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

JÚLIA PELAYO RODRIGUES

Graph Neural Networks for image classification: comparing approaches for building graphs

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

Advisor: Prof. Dr. Joel Luis Carbonera

Porto Alegre
February 2024

ABSTRACT

Graph Neural Networks (GNNs) is an approach that allows applying deep learning techniques to non-Euclidean data, such as graphs and manifolds. Over the past few years, graph convolutional networks (GCNs) and graph attention networks (GATs), among other specific kinds of convolutional GNNs, have been applied to image classification problems. To do so, images should be represented as graphs. This process usually involves over-segmenting images into non-regular regions called superpixels, which are mapped to graph nodes, characterized by features representing the superpixel information, and connected to other nodes. However, there are several different ways of transforming images into graphs. This work focuses on applying graph convolutional networks and graph attention networks in image classification problems for images over-segmented into superpixels. We systematically evaluate the impact of different approaches for representing images as graphs in the performance achieved by the resulting GCN and GAT models, comparing, as well, the differences between them. Namely, we analyze the impacts of the degree of segmentation (number of nodes), the set of features chosen to represent each superpixel as a node, and the method for building the edges between nodes. We concluded that the performance is positively impacted when increasing the number of nodes, considering rich sets of features, and considering only connections between similar regions in the resulting graph for GCNs and one-headed GATs, while multi-headed GAT models can take advantage of information from neighboring regions and region adjacency information provided by region adjacency graphs.

Keywords: Graph neural networks. image classification. superpixels. graph convolutional networks.

Redes Neurais de Grafos para classificação de imagens: comparando métodos para construção de grafos

RESUMO

Redes Neurais de Grafos (GNNs) são uma abordagem que permite aplicar técnicas de aprendizado profundo a dados não-euclidianos, como grafos e variedades. Nos últimos anos, Redes Convolucionais de Grafos (GCNs) e Redes de Atenção a Grafos (GATs), entre outros tipos de GNNs convolucionais, têm sido aplicados a problemas de classificação de imagens. Para isso, as imagens devem ser representadas como grafos. Esse processo geralmente envolve a sobre-segmentação de imagens em regiões não-regulares chamadas superpixeis, que são mapeadas para vértices do grafo, descritas por atributos que representam as informações do superpixel e conectadas a outros vértices em vizinhanças. No entanto, existem diversas maneiras de transformar imagens em grafos. Este trabalho trata da aplicação de GCNs e GATs a problemas de classificação de imagens sobre-segmentadas em superpixels. Avaliamos sistematicamente o impacto de diferentes abordagens para representar imagens como grafos no desempenho alcançado por modelos GCN e GAT simples, comparando também as diferenças entre estes. Em particular, analisamos os impactos do grau de segmentação da imagem (ou, equivalentemente, do número de vértices do grafo), do conjunto de atributos escolhidos para representar cada superpixel como vértice e do método para construir as arestas entre os vértices. Concluímos que o desempenho é positivamente impactado ao aumentar o número de vértices, considerar conjuntos ricos de atributos e considerar apenas conexões entre regiões semelhantes no grafo para GCNs e para GATs de um único foco de atenção, enquanto modelos GATs de vários focos podem se beneficiar das informações de regiões vizinhas e da informação de adjacência fornecidas pelos grafos de regiões adjacentes.

Palavras-chave: Redes neurais de grafos. Classificação de imagens. Superpixeis. Redes convolucionais de grafos.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CNN	Convolutionl Neural Network
GNN	Graph Neural Network
GCN	Graph Convolutional Network
GAT	Graph Attention Network
RAG	Region Adjacency Graph
DAG	Directed Acyclic Graph
KNN	K-Nearest Neighbors

CONTENTS

1 INTRODUCTION

Since the past decade, Machine Learning algorithms have been achieving increasing performances in many tasks due to the growth in computational power and data availability and also due to advances in research in the field. One of the most impacted areas is computer vision, where state-of-the-art deep neural network models such as AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), ResNet (HE et al., 2016) and EfficientNet (TAN; LE, 2019) have reached staggering performances. This is especially true for the task of image classification, where, given an image as input, the algorithm must determine to which of a predetermined number of classes it belongs.

In this context, Graph Neural Networks (GNNs) (SCARSELLI et al., 2009)) is a relatively recent approach that generalizes neural networks, allowing them to process non-Euclidean data such as graphs. This capability makes it possible to apply deep learning approaches to a vast set of problems whose data can be modeled as graphs. Since they were proposed, GNNs have been applied in different areas (WU et al., 2020), such as Bioinformatics (ZHANG et al., 2021), Particle Physics (SHLOMI; BATTAGLIA; VLIMANT, 2020), Neuroscience (BESSADOK; MAHJOUB; REKIK, 2022), natural language processing (WU et al., 2023), material science and Chemistry (REISER et al., 2022), Computer vision (CHEN et al., 2022), etc.

In the last few years, several studies have investigated how to apply GNNs for image classification (HONG et al., 2020; CHEN et al., 2020; ZHANG et al., 2023; DU et al., 2023; TANG et al., 2022). Most of these studies are based on a specific class of GNNs called *convolutional graph neural networks*, which can be understood as a generalization of *convolutional neural networks* to graph-structured data. Among them are especially highlighted the *Graph Convolutional Networks* (GCNs) (KIPF; WELLING, 2017) and the *Graph Attention Networks* (VELIčKOVIć et al., 2018).

However, to apply GNNs to image classification it is necessary to represent the image information as a graph. Typically, these approaches involve over-segmenting images into non-regular regions called superpixels (DEFFERRARD; BRESSON; VANDERGHEYNST, 2016; MONTI et al., 2017) that are mapped to nodes in a graph. Yet, there are many different ways of building the resulting graph, depending on choices made by the designer on different aspects. For example, the images can be segmented in different degrees, resulting in different numbers of nodes in the graph and different densities of pixels per node. Besides that, there are different approaches for defining the edges among the nodes that represent superpixels, such as adopting fully connected graphs (MONTI et al., 2017), K-Nearest Neighbors, region adjacency graphs (AVELAR et al., 2020), dynamic approaches (LINH; YOUN, 2021), etc. Furthermore, there are different ways of assigning features to nodes to describe each superpixel's characteristics.

As far as we are aware, the literature does not provide any systematic comparison of how different ways of building graphs impact the performance achieved by GNN models in image classification tasks. Such systematic evaluation would be a valuable reference for supporting the most effective choices when designing models in this context.

In this work, we focus on using Graph Convolutional Networks and Graph Attention Networks for the classification of superpixel segmented images using the SLIC (ACHANTA et al., 2012) method in its adaptive (SLICO) variant. Our objective is to systematically evaluate the impact of the following graph-building choices on the performance of simple GCN and GAT models: (I) the degree of segmentation (that defines the number of nodes in the resulting graph); (II) the selection of features for characterizing each node and representing the superpixels' information; and (III) the method for defining edges between node pairs in the resulting graph.

We have found that the performance achievable by the GCN and GAT models is, in general, positively impacted by choosing rich representative feature sets, however, some features such as pixel density can have negative impacts. The models also benefit from increasing the number of superpixels per image – although the positive impact grows smaller as the number of pixels per superpixel approaches one. Furthermore, the biggest difference in behavior between GCNs and GATs is seen in their responses to edge-building methods: GCNs respond better to neighborhoods that encompass solely similar regions (that is, considering descriptive features in the calculation of the distance between nodes and limiting the maximum degree); while GATs with multiple attention heads can leverage information from contrasting regions and achieve top performances with region adjacency graphs.

We also conclude that our GAT model outperforms the GCN model in every case. However, neither model achieves the results offered by classical CNN models AlexNet and EfficientNet. Still, the resources needed for the GNN models, time- and, especially, memory-wise, are drastically fewer.

The remainder of this work is structured as follows. In Chapter 2 we discuss the main concepts fundamental to this work, namely the machine-learning fundamentals and the issues involved in representing images as graphs. Chapter 3 concerns the related work. In Chapter 4 we present the methodology of our experiments and discuss our results. Finally, Chapter 5 presents the conclusions.

2 BACKGROUND

In this Chapter, we provide an overview of the main concepts that are fundamental to this work. Section 2.1 presents the machine learning basis upon which the work is based, focusing on Graph Neural Networks and the mechanisms for evaluating them in the context of image classification. Section 2.2 details the design choices involved in transforming images into graphs: a necessary step so that they can be processed by GNNs.

2.1 Machine learning

According to Goodfellow, Bengio and Courville (2016), a *machine learning* algorithm is an algorithm capable of learning from data. That is, it progressively improves its performance in a certain task as it gains experience.

Learning in ML can be classified into roughly three categories: supervised learning, reinforcement learning, and unsupervised learning. This work focuses on *supervised learning*, where each example in a dataset has its features and its label or target, and the model learns a function that maps a given set of features to the suitable label. The learning process is carried out on a training dataset by iteratively comparing the output provided by the model for a given instance to its known ground-truth, and adjusting the model's parameters in order to decrease this error (GOODFELLOW; BENGIO; COURVILLE, 2016).

Specifically, we focus on the task of *image classification*, where the datasets are composed of labeled images, each label specifying to which, of a fixed number of classes, the image belongs. The task of the model is to predict, for any given image, its corresponding class. Image Classification (IC) aims to classify the image as a whole by assigning a specific label to it. Usually, labels in an IC task refer to objects that appear in the image, kinds of images (photographs, drawings, etc.), feelings (sadness, happiness, etc.), etc (LANCHANTIN et al., 2021).

In the machine learning field, many approaches have been proposed in the context of supervised learning, such as decision trees, support vector machines (BOSER; GUYON; VAPNIK, 1992; CORTES; VAPNIK, 1995), neural networks, etc. Especially among them, neural networks have achieved staggering performances in many tasks in the last decade, with impressive impacts in computer vision tasks.

Neural networks (NNs) are machine learning models, typically structured in multiple layers of processing units (e.g.: *perceptrons*). These units are connected to the adjacent layers' units via weighted edges. The goal of training a neural network is to build a model that approximates a function $f^*(\boldsymbol{x})$ that maps data inputs into desired outputs. This is done by adjusting the neural network's set of parameters (the weights of each link and biases of each unit) to minimize a loss function (GOODFELLOW; BENGIO; COURVILLE, 2016). In general, training a neural network involves three steps: *forward propagation*, *backpropagation*, and adjustment of its parameters.

During the *forward propagation*, samples in the training dataset are presented to the neural network. Then, this input flows through the layers, which apply transformations to the information, generating an output in the last layer.

Once an output is generated by the network for a given sample, the *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986) algorithm is used to compute the gradient of the loss function (that measures the dissimilarity between the generated output and the ground truth in supervised learning) with regard to each of the trainable parameters of the model (i.e., the weights of the edges that link the network's processing units) (GOODFELLOW; BENGIO; COURVILLE, 2016).

Finally, the *gradient descent* algorithm is applied to adjust the model parameters. Gradient descent is an optimization method that uses the gradients computed with backpropagation in order to find the set of weights that minimize the loss function, thus causing the model to yield better, more precise results. This is done by adjusting the model's weights in the opposite direction of the gradient.

In the remainder of this section, we discuss further Graph Neural Networks and their context in the field of ML (subsection 2.1.1) and the mechanisms for evaluating ML models (subsection 2.1.2).

2.1.1 Graph neural networks

Graph Neural Networks (GNNs) (SCARSELLI et al., 2009)) can be viewed as an approach that generalizes neural networks, allowing them to deal with non-Euclidean data such as graphs. Since they were proposed, GNNs have been applied in different areas (WU et al., 2020), such as Bioinformatics (ZHANG et al., 2021), Particle Physics (SHLOMI; BATTAGLIA; VLIMANT, 2020), Neuroscience (BESSADOK; MAHJOUB; REKIK, 2022), natural language processing (WU et al., 2023), material science and Chemistry (REISER et al., 2022), Computer vision (CHEN et al., 2022; TODESCATO et al., 2024; TODESCATO et al., 2023), etc. The first use of GNNs for image classification tasks was seen in, to the best of our knowledge, Monti et al. (2017). The most common ML approach for image classification as well as most computer vision tasks, that far predate the use of GNNs, is the use of Convolutional Neural Networks.

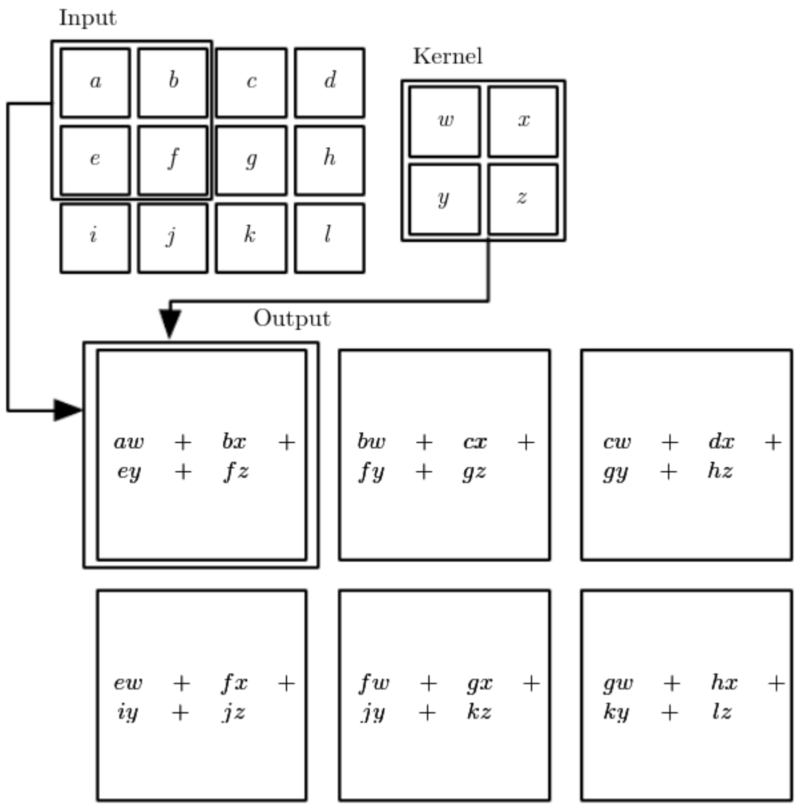
Convolutional Neural Networks (CNNs) (LECUN et al., 1998) are a kind of neural network where at least one of its layers performs a convolutional operation (GOODFELLOW; BENGIO; COURVILLE, 2016). CNNs have achieved breakthrough performances in the past decade, especially in computer vision tasks (e.g.: image classification), with models such as AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), ResNet (HE et al., 2016) and EfficientNet (TAN; LE, 2019).

A *convolution* is a well-known mathematical operation that can be described as the application of a filter along an input image, emphasizing different features of interest. Convolutions have wide use in computer vision, for blurring and sharpening images, emphasizing edges and particular shapes, etc (SZELISKI, 2022). In the context of CNNs, the specialized convolutional layer learns filters (also called *kernels*) that detect features of interest in the input data, resulting in *feature maps*. The feature maps indicate with what intensity the feature of interest was detected by the filter in different regions of the input image. Figure 2.1 shows a simple example of a 2D convolution, where the kernel slides along the input matrix, multiplying, element-wise, the kernel and the patch of the input the kernel overlaps. For every possible position of the kernel along the input, the sum of the referred multiplications corresponds to the value in the corresponding position of the final feature map. The trainable parameters of a convolutional layer are the weights of the kernel (GOODFELLOW; BENGIO; COURVILLE, 2016). It is important to note that every convolution has an equivalent operation in the frequency domain through Fourier transforms (SZELISKI, 2022).

However, CNNs (using convolutional kernels as described above) are only capable of processing grid-like (i.e. euclidian) data, such as traditional images, which are grids of pixels (GOODFELLOW; BENGIO; COURVILLE, 2016). Therefore they are not directly applicable to information that does not follow a regular topology, such as point clouds, meshes, and panoramas, as well as traditional images over-segmented into non-regular regions such as superpixels, all of which can be represented in the form of graphs.

Graph Neural Networks (GNNs) generalize many deep learning techniques to non-euclidian domains (i.e., graphs). Convolutional graph neural networks, in light of the tremendous success of CNNs, generalize the convolutional operation to non-euclidian

Figure 2.1: Example of a 2D convolution, restricted to positions where the kernel lies entirely within the input.



Source: Goodfellow, Bengio and Courville (2016)

data and have wide use in computer vision tasks (CHEN et al., 2022). There are two main approaches to graph convolutions: the *spectral* approach and the *spatial* approach (ZHANG et al., 2019).

The spectral approach to convolutions on graphs is based on spectral graph theory and was first notably used in the work of Bruna et al. (2014). In this approach, the graph Laplacian is used to compute the graph's Fourier transform, upon which is defined its convolutional operation. Many works have built upon this, including Graph Convolutional Networks (GCNs), proposed by Kipf and Welling (2017), which are used in many graph-based computer vision applications and will be discussed further in subsection 2.1.1.1.

In the spatial approach, the convolution is defined by directly gathering information from each node's local neighbors. Monti et al. (2017) proposed a general framework for spatial convolutional GNNs. Graph Attention Networks (GAT), proposed by Veličković et al. (2018), is an example of a widely used GNN with a spatial-based approach to graph convolution that adopts a self-attention mechanism. They will be discussed further in subsection 2.1.1.2.

However, as with CNNs, Convolutional GNN layers are rarely the sole component of a NN, in subsection 2.1.1.3 we discuss the architecture of GCN and GAT models.

2.1.1.1 Graph convolutional networks

GCNs were proposed by Kipf and Welling (2017) in the context of semi-supervised node classification, but they have been used in a great variety of contexts and tasks, including computer vision and image classification, often serving as the building block of more complex network architectures.

As a spectral approach to graph convolution, the layer-wise propagation rule of a GCN model $f(X, A)$ (where $X \in \mathbb{R}^{N \times C}$ is the input, i.e: a C-dimensional feature vector for each of the N nodes, and A is the input graph's adjacency matrix) is defined as the following:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\Big) \tag{2.1}$$

Where:

- $\tilde{A} = A + I_N$ is the adjacency matrix (A) with added self-loops (with I_N as the identity matrix);
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, the degree of each node;
- $W^{(l)}$ is a layer-specific trainable weight matrix;
- $\sigma(\cdot)$ is the activation function of the lth layer, such as ReLU;
- $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations of the lth layer, with $H^{(0)} = X$, the input.

As can be seen in Kipf's and Welling's work, the expression $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is derived from a series of simplifications of the definition of convolution on a graph as the multiplication of a signal by a filter in the Fourier domain, using the eigendecomposition of the graph Laplacian, as was initially used in the work of Bruna et al. (2014) and later expanded by Defferrard, Bresson and Vandergheynst (2016). One of the most notable consequences of the assumptions adopted is that the convolution is K-localized, with $K = 1$. That means the convolution is applied only to a node's immediate neighbors (including the node itself). The authors posit that this has the effect of alleviating the problem of overfitting and that the fixed computational budget allows for deeper networks, noting that stacking GCN layers has the effect of extending the informational flow to a distance equal to the number of layers stacked.

2.1.1.2 Graph attention networks

GATs were proposed by Veličković et al. (2018). They adopt a spatial approach to graph convolutions that is not based on the definition of graph convolution in the Fourier domain but, rather, directly sample each node's neighborhood (which includes the node itself) and apply a self-attention mechanism to determine how much importance the node should give to each neighbor's features. The layer-wise propagation rule of a GAT model $f(\boldsymbol{h})$ (where $\boldsymbol{h} = \{\vec{h_1}, \vec{h_2}, \dots \vec{h_N}\}$, $h_i \in \mathbb{R}^F$ are the feature vectors, each with F features, of the N nodes in the input graph) is defined as the following:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} a_{ij} \boldsymbol{W} \vec{h}_j\right) \tag{2.2}$$

Where:

- $\vec{h}'_i \in \mathbb{R}^{F'}$ is the new set of node features produced at the output of the layer;
- $\sigma(\cdot)$ is the activation function of the layer, such as ReLU;
- $\mathcal{N}_i$ is the neighborhood of the node i;
- $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$ is the layer-specific trainable weight matrix;
- a_{ij} is the attention mechanism that determines the attention coefficient of node i to node j. It is a masked attention mechanism, meaning that it will only be computed for the (in this case, immediate) neighbors of node i. In the work of Veličković et al. (2018), the attention mechanism is itself a single-layer feedforward neural network, whose output is normalized using the softmax function and parameterized by the trainable weight vector $\vec{\boldsymbol{a}} \in \mathbb{R}^{2F'}$:

$$a_{ij} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{\boldsymbol{a}}^T[\boldsymbol{W}\vec{h}_i \| \boldsymbol{W}\vec{h}_j]\Big)\Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\text{LeakyReLU}\Big(\vec{\boldsymbol{a}}^T[\boldsymbol{W}\vec{h}_i \| \boldsymbol{W}\vec{h}_k]\Big)\Big)} \tag{2.3}$$

That means that, for each GAT layer, at least two weight matrices will be trained: the standard weight matrix initially applied to each node's feature vector, $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$, and the attention mechanism's weight vector $\vec{\boldsymbol{a}} \in \mathbb{R}^{2F'}$.

It is possible to consider more than one *attention head* per layer, each one with its own attention mechanism executing the transformation in equation 2.2, so that each head may learn different ways to emphasize the information of the neighboring nodes. In that case, when using K independent attention mechanisms, and by considering $\|$ as representing concatenation, equation 2.2 is rewritten as follows:

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} a_{ij}^k \boldsymbol{W}^k \vec{h}_j\right) \tag{2.4}$$

It is important to note that the storage requirement grows by a factor of K, when we consider K independent attention heads.

2.1.1.3 GNN architecture

The output of the convolutional and attentional layers described in the previous subsections are the node features of a graph with the same structure (i.e., the same number of nodes and the same adjacency matrix) as the input graph. In the case of graph-classification tasks, such as image classification, graph processing layers are usually combined with *pooling* and *readout* operations, which generate compact graph representations from node information. Both operations have similar mechanisms (WU et al., 2021).

Graph pooling is used to reduce the size of the representation of the graph by down-sampling its nodes, analog to the pooling operation in CNNs. It is usually formulated as a clustering assignment problem, where the original graph's nodes are divided into groups based on their characteristics. Each cluster corresponds to a single node in the resulting representation (CHEN et al., 2022).

The *readout* operation is used to generate a *graph-level* representation from *node-level* representations. That is, the output of the operation is a single feature vector that represents the whole graph, while its inputs are the features of each of the nodes. Popular readout functions include global-mean pooling, global-max pooling, and global-sum pooling, where the outputs are, respectively, the feature-wise mean, maximum, and added values across the nodes of a graph (WU et al., 2021). This operation is essential to graph-classification tasks such as image classification. It is the mechanism that generates, from the outputs of the last Convolutional GNN layer of the network, the graph representation that is used as input for the multilayer perceptron (in most cases), that effectively classifies the graph. A typical architecture for graph-classification tasks can be seen in figure 4.2.

