Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Semester Project

From Pixels to Nodes: A Segmentation-Driven Approach to Image Classification

Mateo Diaz-Bone, Philip Toma, Stefan Scholbe

mateodi@ethz.ch
tomap@ethz.ch
sscholbe@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Karolis Martinkus, Ard Kastrati
Prof. Dr. Roger Wattenhofer

October 23, 2023

Acknowledgements

We thank Karolis and Ard for sharing their expertise with us and taking the time and effort to supervise this semester project. Without their professional knowledge and guidance this project wouldn't have been possible. Furthermore we thank Prof. Wattenhofer and the Distributed Computing Group for supplying us with a state-of-the-art computing infrastructure to run our experiments. We couldn't have asked for more!

Abstract

This project presents a non-classical approach to image classification by combining Graph Neural Networks (GNNs) with Convolutional Neural Networks (CNNs) and image segmentation techniques. We built a pre-processing and feature extraction component preceding the Vision GNN Backbone (ViG) and enhanced the model with a UNet-based CNN. The final outcome of our experimentation on the ImageNet dataset failed to demonstrate a performance enhancement of Vision GNN.

Our project was driven by two core objectives. We aimed to develop a compact model, maintaining similar performance metrics to the state-of-the-art. Secondly, we sought to illustrate the feasibility of using graph-based methods in the field of computer vision. Our findings demonstrate the applicability, but also possible caveats of applying graph-based machine learning techniques to the task of image classification.

Our implementation has been made available on GitHub[1].

[1] https://github.com/travelingtomat0/ViG-SP

Contents

Chapter 1

Introduction

Image classification is one of the grand challenges in the computer vision community, and remains a hot topic to this day. While many domain-specific image classification tasks, such as the well-explored handwritten digit classification task (MNIST), have already yielded successful solutions, the ongoing work in image classification promise to enhance and enable a wide range of products across different sectors, such as the automotive and healthcare sector.

State-of-the-art models for image classification rely on a large set of different components and modules, assembled into models consisting of billions of parameters. While such models scale to large computing nodes, e.g. services running in the cloud, using these architectures to run locally on a consumer machines is impossible. This facilitates the need for efficient classification models with regards to time and space complexity.

In this project we explored the possibility to build a neural network, which allows the use of irregular structures (such as image segmentations), to classify natural images. This was made possible by combining the power of convolutional neural networks and graph neural networks.

Convolutional neural networks have been used extensively in computer vision for image classification tasks. They excel at capturing local features and hierarchies of patterns within images, making them a fundamental choice for such tasks. However, these networks are typically designed for regular grid-like data, such as images with fixed dimensions. When dealing with irregular structures like image segmentations, the application of traditional convolutional networks becomes challenging.

Graph neural networks (GNNs), on the other hand, are well-suited for handling irregular structures represented as graphs. They have demonstrated significant success in various domains, including social network analysis and recommendation systems [1]. Leveraging GNNs in image classification tasks opens up the possibility of efficiently processing non-grid data, allowing us to work with images in a more flexible manner.

Graph representations of images have become instrumental in various facets of image processing. A notable application is *image segmentation*, where pixels or regions in an image are mapped to nodes in a graph, such that finding the segmentation can be seen as a network flow problem [2]. Object recognition can also benefit from graph-based approaches, by modeling objects as nodes and their spatial relationships as edges in a graph [3].

In this report we show that the fusion of CNNs, GNNs, and image segmentation to create a novel image classification model have the potential to improve classification accuracy as compared to previous graph-based image classification networks. Through our architecture we aim to bridge the gap between traditional CNNs' grid-like data requirements and the complexity of irregular structures in image data.

Chapter 2

Background

2.1 SLIC Image Segmentation

A key building block of our approach is the simple linear iterative clustering algorithm (SLIC) [4]. It produces so-called superpixels of an image, which are perceptual groupings of pixels into segments of the image. Superpixels carry more information than single pixels and align with natural edges within the image rather than rectangular image patches.

