Supplementary Material

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

A. Superpixel segmentation

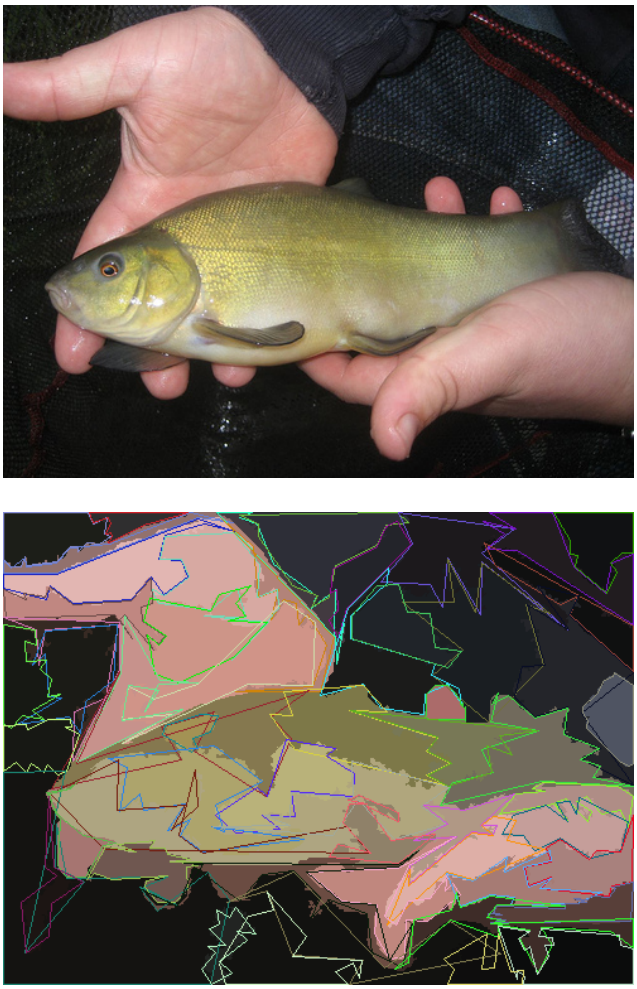


Figure 1. An image of a fish from ImageNet-1k and the visual representation of its graph after using the Image-to-Graph transformation. The superpixel mean colour is used as the colour for each superpixel. The approximations of the contours are illustrated with random colours.

*Joint first authors

B. Implementation details

B.1. Segmentation efficiency

SLIC [1] is more efficient than Felzenszwalb and Huttenlocher's method [3] in the general case and has linear complexity in the number of pixels, provided a small region size independent of the image size is given as hyperparameter. Felzenszwalb requires the construction of a graph of pixels, where each pixel is connected through edges to adjacent ones. In addition, the method relies on these edges being sorted by their weights, which are determined by the Euclidean distances between the pixels' colour values. Such a sort has complexity $\mathcal{O}(n \log n)$, where n is the number of edges. However, if considering images where each colour channel is an integer in the range $[0, 255]$, a common way to store images, and sorting the squares of the edge weights, which are integers between 0 and $3 * 255^2 = 195075$, a counting sort can be used, which has complexity $\mathcal{O}(n)$. Although this approach still does not make the Felzenszwalb algorithm linear in complexity, it significantly improves performance and makes it competitive with SLIC. In our experiments, Felzenszwalb implemented in this way was faster than SLIC.

B.2. Data loading and batching

An important reason for the efficiency of our method is the use of a customised storage format and retrieval procedure. For both training and testing data separately, all images are compressed into three files. One contains the data in dense format describing only the necessary data required to construct the superpixel data and edges between them in an efficient way. Another contains the offsets at which each image starts, and the last consists of image statistics such as mean and standard deviation of the mean colours of all superpixels, which might be used for normalisation. This storage format, for the parameters used in the paper for Felzenszwalb, made the ImageNet-1k dataset three times smaller than the original version, approximately 39 Gigabytes instead of 136 Gigabytes for the training set. This allows us to keep both the training and testing datasets in memory, where the content of each of the files is stored in one tensor.

