed model works very well on multi-class classification

problems, even with a limited amount of data for each class. To provide more quantitative insight into the performance trends, AGCP yields a higher accuracy of 2.58% (72.98% - 70.40%) better than GCN and 0.61% (72.98% - 72.37%) better than DGCN in terms of classification accuracy.

5.3 Performance Evaluation in terms of AUC

We plotted the ROC curve for AGCP, GCN, and DGCN on GINA II and D&D. GINA II has rich attributes in the gene interaction network. This dataset also achieves better classification accuracy than the other two GINA; thus, we plotted ROC for GINA II only amongst the three variations. Figure 31 plots the ROC curve for AGCP, GCN, and DGCN on GINA II, and Figure 32 plots the ROC for AGCP, GCN, and DGCN on D&D. Table 8 gives the AUC for AGCP, DGCN, and GCN on D&D and GINA II.

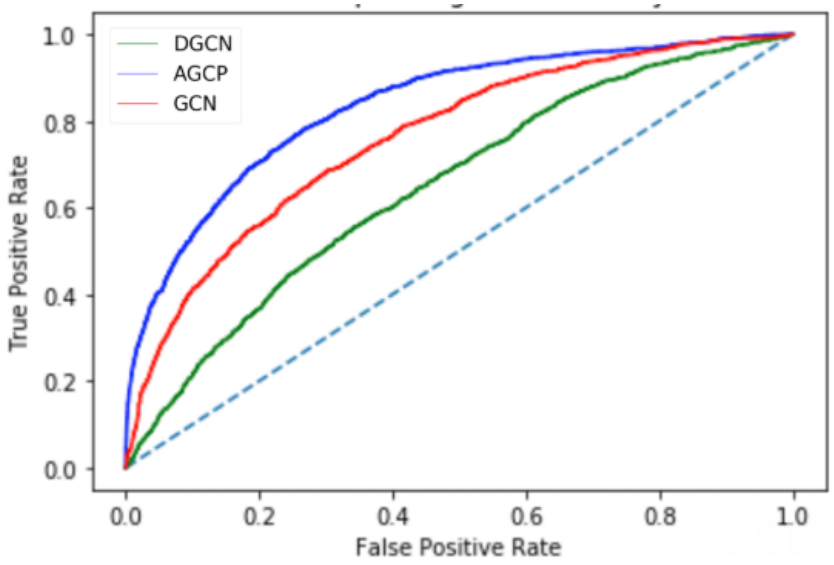


Figure 31: ROC for GINA II.

Figure 31 plots the ROC for AGCP, GCN, and DGCN on GINA II. From the ROC analysis, it is evident that AGCP (AUC = 0.83) outperforms GCN (AUC = 0.75) and DGCN (AUC = 0.71) by a significant margin. The better performance in classification is accounted for by the efficiency of AGCP's convolution layer that

we propose, which is better at capturing the statistical patterns hidden in the graph structure.

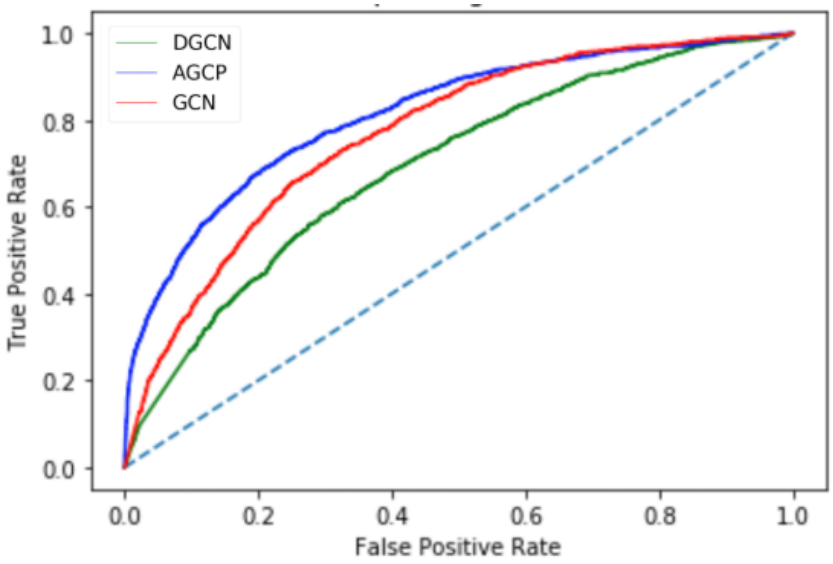


Figure 32: ROC for D&D.

Figure 32 plots the ROC for AGCP, GCN, and DGCN on D&D. Observing the ROC curve it is evident that AGCP (AUC = 0.81) outperforms GCN (AUC = 0.77) and DGCN (AUC = 0.73) by a significant margin even on larger datasets.

Table 8: Summary of AUC for AGCP, GCN and DGCN.

Dataset	**AGCP**	**GCN**	**DGCN**
GINA II	0.83	0.75	0.71
D&D	0.81	0.77	0.73

Evaluation of AGCP using Precision, Recall/Sensitivity, Specificity and F1-score: Precision, specificity, F1-score, and recall/sensitivity are the commonly used statistical measures to illustrate the medical diagnostic classification and primarily used to enumerate the performance and consistency of the classifier (Chitra & Seenivasagam 2013). Sensitivity/recall evaluates the classifier for correctly detecting

an aggressive cancer type. Specificity measures how the proportion of patients classified as non-aggressive cancer types can be correctly ruled out. Precision shows the percentage of predictions accurately made by the classifier. Table 9 shows the precision, recall/sensitivity, F1-score, and specificity of binary-classification using AGCP on GINA I, II, III, and D&D datasets. The performance of AGCP on all three variations of the dataset is a proof-of-concept experiment, which shows that "it works." We can predict the aggressiveness of cancer by analyzing the gene interaction network and embedded attributes.

