2024

# Multilabel defect classification using graph neural networks for autonomous visual inspections

MD Sazzad Hossen

# MULTILABEL DEFECT CLASSIFICATION USING GRAPH NEURAL NETWORKS FOR AUTONOMOUS VISUAL INSPECTIONS

**MD Sazzad Hossen**

**A THESIS**

**Submitted in partial fulfillment of the requirements**
**for the degree of Master of Science in Engineering**
**in**
**The Department of Electrical and Computer Engineering**
**to**
**The Graduate School**
**of**
**The University of Alabama in Huntsville**

**August 2024**

**Approved by:**

Dr. Avimanyu Sahoo, Research Advisor/Committee Chair
Dr. David Pan, Committee Member
Dr. Rahul Badani, Committee Member
Dr. Alexandar Milenkovic, Department Chair
Dr. Sankar Mahalingam, College Dean
Dr. Jon Hakkila, Graduate Dean

# Abstract

## MULTILABEL DEFECT CLASSIFICATION USING GRAPH NEURAL NETWORKS FOR AUTONOMOUS VISUAL INSPECTIONS

**MD Sazzad Hossen**

**A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering**

**Electrical and Computer Engineering**

**The University of Alabama in Huntsville**

**August 2024**

Visual inspections of safety-critical systems are crucial in reducing the risk of equipment failures, downtime, and loss of life. This nondestructive testing (NDT) method uses a portable borescope or camera along with other sensors directly or mounted on robotic platforms to inspect difficult-to-access areas with ease, minimum time, and limited cost. Although state-of-the-art visual inspection platforms are equipped with sensors from multiple modalities, the inspection tasks still require human subject matter experts to identify defects and analyze them. This jeopardizes human safety in a hazardous work environment in energy industries, as well as extending time for inspection and human error. Moreover, defect identification becomes much more challenging, especially in large machinery and structures, such as aircraft engines, concrete bridges, and buildings, because of differences in material appearance, changing lighting, different surface markings, and the possible overlap of varying defect types. In order to automate the process and address the inherent challenge of defect classification, it is imperative to employ a resilient deep-learning approach that can accurately identify the defects. In this thesis, a hybrid deep learn-

ing method for multilable defect classification from visual data by using graph neural networks (GNN), convolutional neural networks (CNN), and feedforward neural networks (FFN) is presented. The first part of the thesis provides a comprehensive review of state-of-the-art GNNs for machine vision to derive motivation for the research. The literature review describes various graph-learning approaches and the challenges associated with generating graph-structured datasets from images. The primary focus of the review is the application of GNNs in machine vision and their mathematical formulations. In the second part, the proposed defect classification methodology is presented, which diverges from conventional deep learning approaches for multilabel defect classification by harnessing the combined strengths of CNN and GNN algorithms. The core idea is to exploit CNNs for their prowess in recognizing the visual characteristics of defects and GNNs for their ability to capture the relational structures of defects, facilitating more precise differentiation. This multilabel hybrid vision GNN algorithm is validated using the open-source CODEBRIM dataset, which contains multilabel images of large-scale concrete structural defects. The model's performance in image classification is validated using the CIFAR-10 dataset, achieving 86 % accuracy during testing. Experimental results demonstrate that the hybrid architectures developed have fewer overall parameters and achieve a 16% improvement in accuracy compared to popular neural architectures for defect classification.

# Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Avimanyu Sahoo, for his invaluable guidance, support, and encouragement throughout my research. His insights and expertise were instrumental in shaping the direction and quality of this work.

I am also immensely grateful to the members of my committee for their constructive feedback and suggestions. I would also like to acknowledge Baker Hughes (industry collaborator) and the Department of Energy (DOE) for their generous funding and resources, without which this research would not have been possible. I am also profoundly grateful to the University of Alabama in Huntsville for providing an excellent academic environment and resources that facilitated my research endeavors.

Finally, I wish to express my heartfelt appreciation to my family. Their unwavering support, patience, and love have been the foundation of my strength and perseverance throughout this journey. To my parents for their endless encouragement and belief in me, and to my siblings for always being there for me, thank you.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Introduction

Visual inspections of safety-critical systems are crucial in reducing the risk of equipment failures, downtime, and loss of life. This nondestructive testing (NDT) method uses a portable borescope or camera along with other sensor modalities, directly or mounted on robotic platforms, to inspect difficult-to-access areas with ease, minimum time, and cost. Although state-of-the-art borescopes and robotic inspection platforms are equipped with sensors from multiple modalities, the inspection tasks still require human subject matter experts for defect identification and their analysis to predict the remaining useful life. This jeopardizes human safety in hazardous work environments, such as energy equipment and aerospace machinery. In addition, human intervention leads to extended inspection time and is subject to human error. A prospective solution for automated defect identification from visual data is to introduce next-generation visual sensing technologies augmented with artificial intelligence (AI).

Defect identification from visual data, such as images and videos, is a challenging problem, especially in large machinery and structures, such as gas turbines, engines, concrete bridges, and buildings. This is due to the differences in material appearance, changing lighting, different surface markings, and the possible overlap of varying defect types [1]. In real-world situations, a number of

environmental factors make it more complex to identify these defects, which include surface wetness due to weather and different non-critical surfaces like minor holes, markings, stains, or graffiti. Therefore, the identification of defects from visual data requires a robust methodology capable of learning the subtle differences and relations between defects and their backgrounds for better classification accuracy.

It is well known that deep learning algorithms have the capability to detect subtle feature variations [2] from visual data. It has been successfully employed in various domains to comprehend and analyze the hidden structures from image datasets. Convolutional neural networks (CNN) perform exceptionally well in the areas of object detection and material recognition [3]. This is primarily due to their capacity to recognize local patterns within images [2]. CNN-based models [2] are also well-suited for defect classification tasks [3] from images, especially for the classification of isolated defects, i.e., image dataset with single defect per image. However, when confronted with the complexities of multiple defects in a single image, the model performs poorly. This is because CNNs primarily rely on sliding filters for information collection, prioritizing pixel-level details. However, for spatial data, CNNs potentially disregard inter-pixel relationships while extracting the features of the images or videos. Therefore, CNNs face intrinsic difficulty in capturing the intricate interconnections and overlapping characteristics that are linked to multiple defect classes present in an image. Furthermore, CNN models may not be entirely suitable for the identification of material defects, as they tend to neglect to acknowledge the interaction that exists between

the defect and the multiscale structures in its vicinity [1]. In real-world scenarios, the characteristics of a material defect often depend on its adjacent components. Therefore, deep learning schemes that take into account the relationships among the features are necessary.

Unlike CNN, graph neural networks (GNNs) present a compelling solution to account for the relationships among the data, especially when dealing with non-Euclidean datasets like images, videos, or 3D datasets [4]. GNNs have applications in a wide range of domains where the available data is naturally in a graphical structure, such as social media networks [5], citation networks [6], and others. It has been demonstrated that GNNs are the most effective method for modeling the intricate patterns that can be found in datasets. GNN employs a message-passing algorithm that is effective in capturing features from other objects or entities in the image, resulting in a far more improved classifier for image classification. Therefore, it is evident that GNN [7] can be deployed to develop a robust model to address the multilabel visual defect classification problem.

However, using GNNs for image classification requires reformulating or representing the image data as a graph since the data set is not inherently in a graphical structure. Several approaches are available in the literature to represent image data as a graph data structure (discussed in detail in Chapter 2). For instance, the vision GNN [7] represents images as a graph by dividing each image into multiple segments or patches, allowing for the extraction of intricate relationships between image components. Vision CNN detects the local characteristics of objects, while GNN assists in identifying the global relationships between them.

Motivated by the capabilities of the GNN and CNN, in this thesis, we propose an enhanced Vision GNN with an edge convolution approach for multilabel defect classification, utilizing a hybrid network that integrates feedforward neural networks (FFN), CNN, and GNN.

## 1.1 Organaization of the Thesis

The thesis is organized as follows: Chapter 1 presents an introduction to the multilabel defect classification problem and GNN. Chapter 2 introduces the state-of-the-art GNN frameworks available in the literature in detail to highlight the significance of the proposed hybrid vision GNN. Chapter 3 presents the proposed hybrid vision GNN for multilabel defect classification and its implementation on the open source image data set, such as CODEBRIM and CIFAR-10. Finally, Chapter 4 presents the conclusions and future work.

## 1.2 Contribution of the Thesis

The contributions of the thesis are twofold. The first part provides a comprehensive review of the state-of-the-art GNN literature for machine vision, and the second part advances the defect classification models for multilabel defect classification. More specifically, the thesis makes the following contributions:

- Provides a comprehensive survey on various graph formulations or representations from image datasets.

- Summarizes various graph embedding techniques with their mathematical formulations for the first time in literature.
- Introduces an edge convolution approach within the graph-based architecture, enhancing the vision GNN model's ability to capture intricate relationships and patterns in structural defect images.
- Addresses the challenge of over-smoothing in deep graph convolutional networks by incorporating a multilayer perceptron (MLP) both before and after the graph processing blocks.
- Extends the capability of vision GNN to multilabel classification and is applied to structural defect problems in large structures.

# Chapter 2: Background of Graph Neural Network

## 2.1 Introduction

The adaptation of GNNs in computer vision has demonstrated significant potential and success in various tasks, including image classification, video classification, object detection, scene graph generation, 3D mesh construction, and vision-language query tasks. Many review papers in the literature discuss the continuous progression of various kinds of GNN [8] algorithms and their myriad number of applications [9]. Chen *et al.* [10] examined the applicability of the different GNN approaches to various categories of vision datasets. A comprehensive examination of the utilization of GNNs and Graph Transformers in the domain of vision tasks was conducted by Chen *et al.* [10]. Jiao *et al.* [11] also divided the vision task into image classification, semantic segmentation, object detection, and tracking and discussed the graph learning methods used for this application.

The GNN application for classification tasks and segmentation of 2D image datasets was succinctly described by Asif *et al.* [12]. In their exhaustive examination of GNN, Wu *et al.* highlighted the ability of GNNs to capture semantic relationships in visual question-answering datasets pertaining to human-object interaction for vision tasks [13]. Later, Zhang *et al.* discuss the graph-learning techniques used to process biomedical image datasets [14]. Senoir *et al.* presented

image captioning, visual question answering, and image retrieval techniques in their review of GNN application in the vision-language task [15]. More recently, However, Cao *et al.* [16] and Zhu *et al.* [17] provide a comprehensive explanation of the graph learning process of GNN, including a description of a few methodologies of graph generation from images, videos, and 3D point cloud data, as well as their application to a variety of vision tasks. The majority of review articles place greater emphasis on the application of GNNS to vision tasks rather than elaborating on the graph formulation procedure that leads to the diverse applications of these techniques in task-specific vision problems. A detailed list of recent review papers and their contributions is presented in Table 2.1.

Motivated by the above limitations, this chapter reviews the underlying principles of GNN, demonstrating their capacity to handle non-euclidean data sets, more specifically for vision applications. It emphasizes the challenges associated with the generation of graph structures from grid-structured image datasets, which include photos, videos, and 3D data points. The review also examines conventional approaches and recent developments in the field of graph learning in computer vision. It focuses on determining the extent to which these algorithms can improve the performance of vision tasks by utilizing visual data and extracting significant insights from the dataset.

**Table 2.1:** Contributions of the review papers available in the literature on graph learning.

| **Author(s)** | **Contribution** |
|---|---|
| Chen *et al.* [10] | Comprehensive analysis of GNNs and Graph Transformers in vision tasks. |
| Jiao *et al.* [11] | Examined graph learning techniques for image classification, segmentation, object detection, and tracking in vision tasks. |
| Asif *et al.* [12] | Review of GNN application for classification and segmentation of 2D image datasets. |
| Wu *et al.* [13] | Highlighted GNNs' semantic relationship capture in visual question-answering datasets for vision tasks. |
| Zhang *et al.* [14] | Discussed the use of graph learning in biomedical image processing. |
| Senoir *et al.* [15] | Showed GNN-based image captioning, visual question answering, and image retrieval in vision-language tasks. |
| Cao *et al.* [16], Zhu *et al.* [16] | GNN graph learning and its application to image, video, and 3D point cloud data vision tasks are explained in detail. |

## 2.2 Background on Graph Learning

In this section, we first introduce several definitions of graphs and their properties, which are necessary for graph learning. Then, we outline the general framework of the GNN learning.

### 2.2.1 Background on Graph Theory

A graph is a data structure consisting of nodes and connections (edges) between them. Mathematically, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an ordered pair, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices or nodes and $\mathcal{E}$ is the set of edges that represents the connection between nodes. The edges define the relationship between the nodes. Based on the inherent structure or topology of a graph, it can be categorized into various types, each representing distinct characteristics and relationships within the data. In the following, definitions of various types of graphs are presented.

**Definition 1** *[18](Directional Graph) A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an ordered pair in which $\mathcal{V}$ represents a collection of vertices and the set $\mathcal{E}$ comprises directed edges, with each edge represented by an ordered pair $(u, v) \in \mathcal{V}$ signifying a directed connection from the source vertex $u$ to the target vertex $v$.*

**Definition 2** *[18](Undirected Graph) An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair of vertices and the edge set $\mathcal{E}$, where the set $\mathcal{E} \subseteq \{\{u, v\} \mid u, v \in V\}$ is an unordered pair of $u$ and $v$.*

**Definition 3** *[19](Bipartite Graph) A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is referred to as bipartite if its vertex set $\mathcal{V}$ can be divided into two disjoint sets, $U$ and $W$, so each vertex in $U$ is connected to a vertex in $W$ by every edge in $\mathcal{E}$. In other words, no edge within the same partition directly connects any two vertices.*

To understand the patterns of local and intermediate connectivity in a graph, the notion of neighborhood and $k$-hop neighborhood are employed, which are defined below.

**Definition 4** *[20](Neighborhood of a graph) The neighborhood $\mathcal{N}(v)$ of a vertex $v \in \mathcal{V}$ in Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the set of vertices that are adjacent to it. For instance, in an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the neighborhood of vertex $v$ is defined as $\mathcal{N}(v) = \{u \in \mathcal{V} \mid (u, v) \in \mathcal{E} \text{ or } (v, u) \in \mathcal{E}\}$.*

**Definition 5** *[20] ($k$-hop neighborhood) A vertex's $k$-hop neighborhood in a graph, $\mathcal{G}$, represented by $\mathcal{N}_k(v)$, is the set of vertices that are reachable from $v$ through $k$ edges. Alternatively, the $k$-hop neighborhood of a vertex $v$ for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as $\mathcal{N}_k(v) = \bigcup_{u \in \mathcal{N}_{k-1}(v)} \mathcal{N}(u)$ where $\mathcal{N}_{k-1}(v)$ represents the $(k-1)$-neighborhood of vertex $v$.*

Based on the evolution of the structure of a graph, it can be classified as static or dynamic. A *static* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is characterized by a structure that does not change or evolve over time, i.e., the nodes $\mathcal{V}$ and edges $\mathcal{E}$ remain fixed [19]. The topology and properties of a static graph also remain unchanged while processing. Whereas, *dynamic graphs* add a temporal dimension to the graph by allowing nodes $\mathcal{V}$, edges $\mathcal{E}$, or the associated attributes to change over

time [19]. On the other hand, based on the similarity of the information, a graph can be classified as homogenous or heterogeneous. A *homogenous graph* is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where there is no distinctive feature between the different kinds of nodes and the edges in the graph. Hence, with regard to their types, every vertex in set $\mathcal{V}$ and every edge in set $\mathcal{E}$ are equivalent. In contrast, a *heterogeneous graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of different types of nodes, and the edges can consist of different types of relationships. A knowledge graph [21] can be considered a heterogeneous graph which can accommodate a wide range of complex relationships between entities, make them an effective tool for representing various complex information structures.

