Technische Universität Berlin

Faculty of Electrical Engineering and Computer Science
Dept. of Computer Engineering and Microelectronics
Remote Sensing Image Analysis Group

Graph Data Augmentation in Remote Sensing Image Classification

Master of Science in Computer Science

July, 2023

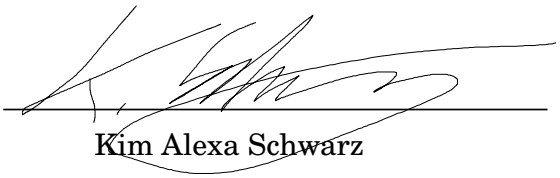
Kim Alexa Schwarz

Matriculation Number: 381838

Supervisor: Prof. Dr. Begüm Demir
Advisor: Tom Burgert

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

UNTERSCHRIFT: 

Kim Alexa Schwarz

DATUM: 04.07.2023

ORT: Berlin

Abstract

Graph Neural Networks (GNNs) are powerful machine learning tools that can model the structural information of graph data. They enable state-of-the-art performance on various graph-related tasks but greatly depend on the quality and quantity of training data. Thus, graph data augmentation (GraphDA) techniques are proposed to tackle this issue. However, the effects of GraphDAs vary based on the type of graphs and their domain. While most research is conducted for classical graph data like molecules or social networks, the field of GraphDAs for remote sensing (RS) image graphs remains unexplored. Therefore, this thesis investigates the usage of GraphDAs for RS multi-label classification (MLC) based on superpixel image graphs. In particular, three novel GraphDAs are proposed and thoroughly evaluated together with existing GraphDAs on two RS MLC datasets. The experimental results indicate that the effect of GraphDAs greatly varies across the datasets, even though the image graphs are generated in the same way. Only the random drop of graph edges consistently improves the performance on all datasets. In detail, even for classes of the same dataset, the effects of GraphDA vary significantly. While the newly proposed GraphDAs cannot improve the overall performance, they positively affect some classes. These differences can be attributed to the underlying characteristics of individual classes that are either texturised classes with repetitive features or object-based classes with individual features.

Zusammenfassung

Graph Neural Networks (GNNs) sind mächtige Werkzeuge des maschinellen Lernens, die die strukturellen Informationen von Graphen erfassen können. Sie erzielen State-of-the-Art Ergebnisse auf verschiedensten graphbezogenen Aufgabenbereichen. Aber dennoch sind sie abhängig von der Qualität und Quantität der Trainingsdaten. Um dem entgegenzuwirken, werden Graph Data Augmentation (GraphDA) Methoden entwickelt. Allerdings variieren deren Effekte stark in Abhängigkeit vom Aufbau und dem Anwendungsgebiet, aus dem die Graphen stammen. Während ein Großteil der Forschung sich auf Daten von klassischen Graphen bezieht, wie z.B. Moleküle oder soziale Netzwerke, gibt es bisher noch keine Forschung bezüglich GraphDA im Kontext von Remote Sensing (RS) Bild Graphen. Auf Grund dessen, untersucht diese Masterarbeit die Anwendung von GraphDA in RS Multi Label Classification (MLC) basierend auf Superpixel Bildgraphen. Drei neue GraphDA Techniken werden vorgestellt und zusammen mit bereits existierenden GraphDA Techniken auf zwei verschiedenen RS MLC Datensätzen evaluiert. Die Ergebnisse zeigen, dass die Effekte der Methoden stark auf den Datensätzen variieren, auch wenn die Graphen alle auf dieselbe Art und Weise erzeugt werden. Nur das zufällige Wegfallen von Kanten im Graph verbessert die Performance konsistent. Variationen können sogar bei verschiedenen Klassen im gleichen Datensatz beobachtet werden. Während die neuen Methoden kaum zur Verbesserung der allgemeinen Performance beitragen, zeigen sie positive Effekte auf einzelne Klassen. Diese Unterschiede werden durch die einzigartigen Charakteristiken der einzelnen Klassen hervorgerufen, die entweder texturiert sind, und sich durch wiederholende Muster auszeichnen, oder objektbasiert sind mit individuellen Merkmalen.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

APmac ..... Macro Average Precision

APmic ..... Micro Average Precision

BR ..... Boundary Recall

CNN ..... Convolutional Neural Network

CV ..... Computer Vision

DA ..... Data Augmentation

GCN ..... Graph Convolutional Network

Gconv ..... Graph Convolution

GNN ..... Graph Neural Network

GraphDA ..... Graph Data Augmentation

LSC ..... Linear Spectral Clustering

MC ..... Markov Clustering

ML ..... Machine Learning

MLC ..... Multi-Label Classification

MLP ..... Multi-Layer Perceptron

RAG ..... Region Adjacency Graph

RandNodeDrop Random Node Dropping

RandSquare .. Graph Random Square

ReLU ..... Rectified Linear Unit

RS ..... Remote Sensing

SEEDS ..... Superpixels Extracted via Energy-Driven Sampling

SLIC ..... Simple Linear Iterative Clustering

UE ..... Undersegmentation Error

1 | Introduction

Graph Neural Networks (GNNs) have gained increasing popularity due to their ability to model the structural information of data. They can be applied to any form of graph data and have shown state-of-the-art performance in various graph-related tasks, such as node, edge, and graph classification [1], [2]. Due to their flexibility, they have been used in various fields, including molecular biology, chemistry, and social networks, as well as computer vision (CV) and remote sensing (RS), where they can also be applied to image data [3].

Unlike images, which are only able to model the geometric information of data, graphs are abstract representations that can model more complex non-euclidean structural information. An image can be easily transformed into a graph structure by sampling clusters of pixels as nodes and connecting them with edges based on their neighbourhood relationship [3]–[5].

In recent years, different GNN architectures have been proposed in the deep learning community. Yet, like other neural networks, their performance greatly depends on the quality and quantity of training data. However, acquiring diverse labelled data is time-consuming and expensive. Thus, data augmentation is generally used to generate slightly modified or synthetically created data from already existing samples, and it has been shown on numerous occasions that it can drastically increase the performance of neural networks [6], [7].

When dealing with image data, data augmentation strategies include simple geometric transformations, like cropping and shearing, as well as channel transformations that slightly alternate brightness, contrast, or colour [8] and more complex methods that morph two samples into a new one [9], [10].

Yet, these transformations cannot be directly applied to graph data due to their complex and unique nature [1], [2]. Hence, an increasing amount of graph data augmentation (GraphDA) techniques has been proposed in recent years. Although their results have been promising, GraphDA remains under-explored and developing new

techniques is challenging. Graphs encode very different information depending on the application domain. Hence, a data augmentation that works great in one domain may negatively affect model performance in another. For example, dropping edges through edge perturbation from a molecule graph can drastically alter structural information, resulting in an entirely different molecule. As a consequence, the ground truth label would be incorrect [2], [11]. You et al. [1] introduced a framework to pre-train GNNs in a self-supervised manner by utilising graph augmentations, which boosted the performance of their classification approach. Additionally, they proposed four different graph augmentation techniques and justified them using human priors. Albeit they tested their framework on different domains, their tests on image datasets are very limited.

To the best of our knowledge, little research has been done on GraphDA for image graphs, and GraphDA has yet to be studied in the context of RS image analysis. Due to the high semantic complexity of RS satellite images, i.e., the high number of classes that can appear within an image and its larger channel size in comparison to other fields [8], transforming image data into graphs can be particularly useful in the RS domain. As not all pixels hold equally important information for the learning process, graphs may potentially reduce the size and complexity of the data while preserving meaningful information [4].

This thesis will focus on evaluating GraphDA's influence on multi-label classification (MLC) in RS. To this end, we transform multi-label RS images into graphs and predict the graph labels with a Graph Convolutional Network (GCN). We then re-implement the augmentations proposed by You et al. [1] and propose three new GraphDA techniques motivated by image augmentations. Finally, GCN and GraphDAs are evaluated on different MLC RS datasets.

Chapter 2 will introduce essential background knowledge on GNNs, data augmentation (DA) in image processing and GraphDAs. Chapter 3 offers a summary of the datasets used throughout this thesis. Next, our approach is outlined in Chapter 4. In particular, it provides an overview of the MLC pipeline and then delves deeper into the specific steps, including superpixel sampling, graph generation and graph augmentation. In the following chapter, we describe our experimental setup and report our results for superpixel segmentation and the proposed GraphDAs. Afterwards, Chapter 7 compares the performance of our approach on texture and object-based classes. Finally, this thesis concludes by summarising the key developments and offering a short outlook for future research.

2 | Background

This chapter introduces important concepts of GNNs, with a specific emphasis on GCNs, a subclass of GNN utilised throughout this thesis. Subsequently, an overview of image DA is provided, along with relevant related work on GraphDA.

2.1 Graph Neural Networks

While much work in Machine Learning (ML) focuses on Euclidean data, such as images, there has been an increasing interest in graphs due to their ability to model the structural information of data in a wide range of application fields, such as molecular biology, chemistry and social networks as well as CV and RS [3].

Graphs, however, can be irregular. In a graph, entities, e.g. pixels or atoms in a molecule, are represented by nodes connected by different types of links, also called edges. A graph's size can vary, and each node can have a different degree, which is defined as the number of neighbours it is directly connected to. As a result, some important operations, including convolutions, are difficult to apply to graph-structured data [12].

Various GNN architectures exist today, and many of them broadly follow a neighbourhood aggregation (or iterative message-passing scheme) to solve graph-related tasks [13], [14]. That means each node repeatedly exchanges information with its neighbours along its edges with an aggregation function that varies with the specific architecture [13]. Thus, the network can capture the structural information of the nodes' neighbourhoods. The graph-related tasks can be categorised into node-level, edge-level and graph-level. Node-level tasks focus on predicting single nodes, e.g. some nodes are annotated with labels, and the objective is to classify the unlabelled nodes based on the given information [15]. Edge-level tasks aim to predict the connections between the nodes [16]. And finally, graph-level tasks aim to classify the graph as a whole [17].

In this thesis, we perform graph-level MLC and use the commonly used GCN proposed by Kipf and Welling [15], an example of which is illustrated in Fig. 2.1. The depicted

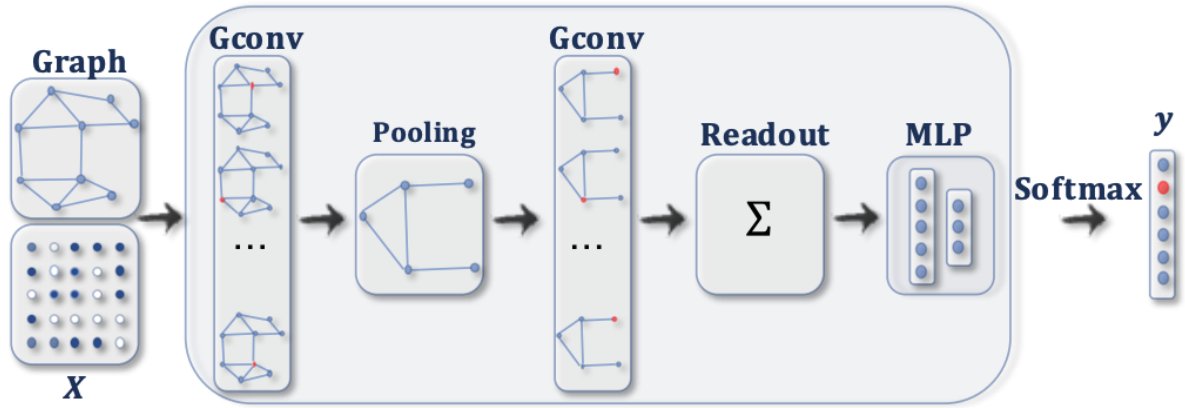


Figure 2.1: A GCN for graph classification with two graph convolutional layers, a pooling layer, a readout layer to summarise the graph information, and an MLP to make the final label prediction [12].

GCN consists of 2 graph convolutional (Gconv) layers, a graph pooling layer, a readout function and a Multi-Layer Perceptron (MLP). The individual components are described in the following. GCN takes an undirected input graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with nodes $\mathcal{V}$ connected by edges $\mathcal{E}$ and a feature matrix $X^{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}| x F}$ that assigns a node $\mathcal{V}$ a F-dimensional feature vector as input and outputs a graph prediction y.

Each graph convolutional layer applies a form of graph convolution as the aggregation method for message passing, which is inspired by the convolutional operation in the image domain. However, since nodes are unordered and have varying node degrees, convolutional kernels can not simply be defined on graphs. Graph convolutions aggregate the neighbours of a node by combining and averaging across the current node's features and the features of the neighbours. This is illustrated in Fig. 2.2, which compares the 2D convolutions in the image domain with the graph convolutions.

Let, $H^{(\ell)} \in \mathbb{R}^{|\mathcal{V}| \times D}$ be the the activation matrix of the ℓ-th convolutional layer with $H^{(0)} = X^{\mathcal{G}}$. $\tilde{A}$ is defined as the adjacency matrix A of $\mathcal{G}$ with added self-loops to ensure that the node's own features are included in the aggregation. Thus, $\tilde{A} := A + I_{|\mathcal{V}|}$ where $I_{|\mathcal{V}|}$ is the identity matrix. Let $\tilde{D}$ be the degree matrix, with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $W^{(\ell)} \in \mathbb{R}^{F_{\ell-1} \times F_\ell}$ the weight matrix of the ℓ-th layer with $F_{\ell-1}$ input features and F_ℓ output features. Kipf and Welling [15] proposed the following layer-wise propagation rule for their graph convolutions:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}\right), \tag{2.1}$$

where σ is a non-linear activation function, such as Rectified Linear Unit (ReLU).

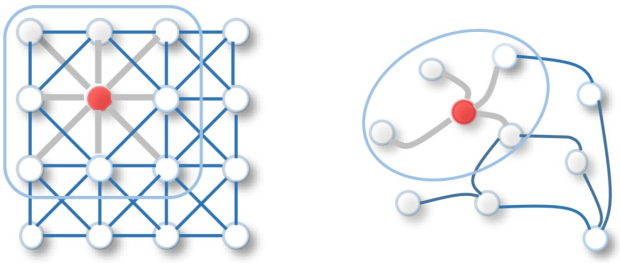


Figure 2.2: Comparison between the convolutional operation in the image (**left**) and the graph domain (**right**). **Left**: A kernel is centred at the red pixel. A 2D convolution then computes the weighted average across all pixels within the kernel. The pixels are ordered, and each pixel's neighbourhood size is the same. **Right**: A graph convolution takes the average across the red node and all its neighbours. The nodes are unordered and have a variable degree [12].

By multiplying $\tilde{D}$ with $\tilde{A}$ and the hidden node feature representation of the previous layer $H^{(\ell)}$, aggregation is performed for all nodes in $\mathcal{V}$. Additionally, they apply symmetric normalisation to $\tilde{A}$ to ensure numeric stability and minimise the vanishing/exploding gradient problem for a large variety in the node degrees, which leads to the formulation $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(\ell)}$ in Eq. 2.1 [15]. After the first forward pass, the nodes are then represented by a feature vector containing information aggregated across the immediate neighbourhood.

If we apply another layer after the first, the same operation is performed, but on the updated node representations. Therefore, in the second layer, a node does not only receive information about their immediate but also about their neighbourhood 2-hops away. In a nutshell, stacking more layers to the network does increase its' local receptive field and enables it to capture larger structures [12], [15].

In Fig. 2.1, a pooling layer is applied in between graph convolutional layers to coarsen the graph structure. For instance, max-pooling reduces the graph by taking the channel-wise maximum across the node dimensions in a cluster [18]. After the last graph convolutional layer, a readout function is applied to the graph, flattening the graph into a vector which can then be fed into a Multi-Layer Perceptron (MLP). Finally, we apply softmax to the MLP output to receive the graph classification result.

Note that the steps we take after receiving the output from the graph convolutional layers can significantly differ depending on the task at hand. Since our objective is to make multi-label predictions for the entire graph, we want the model to output an MLC vector classifying the entire graph, but the aim of a node-level task could be to output the input graph with completely annotated nodes. The readout function, MLP, and softmax would be omitted in that case [12].

2.2 Image Data Augmentation

Before introducing GraphDA, we give a short overview of important concepts of image data augmentation, as well as augmentation in RS, and present image DA techniques, which inspired or correlate to GraphDA techniques proposed in Sec. 4.4.

A deep learning model's performance greatly depends on the quality and quantity of training data. One common challenge when designing new models is overfitting, which happens when a model can not generalise well to new unseen data. Additionally, real-world datasets are susceptible to large image variations such as differences in illumination and deformation errors [19], further emphasising the importance of having a diverse training dataset.