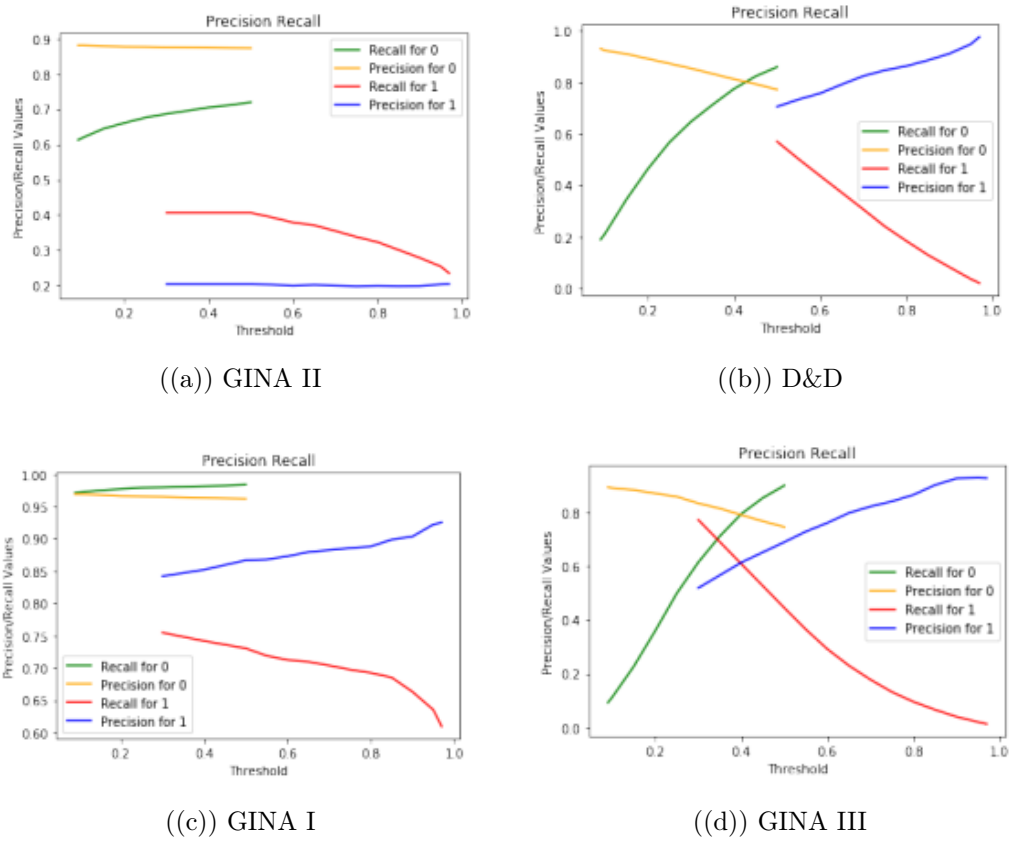
As observed in Table 9, on GINA I dataset, AGCP classifies the cancer types as aggressive or non-aggressive with a precision of 64%, a recall of 69%, the specificity of 71% and F1-score of 66%. AGCP performs comparatively well in distinguishing the aggressive cancer type from non-aggressive types. On the other hand, on GINA II, AGCP classifies the cancer types as aggressive or non-aggressive with a precision of 79%; it performs marginally well in distinguishing the aggressive cancer type from non-aggressive types. On GINA III, AGCP can classify the cancer types as aggressive or non-aggressive with a precision of 33%, a recall of 30%, a specificity of 35%, and F1-score of 30%. AGCP can perform considerably well in distinguishing the aggressive cancer type from non-aggressive types.

The result of classification on D&D is shown in Table 9. AGCP model can classify the graphs as an enzyme or non-enzyme with a precision of 65%, a recall of 77%, a specificity of 66%, and an F1-score of 66%.

Table 9: Results of AGCP for binary classification.

Dataset	**Precision**	**Recall**	**F1-score**	**Specificity**
GINA I	0.64	0.69	0.66	0.71
GINA II	0.82	0.62	0.71	0.79
GINA III	0.33	0.30	0.31	0.35
D&D	0.65	0.77	0.70	0.66

Figure 33 shows the precision-recall curve for GINA I, GINA II, GINA III, and D&D datasets. For ease of representation, we use 0 for non-aggressive cancer type and 1 for aggressive cancer type. The graphs are the plot for different threshold values, starting from 0.1 to 1.0, along with precision and recall for each class at different threshold values. Based on the user requirements, we can adjust the threshold to produce results that are desirable. For example, at threshold .4 for GINA II precision and recall for aggressive cancer type is low. Recall measures the percentage of actual aggressive cancer types that were correctly classified as aggressive cancer types as we increase threshold, the false positive rate decreases, but false-negative rate increase. As a result, precision increases, while recall decreases.



((a)) GINA II

((b)) D&D

((c)) GINA I

((d)) GINA III

Figure 33: The precision-recall graphs for (a) GINA II, (b) D&D , (c) GINA I, and (d) GINA III datasets.

5.4 Training Time per Run

We report the mean training time per run (forward propagation, backpropagation, and cross-entropy error calculation) for AGCP. The model is trained using a CPU and a GPU. Figure 34 compares the training time with each type of microprocessor. We used D&D to calculate the training time per run and varied the input graph sample size from 200 graphs to 1200 graphs. We used the following sets of hyperparameters: 0.001 learning rate, learning rate decay of 0.01, momentum 0.99, 0.5 dropout rate, and a batch size of 64 for 400 epochs. We experimentally proved that the training time and the number of graphs are directly proportional, that is, training time linearly scales as the number of graphs. As expected, the training time taken on GPU is much

lower than the CPU due to the TensorFlow cuDNN optimization (Chen et al. 2015).