Matrix representations are a powerful and efficient way of articulating graphs' inherent and intricate properties. They can capture the essence of graph structures and provide computational benefits, allowing for seamless calculations and analysis. Below are some of the commonly used matrices in the context of graphs.

**Definition 6** *(Adjacency Matrix) The adjacency matrix $A$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a square matrix of size $|\mathcal{V}| \times |\mathcal{V}|$, where $|\mathcal{V}|$ is the cardinality of $\mathcal{V}$, i.e., the number of vertices, is defined as*

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{E}, \\ 0 & \text{if } (i,j) \notin \mathcal{E}. \end{cases} \tag{2.1}$$

Adjacency matrices for undirected graphs are symmetric, i.e., $A_{ij} = A_{ji}$, which may not be true in the case of directed graphs [22].

**Definition 7** *(Degree matrix) The degree matrix $D$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a square matrix of size $|\mathcal{V}| \times |\mathcal{V}|$ where each diagonal entry represents the number of edges incident to each vertex (referred to as the degree of the vertex).*

The degree of a vertex in an undirected graph equals the number of edges that are connected to it. In the case of a directed graph, it is common practice to make separate adjustments to the degree matrix to account for in-degrees and out-degrees [22].

**Definition 8** *(Laplacian matrix [23]) The Laplacian matrix $L$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as $L = D - A$. The normalised Laplacian matrix $L_{norm}$ can be defined as*

$$L_{norm,ij} = \begin{cases} 1 - \frac{1}{\sqrt{d_i d_j}} & \text{if } i \neq j \text{ and } (i,j) \text{ is an edge,} \\ 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

In the context of graph learning, each node is associated with certain attributes or characteristics. The values of the features for the nodes in a graph are often collectively represented as a matrix, referred to as a *graph signal matrix*. For instance, in a social network, node attributes could include information such as age, gender, interests, or the connectivity ratio between nodes. Then, the

signal matrix $X \in \mathbb{R}^{n \times d}$, where $n = |\mathcal{V}|$ is the total number of nodes in the graph, and $d$ is the number of features $x$ for each node. Each row of $X$ corresponds to a node, and each column of $X$ acts as a signal for a particular attribute shared by all nodes [23].

### 2.2.2 Overview of Graph Learning

Graph learning, or graph machine learning, is a powerful approach that uses graphs as a data structure to understand and leverage the complex relationships inherent in the data. It enables tasks such as node classification, edge classification (link prediction), and graph classification, with wide-ranging applications across various domains. The graph-learning process can be broadly categorized under the following three steps: 1) graph formulation, 2) graph embedding, and 3) graph classification tasks, as shown in Figure 2.1.

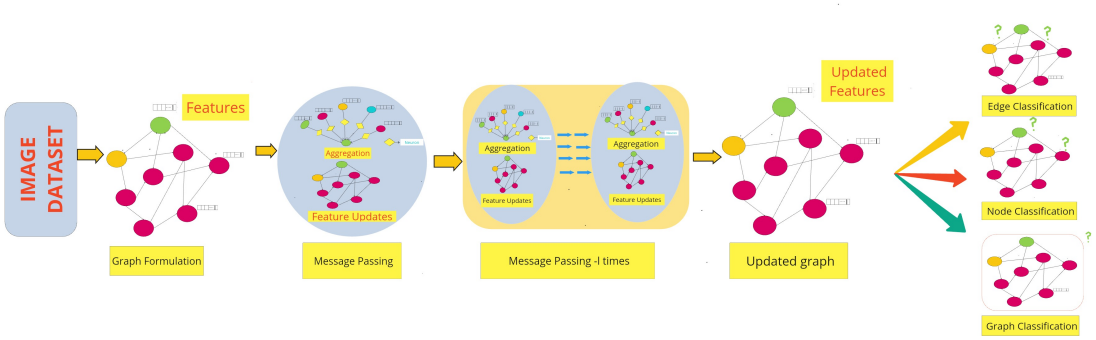



**Figure 2.1:** A generalized architecture of graph learning with various tasks.

**(a) Graph Formulation/Representation**

In domains such as social networks, recommendation systems, biological networks, natural language processing, and financial networks, the data are inherently in a graphical structure depicting the relations among the data points. Therefore, the graph formulation step may not be necessary. However, the image datasets used in machine vision applications do not have a graphical structure. Therefore, one of the most critical steps in graph learning for machine vision is graph formulation (discussed in detail in Section 2.3).

**(b) Graph Embedding**

Graph embedding transforms the features of the graph data (nodes and edges) for machine learning, such as developing a classifier for nodes or edges. It can be achieved with or without using a neural network. The two most prevalent non-neural network techniques for node or edge embeddings are 1) similarity-based and 2) graph kernel-based, discussed in detail in Section **??**. On the other hand, GNNs [4] use neural networks to learn representations (embeddings) for nodes, edges, or entire graphs through a process known as message passing. Message-passing aggregates and transforms the neighborhood node or edge features to update each node or edge information. Some of the important GNN architectures for message-passing are graph recurrent neural networks (GRN), graph convolutional networks (GCN), and graph attention networks (GATs), discussed in detail in Section **??**.

Similar to traditional machine learning approaches, graph learning or GNN can be guided by all three supervision techniques (supervised, semi-supervised,

and unsupervised) by introducing appropriate loss functions based on the tasks. Supervised graph learning uses the labels, categories, or attributes of all the entities (nodes and edges) [24] for training the model. Supervised learning is common in scenarios where obtaining labeled data is simple and includes tasks like node classification and edge classification. In semi-supervised learning, a small subset of nodes or edges is labeled, while the majority remains unlabeled. Semi-supervised learning can be performed in two ways: 1) transductive and 2) inductive. The transductive approach makes predictions for nodes that were part of the original graph during training, and the model also predicts labels for the unlabeled nodes during testing [24]. In an inductive approach, the model is designed to generalize newly introduced, unlabeled nodes with the same distribution as the training set [24]. This approach is used when the graph dataset is dynamic, and the graph's structure needs to be updated constantly. In unsupervised graph learning, the model trains on the input data sets without explicit labels to identify the characteristics of the data without any explicitly labeled data. The popular unsupervised graph learning task is node clustering [25] and dimensionality reduction of the graphs [26].

**(c) Graph Learning Tasks**

As discussed above, the primary goal of graph learning, or GNNs is to perform node, edge, and graph-level tasks. Node classification is a *node-level task* in which the primary goal is to categorize or label each node in a graph according to its characteristics or connections [27] in a supervised learning framework. Node clustering is also a node-level task in an unsupervised learning framework that

organizes nodes in a graph into disjoint clusters. The nodes in the same cluster are more similar than the nodes in other clusters. Finding cohesive groups in biological or social networks and exposing the underlying structures of large, complex datasets can both benefit from clustering. Another type of node-level task is node regression, in which each node is assigned a dynamic value that can represent the quantities associated with the node [27].

Similarly, edge classification is an edge-level task in which particular labels, values, or types are assigned to edges based on their connections [28]. Determining whether or not there is an edge between two nodes allows edge-level tasks to be extended to link prediction tasks between nodes. For example, in recommendation systems, link prediction is used to forecast possible connections between users and items, whereas in social networks, it is used to forecast upcoming friendships or teamwork. On the other hand, graph-level tasks, such as graph classification, entail assigning labels to the entire graph in terms of overall label or category. Graph classification aids in comprehending the global properties and structures of the entire network [27]. For example, graph classification can be used to categorize molecular graphs into different chemical or biological classes based on their structures. Graph matching is also a graph-level task that refers to finding similar graph structures in other graph datasets. It entails determining the correspondence or similarity between nodes and edges in two different graphs. Graph matching can be used to learn the process of identifying structural similarity in visual datasets.

In the following sections, we present a detailed review of the above-mentioned graph learning steps with their mathematical formulation and summarize the results, particularly in the machine vision framework.

## 2.3 Graph Formulation/Representation

In this section, we have focused on discussing graph formulation approaches, such as pixel, superpixel, and $k$-neigborhood-based, from the existing literature. The graph formulation approaches are presented in greater detail with their mathematical intuition.

### 2.3.1 Graph Construction using Pixels

Although images are non-euclidean data sets, it is still possible to create a graph by utilizing each pixel's spatial and semantic relations with others. A deep learning model referred to as "Pixel to Graph" introduced in [29] to detect the vertices and edges of a graph formed from an image dataset in an end-to-end manner. During the training phase, bounding boxes are formed around objects in the image. In this approach, the vertices and edges of the graph are represented using instances of bounding boxes and bounding box-to-bounding box relationships within an image. The model does not require proposed bounding boxes in the testing phase and can produce detection directly from the image by forming a graph from the image [29].

For a given directed graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex $v_i \in \mathcal{V}$ is located at the center of the bounding box $(x_i, y_i)$ in the image, and edges are located at the

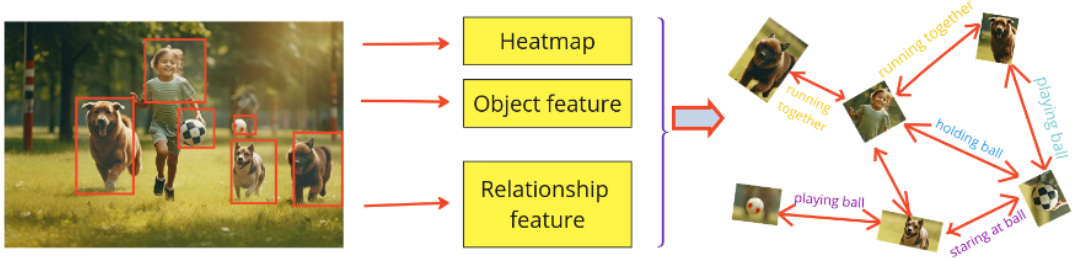



**Figure 2.2:** Generating graph form pixels.

midpoint of the target vertex and source vertex as follows

$$\left( \left\lfloor \frac{x_s + x_t}{2} \right\rfloor, \left\lfloor \frac{y_s + y_t}{2} \right\rfloor \right). \tag{2.3}$$

The corresponding feature vector for each node of the graph follows a separate fully connected layer for the prediction of each element of the graph, and a stacked hourglass network [30] is used for combining the global and local relationship information. By combining the heatmap (intensity or distribution of certain data points or features across the image) feature, individual node feature, and relationship feature, the presence of an object and the relationship (edges) between them can be determined.

### 2.3.2 Superpixel-based Graph Construction

Superpixels are groups of adjacent pixels that share image characteristics such as colors, textures, and/or others. Superpixel segmentation is the procedure of dividing images into coherent and homogeneous regions where the characteristics of the pixels within each region are identical [31]. Hyperspectral images (HSI) can be analyzed more accurately using superpixel segmentation [31]. An

HSI comprises numerous spectral bands, typically hundreds of bands captured at a continuous wavelength. The spatial structures of HSI have been investigated by superpixel segmentations, considering the spectral-spatial characteristics [31]. Simple Linear Iterative Clustering (SLIC) [32] is used to segment the image into superpixels based on the spatial and spectral similarity of the pixels. An association matrix $Q$ is defined based on the characteristics of the pixels to represent the relationship of pixels $XP_i$ and superpixels, with each entry of the matrix $q_{i,j}$ representing whether a pixel $i$ belongs to the superpixels $j$, given by

$$Q_{i,j} = \begin{cases} 1 & \text{if } XP_i \in SP_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

The superpixels are represented by $SP_j$, and the $i$-th pixel is represented by $XP_i$. Now, to formulate the graph for HSI dataset, a graph encoder is applied to transform the hyperspectral data into a vector representation, which is used as a node in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [31]. The process is as follows

$$V = \text{Encode}(XP; Q) = \hat{Q}^T \cdot \text{Flatten}(XP). \tag{2.5}$$

Based on how close the superpixels are to one another, an adjacency matrix $A$ is created, with 1 representing an adjacent pair of superpixels and 0 representing an opposite pair. This matrix representation guides the graph formation using the superpixels [31].

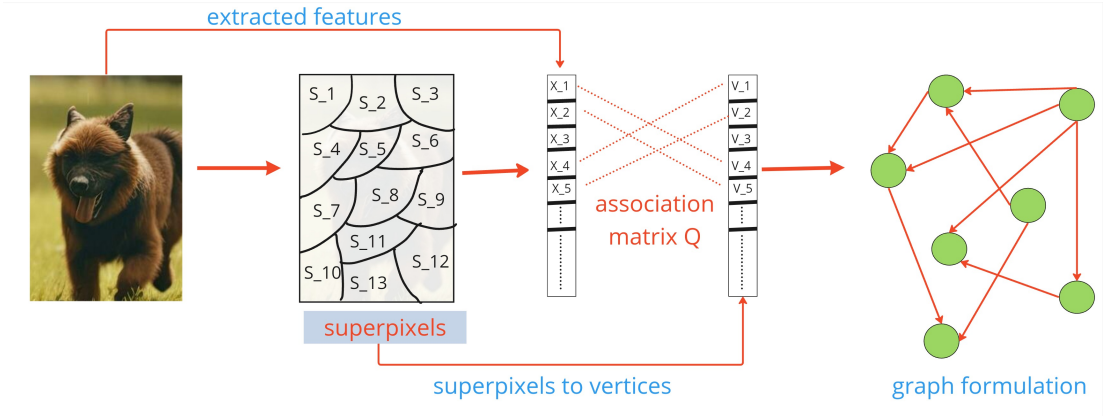



**Figure 2.3:** Graph formulation using superpixels.

The capabilities of capturing the semantics of datasets by the graph largely depend on the quality of the superpixels. There are various ways of formulating superpixels from an image. For example, the Normalized Cuts (NCUTS) algorithm uses segments' total similarity and dissimilarity for the superpixel formation process and transfers the problem to an eigenvalue problem for efficient computation. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is formed by taking each pixel as a node and defining the edge weight $w_{ij}$ between nodes $v_i$ and $v_j$ as the product of a feature similarity term and spatial proximity, given by

$$w_{ij} = \begin{cases} e^{-\frac{\|F(i)-F(j)\|^2}{2\sigma^2}} e^{-\frac{\|SP_i-SP_j\|^2}{2\sigma^2}} & \text{if } \|SP_i - SP_j\|^2 < r, \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

where $SP_i$ is the spatial location of the node $v_i$, r is a threshold value and $F(i)$ is a feature vector based on intensity, color, or texture information of the $i$-th node.