Given a graph G, with N nodes, each described by the feature matrix $\boldsymbol{x}_i, i \in$

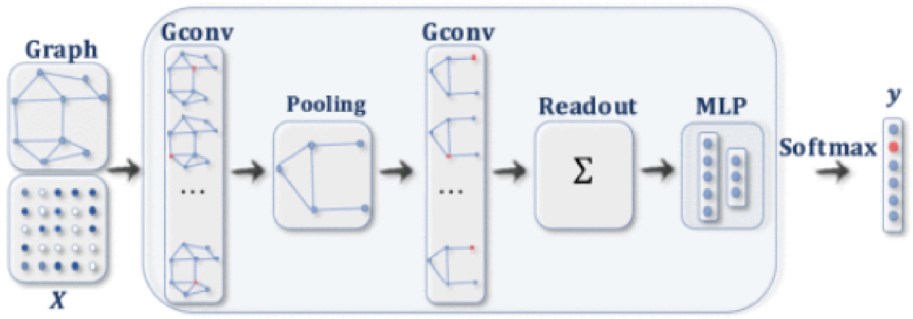
$[1..N]$, the global-mean pooling operation is defined by:

$$\boldsymbol{r} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i \tag{2.5}$$

The global-max pooling is defined by:

$$\boldsymbol{r} = \max_{i=1}^{N}\boldsymbol{x}_i \tag{2.6}$$

Figure 2.2: An example of Convolutional GNN architecture for graph-classification tasks, such as image classification, adapted from (WU et al., 2021). In the example, both pooling and readout operations are applied: pooling between convolutional (Gconv) layers to generate hierarchical representations and readout after the last convolutional layers to generate the graph-level representation.



Source: (WU et al., 2021).

2.1.2 Model evaluation

In this subsection, we discuss methods for evaluating an ML model's performance which are used in this work. This includes the evaluation metrics, explored in 2.1.2.1 and the cross-validation method, in 2.1.2.2.

2.1.2.1 Performance metrics

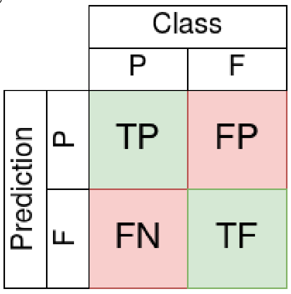
In order to assess an ML model's success, it is necessary to adopt a quantitative measure of its performance. The choice of performance metric is important as it should reflect the desired characteristics of the model (GOODFELLOW; BENGIO; COURVILLE, 2016). In this work, we adopt two main performance metrics: F1-measure and accuracy.

Accuracy is a common performance metric. It is the proportion of predictions the model makes correctly, as shown in equation 2.7. In the context of multi-label image classification, the prediction is the class to which the model attributes the maximum probability, given an input image. A prediction is correct if it matches the label annotated on the image (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{2.7}$$

However, the accuracy measure can be inadequate when it is desirable to weigh the importance of different kinds of mistakes. That is why precision and recall metrics are often used. In the context of binary classification problems (where predictions are either positive or negative), we define *false positives* as the positive predictions that were incorrect and *false negatives* as the false predictions that were incorrect. Similarly, *true positives* and *true negatives* are, respectively, the positive and the negative predictions that were correct. These concepts can be visualized in a *confusion matrix* as shown in figure 2.3. Thus, *precision* is the fraction of positive predictions that were correct (penalizing false positives), as shown in equation 2.8, while *recall* is the fraction of positive examples that were correctly classified (penalizing false negatives), shown in equation 2.9 (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figure 2.3: A confusion matrix, where the axes are the class predicted by the model and the actual class. Here, the letters P and F represent, respectively, True and False, and the letters P and N, Positive and Negative



Source: The Author

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

Precision and recall measures can be combined in the *F1-measure* that is also adopted in this work, as shown in equation 2.10 bellow (GOODFELLOW; BENGIO;

COURVILLE, 2016):

$$\textit{F1-measure} = \frac{2 \times \textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \tag{2.10}$$

In a multi-class classification setting, as is the case for this work, the F1-measure is computed by somehow aggregating the F1-measure computed for each individual class. In this work, we adopt the *macro average* of the F1-measure. Here, the final metric is given by the unweighted average of the individual measures (VANI; RAO, 2019).

2.1.2.2 Cross-validation

When assessing the capacity of an ML model, it is desirable to compute the performance measures (as described in the previous subsection) using data examples the model has never seen before - a test set of data. This way we can evaluate how well the model will perform in the real world. However, if the test set is not large enough, the statistical uncertainty of the computed performance metrics may make it difficult to claim that the result is truly representative of the model's capacity (GOODFELLOW; BENGIO; COURVILLE, 2016).

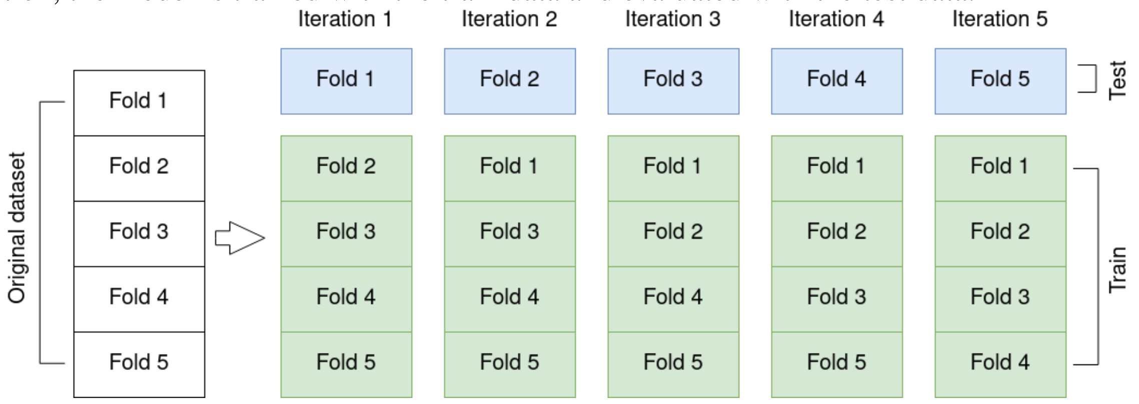
In this context, cross-validation methods aim to increase confidence in the evaluation of a model by repeating the training and testing computations on different subsets of the original dataset.

One of the most common cross-validation methods is the *k-fold cross-validation*. In this approach, the dataset is divided into k non-overlapping subsets and the model is trained and then tested across k trials. In the i^{th} trial, the i^{th} subset or fold of the dataset is used as the test portion of the data, while the model is trained on the remaining $i-1$ folds. The final result, then, is given by the average of the metrics computed in each of the trials. Figure 2.4 illustrates this process. In the *stratified* variant of the k-fold cross-validation process, each subset has the same class distribution as the original dataset.

2.2 Representing images as graphs

Molecules, social networks, citation networks, and physics systems are among the many kinds of data that have an intrinsic graph structure, that is, the elements of the problem (such as the atoms and the chemical bonds in the case of molecules) are immediately mapped to the nodes and connections of the graph. The same is not true

Figure 2.4: Ilustration of the k-folds cross-validation procedure, with K=5. In each iteration, the model is trained with the train data and evaluated with the test data.



Source: The Author

for traditional image data, which is structured as a 2D regular grid of pixels, with no relationships that can be readily interpreted as edges. Thus, in order to use images as inputs for GNNs and therefore possibly take better advantage of the topological structure and relationships in a scene, we must devise a mapping from image data to a graph that represents its information.

This method can be generalized to three fundamental steps:

1. Node generation: partitioning the image into nodes with an image-segmentation method;
2. Node characterization: generating the features that describe each node;
3. Edge building: connecting the graph's nodes into neighborhoods.

Most of the works that apply GNNs for classifying traditional image data have, as far as we are aware, followed the steps described above for transforming input images into graphs. In the following subsections, we go into further detail about what each step entails and describe some of the methods that are commonly used. Subsection 2.2.1 deals with image segmentation methods for node generation; Subsection 2.2.2 discusses node characterization; and Subsection 2.2.3 discusses edge-building methods.

2.2.1 Image segmentation

The simplest way to generate the nodes of a graph from an image is to consider each pixel as a node. However, this generates large graphs, and thus most works in the literature, to the best of our knowledge, initially apply an image segmentation technique to the input image and consider each of the resulting segments as a graph node instead. One of the most popular techniques for this use is based on the notion of superpixel.

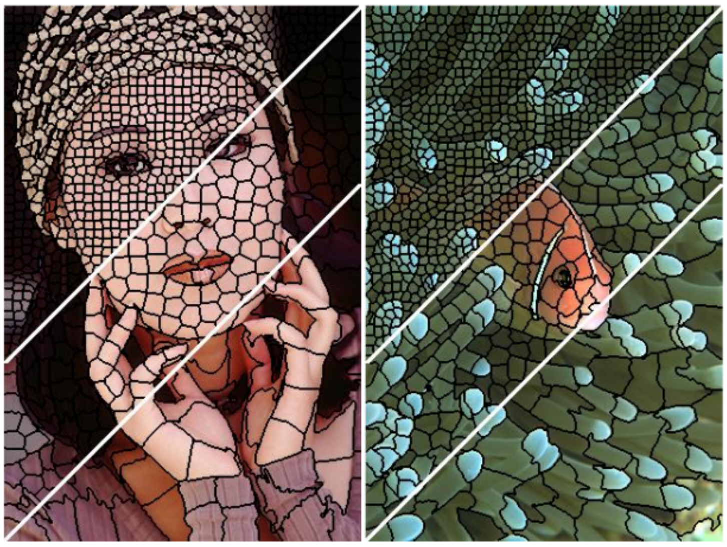
Superpixels group pixels that are visually similar, aiming to generate meaningful segments (i.e., a grouping that perceptually "belongs together") while reducing the computational requirements for the next steps of processing (STUTZ; HERMANS; LEIBE, 2018). They can be seen as a compromise between using each pixel individually (which has no information loss but can be computationally prohibitive) and using the results of object-oriented segmentation (the ideal scenario where each relevant object in a scene is represented as one node, yielding thus the smallest graphs with little semantic information loss, but which remains an open problem)(AVELAR et al., 2020). According to Stutz, Hermans and Leibe (2018), the requirements expected of algorithms for generating superpixels are:

- Partition: the superpixels define a partition of the image;
- Connection: superpixels represent a connected set of pixels (i.e., spatially adjacent);
- Boundary adherence: superpixels preserve image boundaries, with the definition of boundary varying from application to application;
- Compactness, regularity, and smoothness: superpixels should be compact, regularly spaced, and should have smooth boundaries when there are no structures in the image to adhere to;
- Controllable number of superpixels;
- Efficiency.

There are many state-of-the-art algorithms for extracting superpixels from images, such as SEEDS (BERGH et al., 2015), SNIC (ACHANTA; SüSSTRUNK, 2017), ETPS (YAO et al., 2015), and SLIC (ACHANTA et al., 2012). The SLIC algorithm is often recommended among those (STUTZ; HERMANS; LEIBE, 2018).

SLIC is an adaptation of k-means clustering approaches for the generation of superpixels. It takes as a parameter for superpixel generation k, the approximate number of superpixels to be generated. The number of iterations of the algorithm also has to be specified. The connectivity of the superpixels must be reinforced in post-processing: here it is necessary to establish the minimum element size (which determines if two segments to which the same label has been assigned will become their own separate entities or be absorbed by nearby superpixels). To further customize the behavior of SLIC, a compactness parameter can be specified, for balancing the importance of the color distance compared to the spatial distance of the pixels in the computing of the adapted k-means. This parameter dictates whether the superpixels will be more compact (when more weight is given to spatial distances) or more border-adherent (when the emphasis is on color distance) (ACHANTA et al., 2012).

Figure 2.5: Images segmented by the SLIC algorithm into superpixels of approximately 64, 256, and 1024 pixels.



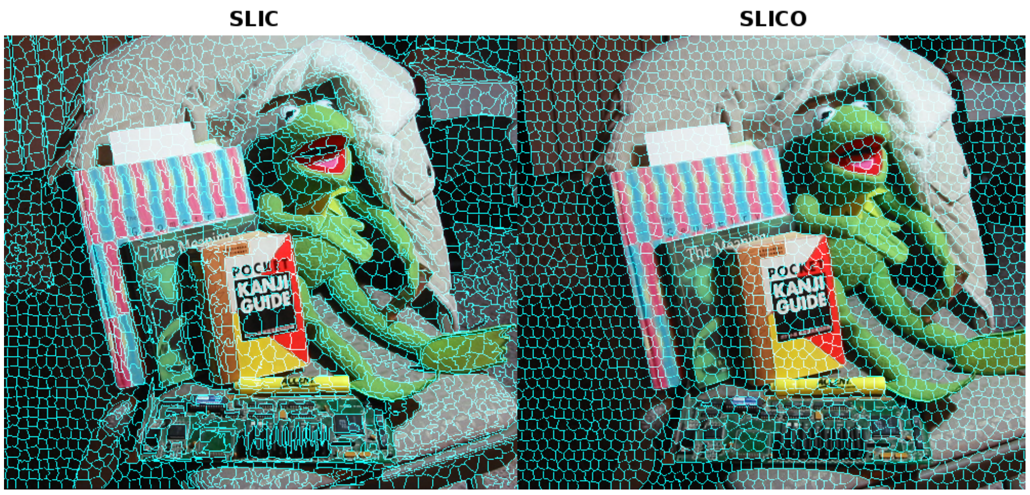
Source: (ACHANTA et al., 2012).

Several optimizations and variants of the original SLIC are available. We highlight SLICO[1], or zero-parameter SLIC, specified in the original SLIC paper (ACHANTA et al., 2012). In this approach, the compactness factor is dynamically calculated, adapting to different textures and color profiles of the images and, thus, often producing more stable segmentation results with respect to the effective number of pixels compared to k and to the shape of the segments. Notice that it has the advantage of eliminating the necessity of tuning the compactness parameter. Figure 2.6 presents an image segmented with both methods for comparison.

It is important to note that many implementations of the SLIC method and its variants, including the one adopted in this work, only admit superpixels composed of a minimum of 2 pixels, limiting thus the number of nodes that can be generated. The connectivity enforcement post-processing step can also influence the final number of superpixels. Consequently, it is important to observe that it is not guaranteed that the SLICO method will produce exactly the desired number of superpixels.

[1]We adopted the OpenCV implementation of SLICO[2]

Figure 2.6: Comparison of an image segmented with SLIC versus SLICO algorithms.



Adapted from (OpenCV. . . ,).

2.2.2 Node characterization

In order to be used as input for a GNN, each node generated from the results of the segmentation process must have at least one descriptive feature. These features should, ideally, summarize the meaningful information about the segments they represent and, in most cases that is achieved by statistics about the color and position of the pixels that compose the segment. Color information can be, for example, average color, standard deviation of color, and correlation matrices. Positional or spatial information can include the segment's geometric centroid (i.e., average position), standard deviation from the centroid, quantity of pixels, and other shape descriptors.

2.2.3 Edge building

Graphs are composed of two elements: nodes and the edges that connect these nodes. As is the case with the nodes, images have no intrinsic relationships that can be readily interpreted as edges, such as, e.g., the bonds in molecular data. Hence, an edge-building method must be devised when using image data with GNNs. Furthermore, this step has a significant impact on the behavior of the model, as the information each node considers in each convolutional step are the feature vectors of the nodes in its immediate neighborhood in GCNs (KIPF; WELLING, 2017) as well as in GATs (VELIčKOVIć et al., 2018), as discussed in section 2.1.1.

Most common in the literature, to the best of our knowledge, are edge-building methods based on K-Nearest Neighbors. In these approaches, each node will be connected to the K nodes (not including itself) that have the least distance to itself. The distance metric varies between works and segmentation methods. When using superpixels, in Monti et al. (2017) and many other studies that followed, consider solely the spatial distance when computing the distance metric. However, it is also possible to use a combination of spatial and color distances, to consider the entire feature space of the nodes, or to consider any combination of features.

Another approach adopted in works such as that of Avelar et al. (2020) is region adjacency graph (RAGs), where two nodes are connected when the corresponding segments are directly adjacent in the original image, that is, if they share a frontier. RAGs encode in their connections adjacency information that is not necessarily obtainable through KNN-based methods.

3 RELATED WORKS

In this Chapter, we discuss related works concerned with GNNs for superpixel image classification. Section 3.1 gives an overview of works that use GNNs for superpixel image classification, noting the design choices for the graphs. In section 3.2 we list works that perform similar analysis to ours.

3.1 Image classification with GNNs

As far as we are aware, Monti et al. (2017) were the first to apply GNNs to image classification tasks and their approach to constructing image-graphs inspired many works that followed. They introduced the MoNet framework for generalizing CNN architectures to graphs and manifolds. They also proposed a model that was applied to, among other tasks, image classification, using both uniform grids and SLIC superpixels as segmentation methods applied to the MNIST dataset. In the superpixel approach the graphs were fully connected, while in the grid approach, each node was connected to its immediate and diagonal neighbors, with grids yielding better results. No references are made regarding the method used for building node features.

More recently, Avelar et al. (2020) used Graph Attention Network (GAT) for superpixel image classification. Their method consists of segmenting the input image into superpixels using the SLIC method, extracting features (namely average color and centroid, although other options are also suggested) from them, and building region adjacency graphs. The resulting graph is then fed into the GAT. They concluded that GAT networks are not able to achieve the same performance achieved by more sophisticated models. They also point out the shortcomings of converting images to graphs due to its inherent information loss and suggest that richer node features may result in better performance, both aspects investigated in this work.

Long, Yan and Chen (2021) proposed the Hierarchical GNN (HGNN) with multiple GAT layers, aggregating each layer's output. The method was applied to superpixel image classification. The graph was built with SLIC superpixels as nodes, with average color and centroid as features. The edges were built with a K-Nearest Neighbors approach, using as distance metric the average distance of each color channel and spatial dimension. They experiment with different numbers of superpixels and posit that, generally, the more superpixels the better the performance of the models, but note that when the superpixels get too small the resulting graph contains more redundant information.

Linh and Youn (2021) proposed the Dynamic Superpixel Cloud GCN (DISCO-GCN) model, using GCN layers with edges generated dynamically before each layer. They use SLIC superpixel segmentation and build node features from the segments using only color information, adopting a separate encoding procedure to incorporate the positional information. They suggest that the main venue for progress is diminishing the information loss caused by the segmentation process via building richer feature vectors, our work reveals that that is not always the case.

Very recently, Cosma et al. (2023) tackled the same task using Felzenszwalb superpixels. They seek to compensate for the information loss of the segmentation by incorporating superpixel orientation and rotation, to favorable results. They also conclude that adding color variance information is beneficial, while the addition of the number of pixels is of little help: results confirmed in this work. They suggest that the dynamic edges proposed by Linh and Youn (2021) have a negative impact on their model, finding that it is beneficial to keep the static region adjancency graph.

3.2 Comparative studies of GNNs

Errica et al. (ERRICA et al., 2022) have compared different GNN architectures' (including GCNs) performance in the task of graph classification, drawing attention to the reproducibility problems present in the literature. They propose a rigorous method for model evaluation and comparison – highlighting the importance of using the same features and number of nodes – and a standardized and reproducible experimental setting. They are also able to establish, using a structure-agnostic baseline model, that not always are GNNs able to take advantage of the structural information in graphs.

Shchur et al. (SHCHUR et al., 2019) also point out limitations in the empirical evaluation process of GNN models, focusing on node classification tasks. They discuss the effects of train/test/validation dataset splits on performance, finding that with the same hyperparameter selection and training procedures simple GCNs may be able to outperform more sophisticated models.

Xu et al. (XU et al., 2019) provide a theoretical analysis of the representative power of different GNN models – in addition to proposing their own model, the Graph Isomorphism Network (GIN) – and experimentally compare them using different graph classification datasets, showing that, in most cases, more representative power implies greater accuracy.

As far as we know, the literature does not provide any systematic comparison of how different ways of building graphs impact the performance achieved by GCNs or GATs in image classification tasks.

4 EXPERIMENTS

In this Chapter, we discuss the experiments performed in this work. Our objective is to systematically analyze the impacts of different choices in the process of building graphs from images on the performance of GCN and GAT models. We have conducted three categories of experiments, one for each design dimension of the graph-building process: (i) node generation, (ii) feature building, and (iii) edge building. Each was repeated for multiple datasets and for both GCN and GAT architectures, which were trained with the same training procedure.

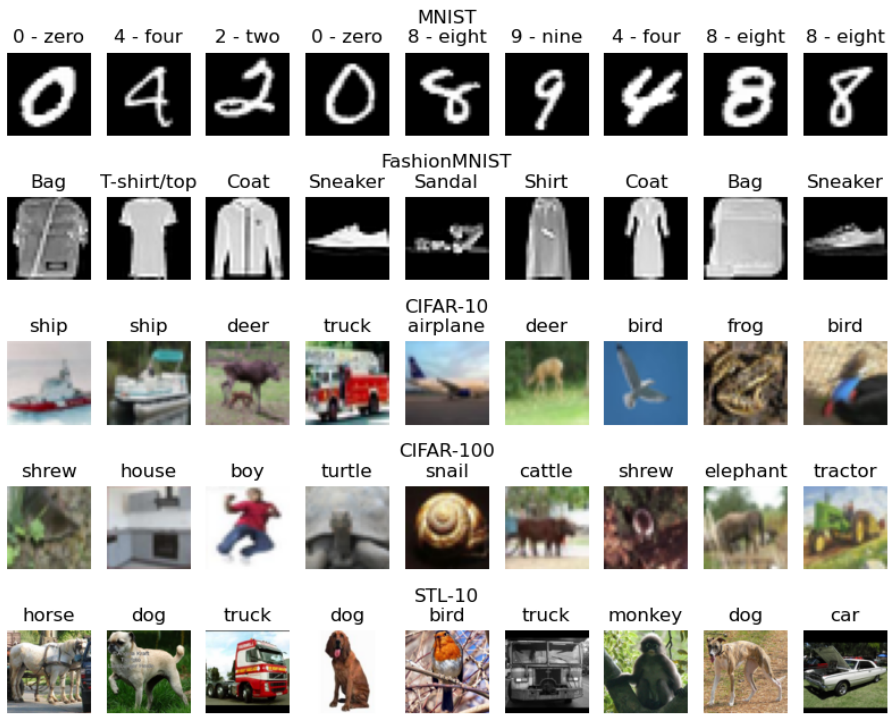
In section 4.1, we list the datasets used in the experiments. Section 4.2 details the architecture of both the GCN and GAT models, while section 4.4 describes the training method to which they were submitted. In section 4.4, we describe the methodology of each of the three experiments and present their results in section 4.5.