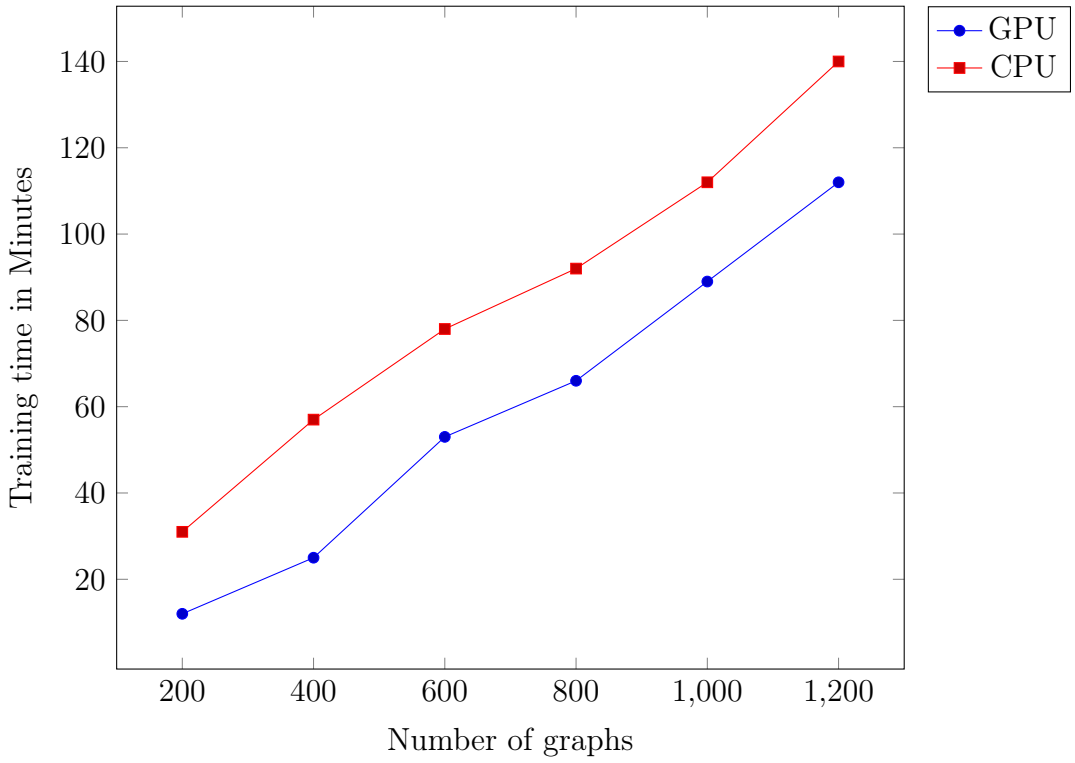


Figure 34: Time per run for different graph sample sizes.

5.5 Experiments with Hyperparameters

Hyperparameters are a set of variables that need to be set before applying a deep learning model to a dataset. The hyperparameters vary depending on the dataset and the classification problem (Goodfellow et al. 2016). Therefore, we need to find the hyperparameters that work well with our proposed model and the targeted datasets, such as GINA, ENZYME, and D&D. We have used the following hyperparameters in this thesis:

Learning Rate: Deep neural networks are trained using an algorithm called stochastic gradient descent (SGD). SGD is an optimization algorithm that calculates the error gradient for the current state of the model by comparing the predicted and actual class labels from the training dataset and then update the weights of the model

using the backpropagation of error to update the state of the algorithm. The learning rate is the amount by which the state of the model is updated during training to produce the results. It is often a positive value and range between 0.0001 and 1.0. For example, a learning rate of 0.1 means that weights in the network are updated $0.1 \times (weight\ error)$. In this thesis, we manually tune AGCP to find the best learning rate (Bengio 2012).

Batch Size: While training a neural network, passing one sample at a time is computationally costly. Thus it is essential to define the number of samples to work through before updating the internal model parameters. This is called setting the *batch size*. In this thesis, we performed experiments with batch sizes of 16, 32, and 64.

Training Epochs: The number of epochs is a hyperparameter that defines the number times that the neural network training will work through the entire training dataset to produce the best results. While we train our neural network with SGD batch by batch, we update the state of the neural network to minimize the error. Each batch is passed multiple times to reduce the error of prediction. The number of passes each batch makes is called *number of epochs*. Thus, one epoch means that each batch in the training dataset has had a chance to update the state. In this thesis, we performed experiments with 100, 200, 300,400, and 500 epochs.

Regularization To avoid overfitting in AGCP architecture, we included a dropout layer. At every iteration during training, the dropout layer selects some nodes at random and removes them along with their connections. In our model, the dropout is only applied to the prediction generation layer. The neural units with a probability value of less than 0.5 are removed (Srivastava et al. 2014). We found out experimentally that the regularization does not affect the performance in terms of accuracy but avoids overfitting.

5.5.1 Results and Discussion of Hyperparameter Tuning

We manually tuned the hyperparameters for AGCP, GCN, and DGCN on each of the five datasets. We tuned the hyperparameters on one random splitting of training (90%) and testing (10%) data to select one pair of hyperparameters for each dataset to use consistently in all ten series of cross-validations instead of tuning independently for ten series of datasets. Following the work done by (Kipf & Welling 2016), the learning rates are selected from 0.1, 0.01, 0.001, and 0.0001 and training epochs from 100, 200, 300, 400, and 500. The combination, which both shows the convergence of optimization and small overfitting, is selected. Table 11 shows the hyper-parameters we adopted to evaluate the model. Figures 35 and 36 plot the changes in the test and training accuracies and the cross-entropy error of AGCP on GINA II as the model learns, respectively. Based on the performance in terms of accuracy of the model, we trained AGCP on GINA II for 400 epochs.

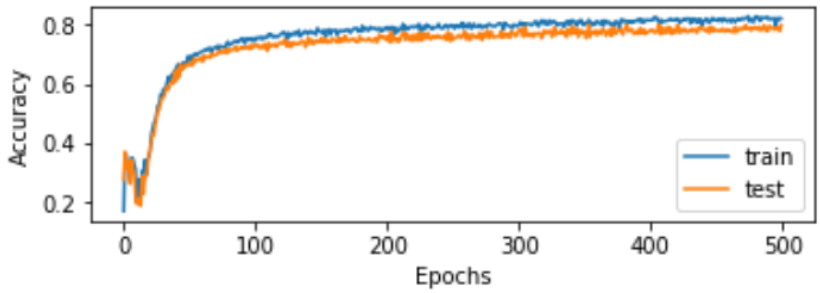


Figure 35: Plot of AGCP model performance with respect to the number of epochs.

Figure 35 clearly shows that the accuracy steadily increases until 100 epochs, and from 200 epochs, the accuracy gradually increases until it gives improved results, after 400 epochs the model accuracy does not change significantly.

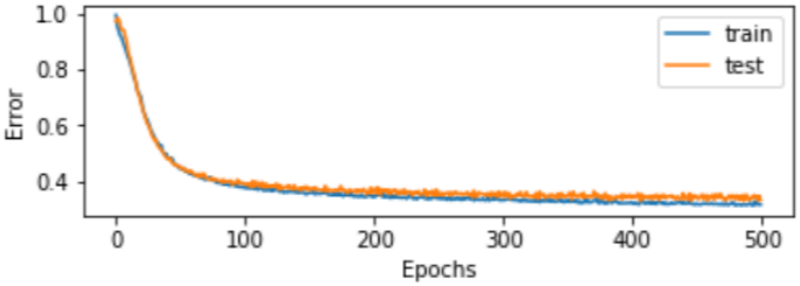


Figure 36: Plot of AGCP model error with respect to the number of epochs.

Figure 36 clearly shows that the error steadily decreases until 100 epochs, and from 200 epochs, the error gradually decreases until it reaches minimum values.

We also plotted the performance of AGCP on GINA II with respect to the different learning rates we selected for experimenting with. Figure 37 shows the variation in performance with respect to 500 epochs for different learning rates.

