Geometric Superpixel Representations for Efficient Image Classification with Graph Neural Networks

Radu A. Cosma*, Lukas Knobel*, Putri van der Linden, David M. Knigge, Erik J. Bekkers
University of Amsterdam
{radu.cosma,lukas.knobel}@student.uva.nl

Abstract

While Convolutional Neural Networks and Vision Transformers are the go-to solutions for image classification, their model sizes make them expensive to train and deploy. Alternatively, input complexity can be reduced following the intuition that adjacent similar pixels contain redundant information. This prior can be exploited by clustering such pixels into superpixels and connecting adjacent superpixels with edges, resulting in a sparse graph representation on which Graph Neural Networks (GNNs) can operate efficiently. Although previous work clearly highlights the computational efficiency of this approach, this prior can be overly restrictive and, as a result, performance is lacking compared to contemporary dense vision methods. In this work, we propose to extend this prior by incorporating shape information into the individual superpixel representations. This is achieved through a separate, patch-level GNN. Together with enriching the previously explored appearance and pose information of superpixels and further architectural changes, our best model, ShapeGNN, surpasses the previous state-of-the-art in superpixel-based image classification on CIFAR-10 by a significant margin. We also present an optimised pipeline for efficient image-to-graph transformation and show the viability of training end-to-end on high-resolution images on ImageNet-1k.[1]

1. Introduction

The dominant trend in image classification is to use deep Convolutional Neural Networks (CNNs) [28] or Vision Transformers (ViTs) [30]. These approaches, while achieving remarkable performance, also come with a prohibitive computational cost. This is largely attributable to them operating on the dense input pixel grid directly, which makes them scale unfavourably with image resolution. Moreover, CNNs progressively build representations of larger-scale features, implying the need for larger depths to incorporate long-range dependencies. While ViTs do not have this problem to the same extent due to the self-attention's global interaction, this mechanism is still computationally very expensive, driving the need for multiple GPUs during training even for efficient architectures [22, 21]. Furthermore, due to this reliance on pixel-level processing, these approaches are not robust to changes in scale and resolution [16]. Invariant qualities, such as invariance of object identity to rotations of the image, are difficult to encode into the networks for the same reason.

To overcome these limitations, we may choose to operate on a structured sparse representation of the image, the Region Adjacency Graph (RAG), using Graph Neural Networks (GNNs) [27]. This lower-dimensional image representation leverages the intuition that pixel-level information is mostly redundant in order to reduce computational complexity and allows for the straightforward construction of scale- and rotation-invariant networks in a parameter-efficient manner. The RAG is a graph representing an image, where nodes correspond to image regions of similar colour, superpixels, and edges encode their adjacency. Since the model now operates on features aggregated over sets of pixels, the RAG drastically decreases the input dimensionality compared to the raw image. Consequently, the simplicity of this representation enables a reduction in model complexity. Furthermore, the model is not constrained to operating on fixed-resolution images. Lastly, there is a great degree of flexibility with respect to which information to use and what invariants to represent in the obtained model. For example, given a rotation equivariant segmentation method yielding superpixels, a rotation invariant model can be obtained simply by discarding any superpixel positional information, or by using specific graph convolutions [26]. This flexibility presents us with a choice regarding the information about a superpixel we wish to include in its graph representation, *i.e.* which features of a superpixel are relevant to the image classification. Features of a superpixel include *pose* (in two-dimensional images made up of its position, rotation, and size), *appearance* (its texture and

*Joint first authors
[1]Code: github.com/lukasknobel/ShapeGNN

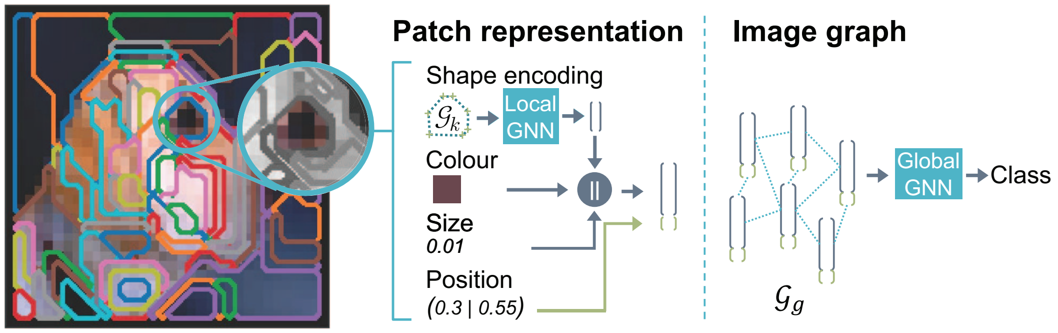

Figure 1. Image graph construction for a given image: the segmentation of the input is computed using the Felzenszwalb algorithm (left). For each patch, its representation is obtained by concatenating different features of its shape, appearance, and pose. Information about the position of each patch is processed separately (center, green arrow). Finally, these feature vectors are used to construct the image graph of the sample by connecting neighbouring patches with edges (right). This serves as the input to the model for classification.

colour information), and *shape* (its contour).