This model recursively calculates normalized cuts for the pixel graph to produce a superpixel graph [33]. Felzenszwalb and Huttenlocher present a minimum spanning tree (MST)-based segmentation technique that connects components incrementally until a predetermined condition is met, which enables computationally efficient superpixel formation [34]. Moore *et al.* effectively generate superpixels by dividing the image into vertical and horizontal strips and finding the shortest path between these strips [35]. Zhang *et al.* use graph cuts instead of finding the shortest path between the overlapping strips [36].

The above-described super-pixel algorithm produces varying sizes of segmentation. Olga *et al.* present a superpixel partitioning problem within the context of energy minimization. They employ an energy function to optimize graph cuts, which leads to the algorithm generating image segments of similar sizes [37]. Ming *et al.* enhance the process by incorporating the concept of entropy rate into the graph-cut minimization step. This inclusion promotes the formation of clusters that are both compact and homogeneous. Additionally, they introduced a balancing constraint function to encourage the creation of clusters with similar sizes [38]. However, all of the above algorithms tend to converge towards centroids over multiple iterations and use a distance map with dimensions comparable to the input pixel size, leading to memory consumption. Taking the intensity difference of the pixel can help in achieving better segmentation with less computation. Felzenswalb *et al.* proposed an efficient graph-based image segmentation method that relies on the intensity differences between region boundaries and neighboring pixels within the same region [39].

All the methodologies described above are simple iterative clustering algorithms for forming superpixels. The Simple Non-Iterative Clustering (SNIC) algorithm [40] can reduce the computational cost of the simple iterative clustering algorithm (SLIC) described above by utilizing a single iteration. SNIC outperforms SLIC by explicitly enforcing connectivity from the start, removing the need for multiple $K$-means iterations. SNIC improves computational efficiency even more by restricting distance calculations to square regions centered on centroids. This way, connectivity is maintained even if the centroid moves [40]. Peer *et al.* propose a preemptive SLIC algorithm, which is a watershed segmentation-based iteration of the SLIC algorithm that operates at a faster rate [32].

QuickShift can be used for faster segmentation, which falls under the category of local mode-seeking algorithm, as it can operate on 5D space by operating both on color information and locations of pixels in the image. Its ability to eliminate background data makes the algorithm much more efficient in image segmentation tasks, which resulted in the formation of superpixels [41, 42]. The formed superpixels are used to create a graph from the image, which is a necessary step for graph learning algorithms in image classification. For example, creating a hypergraph to detect the variation in multi-modal remote sensing images is the most suitable approach for capturing the local and global dependencies of image data sets, which is crucial for evaluating subtle changes in image features. Yuli proposes a model for unsupervised multimodal change detection (MCD) for remote sensing images that is based on the self-similarity of image [43]. The model employs the similarity between the superpixels of the image to form the graph,

and the objective of the graph construction is to capture the global and local information of the superpixels of the image. The objective function is defined as

$$\min_{W} \sum_{i=1}^{NS} \sum_{j=1}^{NS} \frac{(SP_i - SP_j)^2}{2 \cdot pb_{i,j}} + \sum_{i=1}^{NS} \frac{\alpha_i \|W_i\|^2}{2} + \beta g(\epsilon), \tag{2.7}$$

where $SP$ represents superpixels, $W$ is the weight matrix, NS is the number of superpixels, $pb_{i,j}$ is the probability matrix for superpixels and $\beta g(\epsilon)$ is the penalty term.

The initial term in (2.7) captures the local structures by quantifying the similarity between superpixels $SP_i$ and $SP_j$, with the similarity being weighted by $pb_{i,j}$. On the other hand, the second term captures the global structures by evaluating the relationships between each superpixel [43]. This evaluation is subject to regularization to promote sparsity in the weight matrix $W$. The penalty term $\beta g(\epsilon)$ is includedfor regularization. The matrix $W$ can be used to determine node connectivity. The superpixels can be connected as neighbors with probabilities $pb_{i,j}$, where $i = 1, \ldots, NS$, given there is NS number of superpixels that are represented as nodes in the graph. These probabilities satisfy $0 \leq pb_{i,j} \leq 1$ and $\sum_{i=1}^{NS} pb_{i,j} = 1$ [43]. With the learned probability matrix $pb_{i,j}$ [43], we can construct the hypergraph $\mathcal{G}_h = \{\mathcal{V}_h, \mathcal{E}_h, pb_h\}$. The vertex set $\mathcal{V}_h$ is defined as $\mathcal{V}_h = \{SP_1, \ldots, SP_N\}$. Each superpixel $SP_i$ corresponds to a hyperedge $e_i$. Specifically, $SP_i$ treated as a center, and connect $SP_i$ and its neighbors $SP_j$ in $pb_{i,j}$ to generate a hyperedge $e_i$ as follows [43]

$$e_i = \{SP_i\} \cup \{SP_j \mid pb_{j,i} \neq 0, j = 1, \ldots, NS\}.$$

A minimum-cut algorithm [44] on the graph formed by the image dataset can also be used for image segmentation. The minimum graph cut algorithm is to form $k$ sub-graphs from a graph $\mathcal{G}$ datasets; the largest inter sub-graph maximum flow is minimized among all possible $k$ partitions of graph $\mathcal{G}$ [44]. The minimum cut method is biased to construct small components of the graph datasets. The normal cut algorithm solved these biases by considering the total dissimilarities and similarities of components in the graphs [33].

### 2.3.3 K-neighborhood-based (KNN) Graph Construction

KNN graph is a directed graph where the nodes represent images in the data sets, and the edges between the nodes are established based on similarity or distance metrics among a controlled number of nodes. The KNN graph is constructed using the divide and conquer method, which is used to initiate a base approximate neighborhood graph and then ensemble them together to obtain the comprehensive graph for the datasets.

Given a set of data points, the algorithm first starts with the recursive partition of the data points into subsets until the cardinality of the subset reaches a threshold value, denoted as $g$. The suitable feature vector from the image data sets can be achieved using Scale-Invariant Feature Transform (SIFT) [45, 46, 47] and Global Image Structure (GIST) [48]. A neighborhood graph is constructed by representing the data points as nodes $\mathcal{V} = D$, where $D = (d_i, d_j, ..)$ are data points. The edges between the nodes are established by computing the similarity metric and distance of the feature vector of every node. Denoting the distance

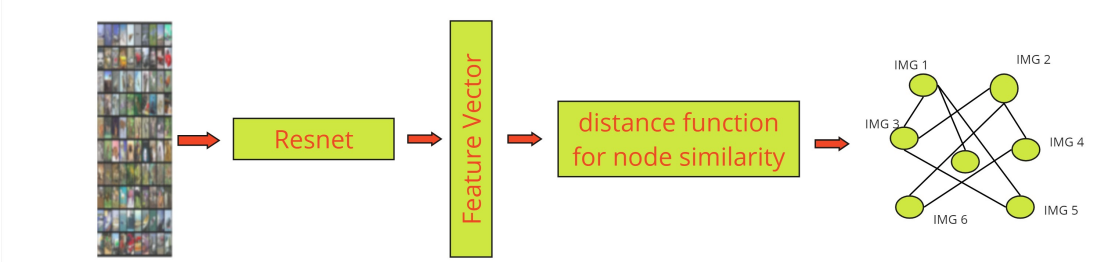



**Figure 2.4:** K-Neighborhood graph formulation.

or similarity metric as $dist(d_i, d_j)$, an edge can be established if $dist(d_i, d_j) \leq g$, where $g$ is a threshold value, as shown in Figure 2.4. In order to obtain an optimal neighborhood graph, the algorithm employs a strategy of minimizing the occurrence of isolated data points by continuously dividing the dataset. As the number of divisions increases, the influence of the neighborhood diminishes in the formation of the neighborhood graph. Consequently, a neighborhood propagation method is utilized, which considers all potential neighborhood pairs, shown in Figure 2.5, based on previously identified neighbors [49]. As the number of random divisions increases, a greater number of neighbors of $p$ are covered as shown in Figure 2.5 (a). The process of propagating neighborhood information through intermediate points from $p$ to $o$ by accessing the neighborhood of $q$ [49], as shown in Figure 2.5 (b). The iterative process of adding new neighbor points ends when the maximum time for visiting a data point is reached [49].

Furthermore, KNN graphs can also be formulated from clustered data points. The clustering problem is a combinatorial optimization problem that involves partitioning a given set of $N$ data points into $M$ clusters. This partitioning is based on the shared property of the data points, which determines


**Figure 2.5:** Formulation of K-neighborhood graph .

their contribution to the common property of the same cluster. Each data point is initially assigned to a distinct cluster by the algorithm, which performs clustering by merging clusters from the neighboring graph until the desired number of clusters is achieved [50]. The measurement of similarity between data points can be facilitated using the Locality-Sensitive Hashing (LSH) model, which uses a hash function with locality-sensitive properties such as similarity measurement of data points in high-dimensional space. The model's goal is to partition similar data points into equal-sized buckets, which results in a significant improvement in time complexity of $O(l(d + \log n)n)$, where $d$ represents dimensionality and $l$ is usually a small constant [51].

However, the process of partitioning the data sets necessitates the model relying on offline data sets, rendering it challenging to execute on dynamic data sets that are constantly being updated in real time. In contrast, the dynamic continuous indexing (DCI) model offers a solution to this limitation by consistently indexing data points through the ordering and creating a dynamic neighborhood graph [52]. The transition from discretizing the vector space to employing a continuous indexing approach improves the model's ability to adapt to changing datasets in real-time situations. To address the challenge of computational cost

due to the redundancy in distance calculation, the Recursive Lanczos Bisection (RSB) method for constructing the KNN graph introduces parameterized region processing during the divide-and-conquer procedure. This approach improves the model's flexibility by using recursive spectral bisection for data partitioning. A hash table is strategically used throughout the divide-and-conquer process to reduce redundancy in distance calculations, improving the algorithm's efficiency [53].

Repetitive data and feature correlation methods make it more challenging to identify redundancy in the data point density calculation. To address the issue, the unsupervised feature selection method [54] uses principal component analysis (PCA) to transform the original co-related data into uncorrelated data with orthogonal features. The model then constructs a weighted bipartite graph consisting of two sets of features: the original features and the non-orthogonal features. The edges of the bipartite graph are defined by the similarity between these two sets of features. This approach tackles redundancy in data point distance calculations and leverages unsupervised feature selection [54].

Weighting the edges of a constructed graph is a crucial and demanding undertaking in graph construction, as the assigned weights represent the degree of similarity between two nodes. The Gaussian kernel similarity (GKS) can be employed to measure the degree of similarity. This approach involves mapping a connected graph onto a line to keep connected points as close together as possible [55]. A heat function is defined to calculate the wights between two points, and

the heat kernel [56] is defined as

$$W_{ij} = e^{-\frac{\|d_i - d_j\|^2}{2\tau^2}}, \tag{2.8}$$

where $W_{ij}$ is a weighted matrix, $\tau$ is a scaling parameter, and d denotes data points.

The shortcoming of the GKS model is that it is sensitive to parameter variance and expensive to find optimal parameters. To address the issue, Wang and Zhang proposed to construct a graph with a set of overlapped linear neighborhood patches, and the edge weights in each patch are then computed by neighborhood linear projection [57]. The model requires a separate phase in order to estimate the neighborhood patches, which consequently leads to an increase in the time complexity. In order to address this issue, a unique methodology named sparsity-induced similarity (SIS) is employed, in which each data point is decomposed into an $\mathcal{L}_1$ sparse linear combination of the remaining data points [58]. The coefficients of the decomposition are designed to retain the information of neighboring data points, thereby enhancing their similarity [58]. This model does not need the Euclidean distance measurement to determine neighborhood sets.

## 2.4 Graph Embedding/Representations

In this section, we provide an overview of prevalent techniques for learning graph representations (embedding) in the context of visual data processing. These approaches encompass non-neural network- and neural networks-based

approaches. The classification diagram of the graph embedding/representation methods is shown in Figure 2.6.

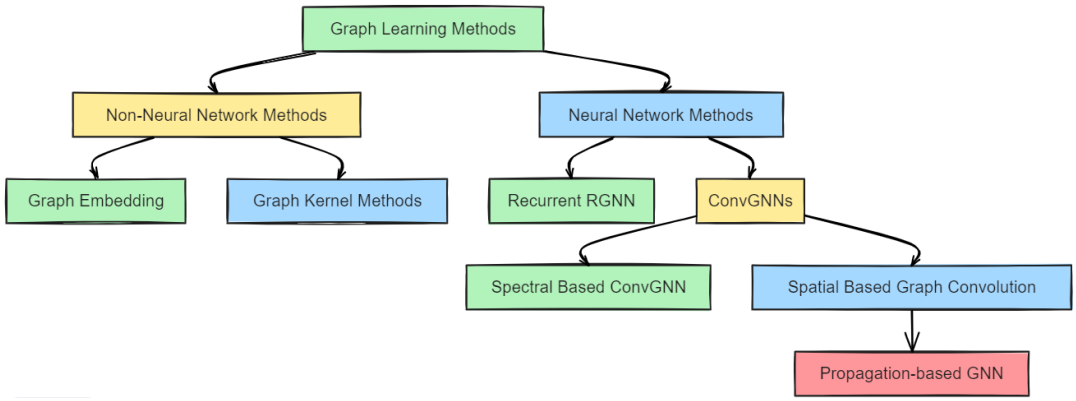



**Figure 2.6:** Overview of graph learning methods.

The non-neural network-based approaches can be classified under two heads: 1) graph embedding and 2) kernel-based approaches. A detailed description of both approaches is presented below.

### 2.4.1 Graph Embedding

The goal of graph embedding is to convert the complex structure and properties of the graph into a predetermined low-dimensional vector space while preserving graph properties, such as structural information, node attributes, and topological features. Most literature quantifies graph properties using proximity measures such as first- and higher-order proximities [24]. First-order proximity involves representing the similarity between nodes based on their direct connections, such as the similarity of nodes that share an edge. Higher-order proximity refers to the similarity of nodes that extend beyond their immediate neighbors,

such as indirect connections or interactions through multiple paths in the graph. The problem of graph embedding can be formulated as an optimization problem such that most similar nodes stay closer in the low-dimensional space. Mathematically, the objective function can be defined as [59]

$$\mathcal{L} = \min \sum_{i \neq j} (Z_i - Z_j)^2 S_{ij} = \min Z^T L Z, \tag{2.9}$$

where $Z_i$ and $Z_j$ are the embeddings of the $i$-th and $j$-th nodes, respectively, $S_{ij}$ is the defined similarity or node approximity between nodes $v_i$ and $v_j$, and $L = D - A$ is the graph Laplacian [60], where $D$ is the diagonal matrix with $D_{ii} = \sum_{j \neq i} S_{ij}$. From (2.9), the optimal embedding $Z^*$ can be derived as

$$Z^* = \arg\min_{i \neq j} \sum_{i \neq j} (Z_i - Z_j)^2 S_{ij} = \arg\min_{Z} Z^T L Z. \tag{2.10}$$

The higher the value of $D_{ii}$, the greater the importance of $Z_i$. The similarity measure $S_{ij}$ weights the quadratic form $(Z_i - Z_j)^2 S_{ij}$ [59], which implies that the optimization process places more emphasis on maintaining the similarity of nodes with a higher $S_{ij}$. After putting a constraint $Z^T D Z = 1$ on the objective function to remove arbitrary scaling, the optimal embedding in (2.10) is reduced to

$$Z^* = \arg \max_{Z^T D Z = 1} \frac{Z^T S Z}{Z^T D Z}. \tag{2.11}$$

The objective function (**??**) is designed in a way to embed the entities in the test phase, making this a transductive learning process [60].

