*J Mater Sci* (2025) 60:6907–6921

Composites & nanocomposites

# Vision GNN (ViG) architecture for a fine-tuned segmentation of a complex Al–Si metal matrix composite XCT volume

M. Lapenna[1,*], A. Tsamos[2,3,*], F. Faglioni[4], R. Fioresi[5], F. Zanchetta[5], and G. Bruno[2,3]

[1] DIFA, Unibo, Via Irnerio 46, 40126 Bologna, Italy
[2] Bundesanstalt fur Materialforschung und-Prufung, BAM, Unter den Eichen 87, 12205 Berlin, Germany
[3] Institute of Physics and Astronomy, University of Potsdam, Karl-Liebknecht-Straße 24/25, 14476 Potsdam, Germany
[4] DGSC, Unimore, Via Campi 103, 41100 Modena, Italy
[5] Fabit, Unibo, Via San Donato 15, 40127 Bologna, Italy

**Received:** 30 July 2024
**Accepted:** 21 March 2025
**Published online:** 23 April 2025

© The Author(s), 2025

## ABSTRACT

In this paper, we implement a vision graph neural network (ViG) architecture to segment microstructures in X-ray computed tomography 3D data. Our ViG architecture is first trained on a synthetic augmented dataset, and then fine-tuned on experimental data to obtain an improved segmentation. Successively, we assess the accuracy of the segmentation on manually-labeled experimental slices. We exemplarily use the approach on a complex microstructure: a metal matrix composite, reinforced with two ceramic phases, intermetallic inclusions and a silicon network, in order to show the generality of our method. ViG model proves to be more efficient than U-Nets in adapting to new data when fine-tuned on a small portion of the experimental data. The fine-tuned ViG shows comparable performance to U-Nets, while largely reducing the number of trainable parameters, with the potential of greater adaptability and efficiency.

## Introduction

Quantitative analysis of X-ray computed tomography (XCT) is going through a new era thanks to the successful application of deep learning algorithms (see [1] and refs therein, for recent developments).

Quantitative microstructural analysis is vital for material qualification, and XCT images significantly reduce the time and cost associated with this process. Given the difficulties and time demands of manually segmenting these microstructures, automatic segmentation using neural networks has proven highly effective [2–4]. However, training supervised deep learning models typically requires a large amount of labeled data, which is where synthetic datasets become valuable, alleviating the burden of

Handling Editor: David Cann.

Address correspondence to E-mail: michela.lapenna4@unibo.it; athanasios.tsamos@bam.de
E-mail Addresses: francesco.faglioni@unimore.it; rita.fioresi@unibo.it; ferdinando.zanchett2@unibo.it; giovanni.bruno@bam.de

https://doi.org/10.1007/s10853-025-10834-5

time-consuming manual-labeling of experimental 3D volumes.

Specifically, 3D deep convolutional neural networks (DCNNs) [5], 2D DCNNs [6], and autoencoders [7] have been utilized for automatic segmentation of XCT reconstructions. Originally designed for detection and segmentation in medical imaging (see [8] for a recent review), these techniques have now established themselves as state-of-the-art in the field of materials science.

In modern computer vision tasks, such as classification, object detection, and semantic segmentation, convolutional neural networks (CNNs) represent the standard technique [9–11]. However, recently, transformer architectures with attention mechanisms have been also exploited for visual tasks [12, 13]. Different networks require the input to be visualized and structured in different ways: CNNs view the image as a regular grid of pixels, while transformers treat the image as a sequence of patches. In this work, we exploit a further approach: we view the image as a graph and we employ graph neural networks (GNNs) architectures (see [14, 15] and refs therein) to segment a challenging six-phase Al–Si alloy composite reinforced with ceramic fibers and particles. Since these microstructures can have irregular shapes, the flexibility of learning from the graph structure may help in the segmentation task.

Due to comparable X-ray attenuation coefficients (and hence similar densities), different microstructural phases can exhibit similar grayscales in reconstructed XCT data [4]. Consequently, the 3D shape of these phases becomes crucial for differentiating them, even when their gray levels are alike. Therefore, segmenting the entire 3D reconstructed volume is the most effective method for distinguishing these multiphase composite materials, rather than relying solely on 2D slices. Additionally, recognizing geometric features can be beneficial when multiple objects (or phases, in this context) are present [3, 4].

By now, the application of GNNs in the field of computer vision mainly include point clouds classification and segmentation [16–18]. To make a GNN suitable for visual tasks, it is important to overcome the common phenomenon of *over-smoothing* [19]. Hence, we exploit the successful framework of vision GNN (ViG) introduced in [20] to enlarge our previous GNN architecture [21] and enhance the final segmentation.

## The advantages of GNNs

One of the primary advantages of GNNs is their ability to process graph-based data, treating the 3D volume as a graph where voxels serve as nodes interconnected by edges. Due to how the graph convolution works (see Sec. 4), the spatial orientation of the graph does not influence the learning. Usually, when segmenting images with 2D and 3D DCNNs, the kernel shifts on 2D and 3D Euclidean grids and the DCNN is translation equivariant with respect to the segmentation task. This means that, if an object's position in an image is shifted by some pixels, the output (e.g., the feature map in a CNN) will also be shifted by the same number of pixels. In the case of segmentation with GNNs, instead, the kernel is not moving by translation and graph convolutions are equivariant also with respect to rotation of the graph and permutation of the nodes. This characteristic of GNNs fits our task, since voxel classification is independent of voxel spatial orientation. Also, having such orientation symmetries automatically built in allows to reduce the number of trainable parameters with respect to usual DCNNs.

Another advantage of GNNs is their ability to solve node classification problems in a semi-supervised setting: the GNN has access to the labels of a restricted set of nodes but exploits the information coming from the graph structure to classify the remaining nodes (see [22] for a clear illustration of this fact in a toy social network example [23]). This behavior can be exploited to fine-tune the trained architecture on experimental data. Once the model is trained on the synthetic dataset, we expect that further training on a small portion of the experimental data can greatly improve the final segmentation of experimental data. To this end, a 3D manual-labeling of experimental data is required, but only for a small percentage of voxels (unlike what happens for classic DCNNs). We give details about fine-tuning in Sec. 4.3 and present the results in Sec. 5.

These structural advantages of GNNs could not only allow to reduce the computational costs of training (training time and hardware requirements) but also to possibly solve more challenging tasks.

## Materials and data

We segment an AlSi12CuMgNi matrix metal composite (MMC) reinforced with 7%vol $Al_2O_3$ short fibers and 15%vol SiC particles, as detailed in [4]. The

synchrotron X-ray computed tomography (SXCT) data were collected at the BAMline beamline at BESSY II synchrotron in Berlin, Germany. More details about the SXCT imaging and the analysis of the microstructures are contained in [3].

In Fig. 1, we present a 512 × 512 pixel cross-section of the XCT volume, illustrating how different microstructures exhibit similar gray levels.

To train supervised deep learning algorithms effectively, a substantial amount of labeled data is essential. Therefore, we train our GNN using a synthetic dataset (see Fig. 1), whose detailed generation procedure is outlined in [4]. Here, we provide a brief summary: synthetic Al–Si MMC microstructures are created using BAM's in-house MATLAB library (BAM SynthMAT [4]). This generation process accounts for structural and grayscale resemblance, positioning the microstructures into a single volume using a priority function [4]. The priority function is necessary due to the inhomogeneous spatial distribution of particles and volume fractions in the experimental data. Additionally, voxel grayscales are assigned based on local contrast, noise, and blur, allowing us to achieve a spatial and grayscale distribution of the constituent phases that closely mimics the experimental one.