Previous work in this area has only focused on pose and appearance [16, 17]. To obtain the superpixels, Simple Iterative Linear Clustering (SLIC) [1] is commonly used, which generally produces similarly-shaped patches based on their local image regions. Thus, their homogeneous shapes offer little discriminative information (see Figure 2). Given the prior knowledge that shape information is important for human perception [5], segmentation methods resulting in patches with shapes more representative of the image's content may improve performance. For this reason, we propose using Felzenszwalb and Huttenlocher's method [8] (referred to as Felzenszwalb in this paper) which results in more globally informed superpixels with heterogeneous shapes. We show that these shapes effectively aid model performance. Furthermore, due to Felzenszwalb's method computing superpixels from the entire image instead of local regions, the adjacency information between superpixels, represented using edges in the graph, is intuitively more informative. Similarly coloured larger regions usually relate to the same semantic object in an image, *e.g*. the sky. Local algorithms such as SLIC, however, tend to decompose such a single semantic entity into smaller semantically equivalent superpixels, requiring additional depth in the GNN to aggregate the information of the whole object. By contrast, the globally informed Felzenszwalb patches allow for bigger regions being represented in a single node. Consequently, edges between nodes might be more representative of adjacent distinct components rather than just visually similar areas of the same object.

With the incentive to incorporate superpixel shapes provided by the Felzenszwalb segmentation algorithm, a good way to include these shapes into the image graph needs to be determined. We propose an end-to-end trainable approach, where each superpixel shape is represented using its contour. This contour is approximated with a polygon, which is interpreted as a graph and passed to a GNN (referred to as *local GNN*) which encodes it into a shape feature. The feature is then combined with information about appearance and pose to construct the feature vector of a superpixel used as input for the *global GNN* (see Figure 1). Hence, the *local GNN* should learn shape representations that are useful for image classification.

In summary, our contributions are as follows:

- We provide a framework for efficiently transforming images into graphs that supports the utilisation of different aspects of the image superpixels, with the novel inclusion of shape information.
- Based on this framework, we propose *ShapeGNN*, a model that outperforms previous GNN and superpixel-based image classification approaches on the CIFAR-10 dataset by more than 7%.
- We verify the usefulness of various superpixel features for image classification, where the added shape information alone accounts for an improvement of 6.7% on ImageNet-1k.
- To the best of our knowledge, we are the first to apply a segmentation-based approach to ImageNet-1k classification, highlighting the strong efficiency gains of our framework while obtaining promising initial results.

2. Related work

Work in this area has mainly focused on the pose and appearance aspects of the superpixel representation, as obtained by the SLIC algorithm [2, 6, 16, 17, 3]. Pose is represented using two-dimensional coordinates directly [2] or linear projections [16], with one recent work using trainable position embeddings [3]. We use the same coordinate projection to *Linh and Youn* [16] and additionally include superpixel size, which previous work does not employ, presumably due to the similar size of SLIC patches. To the best of our knowledge, the orientation and shape of the superpixels have not been incorporated in previous work.

The results obtained by using traditional methods of superpixel segmentation are not competitive with CNNs or Vision Transformers, with the best accuracy achieved on CIFAR-10 [13] by *IMGCN-LPE* [3] being 73.09%. This is compared to the state-of-the-art (SOTA) on CIFAR-10 being 99.5% as obtained by a ViT [7]. Unlike other works performing GNN-based classification based on static RAGs [2], *Linh and Youn* develop a GNN which interprets an image as a point cloud. The edge connections between nodes are formed dynamically for each graph convolution using the K-Nearest Neighbour algorithm (KNN) between node features. By contrast, we utilize a static RAG which decreases the resource requirements and allows us to train on bigger datasets like ImageNet-1k. In an attempt to prevent over-smoothing, a common problem with GNNs leading to similar node features after multiple graph convolutions [14, 34], *Long et al*. amongst others [17, 16] concatenate node features of previous layers to retain information from them. As opposed to concatenations, we employ residual connections [10] using addition and show that they perform better in our architecture while resulting in fewer parameters due to the constant dimensionality.

An alternative approach for image classification using GNNs was taken by *Han et al*. [9]. Here, no superpixels were used, with the image instead being split into square regions which were then passed through Multi-Layer Perceptrons (MLPs) in order to obtain feature representations of the regions' pixels. These regions were then used as nodes in a graph, with edges connecting adjacent regions, and fed through a large Graph Transformer. This approach obtained a much higher accuracy of 82% on ImageNet-1k. However, efficiency was similar to a ViT and this method does not solve any of the presented issues with operating on the raw pixel level. Furthermore, due to the difference in approaches and computational requirements of superpixel-based and region-based GNN classification, the result of this paper was not taken as baseline.

To further motivate the use of superpixel patches for vision tasks, there is evidence that jointly using CNNs and GNNs do provide SOTA in problem spaces where images have easily separable and conceptually cohesive features, such as in the task of aerial image segmentation [6]. Furthermore, using GNNs in this domain allows incorporating additional types of information *e.g*. concept graphs [15].