One approach to tackle these challenges is to augment the training data. DA can artificially enlarge the training dataset and increase diversity, thus improving the generalisation capabilities of the model [19]. Indeed, it has been shown numerous times that DA can drastically increase the performance of neural networks [6], [7]. Many augmentation techniques have been proposed in the past, including simple image transformations, such as geometric transformations like cropping and shearing, as well as more sophisticated approaches, e.g. utilising Generative Adversarial Networks [8], [19]. Most traditional augmentation strategies aim to create novel and realistic data to preserve their labels, i.e. the semantic information is still recognisable to a human observer [20]. These augmentations include geometric and colour transformations.

However, more recently, new augmentation techniques have been proposed that artificially mix images such that the newly resulting samples do not appear useful to a human observer. These multi-sample synthesis approaches, albeit very efficient, are not intuitive, and the performance boost is difficult to interpret [19], [21]. Multi-sample synthesis includes methods such as Mixup [9] and nonlinear blending methods [10].

Non-linear blending methods are a more general collection of multi-sample synthesis augmentations. A new sample is created by combining two random images in different ways. The labels are then merged with linear interpolation to create a new label. Fig. 2.3 illustrates multiple different strategies, namely Vertical Concat, Horizontal Concat, Mixed Concat, Random 2x2, VH-Mixup/BC+, Random Square, Random Column Interval, Random Row Interval, Random Rows, Random Columns, Random Pixels, Random Elements, Noisy Mixup, and Mixup. Almost all of these augmentations lead to a significant performance improvement [21].

Let x_i, x_j be two random image samples. Vertical Concat, for instance, concatenates

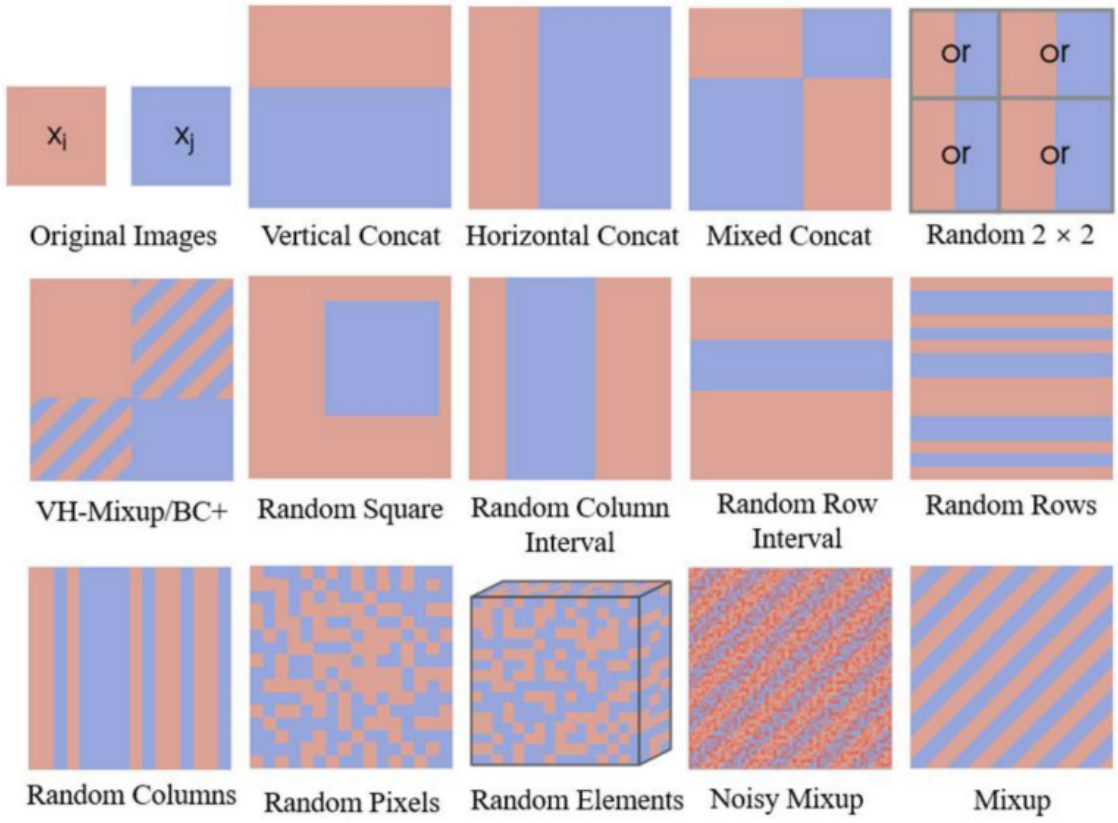


Figure 2.3: Examples of non-linear blending methods for two samples x_i and x_j [21].

the top λ fraction of x_i with the bottom $(1-\lambda)$ fraction of x_j, whereas Random Square cuts out a portion of x_i and replaces it with the same portion of x_j [10]. For more information on the non-linear blending methods that have not been explained here, refer to [10].

Data augmentation has been applied across various domains, including RS. RS images come with unique challenges for data augmentation. They are highly complex and often large. Also, most augmentations are defined in the RGB colour space and do not consider the specific RS imagery types [8]. Thus, the efficiency of specific data augmentation techniques varies significantly. For example, colour transformation can hinder the model performance on RS images since colour is essential in the RS domain. Geometric transformations, on the other hand, are easy to apply but do not add additional semantic information to the data. Hence, their performance improvement is limited [8], [21], [22]. However, it has been shown that RS images (including hyperspectral images) can profit from cropping and local erasure, which can make the model more robust against occlusion, often caused by cloud cover [21], [23].

2.3 Graph Data Augmentation

The augmentations presented in Sec. 2.2 can not simply be adapted for graph data since graph data is non-euclidean. However, GNNs (including GCNs) share many of the challenges affecting other ML models, such as the dependence on high-quality training data and the high cost of annotation efforts, which are addressed by DA. When ground truth labels are missing, GNNs can easily overfit, resulting in a low generalisation capability. Additionally, real-world graphs are extracted from complex systems, which can result in inconsistent and unclean graphs with redundant and missing features as well as connections. Naturally, if the quality of input graphs is low, the performance of the GNN is limited [2], [11], [24].

To tackle those issues, and inspired by the progress made in other ML domains, researchers have become increasingly interested in developing GraphDA techniques. Common techniques include structure-oriented augmentation, such as edge perturbations [1], [25], node dropping, and subgraph sampling [1], [26], or feature-oriented augmentations, such as feature masking [1], [27], where certain elements in the node feature matrix $X^{\mathcal{G}}$ are set to 0. Furthermore, label-oriented augmentations aim to enlarge the limited label space of the training data by, for example, interpolating between existing labels to create new ones [28].

More and more methods are being proposed, and they have shown positive effects on the generalisation capabilities of GNNs [11]. Yet, the unique nature of graphs hampers the development of meaningful GraphDAs. Graphs can represent greatly dissimilar information depending on the domain they originate from. A DA which positively affects performance in one domain might hinder performance in another [2], [11].

To minimise the risk, GraphDA has to be carefully applied, and further research is necessary to evaluate the influence of various GraphDA techniques on different domains, such as RS, where no studies have been conducted so far.

3 | Dataset Description

This chapter offers a short overview of the two satellite image datasets, DeepGlobe and UCMerced Land Use Dataset, utilised during this thesis and the necessary pre-processing steps. The first section introduces DeepGlobe, a land cover classification, and segmentation dataset, which we transform for MLC, while the second presents the multi-label UCMerced Land Use Dataset.

3.1 DeepGlobe

DeepGlobe [29] is a land cover classification and RS dataset released in 2018, which is commonly used for segmentation, object detection, and landcover classification. It contains 1446 2488x2488 pixels high-resolution satellite images and ground truth segmentation maps, a few examples of which can be seen in Figure 3.1.

The segmentation maps are annotated with seven landcover classes: *Agriculture*, *rangeland*, *barren*, *forest*, *water*, *urban* and *unknown*. Before we transform the images into superpixel segmentation maps, each patch is split into 112x112 pixels images and

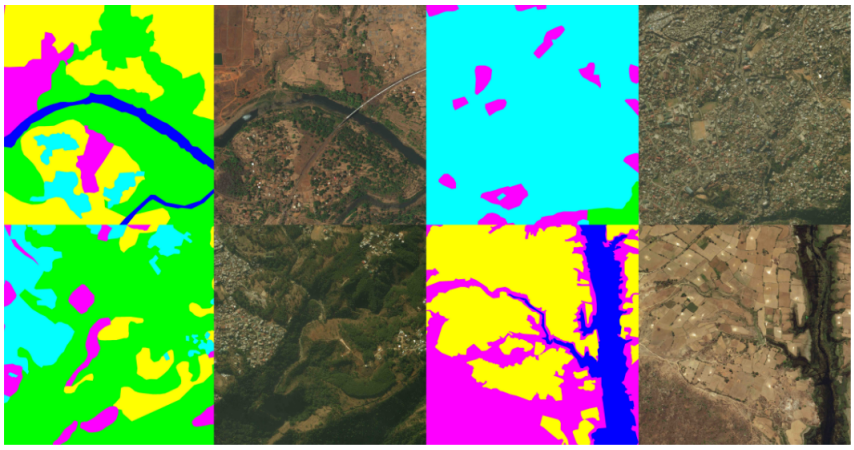


Figure 3.1: Example images and segmentation maps of DeepGlobe.

annotated based on the class occurrences in the ground truth segmentation for the corresponding image, resulting in approximately 223 000 multi-label images. The labels are then transformed into one hot-encoded vectors.

We then generate three new datasets with different complexity to evaluate the label complexity's influence on our models' performance. Label complexity is measured by the number of different classes occurring in a single image. Each dataset contains around 28 000 images. The class distribution of the datasets can be seen in Table 3.1. Globe 1.5 has an average class count per sample of 1.5, Globe 2 of 2.13, and Globe 2.5 of 2.48. Notably, all three datasets have a very similar class distribution, with *agriculture* being the biggest class, followed by *urban* and *rangeland*.

Finally, each dataset is split into 80% training, 16% validation, and 4% test data. The datasets will be referred to as Globe15, Globe2 and Globe25 from now on, while Globe is the overarching term for the three pre-processed datasets.

Table 3.1: Total number of samples and samples per class for each dataset.

Classes	Globe 1.5	Globe 2	Globe 2.5
Urban	6671	11444	12555
Agriculture	18364	19004	20013
Rangeland	6962	13325	16332
Forest	3924	4623	4674
Water	2581	4573	7709
Barren	3904	5819	6887
Unknown	208	366	1102
Total	27929	27823	27865

3.2 UCMerced Land Use Dataset

UCMerced Land Use Dataset [30], denoted as UCMerced, is a high-resolution multi-label RS dataset containing 2100 images. Each image measures 256x256 pixels and is annotated with multi-labels. UCMerced includes 17 classes, namely: *Bare-soil*, *buildings*, *cars*, *chaparral*, *court*, *dock*, *field*, *grass*, *mobile-home*, *pavement*, *sand*, *sea*, *ship*, *tanks*, *trees*, and *water*. Furthermore, ground truth segmentation maps are available [31]. An example image for each of these classes can be seen below (Figure 3.2).

As illustrated in Figure 3.3 the dataset mostly consists of *buildings*, *pavement*, *grass* and *trees*, followed by *water* and *sand* and multiple smaller object and texture-based classes. We split the dataset into 70% training, 15% validation and 15% test data.

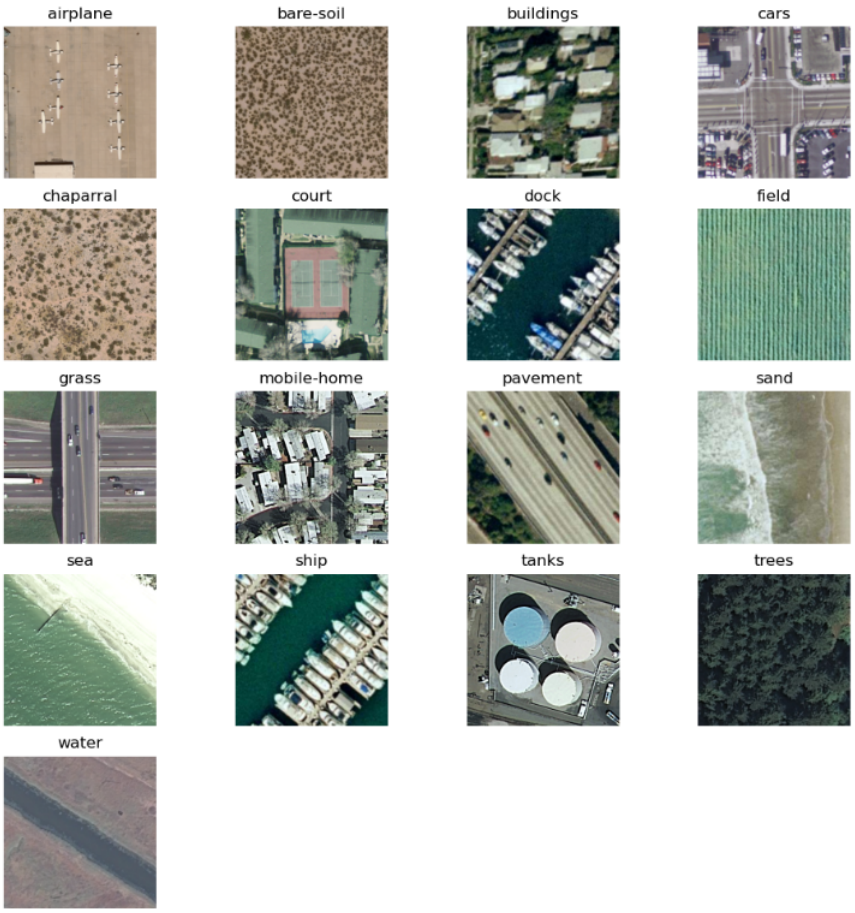


Figure 3.2: Example images from UCMerced that contain the denoted class. For visualisation purposes, only one class per multi-label is given.

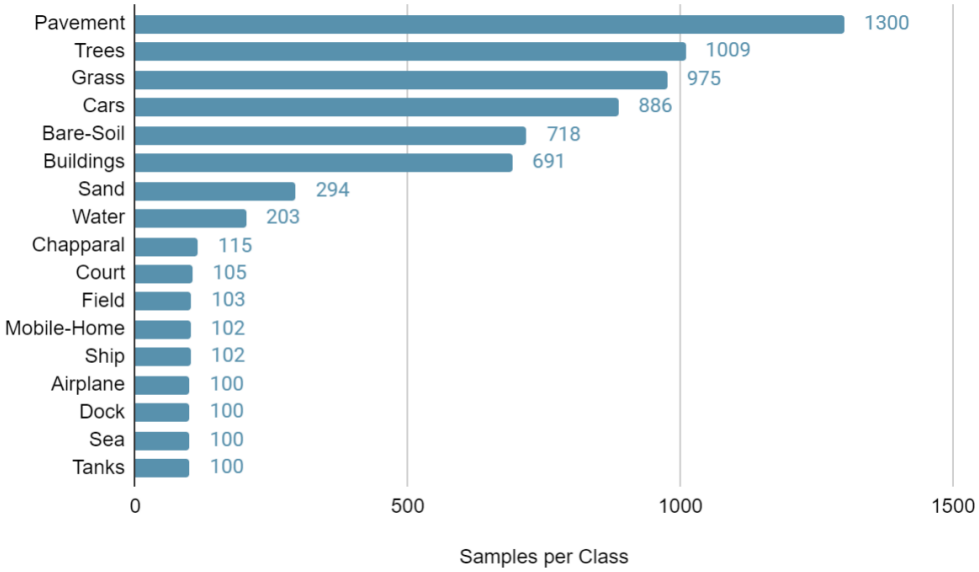


Figure 3.3: Number of images that contain a class label in the UCMerced dataset.

4 | Methodology

The objective of this thesis is to evaluate the influence of different graph data augmentations on the performance of GCN in the RS domain for MLC. To this end, we need to transform an image into a graph. In the following, we will present the general pipeline of our approach, provide a more detailed description of our graph generation method and describe already existing and newly proposed graph data augmentations.