The final synthetic volume comprises two raw eight-bit binary files: one containing the grayscale values (ranging from 0 to 255) and the other containing labels for the various synthetic phases (with labels ranging from 0 to the number of phases-1). For the Al–Si MMC under investigation, there are six distinct phases: voids, $Al_2O_3$ fibers, intermetallics

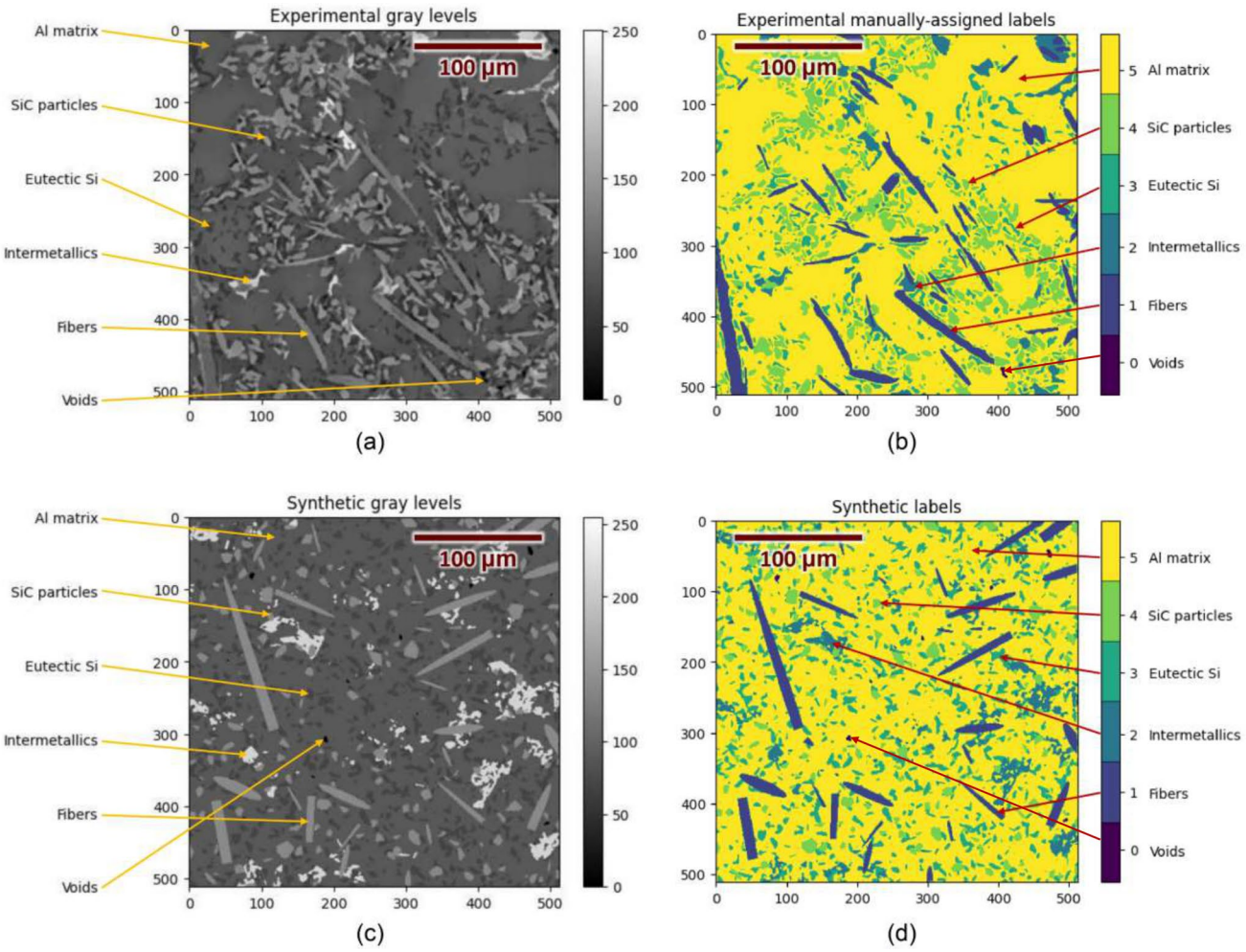


**Figure 1** Experimental (top: (**a**), (**b**)) and synthetic (bottom: (**c**), (**d**)) slices of the AlSi12CuMgNi Metal Matrix Composite (MMC). On the left the gray level (an integer between 0 and 255), on the right the label corresponding to its class (a number ranging from 0 to 5). Arrows point to the six phases for clarity.

(IMs), eutectic Si, SiC particles, and the Al matrix (see Table 1). Since the phases are not homogeneously distributed in the volume, we carefully weight the contribution of each phase when computing the loss function (see Sec. 4). Specifically, we take into account the average occurrence of each phase across the synthetic volumes (see Table 1). This is calculated by counting the average number of voxels associated with each class among the dataset volumes, and then, it is normalized with respect to the sum of all classes average occurrences.

The synthetic dataset comprises eight Al–Si MMCs volumes, each with dimensions of $512 \times 512 \times 512$ voxels and generated using various parameters such as fiber sizes, lengths, orientations, grayscales, and volume fractions. Out of these, seven volumes are designated for training and validation, while one is reserved for testing. Figure 1 displays a cross-section from one of the synthetic volumes along with its corresponding labels.

The experimental dataset used to evaluate the trained model includes four conditioned XCT volumes, also sized at $512 \times 512 \times 512$ voxels, each containing a single manually-labeled slice (see Fig. 1).

**Table 1** Mean occurrence of the six phases among the eight volumes of the synthetic dataset (in increasing order)

| Voids | IMs | $Al_2O_3$ Fibers | SiC Particles | Si | Al Matrix |
|---|---|---|---|---|---|
| 0.13 % | 3.90 % | 4.32 % | 4.41 % | 10.31 % | 76.92 % |

These slices serve as ground truth for assessing the model's performance on the experimental volumes. Experimental data are conditioned with two conditioning methods: by a non-local means filter [24], and by BAM SynthCOND, a deep conditioning framework introduced in [25]. The purpose of conditioning the experimental data is to reduce noise and blur, thus improving the final segmentation. BAM SynthCOND trains conditioning DCNNs with synthetic data. Therefore, since we are training a segmenting architecture with analogous synthetic data, further improvement in the final segmentation can be achieved [26]. In Fig. 2, we compare the different conditioning methods.

Since the synthetic grayscale distributions are not perfect models of the experimental ones, the synthetic training dataset is augmented to increase the generalization ability of the model [4]. Four augmentations are applied: contrast and brightness augmentations, in random orders and intensity (±10% for both); Gaussian noise (random standard deviation of $0-8$); and 3D Gaussian spatial blur (random sigma of $0-1$). The brightness/contrast augmentations have the aim of better including different material interfaces in the experimental volumes, so that the 3D interface geometry is better recognized as well. The purpose of the introduction of artificial noise and blur is to make the training more challenging and thus further improving generalization. The augmentations are randomly applied to sub-volumes during slicing and not to the seven synthetic volumes as a whole, so that the number of combinations of the augmentations is increased

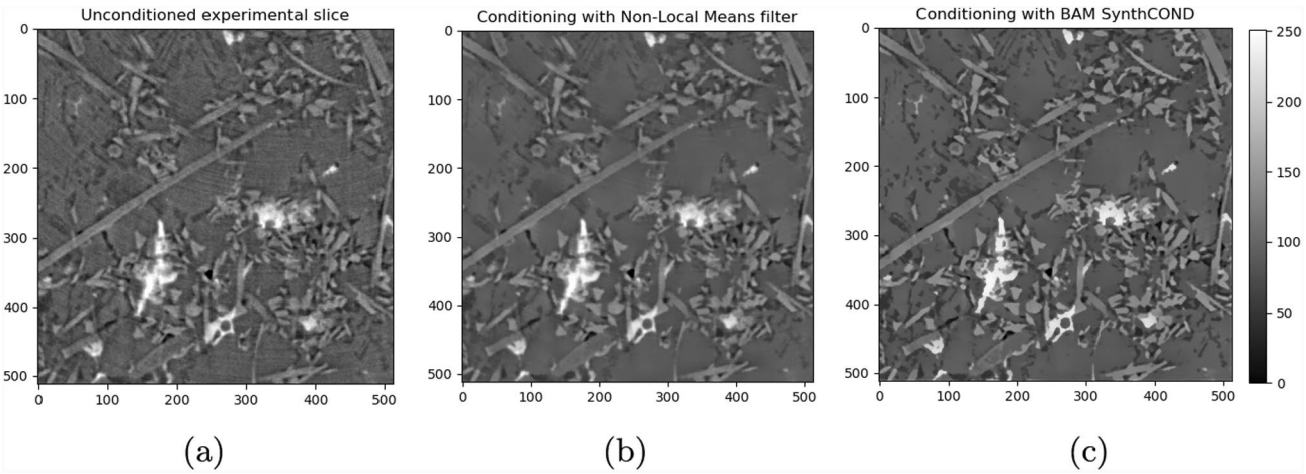


**Figure 2** Conditioned experimental slices of the AlSi12CuMgNi Metal Matrix Composite (MMC). From left to right: **a** the unconditioned experimental slice, **b** the non-local means filter conditioning [24], and (**c**) the conditioning with BAM SynthCOND [25].

and generalization is further improved (random augmentations were applied to more samples.)