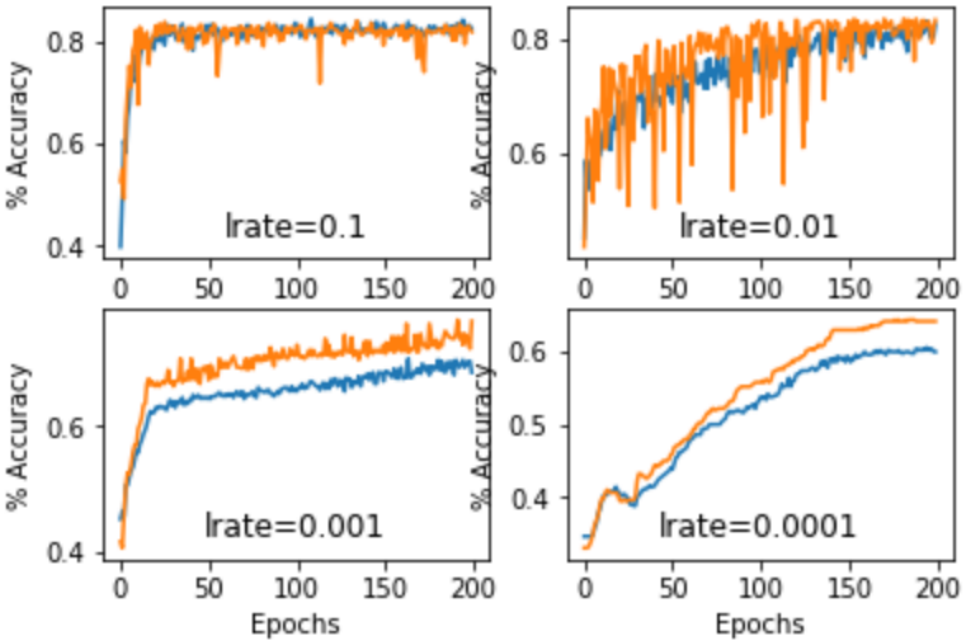


Figure 37: Plot of AGCP accuracy with respect to different learning rates on GINA II.

Figure 37 shows that the accuracy oscillates for higher learning rates such as 1.0 and 0.1. The model hardly learns anything for learning rates that are too low, such as 0.0001. We chose 0.1 to train AGCP on GINA. Likewise, for GCN and DGCN, we found the best number of epoch and learning rates on all datasets. The results are summarized in Table 11. Due to the larger size of this dataset compared to the others, the model converged to optimization without overfitting the data at a higher number of epochs compared to other datasets.

Annealing the Learning Rate: In this thesis, we did experiments with annealing of learning rate over time as follows:

Dataset	**GINA I**	**GINA II**	**GINA III**	**D&D**	**ENZYME**
Hyperparameters for AGCP					
Learning rate	0.1	0.1	0.1	0.01	0.01
Number of epochs	400	400	400	500	300
Hyperparameters for GCN					
Learning rate	0.01	0.01	0.01	0.01	0.01
Number of epochs	300	300	300	400	300
Hyperparameters for DGCN					
Learning rate	0.0001	0.0001	0.0001	0.001	0.001
Number of epochs	300	300	300	500	300

Table 11: Summary of hyperparameters for AGCP, GCN and DGCN.

1. Decay: Learning rate decay helps slowly reduce the learning rate of the algorithm and helps it to converge faster. Figure 38 shows that the desired accuracy is achieved within 20 epochs rather than 100 to 200 epochs.

2. Momentum: Momentum accelerated the learning rate by smoothing the progression of the learning rate. We selected the best learning rate for AGCP, GCN, and DGCN and tried to ease the learning by applying momentum values (0.50, 0.60, 0.80, 0.90, 0.99). We experimentally found that the performance is accelerated by 1.85% for AGCP, 1.05% for GCN, and 1.09% for DGCN when we applied a learning momentum of 0.99 for all three models.

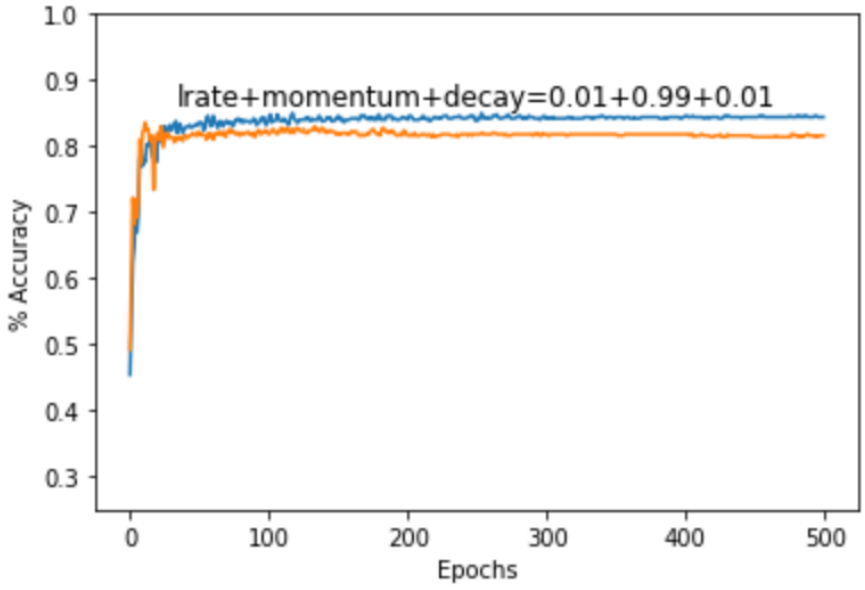


Figure 38: Plot of AGCP accuracy variation with respect to the learning rate, momentum and decay on GINA II.

Batch Size: We performed experiments with batch sizes 64, 32 and 16. We experimentally found that the batch size of 64 gives the best performance and is plotted in Figure 39.

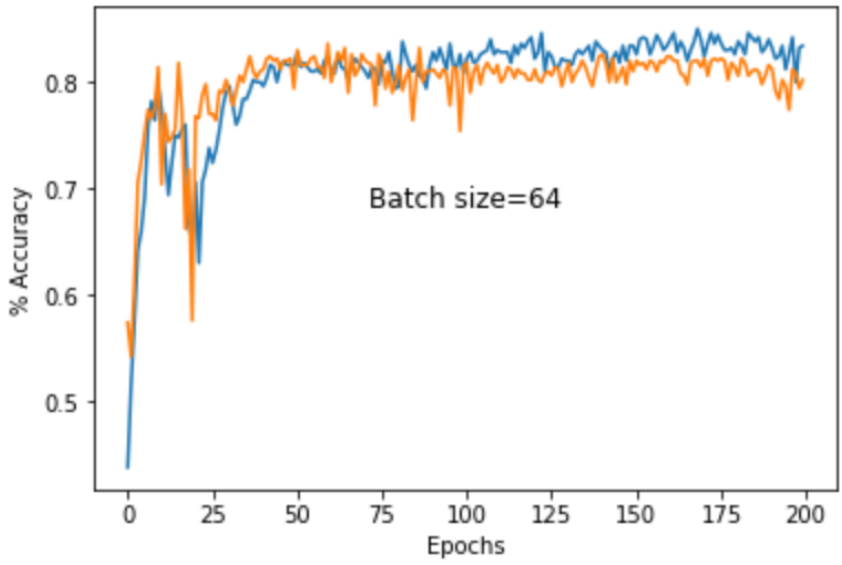


Figure 39: Plot of AGCP accuracy variation with batch size of 64 on GINA II.

