Visual Analytics for Graph Deep Learning: Case Study Neuron Correspondence

Sophie Pichler
Supervised by: Dr. Astrid Berg

VRVis Vienna

Abstract

Many deep learning applications are based on graph data in order to explore relationships or to analyze structures. Labeling this data is expensive and often requires expert knowledge. For the application of graph clustering to neuron data, the SOTA method GraphDINO generates self-supervised graph embeddings combined with the downstream task of clustering these embeddings. We observe on a particularly challenging neuron dataset that this method does not lead to satisfying clustering results. Therefore we use the graph embeddings generated by GraphDINO as an initial starting point to improve the network and to guide the network training. To achieve this, we developed the visual analytics framework *NetDive*. The user can analyze the graph embeddings and label single neurons that are falsely clustered. This annotation information is then used to train a semi-supervised model. To this end, we developed a network architecture, titled *GraphPAWS*, that assembles components of GraphDINO and of the semi-supervised network architecture *PAWS*. The model training can be started from within the visual analytics application NetDive and the resulting graph embeddings are available in NetDive as soon as the retraining is completed. We demonstrate how we iteratively improve the model performance using NetDive and GraphPAWS and evaluate our model against the self-supervised SOTA for our dataset.

Keywords: Visual analytics, Graph embeddings, Graph transformer

1 Introduction

Many deep learning applications are based on graph data, e.g. in the fields of anomaly detection in networks, relationship analyses in social networks and in neuroscience. The use case investigated in this paper is to cluster unlabeled spatial graph data that represents unregistered drosophila melanogaster larval level 1 neurons to reproduce meaningful cell types, addressing the objective of neuroscience to understand the correlation between nerve cells, also named neurons, and behavior [17, 1, 23]. The cell type is an annotation that a neuron receives based on predefined features. Depending on the feature set, the cluster groups vary. Key features explored in the literature are morphology, genetic markers, the neuron position within the nervous system, connectivity and intrinsic electrophysiological signatures [8]. We aim to cluster the neurons solely based on their morphology and aim to find correlations to meaningful cell type assignments as the SOTA method GraphDINO does not produce satisfying results.
This use case embeds in the broader challenge of clustering data without initially having labels to train the deep learning model with a supervised objective function. Experts initially do not know what the network should learn, but want to be able to steer the training while gathering new knowledge about the resulting clusters. This leads us to design a pipeline that addresses these problems by steering the training of the graph network through incremental analysis of the generated embeddings and by incorporating the new knowledge back into the training process.

Contribution: We develop a semi-supervised network architecture *GraphPAWS* that adopts the graph encoder of the self-supervised deep learning architecture GraphDINO and the processing of support samples of the semi-supervised deep learning architecture PAWS. The support samples are sparse annotations for the input data and the count of support samples is variable. The resulting graph latent embeddings are visualized in a visual analytics (VA) web application we title *NetDive* that we developed to analyze the embeddings, to iteratively add new support samples if needed and to retrain a model with this new information. For the evaluation we use manually labeled neurons to compute the performance analytically and we combine this with visual inspection and comparative analysis enabled by the VA application. The GraphPAWS architecture is applicable to a broader range of graph clustering tasks and NetDive is partly data type agnostic and therefore applicable also to embeddings of other input data types.

2 Background and Related Work

Our work combines graph representation learning with VA to utilize the human in the loop to incrementally improve the network performance. We use the labels generated within the VA application as support samples for the semi-supervised network architecture, whilst we make use of *contrastive learning* to process the unlabeled input samples.

2.1 Contrastive Learning

Contrastive learning belongs to the most successful self-supervised learning methods. It involves the derivation of *supervisory signals* from the input data to guide the learning process [14].

Contrastive learning techniques optimize the model output by embedding the latent representations of variations of the same input sample close to each other, while increasing the distance between the embeddings of different input samples. The pairs of samples that are either attracted or repelled by each other are titled *positive pair* or *negative pair* respectively [14]. Phuc Le-Khac et al. [12] explain, that contrastive learning is not about learning from individual samples, but instead from *comparing* multiple samples. Positive pairs are generated by applying data augmentations to an input sample to get variants of input data that are considered *similar*.

The original non-augmented input sample is called *anchor view* and the augmented variant is referred to as the *positive view*. Negative pairs are generally formed by comparing the anchor view with all the other input samples. If contrastive learning is solely based on positive views [21], the model architecture needs to ensure that the latent representations do not collapse to a single node in the embedding space. This phenomenon is called *node collapsing*. Another force needs to increase the space between different samples.