Differently from what we did in our previous work [21], this time we train the GNN architecture on the augmented dataset, to see how augmentations improve the performance on experimental volumes.

## Methods

### Graph Convolutional Layers

Segmenting a volume involves classifying each voxel by assigning to it a label, allowing us to determine the class it belongs to and enabling semantic recognition of objects within the 3D reconstruction (i.e., Aluminum Fibers, Intermetallics, etc.). We approach this task as a supervised node classification problem using GNNs.

We begin by constructing a graph from the XCT volume (as illustrated in Fig. 3), where each voxel serves as a node. The feature of each node corresponds to its gray level, an integer between 0 and 255, while the label represents its class, or material phase, ranging from 0 to 5. Each node is connected to its six nearest neighbors, based on the assumption that, for microstructure reconstruction, spatial proximity is a meaningful criterion for establishing links.

Common GNN architectures follow an *encoder-decoder* structure. The encoder is formed by a series of graph convolutional layers (GCLs), while the decoder is composed of linear layers interspersed with nonlinear activation functions. Unlike the cross-correlation-based convolutions used in CNNs for grid-like data [27], GCLs gather information through a process

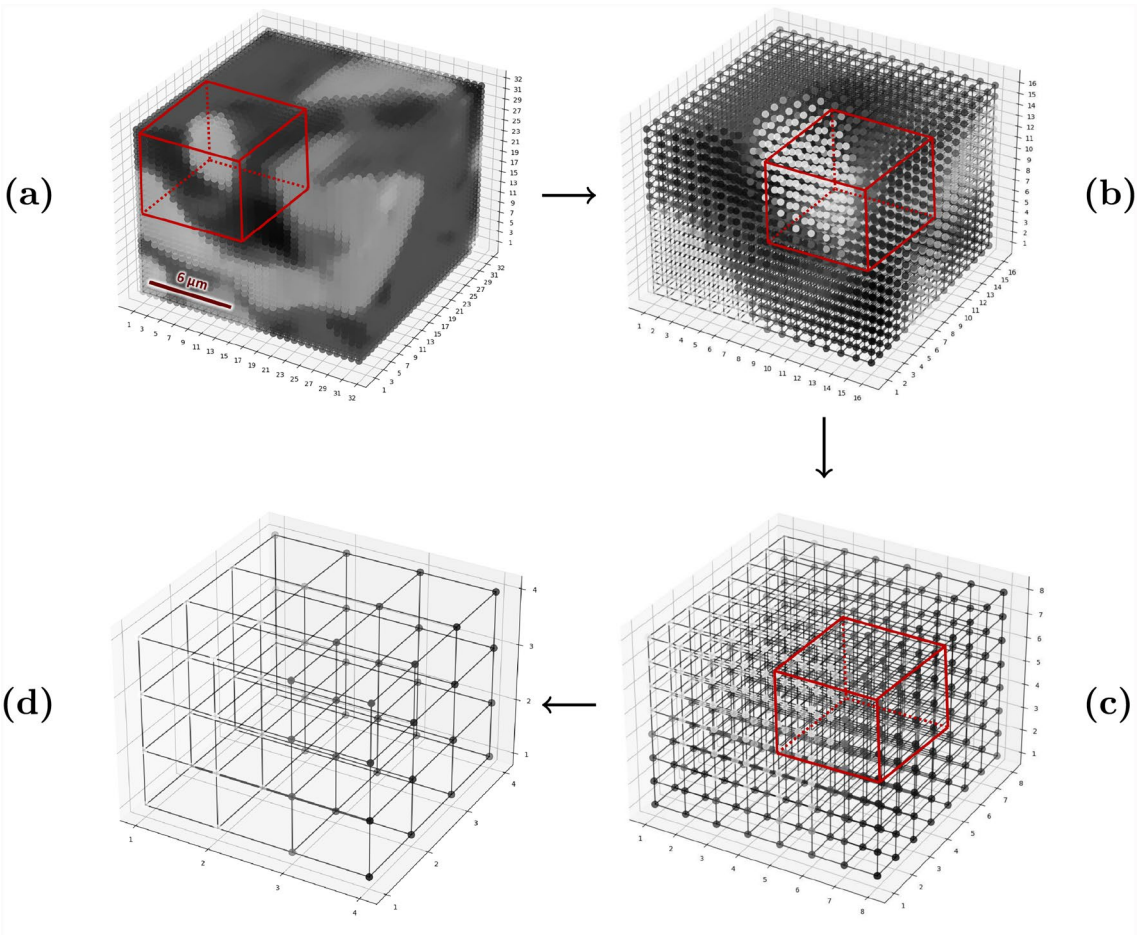


**Figure 3** Successive zoom on the underlying graph structure. From **a** to **d** the side of the sub-volume goes from 32 to 16, from 16 to 8, and from 8 to 4.

known as *Message Passing* [28]. In this process, for each node $v$, information from neighboring nodes $\mathcal{N}(v)$ is aggregated using permutation-equivariant operations (such as sum, mean, max, or min) and then updated by incorporating the node's own features. Unlike CNNs, where each neuron in the receptive field is multiplied by a different weight in the kernel, in GCLs the same matrix of weights is applied to each node in the neighborhood of a given vertex. This generally makes CNNs more expressive for visual tasks. However, several attempts have been made to enhance the expressiveness of GNNs. Among them, the introduction of an *attention mechanism* in graph attention (GAT) layers [29] and the usage of a multilayer perceptron in graph isomorphism network (GIN) layers [30].

More specifically, the implementation of the GCLs in our GNN architectures contains:

- **GIN layer** [30]. A graph isomorphism network (GIN) layer aggregates information from a node's neighbors and updates its representation in a way that is theoretically as powerful as the *Weisfeiler-Lehman* (WL) test [31, 32], a traditional method for checking graph isomorphism. GIN uses a sum aggregation function instead of mean, max, or other common methods. This choice is crucial as it enables GIN layers to capture subtle details of the graph structure that may be overlooked with other aggregation methods. Once the node aggregates the features of its neighbors, the result is passed through a neural network (usually a multilayer perceptron, or MLP) to update the node's representation. While GIN primarily uses sum aggregation, the MLP allows for additional nonlinear processing of this aggregated information. This increases the expressiveness of the network compared to a simple linear transformation, enhancing the network's ability to learn rich, complex representations of graph data. We employ Pytorch GINConv [30], whose update can be expressed as:

$$\mathbf{h}_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon)\cdot \mathbf{h}_v^{(k-1)} + \sum_{w\in\mathcal{N}(v)} \mathbf{h}_w^{(k-1)}\right) \quad (1)$$

  Where $\mathbf{h}_v^{(k)}$ is the updated feature for node $v$ at layer $k$, $\mathbf{h}_v^{(k-1)}$ is the node's feature from the previous layer, $\mathbf{h}_w^{(k-1)}$ is the neighboring node's feature from the previous layer, and $\mathcal{N}(v)$ represents the neighbors of node $v$. The learnable parameter $\epsilon$ controls the balance between the node's own feature and its neighbors, allowing flexibility in how much influence a node's own features have in relation to its neighbors during the update process. As stated above, after the neighboring features are aggregated through the sum function, every GIN layer $k$ employs a multilayer perceptron $\mathrm{MLP}^{(k)}$ to update the node's representation.
- **GraphSage layer** [33]. This approach is akin to GCN [22], but it offers a more flexible framework for customizing the convolution steps. Specifically, we utilize the SAGEConv implementation in PyTorch [33], which employs the sum operation for both aggregation and concatenation:

$$\mathbf{h}_v^{(k)} = W_1\mathbf{h}_v^{(k-1)} + W_2\frac{1}{|\mathcal{N}(v)|}\sum_{w\in\mathcal{N}(v)} \mathbf{h}_w^{(k-1)} \quad (2)$$

  In this equation, $W_1$ and $W_2$ represent the weights associated with the aggregation and concatenation processes. The resulting feature vector $\mathbf{h}_v^{(k)}$ is subsequently passed through a fully connected layer and normalized after being processed by a non-linear activation function.