The degree to which natural edges within an image are respected by the segmentation algorithm is referred to as boundary adherence. It is a measure to quantify how well the segmentation algorithm performs. Because SLIC offers a sensible trade-off between boundary adherence and computational effort we decided to use the algorithm to segment images into superpixels during the preprocessing stage of model training.

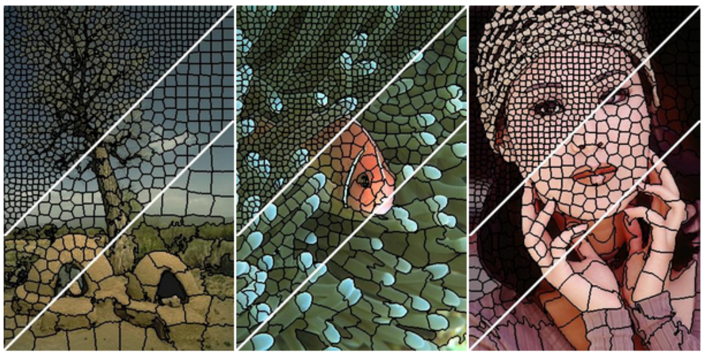

Figure 2.1: Example of SLIC segmentations from the publication [4] with different numbers of superpixels.

SLIC is based on the well-known k-means algorithm and, unlike other max-flow min-cut based methods, uses gradient ascent to iteratively refine an initial 'pseudo-random' clustering of pixels. Here, k-means was adapted, such that each pixel is associated with the most similar center pixel μ_k within a limited local search region. This is different to k-means, which would search the whole image for the most similar center pixel. After reassigning pixels to the most similar centers, they are updated to be the mean of all pixels belonging to the cluster. This process is repeated until the convergence criterion below is met.

$$||\mu_k^{new} - \mu_k^{old}||_2 \leq \text{Threshold}$$

By default, the only parameter of the algorithm is the number of desired, approximately same-sized, superpixels k. This parameter is not a hard limit: the actual number of segments returned by the algorithm is not enforced and can be higher or lower than k.

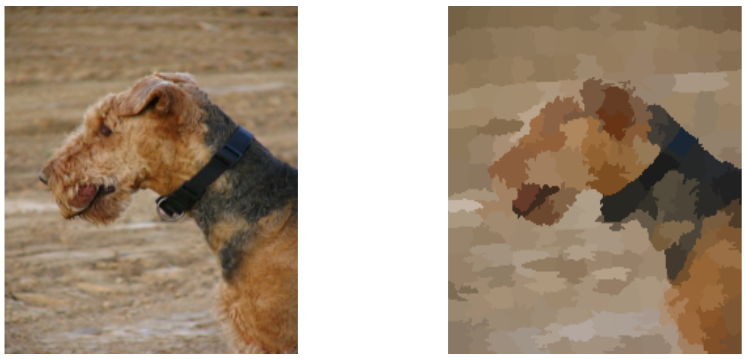

Figure 2.2: Visualization of a SLIC segmentation by coloring each superpixel with the mean RGB-value of all corresponding pixels. The image shows an input image on the left and the ouput of a SLIC segmentation with roughly 196 segments on the right.

As is visible in Figure 2.2, segmenting images with SLIC offers a more meaningful segmentation of the image than a simple quadratic segmentation of the image. Features of the depicted dog, such as its mouth, ears and collar are captured and their shape is preserved in the segmentation. This motivated us to utilize SLIC in our model design.

For our model, it was necessary to enforce an upper bound for k due to technical limitations. In case SLIC returned more than k superpixels, the most similar and physically neighboring superpixels were merged into a single superpixel. This case is extremely rare and its impact on total classification accuracy can be expected to be negligible.

2.2 ImageNet

In the context of image classification, the ImageNet [5] dataset holds an influential role as a foundational benchmark for evaluating the capabilities of large-scale image classification models. The dataset comprises an extensive collection of more than 10^6 images, spanning 1000 categories.