Designing a linear function $Z = X^T a$, where $X$ is the node embedding or signal, for inductive graph embedding can help embed the node. So, finding the optimal $a$ involves minimizing the objective function

$$a^* = \arg\min_{a \neq j} (a^T X_i - a^T X_j)^2 S_{ij} = \arg\min_a a^T X L X^T a. \tag{2.12}$$

With the constraint $a^T X D X^T a = 1$, $a^*$ can be computed from (2.12) as

$$a^* = \arg\min_a \frac{a^T X L X^T a}{a^T X D X^T a} = \arg\max_a \frac{a^T X S X^T a}{a^T X D X^T a}. \tag{2.13}$$

So we can consider the problem of finding the optimal $a^*$ as finding the Eigenvectors for the maximum eigenvalue of $X W^T X^T a = \lambda X D X^T a$ [61].

The embedding method can be optimized more by finding the best similarity function for the nodes. Matrix factorization [59] can also be adapted to directly factorize the node proximity matrix. The objective function to preserve the node's proximity can defined as

$$\mathcal{L} = \min \|S - Z Z_c^T\|_F^2, \tag{2.14}$$

where $Z \in \mathbb{R}^{|V| \times d}$ is the node embedding, and $Z_c \in \mathbb{R}^{|V| \times d}$ is the embedding for the context nodes [62]. With the node embedding, a classifier can be made for the node-level, edge-level, or graph-level task.

### 2.4.2 Graph Kernel Methods

Graph Kernel Methods refers to learning the similarity of data points using kernels. The graph kernel method uses non-linear mapping to transform the data into a higher-dimensional Hilbert space, where linear methods can be applied for graph processing [63].

Given a data point $\mathcal{X} = \{p_1, p_2, p_3, \ldots\}$, where the data points are graphs or subgraphs, it can be transferred to Hilbert space by a graph kernel function $k$ using a feature map $\phi : \mathcal{X} \to \mathcal{H}_k$, where $\mathcal{H}_k$ is a Hilbert space [64]. A graph kernel can be defined as $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^k$, where a feature map $\phi : \mathcal{X} \to \mathcal{H}_k$ exists in such that for $x, y \in \mathcal{X}$, the kernel function $k(x, y) = \langle \phi(x), \phi(y) \rangle$ [64]. A gram matrix $K$ [64] that is positive semidefinite for every pair of data points in $\mathcal{X}$ can also represent the kernel function. The element of the $K_{ij}$ is the kernel value $K_{ij} = K(x_i, x_j)$ between the individual data points. For $\mathcal{X} = \mathbb{R}^d$ and $\phi(x) = x$, the kernel can be defined by the dot product, $K(x, y) = x^T y$. Here, the kernel measures the similarity of the data points.

To minimize the distance between the data points in transformed space, the objective function can be written as

$$\begin{aligned} \beta^* &= \arg \min_{\substack{\beta^T K, \\ \beta = \alpha}} \sum_{i \neq j} (\beta^T K_i - \beta^T K_j)^2 S_{ij} \\ &= \arg \min_{\beta^T K \beta = \alpha} \beta^T K L K^T \beta, \end{aligned} \tag{2.15}$$

where $K_i$ is the $i$-th column vector of the kernel gram matrix $K$ and $\beta^*$ is the optimal weights. Without explicitly computing the transformed feature vectors, algorithms can be developed to operate implicitly in the higher-dimensional space using the kernel function. Since it allows the algorithm to avoid the computational burden associated with high-dimensional data, working with kernels is sometimes called the "kernel trick" [65].

Handcrafted features or intermediate processing steps are needed for non-neural network-based graph embedding. This limits their expressive power, particularly when dealing with complex or multidimensional data. In addition, the ability of kernel methods to capture non-linear relationships between input variables is intrinsically limited, particularly on tasks involving complex data distributions or non-linear decision boundaries, and can become computationally intensive. The emergence of neural network-based graph learning techniques has facilitated complex graph-structured data analysis and processing across various domains [66]. Furthermore, they are highly scalable. The next section discusses neural network-based graph representation (embedding).

On the other hand, neural networks-based approaches are used to embed the features for classification tasks using an end-to-end machine-learning approach, which will be discussed next.

### 2.4.3 Graph Recurrent Neural Network (GRN)

Graph recurrent neural networks (GRN) paved the way for developing GNNs, which can capture both temporal and spatial features. Given a graph,

$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{W} : \mathcal{E} \to \mathbb{R}$ is a weight function that denotes the weights of the edges in $\mathcal{E}$. The weight functions measure the connection's strength. A graph shift operator (GSO) $O \in \mathbb{R}^{N \times N}$ can be introduced to perform a linear local map on the graph signal $X$. Each node's state or signal $X_t$ can change as the system evolves, depending on the previous state $X_{t-1}$ and the input or interaction at the current step [67]. The updated model can be expressed as

$$X_t = \sigma(A(O)X_t + B(O)X_{t-1}), \tag{2.16}$$

where $\sigma$ is the sigmoid activation function, and $A(O)$ and $B(O)$ are linear transformations involving the graph shift operator $O$. This generic architecture for updating the hidden state of the graph is known as GRN. Scarselli *et al.* [**?** ] extended the node hidden states update scheme for more types of graphs, such as cyclic and undirected graphs. They recurrently update the hidden state $h$ of a node as follows

$$h_i^{(t+1)} = \sum_{v_j \in N(v_i)} f\left(x_i, x_j, x_{e_{ij}}, h_j^{(t)}\right), \tag{2.17}$$

where $\mathcal{N}(v_i)$ represents the neighborhood of node $v_i$, $f(\cdot, \cdot, \cdot, \cdot)$ is a feedforward neural network, $x_i \in \mathbb{R}^d$ denotes the features at $v_i$, $x_{e_{ij}} \in \mathbb{R}^c$ denotes the features at the edge between $v_i$ and $v_j$, and $t$ is the iteration number.

### 2.4.4 Convolutional Graph Neural Networks (ConvGNNs)

Graph Convolution is another approach used in GNN for message-passing, i.e., to aggregate and transform the feature information of nodes and edges using a

defined filter. Since graph data is non-Eucledian, traditional convolution can not be performed in graphs. Graph convolution addresses this challenge by designing convolutions that respect the graph structure. Graph convolution is designed as the weighted sum of the shifted version of the signal or feature update of the state. The mathematical expression is as follows

$$A(S) = \sum_{i=0}^{i-1} a_i X_i, \tag{2.18}$$

where the variable $S$ denotes state, influencing the matrix $A(S)$, which is constructed as a linear combination of matrices with coefficients $a_k$. The shifted version of the signal refers to the signal values at different nodes in the graph being weighted and summed to produce the updated feature state. Unlike GRN, which limits how nodes can be updated, ConvGNN uses a set number of layers with different weights in each state to measure how different states depend on each other. This avoids the cyclic mutual dependencies of graph components. ConvGNNs can be categorized into two main types: i) spectral-based and ii) spatial-based [68]. Spectral convolution works in the graph spectral domain and looks at frequency characteristics, and spatial convolution works directly on spatially distributed data on the graph by collecting information from nearby nodes for tasks like node classification and graph feature extraction [69]. The two variants of graph convolution are discussed below.

#### 2.4.4.1 Spectral Based ConvGNN

Spectral-based ConvGNNs reformulated the graph convolutions by solving the graph Laplacian matrix's eigenvalues and eigenvectors. These convolutions operate in the spectral domain, allowing filters to be applied to graph signals for noise removal and graph component feature extraction. Frequency domain convolutions are commonly used in spectral-based methods to achieve Fourier transforms [69]. For the spectral-based method, the graph is considered to be undirected and a normal Laplacian matrix [69]. The Laplacian matrix $L$ is defined as

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \tag{2.19}$$

where $I_N$ is the identity matrix, $D$ is the diagonal degree matrix and $A$ is the adjacency matrix. Due to the semi-positive nature of $L$, it can be mathematically decomposed as $L = U\Lambda U^T$, where $U$ is a matrix containing eigenvectors of eigenvalues and $\Lambda$ is a diagonal matrix of eigenvalues [69]. For a graph signal $X \in \mathbb{R}^n$, the graph Fourier transform of a signal $x$ is defined as $\mathcal{F}(x) = U^T x$ projecting the input graph signal into the orthonormal space formed by the eigenvectors of the normalized graph Laplacian. This transformation facilitates the analysis of graph signals. The transformed signal can be thought of as $\hat{X} = \sum_i X_i \hat{X}_i u_i$ where $\hat{X}_i$ is new coordinates in the transformed space and the convolution on input signal $\hat{X}$ with a filter $f \in \mathbb{R}^n$ is defined as

$$X *_G f = \text{GraphConv}(\hat{X}, f) = \mathcal{F}^{-1}(\mathcal{F}(\hat{X}) \odot \mathcal{F}(f)) = U(U^T \hat{X} \odot U^T f), \tag{2.20}$$

where $\odot$ denotes the elementwise product.

By redefining the filter as $g_\theta = \text{diag}(U^T g)$, parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, the convolution can be written as

$$g_\theta * \hat{X} = U g_\theta U^T \hat{X}. \tag{2.21}$$

The filter $g_\theta(\Lambda)$ can be thought of as a function of the eigenvalues of $L$. Computing the eigenvalue for $L$ is expensive for a large graph [4]. So, a truncated expression in terms of the Chebyshev polynomial $T_k(x)$ up to the $K$-th order [4] can approximate the filter $g_\theta(\Lambda)$ as

$$g_\theta(\tilde{\Lambda}) \approx \sum_{K=0}^{K=K} \theta_k T_K(\tilde{\Lambda}) \tag{2.22}$$

with a rescaled largest eigenvalue, as $\Lambda = \frac{\tilde{\Lambda}}{2\lambda_{\max}} - I_N$ of L [70, 71].

#### 2.4.4.2 Spatial Based ConvGNN

Similar to the convolutional neural network (CNN), where convolution is performed based on the spatial relationship of pixels to the entire image dataset, convolution in a spatial-based ConvGNN is performed based on the spatial relationship of nodes in the graph. The convolution operation involves convolving the central node's representation with its neighbors' representations. Similar to weighted average filtering in CNN, the updated representation of the nodes is derived from the neighboring information of the nodes. Spatial-based ConvGNNs also borrowed the idea of information propagation from RGNNs [4]. The opera-

tion passes the message along the edge between the nodes. Spatial-based Graph Convolution GNN utilizes the propagation method to effectively tackle the specific challenges related to modeling the flow of information and interactions among nodes within a graph [4]. A propagation-based method is the process by which information or signals are propagated through a graph. Nodes in the graph update their features based on the propagation principle, which considers their relative attributes and connectivity within the graph.

ConvGNN uses the propagation principle to start learning and gives us a new way to work with graph-based data by letting convolutions happen directly on graph structures [72]. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the objective of graph learning is to learn a mapping function $f$ that transforms the raw features of each node into an enhanced update representation. Message passing and the subsequent aggregation of these messages comprise the two primary stages that make up the fundamental essence of the graph convolution process. By combining these stages, a node can achieve a more refined feature representation by combining features from its neighbors.

In the realm of graph convolution, each node denoted as $v$, plays a pivotal role in information propagation. This process involves the transmission of a message through the edges that connect node $v$ to its neighboring nodes, collectively represented as $N(v)$. This message essentially encapsulates the feature representation, $h_v$, of the transmitting node $v$ [72].

To formalize this, consider the message $m_{v\to u}^{(l)}$ from node $v$ to its neighboring node $u$ at a given layer. This message is defined as follows

$$m_{v\to u}^{(l)} = h_v^{(l-1)} \tag{2.23}$$

for all $u \in \mathcal{N}(v)$, where $\mathcal{N}(v)$ represents the neighbors of node $v$, where $m_{v\to u}^{(l)}$ is the message sent from node $v$ to node $u$ at layer $l$, and $h_v^{(l-1)}$ is the feature representation of $v$ from the previous layer [72]. Every node in the graph receives messages from its neighbors. This method ensures that information is transferred and aggregated fluidly throughout the network, allowing for refined feature representations for each node based on its local neighborhood context [72]. The aggregation and the transformation are detailed below.

**1. Aggregation:** Every node $v$ aggregates the messages it has received, usually by adding up or averaging them to produce an intermediate representation $a_v$. Therefore, the aggregation process is the final step in this message-passing phase since it requires us to combine all of the received messages into a single message representation for each node [72]. Mathematically, this aggregation is depicted as

$$a_v^{(l)} = AGGREGATE^{(l)}\left(\left\{m_{u\to v}^{(l)} : u \in N(v)\right\}\right), \tag{2.24}$$

where $a_v^{(l)}$ denotes the aggregated feature representation of node $v$ at layer $l$ [72]. The collection of messages $m_{v\to u}^{(l)}$ received from the neighboring nodes is processed by the function $AGGREGATE$, which is usually a summation or averaging operation.

**2. Transformation:** A simple function transforms the aggregated representation using a linear transformation followed by a non-linear activation function. Mathematically, the representation $a_v^{(l)}$ is transformed as

$$h_v^{(l)} = \sigma\left(W^{(l)} a_v^{(l)}\right), \tag{2.25}$$

where $W^{(l)}$ represents the linear transformation, which is typically a weight matrix pertinent to layer $l$ [72]. The weight matrix performs a linear transformation, which ensures the model's adaptability during the learning phase and allows for the capture of linear dependencies in the feature space. This gives us node $v$'s updated representation, $h_v$. The complete network can be expressed as

$$h_v^{(l)} = \sigma\left(W^{(l)} \cdot \text{AGGREGATE}^{(l)}\left(\left\{h_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right)\right), \tag{2.26}$$

where $\mathcal{N}(v)$ represents the neighbors of the node $v$, and $\text{AGGREGATE}^{(l)}$ is the aggregation function applied to the hidden states of neighboring nodes [72].

Nonetheless, GCN assumes that every neighbor is equally significant, even though, in practice, each node may not be equally significant in the context of a graph, so the attention function is used to adjust for each neighbor's importance.