Through consecutive graph convolutional layers, we iteratively update the embedding of a node by aggregating embeddings of its neighbors. After k iterations of aggregation, a node's embedding captures the information within its k-hop network neighborhood. It is intuitive that, since each node aggregates information from the neighbors, a set of neighboring nodes will share a similar embedding. Starting from the hypothesis that neighboring nodes tend to share the same label, this will ultimately ease the classification task. Even if the message passing formalism tends to flatten the distance between neighboring nodes (*smoothing*) to facilitate the classification later, when the architecture becomes deep, nodes will have access to information from nodes that are far away and may not be similar to them. There is then the risk of making all node embeddings look similar, thus decreasing the model ability to classify unlabeled nodes. The search for a model that is more expressive and aware of the graph structure (by adding more layers so that nodes can have a large receptive field) could yield a model that disperses the information and cancels out the differences between the nodes. This common phenomenon is called *over-smoothing* [19] and we take it into account when enlarging our GNN architecture (see Sec. 4.2).

## The ViG architecture

During our initial attempt at segmenting the Al–Si MMC, we trained a simple GNN architecture on the non-augmented synthetic dataset as a proof of concept for applying GNNs to this task (see [21]). Even if this model correctly understands the semantic of microstructures, the performance gets considerably worse when it is trained on the augmented dataset. This is due to the fact that the model is too small, and it is not able to learn equally well when the range of gray levels is enlarged. This leads us to build a second, larger, architecture, to train on the synthetic augmented dataset and further improve the segmentation of experimental data.

The encoding of the simple GNN architecture implemented in [21] is obtained by a sequence of three GCLs, each followed by a non-linearity: GAT (Graph attention networks) [29], GraphSage [33], and GCN (Graph convolutional network) [22]. The decoder consists of two linear layers separated by a nonlinear activation function. The first GCL is a GAT layer, so that we can obtain meaningful attention coefficients by looking directly at the input features before aggregation.

To build the second enlarged GNN architecture (diagram in Fig. 4), we take inspiration from [20] and adapt vision graph neural network (ViG) blocks. ViG blocks proved successful in segmenting 2D images for well-known datasets, as in [34] where they are employed to segment different tissue regions in histopathology images for diagnosis and prognosis of cancer. A ViG block consists of two main parts: the Grapher and the FFN (Feed forward network) modules. The Grapher module contains the graph convolutional layers, while the FFN module is a simple multilayer perceptron with two fully connected layers. The FFN module is introduced to further encourage the feature transformation and alleviate the typical phenomenon of over-smoothing in GNNs [19]. Indeed, introducing a FFN module after the graph convolutions interrupts the message passing flow and projects the node embeddings into a larger space, thus avoiding over-smoothing when enlarging the architecture. The final architecture is built up by concatenating different ViG blocks.

In particular, our enlarged GNN model (diagram in Fig. 4) comprises two adapted ViG blocks, which constitute the encoder of the architecture. The Grapher module contains one initial 3DCNN layer, useful

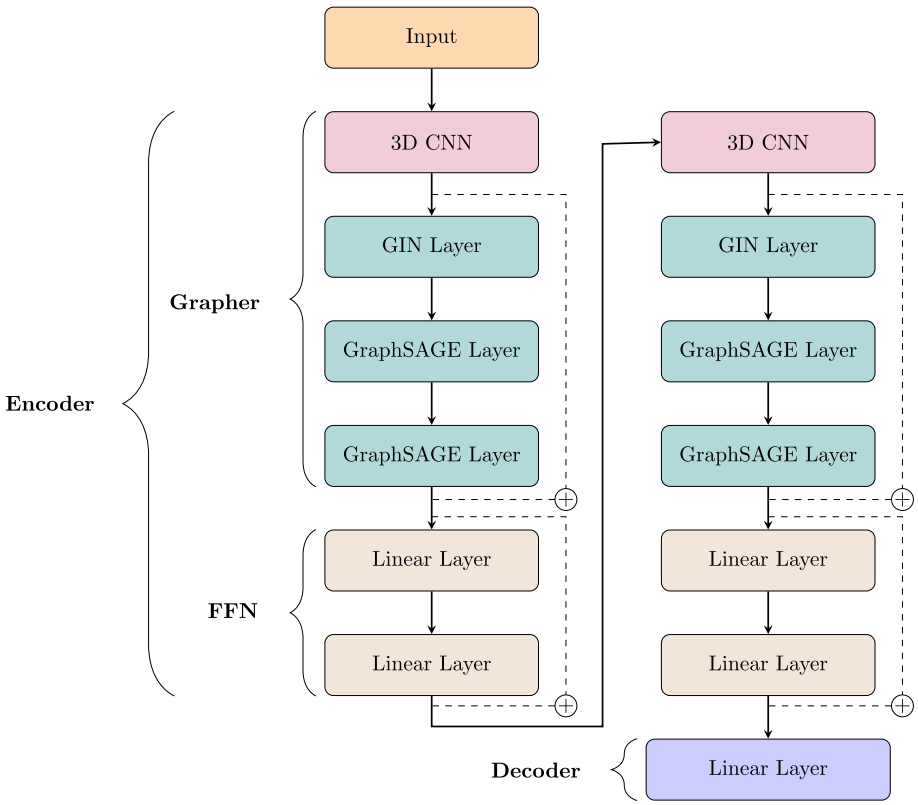


**Figure 4** Diagram of the ViG architecture. The $64 \times 64 \times 64$ sub-volume goes through an encoder composed of 2 ViG blocks with a Grapher and a FFN module each. The Grapher comprises a first 3D CNN and three following GCLs; the FFN simply consists of two feed-forward layers. The final decoder is a feed-forward layer. Skip connections (dashed lines) sum the 3D CNN output to the Grapher output and the result of this first sum to the FFN output.

to extract richer and more significant features for the nodes, and a succession of three GCLs, each followed by a non-linearity: one first GIN (Graph isomorphism network) layer [30] with a two-layer MLP, and two consecutive GraphSage layers [33]. Batch normalization is applied after the 3DCNN and after each GCL layer in the Grapher module. On the other side, the FFN module is composed of two linear layers separated by a nonlinear activation function. We employ SiLU activation function [35] throughout. The decoder of the model is a linear layer. Between the different modules, skip connections are inserted to help the convergence of the model. We stress that the size and the topology of the graph is not changing throughout the architecture. On the other hand, the input gray level is initially embedded in a space of dimension 50 and the embedding gradually increases by multiples of 50 until the final dense layer projects it on the number of classes.