The source code developed for building graphs from images can be found in (BDI-UFRGS, 2023) In the project is also included the source code for the models and the scripts for the experiments described in this Chapter.

4.1 Datasets

Each experiment described in section 4.4 was carried out considering the following datasets: MNIST (LECUN et al., 1998), FashionMNIST (XIAO; RASUL; VOLLGRAF, 2017), CIFAR-10 and CIFAR-100 (KRIZHEVSKY, 2009), and the labeled subset of STL-10 (COATES; LEE; NG, 2011). These datasets were selected because they have different properties (regarding the colors, number of classes, number of samples, and image size) and they are common choices for benchmarking image classification approaches in the literature. All datasets are balanced (i.e., have the same number of images in each class) and, in each dataset, all images have homogeneous sizes. Other characteristics of the selected datasets are described in table 4.1 and a sample of images and their corresponding classes that can be found in each dataset is shown in figure 4.1

Figure 4.1: Samples of images and their classes of each dataset.



Source: The Author

Table 4.1: Dataset characteristics

Dataset	Images	Classes	Color	Area (px)
MNIST	70000	10	Greyscale	28x28
FashionMNIST	70000	10	Greyscale	28x28
CIFAR10	60000	10	Color	32x32
CIFAR100	60000	100	Color	32x32
STL10	13000	10	Color	96x96

Source: The Author

4.2 GNN architectures

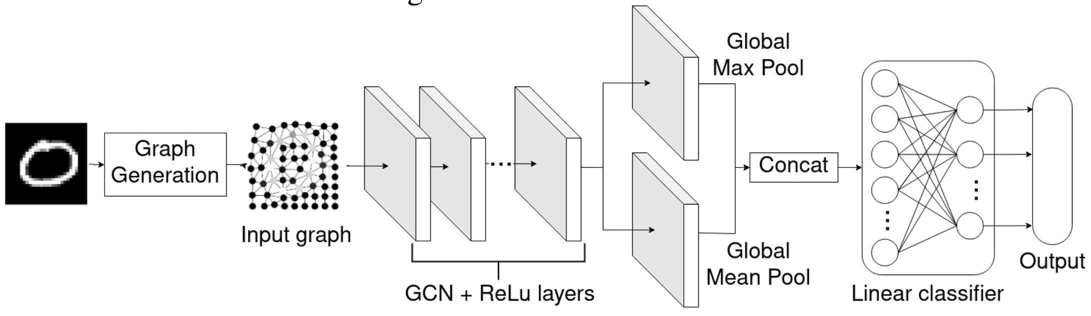
In this section, we describe the two GNN architectures used in the experiments described in the following section. Subsection 4.2.1 concerns the GCN model and the experimental process adopted to determine the model's architecture, likewise, subsection 4.2.2 concerns the GAT model.

4.2.1 GCN model

The GCN model used in this work consists of a group of sequential GCN layers, each followed by a ReLU activation layer. This sequence of layers is followed by a global-mean-pooling and a global-max-pooling operation, both resulting in vectors $r \in \mathbb{R}^{|F|}$, where $|F|$ is the number of features that describe each node. The two vectors resulting from these operations are concatenated and passed through a fully connected layer with linear activation, and the output is given by the softmax module that follows. The model is illustrated in figure 4.2.

The graph-generation module implements the three steps for translating images into graphs described in section 2.2.

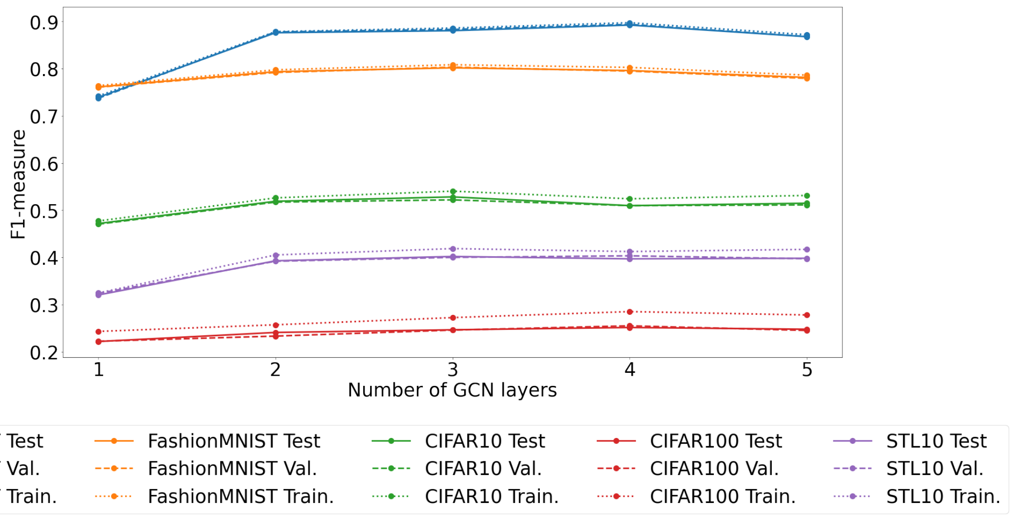
Figure 4.2: The GCN model.



Source: The Author

To determine the number of layers in the model, we evaluated the impact that different quantities have on the model's performance with a fixed graph-generation method. For each dataset, we considered models with 1, 2, 3, 4, and 5 GCN layers. We used region adjacency graphs (where nodes are connected considering all the adjacent neighbors of each superpixel in the original image) with approximately 75 nodes and average color, standard deviation of color, geometric centroid, and standard deviation from centroid as features.

Figure 4.3: Test, validation, and train F1-measure with macro average per number of GCN layers.



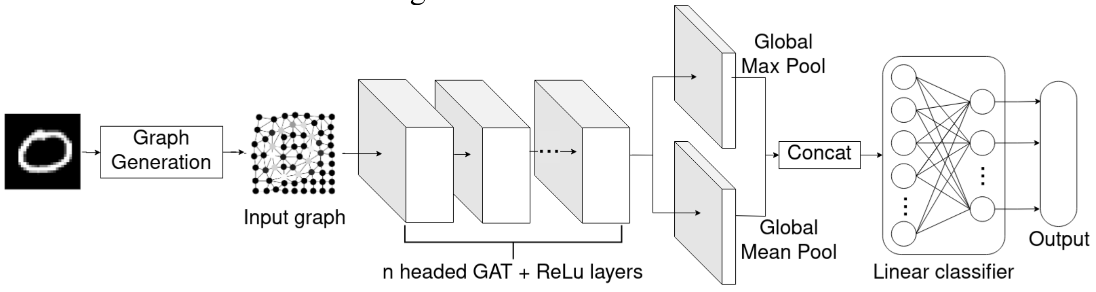
Source: The Author

As can be seen in figure 4.3, except for MNIST – the simplest dataset in the selection – raising the number of layers to four has, at best, no effect on the performance when compared with the three-layered model and, at worst, it decreases the performance (as is the case of CIFAR-10). We attribute this behavior to the well-known over-smoothing problem that is observed when stacking multiple GCN layers (LI; HAN; WU, 2018), with the node features becoming indistinguishable from each other. The decrease in performance could also be explained by over-fitting. However, this hypothesis can be discarded by observing the similarity of the train and test curves in figure 4.3 – as the over-fitting of a model is characterized by the decrease in performance on the test set while the training results continue to improve. However, in most datasets, raising the number of layers from two to three results in performance gains. Based on these results, we used three sequential GCN layers in the following experiments.

4.2.2 GAT model

Similarly to the GCN model described in the previous subsection, our GAT model is composed of a number of stacked multi-headed GAT layers, each followed by a ReLU activation layer. The convolutional layers are followed by global-max- and -mean-pooling operations and the two resulting vectors are concatenated and used as input by a fully-connected layer with linear activation followed by a softmax module that yields the final output. The model is pictured in figure 4.4.

Figure 4.4: The GAT model.



Source: The Author

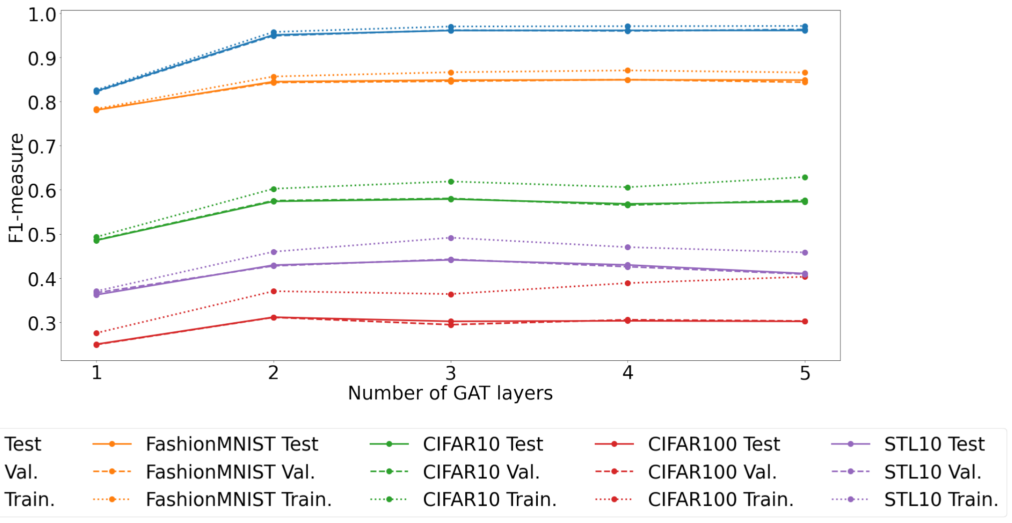
As in the previous section, we experimentally determined the best number of stacked GAT layers for a fixed graph generation method (which yielded region adjacency graphs of approximately 75 nodes, each characterized by the average and standard deviation of color, geometric centroid, and standard deviation from the centroid of the super-pixel). We considered possible quantities of 1, 2, 3, 4, and 5 stacked layers. As GATs also have a multi-headed attention mechanism, to determine the number of layers we initially fixed the number of attention heads to 2, following the work of Avelar et al. (2020).

Considering the results shown in the graph in figure 4.5, and abiding by the same criteria as for determining the number of stacked GCN layers, we adopted two GAT layers in the following experiments, as raising the number of layers from one to two results in better performances across all datasets, while increasing the number of layers up to three either does not significantly improve performance or decreases it.

To determine the number of attention heads for the two convolutional layers, we used the same graph generation method and trained models with the following number of heads (the same for each of the layers): 1, 2, 4, 8, and 16. The exponential growth of the values was inspired by the great range of architectures found in the literature.

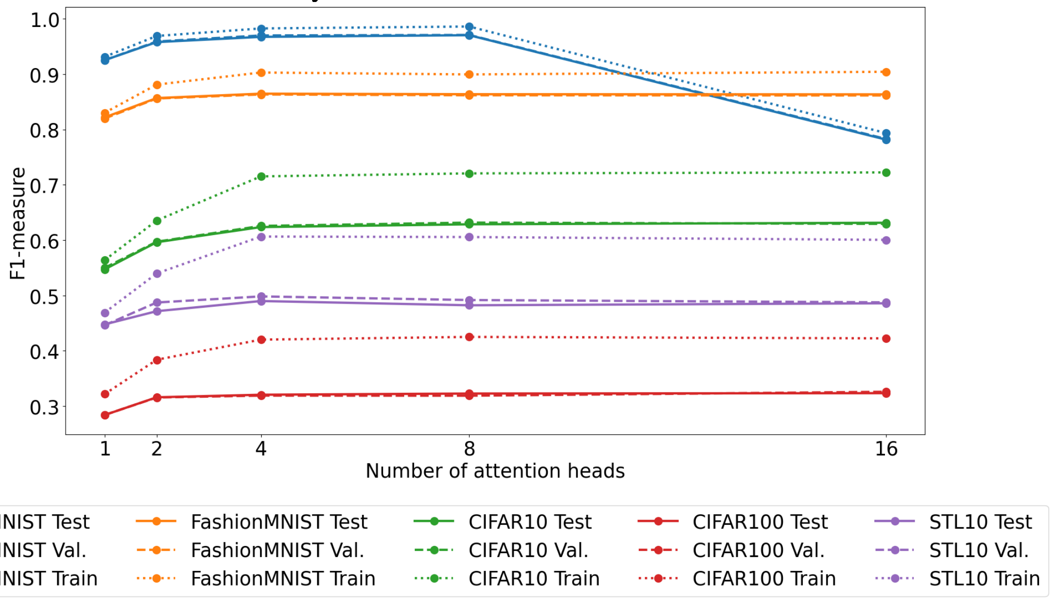
The results shown in the graph in figure 4.6 show an increase in performance across all datasets when increasing the number of attention heads up to four. When raising

Figure 4.5: Test, validation, and train F1-measure with macro average per number of GAT layers.



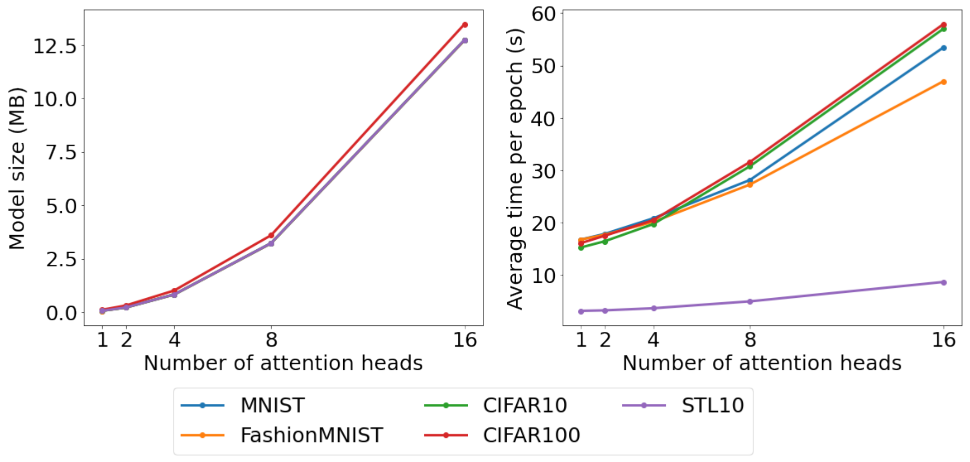
Source: The Author

Figure 4.6: Test, validation, and train F1-measure with macro average per number of attention heads in each GAT layer.



Source: The Author

Figure 4.7: From left to right, the size in MB of the model's parameters and the average time per training epoch with regards to the number of attention heads.



Source: The Author

the quantity to eight, for most datasets, there is still a slight improvement, however, the decrease in performance in MNIST is so significant that we selected four as the number of attention heads in the remainder of the experiments in this work.

Supporting this decision, as shown in figure 4.7, training time and memory requirements grow exponentially as a function of the number of attention heads in the model. Thus, the slight gains obtained by doubling the number of heads from four to eight (in figure 4.6), result in a much greater increase in the consumption of computational resources. We note that all datasets with 10 classes (MINIST, FashionMNIST, CIFAR-10, and STL-10) have a similar amount of parameters and thus are superposed under the STL-10 line in the graph that shows the memory requirements, while CIFAR-100, with 100 unique labels, requires a bigger output layer.

4.3 Training and evaluation methods

For the training procedure of the models presented in the previous section, we used the *Adam optimizer* (KINGMA; BA, 2014), with a *learning rate* of 0.001, no weight decay, and parameters β_1 and β_2 set to, respectively, 0.9 and 0.999, with *cross-entropy loss* as the loss function. Each model was trained for a fixed 100 training epochs, saving and reporting the model with the highest validation F1-measure during the training process. This best-performing model is used for evaluation on the test set. *Accuracy* and *macro f1-measure* where chosen as performance metrics.

We adopted a *stratified 5-fold cross-validation* procedure. In each iteration, one of the folds was used as test data while, of the union of the remaining folds, 10% was used as validation data and 90% as training data (that is, 20% test, 72% train, and 8% validation), maintaining class proportions in each fold. Every result reported henceforth is the mean result of the 5 folds.

4.4 Methodology

In this section, we detail the methodology of the experiments to assess the behavior of GCNs and GATs with regard to the three design dimensions of transforming images into graphs: (I) degree of segmentation, or the number of nodes in the graph, in subsection 4.4.1; (II) the choice of node features 4.4.2; (III) the edge-building method, in subsection 4.4.3.

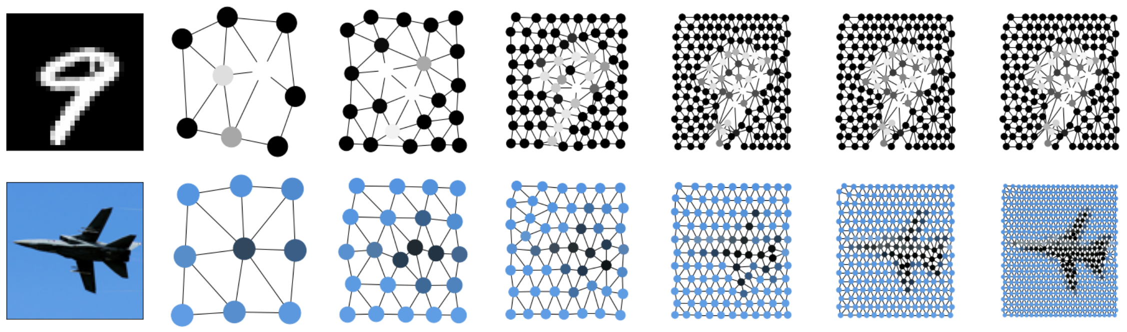
4.4.1 Evaluating the number of nodes

Commonly, when transforming images into graphs, the first step is to determine the mapping of the image's pixels to the graph's nodes by over-segmenting the image with an algorithm such as superpixels, so that every segment becomes a node. Thus, determining the number of nodes in the graph corresponds to determining the degree of segmentation of the image, that is, in superpixel-based approaches (adopted in this work), the number of nodes corresponds to the number of generated superpixels. In this particular work, this is done by choosing a value for the parameter N: the approximate number of generated superpixels in the SLICO algorithm.

In order to understand how the performance of our GCN and GAT models varies according to the number of superpixels, we trained and tested the models with the graphs generated for each of the following values for N: 10, 20, 50, 100, 200, and 400. This process was repeated for each considered dataset. Figure 4.8 shows examples of graphs generated using each of the N values. We highlight that the results for the MNIST dataset when $N = 100$, 200, and 400 are identical, as each superpixel has a minimum number of pixels, as further discussed in 2.2.1.

In this experiment, each node is described with the following features: average color (RGB or greyscale), standard deviation of color, geometric centroid, and standard deviation from centroid. These features were chosen because they are readily available from the superpixels and are commonly used and suggested in the literature, eg.: in the works of Avelar et al. (2020) and Cosma et al. (2023). Furthermore, the graph is a region adjacency graph.

Figure 4.8: Examples of input image and corresponding generated RAG for, from top to bottom, MNIST and STL-10 datasets. The graphs were built using SLICO with N set to, from left to right, 10, 20, 50, 100, 200, and 400.



Source: The Author

4.4.2 Evaluating node features

In this experiment, we evaluated how node characterization impacts the performances of GCN and GAT models. To do that, we considered the following possible features, all extracted from spatial and color (RGB or greyscale, depending on the particular dataset's characteristics) information of the pixels that compose each superpixel extracted from the original image by the SLICO method:

- Geometric centroid: average 2D pixel position in the original image;
- Standard deviation of pixel positions from the centroid;
- Number of pixels: total number of pixels, or pixel density, in the superpixel;
- Average RGB color: average R, G, and B values in colorful datasets or average greyscale value in greyscale datasets;
- Standard deviation from average color: standard deviation of R, G, and B mean values in color datasets or of greyscale mean value in greyscale datasets;
- Average HSV color: only used in color datasets (i.e. CIFAR-10, CIFAR-100, and STL-10), average values in HSV color space;
- Standard deviation from average HSV color: only used in colorful datasets, the standard deviation of values in HSV color space.

Our method consisted of selecting an initial baseline feature vector containing only one feature and then progressively expanding the baseline by adding new features, analyzing how each increment impacted the model's performance. We selected as the baseline feature the average color (RGB in colorful datasets, greyscale otherwise), which is the feature most commonly used in the literature, to the best of our knowledge. That is a one-dimensional feature vector in greyscale datasets and a three-dimensional vector in colorful datasets. The order in which the remaining features were added was, from first to last: geometric centroid, standard deviation of color, standard deviation of centroid, and number of pixels. For colorful datasets were also added, in that order: average HSV color, and standard deviation of HSV color. We adopted this approach in order to avoid the combinational explosion of experiments that would happen had we taken into consideration every possible feature combination. The order in which the features were added was determined by the frequency with which the feature appeared in our review of the literature.

In this experiment, we used the SLICO algorithm for segmentation, with parameter n, the desired number of superpixels, fixed at 75. For defining the edges of the resulting graph, we adopted region adjacency graphs.

4.4.3 Evaluating edge-building methods

In this experiment, we analyze different approaches for building the graph's edges and their impact on the model's performance. The three different methods – the most common in the literature, to the best of our knowledge – chosen for this experiment were, as further explained in subsection 2.2.3:

- Region adjacency graphs (RAG);
- K-Nearest Neighbors with spatial distance (KNN-Spatial);
- K-Nearest Neighbors with combined spatial and color distances (KNN-Combined).

We did not perform experiments with KNN graphs that use solely the color distance. This is because this method was not encountered in the literature and, as will be further exposed in the following sections, the spatial information is vital to the classification task.

In region adjacency graphs, there is an edge between two nodes if the superpixels they represent are directly adjacent. That is, there is at least one pair of pixels i and j, with coordinates (x_i, y_i) and (x_j, y_j), each one belonging to one of the superpixels, that satisfy the following condition:

$$|x_i - x_j| + |y_i - y_j| = 1 \tag{4.1}$$

Both KNN-Spatial and KNN-Combined methods assign edges between each node and its k nearest neighbors, not including the node itself. The difference between the two lies in the distance function: KNN-Spatial uses the spatial distance, i.e. the distance between two superpixels' geometric centroids, while KNN-Combined combines the spatial distance and the distance between average color values.