3. Approach

3.1. Image-to-Graph transformation

SLIC superpixels The first step of the image classification pipeline is transforming the image into a graph (referred to as Image-to-Graph transformation). This is achieved by using a superpixel segmentation algorithm (see Figure 2). Most papers using GNNs for image classifications use Simple Iterative Linear Clustering (SLIC) [1]. This algorithm obtains superpixels by sampling cluster centers and corresponding areas that are overlapping. Then, it employs a KNN algorithm variation to define the superpixels. It assigns pixels to the closest cluster center whose area contains the pixels. However, this results in superpixels that are similar in size and shape and cannot encompass larger regions in a single superpixel. Consequently, their shapes contain little discriminative information.

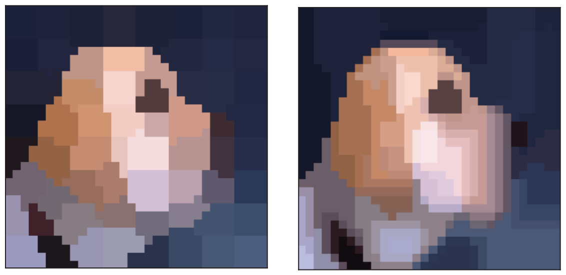
Figure 2. The superpixels generated with SLIC segmentation (left) and Felzenszwalb (right), visualised with the mean colour of their respective pixels.

Felzenszwalb superpixels In contrast, this paper explores using a different segmentation algorithm proposed by Felzenszwalb and Huttenlocher, hereby called Felzenszwalb, where superpixels attempt to minimise the internal adjacent pixel colour difference while maximising the external one [8]. The superpixels produced by this algorithm are globally informed, unlike with SLIC, and can have strongly varying shapes and sizes while the efficiency is preserved in our implementation (see Supplementary Material for more details). It is important to note that, while a region size equal to the image size and requiring an independent number of superpixels could theoretically result in more global SLIC superpixels, this goes against the assumptions made in its original paper [1] and severely hurts the method's efficiency. For Felzenszwalb, superpixels that have less than min_size pixels are merged, in an increasing order determined by the edge colour difference between superpixels.

Global graph creation Once a segmentation produces a labelling that associates all pixels in an image with a superpixel, this labelling is turned into an image graph $\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_g)$. Each superpixel i corresponds to a node $v_i \in \mathcal{V}$ and adjacent superpixels i and j are connected via edges $e_{i,j} = (v_i, v_j) \in \mathcal{E}_g$. Due to this symmetry, $\mathcal{G}_g$ is an undirected graph, *i.e*. $e_{i,j} = e_{j,i}$. We define the neighbourhood $\mathcal{N}_g$ of node v_i as $\mathcal{N}_g(i) \coloneqq \{j \mid e_{i,j} \in \mathcal{E}_g\}$. The features $\mathbf{x}_i$ that describe a superpixel i can be characterised as belonging to three different categories: pose, appearance, and shape. In this paper, pose is taken as the position of the centroid of the superpixel in the image as well as its size in pixels. Furthermore, we experiment with incorporating the orientation of a superpixel, another aspect of its pose. For appearance, we use the mean colour and, as a new addition, the standard deviation per channel of the pixels corresponding to a superpixel. As shape information, we use the encoded contour of the superpixel patch. More details are given in the subsequent paragraphs. All representations are concatenated to form $\mathbf{x}_i$ (see Figure 1).

Shape extraction The *global GNN* operates on node features $\mathbf{x}_i$ with fixed dimension. To incorporate shape information, the contours need to be described in a fixed dimensionality space as well. To this end, we propose considering the contour of a superpixel k as a graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ itself. The contours are extracted using a contour finding algorithm [29] (see Figure 3b) applied to the superpixel segmentation labelling (see Figure 3a). First, the contours of the shapes are represented using the minimum number of points required, with a line being described by its end points and not by the pixels it spans (see Figure 3c). Then, the contours are approximated using a polygon approximation algorithm, namely the Douglas–Peucker algorithm [25], that simplifies a given polygon until an error threshold ϵ is reached (see Figure 3d). This is done to obtain more generalizable shapes with reduced dimensionality. An illustration of the resulting superpixels and accompanying polygons on an ImageNet-1k image can be seen in the Supplementary Material. The vertices of the resulting polygon are used as nodes $v_{k_i} \in \mathcal{V}_k$ while the polygon's edges define the edges $e_{k_{i,j}} \in \mathcal{E}_k$ of the graph. Just like $\mathcal{G}_g$, $\mathcal{G}_k$ is also undirected. Similar to $\mathcal{N}_g$, the neighbourhood $\mathcal{N}_k(i)$ of node v_{k_i} is defined as $\mathcal{N}_k(i) \coloneqq \{j \mid e_{k_{i,j}} \in \mathcal{E}_k\}$. The node feature $\mathbf{x}_{k_i}$ is defined by the position of the vertex i relative to the centroid of the superpixel.