Differently from the simple GNN [21], all GNN layers in ViG architecture can be easily implemented as a sparse matrix multiplication. This is possible for GNNs that do not make use of the central node features or of multi-dimensional edge features when aggregating messages. For example, the message passing update in a GIN layer 1 can be rewritten as:

$$\mathbf{H}^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon)\cdot\mathbf{H}^{(k-1)} + \mathbf{A}\mathbf{H}^{(k-1)}\right) \tag{3}$$

Where $\mathbf{A}$ denotes the sparse adjacency matrix of the graph, while $\mathbf{H}^{(k-1)}$ is a matrix of dimension $N \times d$ grouping the nodes' features from the previous layer, with $N$ the total number of nodes and $d$ the embedding dimension of the previous layer. This reformulation of the message passing scheme allows to parallelize the operations on the number of nodes, resulting in a lower memory footprint and a faster execution time. This memory-efficient aggregation is directly available in Pytorch Geometric for GIN [30] and GraphSage [33] layers, while it has not been implemented yet for GAT [29] and GCN [22] layers. We will compare the execution time and the RAM occupation of the two GNN architectures in Sec. 5.

## Training, testing, and fine-tuning

We will now provide a brief overview of our training process. First, we divide our synthetic dataset into seven volumes for training and validation, with one volume reserved for testing. To ensure a fair comparison of performances, we evaluate the model on the same volume used in [4]. Each volume is subsequently divided into overlapping sub-volumes of size $64 \times 64 \times 64$ voxel, and the graph structure is constructed for each sub-volume. The seven volumes are further partitioned, allocating 80% for training and 20% for validation, following a random shuffling of the extracted $64 \times 64 \times 64$ sub-volumes. This partitioning is summarized in Table 2.

We optimize the model using the Adam optimizer [36] with an initial learning rate of 0.001, an exponential decay rate of 0.96, and a weight decay regularization of 0.001. The model is trained for 50 epochs, with a batch size set to 64 graphs. In particular, we employ an early stopping criterion: We save the weights of the model minimizing the loss on the validation set with a patience of 20 epochs. We utilize cross-entropy loss, and to address class imbalance, we adjust the contribution of each class by the inverse of its average occurrence across the eight synthetic volumes (see Table 1). Furthermore, since voids constitute on average only 0.13% of the voxels in a volume, and they are not annotated in the experimental slices, we set their contribution to the total loss to zero.

After training the model, we assess its accuracy by collecting the probabilities for all $64 \times 64 \times 64$ sub-volumes extracted from the test volume. We then reconstruct six $512 \times 512 \times 512$ voxel probability volumes, one for each class. To effectively segment the overlapping regions between sub-volumes, we reconstruct the probability volume by summing the probabilities of overlapping voxels. The final segmented volume of $512 \times 512 \times 512$ voxel is created by assigning to each voxel the class corresponding to the highest summed probability from the six probability volumes.

Since the model is trained on 3D data, fine-tuning on experimental data has to be done as well on 3D data to be effective. To cut the timing of manually-labeling the voxels, we first use the trained model to segment a sub-volume of dimensions $128 \times 128 \times 128$ voxel from one of the experimental volumes in the dataset. Then, we manually correct this segmentation and we fine-tune the model on it. Again, the experimental volume is divided into overlapping sub-volumes of size $64 \times 64 \times 64$ voxel, and the graph structure is constructed for each sub-volume. The model is further trained for 50 epochs on

**Table 2** Dataset split ratio between training, validation, and test

| Training | Validation | Test |
|---|---|---|
| 70% | 17% | 13% |

these experimental sub-volumes, and the batch size is now set to four graphs because of the reduced size of the dataset. Since the dimension of the dataset for fine-tuning is small, this time we do not create a validation set and we just save the weights corresponding to the last epoch of training.

Also, since the model is originally trained on synthetic data, we choose to fine-tune it on experimental data conditioned with BAM SynthCOND [25]. That is because this conditioning method constraints experimental data to be entropically analogous to synthetic data, and this results in an increased performance with respect to fine-tuning on experimental data conditioned by a non-local means filter [24].

## Results and discussion

To evaluate the accuracy on the test set, we conduct 10 training runs with a fixed random seed and calculate the standard deviation corresponding to a 95% confidence interval. This approach is common practice in the literature (see, for example, [37]).

We assess accuracy using the Dice score, a.k.a. $F1$ score, a widely used metric for image segmentation (see, for example, [2, 4, 34]). The Dice score rewards correctly segmented voxels and penalizes incorrect ones. It is defined in terms of true-positive, false-positive, and false-negative voxels as follows:

$$\text{Dice score} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{4}$$

In Table 3, we present the Dice scores obtained from evaluating the two architectures on the synthetic test dataset and on the four manually-labeled slices of experimental data (conditioned by a non-local means filter [24], and conditioned by BAM SynthCOND [25]). The Dice score for the experimental data is calculated by first averaging the Dice scores of the 10 trained models for each slice and then averaging the resulting Dice scores across the four slices. Since voids are not annotated in the manually-labeled experimental slices, we focus on the segmentation of the remaining five classes. We stress that, even if the model is trained to segment 3D structures, we evaluate the experimental performance on 2D slices. This is due to the fact that manually-labeling the voxels is a time-consuming operation and it is faster to manually-label a 2D slice than a 3D sub-volume. Computing the Dice score in 3D is generally more appropriate for evaluating the segmentation of 3D structures, since it ensures a comprehensive assessment of the segmentation quality by capturing spatial consistency. However, 2D Dice scores constitute a first relevant measure for the quality of the segmentation.

As expected, the model performs worse on experimental data than on synthetic data since it was trained on low-resemblance synthetic data (see also [4]). This discrepancy arises because our synthesized data do

**Table 3** Dice score on synthetic testing volume and experimental slices. The values and their errors represent the mean and standard deviation over 10 training runs. The performance is compared to the single U-Net model proposed in [4]

| | $Al_2O_3$ *Fibers* (%) | IMs (%) | Eutectic Si (%) | SiC Particles (%) | Al Matrix (%) |
|---|---|---|---|---|---|
| *Synthetic* | | | | | |
| Simple GNN | 91.5 ± 0.6 | 90.2 ± 1.2 | 88.4 ± 0.5 | 89.8 ± 0.6 | 95.8 ± 0.1 |
| ViG | 91.1 ± 1.0 | 98.9 ± 0.1 | 93.0 ± 0.9 | 96.4 ± 0.3 | 98.1 ± 0.2 |
| Single U-Net | 97.2 ± 0.5 | 98.1 ± 1.5 | 93.3 ± 0.2 | 96.4 ± 0.7 | 98.5 ± 0.1 |
| *NLM Experimental* | | | | | |
| Simple GNN | 31.1 ± 0.8 | 61.7 ± 0.8 | 62.6 ± 1.1 | 40.3 ± 2.0 | 75.5 ± 1.0 |
| ViG | 33.1 ± 2.7 | 39.5 ± 3.0 | 68.4 ± 1.0 | 61.4 ± 1.1 | 83.4 ± 0.6 |
| Single U-Net | 55.1 ± 0.4 | 53.4 ± 1.8 | 68.6 ± 0.2 | 67.2 ± 0.7 | 85.7 ± 0.1 |
| *SynthCond Experimental* | | | | | |
| Simple GNN | 32.7 ± 0.9 | 64.8 ± 2.1 | 62.9 ± 1.0 | 59.1 ± 1.3 | 77.5 ± 1.0 |
| ViG | 40.5 ± 2.0 | 61.0 ± 2.2 | 72.2 ± 1.1 | 65.0 ± 0.9 | 85.7 ± 0.5 |
| Single U-Net | 57.8 ± 3.0 | 68.5 ± 1.9 | 75.1 ± 1.8 | 70.5 ± 1.5 | 87.4 ± 0.6 |
| *Fine-Tuned (SynthCond)* | | | | | |
| ViG | 47.8 ± 1.0 | 69.9 ± 1.3 | 73.0 ± 1.0 | 67.8 ± 1.1 | 85.6 ± 0.5 |
| Single U-Net | 45.5 ± 2.7 | 68.0 ± 2.3 | 65.9 ± 0.8 | 65.5 ± 2.8 | 82.0 ± 0.5 |

not fully capture the complexity of the experimental data, both in terms of grayscale distributions and structural details. One solution to this problem could be to generate better synthetic data through deep learning algorithms such as GANs and diffusion models ( [38–40]).