For two superpixels s_i and s_j, with geometric centroids (x_i, y_i) and (x_j, y_j), and average color values (r_i, g_i, b_i) and (r_j, g_j, b_j) for RGB color datasets and l_i and l_j for greyscale datasets, the spatial distance between s_i and s_j is given by:

$$d_{\text{spatial}}(s_i, s_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4.2}$$

The combined color and spatial distance is defined as, for color datasets:

$$d_{\text{combined}}(s_i, s_j) = \sqrt{d'_{\text{spatial}}(s_i, s_j) + d'_{\text{color}}(s_i, s_j)} \tag{4.3}$$

Where $d'_{\text{spatial}}(s_i, s_j)$ and $d'_{\text{color}}(s_i, s_j)$ are:

$$d'_{\text{spatial}}(s_i, s_j) = \frac{(x_i - x_j)^2 + (y_i - y_j)^2}{2} \tag{4.4}$$

$$d'_{\text{color}}(s_i, s_j) = \frac{(r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2}{3} \tag{4.5}$$

As for greyscale datasets, the combined color and spatial distance is given by:

$$d_{\text{combined}}(s_i, s_j) = \sqrt{d'_{\text{spatial}}(s_i, s_j) + d'_{\text{grey}}(s_i, s_j)} \tag{4.6}$$

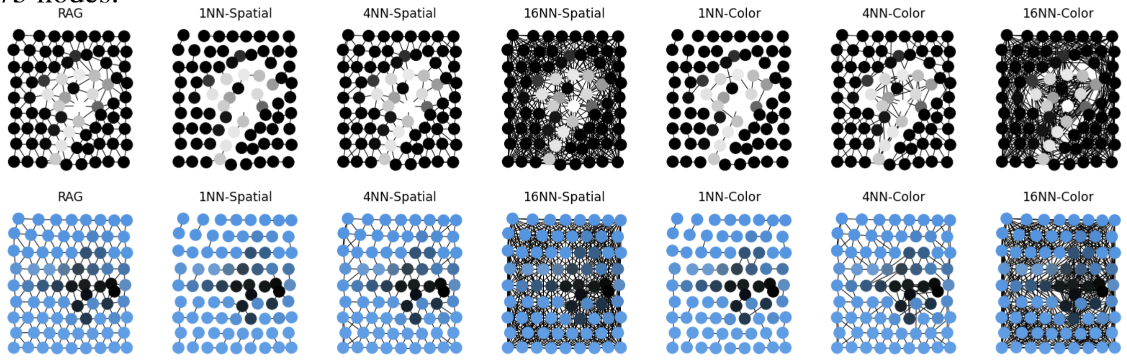
With $d'_{\text{grey}}(s_i, s_j)$ defined as:

$$d'_{\text{grey}} = (l_i - l_j)^2 \tag{4.7}$$

In this experiment, we adopted the following values for the parameter k – that determines the node degree – in the KNN-Spatial and KNN-Combined methods: 1, 2, 4, 8, and 16. It is important to note that self-loops are always added in the training process and that the node itself is not considered when computing its distances from the nodes in the graph. Figure 4.9 shows examples of graphs produced with the different methods and parameter valuations, omitting said self-loops.

We used as node features average color, centroid, standard deviation of color, and standard deviation of centroid. The segmentation method used was SLICO with a fixed desired number of nodes of 75.

Figure 4.9: Examples of selected graphs produced in the experiment from the same original images shown in figure 4.8 for MNIST and STL-10 datasets, each with approximately 75 nodes.



Source: The Author

4.5 Results

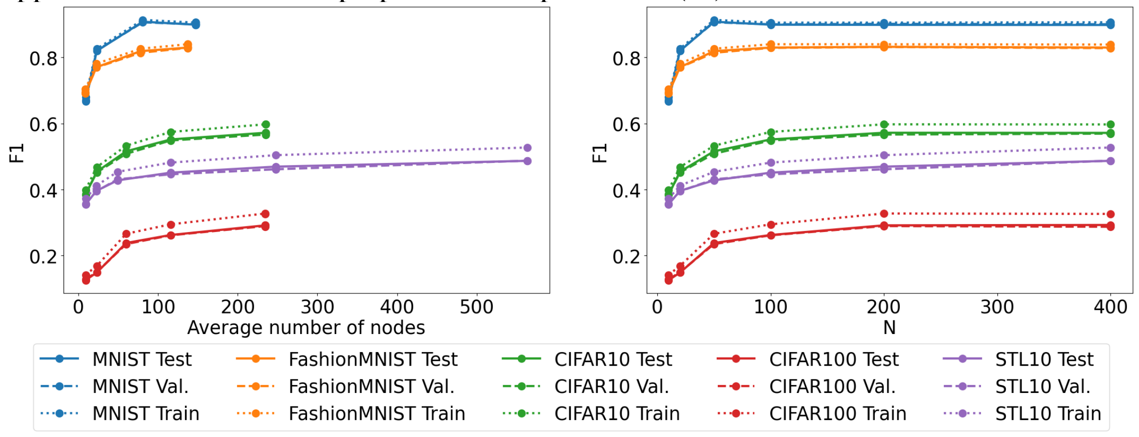
In this section, we present the results obtained from the experiments described in section 4.4. In subsection 4.5.1 we discuss the impacts of the variation of the number of nodes in the graphs, for both GCN and GAT models. Likewise, subsection 4.5.2 delves into the evaluation of the choices of node features and subsection 4.5.3 of the choice of edge-building method. We aggregate the best choice for each of the experiments, train GCN and GAT models on graphs built with these characteristics, and present the results in subsection 4.5.4. Finally, in subsection 4.5.5 we compare GCN and GAT models and the CNN-based AlexNet.

4.5.1 Number of nodes

In this experiment, we analyze the effects of the variation of the number of nodes in the graph or, equivalently, of the degree of segmentation of the original image on the model's performance. The results are shown in the graphs in figures 4.10 for the GCN model and 4.11 for the GAT model. The charts show how the final train, test, and validation F1-measure change as the number of nodes increases. They show both the average number of nodes for the graphs in the datasets and the value of the N parameter of the SLIC algorithm (that is associated with the approximate number of superpixels) with which the graphs were generated.

We highlight that the actual number of superpixels generated by the SLIC algorithm may differ from the algorithm's initial parameter. This happens significantly with the STL-10 dataset: when N is set to 400 the algorithm produces an average of 563 superpixels per graph. We believe that this is due to the connectivity-enforcement step of the post-processing of the SLIC algorithm. SLIC does not guarantee that the pixels with the same superpixel label will be connected (i.e., neighbors). In the connectivity-enforcement step, groups of pixels that are assigned to the same superpixel but are not connected are either absorbed by neighboring superpixels or become two separate segments depending on the clusters' sizes. In the second case, the number of superpixels in the image can become greater than the initially specified parameter.

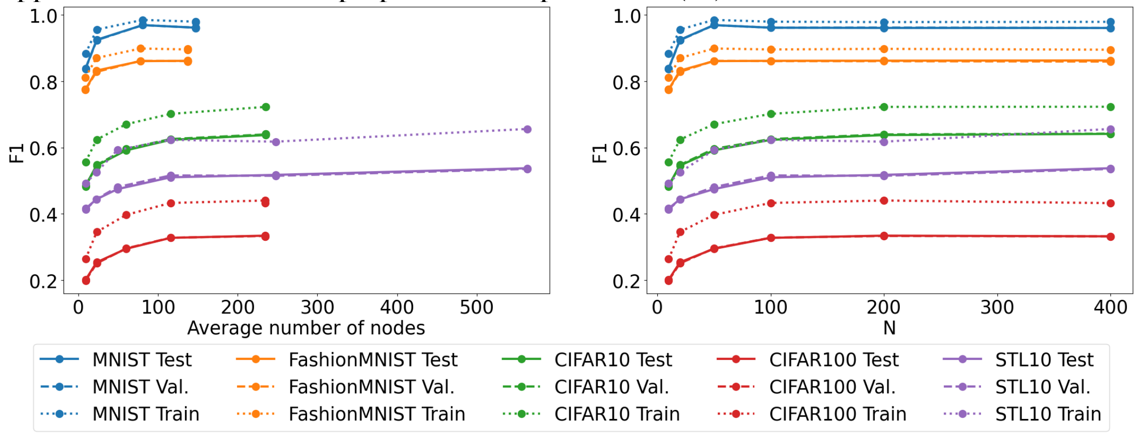
Figure 4.10: The GCN model's test, validation, and train F1-measure with macro average for each dataset w.r.t., from left to right, the average number of nodes and the value of the approximate number of superpixels SLIC parameter (N).



Source: The Author

We observe that the performance does increase for our GCN and GAT models as the number of nodes increases. This can be explained by the fact that the information loss intrinsic to the segmentation is lessened when the segments are small and are thus capable of representing finer detail of the original image. However, the increase in performance tends to follow a logarithmic curve: increasing less as the number of superpixels approaches the total number of pixels in the image.

Figure 4.11: The GAT model's test, validation, and train F1-measure with macro average for each dataset w.r.t., from left to right, the average number of nodes and the value of the approximate number of superpixels SLIC parameter (N).



Source: The Author

This can be explained by the same premise that justifies superpixels: that small groups of nearby pixels tend to be redundant and thus little information is lost by representing them as a single segment. Which segments can be used in the following processing steps (i.e., training the model) with small performance loss and reduced requirements of computational power.

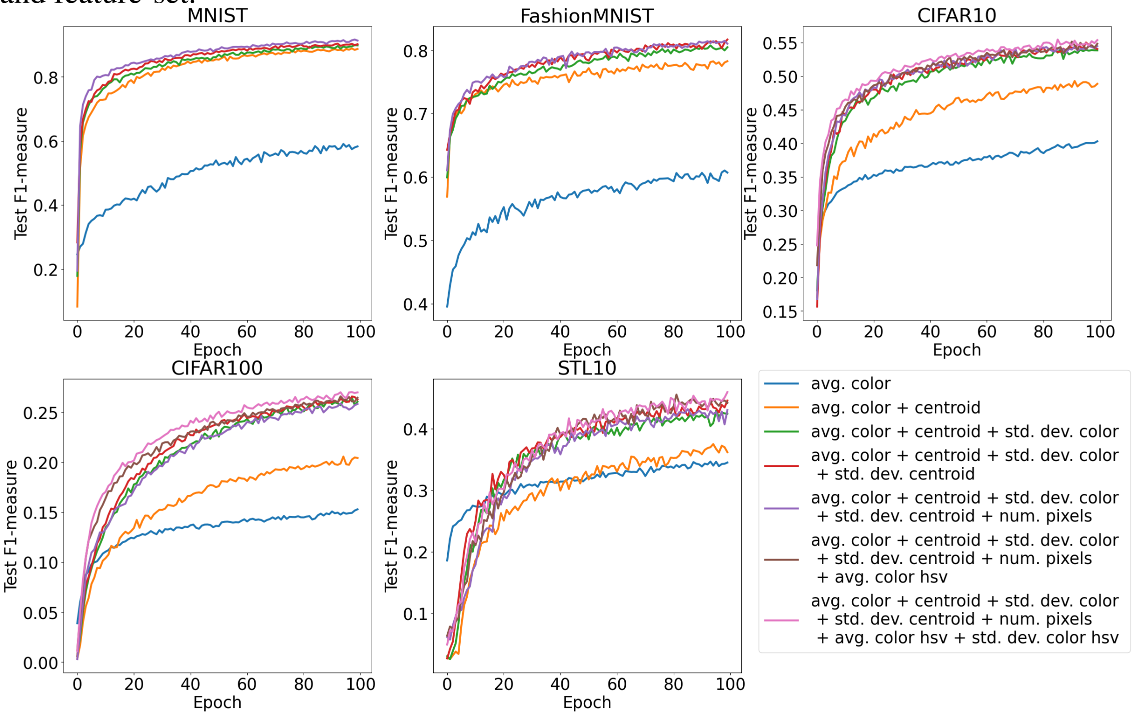
Meanwhile, the number of edges and size of the feature vectors generated for each graph grow linearly with respect to the total number of nodes: affecting thus the memory requirements for the dataset and the training time of the model. This behavior emphasizes a trade-off that must be observed between achievable performance (the predictive capacity of the model) and computational requirements.

4.5.2 Node features

In this experiment, we analyze the impact of different choices of features that describe the nodes of the image graph. Figures 4.12 and 4.13 show the evolution of the macro F1-measure of the test set over the 100 training epochs for, respectively, the GCN and the GAT model. The figures also list in full the sets of features used in each iteration. Figures 4.14 and 4.15 contain the bar plots showing the final F1-measure achieved with each feature added to the cumulative baseline for, respectively, the GCN and the GAT model.

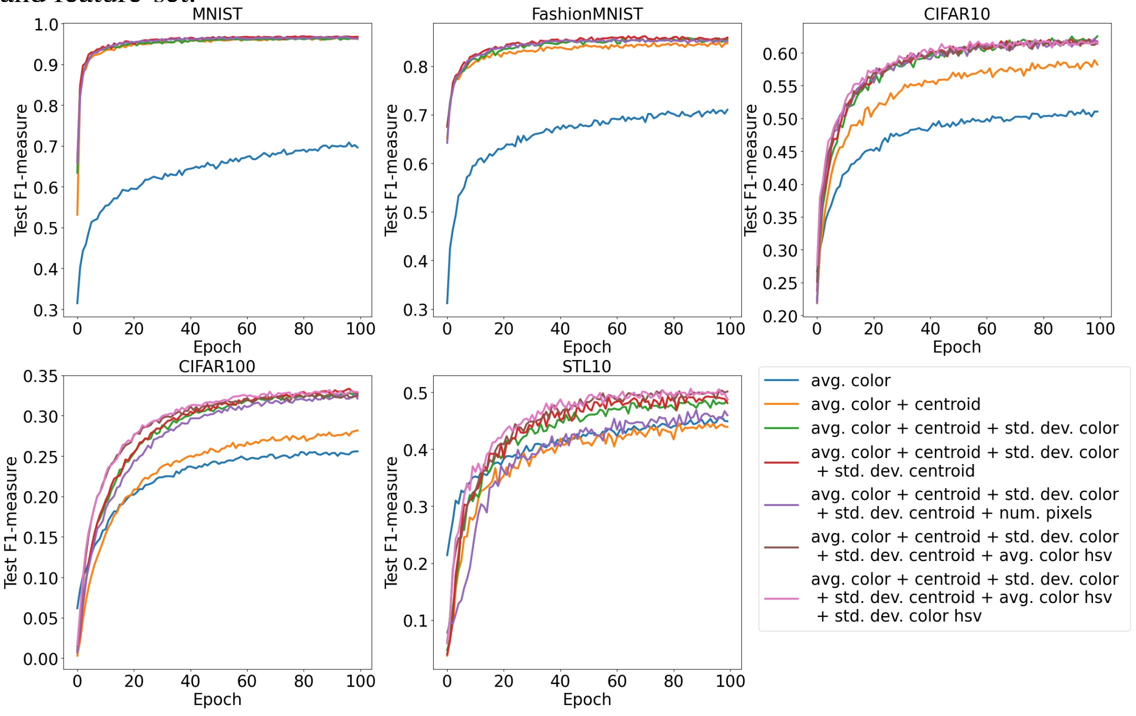
Our results show that, for both models, the most significant performance gain is

Figure 4.12: GCN model's test F1-measure along the 100 training epochs for each dataset and feature-set.



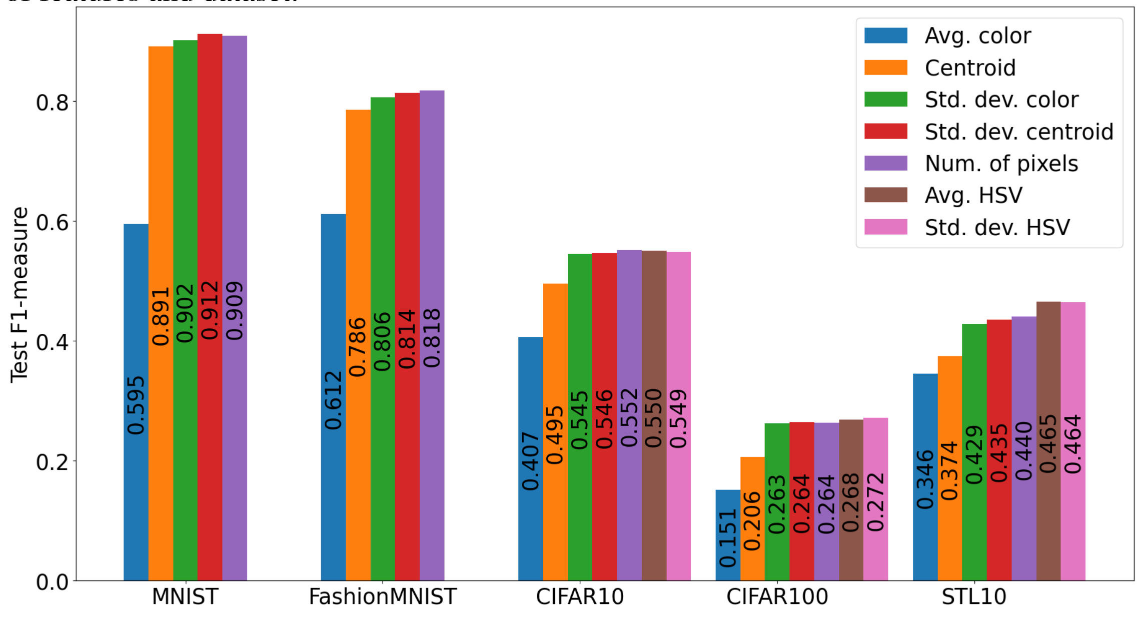
Source: The Author

Figure 4.13: GAT model's test F1-measure along the 100 training epochs for each dataset and feature-set.



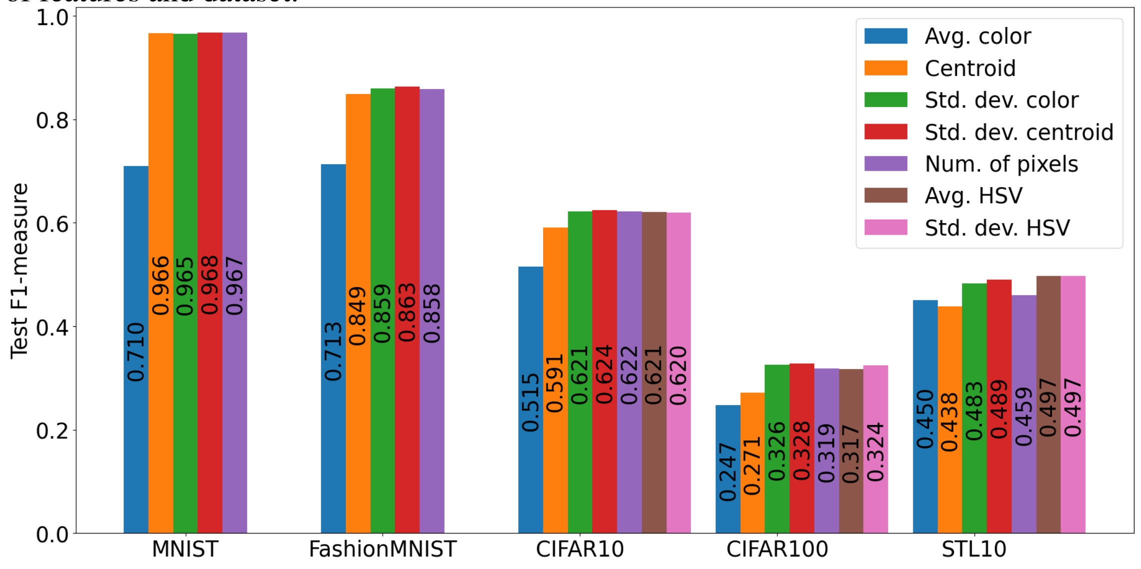
Source: The Author

Figure 4.14: Final test F1-measure achieved by our GCN model with each cumulative set of features and dataset.



Source: The Author

Figure 4.15: Final test F1-measure achieved by our GAT model with each cumulative set of features and dataset.



Source: The Author

obtained when adding the spatial information from the centroid to the baseline RGB/greyscale color information, with the notable exception of the GAT model trained with the STL-10 dataset. STL-10 is the dataset that has the least amount of data and the biggest images in our selection (see table 4.1), and we can see in figure 4.13 that it had the most unstable results during training.

Moreover, a consistent improvement is seen when adding the standard deviation of the RGB/greyscale color, especially with RGB datasets. The inclusion of the standard deviation from the centroid has also resulted in performance improvements for most datasets.

For the GCN model we conclude that, in general, considering the features analysed in this work, a richer choice of features relating to spatial and color information yields better results. The use of the number of pixels in the superpixel and of features in different color spaces is beneficial in most cases.

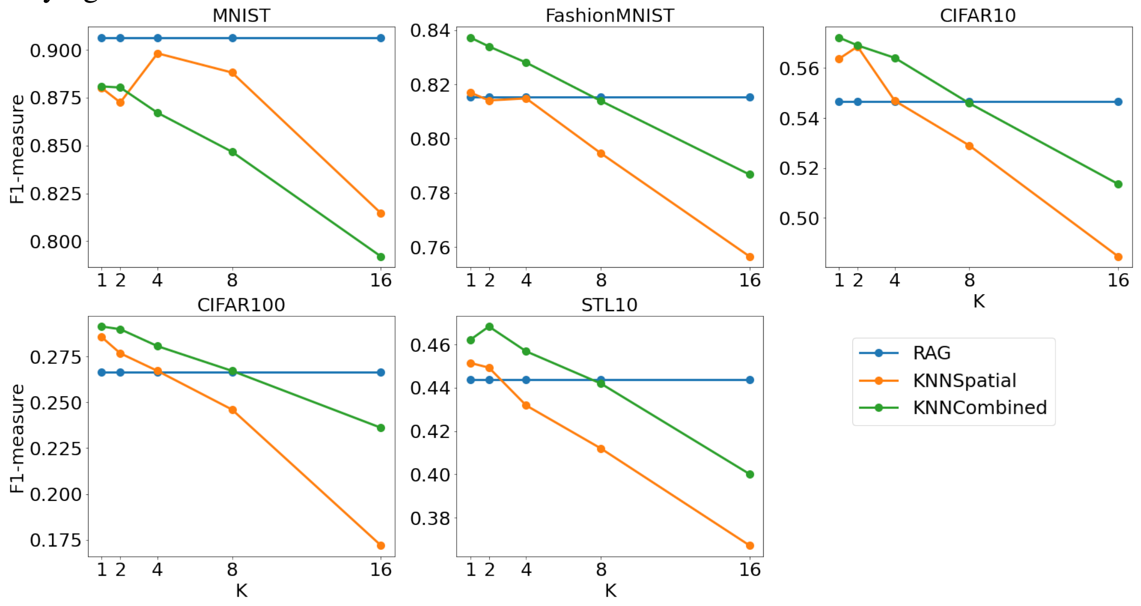
In contrast, for the GAT model, the use of the number of pixel information has made performance worse in every case (we note that this means that the feature is therefore not included in the following feature sets, e.g.: the next feature set is composed of average and standard deviation of color and position and of the average color in the HSV space, as can be seen in figure 4.13). The use of average color and standard deviation of color in the HSV space had varying effects: with the CIFAR-10 and CIFAR-100 datasets, the F1-measure decreased in comparison to the previous baseline (which included the standard deviation from the centroid but not the number of pixels), meanwhile, the F1-measure obtained with the STL-10 dataset increased. Therefore we cannot conclude that, as for the GCN, richer feature sets are always better – except for the average and standard deviation of RGB/greyscale color and centroid. Instead, we observe that features affect differently each dataset and must be chosen carefully to achieve optimal results.