Shape encoding The resulting graph is passed through a GNN encoder called *local GNN* to obtain a fixed-size representation of the shape of a superpixel which is then integrated into $\mathcal{G}_g$. This framework allows for end-to-end learning by backpropagating from the final image class prediction to the input image. Other preliminary approaches for encoding shapes, including a graph autoencoder, a densified shape representation, and a shape curvature measure were also implemented but subsequently discarded as they underperformed compared to the current end-to-end approach presented here. These approaches are outlined in the Supplementary Material. See Figure 1 for an illustration of the pipeline utilised in this paper. In general, the contours do not take into account any holes in a superpixel that would result from a superpixel being completely contained by another. This relationship should be appropriately modelled by the single edge from this inner superpixel to the outer. If a shape is composed of only one node, that node is added two more times and edges are created as usual. This is done for batching purposes, which is key for efficiently loading data, as described in the Supplementary Material. This only occurred for very few images in both datasets.

Position information representation Besides encoding the positions of the centroids explicitly in the node features of the global graph using Cartesian coordinates, which breaks the rotation invariance of the *global GNN*, the Euclidean distances between the centroids of different patches can also be used as edge features. This can be done for both the local and global graph, preserving rotation invariance while still encoding the distance between different patches. However, in preliminary experiments, the inclusion of only edge weights significantly underperformed position embeddings for the global graph and was thus discarded.

Normalisation In order for the Image-to-Graph transformation to be robust to varying image sizes while encoding pose information, all distances, positions, and sizes are normalised. The pixel count, used as size of a superpixel, is normalised by the total number of pixels in the image. The colour channels of the original image are all normalised to be in the range $[0, 1]$. All normalisation of the coordinates is done in polar space. The centroids of the superpixels are converted to polar coordinates with the center of the image taken as the pole. The radius of each centroid is normalised by dividing it by the Euclidean norm of the dimensions of the image img_norm. For the contour graphs, the pole is taken as the centroid of the respective superpixel. The radius of each point on the contour is divided by the maximum radius of a point on the contour. If the approximated contour contains only a single pixel, as was only observed when using SLIC, this step is omitted. All node features can be normalised by the training dataset's mean and standard deviation of each feature dimension. This was not used due to decreasing performance in initial experiments.

Rotation equivariance and shape rotation representation Although including the pose information of a superpixel will break the rotation invariance of the obtained model, rotation can be encoded in a more easily learnable way by using a rotation equivariant graph convolution, such as EGNN [26], presuming that the superpixel segmentation method is rotation equivariant itself. This can be done both at the local level, obtaining a shape encoding that generalizes to arbitrary local rotations, and additionally at the global level, obtaining a model that can easily handle rotated images. Besides this rotationally equivariant GNN, this paper also explores using a simpler representation of rotation in shapes, where the point with the biggest radius is taken as the reference point with $\phi = 0$ and the other points have their ϕ's shifted by the corresponding angle, with this angle added to the superpixel features. This removes orientation information from the local graph, assuming that there is only one maximum radius per shape. Although this assumption might not hold, it could still provide an appropri-

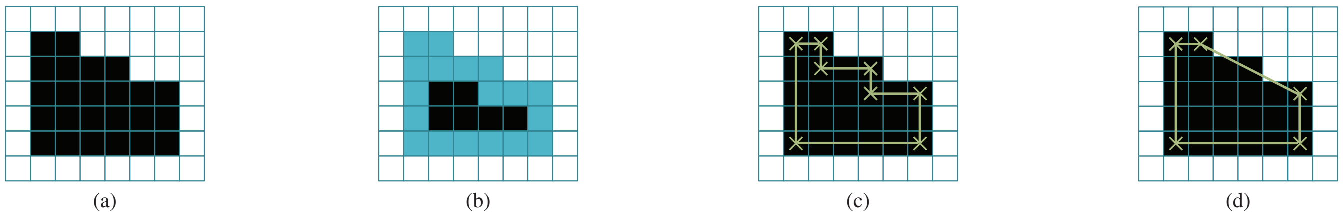

Figure 3. The process of computing the local graph $\mathcal{G}_k$ for a given superpixel: the segmentation labelling indicating which pixels belong to superpixel k (visualised in black) forms the basis (Fig. 3a). Then, a contour finding algorithm identifies the pixels at the superpixel contour (visualised in blue, Fig. 3b). Subsequently, the contour is described using the minimal number of points connected with lines (visualised in green, Fig. 3c). Lastly, the final local graph $\mathcal{G}_k$ is obtained by approximating the previous result (visualised in green, Fig. 3d).

ate training signal without resorting to more complex graph convolutions such as EGNNs.