In Fig. 5, we give an example of segmentation for two of the manually-labeled experimental slices conditioned with BAM SynthCOND [25], while in Fig. 6, we visualize the segmentation in 3D on an experimental sub-volume. We notice that despite the poor accuracy (see Table 3), the GNN models correctly capture the geometry of the objects in the slice, thus giving a correct semantic understanding of the image. Also, since BAM SynthCOND constraints the conditioned experimental slices to be entropically analogous to the synthetic data [26], the segmentation is enhanced compared to the one obtained when conditioning experimental data with a Non-local means filter [24] (see Table 3).

From the Dice scores in Table 3 and the visualization of the segmentation in Fig. 5, we notice that our enlarged ViG architecture, trained on the augmented dataset, actually manages to improve the overall segmentation of the experimental data with respect to our previous model in [21]. More specifically, the Dice scores by ViG architecture improve over the previous ones, apart from Intermetallics (IMs), whose accuracy decreases. The reduction of the Dice score for intermetallics is particularly high when segmenting

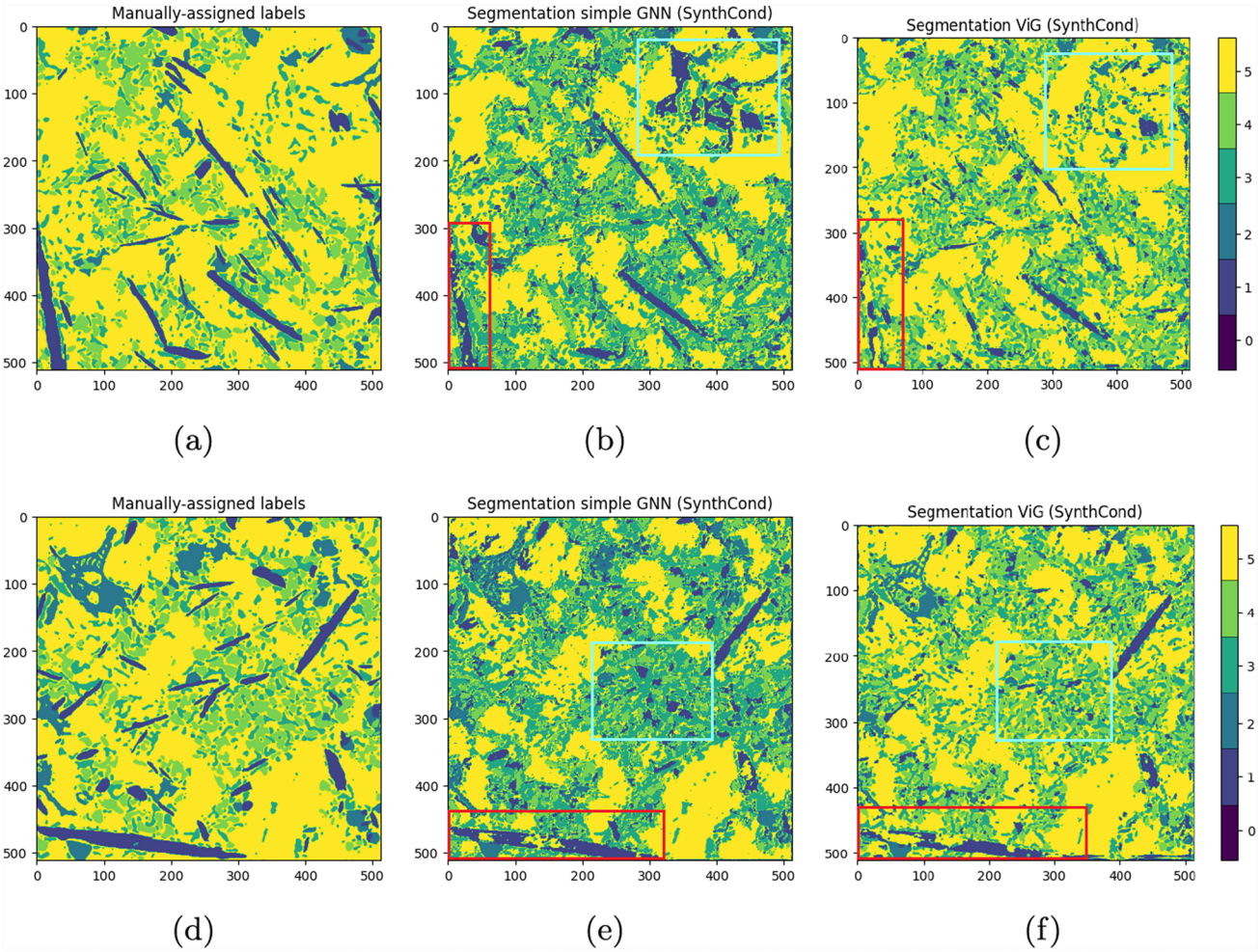


**Figure 5** Resulting segmentation of two experimental slices conditioned with BAM SynthCOND [25]. On the left, the manually-labeled ground truths (**a**,**d**), in the middle the segmentation by the simple GNN model (**b**,**c**), on the right the one by ViG model (**c**,**f**). The color bar shows the label for each class, from 0 to 5: Voids, Al2O3 fibers, Intermetallics (IMs), eutectic Si, SiC particles, and Al matrix.

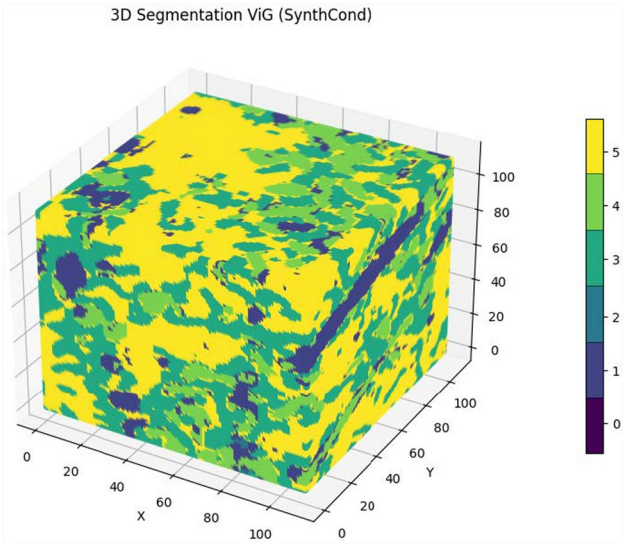


**Figure 6** 3D visualization of the segmentation on an experimental sub-volume conditioned with BAM SynthCOND [25]. The color bar shows the label for each class, from 0 to 5: Voids, Al2O3 fibers, Intermetallics (IMs), eutectic Si, SiC particles, and Al matrix.

experimental data conditioned with a non-local means filter [24], but it gets better when evaluating the model on experimental data conditioned with BAM SynthCOND [25].

By inspecting Fig. 5 more closely, we notice that ViG model is able to correctly distinguish between eutectic Si and SiC particles, while the previous GNN model was generally segmenting the Al matrix portion between these two labels as eutectic Si. This improvement by ViG model is reflected in the enhanced Dice score for eutectic Si, SiC particles, and Al matrix in Table 3. In Fig. 5, we also notice that the simple GNN model better recognize thicker fibers, but it confuses SiC particles and portions of the Al matrix as fibers as well, and the Dice score for fibers is overall enhanced by ViG model. In Fig. 5, we highlight the differences in the segmentation of thicker fibers with a red rectangle, while the aereas where the GNN model confuses fibers with SiC particles and Al matrix are highlighted in light blue.

In Fig. 7, we report the confusion matrix of the two GNN models when segmenting one experimental slice conditioned with a non-local means filter [24] and with BAM SynthCOND [25]. The number of true-positives on the diagonal of the confusion matrix generally reflects the behavior of Dice scores in Table 3. We obtain a similar confusion matrix for each experimental slice (not reported for the sake of brevity). In particular, the segmentation by ViG model splits almost in half the number of voxels belonging to eutectic Si (label 4) and classified as Al matrix (label 5) with respect to the simpler GNN. Also, the number of voxels belonging to fibers (label 1) and classified as SiC particles (label 4) and Al matrix (label 5) is largely reduced.