### 2.4.5 Graph Attention Network (GAT)

The GAT algorithm concentrates on neighbor nodes' more significant and pertinent features for improved feature representation. The GAT algorithm applies a shared linear transformation to each node's feature vector, which is pa-

rameterized by the weight matrix $W$ [28]. The process is defined as follows

$$Z_v^{(l)} = W^{(l)} h_v^{(l-1)}, \forall v \in \mathcal{V},$$

where $Z_v^{(l)}$is the transformed feature vector for node $v$ at layer and $W^{(l)}$ is the weight matrix at layer l [28]. The attention coefficient shows how significant node $j$'s characteristics are to node $i$. This is done using the attention mechanism, which is a shared attention function [28] given by

$$e_{ij}^{(l)} = a\left(Z_i^{(l)}, Z_j^{(l)}\right). \tag{2.27}$$

The softmax function then normalizes the attention coefficients as [28]

$$\alpha_{ij}^{(l)} = \frac{exp(e_{ij}^{(l)})}{\sum_{k \in N(i)} exp(e_{ik}^{(l)})}. \tag{2.28}$$

Lastly, the output features of the node are computed by combining the neighboring features linearly, employing the normalized attention coefficients [28]. given by

$$h_i'^{(l)} = \sigma\left(\sum_{j \in N(i)} \frac{exp(a(Z_i^{(l)}, Z_j^{(l)}))}{\sum_{k \in N(i)} exp(a(Z_i^{(l)}, Z_k^{(l)}))} Z_j^{(l)}\right). \tag{2.29}$$

The update function is implemented on the node. This allows us to obtain a more accurate representation of each node's features because significant neighbors' nodes are now gathered and combined. This method enhances the model's performance and adds realism to the learning process [28].

In the next section, we discussed the improved version of the graph convolution technique, such as the GraphSage Convolutional Neural Network.

### 2.4.6 GraphSage

Graphsage's primary concept is that it creates embeddings from a node's immediate neighborhood and incorporates the ability to create one's own embeddings while updating a node's new features. In order to obtain a comprehensive representation of the graph data sets, this model incorporates the node's feature into the aggregation process [73].

Given a node $v$ and its neighbors $\mathcal{N}(v)$, the first step in GraphSAGE is to aggregate the feature vectors of the node's neighbors, given by

$$\text{AGG}\left(\{h^u_{(l-1)}, \forall u \in \mathcal{N}(v)\}\right) = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} h^u_{(l-1)}, \tag{2.30}$$

where $h^u_{(l-1)}$is the feature vector of node $u$ in the $(l-1)$-th layer and $|\mathcal{N}(v)|$ is the number of neighbors. The aggregated feature vector is then concatenated with the feature vector of the node $v$ itself, and then it is transformed by applying a learned linear transformation and a non-linear activation function as follows

$$h^v_l = \sigma\left(W^{self}_l \cdot h^v_{(l-1)} + W^{neigh}_l \cdot \text{AGG}\left(\{h^u_{(l-1)}, \forall u \in N(v)\}\right)\right). \tag{2.31}$$

And $l_2$ normalization of the node embeddings is applied after each iteration as follows

$$h_l^v = \frac{h_l^v}{\|h_l^v\|}. \tag{2.32}$$

The GraphSAGE model incorporates its own and neighbors' features; nodes get represented in a way that respects individuality and the community's influence on the graph [73]. It offers flexibility by relying on the node's own data and considering its surroundings, giving a holistic view.

## 2.5 Conclusions and Future Directions

In this chapter, state-of-the-art graph learning using GNNs is presented, focusing on their capability to capture relations in the image data sets for computer vision applications. The description covered algorithms for graph learning, challenges associated with generating graphs from images, and various approaches to graph generation. On the basis of our exhaustive survey, we have determined the following future research directions for graph learning in computer vision:

- Eliminating the need for manually crafted features and domain expertise by automating the process of constructing meaningful optimal graph representations from raw data.

- Construct sparse graph methodologies enhancing the effectiveness of GNNs by making use of sparsity in graph representations, especially for large and dense graphs that are frequently encountered in computer vision tasks.

- Enhancing performance in tasks such as action recognition and scene comprehension by integrating graph-based representations with additional data modalities (*e.g.*, text, audio).
- Developing frameworks and instruments to interpret the decisions of GNNs in a manner that increases their clarity and accessibility to users.

# Chapter 3. Multilabel Defect Classification of Large Concrete Structures Using Vision Graph Neural Network with Edge Convolution

## 3.1 Introduction

The safety of large structures, such as concrete bridges, buildings, and pavements, requires precise detection and identification of various defects to estimate the degradation level of various components. Defect identification poses a challenge due to their smaller size compared to the bridge and the possibility of multiple defects overlapping with each other [1]. In real-world scenarios, the complexity of identifying these defects is heightened by various environmental factors. These include changing lighting conditions, surface wetness due to weather, and different non-critical surfaces like minor holes, markings, stains, or graffiti. Traditionally, defect detection and identification relied on domain-specific manual inspections by experts [74], which are subject to human error. This challenge requires a robust methodology capable of learning the subtle differences and relations between defects for better classification accuracy.

Deep learning algorithms [2] have been successfully employed across various fields for defect classification, demonstrating their ability to detect subtle

feature variations. When it comes to object detection and material recognition, convolutional neural networks (CNN) [3] perform admirably due to their ability to capture spatial hierarchies and local patterns within images [2]. While CNNs excel at single-label defect classification, the model performs poorly when faced with the complexities of multilabel images. The inherent challenge is that CNNs cannot capture the nuanced relationships and overlapping features associated with multiple defect classes within an image. Additionally, CNN models may not be optimally suited for detecting material defects, as they might fail to recognize the interplay between a defect and its surrounding multiscale structures [1]. In real-world scenarios, the characteristics of a material defect often depend on its adjacent components. Therefore, deep learning schemes that take the relations among the features are necessary.

Graph neural networks (GNNs) have applications in a wide range of domains where the available data is naturally in a graphical structure, such as social media networks [5], citation networks [6], and others. However, using GNNs for image classification requires reformulating image data within a graph structure. The vision GNN [7] uses a novel approach to represent images as a graph by dividing each image into multiple patches, allowing for the extraction of intricate relationships between image components.

Motivated by the performance of vision GNN in single-label defect classification, we proposed a multilabel vision GNN with edge convolution for concrete bridge defect classification, which uses the relationship between the defects in an image dataset to improve the accuracy significantly. We extended the vision GNN

by introducing edge convolution to capture the defects' intricate intra-dependent structure. In the proposed multilabel vision GNN [7], a single image is converted into patches, which serve as the nodes of the graph. The edges or connections among these nodes (patches) are then established based on the nodes' proximity to each other, resulting in a $k$-neighborhood graph for the image. The model performs edge convolution within the newly created graph for the aggregation of features at each node with its neighbor.

The message aggregation proposed is a unique variation of edge convolution that applies CNN algorithms directly to the features of the graph nodes. By extending the standard convolutional operations to the graph structure, this variation makes it possible to extract features and propagate information among the connected nodes of the graph efficiently. We utilized the combined capacities of CNNs and GNNs, as well as feedforward networks (FFN), and enabled them to learn the features and relationships between the features of different patches. This layout makes it possible for effective updates and the aggregation of information.

A dual linear layer FFN module is applied to transform node features before and after the graph convolution to address the problem of over-smoothing in deep GNNs. We utilized the multilabel loss function with Logit to enable multilabel classification tasks. This allowed us to extend the capabilities of the vision GNN from single-label classification to multiple labels. We also validated the multilabel vision GNN using the concrete detect dataset, referred to as the CODEBRIM dataset [1], which demonstrated significant improvement when compared to the state-of-art multilable defect classification [1].

## 3.2 Related work

In this section, we present a brief review of the de facto backbone models for image classification, including CNN, transformers, and GNN. A brief review of defect classification is also discussed.

### 3.2.1 CNN, Transformer, and MLP-based Models for Image Classification

Researchers have been using CNNs to capture the image features more effectively for classification [75], image segmentation, and object detection tasks. CNN-based models, such as LeNet-5 [76], AlexNet [77], VGGNet [78], Resnet [75], and MobileNet [79] are proposed in the literature to increase the model feature extraction capability as well as the accuracy of various vision tasks. On the other hand, inspired by the achievements of the transformer model in natural language processing, the vision transformer [80] was introduced for image classification. In this approach, images are treated as a sequence of patches, and a self-attention mechanism is employed to capture both the overall and local relationships among these patches. Modified versions of the vision transformer model, such as the Swin transformer [81], which performs sliding window-based self-attention, also showed better performance in image classification. For image datasets with different scales, the multiscale vision transformer was introduced in [82] by employing a hierarchical pyramid-shaped architecture to process images of varying scales.

Researchers in the computer vision area have also shown interest in multilayer perceptrons (MLP) because of their universal approximation capability. MLP enables the model to learn non-linear relationships among the features [83]. MLP satisfies the universal approximation theorem, which states that a simple network with a single hidden layer and a sufficient number of neurons can approximate any continuous function in a compact set. This property makes it suitable for learning the continuous hierarchical features of images. With tailored modules for specific vision tasks, MLP can perform with reasonable accuracy in image classification [84]. Due to its inability to process spatial feature information, MLP is combined with a CNN-based model to construct a more resilient and adaptable model [85]. While effective in vision applications, the above models do not consider the intricate relationships in the features for classification, limiting their capabilities for classifying images with overlapping features.

### 3.2.2 Graph Neural Network for Vision Tasks

GNN models can comprehend the context of any image by capturing the interaction between objects as graphs [9]. The incorporation of multiple edge building and edge convolution techniques enables this context comprehension [86]. This is particularly important in real-world scenarios where various degrees of relationships between image components frequently exist. Researchers exploit the ability of GNNs to continuously propagate label information across all nodes to perform vision tasks, such as image classification [87], image segmentation, and object detection. Building a model that combines CNN and GNN modules

can aid in the comprehensive learning of features, with CNN assisting in learning spatial features and GNN helping in learning relational features. These hybrid models demonstrated great potential in vision tasks, which served as the basis for the design of the vision GNN [7].

### 3.2.3 Defect Classification

With the incorporation of deep learning in structural and machinery defect classification, the capabilities of these defect classification systems have been enhanced, resulting in a paradigm shift in terms of accuracy, adaptability, and efficiency. Recent studies have used CNN algorithms to create an automated detection model for water surface defect classification [3]. A max-pooling convolutional neural network for supervised defect classification is proposed in [88], which operates directly on the raw pixel intensities of detected and segmented images. Surface defects on cement concrete bridges model [89] used a classical convolution-based network (VGG-16) [2] to classify defects in bridges. CNN-based models are also used to perform classification for domain-specific datasets, such as Crackforest[90], CSSC [91], and SDNET2018 [92]. These datasets are domain-specific because they are designed to address the complexities and characteristics of the "crack" defect domain only. Li *et al.* proposed to build a local pattern predictor (LPP) using CNN for crack versus non-crack classification [74]. These models are primarily concentrated on assessing single-label classification and evaluating established CNN baseline models. For multilabel defect classification, the authors in [1] used a meta-learning-based task-specific neural network.

the proposed approach diverges from the emphasis on single-label crack detection. Instead, we use vision GNN with edge convolution to comprehensively model a multilabel defect classification for concrete bridges, ensuring that all complexities in the data are taken into account.

## 3.3 Background on Vision GNN and Problem Statement

This section provides a brief background on the vision GNN, which is the backbone of the proposed multilabel defect classification task.

### 3.3.1 Vision GNN

To enhance the understanding of images beyond what is offered by CNN's grid representations or the transformer model's sequence representations, vision GNN introduces a novel approach. This method constructs a graph from an image, treating the image as a unique form of a grid. This graph-based representation is particularly advantageous for images depicting human action. By isolating various body parts and establishing connections between them, the graph captures meaningful relationships, thus providing a more comprehensive understanding of the image's semantics. This approach allows for a more nuanced and accurate interpretation of the spatial and relational dynamics within the image.

The vision GNN algorithm proceeds with an image of size $H \times W \times C$ and converts the image into $N$ patches, where $H$ is the height, $W$ is the width, and $C = 3$ is the channel of an image. The feature vector $x_i \in \mathbb{R}^d$, where $\mathbb{R}^d$ is a $d$ dimensional real-valued vector space, of each patch is defined based

on the pixel intensity value. The patches are represented as unordered nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ of the image graph. Every node is given a feature that will be processed in the next stage of the algorithm. The edges $\mathcal{E}$ of the graph $\mathcal{G}$ between each node are determined by calculating the Euclidean distance between each node's $k$-neighbors. For each node, a relative positional embedding is integrated with the distance feature to improve the estimation of the $k$-neighbors.

Graph-level processing begins with the grapher module and feedforward network module once the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is established. A graph convolution layer capable of aggregating the graph's node features is used, as specified by

$$\mathcal{G}_0 = \text{Update}(\text{Aggregate}(\mathcal{G}, W_\text{a}), W_\text{u}), \tag{3.1}$$

where $\mathcal{G}_0$ is the processed graph, $W_\text{a}$ and $W_\text{u}$ are the aggregated and updated weights of graph convolution, respectively. The node's representation is calculated by aggregating the features of neighboring nodes and then merging the combined feature further through the update operation given by

$$x_i^0 = h(x_i, \text{gConv}(x_i, N(x_i), W_\text{a}), W_\text{u}), \tag{3.2}$$

where $gConv$ denotes the graph convolution in neighbor nodes $N(x_i)$ of node $x_i$. The aggregated feature $x_i^0$ is first split into $h$ heads, *i.e.*, $\text{head}_1, \text{head}_2, \ldots, \text{head}_h$, and then these heads are updated with different weights, respectively. All the heads can be updated in parallel and are concatenated as the final values. The

multi-head module update process is expressed as

$$x_i^0 = [\text{head}_1 W_1 || \text{head}_2 W_2 || \ldots \text{head}_h W_h], \tag{3.3}$$

where $||$ denotes the concatenation. The updated process uses the multi-head module to capture expressiveness and various aspects of node relationships.

Multi-head modules facilitate updates of nodes' features in $h$ subspaces, enabling diverse feature representations. The feature converges to a similar embedding because the model gathers information about the $k$-hop neighborhood using multiple layers of the grapher module. The phenomenon is known as oversmoothing, and it worsens as the grapher module layer increases, resulting in insufficiently varied features. To increase feature diversity, a two-layer FFN, given by

$$Y = \sigma\left(\left(XW_1\right)W_2\right), \tag{3.4}$$

is used to reduce the oversmoothing problem. The function $\sigma(\cdot)$ denotes the sigmoid activation function, and the matrices $W_1$ and $W_2$ denote the weights of the first layer and the hidden layer, respectively. The activation function introduces non-linearity to the model, which helps the layer learn intricate patterns from the data that the model might overlook. The diversification of the data helps reduce the over-smoothing problem, leading to more accurate and nuanced predictions.