4.1 Pipeline

Figure 4.1 illustrates the complete technical pipeline and can be split into three parts. Let x_i be defined as a sample image and $y_i \in \{0,1\}^L$ as its corresponding binary multi-label vector, where an element $y_{i,l}$ equals 1 if a class $l \in L$ is present in the image with L being the set of classes in a given dataset. A superpixel algorithm SPA is applied to obtain a

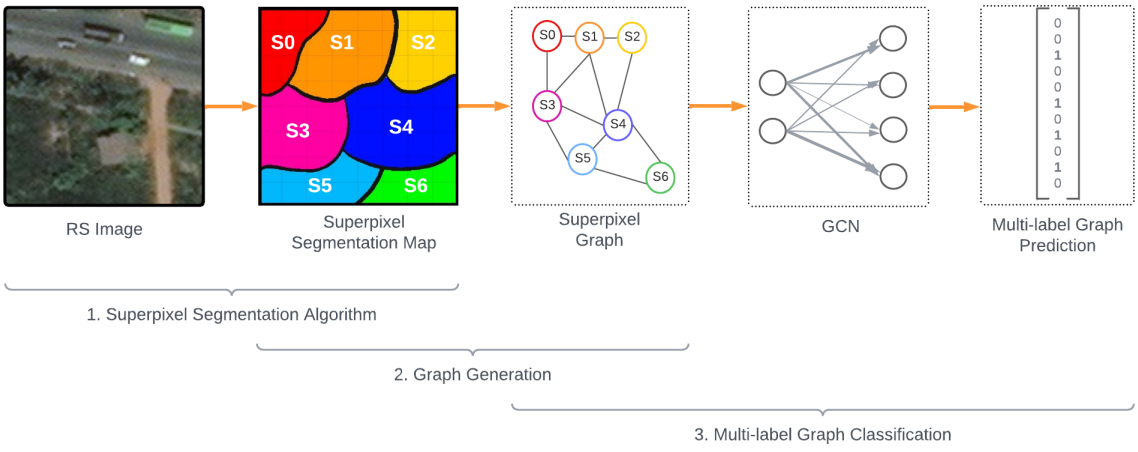


Figure 4.1: Complete Pipeline: First, an RS image is sampled into superpixels, then the superpixels are transformed into a graph, and finally, a GCN is trained on the graphs and applied to the data to make multi-label predictions on graphs.

superpixel segmentation map SSM_i.

$$SSM_i = SPA(x_i). \tag{4.1}$$

Then, an undirected graph $\mathcal{G}_i$ with multi-label vector y_i is generated based on SSM_i with graph generation algorithm GGA (Section 4.3). Thus:

$$\mathcal{G}_i = GGA(SPA(x_i)). \tag{4.2}$$

Finally, we train a GCN on sample pairs $\{\mathcal{G}_i, y_i\}$ to make multi-label predictions. The augmentations are applied during training (Section 4.4).

4.2 Superpixel Sampling

The first step of generating graphs from images is to segment the image into superpixels. Multiple different algorithms have been proposed in the past [32]. To select the best algorithm for the given dataset and gain a better understanding of the influence the choice of segmentation algorithm can have on the GCN performance, we evaluate four different superpixel algorithms, namely Simple Linear Iterative Clustering (SLIC) [33], Superpixels Extracted via Energy-Driven Sampling (SEEDS) [34], Linear Spectral Clustering (LSC) [35] and Quickshift [36].

SLIC is a superpixel algorithm based on K-means-clustering that segments an image into K approximately equally sized superpixel, where K is pre-defined by the user. First, SLIC translates an image into the CIELAB colour space. Then the clustering process is initialised with K initial cluster centres $Z_i = (l_i, a_i, b_i, x_i, y_i)$ sampled on a grid in the image space $\sqrt{\frac{Q}{K}}$ pixels apart, where Q the total number of pixels in the image. In every iteration, each pixel is assigned to its nearest cluster centre, which is determined by a distance function d and a new cluster centre is computed based on the mean (l, a, b, x, y) vector of all pixels in the cluster. A residual error determines when to complete the iterations with the L_2 norm between the new and old cluster centre locations [33]. Examples for SLIC are visualised in Fig. 6.1.

SEEDS is an energy optimisation algorithm [32] that uses hill-climbing to iteratively adjust the superpixel boundaries and maximise the energy. Similar to SLIC, SEEDS initialises clusters on a regular grid. However, instead of computing distance measures between pixels and clusters, it exchanges pixels between clusters by moving the boundaries. Each superpixel is described as a region with a colour distribution and a boundary shape. A high-quality superpixel groups pixels with high similarity that belong to the

same object and adheres to object boundaries. Let SG be the set of all valid superpixel partitioning. The aim is to find the partitioning $s \in SG$ that maximises the energy function $E(s) = T(s) + \gamma B(s)$, where $T(s)$ is the likelihood of the colour space that evaluates the quality of colour density distribution and $B(s)$ a prior of the superpixel boundaries that penalises irregularities in the superpixel boundaries [34].

LSC is a cluster-based algorithm that uses weighted K-means clustering in a 10-dimensional weighted feature space. After the pixels have been mapped into the feature space with a kernel function, K seed pixels are uniformly sampled across the whole image and used as search centres. Their feature vectors represent the initial weighted means of their clusters. Each cluster has a search space $ra_x \times ra_y$, where $\frac{a_x}{a_y}$ is defined as the aspect ratio of the image, limited with parameter $r \geq 1$. Until convergence, pixels are iteratively added to the clusters their vectors are closest to in the feature space, and the clusters are updated accordingly [35].

Quickshift, on the other hand, is a density-based algorithm that does not allow the pre-definition of K. First, a density estimate is computed for the whole image. The objective is to increase the estimate by moving each point to the nearest neighbour, which increases the density. To this end, Quickshift connects all pixels into a tree, where the parents represent the nearest neighbours. The tree is then split into a forest containing smaller trees by breaking the branches that are longer than a given threshold τ. Finally, each subtree in the forest forms a distinct cluster [36].

To evaluate the performance of the superpixel algorithms, we compute the boundary recall (BR) (Eq. 4.3) and undersegmentation error (UE) (Eq. 4.4) with the ground truth segmentation maps [37].

BR is the fraction of ground truth boundaries within a pre-defined distance d of a superpixel boundary. Let TP be defined as the true positives, as in the number of boundary pixels in the ground truth segmentation GT that also exist in the superpixel segmentation, and FN, the false negatives, be the number of boundary pixels in GT that do not exist in the segmentation, then:

$$BR = \frac{TP}{TP + FN}. \tag{4.3}$$

Additionally, UE measures how much superpixels spill across the ground-truth segment boundaries. A ground truth segment GS splits a superpixel P into P_{in} and P_{out} depending on which pixels of P lie within or out of GS. To not penalise large superpixels with only a small boundary overflow, we either omit P_{in} or append P_{out} to the segment,

depending on which produces the smaller error. With $Q \in \mathbb{N}$ defined as the total number of pixels in the image, this yields the following equation [37]:

$$UE = \frac{1}{Q}\left[\sum_{GS \in GT}\left(\sum_{P:P\cap GS\neq\varnothing} \min(P_{\text{in}}, P_{\text{out}})\right)\right]. \tag{4.4}$$

We test different hyperparameters for all four superpixel algorithms. The results are presented in Sec. 6.1.

4.3 Graph Generation

After the image has been segmented into superpixels, it is transformed into an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ consisting of nodes $\mathcal{V}$ connected by edges $\mathcal{E}$. In our superpixel graph, each node represents a superpixel and contains the positional arguments of the superpixel centroid as x- and y-coordinates, as well as the across the superpixel averaged image channels (C) as node features. The corresponding node feature matrix $X^{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}|\times C}$ assigns each node a C-dimensional feature vector.

Throughout this thesis, we use Region Adjacency Graphs (RAG) [17]. Nodes are connected by an edge if the superpixels they represent are neighbours in the image, i.e. if they share a mutual border. We define the edge weights as the cosine similarity sim between the nodes' features, as seen in Eq. 4.5.

$$sim = \frac{x_1^{\mathcal{G}} \cdot x_2^{\mathcal{G}}}{\max\left(\left\|x_1^{\mathcal{G}}\right\|_2 \cdot \left\|x_2^{\mathcal{G}}\right\|_2, \epsilon\right)}, \tag{4.5}$$

with $x_1^{\mathcal{G}}$ and $x_2^{\mathcal{G}}$ being the feature vectors of two nodes connected by an edge and ϵ a small value to avoid division by 0.

Other popular algorithms for graph generations based on superpixels include Radius-Graph and kNN-Graph [1], [38], [39]. However, both of them have their disadvantages. Radius-Graph samples neighbours based on if they lie within a certain radius of the node position. Since, however, some superpixels can be of irregular shape, depending on their compactness, defining the position of the centroid can be challenging, which could result in an inaccurate graph. kNN-Graph, on the other hand, samples neighbours not only based on their node positions but also on the similarity of their features. Consequently, not all kNNs superpixels are actual neighbours in the image but can be scattered far away from each other if their values are similar. Additionally, the node degree is fixed,

which results in a less flexible graph structure. RAG, however, allows varying node degrees and accurately represents the superpixel relations.

4.4 Graph Augmentations

In addition to the four graph augmentations proposed by You et al. [1], namely node dropping, edge perturbation, subgraph, and attribute masking, we further propose four new augmentations which have been inspired by similar augmentations in the image domain: Graph Random Square, Graph Vertical Concat, Graph MixUp and Graph ReOrga.

For all augmentations introduced in the following subsections, let $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)$ and $\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B)$ be two distinct undirected superpixel graphs generated by RAG and sampled from the same dataset with their respective ground truth label y_A and y_B as defined in Sec. 4.1.

4.4.1 Graph Augmentations by You et al.

In addition to proposing our own graph augmentations, we also re-implement and evaluate the four graph augmentations proposed by You et al. [1]:

1. **Node dropping**: Node dropping randomly discards nodes from a graph with a probability $p \in [0,1]$ along with all their connections.

2. **Edge perturbation**: Edge perturbation randomly removes or add edges with a probability $p \in [0,1]$. When adding an edge, we assign the similarity of the connected nodes as the edge weight.

3. **Subgraph**: Starting from a random node, sample a subgraph from $\mathcal{G}$ with a random walk algorithm and discard all edges and nodes which are not part of the subgraph.

4. **Attribute masking**: Node attributes x are randomly masked with a probability $p \in [0,1]$.

4.4.2 Graph Random Square

Graph Random Square (*RandSquare*) is inspired by the non-linear blending method with the same name, "Random Square" [10], which is further described in Sec. 2.2. However,

instead of merging the label of both graphs, the label of $\mathcal{G}_A$ is preserved since we do not know which classes are actually inherited from $\mathcal{G}_B$.

A square can be easily defined for an image since it is a structured data type, a graph, on the other hand, is unstructured. However, since $\mathcal{G}_A$ and $\mathcal{G}_B$ are superpixel graphs, every node is assigned a position based on the superpixel's centroid in the image. Thus, we can easily perform Random Square on this type of graph. First, we pick a random node $v_0 \in \mathcal{V}_A$ from $\mathcal{G}_A$ while ensuring that the node index exists in both graphs since the graphs are most likely not the same size. Next, we sample all nodes of $\mathcal{G}_A$ whose positions lie within a 16x16 square centred at the positional arguments of v_0. Let $\mathcal{I}$ be defined as the set that contains the indices of the selected nodes. Each node with an

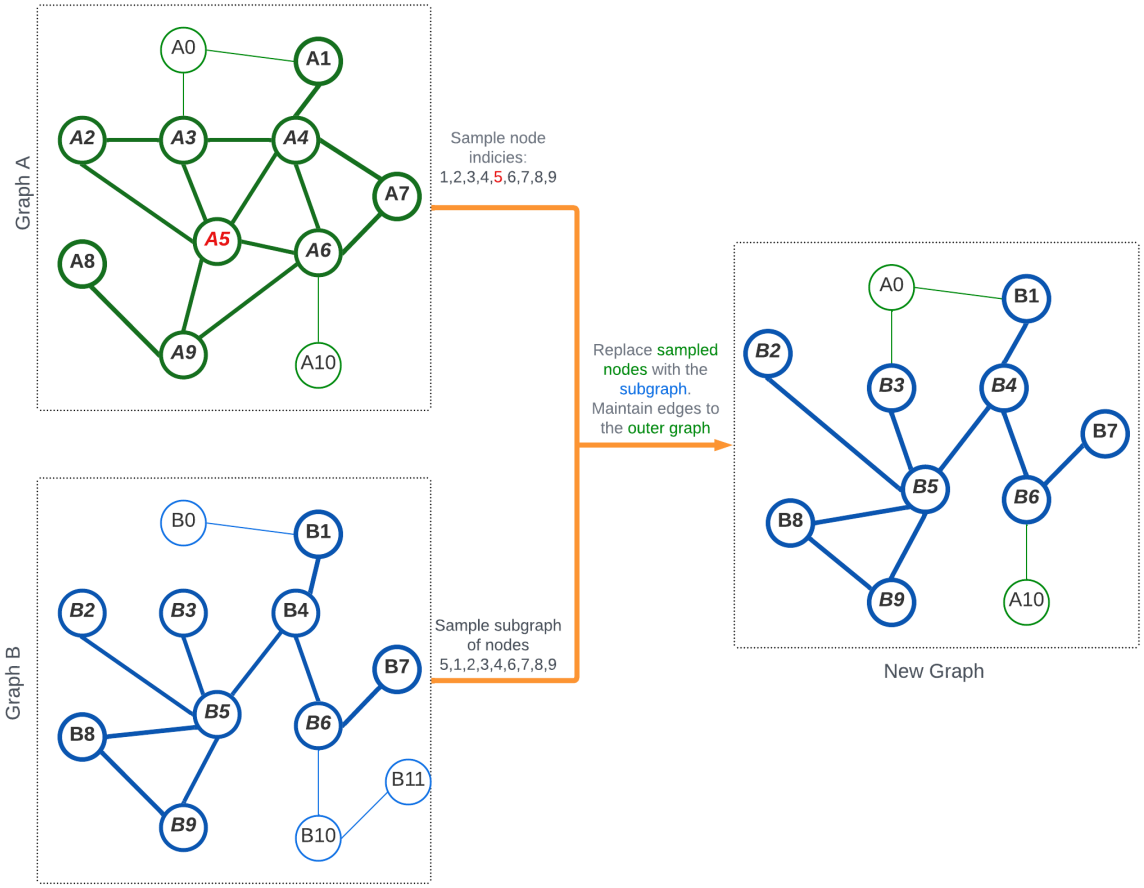


Figure 4.2: Graph RandSquare: Take two input graphs A and B. Sample indices of nodes lying within a square centred around the position of the node $A5$ and remove them from the graph. Sample nodes with the same indices from B and extract the induced subgraph. Finally, insert the extracted subgraph into A. Adopt the subgraph's edges, but maintain the links in A that connect the subgraph to the rest of A.

index in $\mathcal{I}$ is then discarded from $\mathcal{G}_A$ along with all edges between them. The edges connecting a node in $\mathcal{I}$ with a node not in $\mathcal{I}$ are maintained. Next, nodes with indices in $\mathcal{I}$ are sampled from $\mathcal{V}_B$. The induced subgraph is inserted into $\mathcal{G}_A$. $\mathcal{G}_A$ inherits all edges in $\mathcal{E}_B$ within the induced subgraph but maintains the old connections in $\mathcal{E}_A$ between the inner subgraph and the outer graph. Fig. 4.2 illustrates the procedure. Note, that the size of the example graphs in the Figure was reduced for visualisation purposes. The superpixel graphs have a much higher quantity of nodes, thus the data augmentation would be less severe.

4.4.3 Graph Vertical Concat

Graph *VerticalConcat* was inspired by the non-linear blending method with the same name, "Vertical Concat" [10], which is further described in Sec. 2.2. However, the label of $\mathcal{G}_A$ is preserved and not merged with the label of $\mathcal{G}_B$.

Let $|\mathcal{V}_A|$ be defined as the number of nodes in $\mathcal{G}_A$ and $|\mathcal{V}_B|$ of $\mathcal{G}_B$ respectively. First, $\mathcal{V}_A$ and $\mathcal{V}_B$ are ordered in ascending order based on their indices. Fig. 4.3 shows that nodes with high indices correspond to superpixels that are positioned lower in the image. Thus, we can easily define a vertical cut in the graph. With parameter $\alpha \in (0,1)$ we sample the first $\alpha * |\mathcal{V}_A|$ nodes of $\mathcal{G}_A$ and the last $\alpha * |\mathcal{V}_B|$ nodes of $\mathcal{G}_B$. Finally, we construct the two, by the node sets, induced subgraphs and randomly add edges between both to build one connected graph, as illustrated in Fig. 4.4.

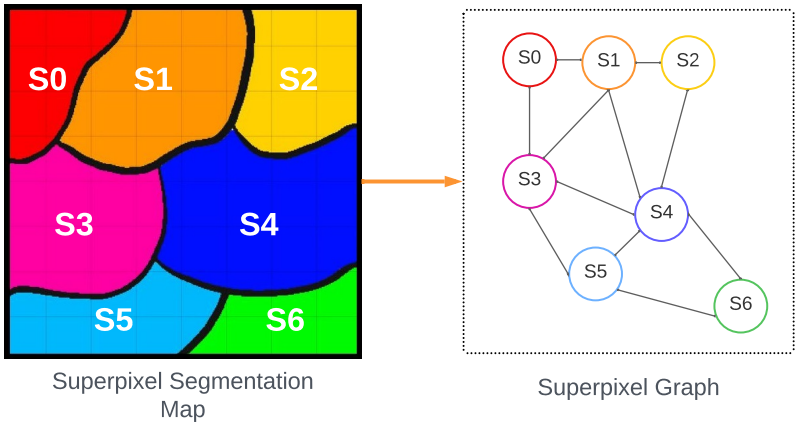


Figure 4.3: The superpixel labels are ordered based on their position in the image, which is mirrored in the node indices in the superpixel graph.