Through the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), researchers have been pushed to develop increasingly sophisticated models, often resulting in paradigm shifts within the field. Examples include the emergence of deep learning with AlexNet during the 2012 competition, as well as the well known models of VGG, ResNet, Inception, and EfficientNet. Each of the models, benchmarked on the ImageNet dataset, refined the understanding of image features and pushed the boundaries of prediction accuracy.

Although there are potential drawbacks to the dataset, such as label noise and dataset bias, we employed ImageNet to train and evaluate our image classification model. The reasoning behind this decision was simple: there is no better option to contextualize and compare our models performance against the state-of-the-art.

We note that we did not simply pursue achieving the highest possible accuracy on the dataset. We also aimed to improve efficiency through a small number of trainable parameters.

2.3 Vision GNN

The idea of Vision GNN (ViG) [6] was pivotal for our decision to pursue a graph-based approach to image classification. Published at NeurIPS 2022, to our knowledge, this was the first paper to show that graph-based methods can, in fact, compete with state-of-the-art convolutional deep learning models as well as vision transformers. The ViG model consists of the following stages.

2.3.1 Graph Representation Inference

Vision GNN transforms images to their corresponding graph representation by transforming the image into same-sized square-shaped patches. These patches make up the set of nodes N. The adjacency relation A is defined as the set of k-nearest nodes, meaning that the graph structure can be understood as the kNN graph of all patches in the image.

2.3.2 ViG Backbone

After the graph representation inference, the authors of Vision GNN concatenate three stages into the so-called ViG Backbone. It comprises of the information sharing, feature transformation and multi-head update stages which are repeated 12 times. Between each of the backbone iterations, the kNN graph is updated to reflect the new similarities between nodes.

Information Sharing

After inferring the graph representation (N, A) of an input image, the *'grapher module'* sequentially employs graph convolutions to aggregate and update node information, to encourage information flow throughout the model.

Feature Transformation

In a last step, node features obtained from the information sharing stage are transformed using a simple 2-layer MLP. This step further encourages the learning of sensible node features post information sharing.

Multi-Head Update

The graph representation of an input image is fed through the information sharing and feature transformation stage, which together build the ViG network. The output of the network is aggregated features x_i^n (per node i). They are split into multiple heads, which are updated using a multi-head update and concatenated to output the final features x_i'.

2.3.3 Prediction

The output of the backbone is provided to a two-layer CNN with batch normalization, which outputs a probability vector for all labels.

Chapter 3

Method

3.1 Graph Representation of Images

An image can be interpreted as a graph (N, A), where nodes represent pixels and edges are defined through the adjacency relationship between pixels. Depending on the application, the graph (N, A) can be adapted to capture closer or more distant relationships between pixels.

For example, using a simple grid-graph representation captures the relationship only between pixels that are 'physically touching'. This equates to a 4-connected graph representation of the image. Instead of using a simple grid-graph, it is also possible to use a more complex structure, which captures relationships between more distant pixels.

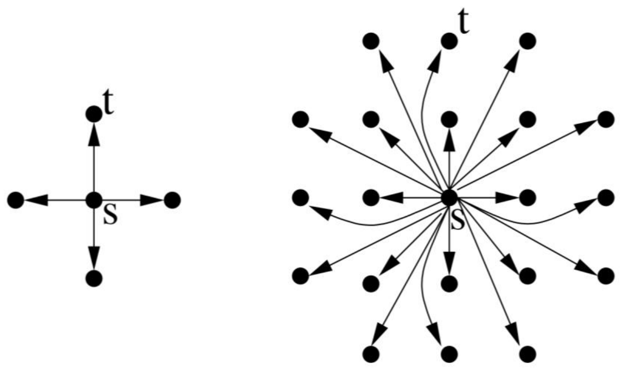

Figure 3.1: Example of different neighborhood representations, where s denotes the center pixel of the neighborhood.

