UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MATHEUS VINÍCIUS TODESCATO

An image classification approach based on graph convolutional networks and patch-based multiscale feature graphs

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Joel Luís Carbonera

Porto Alegre
July 2023

CIP — CATALOGING-IN-PUBLICATION

Todescato, Matheus Vinícius

An image classification approach based on graph convolutional networks and patch-based multiscale feature graphs / Matheus Vinícius Todescato. – Porto Alegre: PPGC da UFRGS, 2023.

80 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor: Joel Luís Carbonera.

1. Image Classification. 2. Graph Convolutional Network. 3. Transfer Learning. 4. Feature Extraction. 5. Multiscale. I. Carbonera, Joel Luís. II. Título.

“You don’t have to be great to start,

but you have to start to be great.”

— Zig Ziglar

AGRADECIMENTOS

Primeiramente agradeço a Deus pelas oportunidades colocadas em meu caminho.

Gostaria de agradecer ao meu orientador, professor Dr. Joel Luís Carbonera, pela oportunidade que me deu ao me aceitar como orientando. Agradeço pela paciência e auxílio durante minha pesquisa e pela parceria de sempre. Espero que possamos continuar compartilhando conhecimento e conquistas nos próximos anos.

Agradeço aos meus colegas de laboratório, especialmente ao Dennis Balreira e Luan Garcia, pelo apoio e auxílio durante meu mestrado.

Agradeço a minha família, por ser a melhor do mundo, sempre me incentivando e me ajudando.

Aos meus pais, muito obrigado por tudo. Todas minhas conquistas são de vocês também, pois não seria nada sem a força e o apoio que vocês me dão.

Agradeço aos meus amigos, que me acompanharam nessa jornada e, com toda certeza, fizeram o caminho ser menos árduo e muito mais divertido.

Agradeço ao CNPq e a Petrobras por garantirem o suporte financeiro para minha pesquisa.

ABSTRACT

Deep learning architectures have demonstrated impressive results in image classification in the last few years. However, applying sophisticated neural network architectures in small datasets remains challenging. In this context, transfer learning is a promising approach for dealing with this scenario. Generally, the available pre-trained architectures adopt a standard fixed input, which usually implies resizing and cropping the input images in the preprocessing phase, causing information loss. Besides, images present visual features in different scales in real-world scenarios, and most common approaches do not consider this fact. In this work, we propose an approach that applies transfer learning for dealing with small datasets and leverages visual features extracted by pre-trained models from different scales. We based our approach on graph convolutional networks (GCN) that take graphs representing the images in different scales as input and whose nodes are characterized by features extracted by pre-trained models from regular image patches of different scales. Since GCN can deal with graphs with different numbers of nodes, our approach can deal naturally with images of heterogeneous sizes without discarding relevant information. We evaluated our approach in two datasets: a set of geological images and a publicly available dataset, presenting characteristics that challenge traditional approaches. We tested our approach by adopting three different pre-trained models as feature extractors: two efficient pre-trained CNN models (DenseNet and ResNeXt) and one Vision Transformer (CLIP). We compared our approach with two conventional approaches for dealing with image classification. The experiments show that our approach achieves better results than the conventional approaches for this task.

Keywords: Image Classification. Graph Convolutional Network. Transfer Learning. Feature Extraction. Multiscale.

Uma abordagem para classificação de imagens baseada em redes convolucionais de grafos e grafos de características multiescala baseados em patches

RESUMO

As arquiteturas de aprendizado profundo demonstraram excelentes resultados na classificação de imagens nos últimos anos. No entanto, a aplicação de arquiteturas de redes neurais sofisticadas em pequenos conjuntos de dados continua sendo um desafio. Nesse contexto, a aprendizagem por transferência é uma abordagem promissora para lidar com esse cenário. Geralmente, as arquiteturas pré-treinadas disponíveis adotam uma entrada fixa padrão, o que geralmente implica em redimensionar e recortar as imagens de entrada na fase de pré-processamento, causando perda de informações. Além disso, no mundo real, as imagens apresentam características visuais em diferentes escalas, e as abordagens mais comuns não consideram esse fato. Neste trabalho, propomos uma abordagem que aplica transferência de conhecimento para lidar com pequenos conjuntos de dados e aproveita características visuais extraídas por modelos pré-treinados de diferentes escalas. Baseamos nossa abordagem em redes convolucionais de grafos (GCN) que recebem como entrada grafos que representam as imagens em diferentes escalas e cujos nós são caracterizados por características extraídas por modelos pré-treinados de partes regulares de imagens em diferentes escalas. Como o GCN pode lidar com grafos com diferentes números de nós, nossa abordagem pode lidar naturalmente com imagens de tamanhos heterogêneos sem descartar informações relevantes. Avaliamos nossa abordagem em dois conjuntos de dados: um conjunto de imagens geológicas e um conjunto de dados disponíveis publicamente, ambos apresentando características que desafiam as abordagens tradicionais. Testamos nossa abordagem adotando três modelos pré-treinados diferentes como extratores de características: dois modelos eficientes de CNN pré-treinados (DenseNet e ResNeXt) e um Vision Transformer (CLIP). Comparamos nossa abordagem com duas abordagens convencionais para lidar com a classificação de imagens. Os experimentos mostram que nossa abordagem alcança melhores resultados do que as abordagens convencionais para esta tarefa.

Palavras-chave: Classificação de Imagens. Rede de Convolução de Grafos. Transferência de Aprendizado. Extração de Características. Multiescala.

LIST OF ABBREVIATIONS AND ACRONYMS

UFRGS	Universidade Federal do Rio Grande do Sul
CNN	Convolutional Neural Network
ViT	Vision Transformer
TL	Transfer Learning
FCN	Fully Convolutional Network
GNN	Graph Neural Network
GCN	Graph Convolutional Network
ML	Machine Learning
NN	Neural Network
DL	Deep Learning
NLP	Natural Language Processing
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
FE	Feature Extractor
MGS	Multiscale Graph Set
SS	Scale Set
GPM	Graph Processing Module
FCL	Fully Connected Layer
SF	Simple Features
MLP	Multilayer Perceptron
RAG	Retrieval-Augmented Generation
GAT	Graph Attention Networks
ReLU	Rectified Linear Unit

LIST OF FIGURES

Figure 2.1 Structure of a neural network. A neural network is parameterized by its weights and its output quality is measured by a loss function. Then, the loss score is used to adjust the weights. 19

Figure 2.2 LeNet-5 model diagram. This architecture introduced the essential components of modern CNN, composed of convolution, subsampling (pooling), and fully connected layers. 21

Figure 2.3 Example of a 2-D convolution. The multiplication between the input matrix and the kernel matrix generates a new representation of the data. 21

Figure 2.4 Example of max pooling layer operation. 22

Figure 2.5 Vision Transformer overview. The process consists of splitting the image into fixed-size patches, linearly embedding each of them, adding position embeddings, and feeding the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, an extra learnable "classification token" is added to the sequence. 23

Figure 2.6 Two traditional GCN architectures. Gconv denotes a convolutional graph layer, MLP is a multilayer perceptron, and readout is the layer that performs an operation that summarizes the output (e.g., global pooling). 25

Figure 2.7 Schema illustrating how global max pooling works. 26

Figure 2.8 Traditional Learning vs. Transfer Learning. 27

Figure 2.9 DenseNet architecture. The architecture comprises a stack of dense blocks that use multiple connections called "short paths" to carry residual information between layers. 29

Figure 2.10 ResNet traditional learning block vs. ResNeXt learning block with cardinality = 32. Keeping the connection of residual information and with a larger set of transformations per block made it possible to increase the performance without going deeper. 30

Figure 2.11 CLIP approach to learning transferable visual models from natural language supervision. First, it presents a contrastive pre-training with image/text pairs feeding a text encoder and an image encoder. Then, using a dataset from label text makes it possible to match the image with the text and use it for zero-shot prediction or feature extraction. 30

Figure 2.12 Early stopping applied in a learning process. 32

Figure 2.13 Schema illustrating a 5-fold cross-validation. 33

Figure 2.14 Confusion matrix for a binary classification problem. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. 34

Figure 4.1 A simplified example of how Algorithm 1 works applied to an image from the Stanford Cars dataset (KRAUSE et al., 2013). The algorithm starts by considering the original image at right. After, the algorithm resizes the image using the value of $N = 4$ (in this example). After, the algorithm extracts patches from the image using the stride value $S = 1$ (without overlapping). Finally, the algorithm builds the resulting graph using the features extracted from each patch (FP) as nodes, and considering the spatial neighborhood of each patch for creating edges. 42

Figure 4.2 An example of the proposed approach for image classification. Given an image, we apply Algorithm 1 with different values of N (1, 2 and 3, in this example), generating a multiscale graph set $MGS(\{1,2,3\})$, with three patch-based feature graphs representing the image at three different scales. After, we train a multi-input graph convolutional architecture that takes as input the patch-based feature graphs in $MGS(\{1,2,3\})$ for predicting the image class. 46

Figure 5.1 Examples of images illustrating some classes of the ontology: (a)ternary diagram, (b) satellite image, (c) geological map, (d) profile, and (e) geological cross-section. 48
Figure 5.2 Representation of the number of instances for each class of the geological dataset. 48
Figure 5.3 Boxplots representing the statistical properties of the images in the dataset of geological images, focusing on their height, width, and aspect ratio. 49
Figure 5.4 Distribution of the images according to their dimensions in the geological dataset. 50
Figure 5.5 Representation of the statistical properties of the images in the Stanford Cars dataset through boxplots, focusing on their height, width, and aspect ratio (as Width/Height). 51
Figure 5.6 Some images from the Stanford cars dataset. The cars are presented in different positions and scenarios, and the images are of different sizes. 52
Figure 5.7 Distribution of the images according to their dimensions in the stanford cars dataset. 53
Figure 5.8 Macro F1 results in geological dataset by varying the stride value (10%, 25%, 50% and 100%) for each MGS (1-5). 56
Figure 5.9 Macro F1 results in Stanford Cars dataset by varying the stride value (10%, 25%, 50% and 100%). 57
Figure 5.10 Learning curves that show the loss during the training and validation phase, using CLIP in the folds with the best result. 60
Figure 5.11 Evolution of the classification performance in the geological dataset as scales are added in the graph set. Each bar represents the F1 measure obtained in the worst and best folds in the cross-validation process and each point represents the average considering all folds. 63
Figure 5.12 Evolution of the classification performance in the Cars dataset as new graph scales is added in the graph set. Each bar represents the F1 measure obtained in the worst and best folds in the cross-validation process and each point represents average considering all folds. 64

LIST OF TABLES

Table 2.1 Pre-trained models information....28

Table 5.1 Experimental results using different stride values for different scales in the geological dataset. CLIP (RADFORD et al., 2021) was used as a feature extractor. We group the results according to the scale size (parameter N) and highlight in green the best results and in red the worsts....54

Table 5.2 Experimental results using different stride values for different scales in the Stanford Cars dataset. CLIP (RADFORD et al., 2021) was used as a feature extractor. We group the results according to the scale size (parameter N) and highlight in green the best results and in red the worsts....55

Table 5.3 Number of patches generated for each stride value and for each value N in an image of 800x600 pixels....55

Table 5.4 Comparison of the metrics achieved by each experiment configuration in the geological dataset. Each row represents a different configuration. We divided these setups in the table according to the input information (Column 4) and the use of transfer learning (Column 3). Column 4 *Raw Image (lossy)* indicates that raw image is used as an input with resize and center crop to 224x224 input size. *SF* indicates that *Simple Features* are used as input. *Lossless* indicates that the original image information is used as input and only images with the smallest dimension small than 224 pixels are resized, whereas *lossy* indicates that resizing and cropping are performed, discarding information. Lines 6-20 show the results achieved by our approach with different scales, while lines 21-32 present the results achieved when combining different scales....58

Table 5.5 Achieved results in the Stanford Cars dataset. This table adopts the same notation adopted in Table 5.4....59

CONTENTS

1 INTRODUCTION **12**
2 THEORETICAL BACKGROUND **16**
2.1 Image Classification **16**
2.2 Machine Learning **17**
2.2.1 Neural Networks 18
2.2.1.1 Convolutional Neural Networks 20
2.2.1.2 Transformers 23
2.2.1.3 Graph Convolutional Networks 24
2.2.2 Transfer Learning 26
2.2.3 Pre-trained models 28
2.2.4 Early stopping 30
2.2.5 K-fold Cross-validation 31
2.2.6 Performance metrics 33
3 RELATED WORKS **36**
4 PROPOSED APPROACH **40**
4.1 Patch-based graph for image representation **40**
4.2 Graph convolutional network for multiscale image classification **44**
5 EXPERIMENTS **47**
5.1 Datasets **47**
5.1.1 Geological dataset 47
5.1.2 Stanford cars dataset 49
5.2 Results **49**
5.2.1 Effects of the stride value 52
5.2.2 Evaluation of the proposed approach 54
6 CONCLUSION **65**
REFERENCES **67**
APPENDIX A — RESUMO EXPANDIDO **73**
A.1 Introdução **73**
A.2 Metodologia **76**
A.3 Resultados e Conclusão **78**

1 INTRODUCTION

Nowadays, social networks, corporate systems, and several applications on the web generate large amounts of data every second. Due to the availability of this big data, managing and retrieving the relevant data for supporting the tasks of interest becomes a challenge. This scenario is also present within companies, which spend considerable resources trying to solve this problem. In this context, dealing with images is an even more challenging task. The main reason is the absence of explicit meaning associated with images, hindering the retrieval of this kind of data through search queries. A common approach for dealing with this scenario involves annotating images with semantic tags to allow searching and retrieving them (HOLLINK et al., 2003). However, manual annotation is laborious and not feasible in contexts in which big data is prevalent, since this process requires significant human resources. Thus, automatic approaches for image classification could be of great value for automatically labeling images in this scenario (WONG; LEUNG, 2008; ZHANG; ISLAM; LU, 2012), allowing the retrieval of this data through conventional queries in a subsequent step. Deep learning techniques are natural candidates for automatically labeling large databases of images in this scenario.

In the last decade, Convolutional Neural Networks (CNN) (SZEGEDY et al., 2015; TAN; LE, 2019; KRIZHEVSKY; SUTSKEVER; HINTON, 2012) and, most recently, Vision Transformers (ViTs) (DOSOVITSKIY et al., 2020) significantly improved the performance on image classification tasks. Researchers have applied these techniques in several distinct domains (SLADOJEVIC et al., 2016; DUNG et al., 2019; ABBAS; ABDELSAMEA; GABER, 2021; HONG et al., 2020). Despite the excellent results, sophisticated neural network architectures generally demand considerable training data to achieve good performances (ZHU et al., 2021). Transfer learning (TORREY; SHAVLIK, 2010) has emerged as a promising approach to deal with this problem (LIANG; ZHENG, 2020; HORRY et al., 2020), since this strategy allows taking advantage of the knowledge learned from significant amounts of data for dealing with tasks in which only small data is available.

In general, when transfer learning is applied in the context of neural network architectures, images used for feeding the pre-trained neural networks are standardized to match the input requirements of these architectures (LECUN et al., 1998; KRIZHEVSKY; HINTON et al., 2009). In general, this standardization process discards critical parts of the image or changes its aspect ratio, causing information loss or introducing noise. However, datasets of images with different sizes and aspect ratios are common in real-world settings. Some studies (ANTHIMOPOULOS et al., 2016; ARAÚJO et al., 2017) address these issues by adopting, for instance, patch-based approaches that deal with images by considering different patches extracted from them and using different strategies for classifying the images according to the information of their patches. Other approaches also adopt *Fully Convolutional Network* (FCN) (WANG et al., 2021; ZHUANG et al., 2021), which can take images of different sizes as input, to avoid discarding relevant image information.

Recently, some works (ZHANG; ZOU; ZHANG, 2022; BAE et al., 2022) have been exploring graph neural networks (GNN) for image classification. These approaches involve representing images as graphs, where nodes represent pixels or sets of adjacent pixels (called superpixels), and edges represent spatial relations between those pixels or superpixels. Each node is characterized by features of each pixel or statistical properties of the set of pixels that constitute each superpixel (ZHANG; ZOU; ZHANG, 2022). Some GNNs, such as graph convolutional networks (GCN), can deal with graphs with heterogeneous numbers of nodes and edges, allowing them to handle images of different sizes.

Another essential characteristic of real-world image datasets is the presence of visual features in different scales of analysis and visual features that are apparent only in some scales of analysis. Recently, some works have dealt with these scenarios by proposing deep learning approaches that leverage different visual features in multiple scales in different ways (CHEN; FAN; PANDA, 2021; MOHAN; VENKATESAN, 2020).

This work focuses on image classification approaches that can deal with real-world image datasets that generally have small amounts of samples, where images can have different sizes and whose visual features can be apparent in different scales.

In this work, we hypothesize that minimizing information loss and considering the visual features of images at multiple scales can improve classification. Based on this hypothesis, we propose an approach based on multiscale GCNs for image classification. In this approach, we build different graphs to represent each image in different scales. For building the graph of each scale of the original image, we first split the images into a set of regular patches. We represent each patch as a node in a graph whose edges represent the spatial neighborhood of the patches in the image. Finally, each node is characterized by features extracted by pre-trained neural network models (used as feature extractors) from the respective patch represented by the node. Once the set of graphs representing the image at multiple scales is built, we use this set of graphs as input for training a multi-input graph convolutional network.

Thus, our approach originally combines the following features:

- The approach adopts a graph representation of images, where nodes represent regular patches extracted from images and edges represent the spatial neighborhood of each patch.
- The features of each node are extracted by pre-trained neural network models from the regular patches, allowing our approach to leverage the knowledge represented in those pre-trained models for dealing with smaller datasets.
- The approach considers visual features that are apparent in different scales of analysis since we build different graphs from each image, where each one represents the features of the image that are apparent in some specific scale.
- The approach can be applied in datasets with images of different sizes and is designed to preserve as much information as possible from each image.

We evaluated our method in a dataset of geological images and a publicly available dataset. These datasets challenge traditional approaches since their images have heterogeneous sizes and present some features in different scales. We measure our approach's performance using three different pre-trained models as feature extractors: DenseNet (HUANG et al., 2017), ResNeXt (XIE et al., 2017)) and CLIP (RADFORD et al., 2021). We compare our method with some well-established approaches for dealing with images of different sizes. Our experiments suggest that the proposed approach outperforms the considered alternatives. Furthermore, our work demonstrates that our approach achieves better performance by using the CLIP pre-trained model as a feature extractor than by using DenseNet or Resnext pre-trained models.

This dissertation was developed in the context of the project "Parameterization of the geometry and architecture of reservoirs", which is a cooperation between UFRGS and Petrobras. The project studies the types of geometry of sedimentary deposits to assist geologists in proposing a uniform conceptual data model and an information system, supporting the development of a description system and parametric queries. Within this context, the geologist needs to access previously produced maps, photographs, and many other different types of images to support and explain the process of interpretation (ABEL et al., 2019). Thus, one of the project's goals is to develop an *image retrieval system* for the petroleum industry. Although the approach proposed in this work is designed to deal with images from different domains, it was primarily developed to classify geological images, in order to support the subsequent image retrieval.

This work is structured as follows. In Chapter 2, we describe the theoretical background of our work. Chapter 3 presents a literature review, where we discuss the related works. Chapter 4 presents a detailed discussion of the proposed approach for image classification. Chapter 5 presents our experiments, detailing the datasets, the methodology, and the results. Finally, Chapter 6 presents our conclusions.

2 THEORETICAL BACKGROUND

This work proposes a machine learning approach for image classification. Thus, in this Section, we present the theoretical background related to image classification (Section 2.1) and machine learning (Section 2.2).

2.1 Image Classification

Image classification is a task that involves assigning a label, or more than one, to an input image based on its contents, in a way that the assigned label classifies the image as a whole. The goal is to correctly identify the image's class or category, such as "cat" or "dog," based on its visual features (SZELISKI, 2022).

In the past, several image classification techniques have been proposed such as subpixel, per-field, and knowledge-based classification. Traditional per-pixel classifiers typically develop a signature by combining the spectra of all training-set pixels for a given feature, working well for remote sensing images (LU; WENG, 2007). In a more general context, the simplest method for classification was the *bag of words* (also known as *bag of features*), where the distribution (histogram) of "visual words" (i.e., features extracted by methods such as *LaPLace*) of the analyzed image was computed and compared with the distribution of the training images (SZELISKI, 2022). From this, different methods have emerged for comparing two feature vectors, such as the use of distance calculations between vectors (GRAUMAN; DARRELL, 2007). Other feature extraction methods were developed and combined with machine learning methods to perform the classification, such as using hierarchies of dense feature transforms inspired by biological (visual cortical) processing combined with support vector machines (SVMs) for final classification (MUTCH; LOWE, 2008).

In recent years, due to the increase in computational power, improvement of techniques algorithms, and the development of large databases of annotated images, we can notice an increasing adoption of machine learning methods for image classification.

2.2 Machine Learning

Machine Learning (ML) is an area of Artificial Intelligence that aims to develop systems that improve their performance through experience (RUSSELL, 2010). For this, machine learning algorithms extract patterns from raw data to build models that can be used in later steps to solve tasks.

The algorithm's performance in this process depends heavily on the quality of the data and its representation (ALPAYDIN, 2020). In ML, we represent data as sets of features representing different aspects of each sample. Choosing the right set of features that suitably represent the data significantly affects the performance of the algorithms (GOODFELLOW; BENGIO; COURVILLE, 2016).

In addition to features, an important characteristic of data used in ML is whether it contains labels or not. From this information, we know the type of feedback the model will access, determining which type of learning algorithm will be used. These are the main types of learning (GOODFELLOW; BENGIO; COURVILLE, 2016; RUSSELL, 2010):

- Unsupervised learning: there is no explicit feedback provided from the data (no labels).
- Semi-supervised learning: there are both labeled and unlabeled data, and it is necessary to use the feedback from the labeled data to deal with the others.
- Reinforcement learning: there is a feedback loop between the learning system and its experience, where the algorithm interacts with the environment.
- Supervised learning: all data are labeled, and the model learns by observing the input and output pairs (*training data*), being able to compare the actual output and the one predicted by the model. This learning process allows us to infer data labels never seen in the training process (*test data*).

Supervised learning is adopted in two different kinds of tasks: classification and regression. In the classification task, the algorithm needs to infer which category (chosen from a set of discrete labels) classifies a given input (GOODFELLOW; BENGIO; COURVILLE, 2016). For example, learning how to classify images of dogs and cats can be viewed as a classification task.

In contrast, regression tasks involve predicting a continuous numerical value based on a given input. For instance, predicting future prices in the stock market is a common example of a regression task (GOODFELLOW; BENGIO; COURVILLE, 2016). In both types of tasks, the goal is to learn a function that maps a given input to the appropriate output. However, the main difference between classification and regression tasks is the nature of the output: classification tasks have a discrete output, representing a category, while regression tasks have a continuous output, representing a numerical value.

Since our work deals with image classification tasks, we characterize this problem as a classification task with supervised learning. Over the past few decades, various approaches have been developed in the field of machine learning to tackle classification tasks using supervised learning. Neural networks have emerged as a powerful tool and have demonstrated remarkable performance gains in different tasks over the years. In fact, in several benchmarks, they have even surpassed human accuracy (WANG et al., 2019). Due to this, in this work, we developed an approach based on neural networks. In Section 2.2.1, the main aspects associated with neural networks will be discussed.