### 3.3.2 Problem Statement

The vision GNN, described above, primarily caters to single-label classification. Our goal is to enhance the model's versatility for multilabel classification. Additionally, the model needs to lessen the effects of the oversmoothing issue. Moreover, the proposed model requires more refined feature representation to improve defect classification. This necessitates a deeper model to capture the defects' fine-grained features. By introducing additional MLP layers within the grapher module, the oversmoothing problem can be reduced. Edge convolution can be used to combine the strengths of CNNs and GNNs to increase the richness of feature representation, as discussed in detail in the next section.

## 3.4 Proposed Vision GNN with Edge Convolution

This section discusses the proposed approach for multilabel defect classification by extending the vision GNN and further discusses the relationship of node convolution and edge convolution of vision GNN .

The core blocks of the proposed multilabel vision GNN integrated with edge convolution for enhanced feature processing are depicted in Figure 3.2. To utilize the capabilities of graph convolution, it is necessary to transform the image dataset into graphs. The images are divided into $N$ patches, each representing a node in the graph, as shown in Figure 3.1. For image $I$, let $\mathcal{P}$ be the set of $N$ patches extracted from the image. Each patch $\mathcal{P}_i \in \mathcal{P}$ is considered a node in the

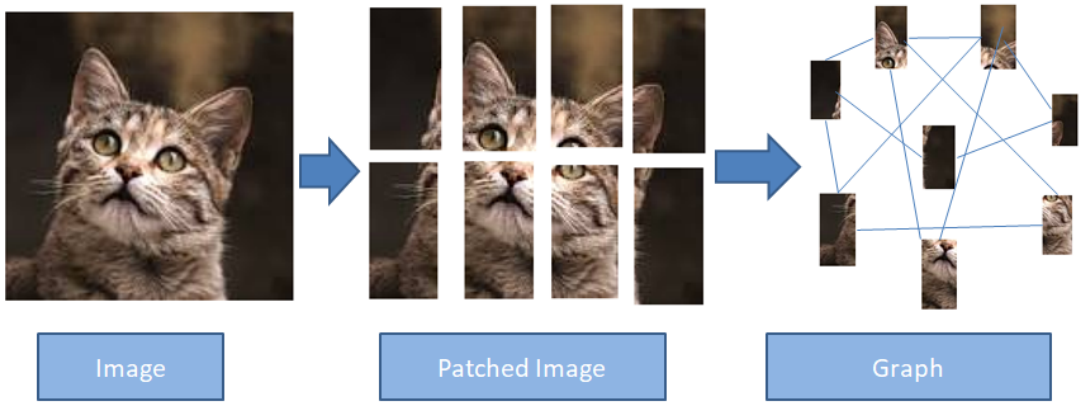



**Figure 3.1:** Graph formulation from an image.

graph $\mathcal{G}$, and can be expressed as

$$\mathcal{V} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}, \quad \mathcal{P}_i \in \mathbb{R}^{H \times W \times C}, \tag{3.5}$$

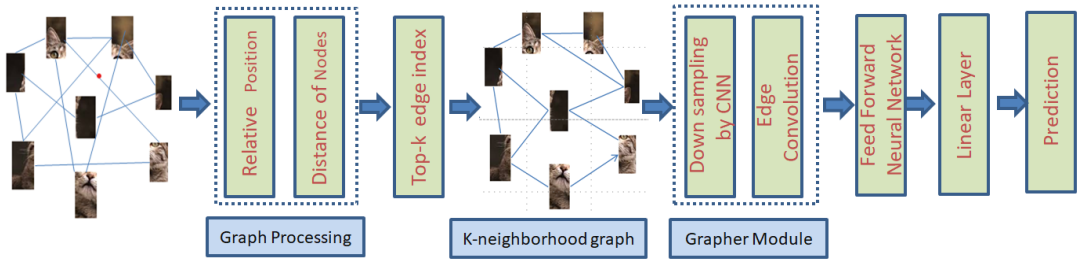



**Figure 3.2:** Sequential steps of the proposed defect detection algorithm.

The edges are established among the top $k$-nearest neighbors of each node $p_i$. The proximity of the nodes is determined by calculating the Euclidean distance between the nodes. To capture the long-range relationship between pixels in the image, the distance vectors are integrated with the relative positional information of the patches. The image's height and width are encoded using a sinusoidal

transformation to determine their positional embeddings. Let $\mathrm{P}_H$ and $\mathrm{P}_W$ denote the positional embeddings for height and width, respectively, then the positional embedding for height can be expressed as

$$\mathrm{P}_H(i) = \sin\left(\frac{i}{10000^{2k/d}}\right) \quad \text{for} \quad i = 1, 2, ..., d, \tag{3.6}$$

and the positional embedding for the width can be expressed as

$$\mathrm{P}_W(j) = \cos\left(\frac{j}{10000^{2k/d}}\right) \quad \text{for} \quad j = 1, 2, ..., d, \tag{3.7}$$

where $k$ represents the dimension of the positional embeddings and $d$ is the dimension of the image patches.

The two separate embeddings $\mathrm{P}_H$ and $\mathrm{P}_W$ are concatenated to form a comprehensive embedding $\mathrm{P}_{\text{concat}}$ of the patches, resulting in the collection of spatial information for each pixel in the image, and can be expressed as

$$\mathrm{P}_{\text{concat}} = \mathrm{P}_H \parallel \mathrm{P}_W. \tag{3.8}$$

Next, by multiplying the transformed positional embedding element-by-element with the original positional embedding, the relative positional embedding is determined, which can be expressed as

$$\mathrm{RP} = (\mathrm{P}_{\text{concat}})^{\text{Transform}} \odot \mathrm{P}_{\text{concat}}, \tag{3.9}$$

where the RP denotes the relative position embeddings. Relative positional embedding allows for the acquisition of spatial information from pixels, which is often overlooked by traditional convolution-based neural networks. This method is learnable and allows for a more comprehensive understanding of pixel relationships. In addition, the positional embeddings are also trained in the training phase, which enables the model to include long-range dependencies between image components.

To determine the edge index, the top $k$-neighbors are found using the integrated distance function and relative position embeddings, expressed as

$$\text{edge\_index(E)} = \text{index}(\textit{topk}(\text{dist} + \text{RP}), \tag{3.10}$$

where dist denotes the distance between the nodes in the graph, and the *topk* denotes the $k$-nearest neighbors of every node in the graph. The computed edge index is the basis for selecting the image features. This feature is passed to the grapher module's block for further graph processing. The grapher module downsamples the features by CNN and then performs edge convolution for future aggregation, followed by a feedforward network. The module is used in a repetitive manner to formulate a deep architecture. This deep architecture is intended to capture graph-based relationships via edge convolution and improve the resulting feature representation using the FFN.

Edge convolution is a specific instance of graph convolution that operates on edges. Node convolution gathers data from every neighboring node, irrespective of the number of edges linking them, while proposed edge convolution focuses

on specific connections between nodes that have been established by the edge indices $E$ in (3.10). This means that edge convolution is more sensitive to changes in how nodes are connected by edges, unlike the vision GNN approach [7]. Feature extraction in edge convolution processes edge features and node features altogether, producing an enhanced feature representation. Edge convolution is used in the Grapher module to gather and update features from the graph's neighbors along the edge defined by the edge indices E and can be expressed as

$$X = \text{EdgeConv}(x, \text{E}) = \max\left(\text{Conv}\left([\mathbf{x}_i \parallel \mathbf{x}_j - \mathbf{x}_i]\right)\right), \tag{3.11}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, are the self and neighbors' features. Edge convolution runs iteratively $k$ times to gather feature information from each node's $k$-hop neighbors, resulting in the convergence or smoothing out of node features. Since all of the features in the graph contain somewhat similar information, the occurrence of smoothed-out features hinders the formation of robust classifiers. An MLP is employed both before and after the grapher module to enhance the diversity of features. The FFN layer's addition to the block lowers the deep graph convolutional network's (GCN's) over-smoothing problem and can be expressed as

$$X_{\text{new}} = Y_{\text{MLP}} = \sigma((X \cdot W_{\text{MLP},1} + b), W_2), \tag{3.12}$$

where $\sigma(\cdot)$ denotes the sigmoid activation function, $W_{\text{MLP},1}$ is the weight of MLP layer, $b$ is the bias, and $W_2$ is the update weight. After going through the specified

block iteratively, predictions are made using a prediction layer given by

$$Y_{\text{pred}} = \sigma(\text{Linear}(X_{\text{new}})), \tag{3.13}$$

where $\sigma(\cdot)$ denotes the sigmoid activation function used in the prediction layer. The predicted output, $Y_{\text{pred}}$, has a size of $(\textit{batch size} \times N)$, where $N$ is the number of classes. The sigmoid activation function is employed to rescale the prediction values within the range of 0 to 1, *i.e.*, $\hat{y}_i = \begin{cases} 1 & \text{if } Y_{\text{pred}(i)} \geq 0.5, \\ 0 & \text{otherwise.} \end{cases}$

If the prediction score of every $N$ class, $Y_{\text{pred(i)}}$ where $i \in \{0, 1, \ldots, N\}$, is higher than the threshold score of 0.5, we assign a value of 1; otherwise, we assign a value of 0. In that way, every image is predicted to have the presence of multiple classes at the same time. Similarly, the ground truth labels also have a size of $(\textit{batch size} \times N)$. If three classes are present, the ground truth class for one sample in the batch could resemble $[1, 0, 1, 0, 1]$. In order to encode the ground truth labels, we employ one-hot encoding.

The loss, or the difference between the ground truth labels, $Y$, and the predicted output, ypred, is then computed using the multilabel cross-entropy loss function. The multilabel cross-entropy loss function is defined as follows:

$$\text{Loss} = -\frac{1}{\text{batch size}} \sum_{i=1}^{\text{batch size}} \sum_{j=1}^{N} \left[ Y_{ij} \log(Y_{\text{pred}(ij)}) + (1 - Y_{ij}) \log(1 - Y_{\text{pred}(ij)}) \right]. \tag{3.14}$$

The algorithm discussed above is summarized in Algorithm 1. The algorithm iterates through blocks, performing subblock operations within each block. Downsampling and FFN are used for blocks other than the first block. The grapher module employs edge convolution iteratively, and FFN is used again. Finally, a linear layer is used for prediction. During execution, the use of 's' represents specific sub-blocks within each block. The validation results of experimenting with a use case scenario for the classification of concrete bridge defects are presented next.

In Figure 3.2 sequential steps of the proposed algorithm is summarized in which employs an Edge Convolution-enabled Vision Graph Neural Network (GNN). First, the images are divided into patches, each representing a node in the graph. Then, the graph is constructed, and relative positional embeddings are computed. The integrated positional embeddings and distance of nodes are used to find the top-$k$ neighbors of the graph. Using the nearest $k$ neighbors, a $k$-neighborhood graph is generated, which is then passed through the grapher module for further graph processing. The grapher module uses convolution for downsampling and EdgeConv for feature aggregation and updating iteratively. An MLP in the feedforward layer is used to increase feature variety, and predictions are made using a prediction layer with sigmoid activation.

**Algorithm 1** Vision GNN with edge convolution Training Algorithm

**Input:**
  Image Patches: $X$ represent the nodes in the graph
  Establish edges between nodes based on top-$k$ neighbors using pairwise Euclidean distances.
  **Relative Position Embedding:** $P_{\text{concat}}$
  Edge Indices Construction: $E = topk(\text{dist}(X) + P_{\text{concat}})$
**Initialization:**
  Initialize model parameters: Feature: $X^{(0)} = X$, blocks, channels, etc.
**Backbone:**
**for** $i$ in blocks **do**
  **if** $i \neq$ first block **then**
    Downsample: $X^{(b)} = \text{Convolution}(X^{(b-1)})$
      $X^{(b,s)} = \text{FFN}(X^{(b,s)})$ here, 's' represents a specific sub-block within the 'b' block
  **end if**
  **for** $j$ in sub-blocks **do**
    Grapher module:
      $X^{(b,s)} = \text{EdgeConv}(X^{(b,s)}, E)$
    FFN layer:
      $X^{(b,s)} = \text{FFN}(X^{(b,s)})$
  **end for**
**end for**
**Prediction:**
Prediction: $Y = \text{Linear}(X^{(b,s)})$

### 3.5 Dataset and Experimentation

For Multilabel classification, we used the CODEBRIM dataset. Moreover, to generalize our model performance for the image classification task, we employed our model using the ImageNet-10 dataset.

### 3.5.1 Dataset Description

**1. CODEBRIM Dataset:** The CODEBRIM dataset [1] is a well-balanced, diverse, and overlapping defect dataset with the potential to test the validity of the described defect classification model. The dataset consists of five distinct bridge defect types: cracks, spallations, exposed reinforcement bars, efflorescence (calcium leaching), and corrosion (stains). Different levels of deterioration in the bridges were selected to create the datasets. To ensure the datasets are robust and diverse, the pictures were taken in various weather conditions and at different times of the day. The dataset consists of 1) a total of $2,506$ generated non-overlapping background bounding boxes; 2) $5,354$ annotated defect bounding boxes (mostly with overlapping defects); and 3) $1,590$ high-resolution images with defects in the context of 30 distinct bridges. The total number of images in each defect class is: 1) exposed bars: $1,507$; 2) corrosion stain: $1,559$; 3) efflorescence: 833; 4) spallation: $1,898$; and 5) crack: $2,507$. The distribution of the defects in the CODEBRIM datasets is shown in the bar chart in Fig. 3.4a.

The majority of the images in the datasets have multiple labels. The raw image of the datasets is annotated with bounding boxes, and every individual bounding box contains several overlapping defects. The dataset poses significant challenges due to variations in the aspect ratio, resolution, and scale of the bounding boxes and defects. For example, a crack might appear elongated in one image but not in another, depending on the image's aspect ratio and resolution.