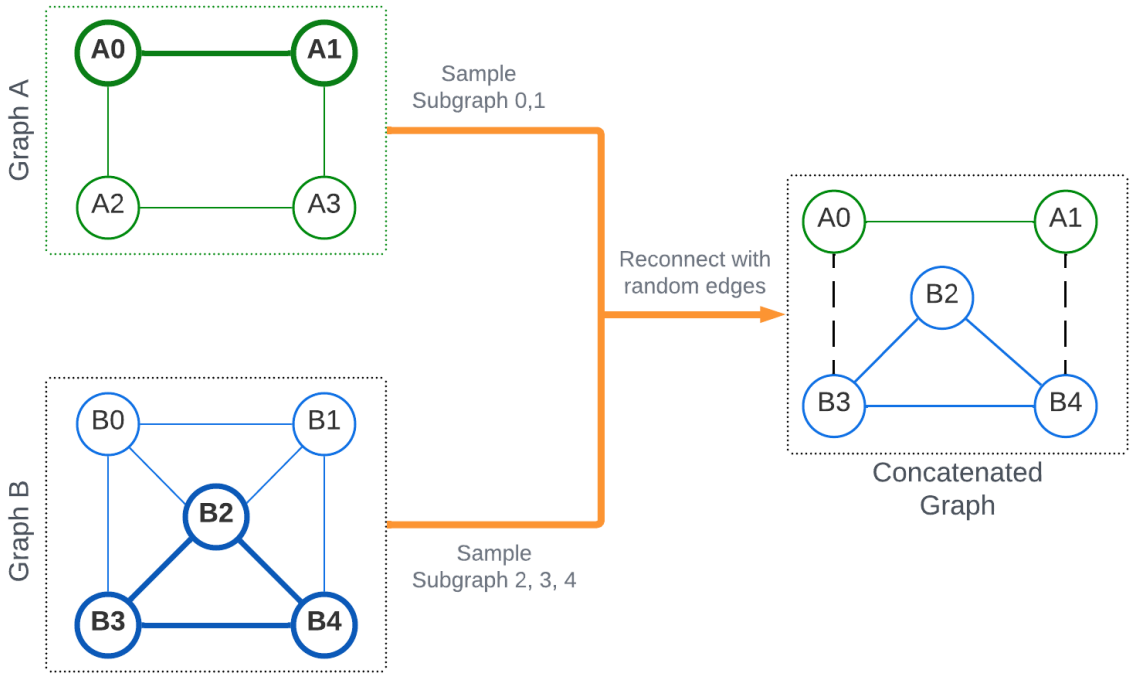


Figure 4.4: Graph VerticalConcat: Let $\alpha = 0.5$. Graph A contains $|\mathcal{V}_A| = 4$ nodes. Thus, we sample a subgraph with $0.5 * 4 = 2$ nodes ($A0$ and $A1$). Graph B contains $|\mathcal{V}_B| = 5$ nodes. Vertical Concat extracts a subgraph induced by the last $5 - 2 = 3$ nodes. Finally, both subgraphs are concatenated by randomly adding new edges $A0 \leftrightarrow B3$ and $A1 \leftrightarrow B4$.

4.4.4 ReOrga

The graph $\mathcal{G}_A$ is segmented into clusters with the Markov Clustering (MC) algorithm. MC makes the assumption that natural clusters contain many edges between the members, and the edges within the clusters generally have higher weights. In $\mathcal{G}_A$, the edge weights represent the similarity of the superpixel nodes. Based on this assumption, it samples clusters by simulating flow in the graph [40].

The resulting clusters are then transformed into disjoint subgraphs and re-organised by reconnecting them with random edges. For each new edge, we compute the edge weight with the cosine similarity (Eq. 4.5).

Fig. 4.5 shows how a sample graph is split into three subgraphs, which are then re-connected by random edges between all clusters. Consequently, the graph is reordered.

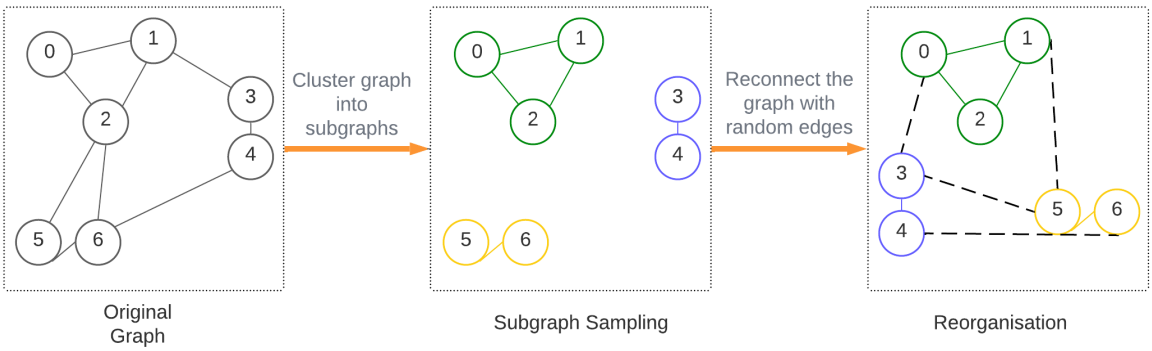


Figure 4.5: ReOrga Graph DA: A graph is divided into three disjoint subgraphs. New edges are randomly inserted between the subgraphs to re-organise the graph in a new way.

4.5 Comparison of Image and Graph Augmentations

With the exception of *EdgeDrop*, *EdgeAdd*, and *ReOrga*, all of the GraphDAs presented in 4.4 have an image augmentation counterpart as depicted in Tab. 4.1. *AttributeMasking* and Random Node Dropping (*RandNodeDrop*) correspond to local erasure and pixel discarding, respectively, while subgraph is correlated to cropping. Furthermore, Graph *RandSquare* and *VerticalConcat* have been inspired by the non-linear blending methods with the same names.

For better readability, we refer to the respective graph augmentations with *RandSquare* and *VerticalConcat* in further sections unless stated otherwise.

Table 4.1: Graph augmentations and their respective image augmentation counterpart.

Image DA	Graph DA
Local Erasure/ Cut Out	AttributeMasking
Random Crop	Subgraph
Pixel Discarding	RandNodeDrop
Random Square	Graph RandSquare
Vertical Concat	Graph VerticalConcat
-	ReOrga
-	EdgeDrop
-	EdgeAdd

5 | Experimental Setup

The models and augmentations are implemented with PyTorch Lightning and PyTorch Geometric. A ResNet18 [41] and 6-layer GCN with a max pooling layer between the convolutional layers and the readout function represent the baselines. All models, with and without augmentations, are trained for 100 epochs and then tested on an unseen test set. The input images for ResNet18 are normalised and scaled to the same size. The input graphs are generated from normalised images using RAG and SLIC superpixel algorithm with $K = 2000$ and $C = 10$ (based on the experimental results in Sec. 6.1). All models use stochastic gradient descent optimiser [42] with a learning rate of 0.1, a momentum of 0.9, a weight decay of 0.00005 and the Multi-Step learning rate scheduler. No dropout is applied. The batch size is 128, regardless of the model and dataset. We use binary cross entropy loss with logits for MLC.

6 | Experimental Results

In this chapter we present and evaluate the results of our experiments on RS MLC datasets. First, the superpixel algorithms are compared and analysed to fix the hyper-parameters for our graph generation pipeline. We then continue with a short initial comparison of the two baseline models. Afterwards, the results of the GraphDA techniques are presented in detail and finally evaluated across all four datasets.

6.1 Superpixel Segmentation

We tested different hyperparameter configurations for all four algorithms by randomly sampling images from the DeepGlobe training split and computing BR (Eq. 4.3) and UE (Eq. 4.4) according to their corresponding ground truth segmentation map.

The results for the best configuration are shown in Table 6.1. Hereby, K is defined as the number of superpixels and Itr as the number of iterations. For LSC and Quickshift the number of superpixels greatly varies across the samples, so no K is provided. While all algorithms achieve a relatively good UE, SLIC outperforms the others by a wide margin with a configuration of $C = 10$ and $K = 3000$, with C being the compactness. Thus, we chose SLIC for our approach.

For further illustration Fig. 6.1 shows some hyperparameter configurations for SLIC on an example from DeepGlobe. A higher number of superpixels K results in a finer segmentation, while the compactness parameter C defines how irregular or compact the

Table 6.1: Best configurations found for different superpixel algorithms on DeepGlobe.

Algorithm	UE	K	Parameters
SLIC	0.000026	3000	C=10
SEEDS	0.023061	2500	Itr=250
LSC	0.043466	-	$r = 5$, Itr=250
Quickshift	0.064637	-	$\tau = 5$

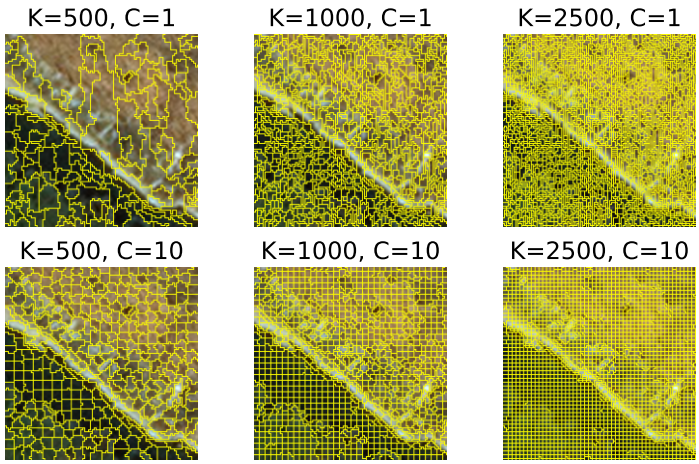


Figure 6.1: Different hyper-parameter configurations for the SLIC superpixel segmentation on an example from the DeepGlobe dataset.

resulting superpixels are. But, more superpixels also result in a bigger graph and higher computational complexity. Hence we need to find an adequate trade-off.

Fig. 6.2 shows the boundary recall and UE for increasing K with different compactness C. Both, boundary recall and UE rapidly improve while K is still relatively small, but after at around $K = 2000$, we stop observing a noticeable improvement. Thus, we define $K = 2000$ and $C = 10$.

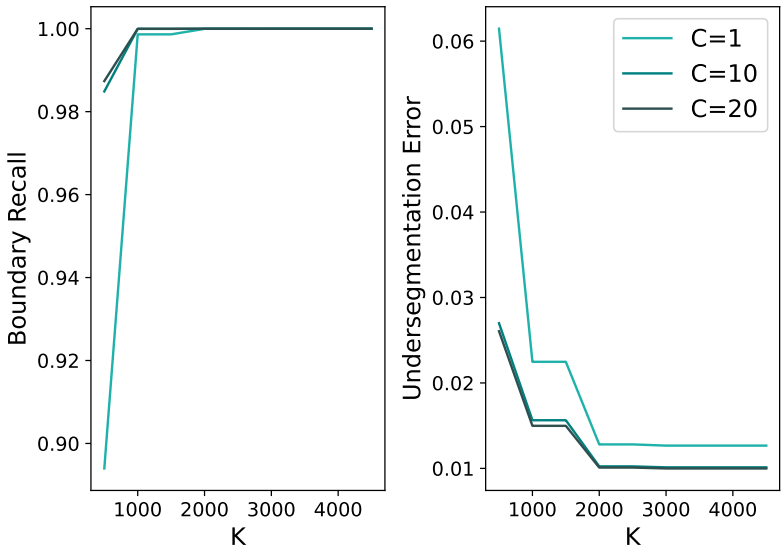


Figure 6.2: Boundary recall and undersegmentation error of SLIC with different K and C averaged across a subset of DeepGlobe training samples.

6.2 Baseline Results of GCN

Fig. 6.3 shows the test sets' average macro precision (APmac) for the ResNet18 and GCN baselines without augmentations. ResNet18 outperforms GCN for all datasets with the most significant margin for UCMerced.

Notably, GCN shows a significant performance improvement based on the dataset's complexity. Whereas ResNet18 shows a slight improvement from Globe15 to Globe2, the test APmac of GCN increases by 0.1507, minimising the performance divergence between the two baselines. Increasing the mean class occurrence per sample further leads to an APmac drop for both models. However, while ResNet18 shows a reduction of 0.072, GCN's APmac only decreases by 0.0041, drawing both scores closer together. ResNet18 performs more poorly on Globe25 than Globe15, albeit slightly better than GCN. This indicates that GCN benefits from datasets with a more complex label space and samples showing higher class interaction for MLC. Yet, it performs very poorly on UCMerced compared to the ResNet18 baseline, which has even more detailed multi-labels with an average class occurrence of 3.3 but is also highly imbalanced. A possible explanation for this behaviour is that UCMerced is a very high-resolution dataset and contains many structural objects, which ResNet18 can significantly benefit from. At the same time, GCN suffers a more severe information loss due to the superpixel segmentation [17].

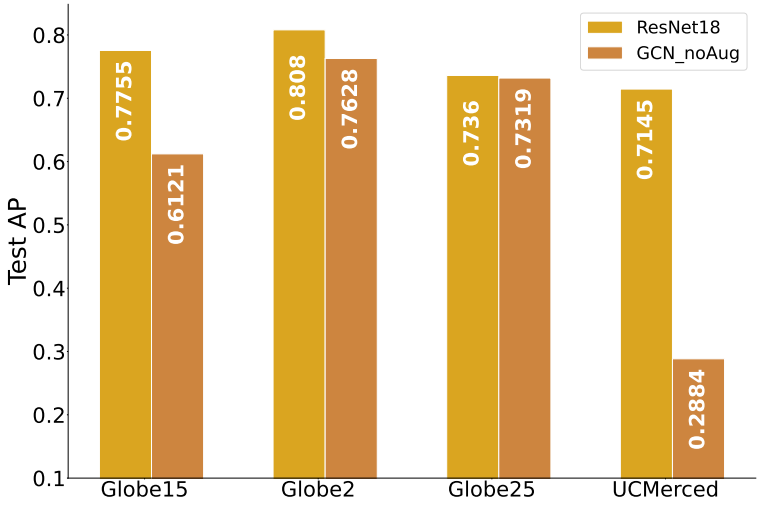


Figure 6.3: Macro average precision for ResNet18 and GCN on Globe15, Globe2, Globe25 and UCMerced test sets.

6.3 Graph Augmentations

In this section, we will separately present and analyse the results of the different GraphDAs for different p. The following charts show the APmac scores for the GraphDAs, proposed in Sec. 4.4) on the test sets.

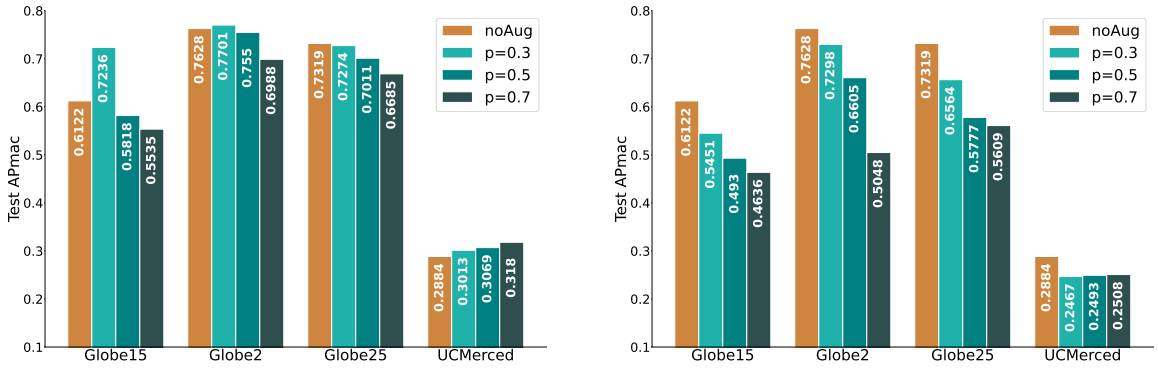
EdgeDrop

EdgeDrop randomly discards edges from a graph. A slight augmentation with a probability $p = 0.3$ shows a significant performance increase for Globe15, as depicted in Fig. 6.4 (a). However, further increasing p leads to a sudden AP drop and greatly hurts the model's performance. Globe2 also profits from a slight *EdgeDrop* augmentation, but the performance suffers for higher p.

EdgeDrop on UCMerced shows the opposite behaviour. All three configurations improve the results, but the strongest augmentation with $p = 0.7$ leads to the most significant improvement.