4.5.3 Edge-building methods

In this Section, we analyse the effect of different edge-building methods on the GCN and GAT models' performances. Figures 4.16 and 4.17 present the test F1-measure achieved for each method and each value of k (the number of neighbors) for, respectively, the GCN and GAT models.

RAG can have a variable number of neighbors. Therefore, to aid in the comparison between the different graph types, table 4.2 presents the average node degree with

Figure 4.16: GCN model's final test F1-measure with each edge-building method, with a varying k.



Source: The Author

standard deviation for the RAGs generated for each dataset. In general, the average degree is close to 5 in all datasets.

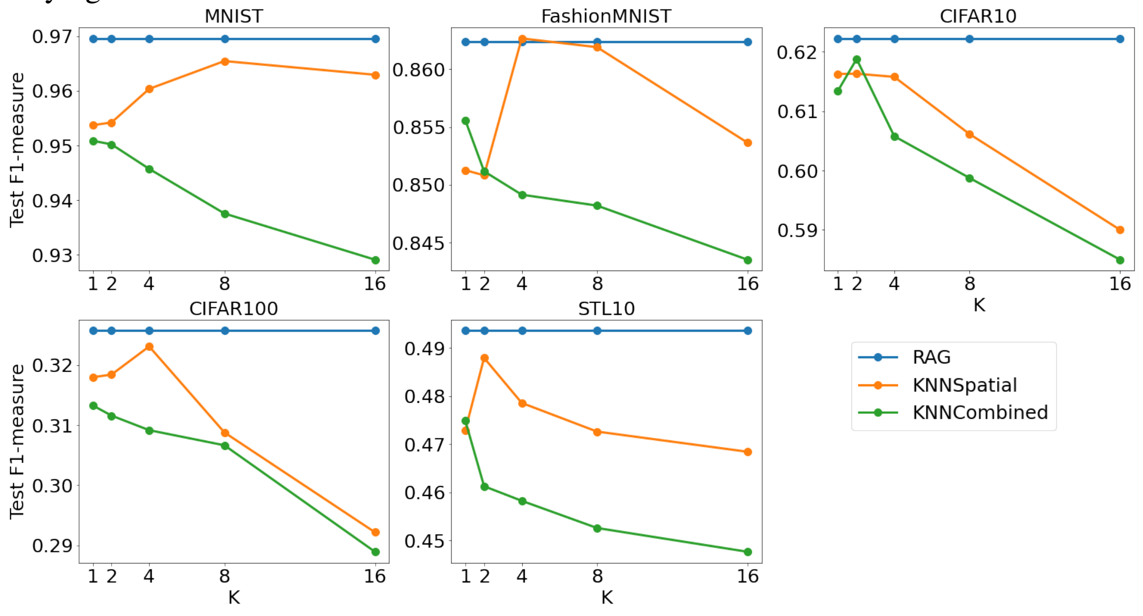
Table 4.2: The average node degree and standard deviation of node degree of RAGs for each dataset.

Dataset	Avg. node degree	Std. dev.
MNIST	5.0	±0.079
Fashion-MNIST	5.0	±0.087
CIFAR-10	5.3	±0.016
CIFAR-100	5.3	±0.02
STL-10	5.1	±0.062

In the GCN model, the best performance is achieved with KNNCombined graphs, with small neighborhoods (i.e., values of K), with the performance decreasing steadily as K increases. And exception to this pattern is MNIST – the simplest dataset in our selection – where RAGs, followed by KNNSpatial graphs, resulted in the best performances. This suggests that the GCN layers are most helpful when the information flows only through uniform regions (as can be seen in figure 4.9, small neighborhoods with combined distance result in graphs where only close, similar regions are connected).

We note that GCN layers – which can be described as a special form of Laplacian smoothing – are well known to face over-smoothing of the input features (LI; HAN; WU, 2018). Keeping neighborhoods restricted to similar regions avoids the issue of having different areas of the image-graph converge to indistinguishable representations. However, with this graph configuration, the GCN layers cannot leverage adjacency information (present in RAGs) or the context in which the region is inserted.

Figure 4.17: GAT model's final test F1-measure with each edge-building method, with a varying k.

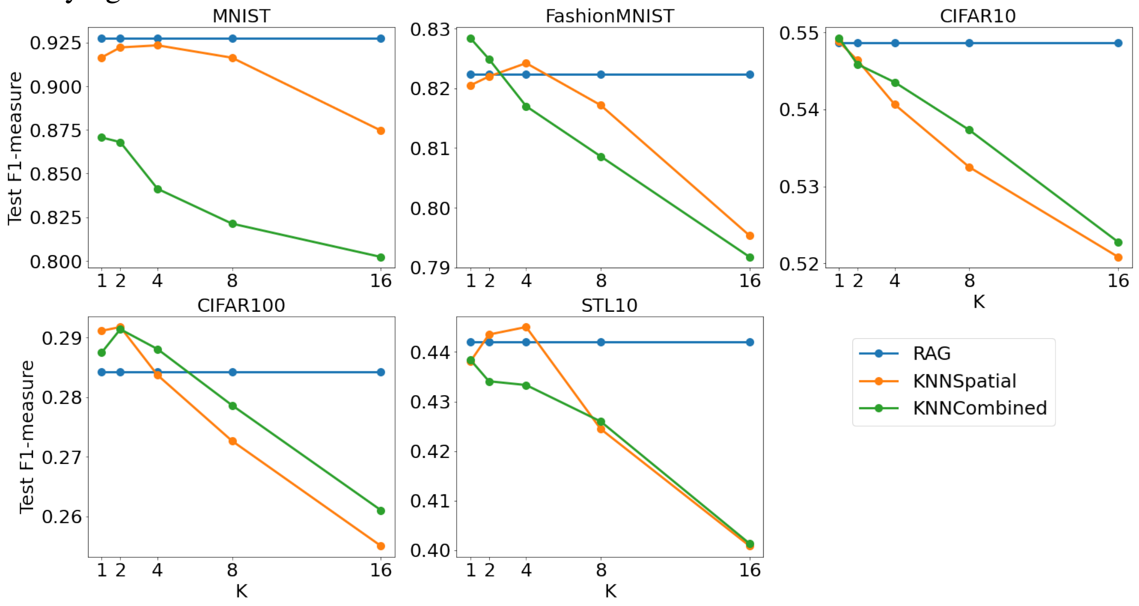


Source: The Author

The GAT model presents a different behavior. The best performance is achieved with RAGs with all datasets except for FashionMNIST, where the model performed better with KNNSpatial graphs with four neighbors, by a small margin. The KNNCombined method, which was the best choice for the GCN model, has yielded the poorest performance for the GAT model. This can be explained by the fact that the attention mechanism in GAT layers allows the model to attribute varying importance to a node's neighbors (e.g.: only pay attention to nodes with similar color and ignore contrasting regions or vice-versa). In this manner, it is more suited to leverage adjacency and context information that is most present in RAGs and is diminished in graphs built with the KNNCombined method.

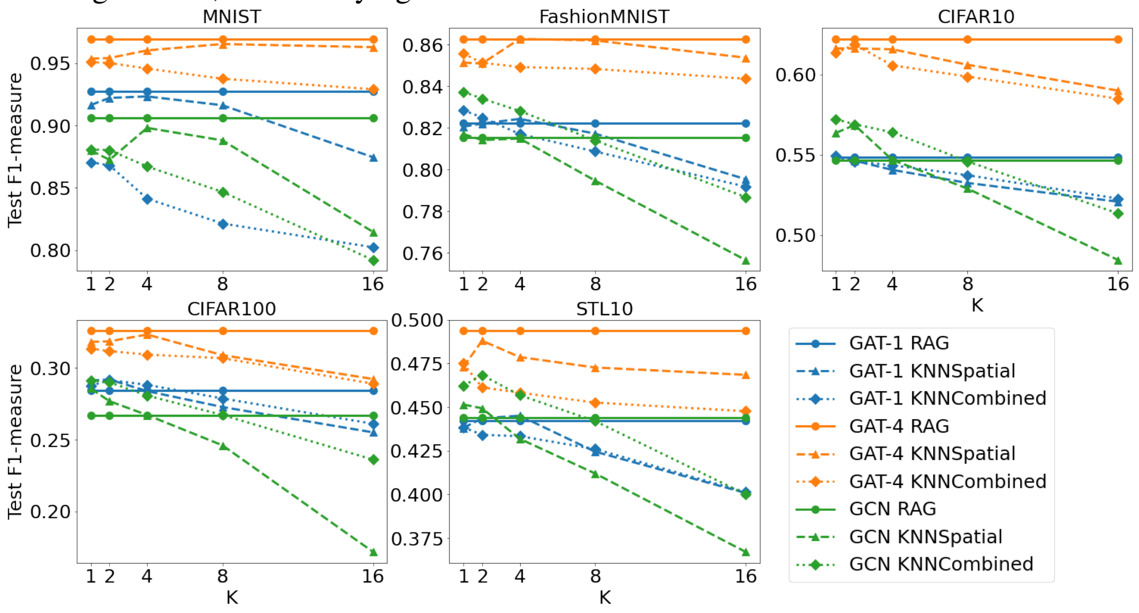
To support this claim, we have repeated the experiment for a GAT model defined exactly as described in subsection 4.2.2 but with only one attention head. Henceforth, we call this model the GAT-1 model. The results obtained – expressed in the graph in figure 4.18 – show a behavior that is more aligned with the GCN model's. The model trained with RAGs achieved the best performance only with the MNIST dataset, and the performances obtained with KNNSpatial and KNNCombined graphs are similar in most cases.

Figure 4.18: GAT-1 model's final test F1-measure with each edge-building method, with a varying k.



Source: The Author

Figure 4.19: GCN, GAT-4 and GAT-1 model's final test F1-measure with each edge-building method, with a varying k.



Source: The Author

In figure 4.19 we show simultaneously the results achieved with the GCN model, the original four-headed GAT model (referred to as GAT-4), and the GAT-1 model. We note that the four-headed GAT model achieves generally better performances, while the GCN and one-headed GAT models tend to be similar.

4.5.4 Combinating best choices

In this section, we select the best-performing choices for building graphs in each of the experiments described in section 4.4 and train GCN and GAT models on graphs built with these characteristics. We consider the best choice for each of the design dimensions the one that yielded the maximum validation F1-measure. These choices are detailed for each model and each dataset in table 4.3. Here, we use the following abbreviations for selected feature sets:

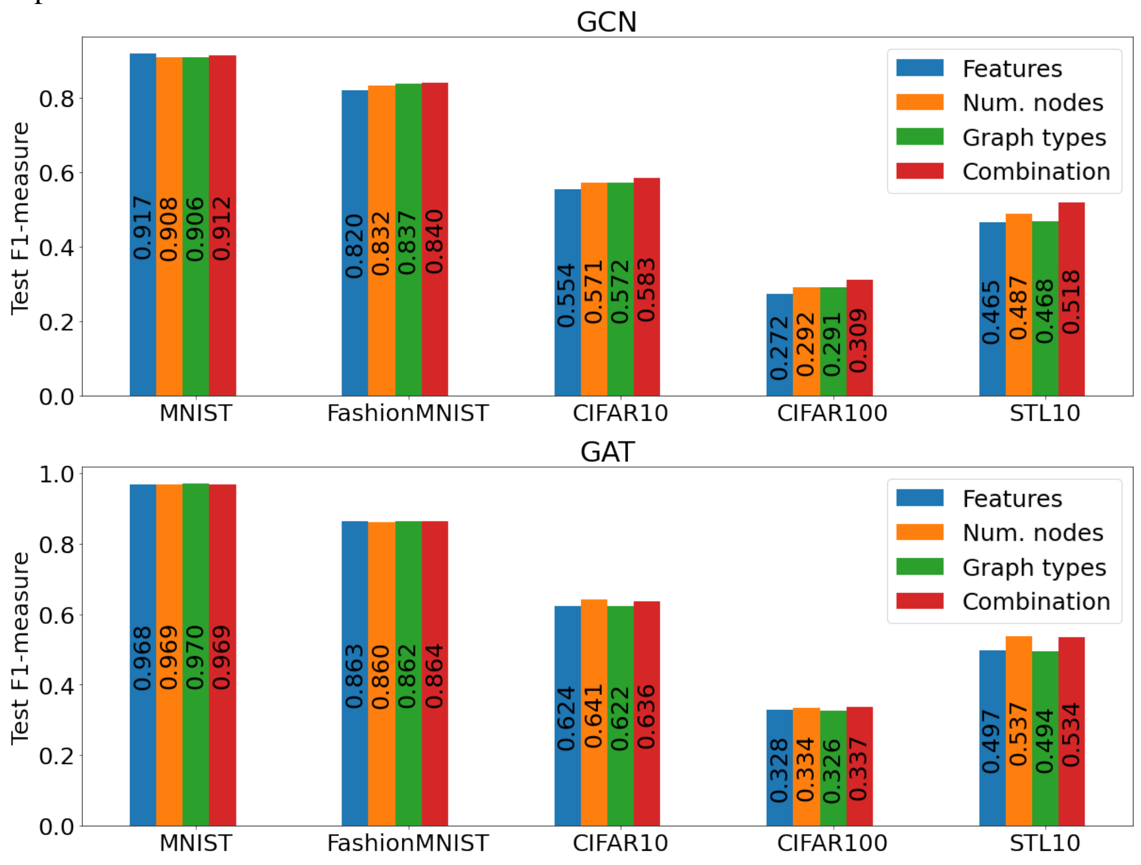
- $F1 = \{$*avg. color, std. dev. of color, centroid, std. dev. from centroid*$\}$
- $F2 = F1 \cup \{$*avg. HSV color*$\}$
- $F3 = F2 \cup \{$*std. dev. of HSV color*$\}$

Table 4.3: The best-performing choice of number of nodes, feature-set, and graph type for each model and dataset.

M	Dataset	n	Features	Graph type
GCN	MNIST	50	F1	RAG
	Fashion-MNIST	200	F1	1NNCombined
	CIFAR-10	400	F3	1NNCombined
	CIFAR-100	200	F3	1NNCombined
	STL-10	400	F3	2NNCombined
GAT	MNIST	50	F1	RAG
	Fashion-MNIST	50	F1	RAG
	CIFAR-10	400	F1	RAG
	CIFAR-100	200	F1	RAG
	STL-10	400	F2	RAG

Figure 4.20 shows the final test F1-measure for the best choice in each experiment discussed above and for the model trained with the graphs built with the best configuration for each dimension. In every case, this combination of best choices has resulted in models that either outperform the rest (e.g.: STL10 with the GCN model) or produce results within the variation range of the best-performing configuration.

Figure 4.20: Test F1-measure for the best-performing choice for building graphs in each experiment and for the combination of these choices for the GCN and the GAT model.



Source: The Author

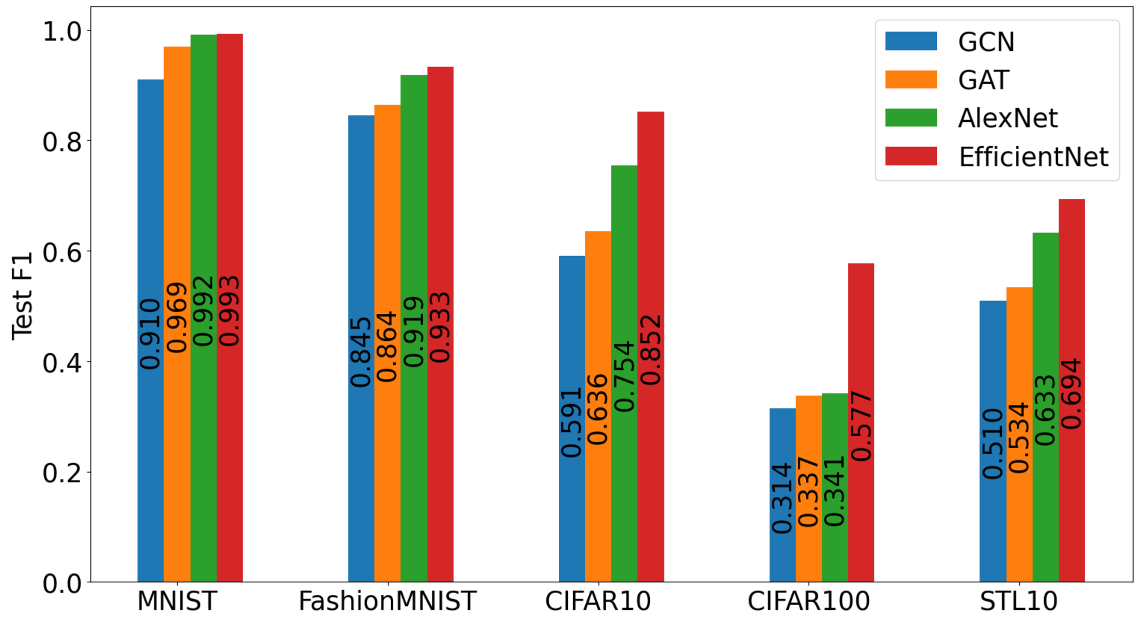
4.5.5 Architecture comparisons

GNNs are a relatively new approach to computer vision tasks – the first use for image classification, as far as we are aware, was in the work of Monti et al. (2017). In this section, we seek to compare the GCN and GAT models' performances and explore how they stand against two established convolutional neural network models for image classification: AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) and EfficientNet-B0 (TAN; LE, 2019). AlexNet was chosen for its historical relevance and for the fact that it is not as sophisticated as more recent models, offering a fairer comparison with the simple GNN models presented. EfficientNet-B0 is more recent and uses relatively few parameters, while also able to achieve similar or even superior performances as larger models.

To do so, we trained both models from scratch on each of the datasets listed in section 4.1. We used the same training method described in section 4.3, except with early stopping, halting the training process after five epochs without a minimum improvement of 0.001 in the validation F1-measure. This procedure was adopted because the training time per epoch is substantially greater for the CNN models than for the GNN models, as can be inferred from figure 4.22, which presents the average training time per epoch for each model. AlexNet was trained with a $1e-5$ learning rate and EfficientNet-B0 with a $1e-3$ learning rate. As a pre-processing step, the images were resized to 224x224 pixels and each color channel was normalized with the average and the standard deviation of color of the training images. For each dataset, we compare the resulting models with the GCN and GAT models trained with the graphs built with the best choice for each design dimension, as described in subsection 4.5.4 above.

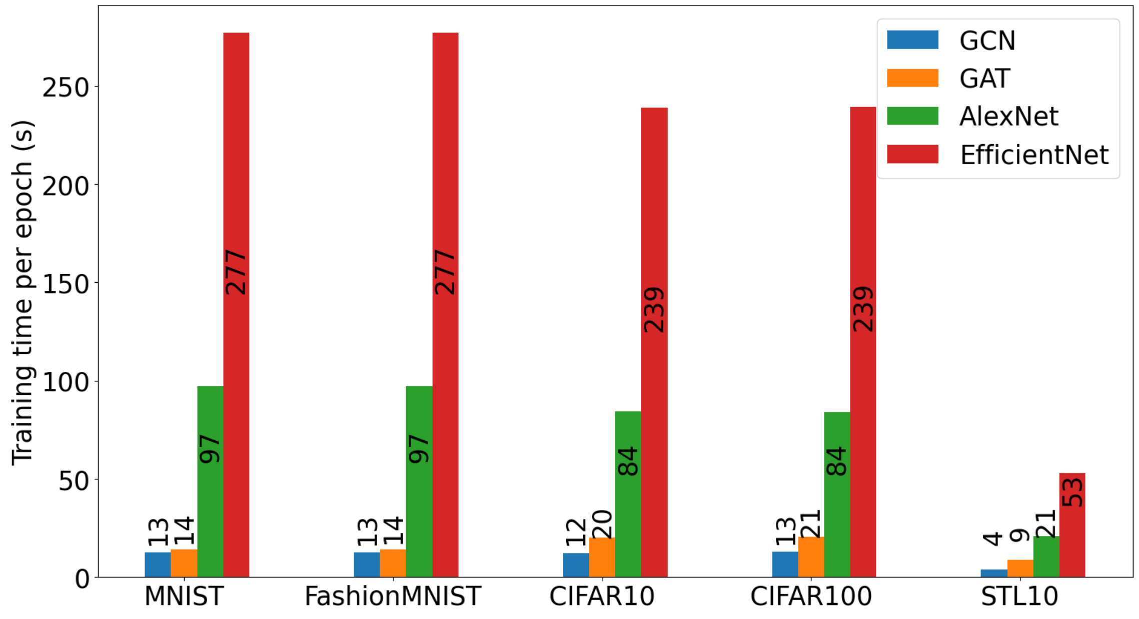
Figure 4.21 contains the final test F1-measure obtained by each model. The CNN models perform better in every case, although the GAT model has reached nearly the same performance as AlexNet on the CIFAR-100 dataset. However – as can be seen in figures 4.22 and 4.23 that show, respectively, the average training time of the five folds and the total size occupied by the parameters for each model – the computational resources required by AlexNet and EfficientNet are far greater than those necessary for training the GNN models, especially memory-wise. The training per epoch of the AlexNet model is, on average, 6.64 times slower than that of the GCN model and 4.84 times slower than that of the GAT model. EfficientNet is, respectively, 18.81 and 13.76 times slower than the GCN and the GAT model. Meanwhile, AlexNet requires an average of 5431 times more

Figure 4.21: Test F1-measure for the best-choices GCN and GAT models and for the AlexNet and EfficientNet models.