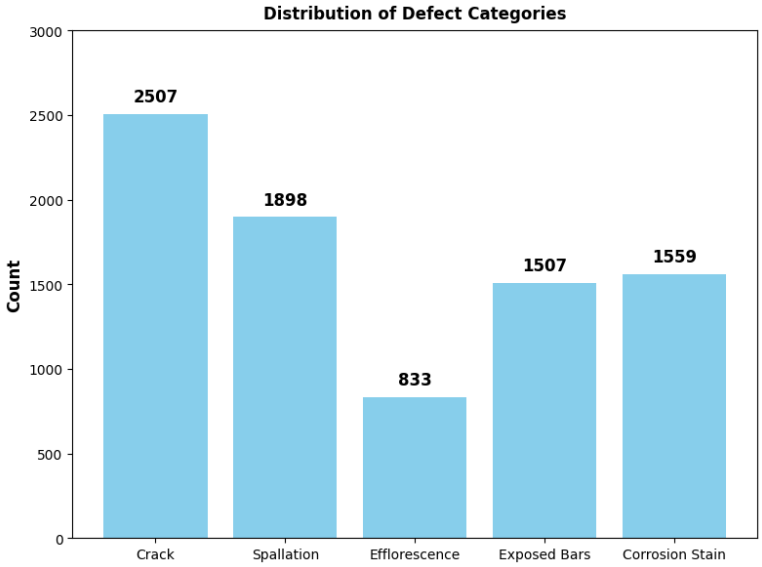
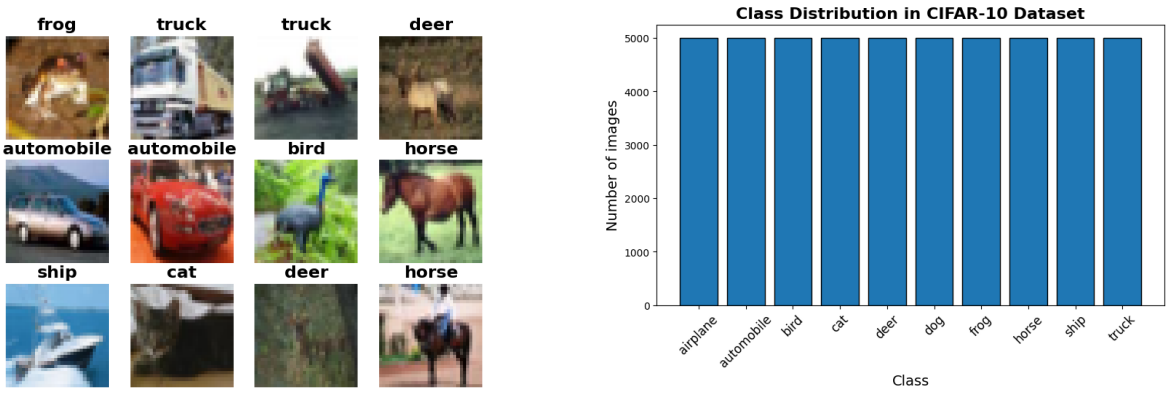



**Figure 3.3:** Distirbution defects in the CODEBRIM datasets.

Similarly, a spalling defect might be more apparent in some images than others, depending on the scale at which the image is captured.

**2. CIFAR-10 Dataset:** The ImageNet-10 dataset, which comprises 10 unique classes, is a condensed version of the larger ImageNet dataset. These classes—which may include categories like "dog," "cat," "car," "aeroplane," "flower," "fruit," "chair," "tree," "house," and "fish"—are thoughtfully chosen to offer a varied representation of objects. In ImageNet-10, the number of images in each class is the same, guaranteeing an equal batch size for every category. The ImageNet-10 dataset can be used to generalize the model performance for real-world classification tasks. The image set includes a variety of images that are distinct from one another in terms of color, texture, and shape for each label.

Due to this particular characteristic, the dataset is also considered to be a significant benchmark dataset for the task of image classification.




**(a)** CIFAR-10 images **(b)** CIFAR-10 dataset distribution

**Figure 3.4:** CIFAR-10 Image Dataset and Distribution of Dataset.

### 3.5.2 Data Processing and Training

The model resorts to the traditional CNN-based approach to handle the variation in scale and resolution. The datasets are balanced by ensuring the training set contains equal defect classes. Every image has multiple labels assigned to it simultaneously. The processing of the defect datasets includes the background as a label for the classifier to learn. Different types of defects and their backgrounds are shown in Figures 3.5 to 3.7. It is clear from the Figures 3.5 to 3.7 that inclusion of the backgrounds in the label representation has the advantage of teaching the model the context of the defects about the background. Better localization of the defects promotes learning their spatial relationship to the surroundings or other defects that overlap within the same background. As the

model is trained with various backgrounds and defects simultaneously, it improves its generalization capability about real-world backgrounds.

(a) (b) (c)

**Figure 3.5:** (a) Crack and background, (b) crack and background, and (c) crack and background.

(a) (b) (c)

**Figure 3.6:** (a) Background only, (b) crack and background, and (c) corrosion and background.

### 3.5.3 Experiment Settings

We carefully select the parameters required for our model's stabilization and performance optimization. We opted for a training cycle of 300 epochs; we selected this duration based on the model's peak performance observed at the $300^{\text{th}}$ epoch and its tendency to stabilize thereafter. We adopted a batch size of

(a) (b) (c)

**Figure 3.7:** (a) Background only, (b) crack, efflorescence, corrosion stain, and (c) crack only.

64 considering computational efficiency, model stability, and memory usage optimization. Every image is divided into 48 patches to formulate the iniital graph for the model input pipeline. A binary cross-entropy loss with logits is used to regularize the training procedure. The sigmoid function is used to scale the prediction layer's output, resulting in a detailed depiction of the model's confidence across several classes. A threshold score of 0.5 is used to determine whether or not a prediction is deemed accurate for a given task. We used Adam Optimizer for the optimization, which directly uses weight decay in the optimization process, resulting in better weight regularization. The cosine learning rate scheduler was employed to facilitate the initial convergence, gradually lowering the learning rate as training progressed. We choose a weight decay of $1 \times 10^{-3}$, which penalizes the large weights contributing to reducing overfitting during training.

We choose stochastic path probabilities of $(0.1, 0.1, 0.1, 0.3)$ to introduce randomness during training. The stochastic path works as an inner control variable, introducing control variability into the training process. The stochastic path controls the distinct module or set of operations that our training process can fol-

low. Rather than using total randomness to select a path during the training process, stochastic path probabilities indicate the likelihood of following a path. This option allows us to make the model prefer a certain path that is more efficient and convergent. To create a robust model for different types of data distribution, stochastic path probabilities incorporate dynamic adaptability into the model by highlighting or underlining specific paths to follow in the model. The model is less likely to overfit specific patterns common in the datasets when trained for adaptive input. A dropout probability of 0.5 is also chosen for the model to reduce overfitting during training. The training configuration of the model is described in Table 3.1.

### 3.6 Results: CODEBRIM Dataset

The vision GNN model with edge convolution is a pyramid architecture model. This means that it has a hierarchical structure, where the number of channels in the image increases while the spatial resolution of the data decreases as we move up the hierarchy. This is accomplished by incrementing the number of filters as we ascend in the hierarchy. The concept behind designing a pyramid model is that the lower layers in the hierarchy capture the local and fine-grained features, while the higher layers capture the long-range dependencies of the features. This model is useful for learning both fine and coarse-grained features at various levels of granularity within the input data for the CODEBRIM dataset [1] with different scales of defects. The model suppresses information at one level while expanding it at another, resulting in a robust model for capturing the fea-

**Table 3.1:** Training configuration.

| **Parameter** | **Value** |
|---|---|
| Epochs | 300 |
| Optimizer | AdamW |
| Batch size | 64 |
| Number of Patch | 48 |
| Start learning rate (LR) | $1 \times 10^{-4}$ |
| Learning rate schedule | Cosine |
| Weight decay | $1 \times 10^{-3}$ |
| Stochastic path | 0.1, 0.1, 0.1, 0.3 |
| Mixup prob | 0.8 |
| Cutmix prob | 1.0 |
| dropout prob | 0.5 |

tures' local and global dependencies. We can see in Figure 3.9 that the model is rapidly improving its accuracy within 40 epochs with the decrease in loss shown in Figure 3.8. The model consistently improves its ability to recognize and utilize the feature, reaching a high accuracy of 96 percent after 500 epochs of training. The increased stability of learning observed after 40 epochs indicates that the deep layer is specifically designed to capture long-range contextual features. This training trend indicates the model's ability to analyze the complex characteristics

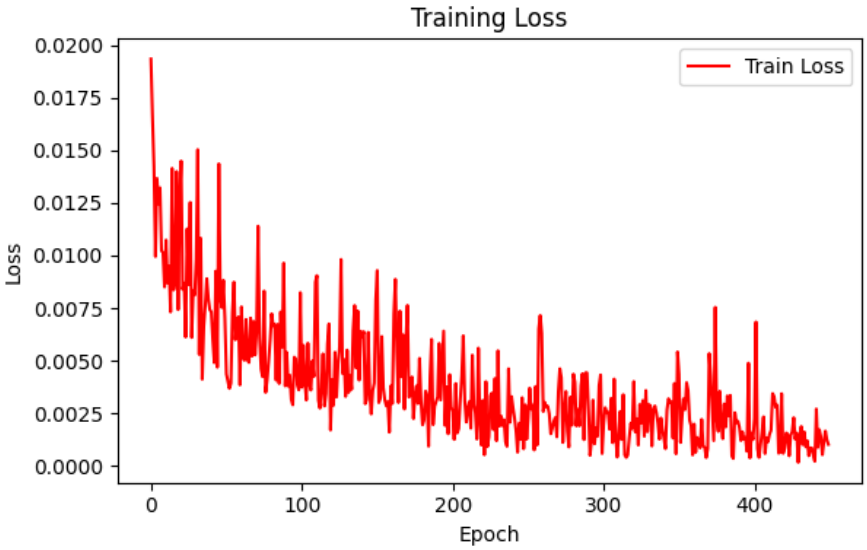



**Figure 3.8:** Training loss.

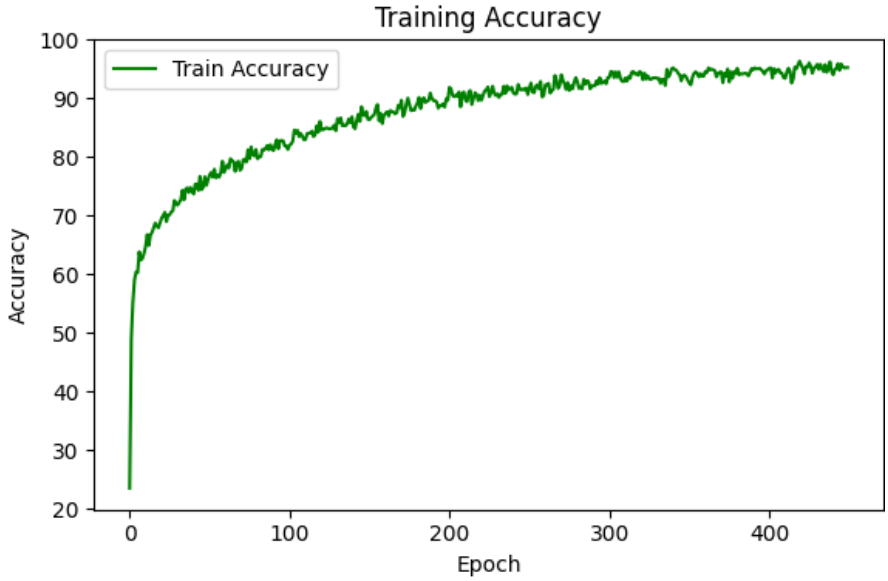



**Figure 3.9:** Training accuracy.

of defects and establish the relationship between overlapping defects in the image

sets. The models achieved 86 percent accuracy on the test set and 83 percent accuracy on the validation set, as shown in Table 3.2

**Table 3.2:** Model performance comparison with popular CNNs.

| **Architecture** | **Best Validation** | **BV-test** | **Params (million)** | **Layers** |
|---|---|---|---|---|
| Alexnet | 63.05 | 66.98 | 57.02 | 8 |
| T-CNN | 64.30 | 67.93 | 58.60 | 8 |
| VGG-A | 64.93 | 70.45 | 128.79 | 11 |
| VGG-D | 64.00 | 70.61 | 134.28 | 16 |
| WRN-28-4 | 52.51 | 57.19 | 5.84 | 28 |
| Densenet-121 | 65.56 | 70.77 | 11.50 | 121 |
| ENAS-1 | 65.47 | 70.78 | 3.41 | 8 |
| ENAS-2 | 64.53 | 68.91 | 2.71 | 8 |
| ENAS-3 | 64.38 | 68.75 | 1.70 | 8 |
| MetaQNN-1 | 66.02 | 68.56 | 4.53 | 6 |
| MetaQNN-2 | 65.20 | 67.45 | 1.22 | 8 |
| MetaQNN-3 | 64.93 | 72.19 | 2.88 | 7 |
| vision GNN | **83.00** | **86.00** | **24.004** | **77** |

We also compared the accuracy of the proposed model with other architectures used for defect classification. The vision GNN outperforms the competition

with an impressive 86% accuracy, as shown in Table 3.2. Notably, it outperforms traditional CNN-based models, such as Alexnet, T-CNN, VGG-A, and VGG-D in terms of validation and test accuracy. The vision GNN also outperforms more recent meta-learning-based few-shot learning models, such as WRN-28-4, Densenet-121, ENAS-1, ENAS-2, ENAS-3, MetaQNN-1, MetaQNN-2, and MetaQNN-3. It is important to note that the CNN-based models, such as Alexnet, T-CNN, VGG-A, and VGG-D, generally exhibit more parameters than vision GNN. The ability of the vision GNN to outperform these CNN-based models with fewer parameters highlights the effectiveness of its unique graph-based architecture in capturing complex relationships within data, which contributes to superior performance in image classification tasks. Moreover, the meta-learning-based model uses fewer parameters due to advanced techniques like few-shot learning.

The model's ability to discern intricate patterns, shapes, and spatial relationships within the images is demonstrated by its high accuracy in this scenario. Furthermore, it highlights the model's efficacy in extrapolating from the training data to unfamiliar samples. Moreover, the model's robustness is consistently validated by the 84% accuracy achieved on the validation set. A slight drop in accuracy compared to the test set is expected, as the validation set allows us to assess the model's generalization of data that was not directly trained on. In summary, the accuracy achieved demonstrates the models' effectiveness in dealing with the complexities of multi-class image prediction, providing a solid foundation for their practical application in fields that require accurate defect detection and categorization.

### 3.7 Results: CIFAR-10 Dataset

We validated the model for the image classification task by performing image classification on CIFAR-10 datasets. We use the same experimental settings 3.1. Since the Imgaenet is a single-label datasets, every image contains only one object. We have used the binary cross-etropy loss function. The testing accuracy for the model we got is 86% . The model achieves high accuracy for single-label image classification tasks.

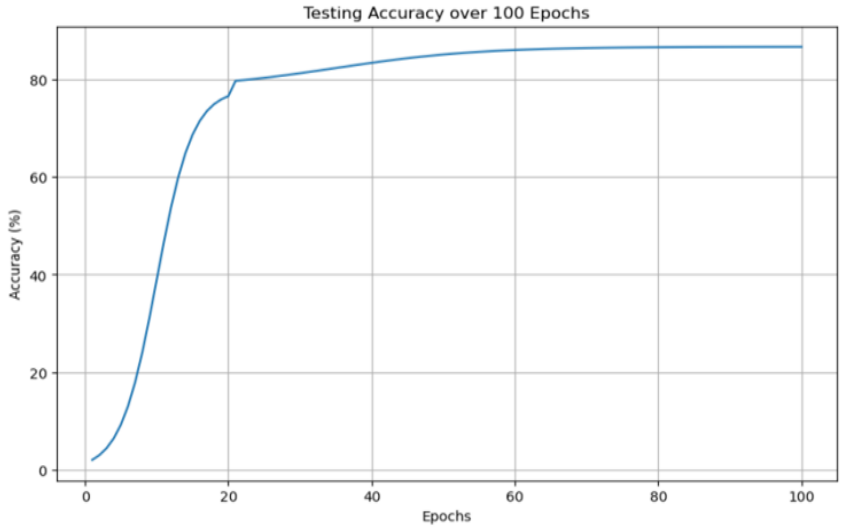



**Figure 3.10:** Testing Accuracy.

The confusion matrix for the classication task is also provided below in Figure 3.11. When it comes to the task of image classification, we can see that our model performs fairly well.