A contrastive model includes an *encoder* that maps the input view $x \in X$ to a representation vector $v \in R^d$ and a *transform head* $h(v;\Phi_h) : V \rightarrow Z$, where Φ represents the model parameters, that are either used to aggregate features from multiple representation vectors or to reduce the dimensionality of a feature representation vector [12].

Prominent models that use contrastive learning to learn image representations are SimCLR [6], MoCo [10], BYOL [9], SwAV [4], PIRL [20] and DINO [5]. GraphCL [24] and GraphDINO [21] are examples for contrastive models that process graph data.

2.2 Visual Analytics for Latent Embeddings

There are in general two user groups in the field of visual analytics (VA) for deep learning [3]: model-driven users that compare model performances and data-driven users that study properties of the underlying data. A crucial criteria for VA applications is *global check and local check*.

Addressing this, a popular approach to compare embedding spaces is the comparison of local neighborhoods of individual objects in combination with a global comparison of the embeddings [11, 3]. The global embedding comparisons are typically implemented using scatter plots that are interlinked with detail views of selected objects [11]. Therefore dimensionality reduction algorithms are used to map the high-dimensional data in 2D or in 3D. The most common dimensionality reduction algorithms are PCA, tSNE, and UMAP [19]. Boggust et al. [3] discovered, that users prefer deterministic dimensionality reduction algorithms and that they distrust t-SNE and therefore use PCA dimensionality reduction as the default setting for the global projection. The visual analytics tool EmbComp [11] implements a binning feature for the scatter plots to manage the scale of big datasets. The scatter plots can be investigated using single object selection or multiple object selection using for example a rectangle selection tool [11, 15].

The investigation of local neighborhoods is built upon varying metrics. EmbComp visualizes point-wise comparison metrics and distribution comparison metrics. The metrics are visualized in bins which can be selected by the user to select the corresponding objects. The Embedding Comparator [3] visualizes metrics corresponding to the local neighborhoods of a selected datapoint with a histogram of scores, with color-encoding in the global embedding plots and with local neighborhood dominoes, i.e. multiple small visualizations. These small visualizations can be filtered and linked views enable the comparison between visualizations. The Embedding Comparator highlights datapoints with least and highest similarities to address the concern of users stating that they make object selections in an unprincipled way and might miss important correlations between the embedding spaces. Emblaze [19] states, that the Embedding Comparator lacks in finding relevant neighborhoods and addresses this issue in their application. The novel approach of Emblaze is comparison of embedding spaces using Star Trail augmentation. The trails connect the embeddings of the same object in different embedding spaces and the transition between the spaces can be animated using a slider. The connection lines, i.e. Star Trails, between the object embeddings quickly reveal datapoints that vary the most between multiple embedding spaces.

While partly being data type agnostic, the Embedding Comparator, EmbComp and Emblaze as well as many other lines of research regarding visual analytics for embeddings focus on NLP use cases.

In the field of graph embeddings the tools EmbeddingVis [13], CorGIE [15], GEMvis [7] and BiaScope [18] were developed, which are focused on node embedding.

CorGIE [15] encodes the graph nodes and trains a GNN to embed the nodes in the latent space. The user can interact with the node embeddings and select clusters of nodes using a rectangle selection tool. The selection leads to a topology space and feature space analysis. Regarding the topology space, the k-hop neighbors, i.e. the neighbors that are reachable by walking along a path with k topological hops, of the selected nodes are visualized within a visualization of the original graph. The user can evalu-