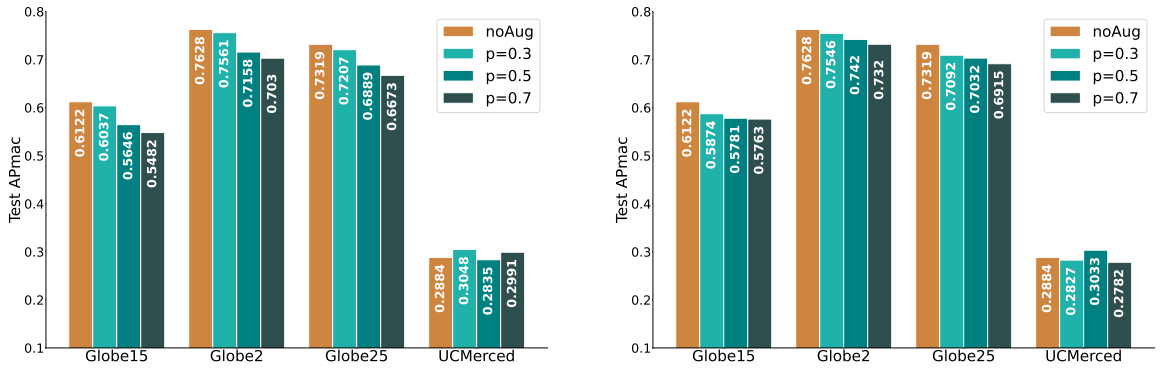
EdgeAdd

EdgeAdd alters a graph's structure by randomly adding edges between nodes. Fig. 6.4 (b) shows the test APmac for all datasets. Noticeably, *EdgeAdd* decreases GCN performance on all datasets and drastically decreases the AP scores for higher p.



(a) EdgeDrop **(b)** EdgeAdd

Figure 6.4: APmac on the test sets for Globe15, Globe2, Globe25, and UCMerced for the GCN baseline (noAug) and GCN with *EdgeDrop* **(a)** and *EdgeAdd* **(b)** for different p.



(a) RandNodeDrop **(b)** Subgraph

Figure 6.5: APmac on the test sets for Globe15, Globe2, Globe25, and UCMerced for the GCN baseline (noAug) and GCN with *RandNodeDrop* **(a)** and *Subgraph* **(b)** for different p.

RandNodeDrop

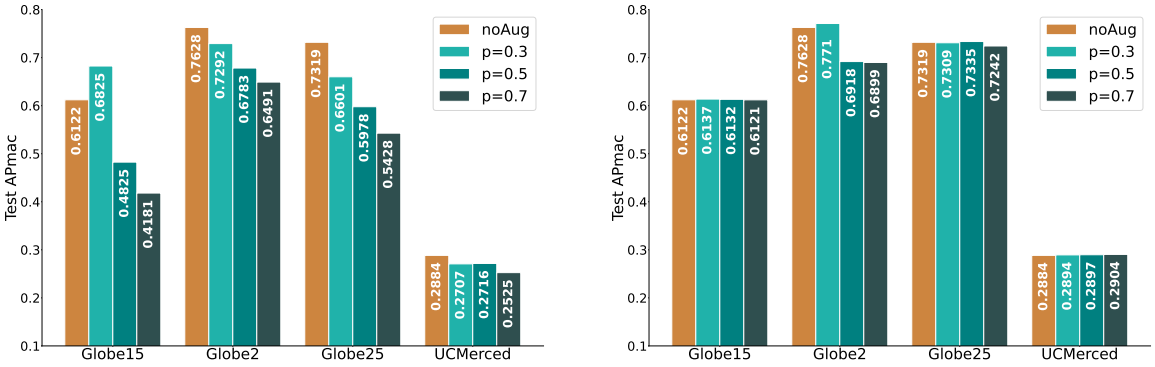
Similar to *EdgeAdd*, *RandNodeDrop* leads to performance degradation for all p on all datasets except UCMerced as depicted in Fig. 6.5 (a), albeit not as drastic as *EdgeAdd*. While it also decreases the APmac score for $p = 0.5$ on UCMerced, *RandNodeDrop* slightly increases APmac for $p = 0.3$ and $p = 0.7$.

Subgraph

Subgraph is equivalent to random cropping in the image domain. But, while RS images profit from random cropping, *Subgraph* tends to decrease the performance on GCN, except for UCMerced and $p = 0.5$ as depicted in Fig. 6.5 (b). However, all deviations from the GCN baseline are minor.

AttributeMasking

Fig. 6.6 (a) shows the results on all four datasets for *AttributeMasking*. Interestingly, Globe15 drastically benefits from *AttributeMasking* with a low $p = 0.3$, while all other configurations significantly hurt performance on all datasets, which is in line with the results of You et al. [1] on superpixel graphs. *AttributeMasking* randomly drops a column from the node feature matrix X. For the superpixel graphs, this is equivalent to masking an entire RGB channel.



(a) AttributeMasking **(b)** ReOrga

Figure 6.6: APmac on the test sets for Globe15, Globe2, Globe25, and UCMerced for the GCN baseline (noAug) and GCN with *AttributeMasking* **(a)** and ReOrga **(b)** for different p.

ReOrga

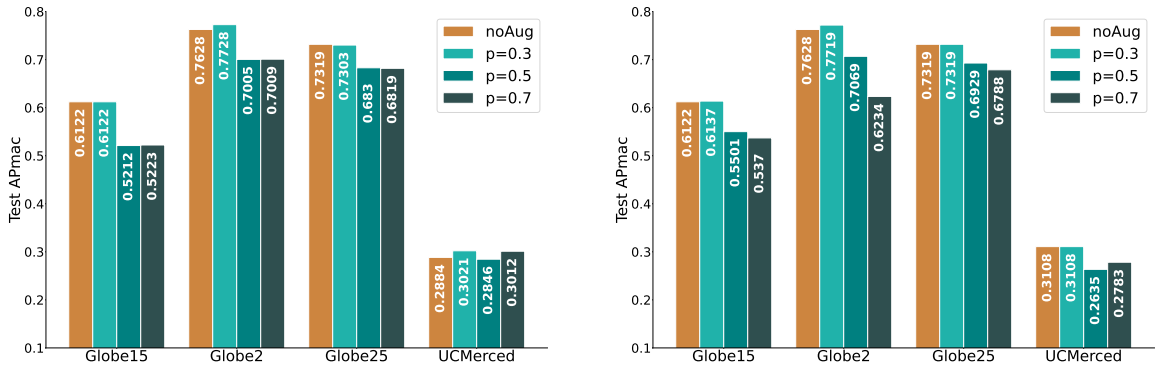
ReOrga splits a graph into clusters and randomly reconnects them. Fig. 6.6 (b) shows the results. *ReOrga* slightly improved the APmac for all datasets and all p. The improvement, however, is barely significant. Despite rearranging the entire graph, the model was not negatively affected, which strongly suggests that the local information is more important than the global graph structure.

Graph RandSquare

RandSquare slightly improves Globe2's APmac for $p = 0.3$ while it hurts the performance for all other Globe datasets and p configurations, as illustrated in Fig. 6.7 (a). Interestingly, however, it leads to a slight performance improvement for $p = 0.3$ and $p = 0.7$ on UCMerced.

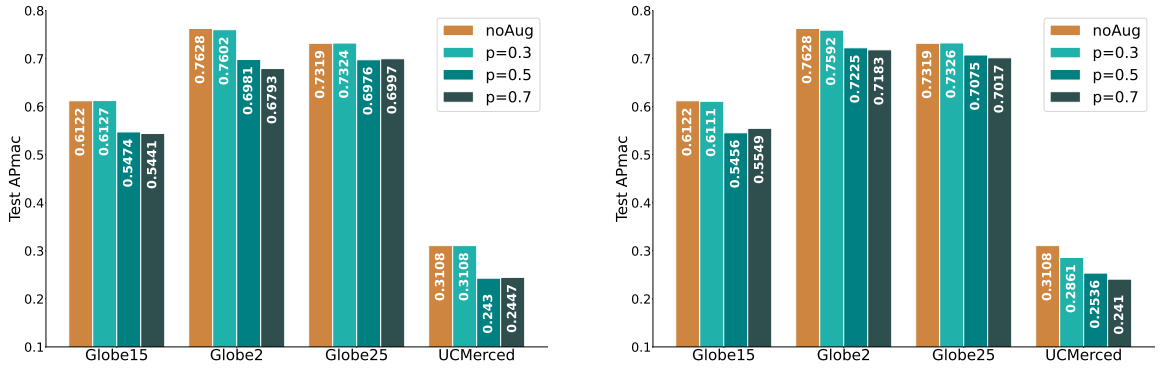
Graph VerticalConcat

VerticalConcat concatenates two graphs inspired by the vertical concatenation of images. The parameter α defines which portion of nodes from the original sample graph is kept. Fig. 6.7 (b) shows the results for different p and $\alpha = 0.3$, wheras Fig. 6.8 shows the results for $\alpha = 0.5$ (a) and $\alpha = 0.7$ (b). Overall, any value for α with a small $p = 0.3$ seems to slightly increase the AP scores across all datasets (except for $\alpha = 0.7$ on UCMerced) or at least do not lead to a considerable drop. The changes, however, are so small that they are barely significant. On the other hand, increasing p can drastically decrease APmac.



(a) RandSquare **(b)** VerticalConcat ($\alpha = 0.3$)

Figure 6.7: APmac on the test sets for Globe15, Globe2, Globe25, and UCMerced for the GCN baseline (noAug) and GCN with *RandSquare* and *VerticalConcat* with $\alpha = 0.3$ for different p.



(a) VerticalConcat ($\alpha = 0.5$) **(b)** VerticalConcat ($\alpha = 0.7$)

Figure 6.8: APmac on the test sets for Globe15, Globe2, Globe25, and UCMerced for the GCN baseline (noAug) and GCN with *VerticalConcat* with $\alpha = 0.5$ and $\alpha = 0.7$ for different p.

A smaller α means less of the original graph is used in the concatenation. Additionally, the label remains unchanged; thus, discarding a large portion of the original graph and replacing it can potentially cause the label to be less accurate, which means that label noise is introduced to the dataset. However, Tab. 4.4 results indicate that the model can benefit from slight label noise ($p = 0.3$).

6.4 Overall GraphDA Performance

The tables in this section show the GCN and ResNet18 baseline results on the datasets and the results of all augmentations that improved GCN performance for MLC. The complete results are found in Appendix A (Tab. A.1, A.2, A.3, A.4).

Table 6.2: Test results for baseline ResNet18, baseline GCN (GCN_noAug), and every augmentation that improved GCN's performance on Globe15. The method identifiers result from the augmentation name and the configuration of p, as well as α if given.

Method	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	0.3	0.7755	0.8622	0.4013	0.754
EdgeDrop0.3	-	0.3	0.7236	0.8031	0.309	0.676
AttributeMasking0.3	-	0.3	0.6825	0.771	0.2747	0.6529
VerticalConcat0.3(α =0.3)	0.3	0.3	0.6137	0.7817	0.4709	0.6804
ReOrga0.3	-	0.3	0.6137	0.7801	0.4707	0.6751
ReOrga0.5	-	0.5	0.6132	0.782	0.4723	0.6788
VerticalConcat0.3(α =0.5)	0.5	0.3	0.6127	0.7814	0.4691	0.6802
RandSquare0.3	-	0.3	0.6122	0.7806	0.4709	0.6806
GCN_noAug	-	-	0.6122	0.7814	0.4699	0.6802

The best augmentations on Globe15 are shown in Table 6.2. As noted previously, ResNet18 outperforms the GCN baseline by a large margin—however, GCN profits from light data augmentation with $p = 0.3$. While most augmentations in the table do not have much effect, *AttributeMasking* and *EdgeDrop* with $p = 0.3$ significantly improve APmac by 0.07 and 0.11, reducing the gap to the ResNet18 baseline. But, increasing p can considerably hurt model performance (Tab. A.2). For example, *AttributeMasking* and *EdgeAdd* with $p = 0.7$ can drop the APmac to 0.4181 and 0.4636, respectively.

Similar to Globe15, Globe2 also profits from less severe data augmentation with $p = 0.3$, as depicted in Tab. 6.3. The difference between the APmac scores between both baselines is reduced, and fewer augmentations lead to a performance boost. Furthermore, the influence of even the best augmentations is less significant. For example, the top augmentation *RandSquare* with $p = 0.3$ only results in an APmac increase of 0.0461. Strong augmentations also show to be harmful in Tab. A.2 with *EdgeAdd* and $p = 0.7$

Table 6.3: Test results for baseline ResNet18, baseline GCN (GCN_noAug), and every augmentation that improved GCN's performance on Globe2. The method identifiers result from the augmentation name and the configuration of p, as well as α if given.

Method	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.8089	0.8671	0.4675	0.7794
RandSquare0.3	-	0.3	0.7728	0.8473	0.451	0.7685
VerticalConcat0.3(α =0.3)	0.3	0.3	0.7719	0.8461	0.4445	0.7612
ReOrga0.3	-	0.3	0.771	0.8436	0.4423	0.7621
EdgeDrop0.3	-	0.3	0.7701	0.838	0.4388	0.7547
GCN_noAug	-	-	0.7628	0.8443	0.447	0.7636

Table 6.4: Test results for baseline ResNet18, baseline GCN (GCN_noAug), and every augmentation that improved GCN's performance on Globe25. The method identifiers result from the augmentation name and the configuration of p, as well as α if given.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	0.3	0.736	0.848	0.5138	0.7557
ReOrga0.5	-	0.5	0.7335	0.8538	0.6231	0.7646
VerticalConcat0.3(α =0.7)	0.7	0.3	0.7326	0.8554	0.6286	0.7643
VerticalConcat0.3(α =0.5)	0.5	0.3	0.7324	0.8529	0.6179	0.7569
VerticalConcat0.3(α =0.3)	0.3	0.3	0.7319	0.8555	0.6259	0.7638
GCN_noAug	-	-	0.7319	0.8551	0.6314	0.7659

dropping the APmac to 0.5048.

In contrast to Globe15 and Globe2, the two baselines perform very similarly for Globe25; ResNet's APmac is only better by 0.0041, which is barely noticeable, as shown in Tab. 6.4. However, augmentation only leads to some improvement. *ReOrga* and *VerticalConcat* for all α with $p = 0.3$ cause a slight increase of APmac and bring the GCN's performance up to the ResNet18 baseline. Again, strong *EdgeAdd* and *AttributeMasking* hurt the model the most (Tab. 6.4).

Table 6.5: Test results for baseline ResNet18, baseline GCN (GCN_noAug), and every augmentation that improved GCN's performance on UCMerced. The method identifiers result from the augmentation name and the configuration of p, as well as α if given.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.7145	0.7639	0.5456	0.6923
EdgeDrop0.7	-	0.7	0.3285	0.5308	0.0728	0.224
EdgeDrop0.7_2layer	-	0.7	0.318	0.5581	0.0715	0.2931
EdgeDrop0.5	-	0.5	0.3069	0.553	0.0813	0.3101
RandNodeDrop0.3	-	0.3	0.3048	0.5472	0.0839	0.3549
Subgraph0.5	-	0.5	0.3033	0.5397	0.1119	0.3917
RandSquare0.3	-	0.3	0.3021	0.5475	0.1015	0.3936
VerticalConcat0.3(α =0.7)	0.7	0.3	0.3021	0.5475	0.1015	0.3936
EdgeDrop0.3	-	0.3	0.3013	0.5498	0.0839	0.3479
RandSquare0.7	-	0.7	0.3012	0.5467	0.0986	0.3843
RandNodeDrop0.7	-	0.7	0.2991	0.5328	0.0838	0.2895
ReOrga0.7	-	0.7	0.2904	0.5475	0.093	0.3761
ReOrga0.5	-	0.5	0.2897	0.5472	0.0923	0.3752
ReOrga0.3	-	0.3	0.2894	0.5468	0.0923	0.3748
VerticalConcat0.3(α =0.3)	0.3	0.3	0.2884	0.5463	0.0926	0.3759
GCN_noAug	-	-	0.2884	0.5463	0.0926	0.3759

Various augmentations positively affect UCMerced, such as *EdgeDrop*, *ReOrga*, *RandNodeDrop*, *Subgraph*, and *VerticalConcat*, illustrated in Tab. 6.5. Mainly *EdgeDrop* works well for all p. Interestingly, unlike the Globe datasets UCMerced also profits from stronger augmentations. On the other hand, strong *VerticalConcat*, *EdgeAdd*, and *AttributeMasking* severely hurt GCN performance.

In conclusion, all Globe datasets generally profit from light augmentations. Since *VerticalConcat* and *RandSquare* do not alter the labels, there is a risk of introducing label noise to the sample. Interestingly, Globe profits from a bit of label noise. UCMerced, on the other hand, prefers strong augmentations. On all datasets, *EdgeDrop*, *ReOrga*, and *VerticalConcat* perform well depending on the hyperparameter configuration, while *EdgeAdd* and *AttributeMasking* can severely impair GCN performance. Nevertheless, further analysis is needed to fully understand the different effects of GraphDA across the datasets.

7 | Discussion

The previously presented results have revealed that the performance of GCN and GraphDA varies significantly between the Globe datasets and UCMerced. Therefore, this chapter will delve deeper into the disparities between them.