The performance changes with respect to the number of hidden layer l are ana-

lyzed. In several experiments, the results show that the best l to use is 3. We report that 3 layers give a boost in accuracy by $5-7\%$, compared to hidden layer size of 2, and beyond 3, the performance gave prohibitively diminishing returns. We used 32 neural units in each hidden layer and prediction generation layer.

The performance changes with respect to the input search depth, k is also analyzed. For AGCP implemented on GINA II, we found that setting $k = 2$ provided a consistent boost in the accuracy of around $8-10\%$ on average compared to $k = 1$. However, increasing k beyond 2 gave marginal performance $(0-4\%)$, while increasing the runtime by a large factor.

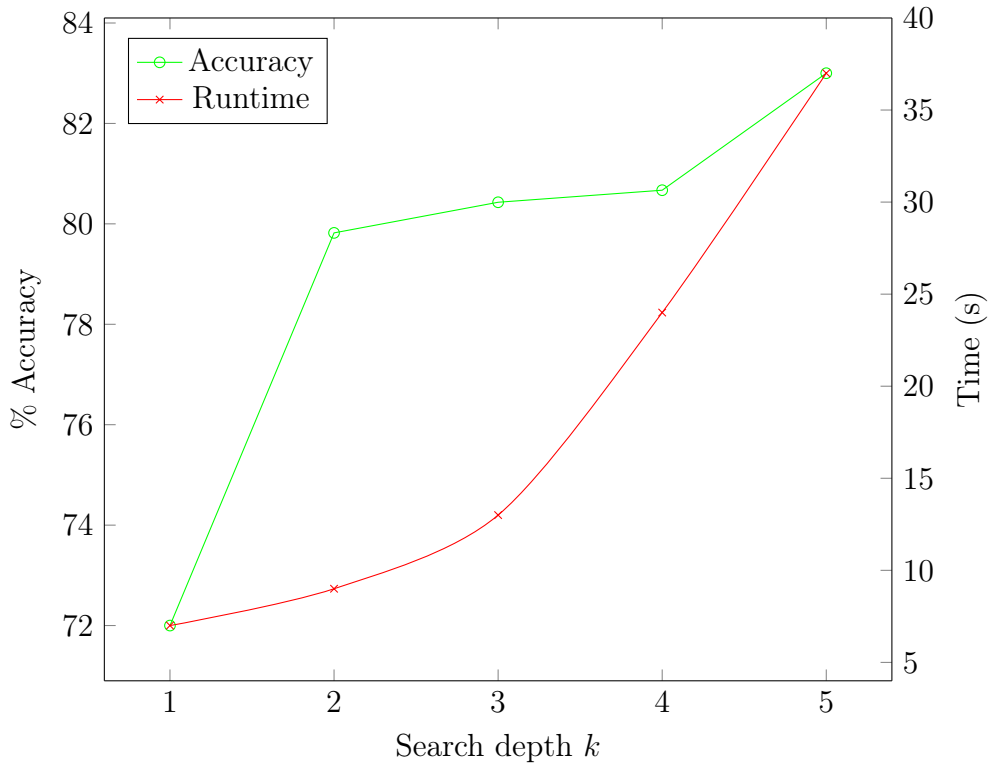


Figure 40: Comparison of accuracy and runtime in seconds with respect to search depth k.

5.5.2 Implementation and Tools

In this section, we discuss the tools that we have used in the implementation of AGCP. For DGCN implementation, the code was based on the work of (Zhang et al. 2018). For GCN implementation, the baseline architecture was based on the code

by (Kipf & Welling 2016). Most of the unique contributions were applying GCN and DGCN on GINA to classify the aggressiveness of cancer and tuning the various hyperparameters. Tensorboard, a graphical visualization using TensorFlow, was used for monitoring and assessing convergence. The code is available for reference at https://github.com/SushaSureshh/AGCP. The environment, followed by the development tools, are explained below.

Testing environment

1. Operation system: 64-bit Windows 10 Enterprise Edition, version 10.0.17763
2. System type: x64 based processor
3. CPU: Intel core i7-4790 with 3.6GHz frequency
4. RAM: 16 GB
5. GPU: NVIDIA Pascal GPUs (12 GB)

Development Tools

1. Languages used: Python (3.5) and Python (3.7)
2. Development tools: PyTorch (1.2), Tensorflow (>0.12), Keras (2.2.4-tf), NetworkX, Matplotlib

Chapter 6

Conclusion and Future work

From social networks to biological networks, graphs are a natural way to represent a diverse set of real-world data. We presented AGCP, an efficient variation of convolutional neural networks (CNN), which operate directly on weighted, attributed graphs. The architecture of AGCP has a linear filter function that convolves over the fixed topological structure of a graph to encode the representation of both local and global attributes. Convolution is followed by a pooling layer that coarsens the graph while preserving the global structure of the original input graph using information gain. In several experiments on the bioinformatics datasets, we empirically proved that AGCP yields better results in terms of performance accuracy relative to the previously proposed models by a considerable margin.

6.1 Summary of Contributions

Below is a summary of the significant contributions made throughout this research:

Novel paradigm of supervised classification of graph data. We proposed a novel supervised classification model called AGCP that works well on graph-structured data generated from non-Euclidean or irregular domains. Our proposed model works well on a dataset of weighted, attributed graphs with arbitrary size and fixed topology. The proposed model performs better in terms of accuracy than the two other benchmark models in graph classification by a significant margin.

New Pooling Strategy for weighted, attributed graph coarsening. Downsampling is very important in any machine learning task. We proposed a pooling layer that downsamples the input graph and then generates a sub-graph with vertices and edges that are the most relevant for the classification problem.

Applied the proposed model to classify the aggressiveness of prostate cancer based on mutation data. We generated a weighted, attributed graph dataset called GINA with three different variations to predict the aggressiveness of Prostate cancer. We used the proposed model in the classification of the aggressiveness of prostate cancer.