We next compare our GNN architectures with the single U-Net architecture developed in [4], when trained on the augmented dataset. The corresponding Dice scores are reported in Table 3 for both synthetic and experimental data. From Table 3, we see a similar drop from synthetic to experimental data.

So far, our ViG architecture is generally slightly outperformed by the single U-Net in [4], while it is largely outperformed in the segmentation of Fibers. This discrepancy in segmenting larger 3D structures may be due to constructing the graph by linking each node with only the six nearest neighbors, which might not provide sufficient connectivity for effective fiber segmentation. However, the structure of the architecture could play a role as well, beyond the type of neural network employed. To this end, to properly compare the usage of CNNs and GNNs for this task, we plan to implement a Graph U-Net [41] in the near future.

To improve the performance on experimental samples, we use the Single U-Net to segment a $128 \times 128 \times 128$ voxel sub-volume from one of the experimental volumes and, after manually correcting some of the errors, fine-tune the single U-Net and our ViG model on it. In the case of ViG architecture, fine-tuning helps enhancing the segmentation on fibers and intermetallics, while the Dice scores for the other classes stay the same (see Table 3). On the contrary, fine-tuning does not increase the performance for the single U-Net, and the Dice scores for fibers, intermetallics, and SiC particles are largely reduced. One possible explanation is that GNNs can generalize more easily than CNNs on a small portion of experimental data. To further investigate this behavior, we plan to enlarge the dataset for fine-tuning and compare again the performance of the two architectures.

In any case, the main advantage of our GNN architecture is that it largely reduces the number of trainable parameters. In Table 4, we report the number of trainable parameters of our two GNN architectures and the single U-Net in [4]. Although the enlarged ViG architecture has 6 times the number of trainable parameters compared to our previous simple GNN

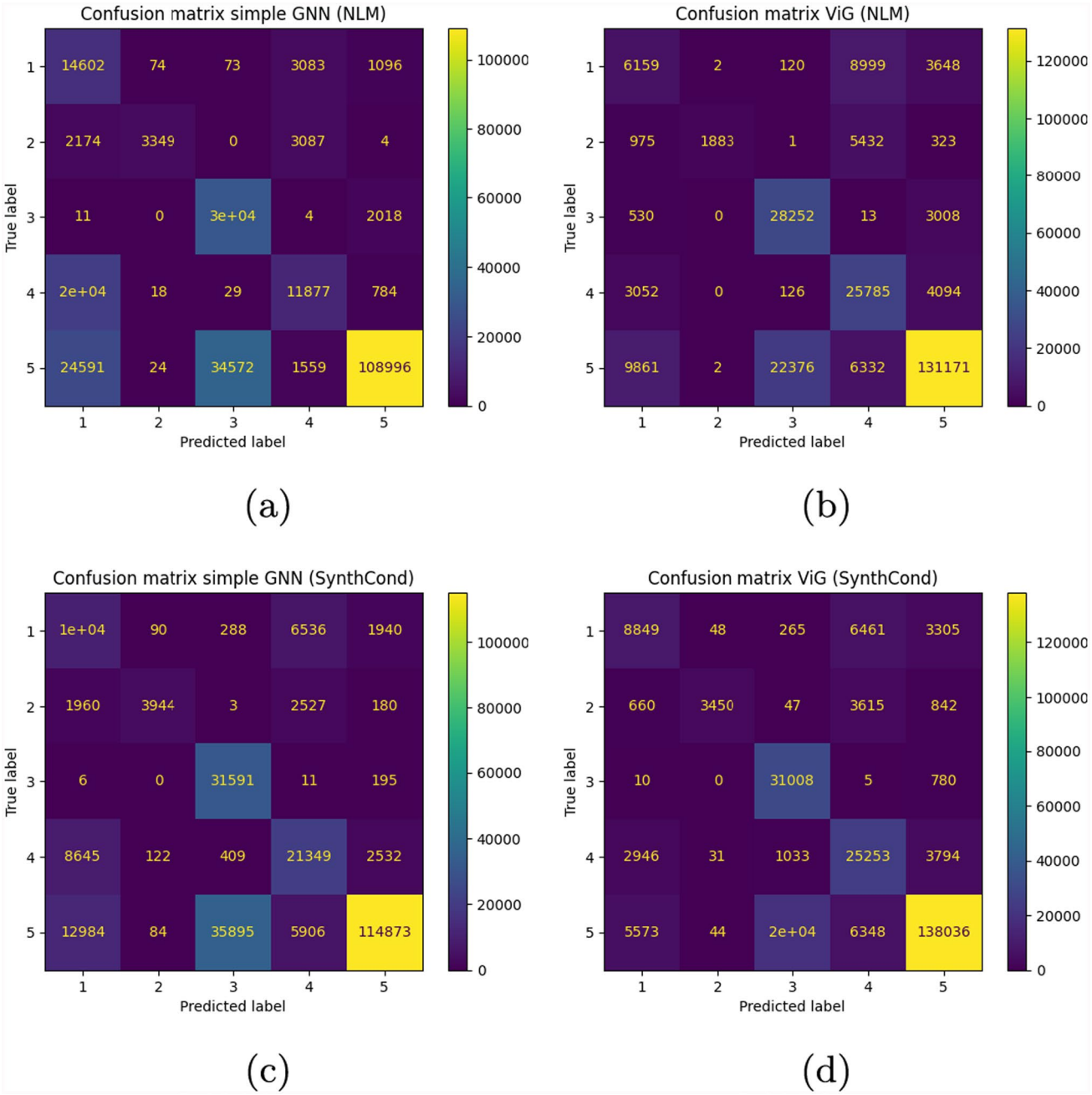


**Figure 7** Confusion matrices resulting from the segmentation of one experimental slice conditioned with a non-local means filter [24] (top: (**a**), (**b**)) and with BAM SynthCOND [25] (bottom: (**c**), (**d**)). The X and Y-axis indicate the class, from 1 to 5: Al2O3 fibers, intermetallics (IMs), eutectic Si, SiC particles, and Al matrix. We discard voids since they are not annotated in the manually-labeled experimental slice.

**Table 4** Comparison of the three architectures under analysis in terms of number of trainable parameters, RAM usage, and time during training and test

| Architecture | Trainable parameters | Memory (Train) | Time (Train) | Memory (Test) | Time (Test) |
|---|---|---|---|---|---|
| Simple GNN | 61,406 | 4,155 MB | 19 min | 8,924 MB | 12 min |
| ViG | 401,006 | 6,570 MB | 30 min | 10,362 MB | 15 min |
| Single U-Net | 667,446 | 6,542 MB | 8 min | 10,333 MB | 10 min |

[21], it is still almost half the size of the single U-Net in [4].

In Table 4, we also compare the models in terms of RAM usage and time execution for both training and test. While training on the augmented dataset, we compute the RAM and the time required for one epoch. During inference, we measure RAM and time when segmenting one experimental volume. RAM usage is comparable between the ViG model and the single U-Net, while the single U-Net is faster in training. On the other side, even though ViG model has 6 times the number of trainable parameters with respect to the simple GNN, the RAM usage and training time are not even twice the size. We trace back this behavior to the fact that, differently from the simple GNN, all GNN layers in ViG model are implemented as a sparse matrix multiplication and this reduces the memory footprint and speeds up the execution time. Then, we believe that, once the libraries employed for GNNs will reach the same optimization level as the ones implementing usual DCNNs, the reduced number of trainable parameters will cut the computational costs both in training and testing.

## Conclusions

We implement a vision graph neural network (ViG) architecture to segment microstructures in XCT volumes of an AlSi12CuMgNi metal matrix composite, and we enhance the final segmentation by fine-tuning the model on experimental data. Our GNN architecture gives a correct semantic understanding of microstructures, despite the highly reduced number of trainable parameters with respect to state-of-the-art U-Nets. As future work, we plan to improve the segmentation on experimental slices by generating highly resembling synthetic data through deep learning algorithms.