Preprocessing hyperparameters for variable resolution datasets For ImageNet-1k, given the different resolutions of the images, superpixel segmentation hyperparameters that are resolution dependent need to be scaled according to the image size. Since, on ImageNet-1k, we only show the Felzenszwalb segmentation, only the parameter min_size, defining the minimum size of a superpixel, is scaled by img_norm. Furthermore, for the shape approximation in the ImageNet-1k case, the passed ϵ error threshold is multiplied by the contour size, defined as the number of points of the contour, such that shapes with fewer points are not excessively approximated. These scaling heuristics were observed in initial experiments to result in more visually meaningful superpixels across different image sizes.

3.2. Model architecture

We propose *ShapeGNN*, a new model architecture which operates on the graph of the image and incorporates shape information to derive a final class prediction. The hyperparameters used for our model and each segmentation method were determined in preliminary experiments. Section 4.3 explores variations of this setup.

Based on the image graph, the *global GNN* is used to predict the class of the corresponding sample. The contours of each superpixel are processed by applying the *local GNN* (see Figure 1). After concatenating information about the shape, colour, and size for each superpixel, these feature vectors are used in combination with the centroid positions of each patch in the final, *global GNN*. The basic building blocks of the GNN models are InteractionBlocks and MLPBlocks (see Figure 4). The MLPBlock is constructed by using a linear layer, followed by LeakyReLU [20] as activation function a, and dropout [11]. The InteractionBlocks feature a residual connection and consist of an EdgeConv layer [32] using an MLPBlock to obtain the edge features $h_{\mathbf{\Theta}}$. Equation (1) defines the EdgeConv where we use "mean" as aggregation function $\square$:

$$\mathbf{x}_i' = \underset{j \in \mathcal{N}_g(i)}{\square} h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i) \quad (1)$$

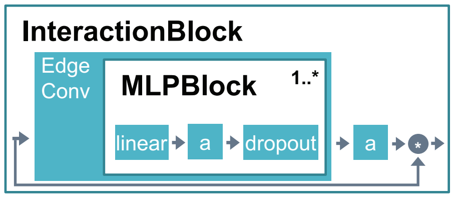

Figure 4. The structure of the building blocks: an InteractionBlock consists of an EdgeConv layer followed by an activation function a. Its output and the original input are combined using the operator $*$, which is addition by default. If the input and output dimensions do not match, the input features are projected using a single linear layer. The edge features h_{Θ} of the EdgeConv layer are computed using an MLPBlock. It consists of a linear layer followed by a and dropout. This can be repeated multiple times. The name "MLPBlock*" in other Figures refers to an MLPBlock where its last linear layer is not followed by a nor dropout.

Pooling the node-wise feature vectors and passing them through another MLPBlock yields the final output.

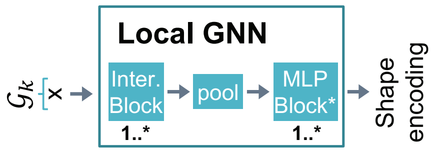

Figure 5. The architecture of the *local GNN*: it consists of an InteractionBlock followed by pooling and a final MLPBlock.

Local GNN The *local GNN* (see Figure 5) operates on the contour graphs $\mathcal{G}_k$ of individual patches using InteractionBlocks with a dimensionality of $d_{l-hidden} = 64$. By default, the number of InteractionBlocks is $num_{l-blocks} = 2$. For pooling, we use the dot-product attention mechanism [4, 31] unlike previous work which used sum [16], mean [17], or max pooling [3]. We first multiply the matrix X_k which contains all node features x_{k_i} as its rows with a learnable weight vector $q \in \mathbb{R}^{1 \times d_{l-hidden}}$. The resulting vector is normalised by applying the softmax function so that the sum of its components equals one. The feature vector x_k for the whole graph $\mathcal{G}_k$ is obtained by another multiplication with X_k, which can be interpreted as a sum of the node features x_{k_i} weighted by their attention weights:

$$x_k = \text{Softmax}(qX_k^T)X_k \tag{2}$$

The final shape representation $\tilde{\mathbf{x}}_k \in \mathbb{R}^{d_{latent}}$ with $d_{latent} = 5$ of superpixel k is obtained using an MLPBlock where the dimensionality of the last layer is d_{latent}. All other layers, if applicable, have $d_{l-hidden}$ nodes. The number of layers in each of the MLPBlocks, *i.e.* the ones in the Interaction-Blocks and the final one, is set to $num_{l-layers} = 2$.

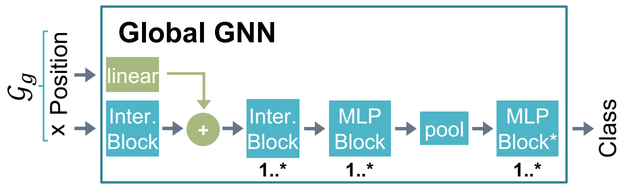

Figure 6. The architecture of the *global GNN*: The node features are processed using an InteractionBlock. If the position of the superpixel centroids is used, their linear projection is added to the resulting feature vectors. After passing them through another InteractionBlock, an MLPBlock is applied to each node feature independently. The node-level features are pooled and the final class prediction is obtained by a final MLPBlock.