Given this representation, we can, given a tensor of indices, construct a batch of image graphs in a vectorised fashion without iterating over the individual samples. The shape contours are also batched together, ensuring the efficiency of the end-to-end training approach. Our implementation can be found in the source code that is provided as part of the Supplementary Material.

C. Alternative shape encoding approaches

This section presents three alternatives to encode shape information. All of them performed worse in preliminary experiments and were therefore discarded in favour of the end-to-end approach used in the paper.

C.1. Shape curvature measure

One simple approach for enriching the image graph with shape information is to use one scalar that denotes the curvature of a shape. At each point on the polygon, the maximum of the interior and exterior angle, normalised to be between $[0, 1]$, is used. Taking the mean of these angles results in a scalar value indicating the shape's curvature (see Figure 2).

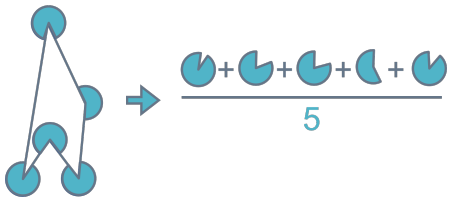


Figure 2. Visualisation showing the calculation of the curvature measure.

C.2. Densified shape representation

Another method is to sample a fixed number of equidistant points on the superpixel contour. The points are represented using polar coordinates, with normalised radians, with respect to the centroid of the patch (see Figure 3). This results in a fixed size feature vector that can be incorporated into the main graph.

Figure 3. Visualisation showing the densified shape representation.

C.3. Graph autoencoder

Instead of training a shape encoder, like the *local GNN*, end-to-end with the *global GNN* on the image classification task, we can also pretrain the shape encoder on a proxy task. For this, we employ an autoencoder-like setup where the graph is encoded into a fixed size vector using a GNN (see Figure 4). Since the nodes are connected to exactly two other nodes on a contour, we can subsequently use an LSTM [5] decoder to predict the next node given the previous prediction. The first input is a special beginning of graph token and the initial hidden state is the latent feature vector. We employ teacher forcing during training to speed up convergence.

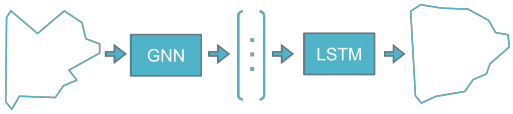


Figure 4. Visualisation showing the autoencoder setup.

While an LSTM assumes a sequential ordering and a starting point, assumptions that do not hold for a graph, it does allow for decoding the feature vector into a node sequence of variable length. This setup enables us to encode shapes with a variable number of corners into a fixed size representation using a GNN.

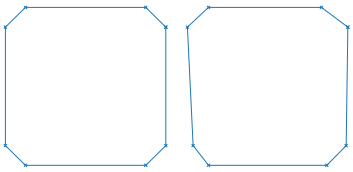
The nodes encode the polar coordinates (with normalised radians) with respect to the centroid of the patch and the edges of the graph encode the distance between the nodes. The output that the decoder needs to predict is the ordered sequence of node features. The starting point is arbitrary, but the polygon approximation method does generate nodes in a counter-clockwise fashion starting from the top right of the shape, which can thus be consistently interpreted by the LSTM. The loss used is Mean Squared Error (MSE). More complex approaches are possible, but for a problem of this small size we consider MSE to be sufficient.

The encoder consists of a GCN layer [6] that operates on the patch graph. After applying the ReLU activation function, sum-pooling is used to obtain a fixed length feature representation of the shape. Finally, a linear layer maps this vector to the latent representation.