Applied the proposed model to classify the six top-level EC numbers. The proposed model is also used in the multi-class classification problem to classify the tertiary graph structures to 6 top-level enzyme numbers using ENZYME and binary classification problem to classify the protein structures to either enzymes or non-enzymes.

6.1.1 Conclusions on Supervised Classification by the Proposed Model AGCP

Our research objective was to design an efficient variant of the graph neural network that works well on the weighted, attributed graph data. We experimentally proved that the model we proposed in this research performs better than the two other existing models in terms of classification accuracy and ROC. We conducted multiple experiments to conclude that our model performs well with the more relevant dataset. We experimentally showed that the quality of graphs is vital in better performance of the supervised classification model that we proposed. The model performance also increases with an increased number of relevant attributes. Our model performs well for both binary-class and multi-class classification, while still comparing favorably in terms of efficiency (training time).

6.2 Future Work

Although our proposed model works well on attributed, weighted graphs, there are some tasks that researchers would like to perform in the future.

Quality of graph dataset: The performance of the model that we proposed depends on the quality of the graph data, which is evident from the performed experiments.

Directed graphs: The model we proposed currently does not naturally support directed graphs. The solution is representing these directed graphs as undirected bipartite graphs. We would like to conduct more in-depth experiments on directed graphs in the future.

Applications: Despite the applications of AGCP on graph classification demonstrated in this research. We would like to apply the proposed model on node classification, network embedding, graph generation, and spatial-temporal graph forecasting, node clustering, link prediction, and graph partitioning. We would also anticipate applying AGCP on domains such as computer vision, natural language processing, traffic analysis, document classification, target interactions, disease-gene associations, and recommendation systems.

Multi-class classification: Even though the proposed model works marginally well for multi-class problems, we need to acquire and conduct more experiments to understand the limitations of the proposed model and to improve the performance for multi-class classification problems.

REFERENCES

Abadi, Martín and Barham, Paul and Chen, Jianmin and Chen, Zhifeng and Davis, Andy and Dean, Jeffrey and Devin, Matthieu and Ghemawat, Sanjay and Irving, Geoffrey and Isard, Michael (2016), TensorFlow: A System for Large-Scale Machine Learning, *in* '12th {USENIX} Symposium on Operating Systems Design and Implementation', pp. 265–283.

Albertsen, P. C., Hanley, J. A., Gleason, D. F. & Barry, M. J. (1998), 'Competing Risk Analysis of Men Aged 55 to 74 Years at Diagnosis Managed Conservatively for Clinically Localized Prostate Cancer', *JAMA* **280**(11), 975–980.

Alkhateeb, A., Rezaeian, I., Singireddy, S., Cavallo-Medved, D., Porter, L. A. & Rueda, L. (2019), 'Transcriptomics Signature from Next-Generation Sequencing Data Reveals New Transcriptomic Biomarkers Related to Prostate Cancer', *Cancer Informatics* **18**, 1176935119835522.

Asim, Y., Shahid, A. R., Malik, A. K. & Raza, B. (2018), 'Significance of Machine Learning Algorithms in Professional Blogger's Classification', *Computers & Electrical Engineering* **65**, 461–473.

Azhagusundari, B. & Thanamani, A. S. (2013), 'Feature Selection based on Information Gain', *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* **2**(2), 18–21.

Belkin, M., Niyogi, P. & Sindhwani, V. (2006), 'Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples ', *Journal of Machine Learning Research* **7**(Nov), 2399–2434.

Bengio, Y. (2012), Practical recommendations for gradient-based training of deep architectures, *in* 'Neural networks: Tricks of the Trade', Springer, pp. 437–478.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford university press.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J. & Kriegel, H.-P. (2005), 'Protein function prediction via graph kernels.', *Bioinformatics* **21**(suppl_1), i47–i56.

Bottou, L. (2010), Large-Scale Machine Learning with Stochastic Gradient Descent, *in* 'Proceedings of COMPSTAT'2010', Springer, pp. 177–186.

Brenda (2019), 'Enzyme Classification', `https://www.brenda-enzymes.org/index.php/`. Accessed: 2019-08-22.

Browne, M. W. (2000), 'Cross-Validation Methods', *Journal of Mathematical Psychology* **44**(1), 108–132.

Bruna, J., Zaremba, W., Szlam, A. & LeCun, Y. (2013), 'Spectral Networks and Locally Connected Networks on Graphs', *arXiv preprint arXiv:1312.6203* .

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R. & Mitchell, T. M. (2010), Toward an Architecture for Never-Ending Language Learning, *in* 'Twenty-Fourth AAAI Conference on Artificial Intelligence'.

cBioPortal (2012), `https://www.cbioportal.org/`. Accessed: 2019-08-22.

Cerami, E. G., Gross, B. E., Demir, E., Rodchenkov, I., Babur, Ö., Anwar, N., Schultz, N., Bader, G. D. & Sander, C. (2010), 'Pathway Commons, a web resource for biological pathway data.', *Nucleic Acids Research* **39**(suppl_1), D685–D690.

Chang, J., Gu, J., Wang, L., Meng, G., Xiang, S. & Pan, C. (2018), Structure-Aware Convolutional Neural Networks, *in* 'Advances in Neural Information Processing Systems', pp. 11–20.

Chen, F.-C. (1990), 'Back-propagation neural networks for nonlinear self-tuning adaptive control', *IEEE Control Systems Magazine* **10**(3), 44–48.

Chen, H., Perozzi, B., Hu, Y. & Skiena, S. (2018), HARP: Hierarchical Representation Learning for Networks, *in* 'Thirty-Second AAAI Conference on Artificial Intelligence'.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C. & Zhang, Z. (2015), 'MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems', *arXiv preprint arXiv:1512.01274* .

Chitra, R. & Seenivasagam, V. (2013), 'Heart Disease Prediction System Using Supervised Learning Classifier', *Bonfring International Journal of Software Engineering and Soft Computing* **3**(1), 01–07.

Creswell, A., Arulkumaran, K. & Bharath, A. A. (2017), 'On denoising autoencoders trained to minimise binary cross-entropy', *arXiv preprint arXiv:1708.08487* .

Dai, A. M., Olah, C. & Le, Q. V. (2015), 'Document Embedding with Paragraph Vectors', *arXiv preprint arXiv:1507.07998* .

Defferrard, M., Bresson, X. & Vandergheynst, P. (2016), Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, *in* 'Advances in Neural Information Processing Systems', pp. 3844–3852.

Deng, L. & Yu, D. (2014), 'Deep Learning: Methods and Applications', *Foundations and Trends® in Signal Processing* **7**(3–4), 197–387.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', *arXiv preprint arXiv:1810.04805* .