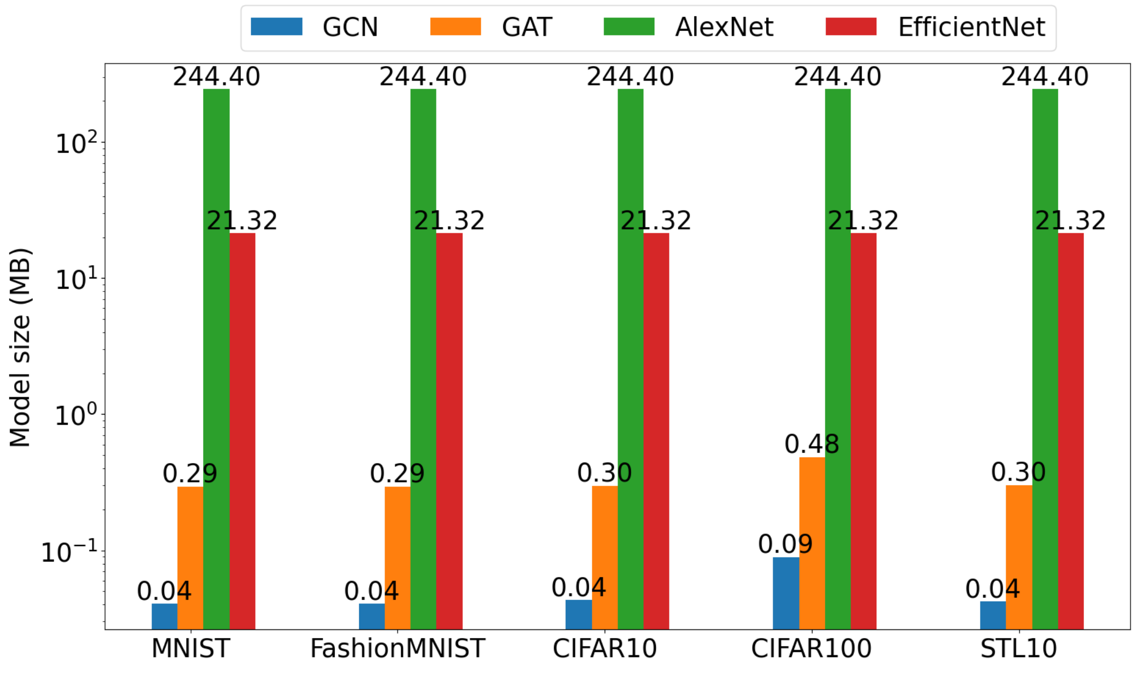
Source: The Author

Figure 4.22: Average training time per epoch in seconds for the best-choices GCN and GAT models and for the AlexNet and EfficientNet models.



Source: The Author

Figure 4.23: Size in MB occupied by the parameters of the best-choices GCN and GAT models and by the AlexNet and EfficientNet models.



Source: The Author

memory in trainable parameters than the GCN model and 765 times more than the GAT model while EfficientNet requires, respectively, 474 and 67 times more.

In every case, the GAT architecture has outperformed the GCN in F1-measure by a margin of approximately 2-6%, with comparatively little increase in memory and time resources when considering the requirements for the CNN models.

5 CONCLUSION

In this work, we have systematically analyzed the performance of simple GCN and GAT architectures for image classification as impacted by the method for building graphs from superpixel images. We evaluated three design dimensions of the graph-building process: the number of nodes (or the degree of segmentation of the original image), node feature selection, and the approaches for building edges.

We have found that, for the selected datasets, using finer-grained segmentation – that is, using more nodes to represent an image –, has a positive impact on the model's performance. This is expected, since, as the number of nodes increases, more details of the original image are represented. However, the gain in performance follows a logarithmic curve: it decreases as the number of nodes approaches the total number of pixels in the image and the graph begins to include more redundant information. Thus, it is important to consider this trade-off when using such approaches, since the memory requirements for storing graph information grow linearly with the number of nodes.

The inclusion of more descriptive features along with the standard average color information tends to have a positive effect on the performances of the models, compensating for the loss of information in the segmentation process. This is especially true for spatial information (i.e., each superpixel's geometric centroid) and standard deviations of color and position. However, some features can be detrimental to the performance. This is the case of the inclusion of the pixel density when training the GAT models. The addition of color information in different color spaces (such as HSV) can also be marginally beneficial (as is the case with the GCN models), but not always (as with the GAT models).

By comparing approaches for building edges we have found the most significant difference in behavior between GCNs and GATs. For the architecture based on GCNs, the best results were achieved when neighborhoods were restricted to similar regions. That is: small neighborhoods and connections that consider color distance along with spatial distance. The performance degrades as the node degrees increase, possibly because wide neighborhoods aggravate the over-smoothing issue that is present in GCN architectures.

GAT architectures, when using more than one attention head, benefit from the extra information provided by region adjacency graphs and perform better with those in every case. They can leverage information from contrasting regions, differently from GCN architectures. However, we note that when using only one attention head the performance decreases and the behavior becomes closer to the GCN models'.

In general, we have observed that our GAT model performs better than the GCN in every case, with a small increase in necessary time and memory resources, which are still expressively lower than those consumed by the CNN models. However, the performance achieved with the GNNs in no moment surpassed that which was achieved with the CNN models selected (namely AlexNet and EfficientNet-B0). Still, it is important to notice that in CIFAR-100 dataset the GAT and the AlexNet models achieved very similar performances. Furthermore, we note that the training time and, especially, the memory requirements are drastically reduced with the GNN models and that these are extremely simple in comparison to the CNN models. In this way, the results described do not indicate in any way that GNNs are intrinsically inferior to CNNs, especially considering that the use of graph-based networks for vision tasks is a relatively recent approach and the research in this field is active and rapidly evolving.

This work can be a reference for indicating more promising approaches for the representation of images as graphs in different tasks. It also can be the base for inspiring the development of mechanisms that explore the results we have reported for enhancing the capabilities of GNN architectures for image classification.

Grounds for future work include analyzing the effects of different node features such as shape descriptors, exploring the impact of the level of irregularity of the image segments (as parameterized by the smoothness factor in the base SLIC algorithm), and considering other methods of superpixel generation and image over-segmentation. Further examination of the correlation of the different design dimensions for graph-building (i.e., number of nodes in the graph, node features, and edge-building method) is also an interesting pursuit.

REFERENCES

ACHANTA, R. et al. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 11, p. 2274–2282, nov. 2012. ISSN 1939-3539.

ACHANTA, R.; SüSSTRUNK, S. Superpixels and polygons using simple non-iterative clustering. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2017. p. 4895–4904.

AVELAR, P. H. C. et al. Superpixel Image Classification with Graph Attention Networks. In: **2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)**. Recife/Porto de Galinhas, Brazil: IEEE, 2020. p. 203–209. ISBN 978-1-72819-274-1. Available from Internet: <https://ieeexplore.ieee.org/document/9265983/>.

BDI-UFRGS. **superpixel-graphs**. Banco de Dados Inteligente, 2023. Available from Internet: <https://github.com/BDI-UFRGS/superpixel-graphs>.

BERGH, M. Van den et al. SEEDS: Superpixels Extracted Via Energy-Driven Sampling. **Int J Comput Vis**, v. 111, n. 3, p. 298–314, feb. 2015. ISSN 1573-1405.

BESSADOK, A.; MAHJOUB, M. A.; REKIK, I. Graph neural networks in network neuroscience. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 45, n. 5, p. 5833–5848, 2022.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: **Proceedings of the Fifth Annual Workshop on Computational Learning Theory**. New York, NY, USA: Association for Computing Machinery, 1992. (COLT '92), p. 144–152. ISBN 089791497X.

BRUNA, J. et al. **Spectral Networks and Locally Connected Networks on Graphs**. 2014. Available from Internet: <http://arxiv.org/abs/1312.6203>.

CHEN, B. et al. Label co-occurrence learning with graph convolutional networks for multi-label chest x-ray image classification. **IEEE journal of biomedical and health informatics**, IEEE, v. 24, n. 8, p. 2292–2302, 2020.

CHEN, C. et al. A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective. **arXiv preprint arXiv:2209.13232**, 2022.

COATES, A.; LEE, H.; NG, A. Y. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. 2011. Available from Internet: <http://cs.stanford.edu/~acoates/stl10>.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, v. 20, n. 3, p. 273–297, sep. 1995. ISSN 0885-6125, 1573-0565. Available from Internet: <http://link.springer.com/10.1007/BF00994018>.

COSMA, R. A. et al. Geometric Superpixel Representations for Efficient Image Classification with Graph Neural Networks. In: . [s.n.], 2023. p. 109–118. Available from Internet: <https://openaccess.thecvf.com/content/ICCV2023W/VIPriors/html/Cosma_Geometric_Superpixel_Representations_for_Efficient_Image_Classification_with_Graph_Neural_ICCVW_2023_paper.html>.

DEFFERRARD, M.; BRESSON, X.; VANDERGHEYNST, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In: **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2016. v. 29.

DU, H. et al. Automatic calcification morphology and distribution classification for breast mammograms with multi-task graph convolutional neural network. **IEEE Journal of Biomedical and Health Informatics**, IEEE, 2023.

ERRICA, F. et al. **A Fair Comparison of Graph Neural Networks for Graph Classification**. arXiv, 2022. ArXiv:1912.09893 [cs, stat]. Available from Internet: <http://arxiv.org/abs/1912.09893>.

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

HE, K. et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016.

HONG, D. et al. Graph convolutional networks for hyperspectral image classification. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 59, n. 7, p. 5966–5978, 2020.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. 2014. Publisher: arXiv Version Number: 9. Available from Internet: <https://arxiv.org/abs/1412.6980>.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. In: **International Conference on Learning Representations**. [s.n.], 2017. Available from Internet: <https://openreview.net/forum?id=SJU4ayYgl>.

KRIZHEVSKY, A. Learning Multiple Layers of Features from Tiny Images. nov. 2009.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2012. v. 25. Available from Internet: <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

LANCHANTIN, J. et al. General multi-label image classification with transformers. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2021. p. 16478–16488.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, nov. 1998. ISSN 1558-2256.

LI, Q.; HAN, Z.; WU, X.-m. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 32, n. 1, abr. 2018. ISSN 2374-3468, 2159-5399. Available from Internet: <https://ojs.aaai.org/index.php/AAAI/article/view/11604>.

LINH, L. V.; YOUN, C.-H. Dynamic Graph Neural Network for Super-Pixel Image Classification. In: **2021 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2021. p. 1095–1099. ISSN: 2162-1233.

LONG, J.; YAN, Z.; CHEN, H. A Graph Neural Network for superpixel image classification. **Journal of Physics: Conference Series**, v. 1871, n. 1, p. 012071, abr. 2021. ISSN 1742-6588, 1742-6596. Available from Internet: <https://iopscience.iop.org/article/10.1088/1742-6596/1871/1/012071>.

MONTI, F. et al. Geometric deep learning on graphs and manifolds using mixture model cnns. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2017.

OpenCV: Superpixels. Available from Internet: <https://docs.opencv.org/3.4/df/d6c/group__ximgproc__superpixel.html#gacf29df20eaca242645a8d3a2d88e6c06>.

REISER, P. et al. Graph neural networks for materials science and chemistry. **Communications Materials**, Nature Publishing Group UK London, v. 3, n. 1, p. 93, 2022.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, oct. 1986. ISSN 0028-0836, 1476-4687. Available from Internet: <https://www.nature.com/articles/323533a0>.

SCARSELLI, F. et al. The Graph Neural Network Model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, jan. 2009. ISSN 1941-0093. Conference Name: IEEE Transactions on Neural Networks.

SHCHUR, O. et al. **Pitfalls of Graph Neural Network Evaluation**. arXiv, 2019. ArXiv:1811.05868 [cs, stat]. Available from Internet: <http://arxiv.org/abs/1811.05868>.

SHLOMI, J.; BATTAGLIA, P.; VLIMANT, J.-R. Graph neural networks in particle physics. **Machine Learning: Science and Technology**, IOP Publishing, v. 2, n. 2, p. 021001, 2020.

STUTZ, D.; HERMANS, A.; LEIBE, B. Superpixels: An evaluation of the state-of-the-art. **Computer Vision and Image Understanding**, v. 166, p. 1–27, jan. 2018. ISSN 10773142. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S1077314217300589>.

SZELISKI, R. **Computer vision: algorithms and applications**. Second edition. Cham: Springer, 2022. (Texts in computer science). ISBN 978-3-030-34371-2.

TAN, M.; LE, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). **Proceedings of the 36th International Conference on Machine Learning**. PMLR, 2019. (Proceedings of Machine Learning Research, v. 97), p. 6105–6114. Available from Internet: <https://proceedings.mlr.press/v97/tan19a.html>.

TANG, T. et al. Image classification based on deep graph convolutional networks. In: IEEE. **2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)**. [S.l.], 2022. p. 1–6.

TODESCATO, M. V. et al. Multiscale context features for geological image classification. In: **ICEIS (1)**. [S.l.: s.n.], 2023. p. 407–418.

TODESCATO, M. V. et al. Multiscale patch-based feature graphs for image classification. **Expert Systems with Applications**, Elsevier, v. 235, p. 121116, 2024.

VANI, S.; RAO, T. V. M. An experimental approach towards the performance assessment of various optimizers on convolutional neural network. In: **2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)**. [S.l.: s.n.], 2019. p. 331–336.

VELIčKOVIć, P. et al. Graph attention networks. In: **International Conference on Learning Representations**. [s.n.], 2018. Available from Internet: <https://openreview.net/forum?id=rJXMpikCZ>.

WU, L. et al. Graph neural networks for natural language processing: A survey. **Foundations and Trends® in Machine Learning**, Now Publishers, Inc., v. 16, n. 2, p. 119–328, 2023.

WU, Z. et al. A comprehensive survey on graph neural networks. **IEEE transactions on neural networks and learning systems**, IEEE, v. 32, n. 1, p. 4–24, 2020.

WU, Z. et al. A comprehensive survey on graph neural networks. **IEEE Transactions on Neural Networks and Learning Systems**, v. 32, n. 1, p. 4–24, 2021.

XIAO, H.; RASUL, K.; VOLLGRAF, R. **Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms**. arXiv, 2017. Available from Internet: <http://arxiv.org/abs/1708.07747>.

XU, K. et al. **How Powerful are Graph Neural Networks?** arXiv, 2019. ArXiv:1810.00826 [cs, stat]. Available from Internet: <http://arxiv.org/abs/1810.00826>.

YAO, J. et al. Real-time coarse-to-fine topologically preserving segmentation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015.

ZHANG, S. et al. Graph convolutional networks: a comprehensive review. **Computational Social Networks**, SpringerOpen, v. 6, n. 1, p. 1–23, 2019.

ZHANG, W. et al. Component segmentation of engineering drawings using graph convolutional networks. **Computers in Industry**, Elsevier, v. 147, p. 103885, 2023.

ZHANG, X.-M. et al. Graph neural networks and their current applications in bioinformatics. **Frontiers in genetics**, Frontiers Media SA, v. 12, p. 690049, 2021.

APPENDIX A — EXPANDED RESULTS

This appendix contains the final test F1-measure, accuracy, and loss results with standard deviation for each of the experiments described in Chapter 4.

A.1 Number of nodes

Tables A.1 through A.15 show the final test F1-measure, accuracy, and loss results for the GCN and GAT models described in 4.2 for the number of nodes experiment described in 4.4.1.

A.2 Node features

Tables A.16 through A.30 show the final test F1-measure, accuracy, and loss results for the GCN and GAT models described in 4.2 for the node features experiment described in 4.4.2.

A.3 Edge-building method

Tables A.31 through A.45 show the final test F1-measure, accuracy, and loss results for the GCN and GAT models described in 4.2 for the edge-building methods experiment described in 4.4.3.

Table A.1: Final F1-measure obtained with the number of nodes experiments for the GCN and GAT models for the **MNIST** dataset

N	N_{avg}	GCN F1-measure			GAT F1-measure		
		Test	Val.	Train	Test	Val.	Train
10	9±0	66.8±0.9	67.0±1.5	68.0±1.2	83.9±0.3	83.7±0.8	88.3±0.2
20	23±0	81.9±0.5	82.1±0.9	82.6±0.6	92.5±0.4	92.3±0.4	95.6±0.6
50	81±1	90.8±0.4	90.6±0.5	91.3±0.5	96.9±0.3	96.9±0.2	98.5±0.2
100	147±7	89.9±0.7	90.0±0.1	90.5±0.4	96.1±0.1	96.1±0.2	97.9±0.1
200	147±7	-	-	-	-	-	-
400	147±7	-	-	-	-	-	-

Table A.2: Final F1-measure obtained with the number of nodes experiments for the GCN and GAT models for the **FashionMNIST** dataset

			GCN			GAT	
N	N_{avg}		F1-measure			F1-measure	
		Test	Val.	Train	Test	Val.	Train
10	9±0	69.2±0.6	69.4±0.3	70.3±0.4	77.5±0.2	77.4±0.3	81.1±0.5
20	23±1	77.0±0.4	76.9±0.8	78.1±0.4	83.2±0.4	82.8±0.2	87.0±0.9
50	78±2	81.9±0.3	81.4±0.3	82.6±0.2	86.0±0.3	86.1±0.5	89.8±0.9
100	137±8	82.9±0.4	82.9±0.7	84.0±0.3	86.1±0.2	86.0±0.6	89.5±0.5
200	137±8	-	-	-	-	-	-
400	137±8	-	-	-	-	-	-

Table A.3: Final F1-measure obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR-10** dataset

			GCN			GAT	
N	N_{avg}		F1-measure			F1-measure	
		Test	Val.	Train	Test	Val.	Train
10	9±0	38.4±0.6	38.6±0.6	39.9±0.4	48.6±0.6	48.2±0.3	55.6±1.4
20	23±0	45.4±0.6	45.2±0.4	46.8±0.5	54.5±0.7	54.7±1.4	62.4±1.9
50	60±0	51.6±0.7	50.8±0.7	53.3±0.6	59.1±0.7	59.6±1.0	67.0±1.3
100	116±1	55.2±1.0	54.9±1.0	57.4±0.6	62.3±0.8	62.5±1.5	70.1±2.0
200	235±5	57.2±0.5	56.6±0.8	59.7±0.5	63.7±0.7	63.9±1.2	72.2±1.0
400	235±5	-	-	-	-	-	-

Table A.4: Final F1-measure obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR-100** dataset

			GCN			GAT	
N	N_{avg}		F1-measure			F1-measure	
		Test	Val.	Train	Test	Val.	Train
10	9±0	12.6±0.5	12.8±0.5	14.2±0.8	20.1±1.0	19.9±0.4	26.3±1.8
20	23±0	14.9±7.4	15.0±7.5	16.9±8.4	25.4±0.3	25.1±0.4	34.5±0.9
50	60±0	23.8±0.6	23.5±0.9	26.6±0.8	29.5±0.9	29.6±1.0	39.7±1.7
100	116±1	26.2±1.1	26.2±1.0	29.5±1.6	32.8±1.4	32.8±0.5	43.3±1.9
200	235±7	29.2±0.8	28.9±0.8	32.7±1.0	33.4±0.2	33.3±0.8	44.0±1.8
400	235±7	-	-	-	-	-	-

Table A.5: Final F1-measure obtained with the number of nodes experiments for the GCN and GAT models for the **STL-10** dataset

			GCN			GAT	
N	N_{avg}		F1-measure			F1-measure	
		Test	Val.	Train	Test	Val.	Train
10	9±0	35.5±0.7	35.7±0.8	37.3±0.4	41.6±0.8	41.3±1.6	49.2±1.0
20	23±1	39.6±1.1	39.5±0.9	41.2±0.7	44.4±0.7	44.4±1.1	52.5±2.7
50	49±0	42.8±1.7	43.1±1.9	45.4±1.1	47.4±1.0	48.1±0.6	59.2±2.2
100	116±1	45.1±1.1	44.6±1.8	48.2±0.5	51.0±1.2	51.6±0.7	62.3±3.0
200	248±1	46.9±1.5	46.1±1.8	50.4±0.8	51.7±1.6	51.5±2.0	61.8±2.4
400	563±2	48.7±1.7	48.7±1.0	52.7±0.4	53.7±1.3	53.5±1.0	65.6±1.1

Table A.6: Final Accuracy obtained with the number of nodes experiments for the GCN and GAT models for the **MNIST** dataset

			GCN			GAT	
N	N_{avg}		Accuracy			Accuracy	
		Test	Val.	Train	Test	Val.	Train
10	9±0	67.3±0.8	67.7±1.5	68.5±1.2	84.1±0.3	84.0±0.8	88.5±0.2
20	23±0	82.3±0.5	82.4±0.9	82.9±0.6	92.6±0.4	92.4±0.4	95.6±0.6
50	81±1	90.9±0.4	90.7±0.5	91.4±0.5	96.9±0.3	96.9±0.2	98.5±0.2
100	147±7	90.0±0.6	90.1±0.1	90.6±0.4	96.1±0.1	96.2±0.2	97.9±0.1
200	147±7	-	-	-			
400	147±7	-	-	-			

Table A.7: Final Accuracy obtained with the number of nodes experiments for the GCN and GAT models for the **FashionMNIST** dataset

			GCN			GAT	
N	N_{avg}		Accuracy			Accuracy	
		Test	Val.	Train	Test	Val.	Train
10	9±0	69.5±0.5	69.7±0.1	70.7±0.2	77.8±0.2	77.7±0.3	81.4±0.5
20	23±1	77.3±0.4	77.2±0.9	78.3±0.4	83.3±0.4	82.9±0.2	87.1±0.9
50	78±2	82.0±0.3	81.6±0.3	82.7±0.2	86.1±0.2	86.2±0.5	89.9±0.8
100	137±8	83.0±0.4	83.0±0.6	84.1±0.3	86.2±0.2	86.2±0.5	89.6±0.5
200	137±8	-	-	-			
400	137±8	-	-	-			

Table A.8: Final Accuracy obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR10** dataset

			GCN			GAT	
N	N_{avg}		Accuracy			Accuracy	
		Test	Val.	Train	Test	Val.	Train
10	9±0	39.2±0.8	39.4±0.3	40.6±0.4	48.9±0.7	48.5±0.4	55.9±1.4
20	23±0	45.9±0.5	45.7±0.4	47.3±0.3	54.8±0.7	55.0±1.3	62.6±1.8
50	60±0	52.1±0.7	51.4±0.8	53.8±0.6	59.3±0.7	59.7±0.9	67.1±1.3
100	116±1	55.4±0.9	55.1±1.0	57.7±0.5	62.5±0.8	62.7±1.4	70.3±1.9
200	235±5	57.5±0.5	57.0±0.8	60.0±0.5	63.9±0.7	64.2±1.1	72.4±1.0
400	235±5	-	-	-			

Table A.9: Final Accuracy obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR100** dataset

			GCN			GAT	
N	N_{avg}		Accuracy			Accuracy	
		Test	Val.	Train	Test	Val.	Train
10	9±0	14.6±0.6	14.9±0.6	16.3±0.8	21.4±0.8	21.3±0.3	27.6±1.7
20	23±0	16.4±7.7	16.5±7.8	18.4±8.7	26.3±0.2	26.1±0.5	35.3±0.9
50	60±0	25.4±0.8	25.2±1.1	28.1±1.0	30.3±1.0	30.4±0.9	40.4±1.7
100	116±1	27.6±1.3	27.6±1.2	30.8±1.8	33.4±1.3	33.6±0.3	43.8±1.9
200	235±7	30.3±0.7	30.3±0.8	33.9±0.9	34.3±0.3	34.3±0.7	44.8±1.6
400	235±7	-	-	-			

Table A.10: Final Accuracy obtained with the number of nodes experiments for the GCN and GAT models for the **STL10** dataset