7.1 Texture and Object-based Classes

To begin with, the Globe datasets are bigger, have fewer classes, and are more balanced than UCMerced. But, the types of classes greatly differ as well. While DeepGlobe's classes are primarily based on texture, UCMerced includes mixed and object-based classes. The texture-based classes consist of *field*, *pavement*, *bare soil*, *airplanes*, *water*, *trees*, *sand*, *sea*, and *chaparral*. *Docks* and *buildings* are mixed classes as they can either appear as single objects or with a pattern. *Cars*, *mobile-home*, *airplane*, *tank*, *court*, and *ship* are categorised as object-based.

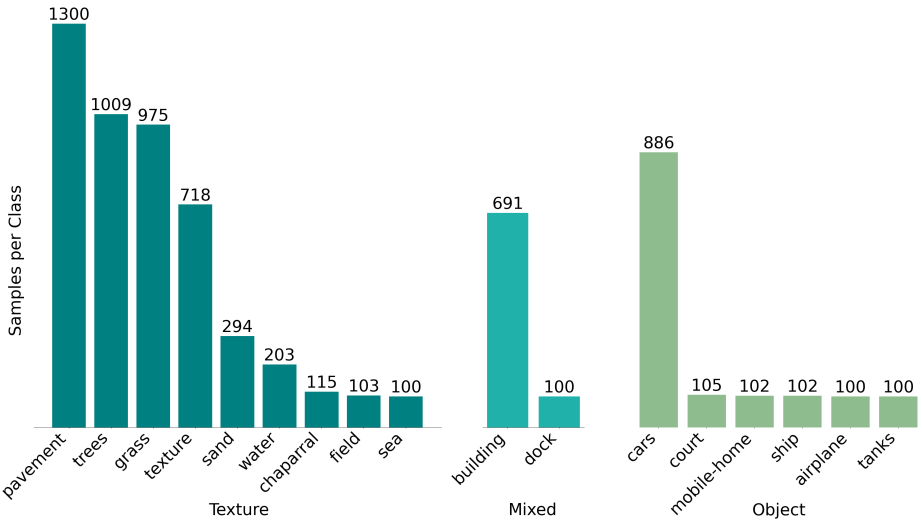


Figure 7.1: Categorisation of classes into mixed, object-, or texture-based.

The categories and the number of samples per class are illustrated in Fig. 7.1. Furthermore, Fig. 7.1 visualises how all object classes, except for *cars*, are minority classes and that the dataset is heavily imbalanced favouring texture-based classes. Additionally, since objects tend to be small on satellite images, only a few pixels can be assigned to them. Those few pixels are then further condensed to superpixels. Unfortunately, this can lead to a loss of information [17], particularly for object-based classes.

Figure 7.2 shows the mean test APs for both baselines averaged across all classes for all three categories. ResNet18 outperforms GCN in all. Most notably, while ResNet18 has a lower AP for object classes as well, the AP of GCN drops as low as 0.1326. However, strong variations within the categories still exist, so we need to examine the various classes closely.

The AP test scores for every object class for all augmentations and baselines are visualised in Fig. 7.4. For comparison reasons, we picked the best augmentation configuration for each class. The GCN baseline performs poorly on all object classes except for *cars*. Yet, the AP is much lower than ResNet18's. As stated, *cars* is the only majority class in the object category, which most likely plays a big part in the rather good performance.

Additionally, *cars* greatly profits from GraphDA, especially *EdgeDrop* with a high probability $p = 0.7$ and using only a 2-layer GCN instead of a 6-layer GCN. The most likely explanation for this is that our network is over-smoothing, a phenomenon that appears when repeated graph convolution during layer propagation leads to a point where all nodes' features converge to the same representation. Consequently, the learned

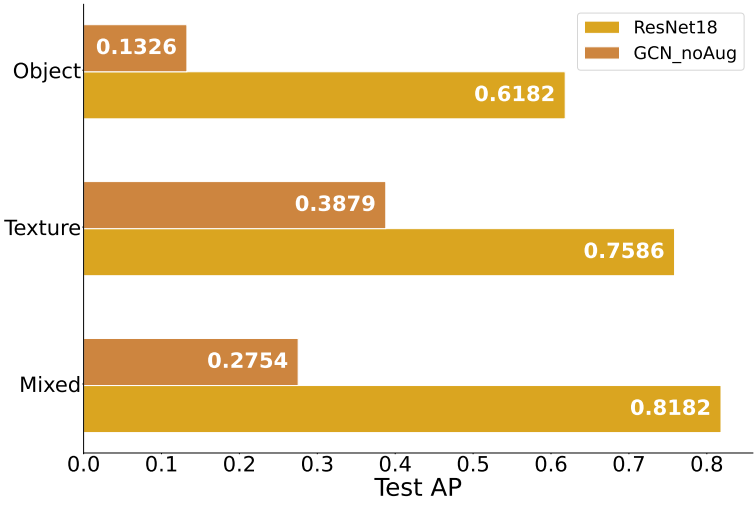


Figure 7.2: Mean AP Results for GCN and ResNet18 baseline on the three class categories, texture, mixed and object.

representations are unrelated to the input features, and gradients are vanishing. This is more likely to happen in deeper GCN architectures [12], [43]–[45]. Both heavy *EdgeDrop*, which reduces the graph's connectivity, and fewer layers reduce the size of the GCN's receptive field and thus alleviate the risk of over-smoothing [44]. Thus, the object's information, represented only by a few nodes, is not as easily lost during aggregation. In general, *EdgeDrop* has a positive effect on all object classes. Strong *EdgeDrop* shows the biggest improvement for four of six classes, showing that randomly reducing graph connectivity can benefit object classification.

GCN achieves its second-best AP for *tanks* out of the object-based classes. Interestingly, ResNet18 also has a comparably low score for this class, and GCN with GraphDA approaches ResNet18's AP. *Tanks* is with 1.4% (Fig. 3.3) a minority class. But, a closer look at the samples in Fig. 7.3 reveals that *tanks* often either cover a larger portion of the image or appear along with others. Thus, even though *tanks* is a minority class, many nodes can represent a single tank in a graph, i.e. more information is preserved, and a GCN can learn their relationship. All GraphDAs positively impact *tanks*, except *ReOrga*, which barely altered the AP. Interestingly, *EdgeAdd* performs better than *EdgeDrop*, and the model profits from strong *VerticalConcat*, i.e. a lot of label noise.

The results for the classes *mobile-home*, *ship*, *airplanes*, and *court* show that GCN completely fails to understand and distinguish them. All of them are minority classes. Aeroplanes do not appear in distinct patterns; mobile homes and ships can be easily confused with cars, which are a majority class, and hence can cause a bias in favour of car predictions; Finally, courts are characterised by distinct lines, which may disappear during superpixel sampling.

In a nutshell, many distinct characteristics and much information can get lost during graph generation, hindering GCN performance, especially if the dataset is imbalanced, which is in line with previous findings [46]–[48].

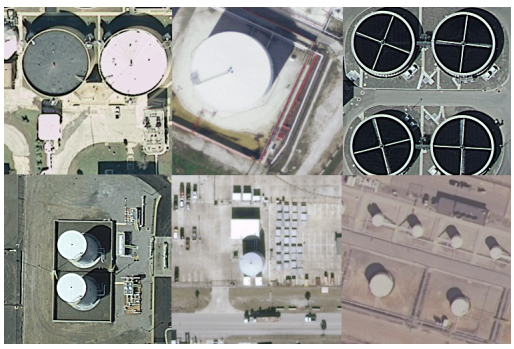


Figure 7.3: Sample images for the *tanks* class of UCMerced.

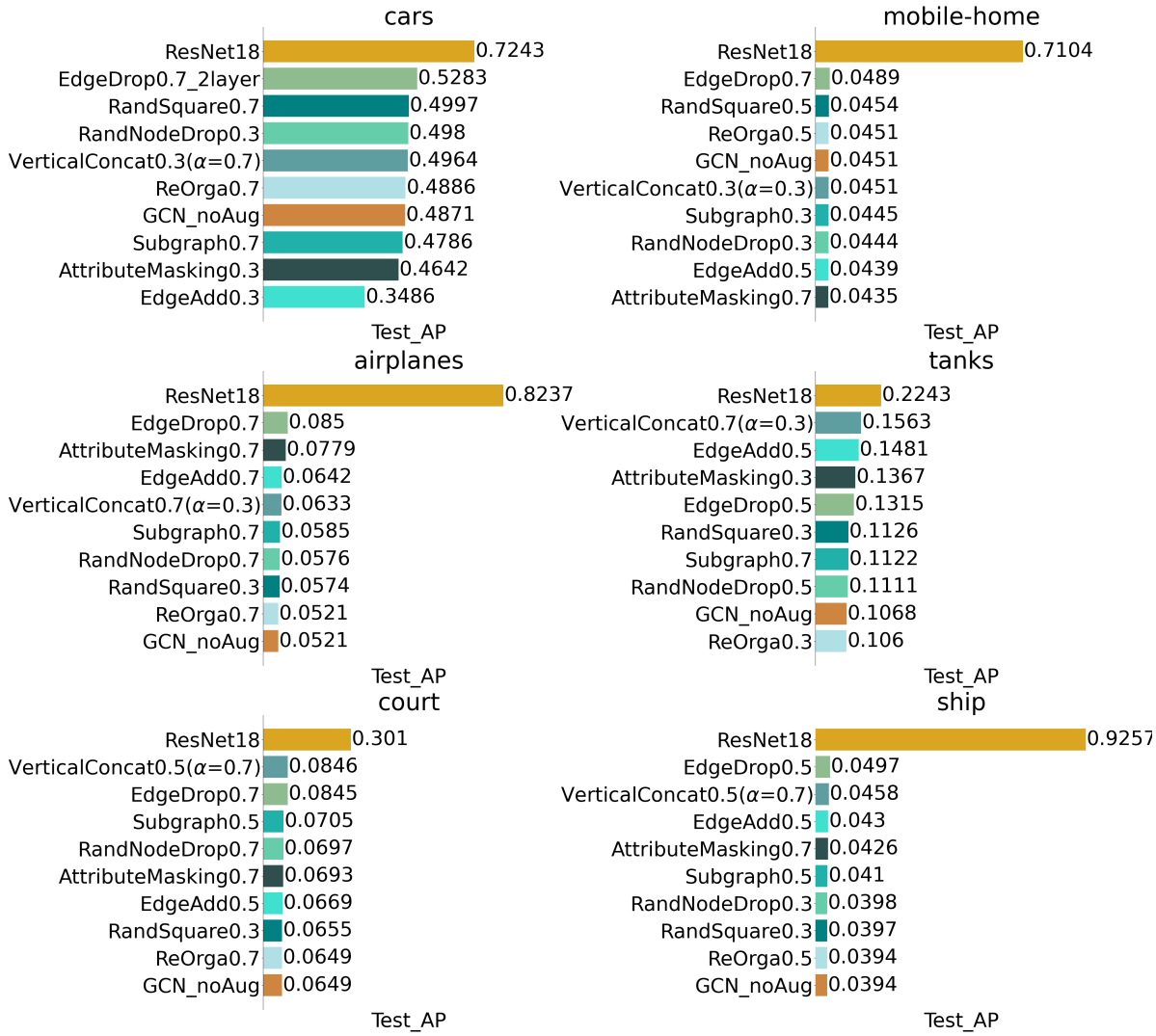


Figure 7.4: AP scores on the object-based classes, *cars*, *mobile-home*, *airplanes*, *tanks*, *court*, and *ship*, for baseline ResNet, GCN and the best configuration for each augmentation.

Mixed classes can either appear as objects or in patterns, such as *buildings* and *docks*. Fig. 7.5 shows the test AP for all augmentations with their best configurations and the baselines for both classes. Notably, GCN performs well for *buildings* but fails for *dock*. *Buildings* is with 9.9% a majority whereas *dock* is a minority class with just 1.4% representation (Fig. 3.3). *RandNodeDrop*, *Subgraph*, *VerticalConcat*, and RandSquare greatly boost performance but are outperformed by 2-layer *EdgeDrop* with $p = 0.7$. Again, indicating that the predictive capabilities of GCN for fine-grained classes can benefit from a smaller receptive field. Additionally, *buildings* profits from the strong relationship between the nodes caused by the patterns, which GCN can recognise.

But, GCNs are difficult to interpret due to the complex underlying information and

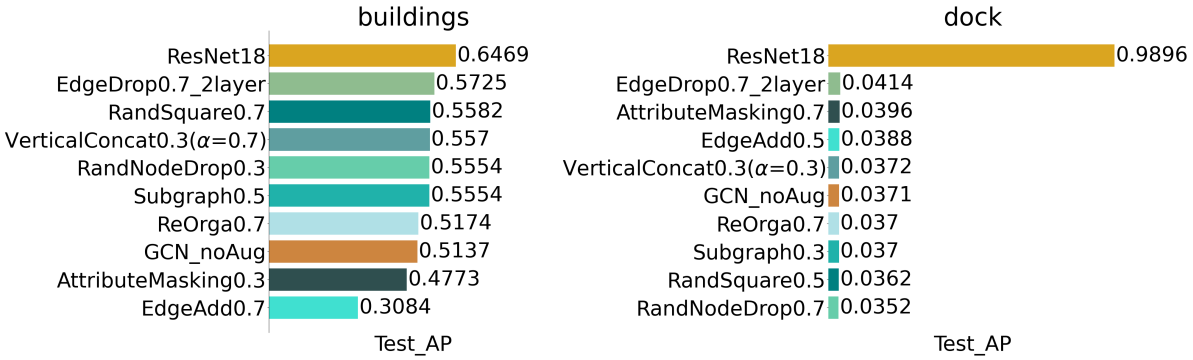


Figure 7.5: AP scores on the mixed classes, *buildings*, and *tanks*, for baseline ResNet, GCN and the best configuration for each augmentation.

their transformations [49], [50]. So, while the small size of the *dock* class can partly explain the poor performance of GCN, it remains unclear why it failed to learn the very distinct patterns of *docks*, which ResNet18 easily distinguishes with an AP of 0.9896. A possible explanation is that *dock* has such a strong correlation with other classes, such as *ship* and *water*, that GCN guesses *dock* every time it recognises one of the correlated classes, resulting in a meagre precision score.

The results for texture-based classes are presented in Fig. 7.6, which shows that GCN performs better on them than on object classes. It is worth noting that *pavement*, *bare-soil*, *trees*, and *grass* are majority classes, which is also reflected in the good performance of the models. Most classes greatly benefit from DA. Again, *EdgeDrop*, as well as *VerticalConcat*, show favourable results. Additionally, randomly removing information from the graph through *AttributeMasking* and *RandNodeDrop* can significantly improve results, e.g. for classes *sand* and *bare-soil*, which suggests that GCN can reconstruct missing pieces from the remaining information encoded in neighbouring nodes and the graph structure for texture-based classes. As textures often cover large portions of the image, DAs such as *Subgraph* and RandSquare can be applied without risking the loss of important information.

Surprisingly, *EdgeAdd*, shown to reduce GCN performance multiple times, can greatly benefit texture-based classes such as *grass* and *sand* by increasing the receptive field to distinguish larger structures and smooth out noise within them.

However, it negatively impacts the *pavement* class, which conversely profits from strong *EdgeDrop* with $p = 0.7$ along with a 2-layer GCN, similar to *cars*. GCN excels at identifying the relationships between instances. *Cars* and *pavement* strongly correlate with each other and often appear together. Thus, both classes might benefit from the same augmentation since detecting one class depends on the other.

Lastly, *ReOrga* has not shown a significant difference from the baseline for all three categories, proving that local structure in a graph is more important than global structure. In other words, a GCN is indifferent to changes in the graph as long as the local structure is preserved.