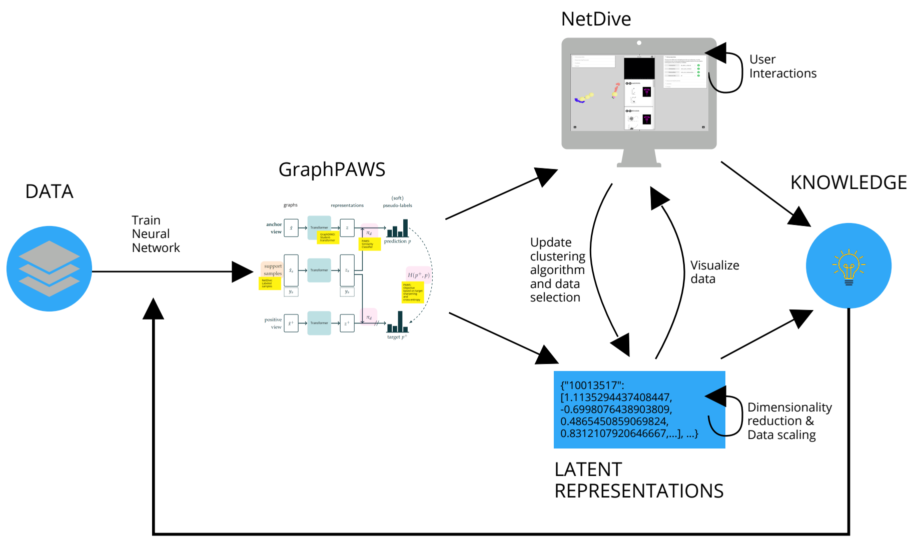


Figure 1: Our pipeline including our visual analytics tool NetDive and our model architecture GraphPAWS.

ate whether the node embeddings corresponds to the topological closeness. The feature space analysis panel shows histograms of feature value distributions of the selected nodes.

GEMvis [7] also interlinks a visualization of the original graph and the node embeddings. The selection of nodes can be applied regarding predefined node metrics. Chen et al. define 9 node metrics, including the node degree and node eccentricity and the node closeness. The metric values for each node are depicted in parallel coordinate plots. The user can interact with these plots to select the according nodes in the original graph and in the embedding space.

While the aforementioned applications EmbeddingVis, CorGIE, GEMvis and BiaScope are developed for node embeddings, we implement an application for whole graph embeddings. Advanced applications exist to leverage VA to compare and analyze the embeddings generated with deep learning models. We add the component of dynamically adding new labels to retrain the model while exploring the latent space that the input graphs are embedded in. We focus the usage of VA for artificial intelligence (AI) for the specific case, in which ground truth is difficult to gather and can only be provided to nudge the training in the right direction. We furthermore integrate detail views specific to the use case of exploring graph embeddings.

3 Methodology

Figure 1 depicts the pipeline that we set up to incrementally gain new knowledge in order to cluster graph data.

The preprocessed data serves as input data to train, validate and test the GraphPAWS model. GraphPAWS is discussed in Section 3.1. The model outputs latent representations of the input graphs. We store the latent representations on the filesystem and the visual analytics application NetDive, discussed in Section 3.2, accesses the data and provides the user with visualizations and user interactions to explore the latent embeddings and the associated neurons. This leads to new knowledge that the user can leverage to retrain the GraphPAWS model.

3.1 GraphPAWS

Our architecture GraphPAWS adopts the graph transformer components of GraphDINO and the semi-supervised architecture of PAWS.

3.1.1 GraphDINO: Self-Supervised Learning for Graph Data

The GraphDINO network [22] implements self-supervised contrastive learning based on transformer networks to find similarities between graphs based on the graph topology and spatial node information. GraphDINO is an adaptation of the DINO network for image data [5].

The GraphDINO model builds upon a student-teacher architecture that is used to generate latent representations of an input graph x. Both the teacher and the student process variations of x. The variations x_1 and x_2 are subsampled to a fixed number of nodes. Graph x_2 is passed to the teacher encoder and graph x_1 is augmented before being passed to the student encoder. The augmentations that are used are subsampling, rotation, node jittering, subgraph deletion, cumulative jittering and a random translation of the soma depth.

The student and the teacher network are identically initialized transformer networks that use the normalized Laplacian for the positional encoding. The outputs of the student and the teacher network are the latent representations z_1 and z_2 respectively. The multi-layer perceptron (MLP) implements a normalization layer and a linear layer to translate the latent representations z_1 and z_2 to p_1 and

p_2. The objective of the network is to decrease the loss that measures the similarity of p_1 and p_2, while not resulting in node collapsing.

GraphDINO uses a cross-entropy loss to measure how similar the latent embeddings of a sample p_1 and p_2 are.

3.1.2 PAWS: Semi-Supervised Learning for Image Data

PAWS [2] implements a semi-supervised deep learning architecture based on contrastive views and support samples to assign one-hot encoded pseudo labels to input images.