2.2.1 Neural Networks

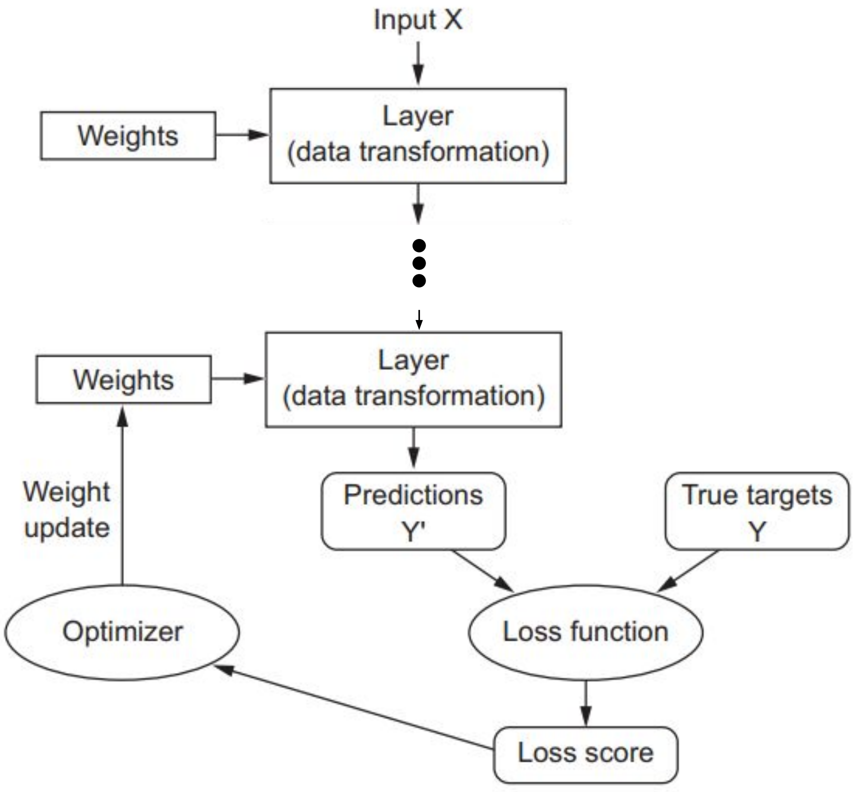
The neural activity of a human being consists of electrochemical activity in the network of brain cells called neurons. Artificial Neural Networks (known simply as Neural Networks) are machine learning approaches initially inspired by this neural process (RUSSELL, 2010). Computationally, a Neural Network (NN) consists of a collection of connected processing units organized in layers and its properties are determined by its topology and the properties of its basic units, the "neurons" (RUSSELL, 2010). Modern Deep Learning involves combining these neurons into layers and combining tens or hundreds of successive layers of these basic processing units. Each layer of a NN transforms the data based on its weights, whose values are adjusted during the training process, based on the network performance feedback. In this context, learning means finding weight values that will make the network correctly map example inputs to their associated targets. Thus, after a suitable training process, a neural network represents a function that approximates the function that maps inputs to the correct outputs in a given task.

Formally, we can define the propagation of the data representation across the network as follows (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$H^{i+1} = \sigma(W^i H^i + B^i) \tag{2.1}$$

The Equation 2.1 represents the forward pass of a neural network, where H is the feature representation at layer $i+1$, σ is the activation function, W are the weights at layer i, H is the feature representation at layer i and B is the bias at layer i.

Figure 2.1 – Structure of a neural network. A neural network is parameterized by its weights and its output quality is measured by a loss function. Then, the loss score is used to adjust the weights.



Source: adapted from (CHOLLET, 2021).

Figure 2.1 shows the workflow of a neural network. In each layer, the data is transformed based on the weights assigned to the activation functions. The loss function computes a distance score from the true target to the prediction provided by the network for a given input. The function calculates the loss score, which is used by the *backpropagation* algorithm to propagate updates on the link's weights between adjacent layers, from the output layer to the input layer.

Initially, all weights in the network are randomly assigned, but during the training process, the weights are updated and adjusted in the right direction as the network processes data. The *gradient descent* algorithm performs this process of minimizing the errors, by updating the weights in the opposite direction of the function's gradient at the current point since this is the direction of the steepest descent. In this process, the *learning rate* defines the step size at each training loop, used by the algorithm to traverse the gradient towards the local minimum point.

Another essential concept in this context is the notion of an epoch, which represents a certain number of iterations, summing up the loss from those iterations and then updating the weights. Due to this strategy, one may process the loss without applying the model to all training samples.

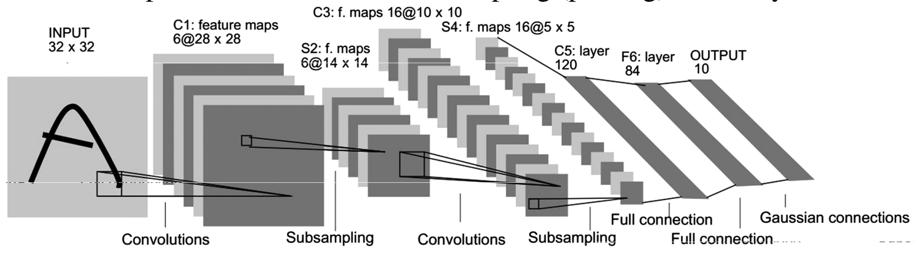
In the past, in the context of image classification to use neural networks, a previous step of feature engineering was required, to perform feature extraction and selection. This is due to the fact that extracting high-level features from images is not trivial since the information, in this case, is pixels. So in this step, the original images went through processing steps in which they were converted into sets of features that represent significant aspects of each image and are more suitable to use as inputs to a neural network and then carry out the learning process in the network.

In the last decade, with the advancement of deep learning (DL) techniques, approaches that can automatically learn appropriate representations from raw data have been developed, eliminating the need for feature engineering. These models have a data representation strategy emphasizing learning on successive layers, increasingly meaningful representations (GOODFELLOW; BENGIO; COURVILLE, 2016). Convolutional neural networks (CNNs) and more recently, transformer architectures, have emerged as prominent techniques in the field of deep learning. Besides that, in recent years, several works have also applied graph convolutional networks in computer vision tasks. We discuss the main concepts related to convolutional neural networks, transformers, and graph convolutional networks in the following subsections.

2.2.1.1 Convolutional Neural Networks

CNNs are NNs whose main characteristic is the adoption of convolution layers, which define trainable convolutional filters and can process data with grid-like topology (e.g., images). Convolution is a specialized kind of operation and the CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers (GOODFELLOW; BENGIO; COURVILLE, 2016). These networks have been widely used for visual tasks involving images due to the spatial nature of the convolution operation. The concepts underlying this kind of network were proposed for the first time in the 60’s (HUBEL; WIESEL, 1962). In 1998, in (LECUN et al., 1998) the authors proposed "LeNet-5", the network that introduced the essential components of the modern deep CNN. Figure 2.2 shows this architecture, which is essentially composed of convolution, subsampling (pooling), and fully connected layers.

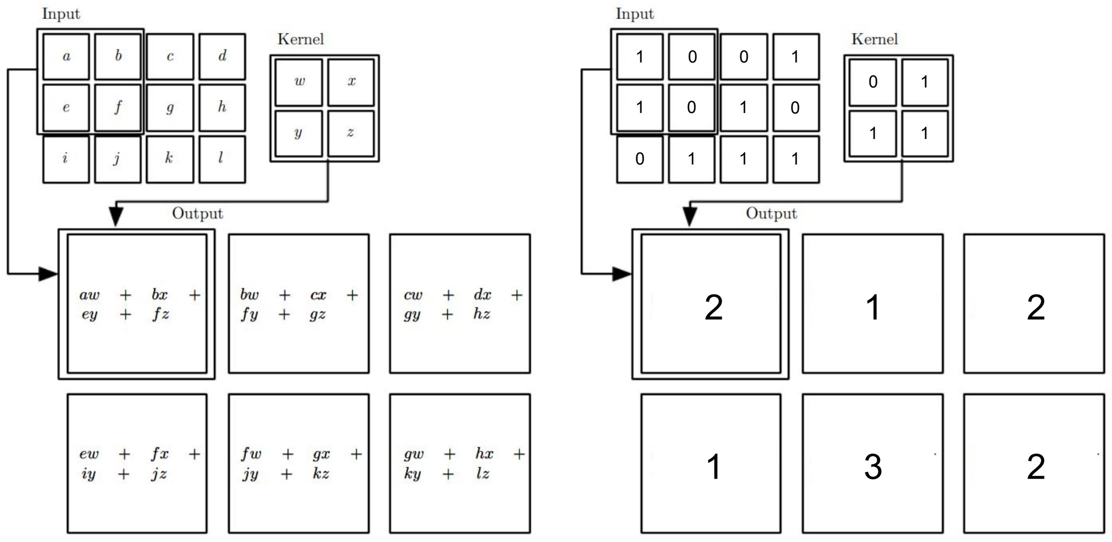
Figure 2.2 – LeNet-5 model diagram. This architecture introduced the essential components of modern CNN, composed of convolution, subsampling (pooling), and fully connected layers.



Source: (LECUN et al., 1998).

The output of a convolution layer consists of multiple feature maps generated by convolution (i.e., dot product) of the *convolution kernel* with the input signal. The convolution kernel is an $N \times N$ weight matrix that slides over the data (e.g., image pixels) performing convolution. Figure 2.3 illustrates an example of the 2-D convolution considering an input of size 3×4, a kernel of size 2×2 and stride 1.

Figure 2.3 – Example of a 2-D convolution. The multiplication between the input matrix and the kernel matrix generates a new representation of the data.

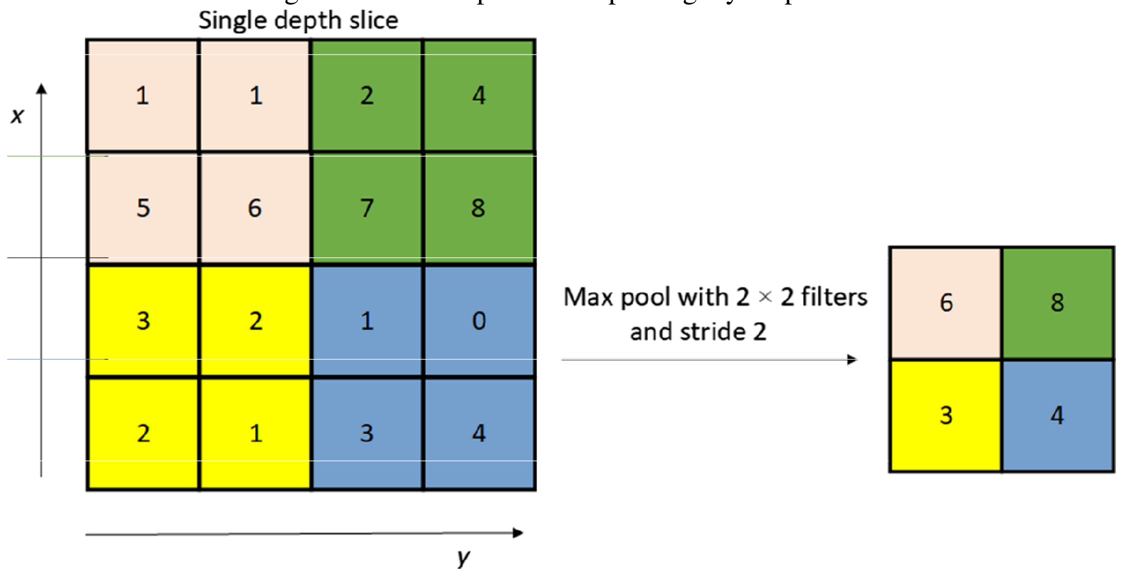


Source: adapted from (GOODFELLOW; BENGIO; COURVILLE, 2016).

Linear transformations limit the output's hypothesis space, and adding more layers that use linear activation functions would not extend this space. So it is necessary to apply non-linear activation functions to access a much richer hypothesis space that would benefit deep representations (CHOLLET, 2021). An example is the rectified linear activation (ReLU) function, the most popular non-linear activation on DL that will output the input directly if it is positive. Otherwise, it will output zero (NAIR; HINTON, 2010).

The pooling layer applies a function that replaces the NN output at a particular location with a summary statistic of the nearby outputs. Pooling helps to make the representation become approximately invariant to small translations of the input. This means that if we translate the input by a small amount, the values of most of the pooled outputs do not change (GOODFELLOW; BENGIO; COURVILLE, 2016). Pooling also can decrease the spatial size of the data and reduce the number of parameters in the network. One of the most common pooling methods is known as max pooling. In this method, we generate a downsampled feature map calculating the maximum value for the region of the original feature map. Figure 2.4 shows an example of how max pooling works, where 2×2 filters are applied with a stride 2, downsampling a 4×4 feature map to a 2×2 feature map, keeping the maximum value of each region.

Figure 2.4 – Example of max pooling layer operation.



Source: adapted from (WANG et al., 2019).

The stack of convolution blocks provides a way to extract different kinds of features from images in a hierarchical way. The first layers extract lower-level features, such as edges, endpoints, and corners. Then the higher layers extract higher-level features from processing these lower-level features (GOODFELLOW; BENGIO; COURVILLE, 2016). These high-level features, in general, can represent semantically meaningful features for a given task, such as human faces, for example.

Traditional CNNs have as their last layer a fully connected layer, which takes as input the features extracted by the previous convolutional blocks and provides an output that represents the classification of the input image.

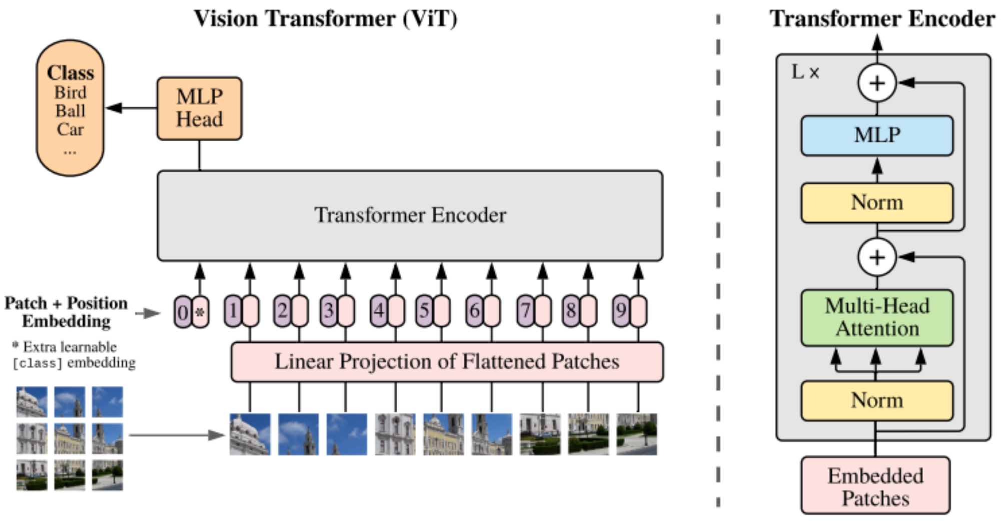
New CNN architectures emerge quickly in the last few years. The modern models have brought advances in the performance of image classification, achieving results superior to human beings' accuracy in several benchmarks (WANG et al., 2019). In addition to CNNs, more recently, after great results in textual tasks, the transformers architecture was adapted and achieved great results in computer vision tasks.

2.2.1.2 Transformers

Since their original appearance, Transformers have been widely used for natural language processing (NLP) and achieved great success in many fields of artificial intelligence. Transformer was initially proposed as a sequence-to-sequence model, using an encoder and decoder, for machine translation (SUTSKEVER; VINYALS; LE, 2014). This new type of architecture was highlighted in NLP tasks based solely on *self-attention mechanism*, replacing the recurrence and convolution mechanisms, weighting the significance of each part of the input (VASWANI et al., 2017).

Transformers began to stand out in computer vision using the standard architecture directly to images (DOSOVITSKIY et al., 2020), which are split into patches and provide the sequence of linear embeddings of these patches as input. Then, image patches are treated in the same way as tokens (words) in NLP applications. This architecture called *Vision Transformer* (ViT), is represented in Figure2.5.

Figure 2.5 – Vision Transformer overview. The process consists of splitting the image into fixed-size patches, linearly embedding each of them, adding position embeddings, and feeding the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, an extra learnable “classification token” is added to the sequence.



Source: (DOSOVITSKIY et al., 2020).

Several other ViT were developed based on this basic architecture, taking advantage of the computational efficiency and scalability of this architecture. These approaches reached a new level in large-scale image recognition (LIN et al., 2022).

2.2.1.3 Graph Convolutional Networks

In image analysis applications, one can consider images as functions on the Euclidean space (plane), sampled on a grid. However, several applications in different domains involve dealing with non-euclidean data, that is, data whose underlying structure is a non-Euclidean space (BRONSTEIN et al., 2017). Some examples include social networks, sensor networks, molecules, regulatory networks in genetics, etc. Graphs constitute a common way of representing this kind of data, including complex relationships and interdependency between objects. In this context, graph neural networks(SCARSELLI et al., 2008; WU et al., 2022) constitute a family of neural network architectures that are designed for taking graphs as inputs, allowing to deal with non-euclidean data. Graph convolutional networks (KIPF; WELLING, 2016) are GNNs that apply convolution operations in graphs, in order to extract high-level node representations. Thus, GCN generalizes the operation of convolution from grid data to graph data. The main idea is to generate a node representation by aggregating its features and the neighbors' features. As in CNN, higher-level representations of node features are obtained by stacking multiple graph convolutional layers (WU et al., 2020).

The distinctive characteristic of GCN's is the graph convolution layer. Considering the i-th convolutional layer, intuitively, the feature matrix taken as input $H^{(i)}$ is multiplied by adjacency matrix A and by a trainable weights matrix W. This matrix product allows aggregating the features with their neighbors' node features. The output matrix $H^{(i+1)}$ resulting from this operation is a new node feature matrix. Thus, $H^{(i+1)}$ is calculated from $H^{(i)}$ as follows (KIPF; WELLING, 2016):

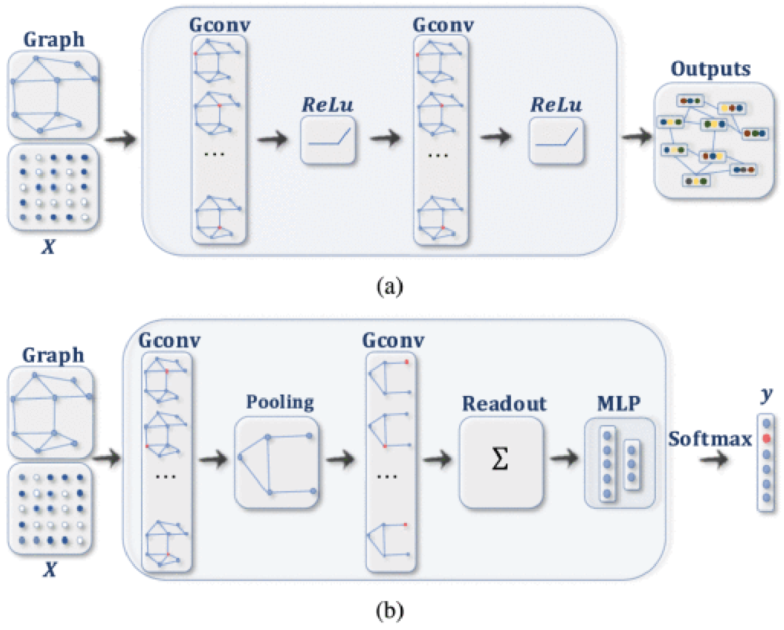
$$H^{(i+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(i)}W) \tag{2.2}$$

, where σ is a given activation function (such as ReLU, hyperbolic tangent, etc.), $\mathbf{I}$ is the identity matrix, so that $\hat{A} = A + I$ represents the adjacency matrix with inserted self-loops. Besides that, $\hat{D}$ is the degree matrix (diagonal matrix which represents the number of edges attached to each vertex), so that $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$. Finally, W is the trainable weight matrix, where $W \in \mathbb{R}^{d_i \times d_{i+1}}$, where d_i is the number of node features considered in the i-th convolutional layer and d_{i+1} is the number of features resulting from this convolution operation. This equation enables the model to learn the feature representations based on neighborhood nodes and their spatial relations.

It is important to notice that the dimensions of the weight matrix do not depend on the size of the graph. Thus, the weights are adjusted regardless of the graph size, allowing the GCN to deal with graphs of different sizes.

Figure 2.6 shows two different GCN architectures. The first (a) uses stacks of convolutions followed by the non-linear activation function, ultimately generating features representing each graph node. The second (b) applies convolution followed by pooling to coarsen a graph into subgraphs, generating higher graph-level representations. After, the readout layer uses the updated node feature vectors to generate a single vectorial representation of the entire graph, which can then be used as input for a set of densely connected layers with a softmax function being applied to the output layer for classifying the input graph. These two architectures show elements that appear in most current GCN, usually containing blocks with a combination of them. An example is convolution, followed by a non-linear activation function and a readout layer for graph classification.

Figure 2.6 – Two traditional GCN architectures. Gconv denotes a convolutional graph layer, MLP is a multilayer perceptron, and readout is the layer that performs an operation that summarizes the output (e.g., global pooling).

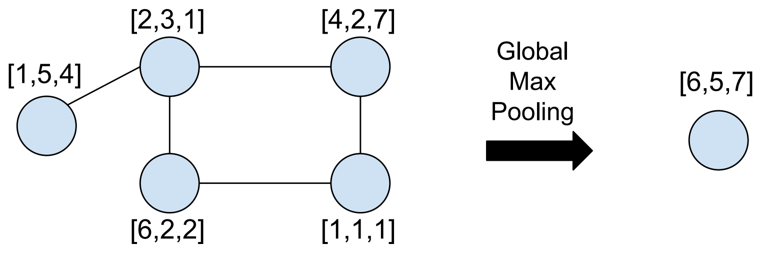


Source: adapted from (WU et al., 2020).

A widely used readout layer is the *global max pooling layer*. Given the input graph with feature matrix $\mathbf{H}$ (with N rows and F columns), the global max pooling layer pools it into a single node computing the feature-wise maximum across the node dimension of the graph. This pooling takes the max value of each feature in X (feature matrix) and puts it into the new single-node vector (Figure 2.7). That is, in this context, the global max pooling layer produces a feature vector $r \in \mathbb{R}^F$, such that:

$$r = max_{n=1}^{N} \mathbf{H}_n \tag{2.3}$$

Figure 2.7 – Schema illustrating how global max pooling works.



Source: The author.

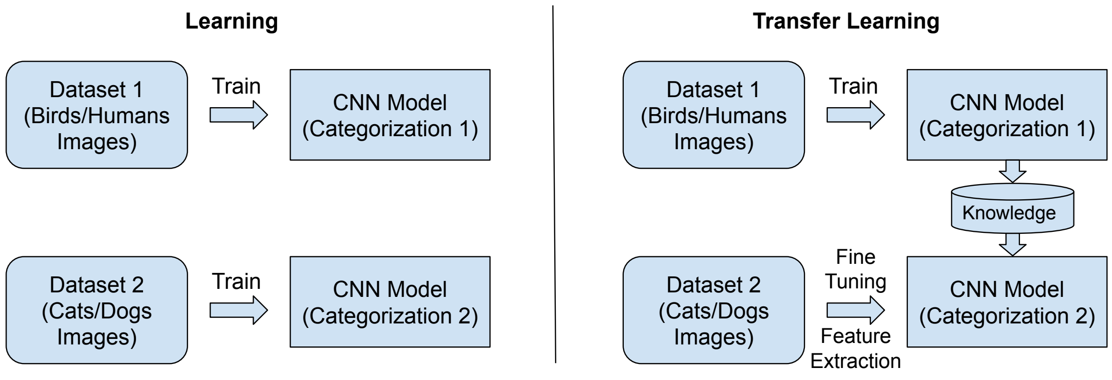
These NN models presented so far (CNN, Transformers, GCN) have achieved, in recent years, impressive results. This is mainly due to the appearance of huge databases making it possible to introduce the transfer learning strategy to reuse training in several tasks (SZELISKI, 2022).