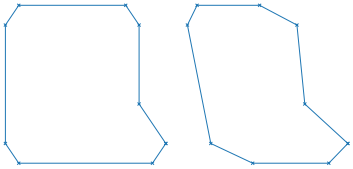
The decoder uses an LSTM layer whose initial hidden state is set to the latent vector. At each step, its output is passed through a linear layer which predicts the polar coordinates. To ensure that the outputs match the target domains, we use the ReLU activation function for the distances and the sine function for the normalised radians to encode their periodic nature.

Figures 5 and 6 show illustrations of superpixels based on images in the CIFAR-10 test set (left) and their reconstructions by the autoencoder (right). It can be seen that, while graphs violate the assumptions made by the LSTM,

the autoencoder is able to decently reconstruct most shapes. However, one main disadvantage of this method compared to the end-to-end approach employed in the paper is that separately training the shape-encoding GNN drastically increases the total training time.



(a) Many SLIC patches have a simple square-like shape which is well reconstructed.



(b) Some SLIC patches have a slightly more complex shape. However, the autoencoder is still able to roughly reconstruct the shape.

Figure 5. Illustrations of SLIC superpixels (left) and their reconstructions by an autoencoder (right).

D. Further ablation details

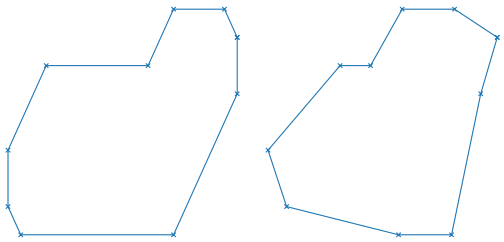
This section presents more details about the ablation study performed in the paper. All results are based on the CIFAR-10 dataset.

D.1. Dynamic edge construction

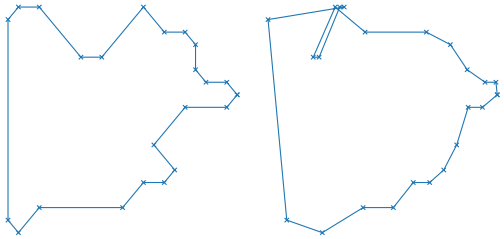
ablation	accuracy	median time per epoch
ShapeGNN	80.44%	19.4s
DynamicEdgeConv	74.8%	355.5s

Table 1. Test accuracy after replacing EdgeConv with DynamicEdgeConv [10], which dynamically constructs edges.

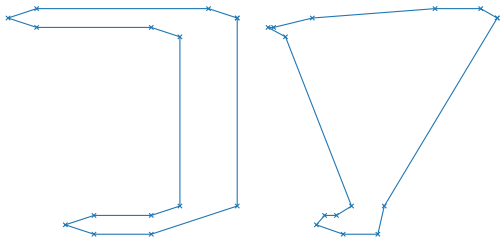
Table 1 shows the accuracy obtained by replacing the EdgeConv layer in *ShapeGNN* with DynamicEdgeConv [10]. By doing that, the original adjacency matrix is ignored. Instead, the graph is interpreted as a point cloud where each DynamicEdgeConv layer dynamically constructs edges using the K-Nearest Neighbour algorithm (KNN) based on the node features. This setup results in a much slower training, with an epoch taking more than 18 times as long, while the accuracy decreases by approximately 5.6%.



(a) Easier Felzenszwalb patches are well reconstructed.



(b) In general, Felzenszwalb superpixels can be much more complex than the ones based on SLIC. Although the general shape of the patch is captured reasonably well, some larger errors occur (see top of the shape).



(c) For even more complex Felzenszwalb superpixels, the patch and its reconstruction only share basic similarities, such as the three corners in the top left, right and bottom left. Between these corners, the two shapes deviate drastically.

Figure 6. Illustrations of Felzenszwalb superpixels (left) and their reconstructions by an autoencoder (right).

D.2. Superpixel position

ablation	accuracy	median time per epoch
ShapeGNN	80.44%	19.4s
no position information	71.3%	18.9s

Table 2. Test accuracy after removing the centroid position information.