In our use case, the computational complexity of the graph structure was a limiting factor. Images of size (224×224) are represented by a graph that contains $|N| = 50176$ nodes, each of which contains 3 features. Using a grid graph representation results in more than 200,000 edges. Because PyTorch Geometric represents the adjacency relationship A as a list of edges, implementing a graph structure such as that shown on the right in Figure 3.1 would mean a 4-fold increase in the size of the graph.

Due to the constraining factors of both time and space complexity, we decided to transform input images into the corresponding 4-connected grid-graph representations. Once the dimensionality of the image has been reduced, the image is represented by a kNN-graph structure.

3.2 Our Model Architecture

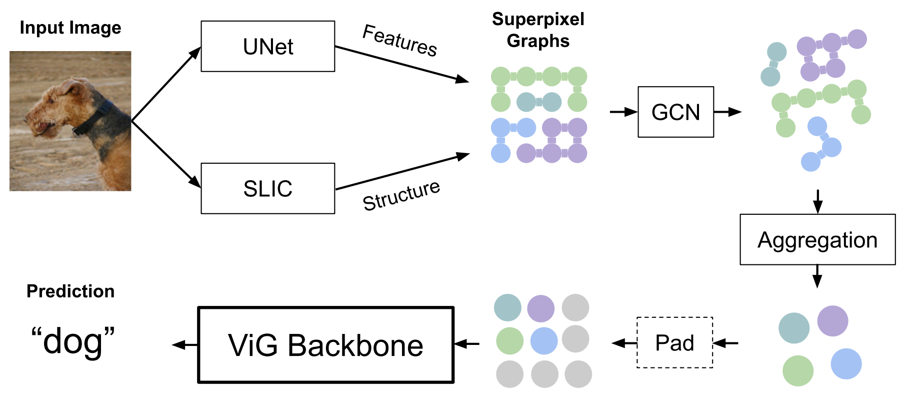

Figure 3.2: High-level overview of the model pipeline.

To extend the Vision GNN architecture, we built a pipeline that generates the SLIC segmentation of an input image, where an input image could be the original image as well as any of the image augmentations that we apply during preprocessing, as listed in Table A.1. This results in a segmentation mask. From it, we construct a set of 4-connected grid graphs, with each graph representing one superpixel.

To derive features for every node, we train a UNet CNN [7] with 4 encoder and decoder layers on the input image. Each stage of the network consists of two 2D convolutions with batch normalization, and we use a ReLU activation function at the very end of the network.

Using the UNet we transform a 3-channel image into a 24-channel image with the same height and width. Thus, each node in the superpixel graphs is initialized with 24 features. We chose the UNet architecture as it allowed us to incorporate strided convolutions (as used in the original feature extraction model from VisionGNN) while retaining the original size of the input image.

Subsequently, the feature enriched grid graphs for each superpixel are fed individually through a graph convolutional network (GCN) composed of 5 layers, each followed by batch normalization. This step increases the feature count from 24 to 192 for every node in the graph and allows the model to share information between nodes within a superpixel.

Each output graph of the GCN is then collapsed into a single node, to more accurately represent the actual superpixels. The features of this node are derived by aggregating features of all nodes within a superpixel.

We experimented with various common aggregation methods, such as LSTM, DeepSets and MLP aggregation, as described in [8, 9]. However, as we show in Table A.2, we found that the mean-aggregator consistently provided superior results, leading to its adoption in our model.

In a final preprocessing step we take the superpixel feature vectors and zero-pad them for a smooth transition into the VisionGNN Backbone. This is necessary because the SLIC algorithm returns a variable amount of segments, different to the feature extraction method used in the original VisionGNN architecture.

The output of the zero-padding stage is then fed to the Vision GNN backbone and prediction stage as described in Section 2.3.

3.3 Model Training

During training, we employed several deterministic and non-deterministic data augmentation techniques, which are listed in Table A.1. We excluded augmentations such as MixUp and CutMix since they require the mixing of several samples from the same batch. We deemed the use of the two as infeasible, because the implementation using segmentation masks would have lead to performance issues during model training.