2.2.2 Transfer Learning

Transfer learning (TL) refers to a family of strategies that allows exploiting the knowledge learned in a given task to improve performance in a different task (PAN; YANG, 2010). In recent years, TL has been gaining attention due to the development and availability of large datasets and due to the positive results of these techniques for improving the performance of machine learning models on small datasets.

Figure 2.8 illustrates the contrast between traditional learning and transfer learning. Consider that we are dealing with two scenarios. In the first one, we need to deal with a large dataset of images of birds and humans, while in the second, we need to deal with a small dataset of images of dogs and cats. In traditional learning, we train a model for each scenario, considering only the information from each of its datasets. Applying transfer learning, we can train a model to classify images of humans and birds in the first scenario, and then, we can use the learned weights of the model learned in this scenario to train a model to recognize images of dogs and cats. Even in cases like this, where the visual categories are different in both scenarios, if there is a suitable amount of significant data in the first learning process, the model can learn representations that can be useful for the second scenario (GOODFELLOW; BENGIO; COURVILLE, 2016; PAN; YANG, 2010).

Figure 2.8 – Traditional Learning vs. Transfer Learning.


Learning
Transfer Learning
Dataset 1 (Birds/Humans Images)
Train
CNN Model (Categorization 1)
Dataset 2 (Cats/Dogs Images)
Train
CNN Model (Categorization 2)
Dataset 1 (Birds/Humans Images)
Train
CNN Model (Categorization 1)
Knowledge
Dataset 2 (Cats/Dogs Images)
Fine Tuning
Feature Extraction
CNN Model (Categorization 2)

Source: adapted from (PAN; YANG, 2010).

There are two main approaches for applying transfer learning (LI; HOIEM, 2017): *feature extraction* and *fine-tuning*. Feature extraction, in the context of transfer learning, is the process of using a pre-trained model (playing the role of a *feature extractor*) for transforming the raw data into informative numerical features that facilitate information processing in downstream tasks. Notice that in this process, we take advantage of patterns learned by the pre-trained model for producing informative features in other datasets. In order to perform the feature extraction, we freeze the weights of parts of a given pre-trained model and use these parts of the model for just transforming the raw data in informative features, in a way that all the newly processed data will not change the model weights. In general, a common practice in feature extraction involves removing the last layers (responsible for performing classification) of a given pre-trained model and using only its first layers for extracting features. For image classification, it is a common practice to perform feature extraction on images from neural networks pre-trained on large datasets (e.g., Imagenet (DENG et al., 2009)). In the fine-tuning process, on the other hand, the weights learned by a pre-trained model are used as a starting point for a new training process of the model in a novel dataset. Notice that the weights of the pre-trained model are adjusted to the new task in this process. Thus, while feature extraction freezes the model weights, in the fine-tuning process they are updated to handle the new task.

Both feature extraction and fine-tuning have advantages and disadvantages, and the choice between them is related to the purpose of each application and available computational resources. If the dataset has fewer training samples or the goal is ensuring a shorter training time, feature extraction is justified (BARBURICEANU; TEREBEȘ, 2022). If the task requires a more specific classification or the data is significantly different from the original data used to pre-train the model, then fine-tuning can provide better results (BOYD; CZAJKA; BOWYER, 2019). In (KIEFFER et al., 2017; MORMONT; GEURTS; MARÉE, 2018), the authors compare the performance achieved with feature extraction and fine-tuning for classifying histopathology images. The experiments demonstrate that fine-tuning achieves higher performance, but the results suggest that feature extraction achieves a reasonable performance while requiring fewer computational resources for training. Due to this, in our work, we used feature extraction from pre-trained models.

2.2.3 Pre-trained models

Nowadays, there are several models of neural networks pre-trained on large datasets that are available for performing transfer learning in visual tasks. In this work, we adopted three different models, based on well-established architectures in the image classification state-of-the-art[1]: (i) ResNeXt-101 (XIE et al., 2017), pre-trained on ImageNet; (ii) DenseNet-121 (HUANG et al., 2017), pre-trained on ImageNet; and (iii) CLIP Vit-B/32 (RADFORD et al., 2021), pre-trained in a dataset with 400 million images called WebImageText (WIT). The first two models present good performance with fine-tuning (BAKER; ZENGELER; HANDMANN, 2022) and as a feature extractor (VARSHNI et al., 2019), respectively, while the last one is a transformer-based model that presented impressive results in recent studies (ZHAI et al., 2022). In Table 2.1 we present the following properties of the selected models: number of output features, number of parameters, training dataset, and architecture.

Models	**Output features**	**Parameters**	**Training dataset**	**Architecture**
DenseNet-121	1024	8.0M	ImageNet-1K	CNN
ResNeXt-101	1000	83.5M	ImageNet-1K	CNN
CLIP Vit-B/32	512	63.0M	WebImageText (WIT)	Transformer

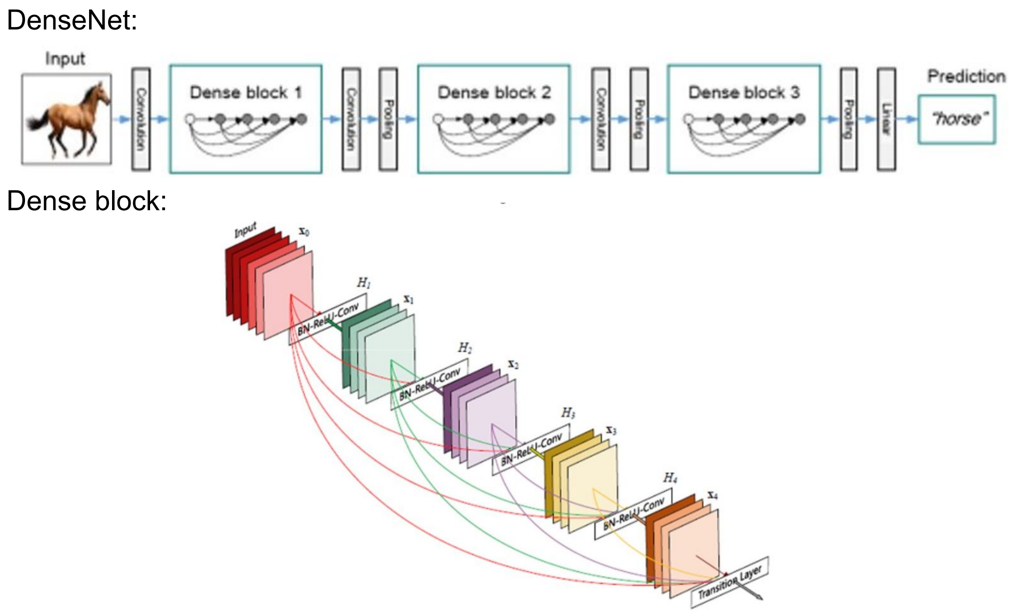
Table 2.1 – Pre-trained models information

Both, ResNeXt and DenseNet, are CNN architectures that have as their main characteristics carrying residual information over the network, improving performance, and

[1]The resNext-101 and DenseNet-121 pre-trained models were obtained from <https://pytorch.org/vision/stable/models.html>, and the CLIP Vit-B/32 pre-trained model was obtained from <https://github.com/openai/CLIP>

reducing network degradation. DenseNet adopts dense blocks, where each layer within that block contains a connection with the layers ahead. Features extracted in initial layers are preserved and can be used directly by deeper layers. ResNeXt, on the other hand, has an architecture based on ResNet (HE et al., 2016), which is based on residual blocks that change the learning function by adding an identity mapping connection. The ResNeXt improves the ResNet architecture by considering information related to the size of the set of transformations (the "cardinality"). In this case, cardinality is an essential factor, in addition to depth and width dimensions, being more effective than going deeper or wider when we increase the capacity. Figure 2.9 and Figure 2.10 show how these architectures work.

Figure 2.9 – DenseNet architecture. The architecture comprises a stack of dense blocks that use multiple connections called "short paths" to carry residual information between layers.

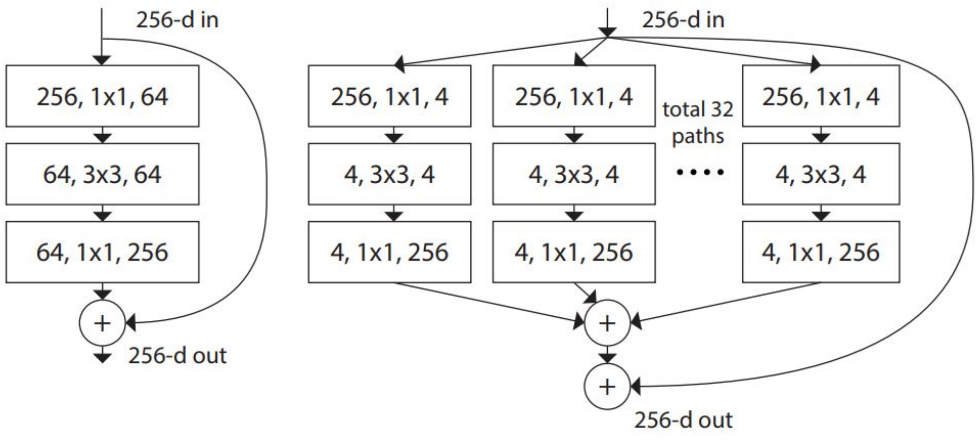


Source: (HUANG et al., 2017)

On the other hand, CLIP (Contrastive Language-Image Pre-Training), is a neural network architecture based on vision transformers that is trained on a wide variety of images and textual labels that is abundantly available on the internet (more than 400 million pairs of images and text). As shown in Figure 2.11, to combine text and images, CLIP trains an image encoder and a text encoder to predict the correct pairings of a batch of image/text training examples. Thus, this model has great potential to perform zero-shot predictions, where no previous training data exists for the new images presented. Due to its characteristics, CLIP also has great potential in feature extraction in images from different contexts.

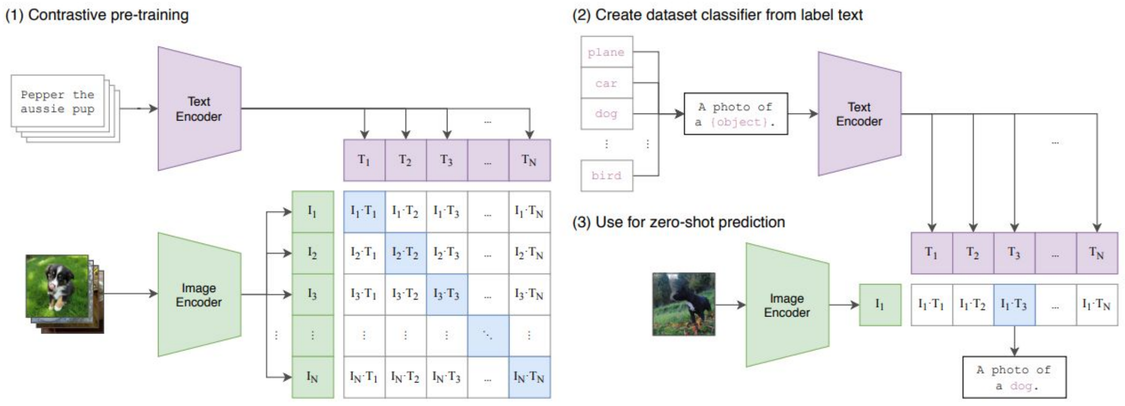
In our work, in addition to using pre-trained models, we also performed training

Figure 2.10 – ResNet traditional learning block vs. ResNeXt learning block with cardinality = 32. Keeping the connection of residual information and with a larger set of transformations per block made it possible to increase the performance without going deeper.



Source: (XIE et al., 2017)

Figure 2.11 – CLIP approach to learning transferable visual models from natural language supervision. First, it presents a contrastive pre-training with image/text pairs feeding a text encoder and an image encoder. Then, using a dataset from label text makes it possible to match the image with the text and use it for zero-shot prediction or feature extraction.


(1) Contrastive pre-training
Pepper the aussie pup
Text Encoder
Image Encoder
(2) Create dataset classifier from label text
plane
car
dog
bird
A photo of a {object}.
(3) Use for zero-shot prediction
A photo of a dog.

Source: (RADFORD et al., 2021)

on a classification module. We used the early-stopping training strategy, which will be presented in the next subsection.

2.2.4 Early stopping

In Machine Learning, the main goal is to achieve models that generalize and perform well on data that has yet to be seen. The main obstacle to this is *overfitting*, which occurs when the model loses generalization power and performs well only on already-seen data.

A common practice in machine learning is to divide the available data into three sets: a training set, a validation set, and a test set. The training set is used to train the model, while the validation set is used to evaluate its performance on unseen data and tune its hyperparameters. Finally, the test set is used to test the model's performance on completely unseen data.

To generate a suitable model, it is essential to find the point during the training phase when the model has the best performance on the validation set. Several methods can be used to identify this stopping point. One of them is the *early stopping* strategy (CHOLLET, 2021). The main idea behind early stopping is to monitor the model's performance for every epoch in the validation set. As the training progresses, the model's performance on the validation set improves, but there comes a point where the model starts to overfit and its performance on the validation set starts to decrease. At this point, the training is stopped, and the model with the best performance on the validation set is selected as the final model.

Figure 2.12 represents a training process applying early stopping, illustrating the point where the best model is found and the training process is stopped (GÉRON, 2022). Two important parameters of this strategy are *minimal improvement* and *patience*. The minimal improvement is how much the model has to improve the performance of the previous epoch for a stop trigger not to occur. Patience is the number of times a trigger must occur to stop the training process.

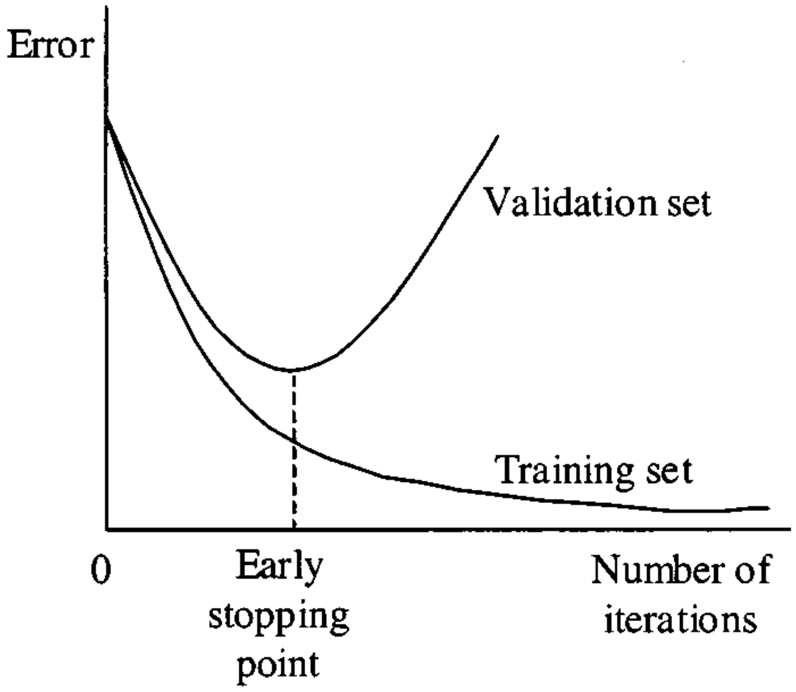
2.2.5 K-fold Cross-validation

Splitting the dataset into fixed sets can be problematic if the data is small, resulting in a small test set that implies statistical uncertainty around the estimated average error, making it difficult to compare the performance of two different models (GOODFELLOW; BENGIO; COURVILLE, 2016). The *k-fold cross-validation* is an evaluation procedure proposed for mitigating these effects.

The K-fold Cross-validation technique randomly divides a dataset into K equal-sized non-overlapping sets (folds) of samples. In each iteration in a ML model, one of the K sets of samples is considered as the test set, and the union of the remaining $K-1$ sets is considered as the training (or training and validation) set.

Being $\mathbf{X}$ the dataset split into a set of k folds $\{X_1, ..., X_k\}$. The definition of the test and train test in each of the k iterations is represented as follows:

Figure 2.12 – Early stopping applied in a learning process.



Source: (GENÇAY; QI, 2001)

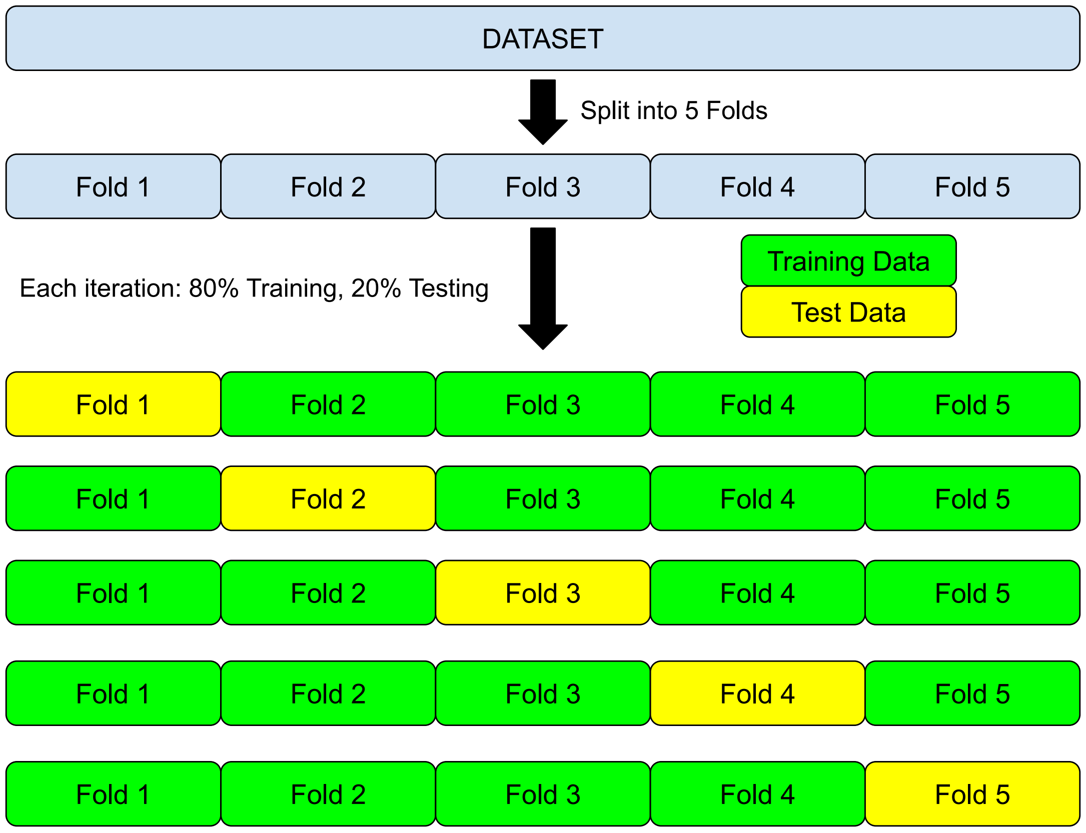
$$\mathbf{X} = \begin{cases} Test_1 = X_1, Train_1 = X_2 \cup X_3 \cup \ldots \cup X_K \\ Test_2 = X_2, Train_2 = X_1 \cup X_3 \cup \ldots \cup X_K \\ \vdots \\ Test_K = X_K, Train_K = X_1 \cup X_2 \cup \ldots \cup X_{K-1} \end{cases} \tag{2.4}$$

For each fold, we compute the scores (e.g, evaluation measures) and merge them using an average (ALPAYDIN, 2020). Figure 2.13 shows an example of using K-fold cross-validation with $K = 5$.

At each iteration of the k-fold cross-validation process, a given model is trained with the training data, and its performance scores are calculated on the test set. In the end, the scores obtained in each iteration are averaged for obtaining the overall performance scores of the model (ALPAYDIN, 2020). Figure 2.13 shows an example of using k-fold cross-validation with $K = 5$.

Several different performance metrics can be obtained when k-fold cross-validation is adopted. In the following Section, we discuss the performance metrics adopted in this work.

Figure 2.13 – Schema illustrating a 5-fold cross-validation.



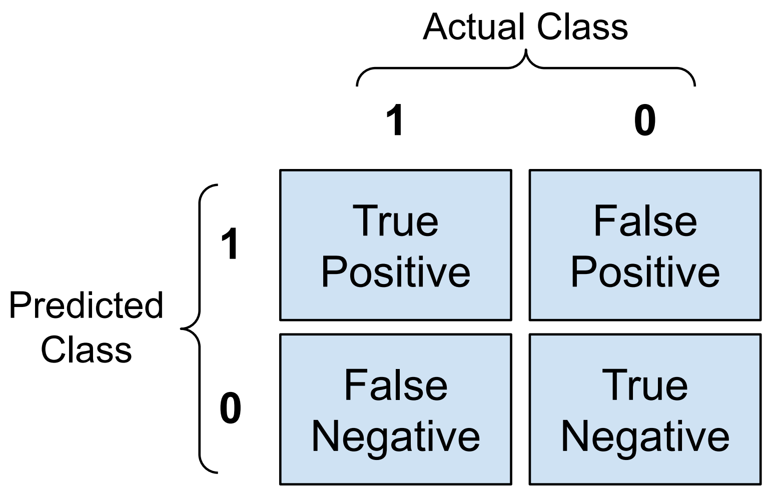
Source: The author

2.2.6 Performance metrics

In ML, the metrics for evaluating a model must be chosen according to the type of task. For classification algorithms, the most popular metrics are precision, recall, and F1-score, which are based on True and False Positives and Negatives, which can be identified in a *confusion matrix*, as shown in Figure 2.14. True positives (TP) are those samples that belong to the positive class and are correctly identified, while true negatives (TN) are samples of the negative class that are correctly identified. False positives (FP) are samples of the negative class incorrectly identified as belonging to the positive class, while false negatives (FN) are samples of the positive class incorrectly identified as belonging to the negative class.

By considering these basic concepts, we can define precision, recall and f1-score. In this context, precision can be viewed as a way to measure what ratio of the samples a classifier predicted as belonging to a particular class was actually correct. It is calculated as follows:

Figure 2.14 – Confusion matrix for a binary classification problem. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.



Source: The author

$$\textbf{Precision} = \left(\frac{TP}{TP + FP}\right) \tag{2.5}$$

Recall, on the other hand, measures what ratio of samples from a particular class in a set was correctly predicted. It is calculated as follows:

$$\textbf{Recall} = \left(\frac{TP}{TP + FN}\right) \tag{2.6}$$

The F1-score summarizes both precision and recall with a single value by taking the harmonic mean of them. It is calculated using both metrics, with the following formula:

$$\textbf{F1} = \left(2 \times \frac{Precision * Recall}{Precision + Recall}\right) \tag{2.7}$$

These metrics can be generated for each class present in the classification dataset, by considering each target class as the positive class and the remaining classes as the negative. In a multiclass classification problem, a typical way to obtain a single measure value for evaluating the overall performance of the model is by calculating an average of these metrics considering the metric's value for each individual class. In this case, there are three different averages: macro average, weighted average, and micro average.

Thus, let M be a given metric (precision, recall or f1-score), the macro average of M is computed as the arithmetic mean of the values of M for each individual class.