Table 2 shows that removing the information about the centroid coordinates leads to a drop in accuracy by 9.1%. This indicates that superpixel positions are crucial for the task of image classification.

ablation	accuracy	median time per epoch
ShapeGNN	80.44%	19.4s
no residual connections	79.0%	18.6s
concatenation for residual connections	79.0%	21.2s

Table 3. Test accuracies for different residual connection ablations.

D.3. Residual connections

A new aspect of our proposed architecture is the use of residual connections using addition. The results of varying the type of residual connection, denoted $*$ in the paper, are shown in Table 3. When we use concatenation for residual connections, similar to [7], the number of nodes for the next layer increases accordingly. It can be seen that using addition in residual connections performs 1.4% better than no connections and concatenation which interestingly perform the same. This is despite the fact that due to the increased dimensionality, the concatenation variant has 540k more parameters. This indicates an advantage of using residual connections with addition instead of the other alternatives.

D.4. Pooling methods

ablation	accuracy	median time per epoch
ShapeGNN	80.44%	19.4s
mean pooling	79.6%	19.5s
max pooling	78.8%	19.5s
sum pooling	79.7%	19.0s

Table 4. Test accuracies for different pooling ablations.

Attention pooling is another architectural change we introduce. Table 4 shows the results for the corresponding ablations. Attention pooling improves upon its alternatives by 0.7%-1.6%. While the difference in performance is relatively small compared to the mean or sum pooling, this result might suggest the benefit of node-wise weights in attention pooling.

D.5. Model capacity

We further explore the influence of model capacity on performance by varying the number of linear layers and InteractionBlocks in the *local* and *global GNN* in addition to the dimensionality of the shape encoding. The results are shown in Table 5. Decreasing the number of parameters of the *global GNN* to less than a third of the vanilla *ShapeGNN*, reduces the performance by only 0.5% while running approximately 1.6 times faster. Increasing the capacity of the *global GNN* to $3.5M$ parameters surpasses *ShapeGNN* in terms of accuracy by 0.3%. Considering that the runtime more than doubled, the small gains in performance are negligible. This might indicate that the image graph does not capture enough valuable information which could be exploited by a more complex model. Although this paper increased the amount of information encoded in the image graph, future work could try to further enrich the image representation.

ablation	accuracy	median time per epoch
ShapeGNN	80.44%	19.4s
$d_{g-hidden} = 150$	79.9%	14.2s
$d_{g-hidden} = 450$ $num_{g-blocks} = 2$ $num_{g-layers} = 2$	80.7%	50.7s
$d_{l-hidden} = 32$ $num_{l-blocks} = 1$ $num_{l-layers} = 1$	79.9%	16.3s
$d_{l-hidden} = 128$ $num_{l-blocks} = 3$ $num_{l-layers} = 3$	80.2%	31.8s
$d_{latent} = 1$	79.2%	19.6s
$d_{latent} = 10$	80.3%	19.2s

Table 5. Test accuracies for *ShapeGNN* variants with different capacities.

When changing the capacity of the *local GNN*, both increasing and decreasing the number of parameters does not improve the performance. This indicates that a small *local GNN* is not sufficient to extract discriminative shape features while a higher capacity increases the risk of overfitting to shapes. This explanation is supported by the ablations varying the amount of information passed between the *local* and *global GNN* by changing d_{latent}.

E. Used software

The following software, that provides implementations of some of the discussed algorithms, was used alongside the code provided in the Supplementary Material:

1. OpenCV 4.5.3 [2], used for the contour finding algorithm [9] and SLIC implementation [1].

2. Pytorch 1.12 [8], used for the neural network-focused modules composing the models.

3. Pytorch Geometric [4], used for the graph convolutions.

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. 1, 4

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 4

[3] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004. 1

[4] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, 5 2019. 4

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016. 2

[7] 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. 4

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4

[9] 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. 4

[10] 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. 3