We used the AdamW optimizer [10] with a learning rate of $4 \cdot 10^{-8}$ and a cosine learning rate schedule. Due to GPU memory constraints, we were limited to a batch size of 176 (minibatches of size 22 distributed across 8 Nvidia RTX3090 GPUs). This resulted in a latency of roughly 10,000 seconds per epoch on the full ImageNet dataset and 1,000 seconds on ImageNet-100. In addition to the common training of the model using gradients, we calculated a model copy using an exponential moving average (EMA)[1] on the trained parameters. This can improve robustness and generalization properties of a classifier by following the target model with momentum [11].

[1]The EMA methodology is explained here: `https://timm.fast.ai/training_modelEMA`.

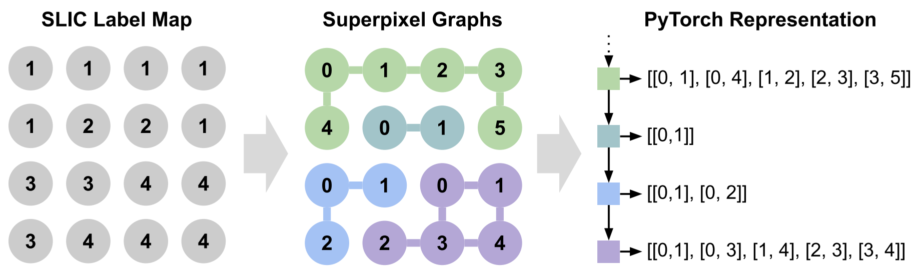

Figure 3.3: An example output of SLIC, the associated superpixel graphs, and the required (list of tensors) graph structure. While vertex IDs need to start at 0, their ordering is irrelevant. The related features have a comparable structure, which is not depicted here.

3.4 Performance Optimizations

In the following, we explore the process of efficiently creating 4-connected superpixel graphs based on a SLIC segmentation. This was a crucial and challenging component of our model pipeline. The graph generation occurs during hundreds of epochs across over a million images. Consequently, even a couple of additional milliseconds latency per image can tremendously increase total training time. Additionally, PyTorch Geometric constrains the desired structure and format of the graphs.

An illustration of the input and the specific graph structure can be seen in Figure 3.3. Due to our model design, each superpixel is translated into a distinct graph. Because of a 4-connection system of the nodes, these graphs exhibit a simple structure that could be generated by using masking. A notable difficulty was that the nodes in each superpixel must start at 0 since the features of the nodes must be provided separately, making edge generation a less trivial task.

3.4.1 Attempt 1: Precomputing Graphs

Initially, we suggested precomputing the graphs offline for all images in the dataset and storing them in a compressed format on disk. The focus was on fast decompression to load them during training. However, we quickly realized this approach was not feasible for our needs. The augmentation phase, which performs, for example, image scaling and random cropping, is essential for the model's generalization performance. These augmentations result in varying SLIC segmentations and, therefore, different superpixel graphs.

We then considered precomputing a smaller set of augmentations and their corresponding superpixel graphs. But due to the immense size of the dataset, even this compressed set of graphs took up a significant amount of storage space.

Approach	**Runtime**
Naive Python	249.0 ms
NumPy Vectorization	34.8 ms
C++ Implementation	1.9 ms

Table 3.1: Mean runtime over 100 runs of the three graph generation methods on a SLIC segmentation: a naive Python approach using a double for-loop, a vectorized NumPy strategy, and our C++ implementation.

Furthermore, limiting the number of augmentations decreased the generalization performance. Seeing the same few images repetitively over hundreds of epochs prevented the model from reaching its target accuracy.

Thus, we adopted a more demanding on-the-fly graph generation.

3.4.2 Attempt 2: On-the-fly Graph Generation

As our initial approach, we created the graph using a naive Python double for-loop to traverse the 224x224 label map. Although this method was easy to implement, the overhead of using pure Python resulted in poor performance. In an effort to improve this, we crafted a vectorized NumPy solution that employs masking and specialized index raveling functions. Even though this increased performance by 7.2x, it still fell short of our needs.