Thus, the macro average for M is given by $\frac{\sum_{i=1}^{n} M_i}{n}$, where n is the number of classes in the dataset and M_i is the metric value of the i-th class. It is important to notice that the macro average considers all classes with a uniform weight regardless of how many instances they have in the dataset. The weighted average of a metric M, on the other hand, considers each class's support (number of instances). Thus, the weighted average of M is given by $\frac{\sum_{i=1}^{n} M_i \times S_i}{T}$, where S_i is the support of the i-th class and T is the total number of instances. Thus, in the weighted average, the number of instances of each class matters.

The micro average counts the sums of the TP, FN, and FP. Micro-averaging essentially computes the proportion of correctly classified observations out of all observations. Due to this, in multi-class classification tasks where each observation has a single label, the micro average of precision, recall, and f1-measure have the same value, which is equivalent to another metric, called accuracy:

$$\textbf{Accuracy} = \left(\frac{TP}{TP + \left(\frac{1}{2}\right)(FP + FN)} \right) \tag{2.8}$$

Conventional accuracy is also known as top-1 accuracy where the model answer (the one with the highest probability) must be exactly the expected answer. There are variations of this, such as the top-5 accuracy, where the prediction will be a true positive if any of the model's five highest probability answers match the expected answer.

The choice for each of the macro and weighted averages must be aligned with the dataset type. If the dataset is unbalanced and all classes are equally important, the Macro will be the best. If the frequency of each class in the dataset is important, then we should use the Weighted. Accuracy (Micro average) is more suitable if the dataset is balanced.

3 RELATED WORKS

In our literature research, we focused on works that deal with the challenges present in our context: small data, images of varying sizes, and visual features represented at different scales. This section presents a discussion of some related works that deal with these challenges.

In general, sophisticated deep learning models demand large amounts of training data to achieve a good performance (ZHU et al., 2021). Due to this, in the last few years, we have witnessed an increasing adoption of transfer learning for dealing with classification tasks in contexts where data are scarce (TORREY; SHAVLIK, 2010; ZHUANG et al., 2020), since transfer learning allows the reuse of knowledge learned in other tasks for which abundant data are available. The adoption of transfer learning techniques has also become common in image classification tasks. CNN's such as ResNeXt (XIE et al., 2017) and DenseNet (HUANG et al., 2017) have been achieving great performances when used for transfer learning, both as being used as pre-trained models for *feature extraction*, as well as for end-to-end classification with *fine-tuning* (BAKER; ZENGELER; HANDMANN, 2022; VARSHNI et al., 2019).

Due to the scarcity of data in the area, the medical field stands out in adopting transfer learning (KIM et al., 2022). For example, in (LIANG; ZHENG, 2020), the authors use model parameters learned on large-scale datasets in the same field to improve the training phase (fine-tuning) to diagnose and detect childhood pneumonia. In (CELIK et al., 2020), the authors applied transfer learning for feature extraction to analyze digital histopathology images to detect breast cancer.

It is important to note that, in general, real-world image datasets include images with different sizes. However, commonly, neural network architectures demand inputs to have a standard size. Due to this, in image classification tasks, usually, the images are normalized to a homogeneous fixed size (LECUN et al., 1998; KRIZHEVSKY; HINTON et al., 2009) to match the input requirements of the adopted neural network architecture. This normalization process is also a common approach for applying transfer learning (AHUJA et al., 2021), since the images should meet the input requirements of the pre-trained models.

In order to deal with datasets with images of heterogeneous sizes, some approaches initially perform pre-processing adjusting the input images to the requirements of neural network architectures. In some of these approaches, distorting or even discarding parts of images is common. For instance, a standard approach applied in different studies (TAYAL et al., 2021; HAN et al., 2020; LI et al., 2021) involves resizing the smallest dimension of each image to match the model's input dimensions and performing a center crop of the image, discarding the remaining image information. This process can result in the loss of relevant information from the images or the insertion of noise in the dataset. These effects are pronounced when the image has one of its dimensions much bigger than the other (very long and narrow images, for example). On the other hand, in order to deal with images with heterogeneous sizes, some works (WANG et al., 2021; ZHUANG et al., 2021) adopt *fully convolutional networks* (FCN), which do not require the input images to have a homogeneous size.

Since objects in images are often of varied sizes and real-world images have varying sizes, leveraging information carried by different scales is critical in computer vision applications. Aggregating multiscale contexts and handling different image sizes are factors that can improve vision systems. The multiscale features are convenient because the CNN is constructed by stacking blocks with similar functionality, which naturally enables a progressive resolution decay (CHEN et al., 2020). Some studies take advantage of this characteristic and apply different spatial scales in order to classify images. In (MOHAN; VENKATESAN, 2020), for example, the authors propose using a multiscale spatio-spectral feature-based hybrid CNN model for hyperspectral image classification, which adopts different window sizes in 3D convolution filters. In general, in these works (SAFARI; PRASAD; LABATE, 2020), different scales are related to different independent convolution layers with different kernels that are later combined. Usually, these approaches do not adopt transfer learning to take advantage of knowledge already acquired in other tasks. Due to this, contexts with small training datasets can be challenging for this kind of strategy.

In the medical field, where images tend to be large and with scattered details (X-ray, CT images), several works use patch-based approaches (ANTHIMOPOULOS et al., 2016; ARAÚJO et al., 2017) for classifying images. These approaches split the image into several parts, making it possible to analyze different parts of the original image in a more focused way. To deal with CT images for classifying cases of COVID-19 and pneumonia, for example, in (XU et al., 2020) the authors adopt attention techniques to select areas and use them as patches to perform classification only in essential parts of the image. In (BARSTUGAN; OZKAYA; OZTURK, 2020), the authors present an approach that preprocesses the image (discarding information) and splits it into patches of different sizes, training just one classifier with these different scales. An example presented in (CHEN; FAN; PANDA, 2021) showed remarkable results using multiscale patch sizes to train a vision transformer, performing a fusion of features extracted at different scales.

Recently, there is an increasing adoption of GNN for image classification. In these approaches, the images are converted into graphs, which are considered the inputs of the GNN. In (SHI et al., 2020), the authors use graphs to represent the skeleton during the movement, while in (PAN et al., 2020), graphs are used to identify objects in a scene to describe what is happening in a video. One of the main advantages of GNN is the possibility of dealing with data with non-Euclidean structures, allowing it to deal with graphs of heterogeneous sizes. Most of the works found in the literature first segment the image into superpixels, considering each superpixel as a graph node and generating edges from the neighborhood regions (ZHANG; ZOU; ZHANG, 2022). In this type of approach, the nodes' features are elaborated from each region's statistical properties (for example, the mean and standard deviation of the information of each pixel). In this line, (BAE et al., 2022) uses this technique to describe and classify HSI images. In (JING et al., 2022), the authors also use superpixels for Hyperspectral Image classification but adopt a multiscale graph sample and aggregate network to capture spatial and spectral features at different scales flexibly. A similar strategy is used by (DING et al., 2021), where multiscale graphs are constructed from split regions of HSI images. These different scales, however, refer to graphs scales, and images are kept in their original size. In (AVELAR et al., 2020), the authors transform the input images into region adjacency graphs (RAGs), in which regions are superpixels and edges connect neighboring superpixels and adopt this image representation for training a graph attention network (GATs). In this work, the authors show that this strategy can be applied to non-euclidean images (e.g., omnidirectional images). However, the information loss in the pixel aggregation for more complex images can result in significant performance degradation, when compared to strategies that can take advantage of fine-grained details in the full image. In general, these techniques also do not consider using transfer learning to take advantage of prior knowledge to deal with small datasets.

Other approaches adopt regular image patches/tiles as nodes. In (KONDA; WU; WANG, 2020), the authors apply transfer learning to generate the features of each patch/tile and characterize each node. However, their approach does not consider the multiple spatial scales of analysis. Another example is in (HAN et al., 2022), where the authors split the image into a number of patches and view them as nodes. The authors realize that directly using graph convolution on the image graph structure implies an over-smoothing of the features, resulting in poor performance. Due to this, they apply feature transformations for each node. The GCN architecture used presents promising results, but they also do not deal with multiple scales and do not apply transfer learning.

In order to deal with challenges similar to those found in our work, (WAN et al., 2019) presents an approach using superpixels, multiple neighborhood scales, and GCN for hyperspectral image classification. Their approach differs from other methods because it does not depend on a fixed input size for the graph and combines multiple scales to improve the classification. However, the authors do not adopt transfer learning and segment the image into superpixels that exclude areas of the image. Our approach, aiming at classifying images in general, uses overlapping patches (to highlight focal features throughout the image) and takes advantage of transfer learning to extract features without retraining the complete model. Another difference is that the multiple scales of our approach refer to the image's size and not to the neighborhood between nodes.

4 PROPOSED APPROACH

Our approach consists of two main aspects that we will present in the following sections: (i) a strategy for representing images as sets of patch-based feature graphs that can capture the visual features of each image at different scales (Section 4.1); and (ii) a multi-input graph convolutional network architecture for multiscale image classification that takes as input a set of patch-based feature graphs representing a given image at different scales (Section 4.2).

4.1 Patch-based graph for image representation

In our work, we represent a given image by a graph constituted of nodes, denoting features of the original image's regular patches; and edges, representing spatial relationships between patches in the original image. An essential aspect of our approach is characterizing each node of the generated graph with features extracted from each patch of the original image by pre-trained models. By adopting *feature extraction* from pre-trained models, our approach can leverage the knowledge captured in models that were pre-trained in large datasets for generating informative representations of the original image in small datasets.

Algorithm 1 represents the process for building a patch-based feature graph representation of each image in a given scale of analysis. The algorithm takes as input a pre-trained model FE, a target image I, a value $N \in \mathbb{Z}^+$ that controls the spatial scale of analysis of I, and the stride $S \in (0, 1]$ as input. We use the pre-trained model FE as a *feature extractor*, while N defines a scale of analysis and S represents the stride of the sliding window used for extracting patches from I. Notice that S is considered as a percentage of FE's input size.

In our approach, a patch-based feature graph $g(I, FE, N, S)$ represents an image I at the scale of analysis N, built from patches extracted from I with a stride S and with visual features extracted by FE. Notice that $g(I, FE, N, S) = (V, E)$, where V is the set of vertices (or nodes) and E is a set of edges that constitutes the resulting graph. As it is common in the context of graph convolutional networks (LEE; LEE; SOHN, 2021), the set of edges E is represented by an adjacency matrix A, and the set of features of nodes/vertices V is represented by a node feature matrix H, which will be detailed in the following.

In order to understand how the adjacency matrix A and the node feature matrix H are built, let us consider $P(I, S)$ as a set of regular patches extracted from I by considering the stride S in both vertical and horizontal axis. Notice that the size of each patch $P_j \in P(I, S)$ is compatible with the input dimensions of FE. Besides that, patches in $P(I, S)$ can overlap, depending on the stride value S.

Let us consider V as a set of vertices of $g(I, FE, N, S)$, where each $V_j \in V$ is a $\mathbb{R}^d$ vector that represents the features extracted from the patch P_j by FE. Thus, the node feature matrix $H \in \mathbb{R}^{|P(I,S)| \times d}$ is built in a way that $H_i = FE(P_i)$. Notice that $|P(I, S)|$ is the cardinality of the set $P(I, S)$ and d is the number of features generated by the feature extractor FE.

As previously mentioned, E is a set of edges of $g(I, FE, N, S)$, where each $E_j \in E$ is a pair $\{P_x, P_y\}$ that represents that $P_x \in P(I, S)$ and $P_y \in P(I, S)$ are adjacent to each other in the original image. Thus, the edges in E represent the spatial neighborhood of the original image patches. Each pair of patches that are spatially adjacent in P has a corresponding edge in E. In this context, the information of E is represented by the adjacency matrix $A \in \mathbb{R}^{|P(I,S)| \times |P(I,S)|}$, such that:

$$A_{i,j} = \begin{cases} 1, & adjacent(P_x, P_y) \\ 0, & otherwise \end{cases} \tag{4.1}$$

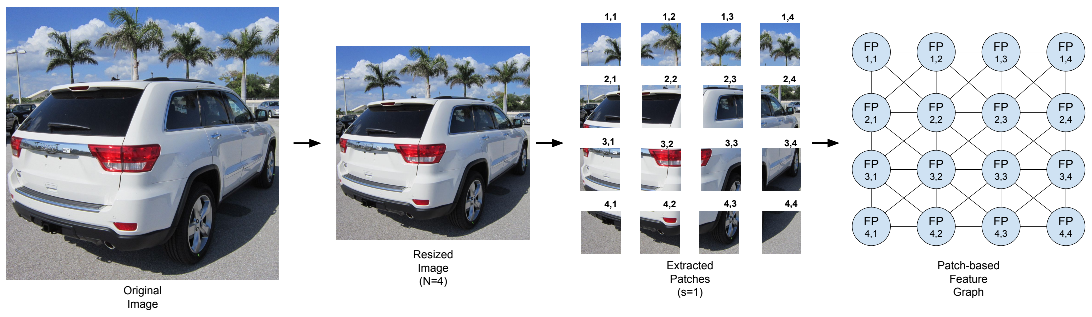
where $adjacent(P_x, P_y)$ means that there is at most L pixels P_x and P_y centroids, considering $L = S \times FE$'s input size. Notice that each patch has a maximum of 8 neighbors.

Algorithm 1 starts by resizing the smallest dimension of I by N times the FE's input size, while maintaining its original aspect ratio. Next, we extract from I the set $P(I, S)$ of patches by sliding a window whose dimensions match the input dimensions of FE. The sliding window moves through the image following the stride S in both the horizontal and vertical axis. It is important to notice that the patches may overlap depending on the sliding window stride. Then, the adjacency matrix A (representing the information of the set of edges E) is created by considering the spatial adjacency of the extracted patches in I, according to the equation 4.1. Finally, the algorithm creates the set V of vertices. In this final step, for each patch, P_j in $P(I, S)$, the algorithm produces a row in the feature matrix H, where each $H_i \in H$ is a vector of features extracted from the patch P_j with the feature extractor FE. In the end, the algorithm outputs the pair (H, A) that represents the patch-based feature graph built for representing I. Figure 4.1 presents a simplified example of how the algorithm works.

Algorithm 1: Patch-based feature graph construction

```
Input: Image as $I$, pre-trained model as $FE$, the approach parameter as $N$ and the stride of the sliding window as
    $S$;
Output: A tuple ($H$,$A$) representing a patch-based feature graph $g(I, FE, N, S)$;
begin
    $W \leftarrow Width(I)$;
    $H \leftarrow Height(I)$;
    $HJ \leftarrow$ height input size of FE;
    $WJ \leftarrow$ width input size of FE;
    if $H \leq W$ then
        $aspect \leftarrow W/H$;
        $NewH \leftarrow HJ * N$;
        $NewW \leftarrow NewH * aspect$;
    else
        $aspect \leftarrow H/W$;
        $NewW \leftarrow WJ * N$;
        $NewH \leftarrow NewW * aspect$;
    $I \leftarrow I$ resized to $NewH$x$NewW$;
    $P(I, S) \leftarrow$ patches extracted from $I$ using $S$;
    $A \leftarrow$ adjacency matrix built according to equation 4.1;
    for $P_i$ in $P(I, S)$ do
        $H_i \leftarrow FE(P_i)$;
    return $(H, A)$;
```

Figure 4.1 – A simplified example of how Algorithm 1 works applied to an image from the Stanford Cars dataset (KRAUSE et al., 2013). The algorithm starts by considering the original image at right. After, the algorithm resizes the image using the value of $N = 4$ (in this example). After, the algorithm extracts patches from the image using the stride value $S = 1$ (without overlapping). Finally, the algorithm builds the resulting graph using the features extracted from each patch (FP) as nodes, and considering the spatial neighborhood of each patch for creating edges.



Source: The author

The proposed approach extracts patches from the image as an intermediate step for obtaining the *local features* related to specific parts of the image. These local features are spatially structured in a graph that captures the spatial information of the original image. Notice that when the original image has the size of the extractor's input, the resulting patch-based feature graph built from this image degenerates into a single node, when $N = 1$.

It is important to notice that by changing the value of N, the resulting patches focus on areas with different sizes relative to the image size. The larger the N, the smaller the area of the image that a patch contains. Then, given an image I, different values of N can highlight different image patterns or the same pattern at different scales in I. Thus, this strategy can be a promising approach for dealing with sets of images of very different sizes representing the same pattern at different scales. Besides that, by keeping the original image's aspect ratio, the algorithm avoids the distortion of essential domain features. Also, our approach is agnostic to the feature extractor (FE) adopted in the process. Thus, different pre-trained models can be used for different datasets, depending on the task.

Another interesting feature of our approach is its flexibility regarding the neighborhood adopted for building graphs from patches. This allows, for example, discarding uninformative patches from the original image, which would introduce irregularities in the graph structure. This capability can be explored in future works.

As previously mentioned, the Algorithm 1 can be used to build a patch-based feature graph representing the image I in a single given scale of analysis N. Thus, in order to obtain multiscale representations of I, we can apply the Algorithm 1 to generate different patch-based feature graphs from each image I for different scales by varying the value of N, as we show in Figure 4.2. Notice that the scales used in the diagram were considered just for the sake of the example since our approach is flexible, allowing the integration of more or fewer scales as needed. These different graphs can then be used as input for our multi-input graph convolutional architecture for training and classifying the image I. Thus, by considering different scales, our approach represents each image I as a multiscale graph set (MGS), defined by a scale set SS of different scales (different values of N), considering a given stride S and a feature extractor FE, such that:

$$MGS(SS) = \{x | i \in SS \land x = g(I, FE, i, S)\} \tag{4.2}$$

Thus, for example, $MGS(\{1, 2, 3\})$ is the multiscale graph set whose elements are three patch-based feature graphs, built with scales $N = 1$, $N = 2$, and $N = 3$, respectively. That is:

$$MGS(\{1, 2, 3\}) = \{g(I, FE, 1, S), g(I, FE, 2, S), g(I, FE, 3, S)\}. \tag{4.3}$$

4.2 Graph convolutional network for multiscale image classification

The second aspect of our approach is a multi-input graph convolutional network for multiscale image classification based on the graph convolutional network (GCN) proposed in (KIPF; WELLING, 2016). GCN can deal with classification problems by considering the features of each node and the relationships among them in data with a graph structure. Our proposed GCN architecture can take as inputs an MGS representing a given image at different scales (with one graph for each scale considered in the task). Thus, our architecture can be adapted for dealing with different scales in a way that the number of inputs of our architecture is defined as a function of the number of different scales in a scale set SS that defines a multiscale graph set $MGS(SS)$. The suitable scales in the scale set can vary according to the characteristics of the task.

The key element of our architecture is the *graph processing module* (GPM), which is composed of just one graph convolution layer (KIPF; WELLING, 2016) with hyperbolic tangent (tanh) as an activation function, followed by a global max pooling layer. Thus, as stated in Section 2.2.1.3 in the graph convolution layer of a given GPM, the feature matrix taken as input $H^{(i)}$ is multiplied by adjacency matrix A and by a trainable weights matrix W. This matrix product allows aggregating the features extracted by FE from each patch $p_i \in P(I, S)$ with their neighbors' node features. The output matrix $H^{(o)}$ resulting from this operation is a new node feature matrix. Thus, $H^{(o)}$ is calculated from $H^{(i)}$ as follows (KIPF; WELLING, 2016):

$$H^{(o)} = tanh(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(i)}W) \tag{4.4}$$

where I is the identity matrix, so that $\hat{A} = A + I$ represents the adjacency matrix with inserted self-loops. Besides that, $\hat{D}$ is the degree matrix, so that $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$. Finally, W is the trainable weight matrix, where $W \in \mathbb{R}^{d \times d_{out}}$, where d is the number of features outputted by the chosen feature extractor FE and d_{out} was chosen as 64.

Then, given a feature matrix $H \in \mathbb{R}^{|P(I,S)| \times d}$, the global max pooling operation, produces a feature vector $r \in \mathbb{R}^d$, such that:

$$r = max_{n=1}^{|P(I,S)|} H_n \tag{4.5}$$

Depending on the target task, the architecture can be adapted for taking different numbers of patch-based feature graphs, one for each scale. This adaptation involves including in the architecture one GPM for each graph considered as input.

Then, our architecture concatenates the outputs of the GPM's in a single vector, which is processed and classified by a fully connected layer with a linear function. Figure 4.2 represents the proposed architecture adapted with 3 GPM, for considering as input a set $MGS(\{1,2,3\})$ of 3 different patch-based feature graphs in a multiscale graph set.

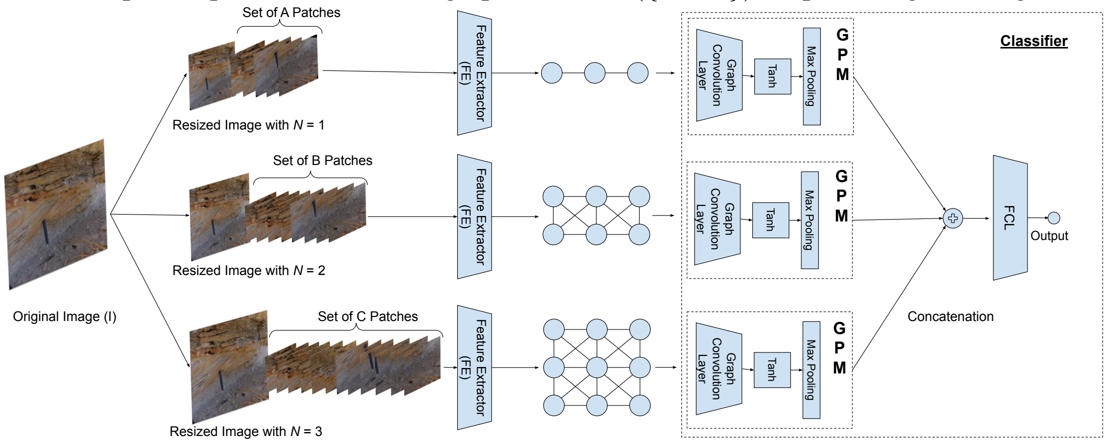
We defined our architecture as a result of empirical experimentation. We tested different architectures, with different numbers of convolutional layers, different kinds of pooling, and different activation functions, with no improvements in the general performance.

In an overview, the main characteristics of our approach are:

- It avoids inserting distortions of relevant visual features by keeping the original image's aspect ratio during all the processing steps.
- It avoids losing relevant information about the original image, since we split the original image into regular patches that cover most of the image area. This method contrasts with common practice in most of the approaches based on neural networks for image classification, which involves a pre-processing step where some parts of the original image are discarded.
- It can deal with images with different sizes in a given dataset since our approach represents these images with graphs of different sizes handled by the proposed graph convolutional architecture.
- It captures informative local features of each patch of the original image using pre-trained models as feature extractors, increasing the capabilities of our approach to dealing with small datasets.
- It represents the spatial relationship of the local features of different patches of a given image in a graph structure.
- It represents the visual features of a given image in different scales of analysis through different patch-based feature graphs that can be collectively used as input for training a multi-input graph convolutional network for classifying images.

To the best of our knowledge, our proposed approach is the first one to combine all these advantages.

Figure 4.2 – An example of the proposed approach for image classification. Given an image, we apply Algorithm 1 with different values of N (1, 2 and 3, in this example), generating a multiscale graph set $MGS(\{1, 2, 3\})$, with three patch-based feature graphs representing the image at three different scales. After, we train a multi-input graph convolutional architecture that takes as input the patch-based feature graphs in $MGS(\{1, 2, 3\})$ for predicting the image class.



Source: The author

5 EXPERIMENTS

In this chapter, we discuss the experiments[1] to evaluate our approach. This chapter is organized as follows. In Section 5.1, we describe the datasets used in our experiments. Next, in Section 5.2, we describe our experiments and present the results.