Global GNN The *global GNN* (see Figure 6) takes the image graph $\mathcal{G}_g$ as input and predicts the class of the corresponding image. The image graph is passed through an InteractionBlock with a dimensionality of $d_{g-hidden} = 300$. If position is used, the node features are modified by adding a linear projection of the centroid positions. This is followed by $num_{g-blocks} = 1$ InteractionsBlock(s). After that, the node features are modified by node-wise applying an MLPBlock where the last layer contains $2 * d_{g-hidden}$ nodes. The other layers, if applicable, have a dimensionality of $d_{g-hidden}$. After pooling the node features using attention to obtain a single feature representation, a final MLPBlock is applied to obtain the class predictions. Its last layer consists of $n_{classes}$ nodes matching the number of classes in the dataset. All other layers have a dimensionality of $d_{g-hidden}$. The number of layers in each MLPBLock in the *global GNN* is by default $num_{g-layers} = 1$.

4. Experiments

4.1. Experimental setup

We use two image datasets, CIFAR-10 [13] and ImageNet-1k [24]. The former contains 50,000 training and 10,000 test images with a constant resolution of 32x32 pixels. The latter consists of 1,281,167 training and 50,000 validation images of varying resolutions and aspect ratios.

In all our experiments, we use the cross-entropy loss with integrated Softmax as training objective and the AdamW optimizer [19, 23] with a weight decay of 0.05. We keep the learning rate constant at $1\text{e}{-3}$ for the first 20 epochs after which we use cosine annealing [18] with a minimum learning rate of $1\text{e}{-5}$. For this, we use the implementation by *Wightman* [33]. Furthermore, we employ label smoothing and a dropout of 0.1. This optimization setup is based on *Han et al.* [9]. We use a batch size of 256 on CIFAR-10. For ImageNet-1k, we used different batch sizes for the individual runs to improve memory utilisation. The exact numbers are specified in the corresponding table (see Table 3). The Image-to-Graph transformation time is not reported as for CIFAR-10 it is under 10 seconds and for ImageNet-1k it is under 3 hours for the entire datasets and thus negligible compared to the training time.

For both segmentation methods, the image is blurred with a two-dimensional Gaussian filter with border reflection with $\sigma = 0.8$ and kernel size of 5 for both dimensions before being segmented, as recommended in [8].

The hyperparameters used for the Felzenszwalb preprocessing are $scale = 10$, which determines the colour threshold for superpixel merging, and $min\_size = 5$. The ϵ used for the polygon approximation algorithm is 0.02 for both superpixel segmentation algorithms. This results in almost pixel-perfect shapes on the CIFAR-10 dataset, which is mostly unavoidable given the very small size of the images. For ImageNet-1k, hyperparameter scaling was used, as described in Section 3.1. The Felzenszwalb algorithm was the only superpixel segmentation algorithm tested due to its superior performance, used with hyperparameters $min\_size = 0.12$ and $scale = 300$, together with $\epsilon = 0.07$.

For SLIC, we used a region size of 4, corresponding to the desired average width and height of a superpixel, and a compactness of 30, enforcing how square the superpixels should be. These hyperparameters result in a similar mean number of superpixels per image for CIFAR-10 for both algorithms, 64 for SLIC versus 70 for Felzenszwalb.

For CIFAR-10, we train all models for 300 epochs while we use 100 epochs on ImageNet-1k. This improved the performance in preliminary runs compared to splitting the training set into a training and validation set and employing early stopping. If not mentioned otherwise, all reported accuracies on CIFAR-10 were achieved on the held-out test set. For ImageNet-1k, we follow the common practice to report the accuracy on the held-out validation set [21, 12, 22].

All our experiments use an NVIDIA Titan RTX GPU and an Intel Xeon Gold 5118 CPU with 96 GB RAM.

Due to computational constraints, we only run a single run for each setup. However, during preliminary experiments, we saw that results can vary across different seeds by less than 1%. This has to be taken into account when interpreting the results in the next sections.

Model	accuracy
DISCO-GCN[16]	70.0%
HGNN[17]	70.6%
IMGCN-LPE[3]	73.1%
ShapeGNN (Ours)	80.4%

Table 1. Test accuracies on CIFAR-10 for *ShapeGNN* compared to other segmentation-based GNNs for image classification.

4.2. CIFAR-10

We test our model on CIFAR-10 with $n_{classes} = 10$ to compare our method with previous segmentation-based GNNs for image classification [16, 17, 3]. The default setup of *ShapeGNN* has 862k learnable parameters which roughly matches *DISCO-GCN* with 881k parameters [16], the only other model we compare to where this number is specified. The median time for training one epoch is 19.4 seconds. The obtained accuracies are shown in Table 1. *ShapeGNN* outperforms all previous models by 7.35%-10.43%.

The model achieves a training accuracy of 99.9%, which indicates strong overfitting. Further regularization might help in achieving even better performance on the test set. Next, we will investigate which components of *ShapeGNN* are the most important for the increase in performance.