Dutil, F., Cohen, J. P., Weiss, M., Derevyanko, G. & Bengio, Y. (2018), 'Towards Gene Expression Convolutions using Gene Interaction Graphs', *arXiv preprint arXiv:1806.06975* .

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. & Adams, R. P. (2015), Convolutional Networks on Graphs for Learning Molecular Fingerprints, *in* 'Advances in Neural Information Processing Systems', pp. 2224–2232.

Elspas, B. & Turner, J. (1970), 'Graphs with circulant adjacency matrices', *Journal of Combinatorial Theory* **9**(3), 297–307.

Fawcett, T. (2006), 'An introduction to ROC analysis', *Pattern Recognition Letters* **27**(8), 861–874.

Fey, M., Eric Lenssen, J., Weichert, F. & Müller, H. (2018), SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 869–877.

Fiedler, M. (1973), 'Algebraic connectivity of graphs', *Czechoslovak mathematical journal* **23**(2), 298–305.

Fortunato, S. & Hric, D. (2016), 'Community detection in networks: A user guide', *Physics Reports* **659**, 1–44.

Futreal, P. A., Coin, L., Marshall, M., Down, T., Hubbard, T., Wooster, R., Rahman, N. & Stratton, M. R. (2004), 'A census of human cancer genes.', *Nature Reviews Cancer* **4**(3), 177.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P. & Aspuru-Guzik, A. (2018), 'Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules', *ACS Central Science* **4**(2), 268–276.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT press.

Gross, J. L. & Yellen, J. (2004), *Handbook of Graph Theory*, CRC press.

Grover, A. & Leskovec, J. (2016), node2vec: Scalable Feature Learning for Networks, *in* 'Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, pp. 855–864.

Hamilton, W., Ying, Z. & Leskovec, J. (2017), GraphSAGE: Inductive Representation Learning on Large Graphs, *in* 'Advances in Neural Information Processing Systems', pp. 1024–1034.

Hamzeh, O., Alkhateeb, A., Rezaeian, I., Karkar, A. & Rueda, L. (2017), Finding Transcripts Associated with Prostate Cancer Gleason Stages Using Next Generation Sequencing and Machine Learning Techniques, *in* 'International Conference on Bioinformatics and Biomedical Engineering (IWBBIO)', Springer, pp. 337–348.

Han, J. & Moraga, C. (1995), The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning, *in* 'International Workshop on Artificial Neural Networks', Springer, pp. 195–201.

Hanley, J. A. & McNeil, B. J. (1982), 'The meaning and use of the area under a receiver operating characteristic (ROC) curve', *Radiology* **143**(1), 29–36.

Henaff, M., Bruna, J. & LeCun, Y. (2015), 'Deep Convolutional Networks on Graph-Structured Data', *arXiv preprint arXiv:1506.05163* .

Hjelmås, E. & Low, B. K. (2001), 'Face Detection: A Survey', *Computer Vision and Image Understanding* **83**(3), 236–274.

Jackson, P. & Moulinier, I. (2007), *Natural Language Processing for Online Applications*, John Benjamins.

Karlik, B. & Olgac, A. V. (2011), 'Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks', *International Journal of Artificial Intelligence and Expert Systems* **1**(4), 111–122.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P. & Neumann, M. (2016), 'Benchmark Data Sets for Graph Kernels'. `http://graphkernels.cs.tu-dortmund.de`.

Kingma, D. P. & Ba, J. (2014), 'Adam: A Method for Stochastic Optimization', *arXiv preprint arXiv:1412.6980* .

Kipf, T. N. & Welling, M. (2016), 'Semi-Supervised Classification With Graph Convolutional Networks', *arXiv preprint arXiv:1609.02907* .

Kodakateri Pudhiyaveetil, A., Gauch, S., Luong, H. & Eno, J. (2009), Conceptual recommender system for CiteSeerX, *in* 'Proceedings of the third ACM conference on Recommender systems', ACM, pp. 241–244.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), ImageNet Classification with Deep Convolutional Neural Networks, *in* 'Advances in Neural Information Processing Systems', pp. 1097–1105.

Langley, P. (1996), *Elements of Machine Learning*, Morgan Kaufmann.

Le, Q. & Mikolov, T. (2014), Distributed Representations of Sentences and Documents, *in* 'International Conference on Machine Learning', pp. 1188–1196.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep Learning', *nature* **521**(7553), 436.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation Applied to Handwritten Zip Code Recognition', *Neural Computation* **1**(4), 541–551.

Li, M., Zhang, T., Chen, Y. & Smola, A. J. (2014), Efficient mini-batch training for stochastic optimization, *in* 'Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, pp. 661–670.

Li, Q., Han, Z. & Wu, X.-M. (2018), Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, *in* 'Thirty-Second AAAI Conference on Artificial Intelligence'.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R. & Battaglia, P. (2018), 'Learning Deep Generative Models of Graphs', *arXiv preprint arXiv:1803.03324* .

Li, Y. & Yuan, Y. (2017), Convergence Analysis of Two-layer Neural Networks with ReLU Activation, *in* 'Advances in Neural Information Processing Systems', pp. 597–607.

Lu, Q. & Getoor, L. (2003), Link-based Classification, *in* 'Proceedings of the 20th International Conference on Machine Learning (ICML-03)', pp. 496–503.

Mathworks (2019), `https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html`. Accessed: 2019-08-22.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J. & Bronstein, M. M. (2017), Geometric deep learning on graphs and manifolds using mixture model CNNs, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 5115–5124.

Neumann, M., Garnett, R., Bauckhage, C. & Kersting, K. (2016), 'Propagation kernels: efficient graph kernels from propagated information', *Machine Learning* **102**(2), 209–245.

Nguyen, H. Q. & Do, M. N. (2014), 'Downsampling of Signals on Graphs Via Maximum Spanning Trees', *IEEE Transactions on Signal Processing* **63**(1), 182–191.

Niepert, M., Ahmed, M. & Kutzkov, K. (2016), Learning Convolutional Neural Networks for Graphs, *in* 'International Conference on Machine Learning', pp. 2014–2023.

Özgür, A., Vu, T., Erkan, G. & Radev, D. R. (2008), 'Identifying gene-disease associations using centrality on a literature mined gene-interaction network', *Bioinformatics* **24**(13), i277–i285.

Pal, N. R. & Pal, S. K. (1991), 'Entropy: A new definition and its applications', *IEEE Transactions on Systems, Man, and Cybernetics* **21**(5), 1260–1270.