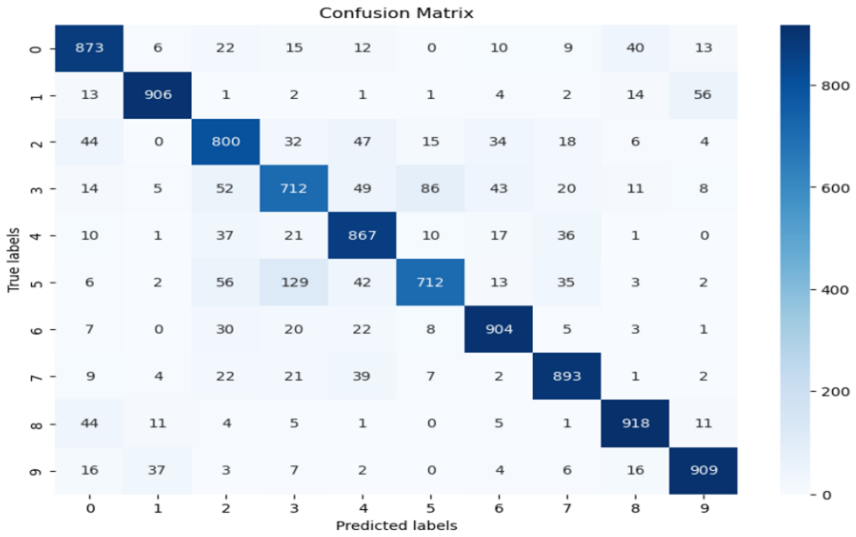



**Figure 3.11:** Confusion matrix of CIFAR-10 testing datasets.

## 3.8 Conclusion

This thesis presented the defect identification of CODEBRIM datasets, focusing on common bridge defects. Traditional convolution-based methods cannot train a comprehensive classifier due to the overlapping and multiscale defects in bridges. The vision GNN with edge convolution, which is a hybrid network enriched with GNN and CNN capabilities, aids in reducing the shortcomings of conventional convolution-based networks. The proposed vision GNN model is advantageous in capturing image components' intricate and irregular relationships. The proposed model incorporates CNNs, GNNs, and FFN layers, allowing for effective information updates and aggregation while addressing the over-smoothing problem commonly observed in deep GNNs. Despite the difficulties of predicting multilabel image by matching the target labels exactly with state-of-the-art

networks, the proposed models achieved 86% accuracy on the test set and 83% accuracy on the validation set. In summary, the proposed approach presents a novel deep-learning algorithm designed for multi-class defect classification, thereby providing a robust method for dealing with the complexities introduced by varying material appearances, changing lighting conditions, and overlapping defects. The improved accuracy highlights the potential of hybrid networks for real-world applications, such as bridge safety assessment and defect classification. In our future work, we plan to explore meta-vision GNN to reduce the number of parameters in the model.

# Chapter 4. Conclusions and Future Work

This thesis demonstrated the prominence of graph neural networks (GNNs) for capturing complex relationships in image datasets with their applications in computer vision. We presented an in-depth exploration of graph learning, focusing on non-neural and neural methods. We also discussed the techniques used to construct graphs from image data. We detailed the state-of-the-art graph learning processes, such as graph kernel-based embeddings, recurrent neural networks, and graph convolutional networks. As a result of the fact that GNN can effectively capture both the long- and short-range dependencies in datasets, we were able to demonstrate that graph convolutional neural networks perform better than conventional neural networks in the field of computer vision. This thesis also supports the assertion by developing and implementing a hybrid graph neural network architecture for multilable defect classification from images. The proposed model incorporates CNNs, GNNs, and FFN layers, allowing for effective information updates and aggregation while addressing the over-smoothing problem commonly observed in deep GNNs. To validate the model's performance, we implemented the developed model on the multilabel defects named CODEBRIM and demonstrated that our methodology increased classification accuracy by 16%. The improved accuracy highlights the potential of hybrid networks for

real-world applications, such as safety assessment and defect classification of large-scale structures, such as concrete bridges, tanks, and large-scale mechanical equipment. Based on our proposed research and available literature in computer vision concerning graph learning, the following are some of the tasks induced in our future work.

- To enhance the performance of the computer vision model, we plan to incorporate supplementary data, including audio, text, and 3D data, to enhance the performance of our model.
- We also plan to automate the graph construction process to generate an optimal graph, thereby reducing the run time complexity.
- We also plan to incorporate meta-learning into the Vision GNN model to reduce the number of parameters and the degree to which it depends on the amount of training data.
- Finally, we plan to develop frameworks and instruments to interpret GNNs in a manner that provides insight into their decision-making capability.

# References


[1] Martin Mundt, Sagnik Majumder, Sreenivas Murali, Panagiotis Panetsos, and Visvanathan Ramesh. Meta-learning convolutional neural architectures for multi-target concrete defect classification with the concrete defect bridge image dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11196–11205, 2019.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[3] Sejune Cheon, Hankang Lee, Chang Ouk Kim, and Seok Hyung Lee. Convolutional neural network for wafer surface defect classification and the detection of unknown defect class. *IEEE Transactions on Semiconductor Manufacturing*, 32(2):163–170, 2019.

[4] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[5] Yi Han, Shanika Karunasekera, and Christopher Leckie. Graph neural networks with continual learning for fake news detection from social media. *arXiv preprint arXiv:2007.03316*, 2020.

[6] Jiaying Liu, Feng Xia, Xu Feng, Jing Ren, and Huan Liu. Deep graph learning for anomalous citation detection. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2543–2557, 2022.

[7] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision GNN: An image is worth graph of nodes. *Advances in Neural Information Processing Systems*, 35:8291–8303, 2022.

[8] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[9] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[10] Chaoqi Chen, Yushuang Wu, Qiyuan Dai, Hong-Yu Zhou, Mutian Xu, Sibei Yang, Xiaoguang Han, and Yizhou Yu. A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective, 2022.

[11] Licheng Jiao, Jie Chen, Fang Liu, Shuyuan Yang, Chao You, Xu Liu, Lingling Li, and Biao Hou. Graph representation learning meets computer vision: A survey. *IEEE Transactions on Artificial Intelligence*, 4(1):2–22, 2022.

[12] Nurul A. Asif, Yeahia Sarker, Ripon K. Chakrabortty, Michael J. Ryan, Md. Hafiz Ahamed, Dip K. Saha, Faisal R. Badal, Sajal K. Das, Md. Firoz Ali, Sumaya I. Moyeen, Md. Robiul Islam, and Zinat Tasneem. Graph neural network: A comprehensive review on non-euclidean space. *IEEE Access*, 9:60588–60606, 2021.

[13] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[14] Xiao-Meng Zhang, Li Liang, Lin Liu, and Ming-Jing Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in genetics*, 12:690049, 2021.

[15] Henry Senior, Gregory Slabaugh, Shanxin Yuan, and Luca Rossi. Graph neural networks in vision-language image understanding: a survey. *The Visual Computer*, March 2024.

[16] Pingping Cao, Zeqi Zhu, Ziyuan Wang, Yanping Zhu, and Qiang Niu. Applications of graph convolutional networks in computer vision. *Neural computing and applications*, 34(16):13387–13405, 2022.

[17] Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Mingtao Feng, Xia Zhao, Qiguang Miao, Syed Afaq Ali Shah, et al. Scene graph generation: A comprehensive survey. *arXiv preprint arXiv:2201.00443*, 2022.

[18] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. *CoRR*, abs/2010.02863, 2020.

[19] Abdul Majeed and Ibtisam Rauf. Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions*, 5(1), 2020.

[20] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[21] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *CoRR*, abs/2002.00388, 2020.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2008.

[23] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[24] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021.

[25] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[26] I. K. Fodor. *A Survey of Dimension Reduction Techniques*. Lawrence Livermore National Laboratory, 2002. UCRL-ID-148494.

[27] Chester Holtz, Onur Atan, Ryan Carey, and Tushit Jain. Multi-task learning on graphs with node and graph level labels. In *NeurIPS Workshop on Graph Representation Learning*, 2019.

[28] Jun Chen and Haopeng Chen. Edge-featured graph attention network. *arXiv preprint arXiv:2101.07671*, 2021.

[29] Alejandro Newell and Jia Deng. Pixels to graphs by associative embedding, 2018.

[30] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.

[31] Qichao Liu, Liang Xiao, Jingxiang Yang, and Zhihui Wei. Cnn-enhanced graph convolutional network with pixel- and superpixel-level feature fusion for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 59(10):8657–8671, 2021.

[32] Peer Neubert and Peter Protzel. Compact watershed and preemptive slic: On improving trade-offs of superpixel segmentation algorithms. In *2014 22nd International Conference on Pattern Recognition*, pages 996–1001, 2014.

[33] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[34] Pedro Felzenszwalb and Daniel Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 09 2004.

[35] Alastair P. Moore, Simon J. D. Prince, Jonathan Warrell, Umar Mohammed, and Graham Jones. Superpixel lattices. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[36] Yuhang Zhang, Richard I. Hartley, John S. Mashford, and Stewart Burn. Superpixels via pseudo-boolean optimization. *2011 International Conference on Computer Vision*, pages 1387–1394, 2011.

[37] Olga Veksler, Yuri Boykov, and Paria Mehrani. Superpixels and supervoxels in an energy optimization framework. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 211–224, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[38] Ming-Yu Liu, Oncel Tuzel, Srikumar Ramalingam, and Rama Chellappa. Entropy rate superpixel segmentation. In *CVPR 2011*, pages 2097–2104, 2011.

[39] Xiaofang Wang. Efficient graph-based image segmentation. In –, pages –, 1999.

[40] Radhakrishna Achanta and Sabine Süsstrunk. Superpixels and polygons using simple non-iterative clustering. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4895–4904, 2017.

[41] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 705–718, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[42] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[43] Yuli Sun, Lin Lei, Xiang Tan, Dongdong Guan, Junzheng Wu, and Gangyao Kuang. Structured graph-based image regression for unsupervised multimodal change detection. *ISPRS Journal of Photogrammetry and Remote Sensing*, 185:16–31, 2022.

[44] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[45] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[46] Andrea Vedaldi and Brian Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1469–1472, New York, NY, USA, 2010. Association for Computing Machinery.

[47] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[48] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175, 2001.

[49] Junchang Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-nn graph construction for visual descriptors. *2012*

*IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, 2012.

[50] O. Virmajoki and P. Franti. Divide-and-conquer algorithm for creating neighborhood graph for clustering. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 264–267 Vol.1, 2004.

[51] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. Fast knn graph construction with locality sensitive hashing. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 660–674, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[52] Ke Li and Jitendra Malik. Fast k-nearest neighbour search via dynamic continuous indexing, 2017.

[53] Jie Chen, Haw-ren Fang, and Yousef Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10(9), 2009.

[54] Firoozeh Beiranvand, Vahid Mehrdad, and Mohammad Bagher Dowlatshahi. Unsupervised feature selection for image classification: A bipartite matching-based principal component analysis approach. *Knowledge-Based Systems*, 250:109085, 2022.

[55] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Neural Information Processing Systems*, pages –, 2001.

[56] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, page 321–328, Cambridge, MA, USA, 2003. MIT Press.

[57] Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008.

[58] Hong Cheng, Zicheng Liu, and Jie Yang. Sparsity induced similarity measure for label propagation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 317–324, 2009.

[59] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 9(2):415–436, 2022.

[60] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 30(9):1616–1637, 2018.

[61] Mengjia Xu. Understanding graph embedding methods and their applications. *SIAM Review*, 63(4):825–853, 2021.

[62] Xin Liu, Tsuyoshi Murata, Kyoung-Sook Kim, Chatchawan Kotarasu, and Chenyi Zhuang. A general view for network embedding as matrix factorization. In *Proceedings of the Twelfth ACM international conference on web search and data mining*, pages 375–383, 2019.

[63] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *The Journal of Machine Learning Research*, 11:1201–1242, 2010.

[64] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *CoRR*, abs/1903.11835, 2019.

[65] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5:1–42, 2020.

[66] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[67] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arxiv pre-print*, 11 2015.

[68] Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. Convolutional graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 452–456. IEEE, 2019.

[69] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. *arXiv preprint*, 12 2013.

[70] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

[71] Muhammet Balcilar, Renton Guillaume, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages –, 2021.

[72] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Processing Magazine*, 37(6):128–138, 2020.

[73] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[74] Yundong Li, Hongguang Li, and Hongren Wang. Pixel-wise crack detection using deep local pattern predictor for robot application. *Sensors*, 18:3042, 09 2018.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[76] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[77] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[79] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[80] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[81] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[82] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6824–6835, 2021.

[83] Michael Egmont-Petersen, Jan L Talmon, Arie Hasman, and Anton W Ambergen. Assessing the importance of features for multi-layer perceptrons. *Neural networks*, 11(4):623–635, 1998.

[84] GM Foody. Supervised image classification by mlp and RBF neural networks with and without an exhaustively defined set of classes. *International Journal of Remote Sensing*, 25(15):3091–3104, 2004.

[85] Ce Zhang, Xin Pan, Huapeng Li, Andy Gardiner, Isabel Sargent, Jonathon Hare, and Peter M Atkinson. A hybrid mlp-cnn classifier for very fine resolution remotely sensed image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:133–144, 2018.

[86] Chao Shang, Qinqing Liu, Ko-Shin Chen, Jiangwen Sun, Jin Lu, Jinfeng Yi, and Jinbo Bi. Edge attention-based multi-relational graph convolutional networks. *arXiv preprint arXiv: 1802.04944*, 2018.

[87] Haojie Hu, Minli Yao, Fang He, and Fenggan Zhang. Graph neural network via edge convolution for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 19:1–5, 2021.

[88] Jonathan Masci, Ueli Meier, Dan Ciresan, Jürgen Schmidhuber, and Gabriel Fricout. Steel defect classification with max-pooling convolutional neural networks. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2012.

[89] Jinsong Zhu and Jinbo Song. An intelligent classification model for surface defects on cement concrete bridges. *Applied Sciences*, 10(3):972, 2020.

[90] Yong Shi, Limeng Cui, Zhiquan Qi, Fan Meng, and Zhensong Chen. Automatic road crack detection using random structured forests. *IEEE Transactions on Intelligent Transportation Systems*, 17(12):3434–3445, 2016.

[91] Liang Yang, Bing Li, Wei Li, Liu Zhaoming, Guoyong Yang, and Jizhong Xiao. Deep concrete inspection using unmanned aerial vehicle towards cssc database. 03 2017.

[92] Marc Maguire, Sattar Dorafshan, and Robert J. Thomas. Sdnet2018: A concrete crack image dataset for machine learning applications. 2018.