4.3. Ablation study

ablation	accuracy	median time per epoch
ShapeGNN	80.4%	19.4s
no shapes	77.9%	12.0s
no shapes $d_{g-hidden} = 310$	78.4%	12.4s
no colour std. dev.	78.1%	19.7s
no size	80.0%	19.7s
SLIC	79.3%	18.2s
SLIC no shapes	77.6%	10.7s

Table 2. Test accuracies for different ablations of *ShapeGNN*.

We first examine to what extent the added information about shape, appearance, and pose affect the final test performance (see Table 2).

Shapes improve performance Removing the shape information reduces the performance by 2.5% while also decreasing the time per epoch by $7.4s$. Because this also reduces the number of parameters, we run another variant with $d_{g-hidden} = 310$. This variation has roughly 17k more parameters than *ShapeGNN* but still underperforms by 2%. This shows that shape information is beneficial for predicting classes albeit at a computational cost. The performance gain might even be higher with less pixelated shapes on higher-resolution images, which is explored in the experiments based on the ImageNet-1k dataset in Section 4.4.

Encoding colour variance helps Incorporating more colour information by using its standard deviation increases the performance by 2.3%. This illustrates that even simple additions enriching appearance information aid prediction. Examining more sophisticated ways of adding further appearance information is a clear avenue for future research.

Superpixel sizes provide little benefit Utilising the size of a patch as additional information about its pose improves performance by only 0.4%. This small difference does not allow for a final assessment of this component, but it was included due to its negligible impact on efficiency and small improvement in accuracy.

Felzenszwalb outperforms SLIC When using the SLIC segmentation method instead of Felzenszwalb, the accuracy decreases by 1.1%. The performance gain provided by the shapes is also reduced by 0.8%. This is in line with our assumptions about the SLIC superpixels, although the magnitude of the effect is weaker than expected. This is perhaps due to the pixelated shapes on CIFAR-10 being in general of limited use as compared to a larger resolution dataset, as will be seen in Section 4.4.

Dynamic edges harm performance To see whether the dynamic construction of edges provides a better basis than the adjacency of superpixels, we employ the dynamic version of EdgeConv which constructs edges using KNN based on node features. We use $K = 20$ following previous work [16]. The model, having the same number of parameters, obtains an accuracy of 74.8% while taking 355 seconds per epoch. Although over-smoothing can also be a problem with the high K used in the previous papers, this might indicate that superpixel adjacency is more informative, supporting our initial intuition.

Attention pooling and bigger models work best Our ablations for attention pooling and residual connections using addition show that the latter outperforms the variant using concatenation by 1.4%. The former improves the accuracy by 0.7%-1.6% as compared to mean, max, and sum pooling. Both results indicate the benefit of these new additions of *ShapeGNN*. As another ablation, we vary the capacity of the *local* and *global GNN*. Drastically increasing the model size to $3.5M$ parameters only improves the accuracy marginally by 0.3%. By contrast, reducing the number of parameters by two-thirds reduces the performance by only 0.5%. While we chose the presented configuration of *ShapeGNN* because preliminary results illustrated its superiority on ImageNet-1k, the smaller variant might be an interesting trade-off between performance and efficiency on CIFAR-10. Changes to the *local GNN*'s capacity reduces the performance by 0.4%-0.5%. This suggests that while a small *local GNN* cannot extract useful shape information, a model that is too big overfits to the different shapes.

Rotationally equivariant representations are worse One advantage of GNNs is the easy incorporation of different equi- and invariances. As an example, we test the rotation information described in Section 3.1, where the superpixel rotation angles are separately added to the global graph. This could enable the *local GNN* to generalize better to shapes of different rotations. However, our results show a performance reduction by 1.8%. Apparently, incorporating the different rotations in the shape encoding is beneficial. Considering the previous ablation results, the incorporation of rotation information in the *local GNN* could also have a regularising effect, mitigating the problem of overfitting. As an alternative, we experiment with using EGNN layers [26], which are rotationally equivariant graph convolutions. Using coordinates in the local and global graph bypasses the equivariant architecture of this layer. Therefore, we use distances to the superpixel center as node features in the local graph and exclude superpixel centroid coordinates in the global graph. In addition, the MLPBlock used in the InteractionBlock is now applied node-wise before the EGNN. Our results show a severe drop in performance by 21.7%. This can be partly explained by the importance of the centroid coordinates, whose removal decreases the accuracy by 9.1%. However, this result still indicates that limiting the model by explicitly using rotationally equivariant graph convolutions drastically reduces performance.

More detailed results of the performed ablations are presented in the Supplementary Material.

4.4. ImageNet-1k

To evaluate our approach on a high-resolution image dataset, we perform experiments using the ImageNet-1k dataset with $n_{classes} = 1000$. The training procedure and *ShapeGNN* architecture on this dataset are equivalent to the one on CIFAR-10, with the exception being the different hyperparameters for the preprocessing, as described in Section 4.1. We also report results when adding one more InteractionBlock and when removing shapes. These analyses should give an indication about the scaling potential of *ShapeGNN* and the relevance of shapes.

	accuracy	median time per epoch	batch size
ShapeGNN	46.8%	1045s	480
no shapes	40.1%	706s	640
$num_{g-blocks} = 2$	50.4%	1430s	320

Table 3. Accuracies on the ImageNet-1k held-out validation set.