5.1 Datasets

We evaluate our approach in two datasets: a dataset of *geological images* (MICHELIN et al., 2021) and the Stanford Cars dataset (KRAUSE et al., 2013). The dataset of geological images is considered in this work because, as stated earlier, this work is part of a project whose goal is to develop an *image retrieval system* for the petroleum industry. It is important to notice that the geological dataset was developed in cooperation with companies and cannot be shared due to copyright issues. The Stanford Cars dataset, which is publicly available, was selected to allow the reproducibility of our results and demonstrate that it can be applied to other domains. In the following sub-section, we describe the characteristics of the geological dataset (Section 5.1.1) and the Stanford cars dataset (Section 5.1.2).

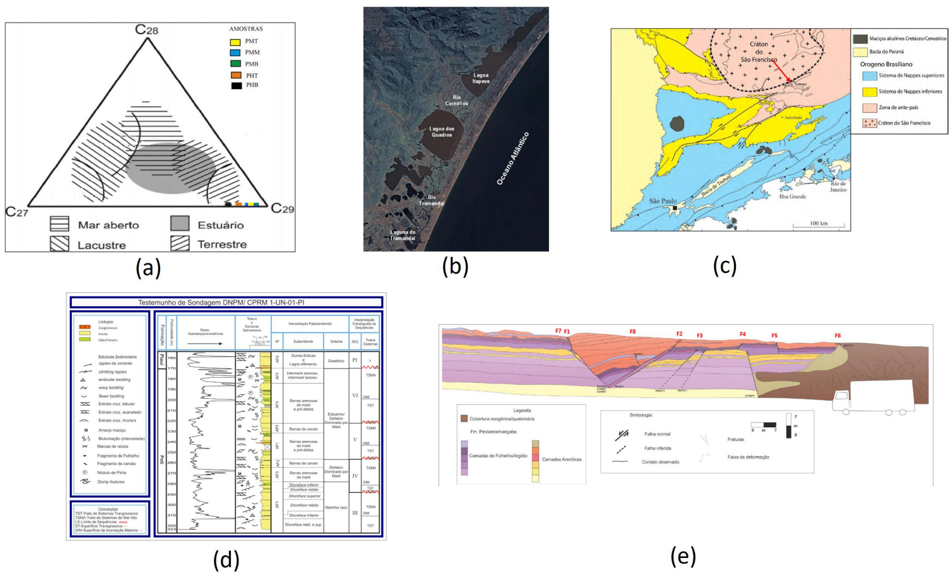
5.1.1 Geological dataset

In (ABEL et al., 2019), the authors present an ontology for image classification in Petroleum Geology. This ontology has 175 classes, such as a ternary diagram, satellite image, geological map, profile, and cross-section. Figure 5.1 presents some examples of images from these classes. After, a set of geological images was collected from geological reports and labeled by experts (MICHELIN et al., 2021) for some of the classes specified in the ontology. The resulting dataset contains 25725 images distributed in 45 classes. It includes classes with only 36 images and classes with 8450 images, with an average of 571.6 images per class and a standard deviation of 1290.9. Figure 5.2 represents the distribution of images in each class, emphasizing its *unbalanced* distribution. Besides that, the images in this dataset are heterogeneous in size and visual features. The average image area in the dataset is 1072336 pixels with a standard deviation of 1861876. There

[1]Our code is available at <https://github.com/mvtodescato/MultiscaleGraphFeatures>

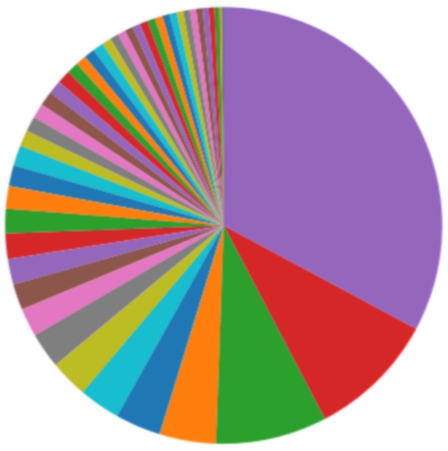
are classes with an average area smaller than half of the others, such as, for example, the class *aerial photograph* that has 495539 pixels of the average area and the class *reference map* that has 1095760 pixels of average area. Also, for some images, one of the dimensions is much larger than the other. These aspects make this dataset challenging for image classification. We emphasize this high variation in Figure 5.3 and in Figure 5.4, which presents some statistical properties regarding this dataset's image sizes and aspect ratios. These figures show that the dataset has several outliers regarding height, width, and aspect ratio.

Figure 5.1 – Examples of images illustrating some classes of the ontology: (a)ternary diagram, (b) satellite image, (c) geological map, (d) profile, and (e) geological cross-section.



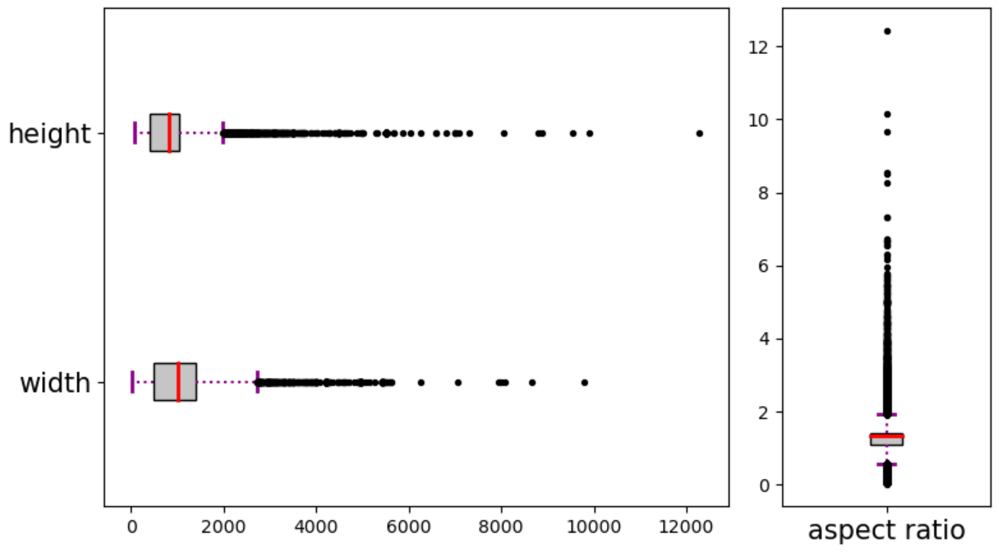
Source: (JúNIOR, 2014), (IVANOFF, 2013), (ARRUZZO, 2016), (PAIVA R, 2018), (SILVA, 2015).

Figure 5.2 – Representation of the number of instances for each class of the geological dataset.



Source: The author

Figure 5.3 – Boxplots representing the statistical properties of the images in the dataset of geological images, focusing on their height, width, and aspect ratio.



Source: The author

5.1.2 Stanford cars dataset

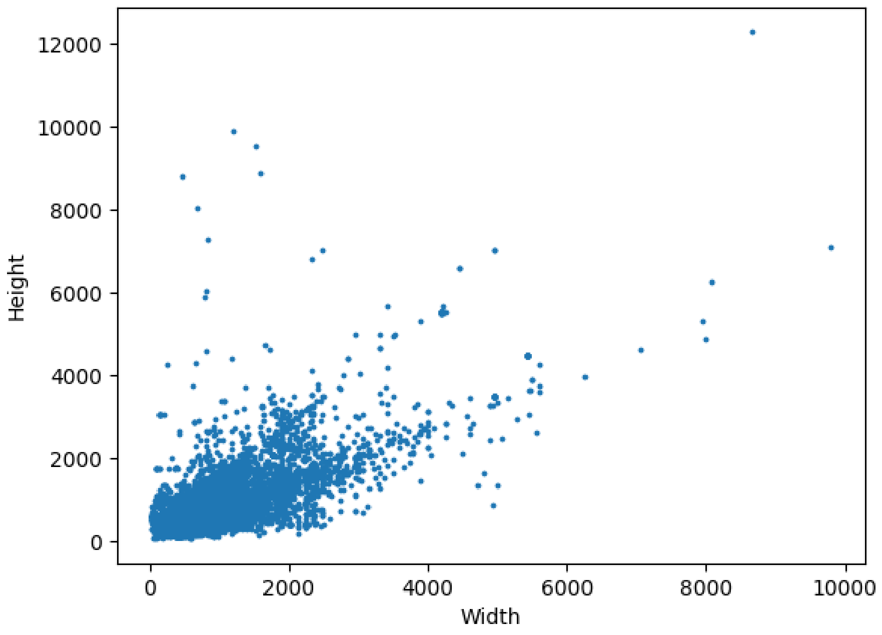
The Stanford Cars dataset (KRAUSE et al., 2013) has 16185 images distributed in 196 classes. The average number of samples in each class is 84 with a standard deviation of 6.28. The samples range from 61 in the smallest class to 110 in the biggest. Unlike the geological dataset, this one is balanced but heterogeneous regarding the size and aspect ratio, as shown in Figure 5.5 and Figure 5.7. In Figure 5.6 we show some images from the dataset. We can notice that the cars have different colors and are presented in different positions and scenarios.

5.2 Results

We performed two experiments. In Section 5.2.1, we present the setup and the results of the first experiment, where the objective was to evaluate how the stride value impacts the performance of our approach. In Section 5.2.2, we present the setup and results of our approach evaluation, where we compare its performance with other approaches for image classification. The structure presented in the following paragraphs is common to all experiments.

In both experiments, we used the datasets previously detailed in Section 5.1. All images were pre-processed to ensure that they are in RGB format. In order to demonstrate

Figure 5.4 – Distribution of the images according to their dimensions in the geological dataset.



Source: The author

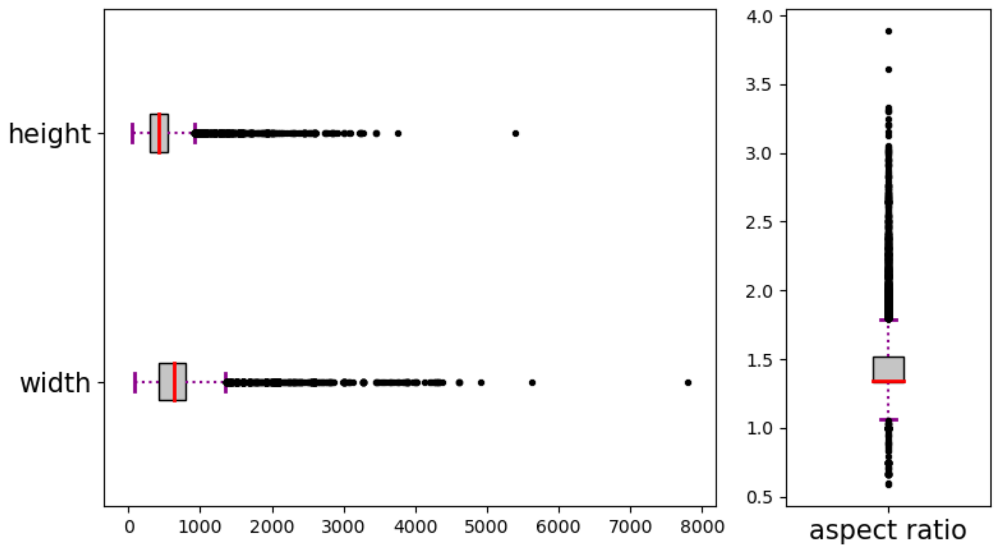
that our approach can adopt different pre-trained models as feature extractors, we applied our approach using the three pre-trained models presented in Section 2.2.3. To facilitate the discussion we will refer to the models by the names of their corresponding architectures. In this work, we used only the feature extraction module of the models so that the classification layer is removed from the original model.

Our algorithm was applied to generate patch-based feature graphs of every image in the datasets, using five different scales represented by the N parameter. We used $N = 1$, $N = 2$, $N = 3$, $N = 4$, and $N = 5$.

In our experiments, we do not use fine-tuning. We also do not apply data augmentation, as our focus is on evaluating feature extraction to deal with small datasets. As described in Chapter 4, our approach uses a classifier module that takes one or more graphs as input, and the output is the predicted class. Thus, in order to use different numbers of patch-based feature graphs (one for each scale), we adapted our multi-input convolutional network architecture by including the necessary number of different graph processing modules for each case. The images in the test set are also transformed into patch-based feature graphs and then evaluated by the trained classifier.

We adopted a *5-fold cross-validation procedure*. At each iteration, each fold is used once as testing data, while from the union of the remaining four folds, we consider

Figure 5.5 – Representation of the statistical properties of the images in the Stanford Cars dataset through boxplots, focusing on their height, width, and aspect ratio (as Width/Height).



Source: The author

10% as validation data and the remaining as training data. We adopted Adam Optimizer with a learning rate of 0.001 and a limit of 100 epochs and cross-entropy loss[2]. We applied early stopping, considering five epochs without a minimal improvement of 0.001 in the loss of the *validation set*. These hyperparameters were defined based on empirical experiments using the approach. We use the following metrics (detailed in Section 2.2.6) to evaluate the results: Top-1 Accuracy, Macro Precision, Weighted Precision, Macro Recall, Weighted Recall, Macro F1, and Weighted F1. These metrics provide a good evaluation of the results since they cover several evaluation aspects in a multiclass classification setting. The results reported in Tables 5.1, 5.2, 5.4, and 5.5 are averages obtained from the test set in the cross-validation procedure. We performed the experiments on a desktop with an Intel i7-10700 CPU, 32 gigabytes of RAM, and an NVIDIA RTX 3060 GPU. The code was implemented in Python, using mainly the PyTorch library[3].

At the beginning of each of the following sections, we explain the experimental setup and, we present the performance metrics, and then we discuss the results.

[2]In the pytorch implementation the cross-entropy loss is equivalent to the combination of *LogSoftmax* and *NLLLoss* (More details in <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>).

[3]<https://pytorch.org/>

Figure 5.6 – Some images from the Stanford cars dataset. The cars are presented in different positions and scenarios, and the images are of different sizes.

Source: (KRAUSE et al., 2013)

5.2.1 Effects of the stride value

In our first experiment, we evaluated how the classification performance of our approach varies according to the stride S in Algorithm 1 for generating patch-based feature graphs. Notice that, as discussed in Section 4.1, the stride S is the length that the window slides (vertically and horizontally) across the image in pixels, as a percentage of the feature extractor's input size.

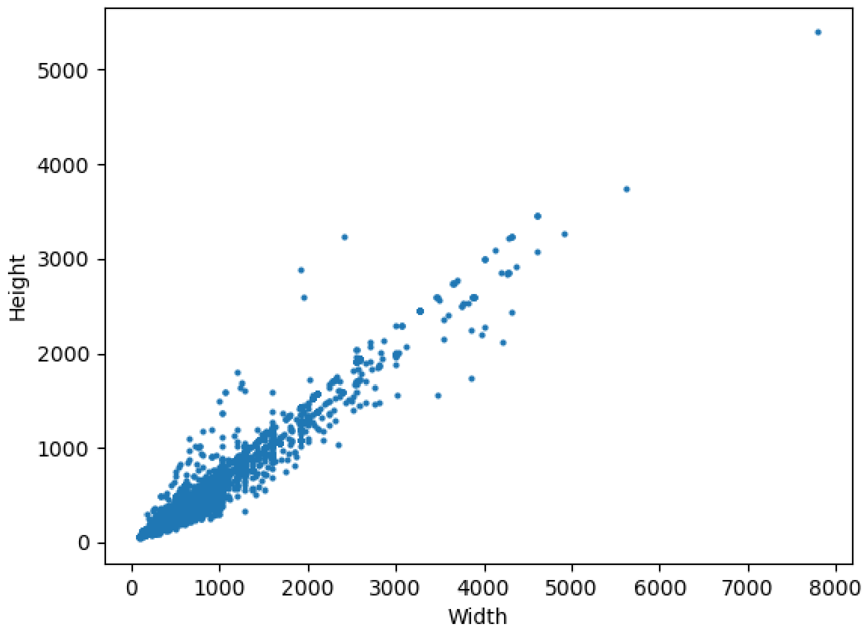
In this experiment, we tested four different stride values: 100%, 50%, 25%, and 10%. We considered only patch-based feature graphs with a single scale, ranging from $N = 1$ to $N = 5$. Notice that, by decreasing the value of S, the number of nodes in the resulting graph and the overlapping between adjacent patches increase.

The experiment was performed considering only CLIP as a feature extractor, since the goal of this experiment is not to compare the performance of each model. We adopted CLIP due to its impressive performance reported in the literature (PULS, 2023) and due to the amount of data used in its training.

Table 5.1 presents the results of this first experiment in the geological dataset and Table 5.2 report the results obtained in the Stanford Cars dataset.

The behavior observed in the two datasets is different, but we can notice a general pattern. The first four lines of Tables 5.1 and 5.2 present the results using $MGS(\{1\})$

Figure 5.7 – Distribution of the images according to their dimensions in the stanford cars dataset.



Source: The author

(i.e., $N = 1$), where we can see an increase in performance as we decrease the stride, in a way that $S = 100\%$ resulted in the worst performance and $S = 10\%$ resulted in the best one. When we analyze the other scales we can see a similar pattern. However, on scales $N = 2$ to $N = 5$, when decreasing S to 10%, the performance decreases drastically. Thus, on these scales, the performance increases as the value of S decreases, up to 25%, and decreases when $S = 10\%$. Thus, in both datasets, the performance obtained with MGS(2), MGS(3), MGS(4), and MGS(5) achieves the best result by adopting 25% as stride value and the worst result by adopting 10%.

The difference in performance caused by the stride S is remarkable. In the geological dataset, the difference between the best and worst performance varies from $\approx 0.5\%$ to $\approx 12\%$, depending on the scale of analysis (value of N). In the Stanford Cars dataset, on the other hand, the performance varies from $\approx 5\%$ to $\approx 60\%$, depending on the scale of analysis. In the cases of MGS(4) and MGS(5), by adopting $S = 10\%$ the resulting performance is remarkably poor on Stanford Cars dataset.

Figures 5.8 and 5.9 represent in a visual way how the value of S impacts the classification performance. In these figures, we adopted macro F1 scores for comparing the performance, since in our context we assign the same importance to every class in the dataset.

Scale	Stride	Top-1 Accuracy	Macro			Weighted		
			Precision	Recall	F1	Precision	Recall	F1
MGS({1})	100%	91.97%	83.31%	81.24%	82.13%	91.86%	91.97%	91.88%
MGS({1})	50%	92.05%	83.87%	81.51%	82.53%	91.94%	92.05%	91.96%
MGS({1})	25%	92.39%	84.36%	82.20%	83.15%	92.30%	92.39%	92.32%
MGS({1})	10%	92.67%	85.07%	82.89%	83.81%	92.60%	92.67%	92.60%
MGS({2})	100%	91.96%	83.15%	81.22%	82.07%	91.90%	91.97%	91.91%
MGS({2})	50%	92.26%	83.76%	81.52%	82.48%	92.21%	92.26%	92.20%
MGS({2})	25%	92.33%	83.79%	81.85%	82.69%	92.26%	92.33%	92.27%
MGS({2})	10%	90.68%	80.44%	77.93%	79.00%	90.50%	90.68%	90.55%
MGS({3})	100%	91.27%	81.66%	79.03%	80.18%	91.17%	91.27%	91.18%
MGS({3})	50%	91.52%	81.73%	79.39%	80.38%	91.44%	91.52%	91.45%
MGS({3})	25%	91.69%	82.03%	79.65%	80.66%	91.58%	91.69%	91.60%
MGS({3})	10%	86.45%	72.64%	68.43%	70.06%	86.15%	86.45%	86.19%
MGS({4})	100%	90.35%	79.14%	76.34%	77.46%	90.17%	90.35%	90.15%
MGS({4})	50%	90.75%	79.71%	77.41%	78.40%	90.58%	90.75%	90.62%
MGS({4})	25%	90.79%	80.01%	77.35%	78.43%	90.69%	90.79%	90.70%
MGS({4})	10%	81.37%	63.40%	57.70%	59.35%	80.86%	81.37%	80.81%
MGS({5})	100%	89.39%	76.78%	74.39%	75.29%	89.24%	89.39%	89.26%
MGS({5})	50%	89.96%	77.57%	75.38%	76.29%	89.79%	89.96%	89.83%
MGS({5})	25%	89.92%	78.26%	75.75%	76.79%	89.81%	89.92%	89.82%
MGS({5})	10%	77.50%	56.46%	50.44%	52.12%	76.72%	77.50%	76.73%

Table 5.1 – Experimental results using different stride values for different scales in the geological dataset. CLIP (RADFORD et al., 2021) was used as a feature extractor. We group the results according to the scale size (parameter N) and highlight in green the best results and in red the worsts.

By conducting this experiment, we were able to determine a threshold for increasing the graph size. We discovered that beyond this threshold, adding additional nodes (i.e., patches with extracted features) could actually have a detrimental effect on performance. Since $S = 25\%$ resulted in the best performance for scales $N = 2$ to $N = 5$, and achieved the second-best performance for $N = 1$, we adopted 25% of stride as the default value for the subsequent experiments discussed in Section 5.2.2.

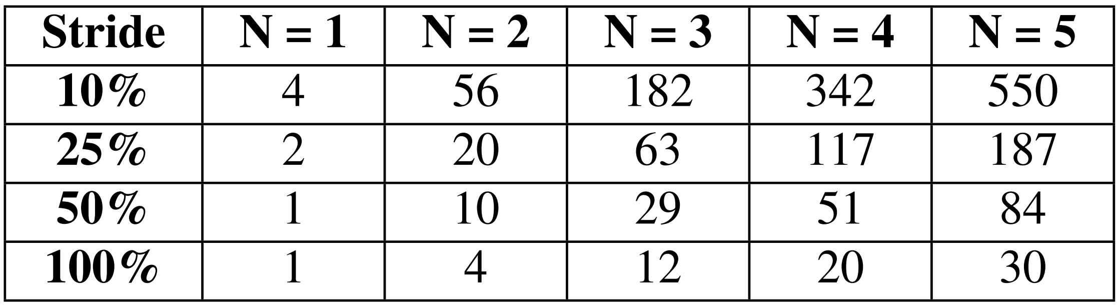
Table 5.3 shows the number of patches generated in each stride and scale variation in an image of 800x600 pixels. We can notice that a smaller stride generates a much bigger amount of patches and that this effect is more pronounced at higher values of N. The increase in the sizes of the resulting graphs can be a reason for the performance degradation observed when we consider 10% of stride and scales bigger than $N = 1$ in Figures 5.8 and 5.9.