PAWS implements three processing streams. Similar to GraphDINO, it processes an input sample and an augmented version of the input sample in order to train invariances that lead to network generalization. PAWS implements the image augmentations random crop, horizontal flip, color distortion and blur. Additionally a mini-batch of labeled support samples is processed in the third stream. The support samples are annotated samples that function as prototype samples for a cluster. PAWS assigns a pseudo label based on the similarity of the latent embeddings of the anchor view and the positive view in relation to the support samples. PAWS expects each mini-batch to be composed by an equal number of instances for each sampled class.

The objective function uses the cross entropy function to measure the similarity of the pseudo-labels of the anchor view and the positive view. To avoid node collapsing, PAWS uses sharpening in the objective function. The sharpening function increases the confidence of the probability distributions, i.e. decreases the entropy.

Additionally the objective function adds a regularization term, titled *mean entropy maximization (ME-MAX)* that aims to increase the entropy of an unlabeled training-batch, to ensure that each label is getting predicted. More concretely, distributions like $[[1.,0.,0.],[1.,0.,0.],[1.,0.,0.]]$ are penalized and distributions like $[[1.,0.,0.],[0.,1.,0.],[0.,0.,1.]]$ are favoured.

3.1.3 GraphPAWS: Semi-Supervised Learning for Graph Data

Our architecture GraphPAWS, depicted in Figure 2, is an adaptation of GraphDINO and PAWS. We adopt the PAWS architecture that processes an anchor view $\hat{x}$, a positive view $\hat{x}^+$ and a support sample mini-batch $\hat{x}_s$. We replace the encoder of PAWS with the GraphDINO graph transformer. The encoder generates the latent embeddings z for input $\hat{x}$ and respectively z_s and z^+ for the support sample mini-batch and the positive view. The support samples are fed into the similarity classifier to compute the pseudo-labels p and respectively p^+ for the anchor view and the positive view. While PAWS expects the support sample mini-batch to be balanced, we implement weight-balancing in the GraphPAWS adaptation of the similarity

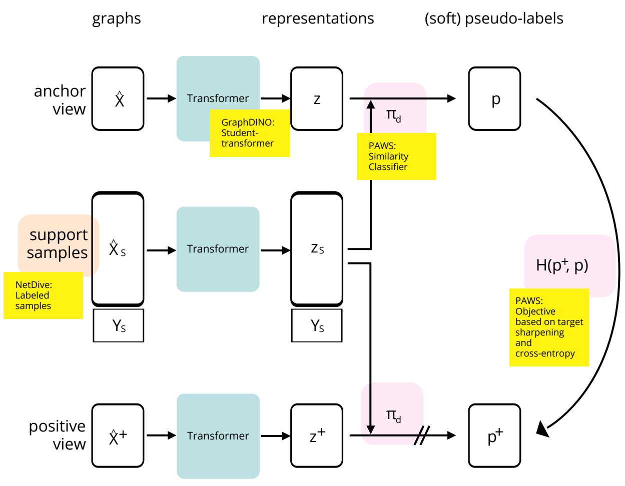


Figure 2: The semi-supervised GraphPAWS architecture for graph data.

classifier to compensate for class imbalances. This gives the user more flexibility while annotating neuron graphs.

The objective function is denoted with $H(p^+,p)$ in Figure 2. It corresponds to the objective function implemented by PAWS [2], which is based on cross entropy. We additionally train on the mean-squared error (mse) objective function.

The regularization term ME-MAX, implemented in PAWS, is added both to the cross-entropy and the mse objective function. We add a second regularization term that we title *One-Hot-Enforcement*, which enforces one-hot encodings of the embedded vectors. While ME-MAX operates over a batch of samples, One-Hot-Enforcement is applied to single training samples and averaged over a batch.

Equation 1 depicts the objective function in relation to the hyperparameters λ and γ that determine the relevance of the regularization terms *ME-MAX* and *One-Hot-Enforcement*,

$$\text{loss} + \lambda * \text{ME-MAX} + \gamma * \text{One-Hot-Enforcement}. \quad (1)$$

After training the latent embeddings are evaluated by latent space clustering through GMM or k-means and evaluated against ground truth labels.