While individual operations in NumPy are typically fast and highly optimized, combining them might not achieve the same efficiency. This can be due to factors such as creating temporary copies of data, missing hardware-level optimizations like vectorized instructions, or following a suboptimal execution path. The overall sequence of operations may not be as efficient as when it is designed as a single, specific function, leading to potential performance inefficiencies.

Therefore, we decided to bypass these constraints by implementing the graph generation method directly in C++, which can be accessed in Python through a C binding. The performance of this implementation is superior to both NumPy (18.3x) and naive (131.1x) approaches. We paid close attention to memory allocation, selected appropriate data structures, and tried to reuse memory wherever feasible. The benchmark results of the three approaches are provided in Table 3.1.

CHAPTER 4

Evaluation

4.1 Benchmark Metrics

The ImageNet Benchmark commonly employs two metrics to evaluate model performance: the top-1 and top-5 accuracy. The top-1 accuracy evaluates which fraction of all predictions is correct. Given that models output a probability vector $y \in [0, 1]^n$ where n is the number of labels, the top-5 accuracy measures how often the ground truth is part of the 5 labels with the highest probability in the output vector y.

In the context of ImageNet it is important to not only evaluate the top-1 accuracy. The top-5 accuracy captures the model's capability to generalize and recognize relevant classes, even when its primary prediction is not the most confident one. It especially accounts for scenarios where multiple plausible predictions might be applicable, as is the case for some specific noisy labels in the ImageNet dataset.

4.2 Ablation Study

We performed an ablation study of the model design on the ImageNet-100 dataset, subset of ImageNet containing images consistent with 100 instead of the whole $1,000$ labels. We compared three of model designs with the results of the original Vision GNN models: one model without SLIC segmentation (ViG-SP-Grid), one with 100 SLIC segments (ViG-SP100) and one with 196 SLIC segments (ViG-SP196). Due to our model architecture, increasing the number of segments above 196 was not possible.

The original Vision GNN models (ViG-Ti & ViG-B) were trained specifically on the ImageNet-100 dataset with the original published hyperparameters for 600 epochs.

Evaluating our model against the accuracy of the original models on the ImageNet-100 dataset demonstrated that it significantly improved classification

	ViG-Ti	**ViG-B**	**ViG-SP196**	**ViG-SP100**	**ViG-SP-Grid**
Top-5 (%)	96.33	97.10	***97.78***	97.28	97.44
Top-1 (%)	83.06	86.46	***87.37***	86.22	86.12

Table 4.1: Ablation Study on the ImageNet-100 Dataset. Note that the model size of ViG-B is roughly 10x that of all others.

accuracy on ImageNet-100. The model consistently outperformed the original models in both metrics. The fact that even the grid-model, which uses a 14×14 grid graph structure instead of the irregular SLIC segmentation, outperforms the original models shows that the additional feature extraction offered by the combination of graph and convolutional neural network helps increase model performance.

When comparing grid segmentation to SLIC segmentation, the difference between models is rather small. A possible explanation could be that segmenting an image (always of shape 224×224) into patches of size 14×14 results in a more fine-grained segmentation into 196 superpixels, unlike the SLIC100 segmentation. However, a comparison between the grid model and the similar SLIC196 model, which segments into approximately 196 superpixels, reveals that the SLIC segmentation offers a slight improvement in classification accuracy.

Due to its superior performance we adopted the ViG-SP196 model design for our benchmark on the full ImageNet dataset.

4.3 Baseline

This project extends the ViG-Ti architecture, which is the smallest of the Vision GNN models. ViG-Ti is an isotropic network, meaning it has uniform receptive fields across width and height. It differs from pyramid architectures, also presented in the Vision GNN paper, which aim to capture scale-invariant properties of the image and produce multi-scale features by gradually shrinking the spatial size of feature maps as the network (backbone) deepens.