5.2.2 Evaluation of the proposed approach

In this experiment, we compare our approach with three other different families of approaches. In the first one, we only used the feature extraction layers of the pre-trained DenseNet-121, which contains only convolution and pooling operations and can

Scale	Stride	Top-1 Accuracy	Macro			Weighted		
			Precision	Recall	F1	Precision	Recall	F1
MGS({1})	100%	79.51%	79.75%	79.44%	79.49%	79.73%	79.51%	79.52%
MGS({1})	50%	81.79%	81.95%	81.77%	81.79%	81.97%	81.79%	81.82%
MGS({1})	25%	83.12%	83.16%	83.02%	83.03%	83.21%	83.12%	83.11%
MGS({1})	10%	84.34%	84.41%	84.27%	84.26%	84.45%	84.34%	84.33%
MGS({2})	100%	62.68%	62.76%	82.69%	62.58%	62.70%	62.68%	62.55%
MGS({2})	50%	78.57%	78.74%	78.52%	78.55%	78.72%	78.57%	78.56%
MGS({2})	25%	79.84%	80.04%	79.76%	79.81%	80.03%	79.84%	79.85%
MGS({2})	10%	57.31%	57.33%	57.19%	57.11%	57.34%	57.31%	57.17%
MGS({3})	100%	56.11%	56.22%	56.06%	55.99%	56.12%	56.11%	55.96%
MGS({3})	50%	72.22%	72.31%	72.14%	72.14%	72.35%	72.22%	72.20%
MGS({3})	25%	75.05%	75.14%	74.93%	74.95%	75.18%	75.05%	75.03%
MGS({3})	10%	24.39%	23.54%	24.26%	23.59%	23.57%	24.39%	23.67%
MGS({4})	100%	48.30%	48.08%	48.31%	48.01%	47.95%	48.30%	47.94%
MGS({4})	50%	65.07%	65.10%	65.06%	64.96%	65.06%	65.07%	64.94%
MGS({4})	25%	67.56%	67.57%	67.48%	67.41%	67.61%	67.56%	67.47%
MGS({4})	10%	6.86%	5.75%	6.74%	5.67%	5.68%	6.86%	5.72%
MGS({5})	100%	40.56%	40.13%	40.65%	40.16%	39.97%	40.56%	40.04%
MGS({5})	50%	57.73%	57.66%	57.74%	57.55%	57.62%	57.73%	57.53%
MGS({5})	25%	61.49%	61.69%	61.43%	61.43%	61.66%	61.49%	61.45%
MGS({5})	10%	3.81%	3.17%	3.70%	2.72%	3.19%	3.81%	2.76%

Table 5.2 – Experimental results using different stride values for different scales in the Stanford Cars dataset. CLIP (RADFORD et al., 2021) was used as a feature extractor. We group the results according to the scale size (parameter N) and highlight in green the best results and in red the worsts.



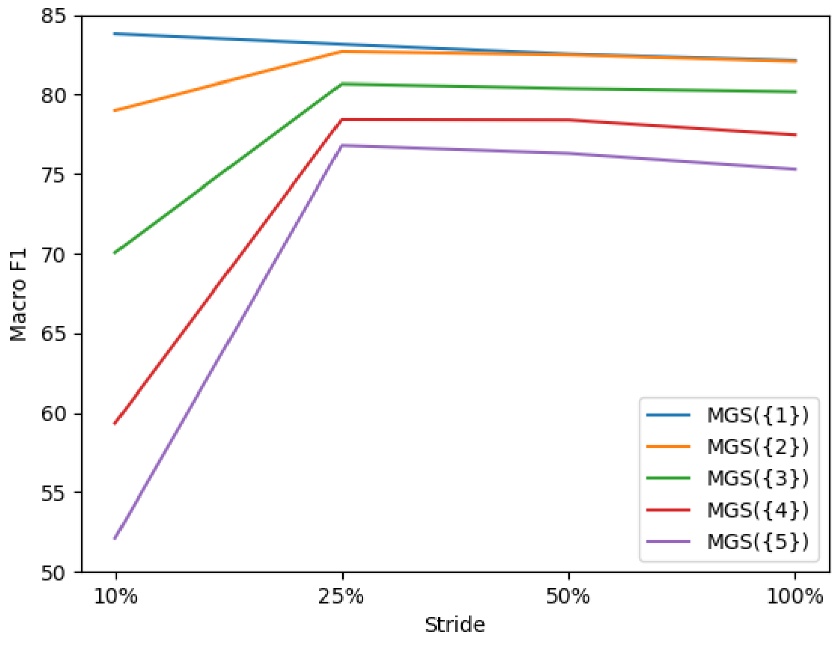
Stride	N = 1	N = 2	N = 3	N = 4	N = 5
10%	4	56	182	342	550
25%	2	20	63	117	187
50%	1	10	29	51	84
100%	1	4	12	20	30

Table 5.3 – Number of patches generated for each stride value and for each value N in an image of 800x600 pixels.

take input images of varying sizes. The only constraint of this model is that the smallest image dimension should be greater or equal to 224 pixels. Thus, in this case, the only preprocessing involved is for resizing (preserving the aspect ratio) images that are smaller than the input requirements of this model. We used the preprocessed raw images as input for this model, then, we used the features produced as an output for training a simple classifier of a fully-connected layer with linear activation. Notice that in this approach, we do not discard image information. Due to this, we refer to this methodology as *lossless* in Table 5.4 and Table 5.5. For testing, we performed feature extraction with this model over the images in the test set, and then, the features were evaluated by the trained model.

For the other two families of approaches, for each raw image, we resized its

Figure 5.8 – Macro F1 results in geological dataset by varying the stride value (10%, 25%, 50% and 100%) for each MGS (1-5).

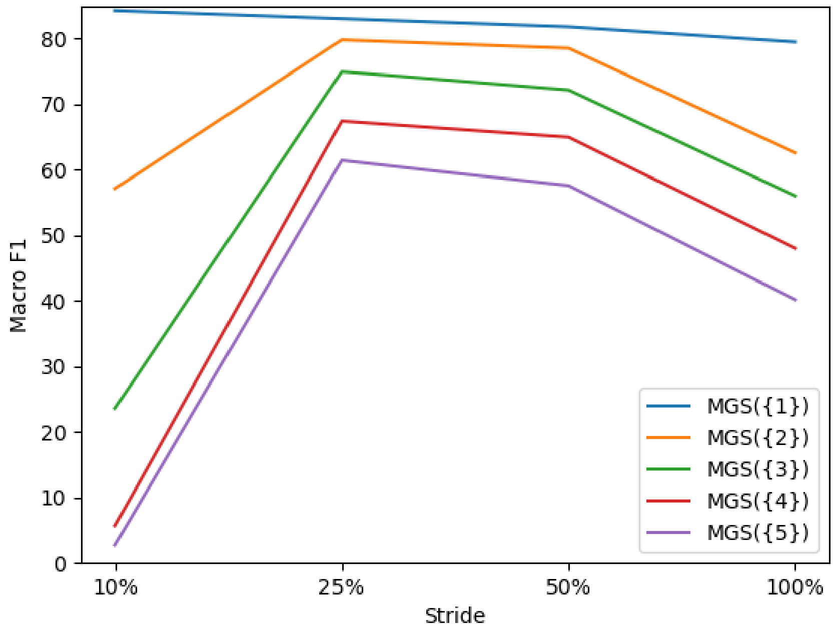


Source: The author

smaller edge to the size required as input by each feature extractor, preserving the original aspect ratio, and performed a center crop, keeping only the central part of the image. The first of these two families of approaches considers the resulting fragment of the original image as input of a pre-trained model used as a feature extractor, and the resulting features are considered for representing the image. In this case, we used the same three pre-trained models previously mentioned for extracting these *simple features* (SF). Notice that this approach also uses transfer learning from the same pre-trained models, but uses only the information of an arbitrary (central) part of the original image. We used the same simple classifier of the first approach for evaluating this one. We applied in the test set the same pre-processing step applied in the training images. This approach was included in the comparison because it represents a common approach for image classification based on neural networks using feature extraction.

Finally, we also trained an end-to-end DenseNet architecture (with the default classifier layer) without pre-training, using as input only the fragments of the raw image resulting from resizing and cropping (as previously discussed). We include this approach in our experiments to compare our approach with the performance achieved by a sophisticated architecture without adopting transfer learning. In this approach, the model input is the *raw image* information, contrasting with the other approaches that use input features

Figure 5.9 – Macro F1 results in Stanford Cars dataset by varying the stride value (10%, 25%, 50% and 100%).



Source: The author

provided directly by a pre-trained feature extractor.

We refer to these two last families of approaches that discard some image information as *lossy* in Table 5.4 and Table 5.5.

Each experiment is organized in a separate row in the Tables 5.4 and 5.5, corresponding to each specific approach. We grouped our results according to the input information required for each compared approach. In order to improve readability, we also specify in the tables if the approach uses transfer learning (TL column) and the metrics mentioned in the methodology. In the following, we will discuss the results of Table 5.4, which reports the results obtained in the dataset of geological images.

Line 1 presents the performance of the ResNeXt model trained using only parts of the raw images (lossy information) as input, without taking advantage of transfer learning. This approach presents the worst performance amongst the compared approaches, demonstrating that using transfer learning significantly improved the results in this dataset. This result is expected since the dataset is reasonably small and contains a high variability regarding visual features for each class.

Lines 2-5 present the approaches that adopt simple features (SF) as input. Line 2 presents the results of using the features extracted by DenseNet, taking the whole image as input (*lossless* information). The results obtained by this approach are very similar to

Row	Model	TL	Input Information	Top-1 Accuracy	Macro			Weighted		
					Precision	Recall	F1	Precision	Recall	F1
1	ResNeXt	⨯	Raw image(lossy)	84.26%	72.49%	69.23%	70.4%	84.36%	84.26%	84.15%
2	DenseNet	✓	SF(lossless)	90.23%	81.16%	78.08%	79.39%	90.11%	90.23%	90.1%
3	ResNeXt	✓	SF(lossy)	87.77%	76.66%	72.77%	74.37%	87.45%	87.77%	87.53%
4	DenseNet	✓	SF(lossy)	89.87%	81.73%	77.91%	79.45%	89.65%	89.87%	89.68%
5	**CLIP**	✓	**SF(lossy)**	**92.86%**	**86.33%**	**83.16%**	**84.45%**	**92.74%**	**92.86%**	**92.74%**
6	ResNeXt	✓	*MGS({1})*	87.73%	75.43%	72.41%	73.51%	87.50%	87.73%	87.52%
7	DenseNet	✓	*MGS({1})*	89.46%	79.49%	76.14%	77.57%	89.25%	89.46%	89.30%
8	**CLIP**	✓	***MGS({1})***	**92.39%**	**84.36%**	**82.20%**	**83.15%**	**92.30%**	**92.39%**	**92.32%**
9	ResNeXt	✓	*MGS({2})*	88.44%	76.92%	73.67%	75.00%	88.30%	88.44%	88.31%
10	DenseNet	✓	*MGS({2})*	90.17%	80.43%	77.43%	78.68%	89.99%	90.17%	90.03%
11	**CLIP**	✓	***MGS({2})***	**92.33%**	**83.79%**	**81.85%**	**82.69%**	**92.26%**	**92.33%**	**92.27%**
12	ResNeXt	✓	*MGS({3})*	88.38%	76.38%	72.44%	73.92%	88.11%	88.38%	88.16%
13	DenseNet	✓	*MGS({3})*	89.79%	78.86%	75.81%	77.03%	89.59%	89.79%	89.64%
14	**CLIP**	✓	***MGS({3})***	**91.69%**	**82.03%**	**79.65%**	**80.66%**	**91.58%**	**91.69%**	**91.60%**
15	ResNeXt	✓	*MGS({4})*	87.88%	75.05%	71.29%	72.74%	87.61%	87.88%	87.66%
16	DenseNet	✓	*MGS({4})*	89.37%	78.25%	74.86%	76.25%	89.16%	89.37%	89.21%
17	**CLIP**	✓	***MGS({4})***	**90.79%**	**80.01%**	**77.35%**	**78.43%**	**90.69%**	**90.79%**	**90.70%**
18	ResNeXt	✓	*MGS({5})*	87.49%	74.10%	70.50%	71.89%	87.20%	87.49%	87.28%
19	DenseNet	✓	*MGS({5})*	88.86%	77.22%	73.66%	74.96%	88.66%	88.86%	88.69%
20	**CLIP**	✓	***MGS({5})***	**89.92%**	**78.26%**	**75.75%**	**76.79%**	**89.81%**	**89.92%**	**89.82%**
21	ResNeXt	✓	*MGS({1,2})*	89.55%	79.19%	75.90%	77.26%	89.37%	89.55%	89.39%
22	DenseNet	✓	*MGS({1,2})*	90.83%	82.28%	78.68%	80.13%	90.70%	90.83%	90.71%
23	**CLIP**	✓	***MGS({1,2})***	**93.22%**	**85.78%**	**85.60%**	**84.56%**	**93.12%**	**93.22%**	**93.13%**
24	ResNeXt	✓	*MGS({1,2,3})*	90.01%	79.49%	76.71%	77.83%	89.88%	90.01%	89.90%
25	DenseNet	✓	*MGS({1,2,3})*	91.36%	82.63%	80.19%	81.27%	91.19%	91.36%	91.24%
26	**CLIP**	✓	***MGS({1,2,3})***	**93.46%**	**86.78%**	**84.31%**	**85.36%**	**93.37%**	**93.46%**	**93.39%**
27	ResNeXt	✓	*MGS({1,2,3,4})*	90.27%	80.04%	77.67%	78.64%	90.19%	90.27%	90.17%
28	DenseNet	✓	*MGS({1,2,3,4})*	91.61%	83.31%	80.40%	81.63%	91.56%	91.61%	91.54%
29	**CLIP**	✓	***MGS({1,2,3,4})***	**93.46%**	**86.01%**	**84.43%**	**85.10%**	**93.43%**	**93.46%**	**93.42%**
30	ResNeXt	✓	*MGS({1,2,3,4,5})*	90.56%	80.27%	77.59%	78.74%	90.40%	90.56%	90.43%
31	DenseNet	✓	*MGS({1,2,3,4,5})*	91.67%	83.10%	80.85%	81.83%	91.58%	91.67%	91.60%
32	**CLIP**	✓	***MGS({1,2,3,4,5})***	**93.51%**	**86.36%**	**84.63%**	**85.41%**	**93.45%**	**93.51%**	**93.45%**

Table 5.4 – Comparison of the metrics achieved by each experiment configuration in the geological dataset. Each row represents a different configuration. We divided these setups in the table according to the input information (Column 4) and the use of transfer learning (Column 3). Column 4 *Raw Image (lossy)* indicates that raw image is used as an input with resize and center crop to 224x224 input size. *SF* indicates that *Simple Features* are used as input. *Lossless* indicates that the original image information is used as input and only images with the smallest dimension small than 224 pixels are resized, whereas *lossy* indicates that resizing and cropping are performed, discarding information. Lines 6-20 show the results achieved by our approach with different scales, while lines 21-32 present the results achieved when combining different scales.

those of Line 4, which also uses DenseNet but discards some image information by cropping the images after resizing them. Thus, the results suggest that resizing and cropping images produce a reasonable representation of the information contained in the whole image. In Line 4, only macro precision and macro F1 are slightly superior to the results in Line 2.

In Lines 3-5 we report the results achieved by using the simple features with the resize and crop (*lossy*). CLIP as a feature extractor outperformed DenseNet and Resnext. These results suggest that CLIP can produce more representative features from the lossy information of the images obtained after resizing and cropping operations.

In the following discussion, we focus on analyzing the performance of our approach and comparing it with the approaches that adopt *simple features* as input. Lines 6-20 present the results achieved using just a single scale of analysis to represent each image ($MGS(\{1\})$, $MGS(\{2\})$, $MGS(\{3\})$, $MGS(\{4\})$, and $MGS(\{5\})$). Thus, in this context, the multiscale graph set consists of only one patch-based feature graph per

Row	Model	TL	Input Information	Top-1 Accuracy	Macro			Weighted		
					Precision	Recall	F1	Precision	Recall	F1
1	ResNeXt	✗	Raw Image (lossy)	23.85%	25.44%	23.92%	23.93%	25.38%	23.85%	23.87%
2	DenseNet	✓	SF(lossless)	57.94%	59.30%	57.90%	58.21%	59.08%	57.94%	58.12%
3	ResNeXt	✓	SF(lossy)	47.61%	47.56%	47.67%	47.41%	47.37%	47.61%	47.29%
4	DenseNet	✓	SF(lossy)	57.33%	57.59%	57.34%	57.25%	57.39%	57.33%	57.15%
5	**CLIP**	✓	SF(lossy)	**83.29%**	**83.44%**	**83.20%**	**83.24%**	**83.44%**	**83.29%**	**83.29%**
6	ResNeXt	✓	*MGS({1})*	45.98%	45.82%	46.12%	45.82%	45.60%	45.98%	45.63%
7	DenseNet	✓	*MGS({1})*	55.83%	55.81%	55.81%	55.65%	55.70%	55.83%	55.61%
8	**CLIP**	✓	***MGS({1})***	**83.12%**	**83.16%**	**83.02%**	**83.03%**	**83.21%**	**83.12%**	**83.11%**
9	ResNeXt	✓	*MGS({2})*	55.49%	55.56%	55.46%	55.37%	55.47%	55.49%	55.35%
10	DenseNet	✓	*MGS({2})*	66.76%	66.93%	66.68%	66.70%	66.91%	66.76%	66.73%
11	**CLIP**	✓	***MGS({2})***	**79.84%**	**80.04%**	**79.76%**	**79.81%**	**80.03%**	**79.84%**	**79.85%**
12	ResNeXt	✓	*MGS({3})*	55.80%	55.74%	55.76%	55.61%	55.72%	55.80%	55.62%
13	DenseNet	✓	*MGS({3})*	66.18%	66.31%	66.07%	66.08%	66.34%	66.18%	66.15%
14	**CLIP**	✓	***MGS({3})***	**75.05%**	**75.14%**	**74.93%**	**74.95%**	**75.18%**	**75.05%**	**75.03%**
15	ResNeXt	✓	*MGS({4})*	53.64%	53.77%	53.49%	53.47%	53.74%	53.64%	53.53%
16	DenseNet	✓	*MGS({4})*	64.18%	64.43%	64.16%	64.17%	64.42%	64.18%	64.18%
17	**CLIP**	✓	***MGS({4})***	**67.56%**	**67.57%**	**67.48%**	**67.41%**	**67.61%**	**67.56%**	**67.47%**
18	ResNeXt	✓	*MGS({5})*	51.24%	51.39%	51.20%	51.17%	51.33%	51.24%	51.15%
19	DenseNet	✓	*MGS({5})*	62.70%	62.72%	62.65%	62.57%	62.75%	62.70%	62.62%
20	**CLIP**	✓	***MGS({5})***	**61.49%**	**61.69%**	**61.43%**	**61.43%**	**61.66%**	**61.49%**	**61.45%**
21	ResNeXt	✓	*MGS({1,2})*	59.87%	60.20%	59.89%	59.92%	60.03%	59.87%	59.83%
22	DenseNet	✓	*MGS({1,2})*	70.45%	70.71%	70.39%	70.44%	70.67%	70.45%	70.45%
23	**CLIP**	✓	***MGS({1,2})***	**85.88%**	**86.01%**	**85.82%**	**85.86%**	**86.05%**	**85.88%**	**85.91%**
24	ResNeXt	✓	*MGS({1,2,3})*	63.48%	63.84%	63.47%	63.52%	63.76%	63.48%	63.49%
25	DenseNet	✓	*MGS({1,2,3})*	74.62%	74.92%	74.55%	74.64%	74.92%	74.62%	74.67%
26	**CLIP**	✓	***MGS({1,2,3})***	**86.79%**	**86.90%**	**86.69%**	**86.72%**	**86.98%**	**86.79%**	**86.81%**
27	ResNeXt	✓	*MGS({1,2,3,4})*	64.60%	65.09%	64.52%	64.65%	65.05%	64.60%	64.68%
28	DenseNet	✓	*MGS({1,2,3,4})*	76.02%	76.32%	76.00%	76.04%	76.28%	76.02%	76.04%
29	**CLIP**	✓	***MGS({1,2,3,4})***	**86.57%**	**86.69%**	**86.50%**	**86.52%**	**86.77%**	**86.57%**	**86.60%**
30	ResNeXt	✓	*MGS({1,2,3,4,5})*	66.74%	67.18%	66.65%	66.77%	67.14%	66.74%	66.79%
31	DenseNet	✓	*MGS({1,2,3,4,5})*	77.61%	77.83%	77.52%	77.57%	77.83%	77.61%	77.62%
32	**CLIP**	✓	***MGS({1,2,3,4,5})***	**86.28%**	**86.32%**	**86.18%**	**86.18%**	**86.43%**	**86.28%**	**86.29%**

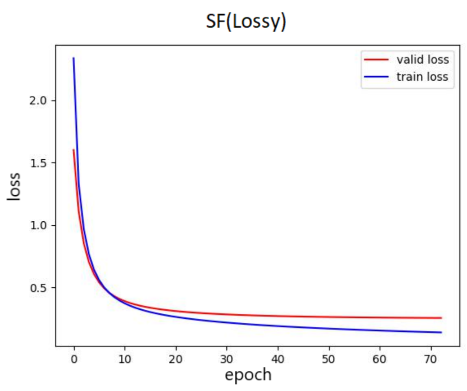
Table 5.5 – Achieved results in the Stanford Cars dataset. This table adopts the same notation adopted in Table 5.4.

image. In these settings, by using ResNeXt with scales $N = 2$ and $N = 3$ and by using DenseNet with scale $N = 2$, our approach outperforms the approaches that take as input only simple features and the same feature extractors. However, in most cases, the results obtained by our approach are lower or similar to those achieved by approaches that adopt simple features as input. As we will show later, our approach effectively improves the results only when we combine different scales. We can also notice a superiority of CLIP over DenseNet and ResNeXt. In these experiments, we can also notice that by adopting bigger scales, the performance of our approach decreases. The worst results in these settings were obtained by $MGS(\{4\})$ and $MGS(\{5\})$. A possible hypothesis for explaining the performance degradation observed when we consider scales $N = 4$ and $N = 5$ is that contextual information may be lost, since patches capture smaller areas at bigger scales. Since the performance decreased as the scale increased, we decided to limit the maximum scale considered in this experiment as $N = 5$. It is important to emphasize that our method can generate patch-based feature graphs for greater scales. However, as the scale increases, the memory necessary for storing patch-based feature graphs increases, and, due to this, it is necessary to evaluate if it is worth considering larger scales.

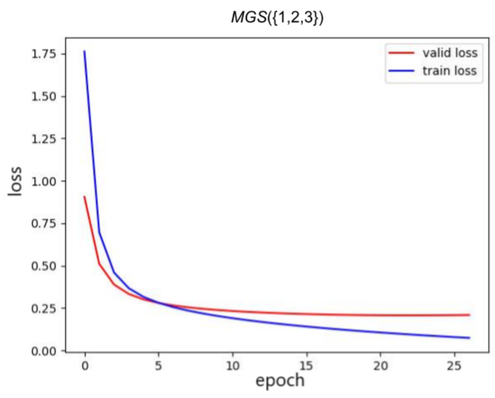
Lines 21-32 present the results achieved by combining in the input graph set multiple patch-based feature graphs representing multiple scales of each image ($MGS(\{1, 2\})$,

Figure 5.10 – Learning curves that show the loss during the training and validation phase, using CLIP in the folds with the best result.

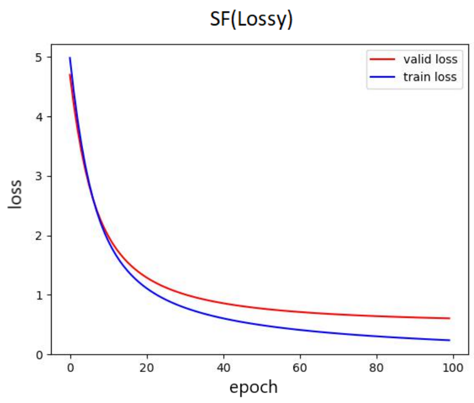
(a) Training on the geological dataset using simple features (SF - Lossy).