N	N_{avg}	GCN Accuracy Test	GCN Accuracy Val.	GCN Accuracy Train	GAT Accuracy Test	GAT Accuracy Val.	GAT Accuracy Train
10	9±0	36.6±0.6	36.6±1.1	38.2±0.2	42.1±0.6	41.9±1.5	49.6±0.8
20	23±1	40.5±0.6	40.4±0.7	42.1±0.5	45.0±0.6	45.2±1.0	53.0±2.7
50	49±0	43.5±1.2	43.9±1.3	46.1±0.8	47.5±1.0	48.1±1.0	59.3±2.3
100	116±1	46.1±1.1	45.6±1.9	49.1±0.4	51.0±1.1	51.7±0.6	62.4±2.8
200	248±1	47.9±1.6	47.1±2.0	51.3±0.7	51.9±1.4	51.8±2.0	61.9±2.3
400	563±2	49.0±1.7	49.1±0.8	53.1±0.4	54.1±1.3	53.9±0.8	65.8±1.0

Table A.11: Final Loss obtained with the number of nodes experiments for the GCN and GAT models for the **MNIST** dataset

N	N_{avg}	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
10	9±0	1.00±0.02	1.00±0.05	0.96±0.03	0.51±0.01	0.51±0.02	0.35±0.01
20	23±0	0.55±0.01	0.54±0.03	0.53±0.02	0.24±0.01	0.24±0.01	0.13±0.02
50	81±1	0.29±0.02	0.29±0.02	0.27±0.02	0.10±0.01	0.11±0.01	0.05±0.01
100	147±7	0.32±0.01	0.32±0.00	0.29±0.01	0.13±0.01	0.13±0.01	0.06±0.00
200	147±7	-	-	-			
400	147±7	-	-	-			

Table A.12: Final Loss obtained with the number of nodes experiments for the GCN and GAT models for the **FashionMNIST** dataset

N	N_{avg}	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
10	9±0	0.82±0.01	0.82±0.01	0.79±0.01	0.59±0.01	0.59±0.00	0.49±0.01
20	23±1	0.60±0.01	0.61±0.01	0.57±0.01	0.46±0.01	0.46±0.01	0.35±0.03
50	78±2	0.49±0.01	0.49±0.01	0.46±0.01	0.38±0.01	0.38±0.01	0.27±0.02
100	137±8	0.46±0.01	0.45±0.01	0.43±0.01	0.38±0.00	0.38±0.01	0.28±0.01
200	137±8	-	-	-			
400	137±8	-	-	-			

Table A.13: Final Loss obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR10** dataset

N	N_{avg}	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
10	9±0	1.68±0.01	1.68±0.01	1.65±0.01	1.47±0.01	1.47±0.01	1.25±0.04
20	23±0	1.53±0.01	1.53±0.01	1.48±0.01	1.29±0.01	1.29±0.02	1.06±0.05
50	60±0	1.35±0.02	1.36±0.02	1.29±0.01	1.16±0.01	1.16±0.02	0.94±0.04
100	116±1	1.26±0.02	1.27±0.01	1.19±0.01	1.08±0.01	1.08±0.03	0.85±0.06
200	235±5	1.20±0.01	1.20±0.01	1.12±0.01	1.04±0.02	1.04±0.02	0.79±0.03
400	235±5	-	-	-			

Table A.14: Final Loss obtained with the number of nodes experiments for the GCN and GAT models for the **CIFAR100** dataset

		GCN			GAT		
N	N_{avg}	Loss			Loss		
		Test	Val.	Train	Test	Val.	Train
10	9±0	3.68±0.03	3.67±0.02	3.52±0.05	3.36±0.02	3.34±0.01	2.93±0.09
20	23±0	3.60±0.50	3.60±0.50	3.47±0.57	3.13±0.02	3.12±0.02	2.56±0.04
50	60±0	3.07±0.04	3.07±0.05	2.91±0.05	2.90±0.05	2.89±0.06	2.32±0.08
100	116±1	2.95±0.07	2.94±0.06	2.78±0.09	2.74±0.05	2.74±0.03	2.16±0.08
200	235±7	2.82±0.03	2.81±0.03	2.62±0.04	2.71±0.02	2.68±0.03	2.11±0.07
400	235±7	-	-	-			

Table A.15: Final Loss obtained with the number of nodes experiments for the GCN and GAT models for the **STL10** dataset

		GCN			GAT		
N	N_{avg}	Loss			Loss		
		Test	Val.	Train	Test	Val.	Train
10	9±0	1.66±0.01	1.64±0.02	1.62±0.00	1.57±0.02	1.56±0.02	1.35±0.03
20	23±1	1.59±0.03	1.57±0.02	1.54±0.01	1.49±0.02	1.47±0.02	1.28±0.06
50	49±0	1.53±0.04	1.52±0.02	1.46±0.02	1.44±0.02	1.41±0.02	1.12±0.06
100	116±1	1.48±0.03	1.47±0.03	1.40±0.01	1.36±0.04	1.33±0.02	1.04±0.08
200	248±1	1.44±0.03	1.43±0.03	1.35±0.01	1.32±0.03	1.30±0.04	1.06±0.06
400	563±2	1.39±0.03	1.39±0.02	1.29±0.01	1.29±0.02	1.25±0.03	0.95±0.03

Table A.16: Final F1-measure obtained with the node features experiments for the GCN and GAT models for the **MNIST** dataset

	GCN			GAT		
Feature	F1-measure			F1-measure		
	Test	Val.	Train	Test	Val.	Train
Avg. color	59.5±1.5	59.4±1.3	59.8±1.6	71.0±1.5	70.8±1.1	71.8±1.3
Centroid	89.1±0.4	89.2±0.5	89.6±0.6	96.6±0.2	96.5±0.3	98.1±0.1
Std. dev. color	90.2±0.3	90.1±0.5	90.6±0.5	96.5±0.4	96.6±0.3	98.1±0.5
Std. dev. centroid	91.2±0.3	91.3±0.4	91.7±0.3	96.8±0.3	96.8±0.1	98.4±0.2
Num. of pixels	90.9±0.6	91.0±0.3	91.6±0.3	96.7±0.1	96.7±0.2	98.3±0.2

Table A.17: Final F1-measure obtained with the node features experiments for the GCN and GAT models for the **FashionMNIST** dataset

	GCN			GAT		
Feature	F1-measure			F1-measure		
	Test	Val.	Train	Test	Val.	Train
Avg. color	61.2±1.4	61.6±1.3	61.6±1.0	71.3±0.6	70.9±0.6	72.2±0.6
Centroid	78.6±0.6	78.5±0.6	79.4±0.4	84.9±0.3	84.6±0.2	88.0±0.5
Std. dev. color	80.6±0.3	80.3±0.5	81.4±0.4	85.9±0.2	85.8±0.3	89.4±0.8
Std. dev. centroid	81.4±0.4	81.4±0.4	82.3±0.4	86.3±0.2	86.0±0.4	89.6±0.4
Num. of pixels	81.8±0.4	81.6±0.4	82.7±0.2	85.8±0.2	85.6±0.3	89.3±0.7

Table A.18: Final F1-measure obtained with the node features experiments for the GCN and GAT models for the **CIFAR10** dataset

Feature	GCN F1-measure			GAT F1-measure		
	Test	Val.	Train	Test	Val.	Train
Avg. color	40.7±0.8	40.2±0.6	42.1±0.5	51.5±0.4	51.6±0.4	56.8±0.2
Centroid	49.5±0.5	48.9±0.5	51.1±0.6	59.1±0.6	59.3±0.3	66.7±1.0
Std. dev. color	54.5±0.3	54.3±0.6	56.6±0.3	62.1±0.6	62.5±0.8	70.0±0.8
Std. dev. centroid	54.6±0.6	54.0±0.7	57.0±0.5	62.4±0.5	62.6±1.1	70.4±0.8
Num. of pixels	55.2±0.4	55.0±0.8	57.4±0.4	62.2±0.4	62.0±1.0	70.4±1.7
Avg. HSV	55.0±0.6	54.9±0.6	57.8±0.6	62.1±0.8	61.8±0.7	70.7±1.4
Std. dev. HSV	54.9±0.5	55.1±0.4	57.5±0.6	62.0±0.3	61.7±0.7	69.6±1.3

Table A.19: Final F1-measure obtained with the node features experiments for the GCN and GAT models for the **CIFAR100** dataset

Feature	GCN F1-measure			GAT F1-measure		
	Test	Val.	Train	Test	Val.	Train
Avg. color	15.1±0.5	15.2±0.5	17.4±0.4	24.7±0.4	25.0±0.1	32.7±0.6
Centroid	20.6±0.6	20.8±0.8	22.5±1.1	27.1±0.1	27.3±0.6	35.6±1.1
Std. dev. color	26.3±0.8	26.3±0.2	29.5±0.9	32.6±0.3	32.3±1.0	43.2±1.6
Std. dev. centroid	26.4±1.3	25.8±0.8	29.4±1.5	32.8±0.6	33.1±0.6	42.9±0.5
Num. of pixels	26.4±1.3	25.7±1.0	29.3±1.7	31.9±0.6	31.6±1.1	41.5±1.7
Avg. HSV	26.8±1.0	26.7±1.2	30.7±1.6	31.7±0.8	31.8±0.5	42.4±1.4
Std. dev. HSV	27.2±1.1	27.1±1.0	30.9±1.6	32.4±0.4	32.5±0.7	43.9±1.5

Table A.20: Final F1-measure obtained with the node features experiments for the GCN and GAT models for the **STL10** dataset

Feature	GCN F1-measure			GAT F1-measure		
	Test	Val.	Train	Test	Val.	Train
Avg. color	34.6±0.5	34.9±1.3	36.5±0.6	45.0±0.9	45.6±1.4	51.0±0.5
Centroid	37.4±0.3	37.8±1.8	39.1±1.0	43.8±1.2	44.0±1.2	52.1±2.6
Std. dev. color	42.9±1.6	42.7±1.8	45.1±0.7	48.3±0.8	49.0±2.3	59.1±1.9
Std. dev. centroid	43.5±0.8	43.2±0.9	46.1±0.7	48.9±1.3	49.0±0.6	59.3±2.9
Num. of pixels	44.0±1.0	43.6±1.4	47.4±0.5	45.9±0.6	45.5±1.8	54.9±1.3
Avg. HSV	46.5±0.3	47.1±0.8	50.1±0.9	49.7±0.8	51.3±1.6	60.7±3.8
Std. dev. HSV	46.4±0.8	46.2±1.5	50.4±1.3	49.7±1.8	50.4±1.5	60.2±4.6

Table A.21: Final Accuracy obtained with the node features experiments for the GCN and GAT models for the **MNIST** dataset

Feature	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
Avg. color	60.8±1.4	60.8±1.2	61.1±1.4	71.7±1.4	71.5±1.0	72.5±1.2
Centroid	89.2±0.4	89.3±0.5	89.7±0.6	96.6±0.2	96.5±0.3	98.1±0.1
Std. dev. color	90.3±0.3	90.2±0.5	90.7±0.5	96.5±0.4	96.6±0.3	98.1±0.5
Std. dev. centroid	91.2±0.3	91.4±0.4	91.8±0.2	96.8±0.3	96.8±0.1	98.4±0.2
Num. of pixels	91.0±0.6	91.1±0.3	91.6±0.3	96.8±0.1	96.7±0.2	98.3±0.2

Table A.22: Final Accuracy obtained with the node features experiments for the GCN and GAT models for the **FashionMNIST** dataset

Feature		GCN Accuracy			GAT Accuracy	
	Test	Val.	Train	Test	Val.	Train
Avg. color	61.4±1.0	61.8±1.1	61.8±0.7	71.6±0.7	71.2±0.6	72.5±0.7
Centroid	78.9±0.5	78.8±0.6	79.7±0.4	85.0±0.3	84.8±0.2	88.0±0.4
Std. dev. color	80.9±0.3	80.7±0.4	81.7±0.3	86.0±0.1	85.9±0.3	89.5±0.7
Std. dev. centroid	81.7±0.4	81.8±0.4	82.7±0.3	86.4±0.2	86.1±0.4	89.7±0.4
Num. of pixels	81.9±0.4	81.7±0.4	82.9±0.3	86.0±0.2	85.7±0.2	89.3±0.7

Table A.23: Final Accuracy obtained with the node features experiments for the GCN and GAT models for the **CIFAR10** dataset

Feature		GCN Accuracy			GAT Accuracy	
	Test	Val.	Train	Test	Val.	Train
Avg. color	41.2±0.9	40.8±0.8	42.6±0.5	51.6±0.4	51.7±0.5	57.0±0.2
Centroid	50.0±0.6	49.4±0.4	51.6±0.6	59.2±0.6	59.6±0.2	66.8±0.9
Std. dev. color	54.9±0.2	54.7±0.6	57.0±0.3	62.2±0.7	62.6±1.0	70.0±0.8
Std. dev. centroid	55.0±0.6	54.5±0.6	57.4±0.5	62.5±0.4	62.7±1.1	70.6±0.8
Num. of pixels	55.6±0.3	55.5±0.6	57.8±0.3	62.4±0.4	62.2±1.0	70.5±1.6
Avg. HSV	55.4±0.4	55.2±0.4	58.2±0.6	62.3±0.7	62.0±0.8	70.8±1.3
Std. dev. HSV	55.4±0.5	55.6±0.2	58.0±0.5	62.1±0.3	62.0±0.7	69.8±1.2

Table A.24: Final Accuracy obtained with the node features experiments for the GCN and GAT models for the **CIFAR100** dataset

Feature		GCN Accuracy			GAT Accuracy	
	Test	Val.	Train	Test	Val.	Train
Avg. color	16.9±0.4	17.0±0.5	19.1±0.3	26.0±0.5	26.3±0.2	34.0±0.7
Centroid	22.1±0.5	22.5±1.0	24.1±1.0	28.3±0.2	28.5±0.6	36.7±1.1
Std. dev. color	27.7±0.6	27.6±0.4	30.9±0.8	33.4±0.6	33.1±1.2	43.9±1.5
Std. dev. centroid	27.7±1.1	27.1±0.5	30.7±1.4	33.7±0.7	34.1±0.5	43.7±0.6
Num. of pixels	27.7±1.1	27.3±0.8	30.7±1.6	32.8±0.3	32.6±1.0	42.3±1.6
Avg. HSV	28.1±0.9	28.0±1.1	31.9±1.4	32.7±0.8	32.9±0.4	43.3±1.3
Std. dev. HSV	28.2±1.2	28.4±1.0	32.0±1.6	33.2±0.5	33.3±0.7	44.5±1.5

Table A.25: Final Accuracy obtained with the node features experiments for the GCN and GAT models for the **STL10** dataset

Feature		GCN Accuracy			GAT Accuracy	
	Test	Val.	Train	Test	Val.	Train
Avg. color	35.6±0.3	36.0±1.2	37.6±0.8	45.2±0.8	45.9±1.3	51.1±0.5
Centroid	38.6±0.5	38.9±1.7	40.2±0.8	44.7±1.4	44.7±1.0	52.8±2.8
Std. dev. color	43.5±2.0	43.3±1.7	45.7±0.7	48.9±0.6	49.7±2.2	59.6±1.9
Std. dev. centroid	44.4±0.6	44.4±0.9	46.9±0.5	49.3±1.2	49.4±0.7	59.6±2.8
Num. of pixels	44.7±1.0	44.3±1.5	47.9±0.7	46.7±0.6	46.4±1.7	55.6±1.3
Avg. HSV	46.8±0.5	47.4±0.7	50.4±0.6	50.3±0.7	51.8±1.8	61.0±3.6
Std. dev. HSV	47.1±0.7	46.9±1.4	51.1±0.9	50.4±1.6	50.9±1.4	60.7±4.3

Table A.26: Final Loss obtained with the node features experiments for the GCN and GAT models for the **MNIST** dataset

Feature		GCN Loss			GAT Loss	
	Test	Val.	Train	Test	Val.	Train
Avg. color	1.19±0.03	1.20±0.03	1.18±0.03	0.87±0.04	0.88±0.03	0.85±0.04
Centroid	0.34±0.01	0.34±0.01	0.33±0.02	0.11±0.00	0.11±0.01	0.06±0.01
Std. dev. color	0.31±0.01	0.32±0.02	0.29±0.01	0.12±0.01	0.11±0.01	0.06±0.01
Std. dev. centroid	0.28±0.01	0.28±0.01	0.26±0.01	0.11±0.01	0.11±0.00	0.05±0.01
Num. of pixels	0.29±0.02	0.28±0.01	0.26±0.01	0.11±0.01	0.11±0.01	0.05±0.01

Table A.27: Final Loss obtained with the node features experiments for the GCN and GAT models for the **FashionMNIST** dataset

Feature		GCN Loss			GAT Loss	
	Test	Val.	Train	Test	Val.	Train
Avg. color	1.01±0.02	1.00±0.03	1.00±0.02	0.80±0.02	0.80±0.02	0.77±0.02
Centroid	0.56±0.01	0.56±0.01	0.54±0.01	0.41±0.00	0.42±0.01	0.32±0.01
Std. dev. color	0.51±0.01	0.51±0.01	0.48±0.01	0.39±0.01	0.39±0.01	0.29±0.02
Std. dev. centroid	0.49±0.01	0.49±0.01	0.46±0.01	0.38±0.01	0.38±0.01	0.28±0.01
Num. of pixels	0.49±0.01	0.49±0.01	0.46±0.01	0.39±0.01	0.39±0.01	0.29±0.02

Table A.28: Final Loss obtained with the node features experiments for the GCN and GAT models for the **CIFAR10** dataset

Feature		GCN Loss			GAT Loss	
	Test	Val.	Train	Test	Val.	Train
Avg. color	1.67±0.02	1.67±0.02	1.63±0.01	1.38±0.01	1.38±0.01	1.23±0.01
Centroid	1.41±0.01	1.42±0.01	1.36±0.02	1.17±0.01	1.18±0.01	0.94±0.03
Std. dev. color	1.28±0.01	1.29±0.01	1.21±0.01	1.08±0.01	1.08±0.02	0.85±0.03
Std. dev. centroid	1.27±0.02	1.27±0.01	1.20±0.01	1.08±0.01	1.09±0.03	0.84±0.02
Num. of pixels	1.26±0.01	1.26±0.01	1.19±0.01	1.09±0.00	1.09±0.01	0.84±0.05
Avg. HSV	1.25±0.01	1.27±0.01	1.18±0.01	1.09±0.02	1.10±0.01	0.83±0.04
Std. dev. HSV	1.25±0.01	1.26±0.01	1.18±0.01	1.10±0.01	1.10±0.02	0.86±0.03

Table A.29: Final Loss obtained with the node features experiments for the GCN and GAT models for the **CIFAR100** dataset

Feature		GCN Loss			GAT Loss	
	Test	Val.	Train	Test	Val.	Train
Avg. color	3.53±0.01	3.52±0.01	3.37±0.02	3.09±0.02	3.08±0.03	2.63±0.03
Centroid	3.23±0.03	3.23±0.02	3.11±0.04	2.99±0.04	2.98±0.03	2.48±0.05
Std. dev. color	2.96±0.02	2.94±0.02	2.77±0.04	2.76±0.03	2.74±0.04	2.16±0.08
Std. dev. centroid	2.95±0.05	2.95±0.03	2.78±0.07	2.72±0.02	2.72±0.02	2.17±0.03
Num. of pixels	2.95±0.06	2.94±0.05	2.77±0.08	2.77±0.02	2.76±0.03	2.23±0.07
Avg. HSV	2.92±0.03	2.90±0.04	2.71±0.06	2.77±0.03	2.75±0.02	2.18±0.06
Std. dev. HSV	2.91±0.04	2.90±0.05	2.70±0.08	2.74±0.01	2.72±0.02	2.13±0.07

Table A.30: Final Loss obtained with the node features experiments for the GCN and GAT models for the **STL10** dataset

Feature	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
Avg. color	1.77±0.02	1.75±0.02	1.72±0.02	1.48±0.02	1.45±0.03	1.34±0.02
Centroid	1.68±0.01	1.66±0.04	1.64±0.02	1.50±0.03	1.48±0.02	1.28±0.07
Std. dev. color	1.54±0.03	1.53±0.03	1.49±0.02	1.39±0.02	1.38±0.03	1.12±0.05
Std. dev. centroid	1.51±0.01	1.49±0.02	1.43±0.01	1.39±0.03	1.37±0.02	1.11±0.07
Num. of pixels	1.51±0.03	1.50±0.03	1.42±0.02	1.46±0.03	1.45±0.03	1.20±0.03
Avg. HSV	1.46±0.01	1.43±0.03	1.35±0.02	1.38±0.04	1.33±0.03	1.07±0.09
Std. dev. HSV	1.44±0.02	1.42±0.03	1.33±0.02	1.36±0.03	1.34±0.03	1.09±0.11

Table A.31: Final F1-measure obtained with the edge-building methods experiments for the GCN and GAT models for the **MNIST** dataset

Method	GCN F1-measure Test	GCN F1-measure Val.	GCN F1-measure Train	GAT F1-measure Test	GAT F1-measure Val.	GAT F1-measure Train
RAG	90.6±0.6	90.8±0.5	91.3±0.5	92.8±0.6	92.8±0.6	93.3±0.5
1NNSpatial	88.0±0.7	87.6±0.9	89.0±0.8	91.6±1.3	91.7±1.2	92.3±1.1
2NNSpatial	87.2±0.9	87.2±1.1	88.3±1.1	92.2±0.8	92.2±0.9	92.9±0.9
4NNSpatial	89.8±0.3	90.0±0.6	90.5±0.3	92.3±0.6	92.4±0.4	93.0±0.7
8NNSpatial	88.8±0.7	88.7±1.1	89.3±0.9	91.6±1.0	91.7±0.9	92.3±0.8
16NNSpatial	81.5±0.8	81.3±1.1	81.9±0.9	87.5±6.5	87.4±6.3	88.0±6.3
1NNCombined	88.1±0.7	88.1±0.8	89.0±1.0	87.1±1.2	87.0±1.1	87.6±1.0
2NNCombined	88.0±1.0	87.8±0.9	88.8±0.9	86.8±0.9	87.0±0.8	87.2±0.6
4NNCombined	86.7±0.6	86.6±0.8	87.3±0.6	84.1±0.5	84.4±0.8	84.8±0.5
8NNCombined	84.7±0.5	84.4±1.0	85.1±0.8	82.1±1.5	82.1±1.4	82.5±1.4
16NNCombined	79.2±0.7	79.1±1.0	79.6±0.6	80.2±2.5	80.3±2.4	80.6±2.2

Table A.32: Final F1-measure obtained with the edge-building methods experiments for the GCN and GAT models for the **FashionMNIST** dataset