The accuracy obtained by *ShapeGNN* is 46.8%, which is relatively low compared to modern, efficient CNN and ViT architectures such as EdgeNeXt-S with 79.4% [21]. This suggests that the appearance aspect is still most likely underrepresented for ImageNet-1k. However, this accuracy and the efficiency with which it was obtained proves the feasibility of our approach for more complex datasets. Training on ImageNet-1k takes a comparable amount of time as EdgeNeXt-S, while only requiring a single GPU, an RTX Titan, as compared to 8 stronger A100 GPUs [21].

Considering that only two InteractionBlock were used, resulting in nodes passing information exclusively to very closely connected nodes, the result indicates that the graph representation of the images has meaningful component parts that can be used by a model with little depth. The positive effects of depth, however, are still present. Adding one more InteractionBlock to the network increased the accuracy to 50.4% while increasing the epoch time by 40%.

Crucially, the results show that incorporating shape information has a much bigger effect than in the previous experiments, increasing the accuracy by 6.7%. This shows that, as hypothesised before, higher resolution images might feature more representative shapes, whose encoding is especially useful in more complex classification tasks.

5. Conclusion

The models and techniques explored in this paper have managed to surpass the previously established SOTA in segmentation-based image classification on CIFAR-10 by a significant amount. Furthermore, they prove promising for training on high-resolution image datasets such as ImageNet-1k in a very efficient manner.

An important component that helped achieve this result was the shift to a superpixel segmentation algorithm where superpixels are globally informed and do not have the same size and shape. These superpixels seem more representative of the underlying image for the task of image classification.

Incorporating the shape and the colour standard deviation into the prior does seem to provide decent accuracy improvements on CIFAR-10. More complex representations of the appearance might be a promising future research path. While the inclusion of standard deviation comes at almost no computational cost, the shape encoding setup does come with a relevant performance trade-off. Although this trade-off might not be worth making on CIFAR-10, it yields significant gains on ImageNet-1k. This is most likely because, on images that were not intentionally downscaled, the shapes are more representative of the objects in the image and are not very fine-grained representations of the pixel grid that cannot be utilised in a robust manner.

Thus, the combination of novel superpixel features and a different segmentation algorithm has proven to be useful additions to segmentation-based image classification. The efficient training on large datasets and initial results suggest that increasing the amount of information in the image graph could further narrow the model's performance gap to CNNs and ViTs while maintaining its superior efficiency.

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[2] Pedro HC Avelar, Anderson R Tavares, Thiago LT da Silveira, Clíudio R Jung, and Luís C Lamb. Superpixel image classification with graph attention networks. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 203–209. IEEE, 2020.

[3] Ji-Hun Bae, Gwang-Hyun Yu, Ju-Hwan Lee, Dang Thanh Vu, Le Hoang Anh, Hyoung-Gook Kim, and Jin-Young Kim. Superpixel image classification with graph convolutional neural networks based on learnable positional embedding. *Applied Sciences*, 12(18):9176, 2022.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Lina I Davitt, Filipe Cristino, Alan C-N Wong, and E Charles Leek. Shape information mediating basic-and subordinate-level object recognition revealed by analyses of eye movements. *Journal of Experimental Psychology: Human Perception and Performance*, 40(2):451, 2014.

[6] Qi Diao, Yaping Dai, Ce Zhang, Yan Wu, Xiaoxue Feng, and Feng Pan. Superpixel-based attention graph neural network for semantic segmentation in aerial images. *Remote Sensing*, 14(2):305, 2022.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[8] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[9] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: An image is worth graph of nodes. *arXiv preprint arXiv:2206.00272*, 2022.

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

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

[13] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[14] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[15] Dan Lin, Jianzhe Lin, Liang Zhao, Z Jane Wang, and Zhikui Chen. Multilabel aerial image classification with a concept attention graph neural network. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–12, 2021.

[16] Vu Le Linh and Chan-Hyun Youn. Dynamic graph neural network for super-pixel image classification. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1095–1099. IEEE, 2021.

[17] Jianwu Long, Zeran yan, and Hongfa chen. A graph neural network for superpixel image classification. In *Journal of Physics: Conference Series*, volume 1871, page 12071. IOP Publishing, 2021.

[18] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[20] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

[21] Muhammad Maaz, Abdelrahman Shaker, Hisham Cholakkal, Salman Khan, Syed Waqas Zamir, Rao Muhammad Anwer, and Fahad Shahbaz Khan. Edgenext: efficiently amalgamated cnn-transformer architecture for mobile vision applications. *arXiv preprint arXiv:2206.10589*, 2022.

[22] Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[23] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Alan Saalfeld. Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999.

[26] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[30] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[32] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[33] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

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