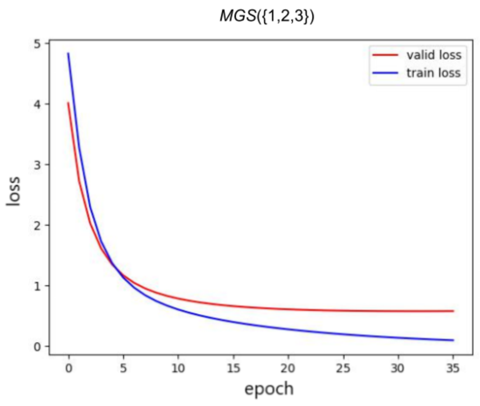
(b) Training on the geological dataset using three graph scales ($MGS(\{1,2,3\})$).



(c) Training on Cars dataset using simple features (SF - Lossy).



(d) Training on Cars dataset using three graph scales ($MGS(\{1,2,3\})$).



Source: The author

$MGS(\{1,2,3\})$, $MGS(\{1,2,3,4\})$, and $MGS(\{1,2,3,4,5\})$). Since the previous experiments showed that smaller scales achieved better performance than larger scales, we performed the experiments using the smaller scales first and iteratively added a larger scale in each experiment.

Lines 21-23 present the performance of our approach combining scales $N = 1$ and $N = 2$ (i.e., $MGS(\{1,2\})$). In these settings, our approach obtained presents better results than by considering only single scales (lines 6-17). This experiment showed that combining scales can increase the overall performance of our approach. In lines 24-26, when we add the scale $N = 3$ (i.e., $MGS(\{1,2,3\})$), our approach achieves superior results when compared to the scenario in which we adopt only simple features (lines 2-5), reaching 3.46% of improvement in Macro F1 using ResNeXt, 1.82% of improvement using DenseNet, and around 1% of improvement by using CLIP.

Lines 27-29 present the performance of our approach when we combine the scales $N = 1$, $N = 2$, $N = 3$ and $N = 4$ (i.e., $MGS(\{1, 2, 3, 4\})$). In this setting, when adopting ResNeXt and DenseNet, our approach outperforms the results achieved by adopting $MGS(1, 2, 3)$ as input. We can notice an improvement of around 1% in Macro F1 for ResNeXt over the performance achieved using $MGS(\{1, 2, 3\})$. By using DenseNet, our approach improves on all metrics, most notably in Macro precision, increasing around 1% over the performance achieved using $MGS(\{1, 2, 3\})$. Also, by using CLIP, our approach achieves similar results to the previous combination, with weighted metrics improving slightly, but showing a more significant decrease in performance in the macro precision.

Finally, lines 30-32 present the results achieved by our approach when we add the scale $N = 5$ to the MGS (i.e. $MGS(\{1, 2, 3, 4, 5\})$). This graph set shows the best result for most metrics in all three models. The difference between the performance achieved in this setting and the performance achieved by using only simple features (Lines 3-5) as input is around 4.5% of Macro F1 when using ResNeXt as feature extractor and around 2.5% when using DenseNet. By using CLIP the difference is smaller, reaching around 1% of improvement in all metrics. However, it achieves 85.41% of macro F1, which is the best overall performance achieved in the geological dataset.

Next, we will discuss the results regarding the Stanford Cars dataset, reported in Table 5.5. Generally, the results obtained in this dataset follow the patterns observed in the dataset of geological images. However, in this dataset, the results show that our approach achieves a more significant performance increase, when compared with the alternative approaches.

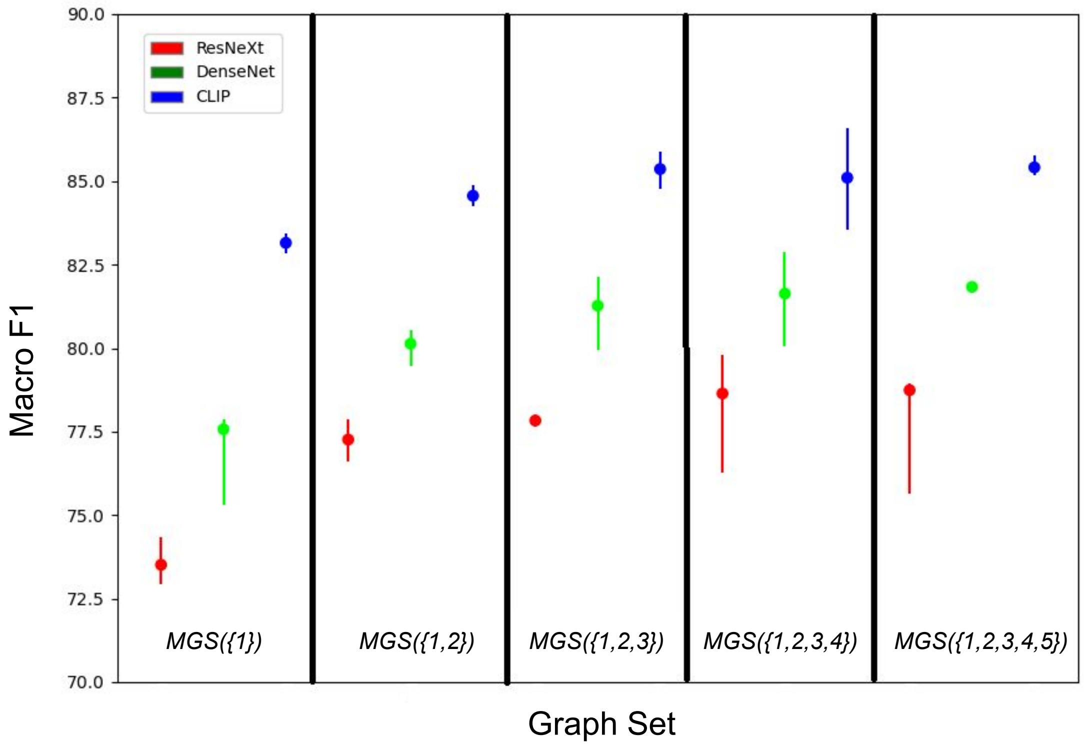
We can notice an important difference compared to the behavior observed in the dataset of geological images regarding the performance of our approach when adopting single scales ($MGS(\{1\})$, $MGS(\{2\})$, $MGS(\{3\})$, $MGS(\{4\})$, and $MGS(\{5\})$). When considering $N = 2$, $N = 3$, $N = 4$, and $N = 5$, and adopting DenseNet and ResNeXt (Lines 9, 10, 12, 13, 15, 16, 18, and 19), our approach achieves a performance comparable to the performance achieved when adopting simple features as input (Lines 2-5). We can notice a performance increase ranging from $\approx$5% to $\approx$9% in all metrics. This improvement achieved by adopting single scales does not occur when we adopt CLIP as a feature extractor. We can also notice that CLIP performs better in single scales using smaller scales, such as $N = 1$, while by adopting Resnext and Densenet, our approach achieves the best results using $N = 2$ and $N = 3$, respectively. These results also demonstrate a heterogeneity regarding the scales at which each model performs better in each dataset.

When combining different scales in this dataset, our approach achieves the best results, following the same pattern observed in the geological dataset. When scales $N = 1$ and $N = 2$ are combined (i.e., $MGS(\{1, 2\})$), in lines 21-23, it is possible to notice an overall increase in all metrics, with the three different feature extractors, in comparison with the performance obtained by the approaches using only simple features and the performance obtained by our approach considering only single scales. The performance of our approach still increases when we combine the scales $N = 1$, $N = 2$, and $N = 3$ (i.e., $MGS(\{1, 2, 3\})$), as shown in lines 24-26. We can observe this effect for all the feature extractors. When we combine the scales $N = 1$, $N = 2$, $N = 3$, and $N = 4$ (i.e., $MGS(\{1, 2, 3, 4\})$), our approach's performance still increases when adopting ResNeXt and Densenet as feature extractors, but we can notice a performance drop when adopting CLIP. The same behavior can be observed when we use the set with five scales $MGS(\{1, 2, 3, 4, 5\})$. In the geological dataset, CLIP still achieves a slight performance improvement using scales greater than $N = 3$ in the graph set, while in the Stanford cars dataset this behavior does not occur. It is possible to notice that the characteristics of the datasets and models can influence the behavior in different scales.

In the stanford cars dataset, when adopting DenseNet and ResNeXt as feature extractors, the best results were achieved by considering $MGS(\{1, 2, 3, 4, 5\})$ as input (Lines 30 and 31). On the other hand, when adopting CLIP as feature extractor, the best result was achieved when considering $MGS(\{1, 2, 3\})$ as input (Line 26), which is also the best overall result in this dataset, reaching 86.79% of accuracy and 86.72% of macro F1. This result follows the same pattern observed in the geological dataset, whose best result was also achieved by using $MGS(\{1, 2, 3\})$ as input and CLIP as feature extractor. When compared with the approaches using simple features as input (Lines 2-5), our approach using these graph sets improves, in all metrics, ≈19% when adopting ResNeXt, ≈19% when adopting DenseNet, and ≈3% with CLIP.

Figures 5.11 and 5.12 visually represent the performance achieved by our approach with different scales and their combinations. In these Figures, each bar represents the Macro F1 and best and worst F1 measures among all the folds during the cross-validation process and each point represents the average considering all folds. These figures emphasize the pattern of performance improvement obtained by adding scales to represent the images. They also show that by using CLIP as feature extractor, there is no

Figure 5.11 – Evolution of the classification performance in the geological dataset as scales are added in the graph set. Each bar represents the F1 measure obtained in the worst and best folds in the cross-validation process and each point represents the average considering all folds.

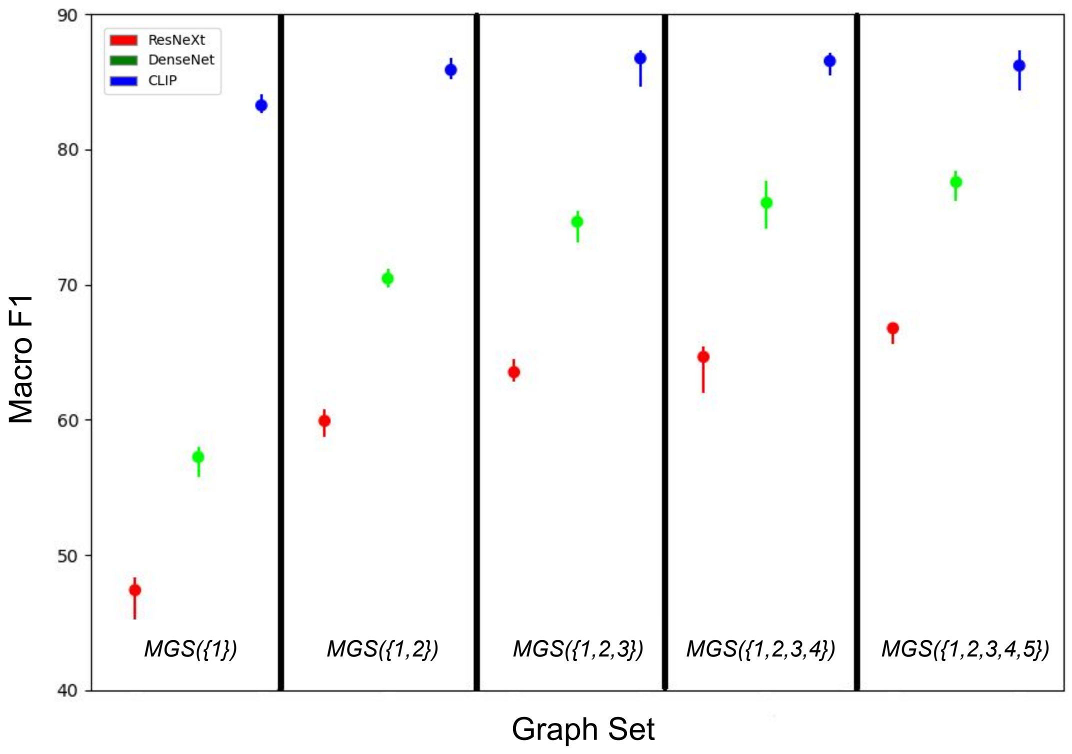


Source: The author

performance improvement by adding the $N = 4$ and $N = 5$ scales in the Stanford Cars, differently from the case of the geological dataset. We can also notice that by adopting DenseNet and ResNeXt as feature extractors, the best results were achieved by using $MGS(\{1, 2, 3, 4, 5\})$ as input.

Figure 5.10 shows the learning curves of the training process of our approach and the approach using simple features as input. In our approach, we adopted CLIP as feature extractor and the $MGS(\{1, 2, 3\})$ as input. This configuration achieved the best result in Stanford Cars and the third-best result in the geological dataset. The charts represent the training process focusing on the folds with the best results in our cross-validation procedure. In the first two charts at the top, we present the training on the geological dataset (a) adopting simple features as input and (b) using our approach and considering $MGS(\{1, 2, 3\})$ as input. The two graphs at the bottom (c,d) show the training of the same approaches on the Stanford Cars dataset. We can observe that the proposed approach allows the classifier to achieve a lower loss in fewer epochs than when considering as input simple features extracted after the pre-processing of the images. These learning curves show that our approach performs better with less training time. The learning curves also

Figure 5.12 – Evolution of the classification performance in the Cars dataset as new graph scales is added in the graph set. Each bar represents the F1 measure obtained in the worst and best folds in the cross-validation process and each point represents average considering all folds.



Source: The author

demonstrate that overfitting does not happen during the training process.

As a general conclusion of these experiments, in both datasets, representing images with patch-based feature graphs of different scales showed performance improvements when compared to the approaches adopting simple features as input. This suggests that multiscale graph sets capture a more informative representation of the image information that our multiscale graph convolutional network architecture was able to exploit for obtaining results that outperformed the alternative approaches. It is important to notice that the performance improvement by using our approach was observed for all the pre-trained models used as feature extractors. This also demonstrated that our approach is agnostic to the adopted feature extraction model.

6 CONCLUSION

In this work, we propose an approach for image classification using multiscale patch-based feature graphs. We resize images to multiple scales and split them into overlapping patches. After, we use transfer learning to extract features from the patches and join them into graphs. A node represents the features extracted from each patch, and the edges represent the neighborhood between adjacent patches. Finally, we use multiple patch-based feature graphs for representing each image and this set of graphs is taken as input by a graph convolutional network.

Our approach was designed for dealing with challenges that are common when dealing with real-world datasets such as the geological dataset which is the main focus of this work. Our methodology allows us to deal with images of varying sizes without introducing noise or discarding information. By adopting overlapping patches at different scales, we highlight different visual features of the original image, and by using graph convolution networks, we can process graphs of multiple sizes. The adoption of transfer learning also allows our approach to deal with small datasets.

We performed two different experiments. The first was carried out for understanding how beneficial the overlapping between patches is for the results. In this experiment, we compared the performance achieved by our approach with different values of the stride that controls the sliding window that extracts patches from the images. This experiment showed that a tiny stride is detrimental, and the highest performance point was using around 25% of stride value.

In the second experiment, we compared our approach with conventional image classification approaches. Our results showed that the proposed approach achieved the best performance in the two datasets considered in the proposed experiments. Our approach improved the classification performance with all feature extraction models considered in the experiment, suggesting that the proposed approach is agnostic to the choice of feature extractor. The experiments also showed that by using CLIP as feature extractor our approach achieves the best results in all experimental setups, outperforming the results achieved by adopting DenseNet and ResNeXt as feature extractors.

It is worth noting that while fine-tuning is a powerful technique, we were able to achieve satisfactory performance through feature extraction alone. One key advantage of feature extraction is that it requires retraining only in the classifier module, which can save time and resources.

The main limitations of our approach are related to the process of generating patch-based feature graphs from images and the size of the resulting graphs. The pre-processing phase of our approach that is necessary to transform images in patch-based feature graphs is complex and computationally costly. In addition, after building the patch-based feature graphs, it is necessary to keep them in memory, which can become an issue, since the memory cost increases depending on the stride, the scales, and the number of scales considered simultaneously.

In future works, we plan to investigate how and why the results vary depending on the scale used and how much this relates to the characteristics of the pre-trained models and the datasets used. Also, it would be interesting to investigate methods for reducing the size of the graphs and then reducing memory costs. One possible solution would be to analyze each patch's importance to dropout uninformative nodes and keep only the most relevant information represented in each graph. This would also allow us to explore the flexibility of representing images as graphs, since by discarding some patches of the original image, the graphs built from the remaining patches can have irregular structures. We plan also to explore different types of neighborhoods between nodes and different schemes for assigning weights to edges. Performing experiments on more datasets and using other models can also bring insights. Future works can also investigate how data augmentation techniques affect the performance of our approach.

Part of the contributions of this work was presented in (TODESCATO. et al., 2023).

REFERENCES

ABBAS, A.; ABDELSAMEA, M. M.; GABER, M. M. Classification of covid-19 in chest x-ray images using detrac deep convolutional neural network. **Applied Intelligence**, Springer, v. 51, n. 2, p. 854–864, 2021.

ABEL, M. et al. A knowledge organization system for image classification and retrieval in petroleum exploration domain. In: **ONTOBRAS**. [S.l.: s.n.], 2019.

AHUJA, S. et al. Deep transfer learning-based automated detection of covid-19 from lung ct scan slices. **Applied Intelligence**, Springer, v. 51, p. 571–585, 2021.

ALPAYDIN, E. **Introduction to machine learning**. [S.l.]: MIT press, 2020.

ANTHIMOPOULOS, M. et al. Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. **IEEE transactions on medical imaging**, IEEE, v. 35, n. 5, p. 1207–1216, 2016.

ARAÚJO, T. et al. Classification of breast cancer histology images using convolutional neural networks. **PloS one**, Public Library of Science San Francisco, CA USA, v. 12, n. 6, p. e0177544, 2017.

ARRUZZO, F. **Mapeamento geológico estrutural da Formação Tiradentes, Mesoproterozóico, São João del Rei**. Bachelor's Thesis — Instituto de Geociências, UFRJ, 2016.

AVELAR, P. H. et al. Superpixel image classification with graph attention networks. In: IEEE. **2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)**. [S.l.], 2020. p. 203–209.

BAE, J.-H. et al. Superpixel image classification with graph convolutional neural networks based on learnable positional embedding. **Applied Sciences**, MDPI, v. 12, n. 18, p. 9176, 2022.

BAKER, N. A.; ZENGELER, N.; HANDMANN, U. A transfer learning evaluation of deep neural networks for image classification. **Machine Learning and Knowledge Extraction**, MDPI, v. 4, n. 1, p. 22–41, 2022.

BARBURICEANU, S.; TEREBEș, R. Automatic detection of melanoma by deep learning models-based feature extraction and fine-tuning strategy. In: IOP PUBLISHING. **IOP Conference Series: Materials Science and Engineering**. [S.l.], 2022. v. 1254, n. 1, p. 012035.

BARSTUGAN, M.; OZKAYA, U.; OZTURK, S. Coronavirus (covid-19) classification using ct images by machine learning methods. **arXiv preprint arXiv:2003.09424**, 2020.

BOYD, A.; CZAJKA, A.; BOWYER, K. Deep learning-based feature extraction in iris recognition: Use existing models, fine-tune or train from scratch? In: IEEE. **2019 IEEE 10th International Conference on Biometrics Theory, Applications and Systems (BTAS)**. [S.l.], 2019. p. 1–9.

BRONSTEIN, M. M. et al. Geometric deep learning: going beyond euclidean data. **IEEE Signal Processing Magazine**, IEEE, v. 34, n. 4, p. 18–42, 2017.

CELIK, Y. et al. Automated invasive ductal carcinoma detection based using deep transfer learning with whole-slide images. **Pattern Recognition Letters**, Elsevier, v. 133, p. 232–239, 2020.

CHEN, C.-F. R.; FAN, Q.; PANDA, R. Crossvit: Cross-attention multi-scale vision transformer for image classification. In: **Proceedings of the IEEE/CVF international conference on computer vision**. [S.l.: s.n.], 2021. p. 357–366.

CHEN, X. et al. Cyclic cnn: image classification with multiscale and multilocation contexts. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 9, p. 7466–7475, 2020.

CHOLLET, F. **Deep learning with Python**. [S.l.]: Simon and Schuster, 2021.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255.

DING, Y. et al. Multiscale graph sample and aggregate network with context-aware learning for hyperspectral image classification. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, IEEE, v. 14, p. 4561–4572, 2021.

DOSOVITSKIY, A. et al. An image is worth 16x16 words: Transformers for image recognition at scale. **arXiv preprint arXiv:2010.11929**, 2020.

DUNG, C. V. et al. Autonomous concrete crack detection using deep fully convolutional neural network. **Automation in Construction**, Elsevier, v. 99, p. 52–58, 2019.

GENÇAY, R.; QI, M. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. **IEEE Transactions on Neural Networks**, IEEE, v. 12, n. 4, p. 726–734, 2001.

GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow**. [S.l.]: " O'Reilly Media, Inc.", 2022.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016.

GRAUMAN, K.; DARRELL, T. The pyramid match kernel: Efficient learning with sets of features. **Journal of Machine Learning Research**, v. 8, n. 4, 2007.

HAN, F. et al. Underwater image processing and object detection based on deep cnn method. **Journal of Sensors**, Hindawi, v. 2020, 2020.

HAN, K. et al. Vision gnn: An image is worth graph of nodes. **arXiv preprint arXiv:2206.00272**, 2022.

HE, K. et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778.

HOLLINK, L. et al. Semantic annotation of image collections. In: **Knowledge capture**. [S.l.: s.n.], 2003. v. 2.

HONG, D. et al. More diverse means better: Multimodal deep learning meets remote-sensing imagery classification. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 59, n. 5, p. 4340–4354, 2020.

HORRY, M. J. et al. Covid-19 detection through transfer learning using multimodal imaging data. **Ieee Access**, IEEE, v. 8, p. 149808–149824, 2020.

HUANG, G. et al. Densely connected convolutional networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 4700–4708.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, Wiley-Blackwell, v. 160, n. 1, p. 106, 1962.

IVANOFF, M. **Sedimentação da Lagoa de Itapeva, RS - BRASIL**. Dissertation (Master) — Instituto de Geociências, UFRGS, 2013.

JING, H. et al. Hyperspectral image classification with a multiscale fusion-evolution graph convolutional network based on a feature-spatial attention mechanism. **Remote Sensing**, MDPI, v. 14, n. 11, p. 2653, 2022.

JúNIOR, V. M. **Geoquímica orgânica das turfeiras das praias de Hermenegildo e Maravilhas**. Dissertation (Master) — Instituto de Química, UFRGS, 2014.

KIEFFER, B. et al. Convolutional neural networks for histopathology image classification: Training vs. using pre-trained networks. In: IEEE. **2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)**. [S.l.], 2017. p. 1–6.

KIM, H. E. et al. Transfer learning for medical image classification: a literature review. **BMC medical imaging**, Springer, v. 22, n. 1, p. 1–13, 2022.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. **arXiv preprint arXiv:1609.02907**, 2016.

KONDA, R.; WU, H.; WANG, M. D. Graph convolutional neural networks to classify whole slide images. In: IEEE. **2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)**. [S.l.], 2020. p. 754–758.

KRAUSE, J. et al. 3d object representations for fine-grained categorization. In: **Proceedings of the IEEE international conference on computer vision workshops**. [S.l.: s.n.], 2013. p. 554–561.

KRIZHEVSKY, A.; HINTON, G. et al. Learning multiple layers of features from tiny images. Toronto, ON, Canada, 2009.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Advances in neural information processing systems**, v. 25, 2012.

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

LEE, J. W.; LEE, W. K.; SOHN, S. Y. Graph convolutional network-based credit default prediction utilizing three types of virtual distances among borrowers. **Expert Systems with Applications**, Elsevier, v. 168, p. 114411, 2021.

LI, Y. et al. Fully convolutional networks for panoptic segmentation. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2021. p. 214–223.

LI, Z.; HOIEM, D. Learning without forgetting. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 40, n. 12, p. 2935–2947, 2017.