Method	GCN F1-measure Test	GCN F1-measure Val.	GCN F1-measure Train	GAT F1-measure Test	GAT F1-measure Val.	GAT F1-measure Train
RAG	81.5±0.5	81.4±0.5	82.5±0.2	82.2±0.4	82.2±0.7	83.2±0.3
1NNSpatial	81.7±0.2	81.4±0.2	83.3±0.3	82.0±0.4	81.8±0.4	83.1±0.4
2NNSpatial	81.4±0.3	81.3±0.2	82.7±0.3	82.2±0.2	81.8±0.7	83.2±0.5
4NNSpatial	81.5±0.4	81.5±0.4	82.5±0.1	82.4±0.7	82.4±0.6	83.3±0.6
8NNSpatial	79.5±0.5	79.4±0.7	80.2±0.3	81.7±0.7	81.7±0.7	82.6±0.8
16NNSpatial	75.7±0.6	76.0±0.6	76.3±0.4	79.5±1.0	79.5±0.6	80.2±0.9
1NNCombined	83.7±0.5	83.5±0.7	85.1±0.3	82.8±0.3	82.3±0.2	83.5±0.3
2NNCombined	83.4±0.3	83.4±0.2	84.6±0.2	82.5±0.3	82.2±0.6	83.4±0.4
4NNCombined	82.8±0.1	82.7±0.5	83.8±0.2	81.7±0.4	81.4±0.6	82.5±0.3
8NNCombined	81.4±0.3	81.2±0.5	82.3±0.2	80.9±0.3	80.8±0.5	81.7±0.2
16NNCombined	78.7±0.4	78.5±0.9	79.5±0.4	79.2±0.8	79.3±1.0	79.9±1.0

Table A.33: Final F1-measure obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR10** dataset

	GCN			GAT		
Method	F1-measure			F1-measure		
	Test	Val.	Train	Test	Val.	Train
RAG	54.7±0.5	54.2±0.5	57.1±0.5	54.9±0.5	54.6±0.8	56.5±0.6
1NNSpatial	56.4±0.8	56.3±0.8	59.4±1.0	54.9±0.4	54.6±0.4	56.7±0.7
2NNSpatial	56.9±0.5	56.8±0.4	59.5±0.6	54.6±0.4	54.6±1.0	56.5±0.6
4NNSpatial	54.7±0.8	54.5±0.8	56.7±0.9	54.1±0.6	54.0±0.9	55.7±0.7
8NNSpatial	52.9±0.6	52.5±0.5	54.7±0.6	53.3±0.7	53.3±1.1	54.9±0.9
16NNSpatial	48.5±0.5	48.2±0.8	49.7±0.5	52.1±0.9	51.9±0.5	53.6±0.6
1NNCombined	57.2±0.8	57.3±0.5	60.3±0.7	54.9±0.5	55.0±0.8	56.8±0.5
2NNCombined	56.9±0.4	56.8±0.7	59.9±0.3	54.6±0.6	54.8±0.5	56.4±0.8
4NNCombined	56.4±0.4	56.2±0.8	58.9±0.4	54.4±0.8	54.7±0.7	56.2±0.6
8NNCombined	54.6±0.9	54.0±0.7	56.6±0.7	53.7±0.7	53.7±0.7	55.3±0.7
16NNCombined	51.4±0.5	51.4±0.7	53.2±0.5	52.3±0.5	52.2±0.6	53.7±0.4

Table A.34: Final F1-measure obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR100** dataset

	GCN			GAT		
Method	F1-measure			F1-measure		
	Test	Val.	Train	Test	Val.	Train
RAG	26.7±0.8	26.4±0.6	29.7±1.1	28.4±0.6	28.4±0.4	32.0±0.5
1NNSpatial	28.6±0.8	28.2±1.1	32.2±1.4	29.1±0.5	28.9±0.4	33.1±0.5
2NNSpatial	27.7±1.2	27.6±0.8	31.0±1.3	29.2±0.4	28.9±0.8	32.8±0.7
4NNSpatial	26.7±1.1	26.6±0.7	29.8±1.5	28.4±0.7	27.8±0.6	31.8±0.8
8NNSpatial	24.6±0.9	24.4±0.7	27.0±1.2	27.3±1.1	27.1±1.0	31.0±0.9
16NNSpatial	17.2±8.6	16.7±8.3	18.8±9.4	25.5±1.0	25.3±1.1	28.8±1.1
1NNCombined	29.1±1.4	29.1±0.8	33.4±1.5	28.7±0.5	28.4±0.7	32.8±0.7
2NNCombined	29.0±1.0	28.8±1.2	33.0±1.6	29.1±0.4	29.0±0.7	33.1±0.5
4NNCombined	28.1±1.1	28.0±0.9	31.8±1.4	28.8±0.2	28.8±0.9	32.7±0.5
8NNCombined	26.7±1.0	26.3±1.0	30.2±1.4	27.9±0.4	27.7±0.4	31.7±0.2
16NNCombined	23.6±1.0	23.4±0.9	26.4±1.4	26.1±0.6	25.9±0.5	29.4±0.7

Table A.35: Final F1-measure obtained with the edge-building methods experiments for the GCN and GAT models for the **STL10** dataset

	GCN			GAT		
Method	F1-measure			F1-measure		
	Test	Val.	Train	Test	Val.	Train
RAG	44.4±1.5	44.2±1.2	47.4±0.6	44.2±1.1	44.4±1.6	46.9±1.3
1NNSpatial	45.1±1.4	45.5±1.2	49.4±1.2	43.8±1.9	44.2±1.1	46.9±1.5
2NNSpatial	44.9±1.4	45.1±1.2	48.7±1.2	44.4±0.2	44.5±1.0	47.3±0.5
4NNSpatial	43.2±1.7	43.3±1.7	46.0±1.2	44.5±0.4	44.5±0.7	47.3±1.0
8NNSpatial	41.2±1.6	41.1±1.5	43.3±1.2	42.4±0.9	43.8±1.1	45.2±0.9
16NNSpatial	36.7±2.1	37.2±2.2	38.5±1.6	40.1±1.1	40.5±1.7	42.2±1.5
1NNCombined	46.2±1.5	46.5±1.4	51.6±1.1	43.8±1.0	44.8±1.6	47.7±0.7
2NNCombined	46.8±1.5	47.0±1.4	52.4±1.2	43.4±0.6	44.5±1.6	47.0±0.8
4NNCombined	45.7±1.7	46.1±1.4	50.1±1.0	43.3±0.5	43.5±2.0	46.7±1.0
8NNCombined	44.2±1.7	44.3±1.2	48.0±0.8	42.6±0.4	42.6±0.6	45.6±1.0
16NNCombined	40.0±2.7	39.6±1.8	42.4±1.6	40.1±0.8	40.4±1.2	42.4±1.5

Table A.36: Final Accuracy obtained with the edge-building methods experiments for the GCN and GAT models for the **MNIST** dataset

Method	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
RAG	90.7±0.6	90.9±0.5	91.4±0.5	92.8±0.6	92.8±0.6	93.3±0.5
1NNSpatial	88.1±0.7	87.8±0.9	89.2±0.8	91.7±1.2	91.7±1.1	92.4±1.1
2NNSpatial	87.4±0.9	87.4±1.1	88.5±1.1	92.3±0.8	92.3±0.9	93.0±0.9
4NNSpatial	89.9±0.3	90.1±0.5	90.7±0.3	92.4±0.6	92.5±0.4	93.1±0.6
8NNSpatial	89.0±0.6	88.8±1.1	89.5±0.8	91.7±1.0	91.8±0.9	92.3±0.8
16NNSpatial	81.8±0.8	81.6±1.1	82.2±0.9	87.6±6.3	87.5±6.1	88.2±6.2
1NNCombined	88.2±0.7	88.3±0.8	89.1±1.0	87.2±1.1	87.1±1.0	87.8±1.0
2NNCombined	88.2±1.0	87.9±0.9	88.9±0.9	87.0±0.9	87.1±0.8	87.4±0.5
4NNCombined	86.9±0.6	86.7±0.9	87.5±0.6	84.4±0.5	84.6±0.8	85.0±0.5
8NNCombined	84.9±0.5	84.7±1.0	85.3±0.8	82.4±1.5	82.4±1.4	82.8±1.4
16NNCombined	79.6±0.8	79.5±1.0	79.9±0.7	80.6±2.4	80.6±2.3	81.0±2.2

Table A.37: Final Accuracy obtained with the edge-building methods experiments for the GCN and GAT models for the **FashionMNIST** dataset

Method	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
RAG	81.8±0.4	81.7±0.5	82.7±0.2	82.4±0.5	82.4±0.7	83.4±0.2
1NNSpatial	81.9±0.2	81.6±0.3	83.4±0.3	82.3±0.3	82.1±0.4	83.4±0.4
2NNSpatial	81.6±0.2	81.6±0.2	82.9±0.2	82.2±0.2	81.8±0.7	83.2±0.4
4NNSpatial	81.7±0.4	81.7±0.4	82.6±0.1	82.6±0.6	82.5±0.5	83.5±0.5
8NNSpatial	79.7±0.4	79.7±0.6	80.4±0.2	81.9±0.7	82.0±0.6	82.8±0.7
16NNSpatial	76.0±0.5	76.3±0.5	76.6±0.4	79.9±0.9	79.9±0.5	80.5±0.8
1NNCombined	83.8±0.6	83.6±0.8	85.1±0.4	83.1±0.3	82.7±0.2	83.8±0.3
2NNCombined	83.5±0.4	83.5±0.4	84.7±0.2	82.7±0.3	82.4±0.6	83.5±0.4
4NNCombined	83.0±0.3	82.9±0.6	84.0±0.3	81.9±0.4	81.6±0.7	82.7±0.3
8NNCombined	81.6±0.3	81.4±0.5	82.4±0.2	81.0±0.3	81.0±0.5	81.9±0.2
16NNCombined	78.9±0.4	78.7±0.9	79.7±0.4	79.3±0.9	79.4±1.0	80.0±1.0

Table A.38: Final Accuracy obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR10** dataset

Method	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
RAG	55.0±0.5	54.5±0.5	57.4±0.5	55.0±0.5	54.8±0.7	56.6±0.5
1NNSpatial	56.7±0.7	56.7±0.8	59.7±0.9	55.2±0.3	54.9±0.4	57.0±0.6
2NNSpatial	57.0±0.5	57.0±0.5	59.6±0.6	54.9±0.4	54.8±1.0	56.8±0.5
4NNSpatial	55.2±0.7	55.0±0.7	57.1±0.9	54.4±0.6	54.3±0.8	56.0±0.6
8NNSpatial	53.1±0.5	52.7±0.5	55.0±0.5	53.6±0.6	53.6±1.1	55.3±0.8
16NNSpatial	48.9±0.4	48.6±0.8	50.2±0.5	52.3±0.8	52.1±0.4	53.8±0.5
1NNCombined	57.6±0.8	57.7±0.5	60.7±0.7	55.2±0.6	55.3±0.8	57.1±0.6
2NNCombined	57.3±0.3	57.3±0.5	60.2±0.1	54.9±0.5	55.2±0.2	56.7±0.6
4NNCombined	56.7±0.3	56.6±0.9	59.2±0.4	54.6±0.8	54.9±0.7	56.3±0.5
8NNCombined	55.0±0.8	54.4±0.7	57.0±0.6	53.9±0.5	53.9±0.6	55.5±0.6
16NNCombined	51.9±0.6	52.0±0.5	53.8±0.5	52.6±0.5	52.5±0.4	54.1±0.3

Table A.39: Final Accuracy obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR100** dataset

Method	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
RAG	27.9±0.8	27.8±0.5	30.9±1.0	29.8±0.6	29.8±0.3	33.3±0.5
1NNSpatial	30.1±0.7	29.6±1.0	33.6±1.4	30.5±0.6	30.3±0.4	34.4±0.5
2NNSpatial	29.1±1.0	29.2±0.8	32.4±1.3	30.6±0.5	30.4±0.6	34.1±0.6
4NNSpatial	28.2±1.0	28.1±0.6	31.2±1.3	29.8±0.5	29.3±0.3	33.1±0.6
8NNSpatial	26.1±1.0	25.9±1.0	28.5±1.3	28.7±0.8	28.6±0.8	32.4±0.8
16NNSpatial	18.6±8.8	18.2±8.6	20.2±9.6	26.7±0.9	26.7±1.1	30.0±1.0
1NNCombined	30.5±1.4	30.4±0.9	34.7±1.6	30.1±0.5	29.8±0.7	34.1±0.6
2NNCombined	30.3±0.9	30.2±1.0	34.3±1.5	30.6±0.3	30.6±0.6	34.4±0.3
4NNCombined	29.2±1.2	29.2±1.0	33.0±1.4	30.2±0.3	30.2±0.7	34.0±0.2
8NNCombined	28.0±0.9	27.6±1.0	31.5±1.3	29.3±0.4	29.1±0.4	33.0±0.2
16NNCombined	25.2±0.9	25.2±0.7	28.0±1.3	27.5±0.6	27.3±0.5	30.8±0.5

Table A.40: Final Accuracy obtained with the edge-building methods experiments for the GCN and GAT models for the **STL10** dataset

Method	GCN Accuracy			GAT Accuracy		
	Test	Val.	Train	Test	Val.	Train
RAG	45.3±1.5	44.9±1.1	48.1±0.5	45.0±1.2	45.0±1.3	47.6±1.1
1NNSpatial	45.8±1.2	46.2±1.1	50.0±1.0	44.8±1.5	45.2±0.7	47.8±1.2
2NNSpatial	45.9±1.0	46.0±1.2	49.5±0.9	45.1±0.3	45.1±1.0	47.8±0.4
4NNSpatial	44.5±1.5	44.6±1.4	47.2±1.0	45.1±0.5	45.0±0.6	47.8±0.9
8NNSpatial	42.4±1.2	42.3±1.3	44.4±0.9	43.3±0.8	44.5±1.1	46.0±0.8
16NNSpatial	37.9±1.2	38.4±2.0	39.7±0.9	41.5±0.8	41.7±1.4	43.4±1.0
1NNCombined	46.8±1.5	47.0±1.2	52.1±1.1	44.6±1.1	45.5±1.5	48.2±0.7
2NNCombined	47.3±1.6	47.4±1.8	52.7±1.1	44.4±0.8	45.4±1.1	47.9±0.4
4NNCombined	46.3±1.8	46.6±1.5	50.7±1.1	44.1±0.3	44.0±1.9	47.3±0.8
8NNCombined	45.1±1.4	45.1±0.9	48.7±0.6	43.5±0.4	43.4±0.5	46.5±0.7
16NNCombined	41.2±2.2	40.9±1.7	43.5±1.3	41.2±0.8	41.3±1.3	43.5±1.4

Table A.41: Final Loss obtained with the edge-building methods experiments for the GCN and GAT models for the **MNIST** dataset

Method	GCN Loss			GAT Loss		
	Test	Val.	Train	Test	Val.	Train
RAG	0.29±0.02	0.29±0.02	0.27±0.01	0.23±0.02	0.23±0.02	0.21±0.02
1NNSpatial	0.37±0.02	0.38±0.02	0.34±0.02	0.27±0.04	0.27±0.04	0.24±0.03
2NNSpatial	0.39±0.02	0.39±0.03	0.36±0.03	0.25±0.02	0.25±0.03	0.22±0.03
4NNSpatial	0.32±0.01	0.31±0.01	0.29±0.01	0.24±0.02	0.24±0.02	0.22±0.02
8NNSpatial	0.35±0.02	0.36±0.03	0.33±0.02	0.27±0.03	0.26±0.03	0.24±0.03
16NNSpatial	0.57±0.03	0.57±0.03	0.56±0.03	0.40±0.19	0.40±0.19	0.38±0.19
1NNCombined	0.38±0.02	0.37±0.03	0.35±0.03	0.41±0.03	0.40±0.03	0.39±0.02
2NNCombined	0.38±0.02	0.38±0.03	0.35±0.03	0.41±0.02	0.41±0.02	0.40±0.02
4NNCombined	0.41±0.02	0.42±0.03	0.40±0.02	0.49±0.01	0.48±0.01	0.47±0.01
8NNCombined	0.48±0.01	0.48±0.04	0.47±0.03	0.55±0.05	0.55±0.05	0.54±0.04
16NNCombined	0.64±0.03	0.64±0.03	0.63±0.02	0.61±0.07	0.61±0.07	0.60±0.06

Table A.42: Final Loss obtained with the edge-building methods experiments for the GCN and GAT models for the **FashionMNIST** dataset

Method	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
RAG	0.48±0.01	0.49±0.01	0.46±0.00	0.48±0.01	0.48±0.01	0.45±0.01
1NNSpatial	0.49±0.00	0.50±0.01	0.45±0.01	0.48±0.01	0.49±0.01	0.45±0.01
2NNSpatial	0.50±0.01	0.50±0.01	0.46±0.01	0.49±0.01	0.49±0.01	0.46±0.01
4NNSpatial	0.49±0.01	0.50±0.01	0.47±0.01	0.47±0.01	0.47±0.01	0.45±0.01
8NNSpatial	0.54±0.01	0.54±0.01	0.52±0.01	0.49±0.02	0.49±0.02	0.47±0.02
16NNSpatial	0.63±0.01	0.63±0.01	0.61±0.01	0.54±0.03	0.54±0.02	0.52±0.02
1NNCombined	0.45±0.01	0.45±0.02	0.40±0.01	0.47±0.01	0.48±0.01	0.44±0.01
2NNCombined	0.45±0.00	0.45±0.01	0.41±0.01	0.47±0.01	0.48±0.01	0.45±0.01
4NNCombined	0.46±0.01	0.46±0.01	0.43±0.00	0.49±0.01	0.50±0.01	0.47±0.01
8NNCombined	0.49±0.01	0.50±0.01	0.47±0.00	0.51±0.01	0.51±0.01	0.49±0.00
16NNCombined	0.56±0.01	0.56±0.01	0.54±0.01	0.56±0.03	0.55±0.03	0.54±0.03

Table A.43: Final Loss obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR10** dataset

Method	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
RAG	1.26±0.02	1.27±0.01	1.20±0.01	1.27±0.01	1.28±0.02	1.22±0.02
1NNSpatial	1.22±0.02	1.22±0.02	1.13±0.02	1.26±0.01	1.26±0.02	1.21±0.02
2NNSpatial	1.22±0.01	1.21±0.01	1.14±0.01	1.27±0.01	1.27±0.02	1.22±0.01
4NNSpatial	1.27±0.02	1.27±0.01	1.20±0.02	1.29±0.02	1.29±0.02	1.24±0.01
8NNSpatial	1.32±0.01	1.33±0.00	1.27±0.01	1.31±0.02	1.32±0.02	1.26±0.02
16NNSpatial	1.44±0.02	1.45±0.02	1.40±0.02	1.34±0.02	1.35±0.01	1.30±0.01
1NNCombined	1.19±0.01	1.20±0.01	1.11±0.02	1.25±0.01	1.26±0.02	1.20±0.02
2NNCombined	1.20±0.01	1.21±0.01	1.12±0.01	1.26±0.02	1.26±0.02	1.21±0.01
4NNCombined	1.22±0.01	1.22±0.02	1.14±0.01	1.26±0.02	1.27±0.01	1.22±0.01
8NNCombined	1.27±0.02	1.28±0.01	1.20±0.01	1.29±0.01	1.30±0.01	1.25±0.01
16NNCombined	1.35±0.01	1.35±0.01	1.29±0.01	1.33±0.01	1.34±0.01	1.29±0.00

Table A.44: Final Loss obtained with the edge-building methods experiments for the GCN and GAT models for the **CIFAR100** dataset

Method	GCN Loss Test	GCN Loss Val.	GCN Loss Train	GAT Loss Test	GAT Loss Val.	GAT Loss Train
RAG	2.94±0.04	2.93±0.02	2.76±0.05	2.86±0.03	2.86±0.02	2.66±0.03
1NNSpatial	2.84±0.04	2.83±0.03	2.63±0.06	2.82±0.02	2.80±0.01	2.61±0.03
2NNSpatial	2.89±0.05	2.88±0.04	2.70±0.06	2.82±0.02	2.81±0.02	2.62±0.02
4NNSpatial	2.93±0.04	2.91±0.04	2.75±0.06	2.87±0.02	2.86±0.02	2.68±0.03
8NNSpatial	3.06±0.05	3.04±0.05	2.89±0.07	2.92±0.04	2.90±0.03	2.72±0.05
16NNSpatial	3.49±0.56	3.48±0.56	3.38±0.61	3.02±0.04	3.01±0.04	2.82±0.06
1NNCombined	2.80±0.05	2.78±0.04	2.58±0.07	2.84±0.02	2.82±0.02	2.62±0.03
2NNCombined	2.81±0.05	2.79±0.05	2.59±0.07	2.83±0.01	2.82±0.02	2.61±0.02
4NNCombined	2.87±0.05	2.85±0.03	2.67±0.07	2.85±0.00	2.83±0.02	2.64±0.01
8NNCombined	2.94±0.04	2.92±0.03	2.74±0.07	2.89±0.02	2.88±0.01	2.69±0.02
16NNCombined	3.10±0.03	3.08±0.02	2.92±0.06	2.98±0.02	2.97±0.01	2.79±0.02

Table A.45: Final Loss obtained with the edge-building methods experiments for the GCN and GAT models for the **STL10** dataset

Method	GCN			GAT		
	Loss			Loss		
	Test	Val.	Train	Test	Val.	Train
RAG	1.50±0.02	1.49±0.02	1.42±0.01	1.49±0.02	1.47±0.02	1.42±0.02
1NNSpatial	1.47±0.03	1.44±0.03	1.35±0.03	1.49±0.03	1.46±0.03	1.42±0.03
2NNSpatial	1.46±0.02	1.44±0.03	1.36±0.02	1.49±0.01	1.47±0.02	1.41±0.01
4NNSpatial	1.51±0.04	1.50±0.03	1.44±0.03	1.50±0.02	1.48±0.02	1.42±0.01
8NNSpatial	1.56±0.02	1.55±0.03	1.50±0.02	1.53±0.02	1.50±0.02	1.46±0.02
16NNSpatial	1.65±0.03	1.65±0.04	1.61±0.01	1.58±0.02	1.56±0.03	1.53±0.03
1NNCombined	1.44±0.03	1.42±0.01	1.30±0.03	1.48±0.02	1.46±0.02	1.40±0.02
2NNCombined	1.42±0.02	1.41±0.02	1.29±0.03	1.48±0.01	1.46±0.02	1.40±0.01
4NNCombined	1.45±0.03	1.42±0.02	1.33±0.03	1.51±0.01	1.48±0.03	1.43±0.01
8NNCombined	1.49±0.03	1.47±0.02	1.39±0.01	1.52±0.01	1.52±0.01	1.45±0.01
16NNCombined	1.59±0.03	1.57±0.02	1.52±0.03	1.57±0.02	1.55±0.03	1.52±0.04