Perozzi, B., Al-Rfou, R. & Skiena, S. (2014), DeepWalk: Online Learning of Social Representations, *in* 'Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, pp. 701–710.

Porfiri, M. & Stilwell, D. J. (2007), 'Consensus Seeking Over Random Weighted Directed Graphs', *IEEE Transactions on Automatic Control* **52**(9), 1767–1773.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G. & Schomburg, D. (2004), 'BRENDA, the enzyme database: updates and major new developments', *Nucleic Acids Research* **32**(suppl_1), D431–D433.

Schuster, M. & Paliwal, K. K. (1997), 'Bidirectional Recurrent Neural Networks', *IEEE Transactions on Signal Processing* **45**(11), 2673–2681.

Sedgewick, R. & Wayne, K. (2011), *Algorithms*, 4th edn, Addison-Wesley Professional.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B. & Eliassi-Rad, T. (2008), 'Collective Classification in Network Data', *AI Magazine* **29**(3), 93–93.

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., Amin, N., Schwikowski, B. & Ideker, T. (2003), 'Cytoscape: a software environment for integrated models of biomolecular interaction networks', *Genome research* **13**(11), 2498–2504.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K. & Borgwardt, K. M. (2011), 'Weisfeiler-Lehman Graph Kernels', *Journal of Machine Learning Research* **12**(Sep), 2539–2561.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K. & Borgwardt, K. (2009), Efficient graphlet kernels for large graph comparison, *in* 'Artificial Intelligence and Statistics', pp. 488–495.

Smoot, M. E., Ono, K., Ruscheinski, J., Wang, P.-L. & Ideker, T. (2010), 'Cytoscape 2.8: new features for data integration and network visualization', *Bioinformatics* **27**(3), 431–432.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *The Journal of Machine Learning Research* **15**(1), 1929–1958.

Stark, C., Breitkreutz, B.-J., Reguly, T., Boucher, L., Breitkreutz, A. & Tyers, M. (2006), 'BioGRID: a general repository for interaction datasets', *Nucleic Acids Research* **34**(suppl_1), D535–D539.

String (2018), `http://version10.5.string-db.org/cgi/input.pl?sessionId=En4EqYMsqDOW&input_page_show_search=on`. Accessed: 2019-08-22.

Szklarczyk, D., Franceschini, A., Wyder, S., Forslund, K., Heller, D., Huerta-Cepas, J., Simonovic, M., Roth, A., Santos, A. & Tsafou, K. P. (2014), 'STRING v10: protein–protein interaction networks, integrated over the tree of life', *Nucleic Acids Research* **43**(D1), D447–D452.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. & Mei, Q. (2015), LINE: Large-scale Information Network Embedding, *in* 'Proceedings of the 24th international conference on World Wide Web', pp. 1067–1077.

Ugander, J., Karrer, B., Backstrom, L. & Marlow, C. (2011), 'The Anatomy of the Facebook Social Graph', *arXiv preprint arXiv:1111.4503* .

Vaishnav, N. & Tatu, A. (2016), 'A Graph Downsampling Technique Based on Graph Fourier Transform', *arXiv preprint arXiv:1612.07542* .

Van der Maaten, L. (2009), 'A New Benchmark Dataset for Handwritten Character Recognition', *Tilburg University* pp. 2–5.

Verma, S. & Zhang, Z.-L. (2018), 'Graph Capsule Convolutional Neural Networks', *arXiv preprint arXiv:1805.08090* .

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. (2010), 'Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion', *Journal of Machine Learning Research* **11**(Dec), 3371–3408.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R. & Borgwardt, K. M. (2010), 'Graph Kernels', *Journal of Machine Learning Research* **11**(Apr), 1201–1242.

Wallach, H. M., Murray, I., Salakhutdinov, R. & Mimno, D. (2009), Evaluation Methods for Topic Models, *in* 'Proceedings of the 26th Annual International Conference on Machine Learning', ACM, pp. 1105–1112.

Warde-Farley, D., Donaldson, S. L., Comes, O., Zuberi, K., Badrawi, R., Chao, P., Franz, M., Grouios, C., Kazi, F. & Lopes, C. T. (2010), 'The GeneMANIA prediction server: biological network integration for gene prioritization and predicting gene function', *Nucleic Acids Research* **38**(suppl_2), W214–W220.

Weston, J., Ratle, F., Mobahi, H. & Collobert, R. (2012), Deep Learning via Semi-Supervised Embedding, *in* 'Neural Networks: Tricks of the Trade', Springer, pp. 639–655.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Yu, P. S. (2019), 'A Comprehensive Survey on Graph Neural Networks', *arXiv preprint arXiv:1901.00596* .

Xiao, F., Honma, Y. & Kono, T. (2005), 'A simple algebraic interface capturing scheme using hyperbolic tangent function', *International Journal for Numerical Methods in Fluids* **48**(9), 1023–1040.

Xu, Z., Ke, Y., Wang, Y., Cheng, H. & Cheng, J. (2012), A Model-based Approach to Attributed Graph Clustering , *in* 'Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data', ACM, pp. 505–516.

Yang, Z., Cohen, W. W. & Salakhutdinov, R. (2016), 'Revisiting Semi-Supervised Learning with Graph Embeddings', *arXiv preprint arXiv:1603.08861* .

Yilmaz, E. & Aslam, J. A. (2006), Estimating Average Precision with Incomplete and Imperfect Judgments, *in* 'Proceedings of the 15th ACM International Conference on Information and Knowledge Management', ACM, pp. 102–111.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L. & Leskovec, J. (2018), Graph Convolutional Neural Networks for Web-Scale Recommender Systems, *in* 'Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining', ACM, pp. 974–983.

Yu, M.-C., Wu, Y.-C. J., Alhalabi, W., Kao, H.-Y. & Wu, W.-H. (2016), 'ResearchGate: An effective altmetric indicator for active researchers?', *Computers in Human Behavior* **55**, 1001–1006.

Zhang, M., Cui, Z., Neumann, M. & Chen, Y. (2018), An End-to-End Deep Learning Architecture for Graph Classification, *in* 'Thirty-Second AAAI Conference on Artificial Intelligence'.

Zhu, X., Ghahramani, Z. & Lafferty, J. D. (2003), Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions, *in* 'Proceedings of the 20th International Conference on Machine Learning (ICML-03)', pp. 912–919.

Zurek, W. H. (2018), *Complexity, Entropy and the Physics of Information*, CRC Press.

VITA AUCTORIS

NAME: SUSHA POZHAMPALLAN SURESH

PLACE OF BIRTH: INDIA

YEAR OF BIRTH: 1990

EDUCATION: Bachelor in Computer Science, Cochin University of Science and Technology