LIANG, G.; ZHENG, L. A transfer learning method with deep residual network for pediatric pneumonia diagnosis. **Computer methods and programs in biomedicine**, Elsevier, v. 187, p. 104964, 2020.

LIN, T. et al. A survey of transformers. **AI Open**, Elsevier, 2022.

LU, D.; WENG, Q. A survey of image classification methods and techniques for improving classification performance. **International journal of Remote sensing**, Taylor & Francis, v. 28, n. 5, p. 823–870, 2007.

MICHELIN, C. R. L. et al. **Geologia Digital: Busca Integrada de Dados Geocientíficos Heterogêneos**. 2021.

MOHAN, A.; VENKATESAN, M. Hybridcnn based hyperspectral image classification using multiscale spatiospectral features. **Infrared Physics & Technology**, Elsevier, v. 108, p. 103326, 2020.

MORMONT, R.; GEURTS, P.; MARÉE, R. Comparison of deep transfer learning strategies for digital pathology. In: **Proceedings of the IEEE conference on computer vision and pattern recognition workshops**. [S.l.: s.n.], 2018. p. 2262–2271.

MUTCH, J.; LOWE, D. G. Object class recognition and localization using sparse features with limited receptive fields. **International Journal of Computer Vision**, Springer, v. 80, p. 45–57, 2008.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Icml**. [S.l.: s.n.], 2010.

PAIVA R, G. **Estratigrafia de sequências aplicada à Formação Poti, Viseano da Bacia do Parnaíba**. Dissertation (Master) — Instituto de Geociências, UnB, 2018.

PAN, B. et al. Spatio-temporal graph for video captioning with knowledge distillation. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 10870–10879.

PAN, S. J.; YANG, Q. A survey on transfer learning. **IEEE Transactions on knowledge and data engineering**, IEEE, v. 22, n. 10, p. 1345–1359, 2010.

PULS, E. d. S. **An evaluation of pre-trained models for feature extraction in image classification**. 64 p. Monografia (undergraduate thesis) — Universidade Federal do Rio Grande do Sul, 2023.

RADFORD, A. et al. Learning transferable visual models from natural language supervision. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2021. p. 8748–8763.

RUSSELL, S. J. **Artificial intelligence a modern approach**. [S.l.]: Pearson Education, Inc., 2010.

SAFARI, K.; PRASAD, S.; LABATE, D. A multiscale deep learning approach for high-resolution hyperspectral image classification. **IEEE Geoscience and Remote Sensing Letters**, IEEE, v. 18, n. 1, p. 167–171, 2020.

SCARSELLI, F. et al. The graph neural network model. **IEEE transactions on neural networks**, IEEE, v. 20, n. 1, p. 61–80, 2008.

SHI, L. et al. Skeleton-based action recognition with multi-stream adaptive graph convolutional networks. **IEEE Transactions on Image Processing**, IEEE, v. 29, p. 9532–9545, 2020.

SILVA, C. **Análise da deformação tectônica em afloramento da Formação Pindamonhangaba (Bacia de Taubaté, Rift Continental do Sudeste do Brasil)**. Bachelor's Thesis — Instituto de Geociências, UFRJ, 2015.

SLADOJEVIC, S. et al. Deep neural networks based recognition of plant diseases by leaf image classification. **Computational intelligence and neuroscience**, Hindawi, v. 2016, 2016.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. **Advances in neural information processing systems**, v. 27, 2014.

SZEGEDY, C. et al. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9.

SZELISKI, R. **Computer vision: algorithms and applications**. [S.l.]: Springer Nature, 2022.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: PMLR. **International conference on machine learning**. [S.l.], 2019. p. 6105–6114.

TAYAL, A. et al. Dl-cnn-based approach with image processing techniques for diagnosis of retinal diseases. **Multimedia Systems**, Springer, p. 1–22, 2021.

TODESCATO., M. et al. Multiscale context features for geological image classification. In: INSTICC. **Proceedings of the 25th International Conference on Enterprise Information Systems - Volume 1: ICEIS,**. [S.l.]: SciTePress, 2023. p. 407–418. ISBN 978-989-758-648-4. ISSN 2184-4992.

TORREY, L.; SHAVLIK, J. Transfer learning. In: **Handbook of research on machine learning applications and trends: algorithms, methods, and techniques**. [S.l.]: IGI global, 2010. p. 242–264.

VARSHNI, D. et al. Pneumonia detection using cnn based feature extraction. In: IEEE. **2019 IEEE international conference on electrical, computer and communication technologies (ICECCT)**. [S.l.], 2019. p. 1–7.

VASWANI, A. et al. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

WAN, S. et al. Multiscale dynamic graph convolutional network for hyperspectral image classification. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 58, n. 5, p. 3162–3177, 2019.

WANG, J. et al. End-to-end object detection with fully convolutional network. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2021. p. 15849–15858.

WANG, W. et al. Development of convolutional neural network and its application in image classification: a survey. **Optical Engineering**, Society of Photo-Optical Instrumentation Engineers, v. 58, n. 4, p. 040901–040901, 2019.

WONG, R. C.; LEUNG, C. H. Automatic semantic annotation of real-world web images. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 30, n. 11, p. 1933–1944, 2008.

WU, L. et al. Graph neural networks: foundation, frontiers and applications. In: **Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2022. p. 4840–4841.

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.

XIE, S. et al. Aggregated residual transformations for deep neural networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 1492–1500.

XU, X. et al. A deep learning system to screen novel coronavirus disease 2019 pneumonia. **Engineering**, Elsevier, v. 6, n. 10, p. 1122–1129, 2020.

ZHAI, X. et al. Scaling vision transformers. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2022. p. 12104–12113.

ZHANG, D.; ISLAM, M. M.; LU, G. A review on automatic image annotation techniques. **Pattern Recognition**, Elsevier, v. 45, n. 1, p. 346–362, 2012.

ZHANG, H.; ZOU, J.; ZHANG, L. Ems-gcn: An end-to-end mixhop superpixel-based graph convolutional network for hyperspectral image classification. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 60, p. 1–16, 2022.

ZHU, W. et al. Investigation of transfer learning for image classification and impact on training sample size. **Chemometrics and Intelligent Laboratory Systems**, Elsevier, v. 211, p. 104269, 2021.

ZHUANG, F. et al. A comprehensive survey on transfer learning. **Proceedings of the IEEE**, IEEE, v. 109, n. 1, p. 43–76, 2020.

ZHUANG, P. et al. Image tampering localization using a dense fully convolutional network. **IEEE Transactions on Information Forensics and Security**, IEEE, v. 16, p. 2986–2999, 2021.

APPENDIX A — RESUMO EXPANDIDO

This chapter presents a summary of this master thesis in the Portuguese language, as required by the PPGC Graduate Program in Computing.

Este capítulo apresenta um resumo desta dissertação de mestrado em língua portuguesa, conforme exigido pelo Programa de Pós-Graduação em Computação.

A.1 Introdução

Atualmente, redes sociais, sistemas corporativos e diversos aplicativos na web geram grandes quantidades de dados a cada segundo. Devido à disponibilidade, gerenciar e recuperar os dados relevantes para apoiar as tarefas de interesse tornam-se desafios. Esse cenário também está presente dentro das empresas, que gastam recursos consideráveis tentando solucionar esse problema. Nesse contexto, lidar com imagens é uma tarefa ainda mais desafiadora. O principal motivo é a ausência de significado explícito associado às imagens, dificultando a recuperação desse tipo de dado por meio de consultas de pesquisa. Uma abordagem comum para lidar com este cenário envolve a anotação de imagens com tags semânticas para permitir buscas e recuperá-las (HOLLINK et al., 2003). No entanto, a anotação manual é trabalhosa e inviável. Assim, abordagens automáticas para classificação de imagens podem ser de grande valia para rotular imagens automaticamente neste cenário (WONG; LEUNG, 2008; ZHANG; ISLAM; LU, 2012), possibilitando uma posterior recuperação destes dados por meio de consultas convencionais. As técnicas de aprendizado profundo são candidatas naturais para rotular automaticamente grandes bancos de dados de imagens.

Na última década, Redes Neurais Convolucionais (CNN) (SZEGEDY et al., 2015; TAN; LE, 2019; KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e, mais recentemente, Vision Transformers (ViTs) (DOSOVITSKIY et al., 2020) melhoraram significativamente o desempenho em tarefas de classificação de imagens. Os pesquisadores usaram essas técnicas em vários domínios distintos (SLADOJEVIC et al., 2016; DUNG et al., 2019; ABBAS; ABDELSAMEA; GABER, 2021; HONG et al., 2020). Apesar desses excelentes resultados, arquiteturas de redes neurais sofisticadas geralmente demandam dados de treinamento consideráveis para atingir bons desempenhos (ZHU et al., 2021). Aprendizado por transferência (TORREY; SHAVLIK, 2010) surgiu como uma abordagem promissora para lidar com este problema (LIANG; ZHENG, 2020; HORRY et al., 2020) uma vez que estas abordagens permitem aproveitar o conhecimento aprendido a partir de quantidades significativas de dados para lidar com tarefas em que apenas poucos dados estão disponíveis.

Em geral, quando o aprendizado por transferência é aplicado, as imagens usadas para alimentar as redes neurais pré-treinadas são padronizadas para corresponder aos requisitos de entrada das arquiteturas (LECUN et al., 1998; KRIZHEVSKY; HINTON et al., 2009). Em geral, esse processo de padronização descarta partes críticas da imagem ou altera sua proporção, causando perda de informações ou introdução de ruído. No entanto, conjuntos de dados de imagens com diferentes tamanhos e proporções são comuns em configurações do mundo real. Alguns trabalhos (ANTHIMOPOULOS et al., 2016; ARAÚJO et al., 2017) abordam essas questões adotando, por exemplo, abordagens baseadas em patches que lidam com imagens considerando diferentes patches extraídos delas. Outras abordagens também adotam *Fully Convolutional Network* (FCN) (WANG et al., 2021; ZHUANG et al., 2021) para evitar o descarte de informações relevantes da imagem.

Recentemente, alguns trabalhos (ZHANG; ZOU; ZHANG, 2022; BAE et al., 2022) têm explorado redes neurais de grafos (GNN) para classificação de imagens. Essas abordagens envolvem a representação de imagens como grafos, onde os nós representam pixels ou regiões irregulares de pixels (chamados superpixels) e as arestas representam relações espaciais entre pixels ou superpixels. Cada nó é caracterizado por características de cada pixel ou propriedades estatísticas do conjunto de pixels que constituem cada superpixel (ZHANG; ZOU; ZHANG, 2022). Como algumas GNNs podem lidar com grafos com números heterogêneos de nós e arestas, essas abordagens também podem lidar com imagens de tamanhos diferentes.

Outra característica essencial dos conjuntos de dados de imagens do mundo real são as características visuais em diferentes escalas de análise e algumas características visuais que são aparentes apenas em algumas escalas de análise. Recentemente, alguns trabalhos têm lidado com essas características propondo abordagens de aprendizado profundo que alavancam diferentes características visuais em múltiplas escalas de maneiras diferentes (CHEN; FAN; PANDA, 2021; MOHAN; VENKATESAN, 2020).

O conjunto de dados de imagens geológicas utilizado durante o desenvolvimento do sistema de recuperação de imagens possui características que dificultam o processo de classificação: é desbalanceado, possui imagens de tamanhos variados e não possui grande volume de dados. Nesse contexto, este trabalho se concentra em conjuntos de dados de imagens do mundo real que geralmente possuem pequenas amostras, onde as imagens podem ter tamanhos diferentes e cujas características visuais podem ser aparentes em diferentes escalas. Para lidar com esses conjuntos de dados, propomos uma abordagem baseada em redes convolucionais de grafos multiescala para classificação de imagens.

Nossa abordagem consiste em dois aspectos principais que apresentaremos a seguir: (i) uma estratégia para representar imagens como conjuntos de grafos de características baseados em patches que podem capturar os características visuais de cada imagem em diferentes escalas; e (ii) uma arquitetura de rede convolucional de grafos multi-entrada para classificação de imagens multiescala que recebe como entrada um conjunto de grafos de características baseados em patches representando uma dada imagem em diferentes escalas.

Primeiro, então, representamos uma dada imagem por um grafo constituído de nós, denotando características dos patches e arestas regulares da imagem original, representando relações espaciais entre patches na imagem original. Um aspecto essencial de nossa abordagem é caracterizar cada nó do grafo gerado com características extraídas de cada patch da imagem original por modelos pré-treinados. Ao adotar *extração de características* de modelos pré-treinados, nossa abordagem pode aplicar o aprendizado de transferência para alavancar o conhecimento capturado em modelos pré-treinados para gerar representações informativas da imagem original.

Na sequência, temos em nossa abordagem uma rede convolucional de grafos multi-entrada para classificação de imagens multiescala baseada na rede convolucional de grafos (GCN) proposta em (KIPF; WELLING, 2016). A GCN pode lidar com problemas de classificação considerando as características de cada nó e o relacionamento entre eles em dados com uma estrutura de grafo. Nossa arquitetura GCN proposta pode receber como entrada um conjunto de grafos multiescala (MGS) representando uma determinada imagem em diferentes escalas (com um grafo para cada escala considerada na tarefa). Assim, nossa arquitetura pode ser adaptada para lidar com diferentes escalas de forma que o número de entradas de nossa arquitetura seja definido em função do número de diferentes escalas é um conjunto de escalas SS que define um conjunto de grafos multiescala $MGS(SS)$. As escalas adequadas no conjunto de escalas podem variar de acordo com as características da tarefa.

O elemento-chave de nossa arquitetura é o *módulo de processamento de grafos* (GPM), que é composto de apenas uma camada de convolução de grafo (KIPF; WELLING, 2016) com tangente hiperbólica (tanh) como função de ativação, seguida por uma camada de pooling.

Então, nossa arquitetura concatena as saídas dos GPMs em um único vetor, que é processado e classificado por uma camada totalmente conectada com uma função linear.

Definimos nossa arquitetura como resultado da experimentação empírica. Testamos diferentes arquiteturas, com diferentes números de camadas convolucionais, diferentes tipos de pooling e diferentes funções de ativação.

Em uma visão geral, as principais características de nossa abordagem são:

- Evitar a inserção de distorções em características visuais relevantes, mantendo a proporção da imagem original durante todas as etapas do processamento.
- Evitando a perda de informações relevantes sobre a imagem original, pois dividimos a imagem original em patches regulares que cobrem a maior parte da área da imagem. Este método contrasta com a prática comum na maioria das abordagens baseadas em redes neurais para classificação de imagens, que envolve uma etapa de pré-processamento onde algumas partes da imagem original são descartadas.
- Capacidade de lidar com imagens com tamanhos diferentes em um determinado conjunto de dados, pois nossa abordagem representa essas imagens com grafos de tamanhos diferentes manipulados pela arquitetura convolucional de grafos proposta.
- Capturando características locais informativos de cada patch da imagem original usando modelos pré-treinados como extratores de características, aumentando as capacidades de nossa abordagem para lidar com pequenos conjuntos de dados.
- Representando a relação espacial das características locais de diferentes partes de uma determinada imagem em uma estrutura de grafo.
- Representação das características visuais de uma determinada imagem em diferentes escalas de análise através de diferentes grafos de características baseados em patches que podem ser usados coletivamente como entrada para treinar uma rede convolucional de grafos de múltiplas entradas para classificação de imagens.

Até onde sabemos, nossa abordagem proposta é a primeira a combinar todas essas vantagens.

A.2 Metodologia

Avaliamos nossa abordagem em dois conjuntos de dados: um conjunto de dados de *imagens geológicas* (MICHELIN et al., 2021) e o conjunto de dados stanford cars

(KRAUSE et al., 2013). O conjunto de dados de imagens geológicas é considerado neste trabalho porque este trabalho faz parte de um projeto cujo objetivo é desenvolver um *sistema de recuperação de imagens* para a indústria do petróleo. É importante notar que o conjunto de dados geológicos foi desenvolvido em cooperação com empresas e não pode ser compartilhado devido a questões de direitos autorais. O conjunto de dados Stanford Cars, que está disponível publicamente, foi selecionado para permitir a reprodutibilidade de nossos resultados e demonstrar que pode ser aplicado a outros domínios. Os aspectos desses conjuntos de dados tornam-os desafiadores a classificação das imagens.

Para demonstrar que nossa abordagem pode adotar diferentes modelos pré-treinados como extratores de características, aplicamos nossa abordagem usando três modelos pré-treinados distintos[1]: (i) resNext-101 (XIE et al., 2017), pré-treinado no ImageNet, (ii) DenseNet-121 (HUANG et al., 2017), pré-treinado no ImageNet; e (iii) CLIP Vit-B/32 (RADFORD et al., 2021), pré-treinado em um conjunto de dados com 400 milhões de imagens. Os dois primeiros apresentam bom desempenho com ajuste fino (BAKER; ZENGELER; HANDMANN, 2022) e como extrator de características (VARSHNI et al., 2019), respectivamente, enquanto o último é um modelo baseado em Transformers que apresentou resultados impressionantes em estudos recentes (ZHAI et al., 2022) . Neste trabalho, utilizamos apenas o módulo de extração de características dos modelos, para que a camada de classificação seja removida do modelo original.

Avaliamos a eficácia de nossa abordagem em representar a informação da imagem em diferentes escalas simples e por suas combinações. Nosso algoritmo foi aplicado para gerar grafos de características baseados em patches de cada imagem nos conjuntos de dados, aplicando os três modelos pré-treinados como extratores de características usando cinco escalas diferentes representadas pelo parâmetro N. Usamos $N = 1$, $N = 2$, $N = 3$, $N = 4$ e $N = 5$. Para usar diferentes números de grafos de características baseados em *patches* (um para cada escala), adaptamos nossa arquitetura de rede convolucional multi-entrada incluindo o número necessário de diferentes módulos de processamento de grafos para cada caso.

Em nossos experimentos, usamos modelos pré-treinados como extratores de características sem ajuste fino. Conforme descrito na seção anterior, nossa abordagem usa um módulo classificador que recebe um ou mais grafos como entrada e a saída é a classe prevista. As imagens no conjunto de teste também são transformadas em grafos de carac-

[1]Os modelos pré-treinados resNext-101 e DenseNet-121 foram obtidos de <https://pytorch.org/vision/stable/models.html>, e o modelo pré-treinado CLIP Vit-B/32 foi obtido de <https://github.com/openai/CLIP>

terísticas baseados em patches e, em seguida, avaliadas pelo classificador treinado.

Em nossos experimentos, adotamos validação cruzada com cinco *folds*. A cada iteração, cada *fold* é usado uma vez como dados de teste, enquanto que a partir da união dos quatro *folds* restantes. Consideramos 10% como dados de validação e o restante como dados de treinamento. Adotamos o *adam optimizer* com uma taxa de aprendizado de $0,001$ e um limite de 100 épocas. Aplicamos a parada antecipada, considerando cinco épocas sem uma melhora mínima de $0,001$ na perda do *conjunto de validação*. Esses hiperparâmetros foram definidos com base em experimentos empíricos usando a abordagem. Realizamos os experimentos em um desktop com CPU Intel i7-10700, 32 gigabytes de RAM e GPU NVIDIA RTX 3060. O código foi implementado em python, usando principalmente a biblioteca PyTorch[2].

Em nosso primeiro experimento, testamos diferentes valores para o *stride* S e realizamos um segundo experimento no conjunto de dados geológicos e no conjunto de dados stanford cars, comparando nossa abordagem com três outras famílias diferentes de abordagens.

A.3 Resultados e Conclusão

No primeiro experimento, onde avaliamos a variação de valores do *stride*, ambos datasets $MGS(\{2\})$, $MGS(\{3\})$, $MGS(\{4\})$, e $MGS(\{5\})$ tem os melhores resultados usando 25% como valor da passada e o pior usando 10%. Essa diferença de desempenho é notável: no conjunto de dados geológicos a diferença entre o melhor e o pior desempenho varia de $\approx$0,5% a $\approx$ 12% e nos stanford cars varia de $\approx$ 5% a $\approx$ 60%. $MGS(\{4\})$ e $MGS(\{5\})$ com valor de *stride* 10% não conseguiram convergir e tiveram um desempenho muito ruim no dataset stanford cars.

Como obtivemos o melhor desempenho geral adotando 25% de *stride*, com apenas a escala $N = 1$ desviando desse padrão, padronizamos e adotamos 25% de *stride* como padrão para os experimentos subsequentes.

No segundo experimento tivemos a nossa abordagem superando os resultados dos métodos comparados. Com o conjunto de grafos multiescala $MGS(\{1, 2, 3, 4, 5\})$ tivemos o melhor resultado para a maioria das métricas em todos os três modelos no conjunto de dados geológicos. A diferença entre o desempenho alcançado por essa configuração em relação ao desempenho obtido usando features simples é de cerca de $4,5\%$ de Macro

[2]<https://pytorch.org/>

F1 no ResNeXt e cerca de $2,5\%$ para o DenseNet. O CLIP, tem uma diferença mais sucinta, chegando em torno de 1% de melhoria em todas as métricas, porém, consegue a melhor Macro F1 em relação a todos os experimentos, alcançando $85,41\%$.

Os melhores resultados no conjunto de dados stanford cars de cada um dos modelos são $MGS(\{1,2,3,4,5\})$ para DenseNet e ResNeXt e $MGS(\{1,2,3\})$ para CLIP, sendo este último o melhor resultado geral neste conjunto de dados. Quando comparada com as abordagens que usam features simples, nossa abordagem usando esses conjuntos de grafos melhora, em todas as métricas, ≈19% em ResNeXt, ≈19% em DenseNet e ≈3% em CLIP. Seguindo o padrão do conjunto de dados anterior, $MGS(\{1,2,3\})$ com CLIP tem o melhor resultado, atingindo 86,79% de precisão e 86,72% Macro F1.

Nesses experimentos, houve um padrão geral de melhoria de desempenho obtido pela adição de escalas para representar as imagens. Fugindo desse padrão temos o CLIP, que não teve melhora de desempenho adicionando as escalas $N = 4$ e $N = 5$ no stanford cars, diferentemente do desempenho obtido no conjunto de dados geológicos e os outros modelos, que têm seus melhores resultados em $MGS(\{1,2,3,4,5\})$.

A abordagem do grafo multiescala permite que o classificador obtenha uma perda menor em menos épocas do que as características simples extraídas após um simples pré-processamento das imagens. Nossa abordagem funciona melhor com menos tempo de treinamento e o *overfitting* não acontece durante o processo, com a parada antecipada definindo o melhor ponto de parada.

Esses resultados confirmam que nossa abordagem é eficaz e melhora a classificação. Em ambos os conjuntos de dados, a representação de imagens com grafos de características baseados em patches de diferentes escalas resulta em uma representação mais informativa das informações da imagem que podem ser exploradas por nossa arquitetura de rede convolucional de grafos multiescala. É essencial notar que esses efeitos acontecem com todos os modelos pré-treinados usados como extratores de características. Isso representa que nossa abordagem é independente do modelo de extração de características.

Em trabalhos futuros, podemos investigar como e por que os resultados variam dependendo da escala utilizada e o quanto isso está relacionado com as características dos modelos e *datasets* utilizados. A busca por métodos que possam reduzir o tamanho dos grafos e assim reduzir o custo de memória utilizada pode ser muito interessante, pois assim será possível adicionar mais escalas ao conjunto de grafos. Uma possível solução seria usar uma análise da importância de cada patch para eliminar os nós e melhorar as informações representadas por cada grafo. Realizar experimentos em mais conjuntos de dados e usar outros modelos também pode trazer informações. Com isso, podemos entender como podemos melhorar nossa abordagem.