	ViG-Ti
Top-5 (%)	92.0
Top-1 (%)	73.9

Table 4.2: Baseline metrics for the evaluation.

While popular networks, such as ResNet [12] and Swin Transformer [13], often rely on the positive effects of pyramid architectures, we chose to adopt an isotropic approach to leverage increased computational efficiency. Therefore, ViG-Ti serves as the baseline for our evaluation.

4.4 State-of-the-Art

The state-of-the-art on the ImageNet dataset (as of August 2023) is the BASIC-L model [14]. It is a combination of feature extraction through convolutional deep neural networks and strategically placed attention mechanisms using transformers. With a model size of more than 2.4 billion parameters it achieves a top-1 accuracy of 91.1%. The top-5 accuracy was not reported.

When looking only at models in a comparable range (between 5 and 15 million parameters), the state-of-the-art lies roughly around 83%:

- NoisyStudent (EfficientNet-B3) [15]
  with a top-1 accuracy of 84.1% (top-5 accuracy 96.9%)
- FixEfficientNet-B2 [16]
  with a top-1 accuracy of 83.6% (top-5 accuracy 96.9%)
- TinyViT-11M-distill [17]
  with a top-1 accuracy of 83.2% (top-5 accuracy 96.5%)

Vision GNN achieved a top-1 accuracy of 83.7% with a model size of 83 million parameters (Pyramid ViG-B). In our comparison group based on model size, the smaller Pyramid-ViG-Ti model with 10.7 million parameters achieved a top-1 accuracy of 78.2%.

4.5 Results

We trained and evaluated our final model on the ImageNet-1k dataset, consisting of over 1.28M training images, 50K validation images and 100K test images. Images in the dataset are mapped to one of 1K classes. To save time at the beginning of training, we utilized the published parameters of Vision GNN (ViG-Ti) which are shared with our model's backbone. Nonetheless, completing the training required roughly 3 weeks.

On the validation set, our model achieved a top-1 accuracy of 68.26% and a top-5 accuracy of 88.38%. On the test set, we achieved a top-1 accuracy of 67.37% and a top-5 accuracy of 87.67%.

Comparing this result with the baseline metrics of ViG-Ti, we were roughly 5% less accurate in both metrics. The discrepancy between our results on ImageNet-100 and ImageNet-1k might be attributed to our inability to increase the batch size or the limited number of epochs available for training the final model.

	Validation-Set	**Test-Set**
Top-5 (%)	88.38	87.67
Top-1 (%)	68.26	67.37

Table 4.3: ViG-SP196 evaluation metrics.

Chapter 5

Discussion

Prior to starting with this project, the idea of applying graph-based methods for image processing seemed far-fetched. Convolutional neural networks, the de-facto standard to tackle computer vision tasks, are easy to train and efficiently solve many of the tasks in the vision domain.

By stumbling upon the Vision GNN architecture, we were motivated to look into the applicability of graph neural networks in the vision domain. The authors published promising results on the ImageNet dataset, laying the groundwork for a possible application of graph neural networks in image classification.

The advantages of using a graph representation for image processing are well known: graphs are generalized data structures, which are more flexible than the 'simple' grid representation of images. They allow the modeling of irregular (non-quadratic) structures, such as a superpixel representation of an image. Furthermore, similar components of an image can simply be connected with each other through edges, even if they are not physically neighboring in the input image.

Through our experiments, combining the ability of convolutional neural networks to extract feature maps with the ability to share information between nodes using graph neural networks, we showed that the performance of GNNs tasked with image classification can further be increased.

Despite experimenting with various different aggregators for the feature extraction module during model development, we were not able to improve the accuracy compared to a simple mean aggregation. Although it seemed promising at first, implementing a deep sets aggregation as suggested in [9] did not achieve an accuracy above 79% on the ImageNet-100 dataset.