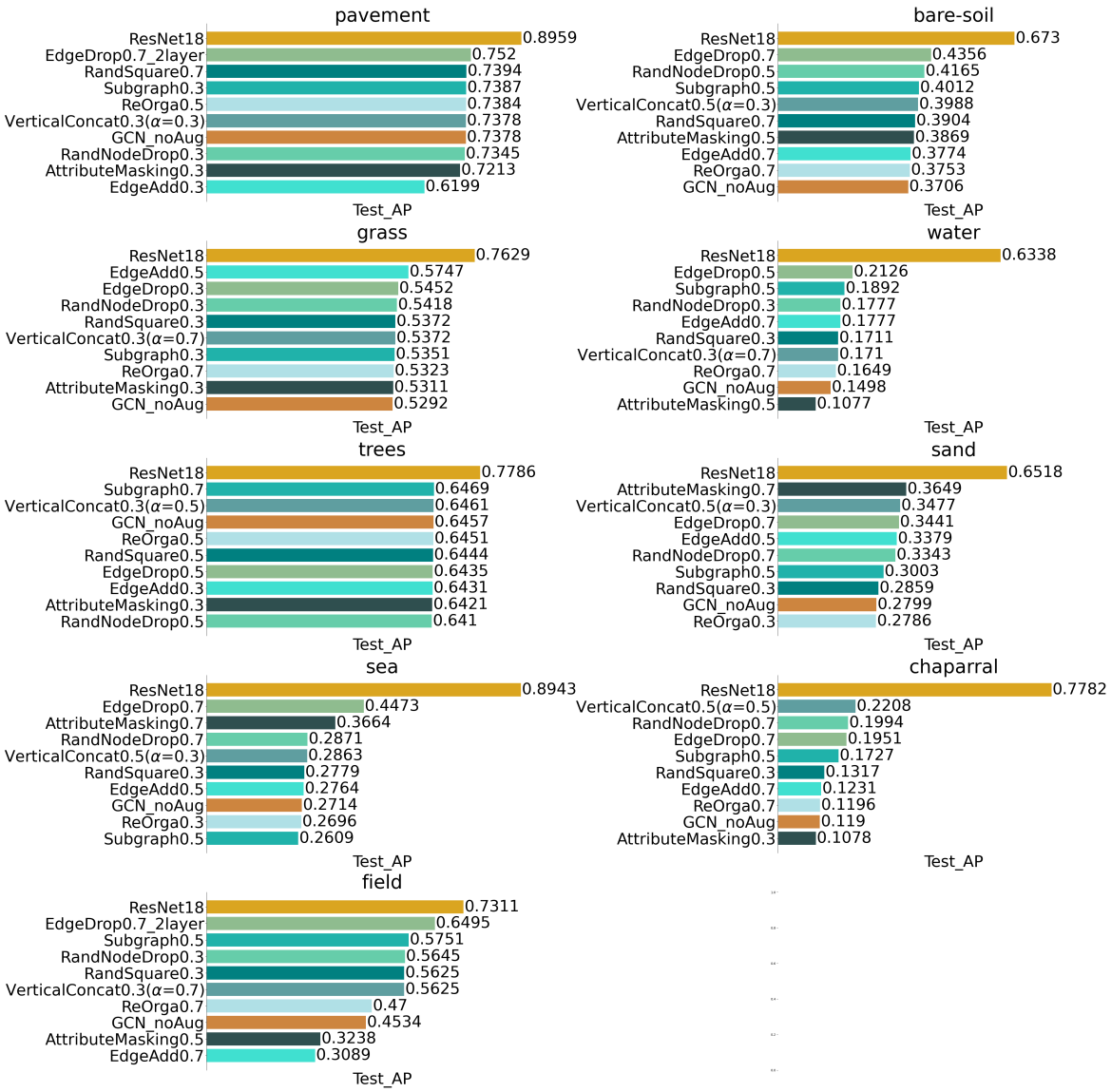


Figure 7.6: AP scores on the texture-based classes, *field*, *pavement*, *bare-soil*, *grass*, *water*, *trees*, *sand*, *sea*, and *chaparral*, for baseline ResNet, GCN and the best configuration for each augmentation.

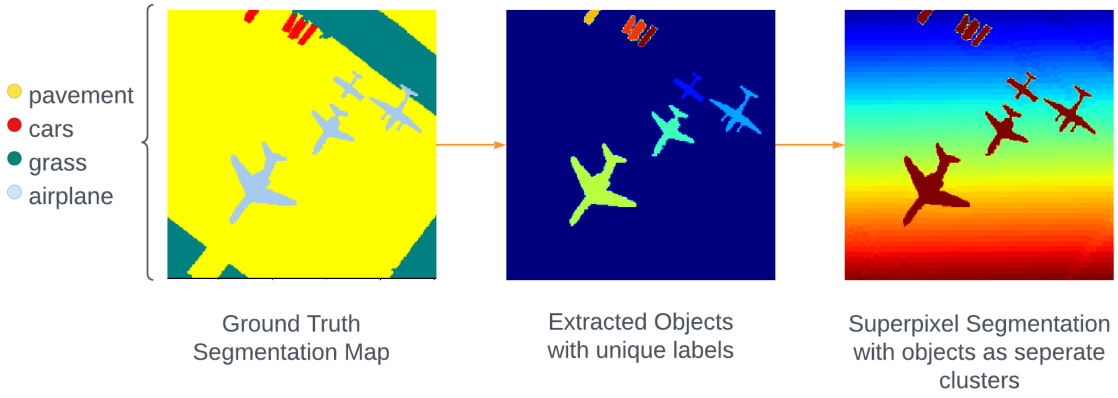


Figure 7.7: Left: Example ground truth segmenation map with classes *pavement*, *cars*, *airplane*, *grass*. **Middle:** Objects are extracted from the segmentation map and assigned a unique label. Finally, SLIC is applied to the rest of the image (**right**). Every superpixel and object cluster is converted into a distinct node.

7.2 Designing Graphs with Clean Object Nodes

The results discussed in Sec. 7.1 show that GCN struggles with identifying objects. The main reason for this is the construction of the graph itself. Small-grained objects, represented by only a few pixels in the image, are represented by even fewer nodes in the graph due to superpixel segmentation. Additionally, object pixels can get clustered into a superpixel with other background pixels, resulting in severe information loss. The performance of GCN, however, greatly hinges on the quality of input graphs, i.e. if the graph is noisy, the GCN's predictive power is hindered.

To tackle this issue and test if a different graph generation method can influence performance, we re-designed our graphs for UCMerced. We follow the same procedure as described in Cha. 4, but with a slight alteration to the superpixel sampling (Sec. 4.2). First, we extract the object pixels based on the ground truth segmentation maps [31]. After that, the rest of the image is segmented into superpixels, and we proceed with the graph generation.

However, every distinct object is transformed into a single node. An object node contains the averaged object RGB values as features and the coordinates of the object's centre as the node position, like the superpixel nodes. Thus, all objects in an image are clearly separated from their surroundings in the generated graph. An example segmentation is visualised in Fig. 7.7. Since only object pixels need to be labelled, the annotation effort remains relatively low.

Fig.7.8 compares the AP scores of the GCN baseline with the GCN results on the object graphs (denoted by GCN_noAug_obj). Overall, the cleaner object graphs improve GCN performance, as reflected by the increased APmac score of 0.3251. It shows improvements for all classes except for *airplane*, *tanks*, *pavement*, *sea*, *trees*. With the exception of *airplane* and *tanks*, all these classes are texture-based. *court*, *ship*, *dock*, *field*, *sand*, and *water* profited the most from the object graphs.

One drawback of this method is that objects which are generally larger or have a very distinct shape are summarised into a single node, which leads to a significant information loss when compared to the superpixels as reflected in the performance on *airplane* and *tanks*. However, small objects without very distinct shapes greatly profit from this type of segmentation, such as ships. As a result, the performance on classes that are strongly correlated with *ship*, e.g. *water* or *docks* also increases.

Finally, a small alteration to the superpixel sampling and graph generation method achieved almost the same APmac as the best augmentation on standard superpixel UCMerced graphs, which shows that generating clean and meaningful graphs can have a greater influence on the performance of GCN than GraphDA.

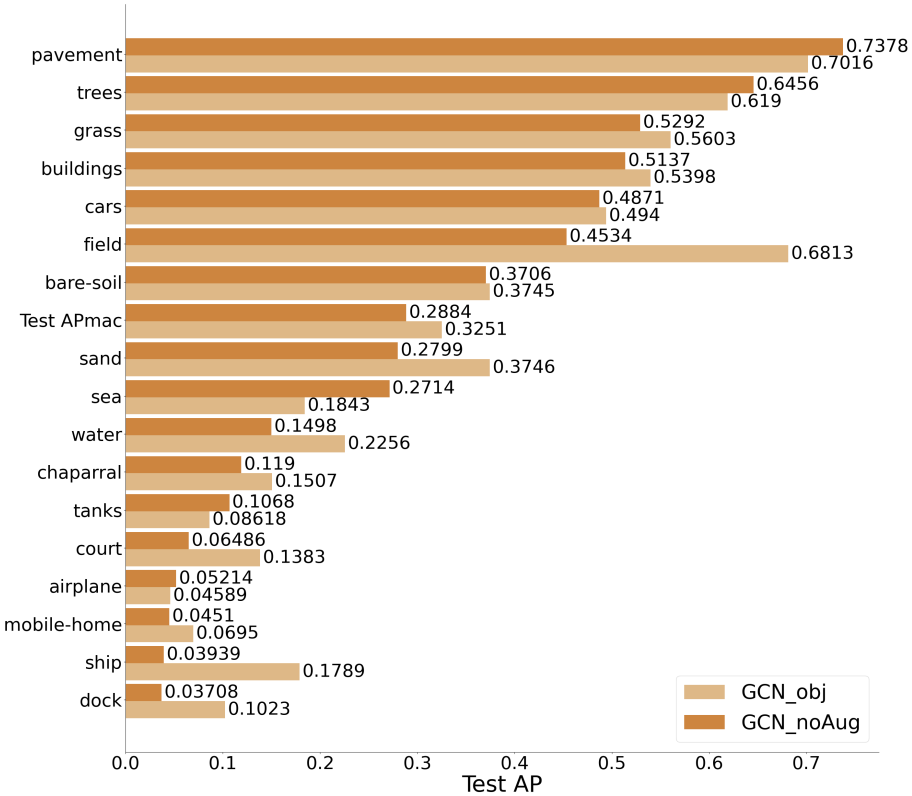


Figure 7.8: Test AP scores of GCN_noAug and baseline GCN on object graphs GCN_obj.

8 | Conclusion

This thesis investigated how MLC on RS satellite images can profit from GCNs and GraphDA. We thoroughly tested and evaluated the four already existing GraphDA techniques, *EdgeDrop*, *EdgeAdd*, *AttributeMasking*, and *Subgraph*, as well as the newly proposed GraphDAs, *ReOrga*, *VerticalConcat*, and *RandSquare*, on multiple MLC satellite datasets. The analysis leads to the following conclusions.

Similar to other ML methods, the performance of GCN greatly depends on the dataset but even more on the quality of the input graphs. Converting an image into a graph can severely compress the image data and lead to information loss. Small object classes only represented by a few pixels in a multi-label image are represented by even fewer nodes in a superpixel graph. Furthermore, the objects' information can get blurred if the superpixels have an imprecise boundary adherence. This disadvantages GCN against architectures specifically designed for images, such as ResNet18, that can directly be applied to high-resolution image data.

But, GCN is indifferent to varying sizes of input images and very lightweight. Additionally, it excels at understanding the relationship between classes and has shown promising results on texture-based datasets. It achieved good average precision despite the compressed information, especially for MLC datasets with a higher average class occurrence per sample. Thus, GCN could be well applied to complex texture-based MLC datasets whose samples contain strong relationships between the classes., e.g. crop maps.

Our results further emphasise how important graph generation and the choice of GCN architecture are. Too many layers can increase the risk of over-smoothing, which can cause even more severe information loss for object classes, while coarse texture classes seem to profit from a larger receptive field. These sensitivities can limit the initial flexibility of GCN. However, GraphDA can alleviate some of these drawbacks.

Our analysis has shown that different types of classes can profit from different GraphDAs. For example, object-based classes profit from augmentations that reduce the receptive field of GCN, such as *EdgeDrop*. Thus, by randomly changing the graph

structure and connectivity, GraphDAs can help the network to dynamically adjust to new datasets making the design choices more flexible and increasing the network's generalisation capabilities.

It is, however, important to consider the domain the graphs originate from since augmentations can drastically change a graph's semantic and structural information. The already existing GraphDAs, we examined have been shown to improve model performance in previous works focused on graph domains other than RS. Yet, only *EdgeDrop* yielded consistently favourable results on the RS datasets, while *Subgraph* could only improve APmac on UCMerced, and *EdgeAdd* and Attribute Masking generally hindered the model. On the other hand, Our newly proposed GraphDAs developed based on superpixel graphs have all shown positive effects on average precision. Although their improvement on the overall performance is marginal, they positively affect some classes. These differences can be attributed to the unique characteristics of texturised classes with repetitive patterns and object classes with fine-grained individual features. In short, GraphDAs specifically developed for superpixel graphs have a much lower risk of harming model performance and, consequently, are more stable.

Graph generation and domain knowledge are essential not only for the network design itself but also for developing new GraphDAs. However, GNNs lack interpretability, making it difficult to understand which information is important and benefits the model and which does not. Hence, the design of graph generation pipelines and universal GraphDAs is challenging and a limiting factor in this thesis. Many image DAs that benefit RS images can not simply be translated into GraphDAs.

Taken together, this work has given insight into how challenges in RS MLC, such as imbalanced data and a large variety of classes and their complex relationships, can affect GCNs, or GNNs in general, and influence the effectiveness of GraphDA techniques. Additionally, we analysed how existing GraphDAs can be applied to RS data and proposed new augmentations specifically for RS superpixel graphs.

Future research should conduct additional tests to determine if the proposed augmentations also work on other graph types. Moreover, this thesis focused on RGB images, but hyper-spectral data is essential to RS image analysis. Thus, investigating how GCN and GraphDAs perform on hyper-spectral multi-label RS images should be of great importance for future work, especially since GCN could potentially profit from the additional information in the nodes. Lastly, a key component will be discovering new ways of generating meaningful image graphs since they built the foundation for good graph classification and the effectiveness of GraphDA.

Bibliography

[1] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations", in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 5812–5823.

[2] M. Marrium and A. Mahmood, "Data augmentation for graph data: Recent advancements", 2022. DOI: 10.48550/ARXIV.2208.11973.

[3] Q. Diao, Y. Dai, C. Zhang, Y. Wu, X. Feng, and F. Pan, "Superpixel-based attention graph neural network for semantic segmentation in aerial images", *Remote Sensing*, vol. 14, no. 2, 2022, ISSN: 2072-4292. DOI: 10.3390/rs14020305.

[4] V. Vasudevan, M. Bassenne, M. T. Islam, and L. Xing, "Image classification using graph neural network and multiscale wavelet superpixels", *Pattern Recogn. Lett.*, vol. 166, no. C, pp. 89–96, Feb. 2023, ISSN: 0167-8655. DOI: 10.1016/j.patrec.2023.01.003.

[5] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks", *Journal of Machine Learning Research*, vol. 24, May 11, 2022.

[6] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 3008–3017. DOI: 10.1109/CVPRW50498.2020.00359.

[7] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation strategies from data", in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 113–123, ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00020.

[8] A. Oubara, F. Wu, A. Amamra, and G. Yang, "Survey on remote sensing data augmentation: Advances, challenges, and future perspectives", in *Advances in Computing Systems and Applications*, M. R. Senouci, S. Y. Boulahia, and M. A. Benatia, Eds., Cham: Springer International Publishing, 2022, pp. 95–104, ISBN: 978-3-031-12097-8.

[9] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization", *International Conference on Learning Representations*, 2018.

[10] C. Summers and M. J. Dinneen, "Improved mixed-example data augmentation", in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019, pp. 1262–1270. DOI: 10.1109/WACV.2019.00139.

[11] K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey", *SIGKDD Explor. Newsl.*, vol. 24, no. 2, pp. 61–77, Dec. 2022, ISSN: 1931-0145. DOI: 10.1145/3575637.3575646.

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021, ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386.

[13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?", 2019. DOI: 10.48550/arXiv.1810.00826.

[14] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks", in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Oct. 2018, pp. 5453–5462.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[16] M. Zhang and Y. Chen, "Link Prediction Based on Graph Neural Networks", in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018.

[17] P. C. Avelar, A. R. Tavares, T. T. da Silveira, C. R. Jung, and L. C. Lamb, "Superpixel image classification with graph attention networks", in *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2020, pp. 203–209. DOI: 10.1109/SIBGRAPI51738.2020.00035.

[18] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks", in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 2220–2231.

[19] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning", *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[20] Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le, "Unsupervised Data Augmentation for Consistency Training", in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 6256–6268.

[21] X. Hao, L. Liu, R. Yang, L. Yin, L. Zhang, and X. Li, "A Review of Data Augmentation Methods of Remote Sensing Image Target Recognition", en, *Remote Sensing*, vol. 15, no. 3, p. 827, Jan. 2023, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2072-4292. DOI: 10.3390/rs15030827.

[22] X. Yu, X. Wu, C. Luo, and P. Ren, "Deep learning in remote sensing scene classification: A data augmentation enhanced convolutional neural network framework", *GIScience & Remote Sensing*, vol. 54, no. 5, pp. 741–758, 2017. DOI: 10.1080/15481603.2017.1323377.

[23] J. M. Haut, M. E. Paoletti, J. Plaza, A. Plaza, and J. Li, "Hyperspectral Image Classification Using Random Occlusion Data Augmentation", *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 11, pp. 1751–1755, Nov. 2019, Conference Name: IEEE Geoscience and Remote Sensing Letters, ISSN: 1558-0571. DOI: 10.1109/LGRS.2019.2909495.

[24] E. Dai, C. Aggarwal, and S. Wang, "Nrgnn: Learning a label noise resistant graph neural network on sparsely and noisily labeled graphs", in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21, Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 227–236, ISBN: 9781450383325. DOI: 10.1145/3447548.3467364.