## Acknowledgements

The research ML, RF, FZ was supported by Gnsaga-Indam, by COST Action CaLISTA CA21109, by HORIZON-MSCA-2022-SE-01-101086123 CaLIGOLA and PNRR MNESYS. Funding for ML from BAM Project QI Digital is acknowledged. FF acknowledges UNIMORE FAR-DIP DSCG 2023.

## Author contributions

ML, AT, and FZ were contributed conceptualization, data curation, formal analysis, investigation, methodology, and writing–original draft. FF, RF, AT, and GB: supervision and writing–review and editing.

## Funding

Open Access funding enabled and organized by Projekt DEAL.

## Data availability

Samples of the synthetic and experimental datasets discussed and classified in the paper are available together with the full code at: https://github.com/michelalapenna/GNN-for-segmentation-of-XCT

## Declarations

**Conflict of interest** The authors declare that no conflict of interest may have influenced the work in this paper.

**Ethical approval** Not applicable.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References


[1] Choudhary K et al (2021) Recent advances and applications of deep learning methods in materials science. npj Comput Mater 8:1–26

[2] Strohmann T et al (2019) Semantic segmentation of synchrotron tomography of multiphase al-si alloys using a convolutional neural network with a pixel-wise weighted loss function. Sci Rep 1–10

[3] Evsevleev S, Paciornik S, Bruno G (2020) Advanced deep learning-based 3d microstructural characterization of multiphase metal matrix composites. Adv Eng Mater 22:1–6

[4] Tsamos A, Evsevleev S, Fioresi R, Faglioni F, Bruno G (2023) Synthetic data generation for automatic segmentation of x-ray computed tomography reconstructions of complex microstructures. J Imaging 9:1–23

[5] Wong VWH, Ferguson M, Law KH, Lee YTT, Witherell P (2021) Automatic volumetric segmentation of additive manufacturing defects with 3d u-net. arXiv:2101.08993

[6] Du W et al (2020) Automated detection of defects with low semantic information in x-ray images based on deep learning. J Intell Manuf 32:141–156

[7] Fotos G, Campbell A, Murray P, Yakushina E (2023) Deep learning enhanced watershed for microstructural analysis using a boundary class semantic segmentation. J Mater Sci 58:1–21

[8] Sarvamangala DR, Kulkarni RV (2021) Convolutional neural networks in medical image understanding: a survey. Evol Intell 15:1–22

[9] Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2324

[10] Krizhevsky A, Sutskever I, Hinton GE, Pereira F, Burges C, Bottou L, Weinberger K (eds) (2012) Imagenet classification with deep convolutional neural networks. (eds Pereira, F., Burges, C., Bottou, L. & Weinberger, K.) Advances in Neural Information Processing Systems, Vol. 25 (Curran Associates, Inc., 2012), pp 1–9

[11] He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. 2016 IEEE conference on computer vision and pattern recognition (CVPR) 770–778

[12] Carion N et al (2020) End-to-end object detection with transformers. Computer Vision-ECCV 2020: 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I 213-229 . https://doi.org/10.1007/978-3-030-58452-8_13

[13] Dosovitskiy A et al (2020) An image is worth 16x16 words: Transformers for image recognition at scale. pp 1–22. arXiv:2010.11929. https://api.semanticscholar.org/CorpusID:225039882

[14] Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond euclidean data. IEEE Signal Proc Mag 34:18–42

[15] Bronstein MM, Bruna J, Cohen T, Velivckovic P (2021) Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. pp 1–160. arXiv:abs/2104.13478. https://api.semanticscholar.org/CorpusID:233423603

[16] Landrieu L, Simonovsky M (2017) Large-scale point cloud semantic segmentation with superpoint graphs. 2018 IEEE/CVF conference on computer vision and pattern recognition 4558–4567. https://api.semanticscholar.org/CorpusID:4396837

[17] Wang Y et al (2019) Dynamic graph cnn for learning on point clouds. ACM Trans Graph 38:1–13

[18] Yang Y, Qiu J, Song M, Tao D, Wang X (2020) Distilling knowledge from graph convolutional networks. 2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR) 7072–7081. https://api.semanticscholar.org/CorpusID:214713848

[19] Rusch TK, Bronstein MM, Mishra S (2023) A survey on oversmoothing in graph neural networks. pp 1–10. arXiv:abs/2303.10993. https://api.semanticscholar.org/CorpusID:257632346

[20] Han K, Wang Y, Guo J, Tang Y, Wu E (2022) Vision gnn: an image is worth graph of nodes. In: Proceedings of the 36th international conference on neural information processing systems. pp 1–15

[21] Lapenna M et al (2024) Geometric deep learning for enhanced quantitative analysis of microstructures in x-ray computed tomography data. Discover Appl Sci 6:1–9

[22] Kipf T, Welling M (2016) Semi-supervised classification with graph convolutional networks. pp 1–14. arXiv:abs/1609.02907. https://api.semanticscholar.org/CorpusID:31442186

[23] Zachary WW (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33:452–473

[24] Buades A, Coll B, Morel J.-M (2005) A non-local algorithm for image denoising. In: Computer society conference on computer vision and pattern recognition, vol 2. pp 60–65

[25] Tsamos A, Evsevleev S, Bruno G (2023) Noise and blur removal from corrupted x-ray computed tomography scans: A multilevel and multiscale deep convolutional framework approach with synthetic training data (bam synthcond). Tomography Mater Struct 1–16

[26] Tsamos A, Evsevleev S, Fioresi R, Faglioni F, Bruno G (2023) A complete strategy to achieve high precision

automatic segmentation of challenging experimental x-ray computed tomography data using low-resemblance synthetic training data. Adv Eng Mater 26:1–9

[27] Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, pp 330–371. http://www.deeplearningbook.org

[28] Velickovic P (2022) Message passing all the way up. pp 1–10. arXiv:abs/2202.11097. https://api.semanticscholar.org/CorpusID:247026068

[29] Veličković P et al (2018) Graph attention networks. In: International conference on learning representations. pp 1–12

[30] Keyulu Xu J. L, Weihua Hu, Jegelka S (2019) How powerful are graph neural networks? In: International conference on learning representations. pp 1–17. https://openreview.net/forum?id=ryGs6iA5Km

[31] Weisfeiler B, Leman A (1968) The reduction of a graph to a canonical form and the algebra arising during this reduction. Nauchno-Technicheskaya Informatsia 9:1–11

[32] Shervashidze Nino VLEJ, Schweitzer P, Kurt M, Borgwardt KM (2011) Weisfeiler-lehman graph kernels. J Mach Learn Res 12:1–23

[33] Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st international conference on neural information processing systems. pp 1025–1035

[34] He P, Qu A, Xiao S, Ding M (2023) A gnn-based network for tissue semantic segmentation in histopathology image. J Phys: Conf Ser 2504:012047

[35] Stefan Elfwing EU, Doya K (2017) Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. pp 1–18. arxiv.org/abs/1702.03118

[36] Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: 3rd International conference on learning representations. ICLR, pp 1–15

[37] Genç A, Kovarik L, Fraser HL (2022) A deep learning approach for semantic segmentation of unbalanced data in electron tomography of catalytic materials. Sci Rep 12:1–12

[38] Thambawita V et al (2022) Singan-seg: Synthetic training data generation for medical image segmentation. PLOS ONE 17:1–24

[39] Nguyen Q, Vu T, Tran A, Nguyen K (2023) Diffusion-based synthetic data generation for pixel-level semantic segmentation. Adv Neural Inf Proc Syst 36:1–21

[40] Corvi R, Cozzolino D, Poggi G, Nagano K, Verdoliva L (2023) Intriguing properties of synthetic images: from generative adversarial networks to diffusion models. In: 2023 IEEE/CVF Conference on computer vision and pattern recognition workshops (CVPRW). pp 973–982. https://api.semanticscholar.org/CorpusID:258107869

[41] Gao H, Ji S (2022) Graph u-nets. IEEE Trans Pattern Anal Mach Intell 44:4948–4960

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.