We did not experiment with the graph representation of the image. Increasing the max-degree of nodes is thinkable, however this would greatly increase the space complexity of the representation. Furthermore, building a hierarchical graph to capture features at different scales would be a possible approach to increase prediction accuracy. We believe that the highest potential lies in exchanging the UNet architecture by another, potentially more expressive model.

Unlike convolutional neural networks and transformer based image classifiers, the time to train graph neural networks is a concern for tasks in computer vision. Despite our best efforts to optimize the time complexity of training, the latency remained high. This is likely due to bottlenecks in the used library as well as inefficiencies within our implementation [18].

A further concern when using graph-based methods for image processing is space complexity. A 224×224 image typically requires roughly 49 KB in compressed format and 157 KB when stored as a decompressed numpy array. When storing the graph representation, where each node has at most 4 neighbors, the storage required increases to at least 1.2 MB (in an optimised setting). The set of images that can be processed simultaneously on GPU is therefore constrained by how many graphs, in addition to the model, fit into VRAM.

Although our model showed promising results on ImageNet-100, a commonly used subset of ImageNet, we were not able to translate the results to the more general ImageNet-1k. Given the high computational effort to find better hyperparameters in order to have optimal performance, we believe that our approach is not the way to go for image classification.

Chapter 6

Conclusion

In this project, we reviewed the applicability of graph representations for image classification. We found that they introduce the opportunity to learn on irregular structures, such as segmented images, instead of being limited to regular, quadratic structures. By combining convolutional with graph neural networks, we were able to achieve results comparable to published results on the ImageNet dataset.

Due to the high latency of training on the graph representations, we believe that the downsides outweigh the benefits of using graph representations. While the ability to learn on irregular structures can be advantageous for model performance, we noticed that the impact is not significant enough to present a future direction for research in the domain of image classification.

Bibliography

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining.* ACM, jul 2018. [Online]. Available: https://doi.org/10.1145%2F3219819.3219890

[2] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, September 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000022288.19776.77

[3] H. Wei, C. Yang, and Q. Yu, "Efficient graph-based search for object detection," *Information Sciences*, vol. 385-386, pp. 395–414, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025516322630

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

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 2009, pp. 248–255.

[6] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," 2022.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[9] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," 2022.

[10] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[11] D. Busbridge, J. Ramapuram, P. Ablin, T. Likhomanenko, E. G. Dhekane, X. Suau, and R. Webb, "How to scale your ema," 2023.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” 2021.

[14] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le, “Symbolic discovery of optimization algorithms,” 2023.

[15] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, “Self-training with noisy student improves imagenet classification,” 2020.

[16] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, “Fixing the train-test resolution discrepancy: Fixefficientnet,” 2020.

[17] K. Wu, J. Zhang, H. Peng, M. Liu, B. Xiao, J. Fu, and L. Yuan, “Tinyvit: Fast pretraining distillation for small vision transformers,” 2022.

[18] Z. Wang, Y. Wang, C. Yuan, R. Gu, and Y. Huang, “Empirical analysis of performance bottlenecks in graph neural network training and inference with gpus,” *Neurocomputing*, vol. 446, pp. 165–191, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221003659

Appendix A

Appendix

Spatial Transformations
RandomResizedCropAndInterpolation
RandomHorizontalFlip
Rotate
ShearX
ShearY
TranslateXRel
TranslateYRel
Color Transformations
AutoContrast
Equalize
Invert
PosterizeIncreasing
SolarizeIncreasing
SolarizeAdd
ColorIncreasing
ContrastIncreasing
BrightnessIncreasing
SharpnessIncreasing

Table A.1: An overview of the applied transformations during the training process.

Approach	**Top-1 Accuracy (%)**	**Top-5 Accuracy (%)**
Mean Aggregation	87.37	97.78
DeepSets Aggregation	78.34	94.60
MLP Aggregation	69.82	90.98
LSTM Aggregation	66.23	89.29

Table A.2: Accuracy of our model depending on the aggregation operator. The ablations were performed on ImageNet-100.