[25] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 4, pp. 3438–3445, Apr. 2020. DOI: 10.1609/aaai.v34i04.5747.

[26] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Graphcrop: Subgraph cropping for graph classification", *CoRR*, vol. abs/2009.10564, 2020. arXiv: 2009.10564. [Online]. Available: https://arxiv.org/abs/2009.10564.

[27] P. Mishra, A. Piktus, G. Goossen, and F. Silvestri, "Node masking: Making graph neural networks generalize and scale better", no. arXiv:2001.07524, May 16, 2021. arXiv: 2001.07524[cs,stat].

[28] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification", in *Proceedings of the Web Conference 2021*, Ljubljana Slovenia: ACM, Apr. 19, 2021, pp. 3663–3674, ISBN: 978-1-4503-8312-7. DOI: 10.1145/3442381.3449796.

[29] I. Demir, K. Koperski, D. Lindenbaum, *et al.*, "DeepGlobe 2018: A challenge to parse the earth through satellite images", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Salt Lake City, UT, USA: IEEE, Jun. 2018, pp. 172–17 209, ISBN: 978-1-5386-6100-0. DOI: 10.1109/CVPRW.2018.00031.

[30] B. Chaudhuri, B. Demir, S. Chaudhuri, and L. Bruzzone, "Multilabel remote sensing image retrieval using a semisupervised graph-theoretic method", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 2, pp. 1144–1158, 2018. DOI: 10.1109/TGRS.2017.2760909.

[31] Z. Shao, K. Yang, and W. Zhou, "Performance evaluation of single-label and multi-label remote sensing image retrieval using a dense labeling dataset", *Remote Sensing*, vol. 10, no. 6, 2018, ISSN: 2072-4292. DOI: 10.3390/rs10060964.

[32] D. Stutz, A. Hermans, and B. Leibe, "Superpixels: An evaluation of the state-of-the-art", *Computer Vision and Image Understanding*, vol. 166, pp. 1–27, 2018, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2017.03.007.

[33] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012. DOI: 10.1109/TPAMI.2012.120.

[34] M. Van den Bergh, X. Boix, G. Roig, and L. Van Gool, "SEEDS: Superpixels Extracted Via Energy-Driven Sampling", *International Journal of Computer Vision*, vol. 111, no. 3, pp. 298–314, Feb. 2015, ISSN: 1573-1405. DOI: 10.1007/s11263-014-0744-2.

[35] Z. Li and J. Chen, "Superpixel segmentation using linear spectral clustering", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1356–1363. DOI: 10.1109/CVPR.2015.7298741.

[36] A. Vedaldi and S. Soatto, "Quick Shift and Kernel Methods for Mode Seeking", in *Computer Vision – ECCV 2008*, D. Forsyth, P. Torr, and A. Zisserman, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 705–718, ISBN: 978-3-540-88693-8.

[37] P. Neubert and P. Protzel, "Superpixel Benchmark and Comparison", 2012.

[38] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering", in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[39] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 5425–5434, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.576.

[40] S. Van Dongen, "Graph clustering via a discrete uncoupling process", *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 121–141, 2008. DOI: 10.1137/040608635.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", en, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[42] S. Ruder, "An overview of gradient descent optimization algorithms", Jun. 2017, arXiv:1609.04747 [cs]. DOI: 10.48550/arXiv.1609.04747.

[43] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning", in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial*

Intelligence, ser. AAAI'18/IAAI'18/EAAI'18, New Orleans, Louisiana, USA: AAAI Press, 2018, ISBN: 978-1-57735-800-8.

[44] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification", in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[45] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank", in *International Conference on Learning Representations*, 2018.

[46] S. Shi, K. Qiao, J. Yang, B. Song, J. Chen, and B. Yan, "Over-sampling strategy in feature space for graphs based class-imbalanced bot detection", 2023. arXiv: 2302.06900 [cs.CV].

[47] F. Hu, L. Wang, Q. Liu, S. Wu, L. Wang, and T. Tan, "GraphDIVE: Graph classification by mixture of diverse experts", in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, Vienna, Austria: International Joint Conferences on Artificial Intelligence Organization, Jul. 2022, pp. 2080–2086, ISBN: 978-1-956792-00-3. DOI: 10.24963/ijcai.2022/289.

[48] S. Pan and X. Zhu, "Graph classification with imbalanced class distributions and noise", in *International Joint Conference on Artificial Intelligence*, vol. 23, 2013.

[49] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks", in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[50] Q. Huang, M. Yamada, Y. Tian, D. Singh, and Y. Chang, "Graphlime: Local interpretable model explanations for graph neural networks", *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6968–6972, 2023. DOI: 10.1109/TKDE.2022.3187455.

A | Appendix

Table A.1: Globe15 all results.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.7755	0.8622	0.4013	0.754
EdgeDrop0.3	-	0.3	0.7236	0.8031	0.309	0.676
AttributeMasking0.3	-	0.3	0.6825	0.771	0.2747	0.6529
VerticalConcat0.3(α =0.3)	0.3	0.3	0.6137	0.7817	0.4709	0.6804
ReOrga0.3	-	0.3	0.6137	0.7801	0.4707	0.6751
ReOrga0.5	-	0.5	0.6132	0.782	0.4723	0.6788
VerticalConcat0.3(α =0.5)	0.5	0.3	0.6127	0.7814	0.4691	0.6802
RandSquare0.3	-	0.3	0.6122	0.7806	0.4709	0.6806
GCN_noAug	-	-	0.6122	0.7814	0.4699	0.6802
ReOrga0.7	-	0.7	0.6121	0.781	0.4646	0.6746
VerticalConcat0.3(α =0.7)	0.7	0.3	0.6111	0.7806	0.4725	0.6811
RandNodeDrop0.3	-	0.3	0.6037	0.7672	0.4337	0.6545
Subgraph0.3	-	0.3	0.5874	0.7534	0.4384	0.6586
EdgeDrop0.5	-	0.5	0.5818	0.7474	0.3522	0.6096
Subgraph0.5	-	0.5	0.5781	0.7395	0.4317	0.6521
Subgraph0.7	-	0.7	0.5763	0.7203	0.4256	0.6406
RandNodeDrop0.5	-	0.5	0.5646	0.7279	0.3732	0.6224
VerticalConcat0.7(α =0.7)	0.7	0.7	0.5549	0.745	0.4376	0.6523
EdgeDrop0.7	-	0.7	0.5535	0.716	0.3128	0.5847
VerticalConcat0.5(α =0.3)	0.3	0.5	0.5501	0.7381	0.4571	0.661
RandNodeDrop0.7	-	0.7	0.5482	0.7081	0.3725	0.6181
VerticalConcat0.5(α =0.5)	0.5	0.5	0.5474	0.7323	0.4437	0.6484
VerticalConcat0.5(α =0.7)	0.7	0.5	0.5456	0.7394	0.4434	0.6534
EdgeAdd0.3	-	0.3	0.5451	0.728	0.4491	0.6562
VerticalConcat0.7(α =0.5)	0.5	0.7	0.5441	0.7363	0.4455	0.657
VerticalConcat0.7(α =0.3)	0.3	0.7	0.537	0.7282	0.436	0.652
RandSquare0.7	-	0.7	0.5223	0.7162	0.4007	0.6183
RandSquare0.5	-	0.5	0.5212	0.7149	0.3927	0.614
EdgeAdd0.5	-	0.5	0.493	0.6702	0.4011	0.6174
AttributeMasking0.5	-	0.5	0.4825	0.6917	0.2802	0.5846
EdgeAdd0.7	-	0.7	0.4636	0.6582	0.3176	0.6026
AttributeMasking0.7	-	0.7	0.4181	0.6446	0.2432	0.5626

Table A.2: Globe2 all results.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.8089	0.8671	0.4675	0.7794
RandSquare0.3	-	0.3	0.7728	0.8473	0.451	0.7685
VerticalConcat0.3(α =0.3)	0.3	0.3	0.7719	0.8461	0.4445	0.7612
ReOrga0.3	-	0.3	0.771	0.8436	0.4423	0.7621
EdgeDrop0.3	-	0.3	0.7701	0.838	0.4388	0.7547
GCN_noAug	-	-	0.7628	0.8443	0.447	0.7636
VerticalConcat0.3(α =0.5)	0.5	0.3	0.7602	0.8433	0.4418	0.7614
VerticalConcat0.3(α =0.7)	0.7	0.3	0.7592	0.844	0.4391	0.7614
RandNodeDrop0.3	-	0.3	0.7561	0.8298	0.4289	0.7518
EdgeDrop0.5	-	0.5	0.755	0.816	0.4115	0.7251
Subgraph0.3	-	0.3	0.7546	0.8126	0.3908	0.7297
Subgraph0.5	-	0.5	0.742	0.8024	0.3826	0.7229
Subgraph0.7	-	0.7	0.732	0.7905	0.3755	0.7141
EdgeAdd0.3	-	0.3	0.7298	0.787	0.3885	0.7191
AttributeMasking0.3	-	0.3	0.7292	0.8009	0.3838	0.7152
VerticalConcat0.5(α =0.7)	0.7	0.5	0.7225	0.7934	0.4022	0.7047
VerticalConcat0.7(α =0.7)	0.7	0.7	0.7183	0.7876	0.4085	0.7051
RandNodeDrop0.5	-	0.5	0.7158	0.8051	0.3923	0.7188
VerticalConcat0.5(α =0.3)	0.3	0.5	0.7069	0.7679	0.3903	0.6898
VerticalConcat0.7(α =0.5)	0.5	0.7	0.7068	0.7812	0.3868	0.6891
RandNodeDrop0.7	-	0.7	0.703	0.7823	0.3659	0.7032
RandSquare0.7	-	0.7	0.7009	0.793	0.4061	0.7135
RandSquare0.5	-	0.5	0.7005	0.7926	0.3994	0.7081
EdgeDrop0.7	-	0.7	0.6988	0.7854	0.3658	0.6903
VerticalConcat0.5(α =0.5)	0.5	0.5	0.6981	0.7826	0.3953	0.694
ReOrga0.5	-	0.5	0.6918	0.8452	0.5877	0.7557
ReOrga0.7	-	0.7	0.6899	0.8446	0.5915	0.7574
AttributeMasking0.5	-	0.5	0.6783	0.7619	0.3481	0.6789
EdgeAdd0.5	-	0.5	0.6605	0.7268	0.3328	0.6626
AttributeMasking0.7	-	0.7	0.6491	0.7284	0.2869	0.6322
VerticalConcat0.7(α =0.3)	0.3	0.7	0.6234	0.7698	0.541	0.6915
EdgeAdd0.7	-	0.7	0.5048	0.6979	0.3578	0.6487

Table A.3: Globe25 all results.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.736	0.848	0.5138	0.7557
ReOrga0.5	-	0.5	0.7335	0.8538	0.6231	0.7646
VerticalConcat0.3(α =0.7)	0.7	0.3	0.7326	0.8554	0.6286	0.7643
VerticalConcat0.3(α =0.5)	0.5	0.3	0.7324	0.8529	0.6179	0.7569
VerticalConcat0.3(α =0.3)	0.3	0.3	0.7319	0.8555	0.6259	0.7638
GCN_noAug	-	-	0.7319	0.8551	0.6314	0.7659
ReOrga0.3	-	0.3	0.7309	0.8527	0.6177	0.7622
RandSquare0.3	-	0.3	0.7303	0.8548	0.627	0.7655
EdgeDrop0.3	-	0.3	0.7274	0.8498	0.599	0.7506
ReOrga0.7	-	0.7	0.7242	0.8489	0.6214	0.7623
RandNodeDrop0.3	-	0.3	0.7207	0.8433	0.6091	0.7558
Subgraph0.3	-	0.3	0.7092	0.8224	0.5509	0.7335
VerticalConcat0.5(α =0.7)	0.7	0.5	0.7075	0.8344	0.5766	0.7365
Subgraph0.5	-	0.5	0.7032	0.8138	0.537	0.7262
VerticalConcat0.7(α =0.7)	0.7	0.7	0.7017	0.8323	0.5828	0.7382
EdgeDrop0.5	-	0.5	0.7011	0.8297	0.5675	0.7266
VerticalConcat0.7(α =0.5)	0.5	0.7	0.6997	0.831	0.5676	0.7401
VerticalConcat0.5(α =0.5)	0.5	0.5	0.6976	0.8285	0.5676	0.737
VerticalConcat0.5(α =0.3)	0.3	0.5	0.6929	0.8239	0.5622	0.7328
Subgraph0.7	-	0.7	0.6915	0.7994	0.5188	0.7128
RandNodeDrop0.5	-	0.5	0.6889	0.8191	0.5653	0.735
RandSquare0.5	-	0.5	0.683	0.8148	0.5794	0.7349
RandSquare0.7	-	0.7	0.6819	0.8087	0.5624	0.7208
VerticalConcat0.7(α =0.3)	0.3	0.7	0.6788	0.8136	0.5471	0.7242
EdgeDrop0.7	-	0.7	0.6685	0.7887	0.5062	0.6884
RandNodeDrop0.7	-	0.7	0.6673	0.7955	0.5544	0.7146
AttributeMasking0.3	-	0.3	0.6601	0.8046	0.5412	0.717
EdgeAdd0.3	-	0.3	0.6564	0.7997	0.5506	0.7257
AttributeMasking0.5	-	0.5	0.5978	0.7599	0.4714	0.6896
EdgeAdd0.5	-	0.5	0.5777	0.7169	0.4587	0.6733
EdgeAdd0.7	-	0.7	0.5609	0.6882	0.3919	0.6659
AttributeMasking0.7	-	0.7	0.5428	0.7275	0.4079	0.6657

Table A.4: UCMerced all results.

Name	α	p	APmac	APmic	f1mac	f1mic
ResNet18	-	-	0.7145	0.7639	0.5456	0.6923
EdgeDrop0.7	-	0.7	0.3285	0.5308	0.0728	0.224
EdgeDrop0.7_2layer	-	0.7	0.318	0.5581	0.0715	0.2931
EdgeDrop0.5	-	0.5	0.3069	0.553	0.0813	0.3101
RandNodeDrop0.3	-	0.3	0.3048	0.5472	0.0839	0.3549
Subgraph0.5	-	0.5	0.3033	0.5397	0.1119	0.3917
RandSquare0.3	-	0.3	0.3021	0.5475	0.1015	0.3936
VerticalConcat0.3(α =0.7)	0.7	0.3	0.3021	0.5475	0.1015	0.3936
EdgeDrop0.3	-	0.3	0.3013	0.5498	0.0839	0.3479
RandSquare0.7	-	0.7	0.3012	0.5467	0.0986	0.3843
RandNodeDrop0.7	-	0.7	0.2991	0.5328	0.0838	0.2895
ReOrga0.7	-	0.7	0.2904	0.5475	0.093	0.3761
ReOrga0.5	-	0.5	0.2897	0.5472	0.0923	0.3752
ReOrga0.3	-	0.3	0.2894	0.5468	0.0923	0.3748
VerticalConcat0.3(α =0.3)	0.3	0.3	0.2884	0.5463	0.0926	0.3759
GCN_noAug	-	-	0.2884	0.5463	0.0926	0.3759
VerticalConcat0.3(α =0.3)	0.3	0.3	0.2883	0.5462	0.0926	0.3759
VerticalConcat0.3(α =0.5)	0.5	0.3	0.2882	0.5494	0.1054	0.4034
RandSquare0.5	-	0.5	0.2846	0.5445	0.0915	0.3743
RandNodeDrop0.5	-	0.5	0.2835	0.5449	0.0812	0.3276
Subgraph0.3	-	0.3	0.2827	0.5434	0.0834	0.3676
Subgraph0.7	-	0.7	0.2782	0.5415	0.0835	0.3763
VerticalConcat0.5(α =0.3)	0.3	0.5	0.2749	0.515	0.0681	0.3065
AttributeMasking0.5	-	0.5	0.2716	0.524	0.0741	0.3392
AttributeMasking0.3	-	0.3	0.2707	0.529	0.081	0.3684
AttributeMasking0.7	-	0.7	0.2525	0.5149	0.0449	0.2726
VerticalConcat0.7(α =0.3)	0.3	0.7	0.252	0.5177	0.0449	0.2726
EdgeAdd0.7	-	0.7	0.2508	0.4975	0.077	0.3506
EdgeAdd0.5	-	0.5	0.2493	0.5019	0.0788	0.359
EdgeAdd0.3	-	0.3	0.2467	0.5062	0.0783	0.3563
VerticalConcat0.7(α =0.7)	0.7	0.7	0.2433	0.5145	0.0449	0.2726
VerticalConcat0.5(α =0.5)	0.5	0.5	0.2337	0.5094	0.0727	0.3344
VerticalConcat0.7(α =0.5)	0.5	0.7	0.2211	0.5111	0.0627	0.3064
VerticalConcat0.5(α =0.7)	0.7	0.5	0.2186	0.5126	0.0499	0.2809