3.2 NetDive

NetDive is developed for model engineers that design and refine the model and for domain experts to explore the graph data and to choose a model from the pre-trained model database. NetDive consists of a backend server to read and write data from and to the filesystem and a React frontend that communicates with the backend. On demand, i.e. using the refresh buttons in the user interface, the backend applies dimensionality reduction to the requested pre-computed latent embeddings. The frontend

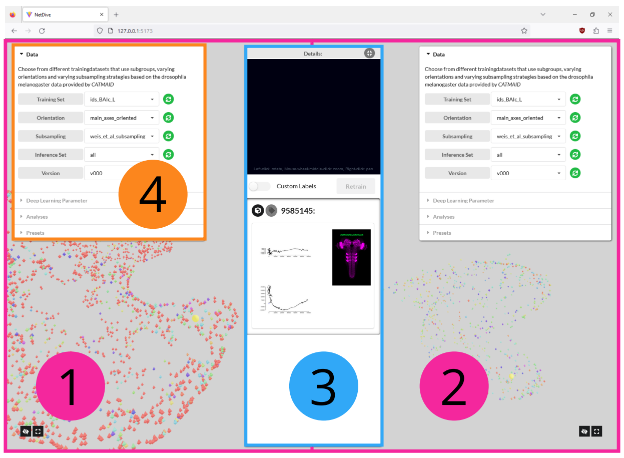

Data
Choose from different trainingdatasets that use subgroups, varying orientations and varying subsampling strategies based on the drosophila melanogaster data provided by CATMAID
Training Set
Orientation
Subsampling
Inference Set
Version
Deep Learning Parameter
Analyses
Presets
Details
Custom Labels
Retrain
9585145:
4
1
3
2

Figure 3: NetDive layout: (1) First 3D View, (2) Second 3D View, (3) Parameter Panel, (4) Detail View.

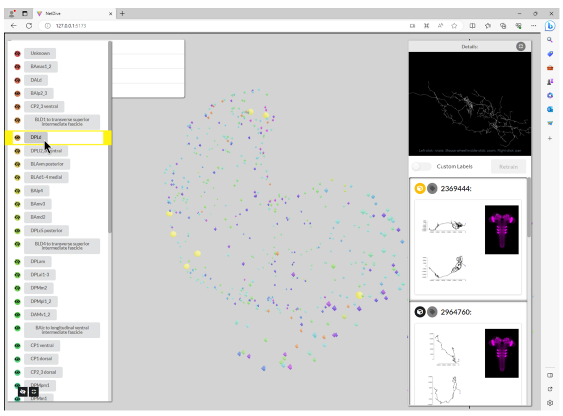


Figure 4: Selecting a lineage using the legend panel.

visualizes the dimensionality reduced latent embeddings in three dimensions. We chose to visualize the data in three instead of two dimensions to achieve a clearer cluster separation. This can come with the downside of distortion and occlusion. As clustering is not dealing with issues of length and angle preservation, and the points representing our use case of neurons are not covering a lot of space, which limits the issue of occlusion, we accepted these downsides. The scatterplots are implemented with THREE.JS. We use the clustering algorithms k-means and GMM to display the cluster predictions by color coding the datapoints in the scatterplots. The clustering algorithms are applied in the backend to the k latent dimensions that GraphDINO outputs before applying the dimensionality reduction. The prediction labels support the user to evaluate the quality of the latent embedding space.

3.2.1 User Interface

Figure 3 depicts the layout of NetDive. The user interface (UI) consists of three panels. Two views, annotated with (1) and (2), and a detail panel annotated with (3). The two view panels provide the user with parameters embedded in an accordion menu, annotated with (4), to select a model and analyses values to load and explore the latent embeddings in form of scatterplots in 3D generated by the corresponding selected model.

The user can choose between UMAP, t-SNE and PCA to reduce the 32 latent dimensions to three dimensions. Following Boggust et al. [3] we set PCA as default value. The datapoints in the scatterplots are color coded. The color of the datapoints is either assigned based on a selected ground truth or based on the cluster predictions generated with the selected clustering algorithm. The implemented clustering algorithms are k-means and Gaussian Mixture Models (GMM). The user can select the number of clusters to generate. The color codes can be used to toggle between the ground truth and the predicted clusters in order to detect similarities and dissimilarities. The color codes further aid with the comparative analyses using the two views, annotated with (1) and (2) in Figure 3.

Expandable legends, depicted in Figure 4, list the cluster labels that represent the latent embeddings. The labels are associated with the selected color codes. In Figure 4 the Hartenstein lineages, discussed in Section 4.1, are selected as a ground truth and the color codes correspond to the ground truth. The user can hide/show and select all datapoints with specific color codes within the legends.

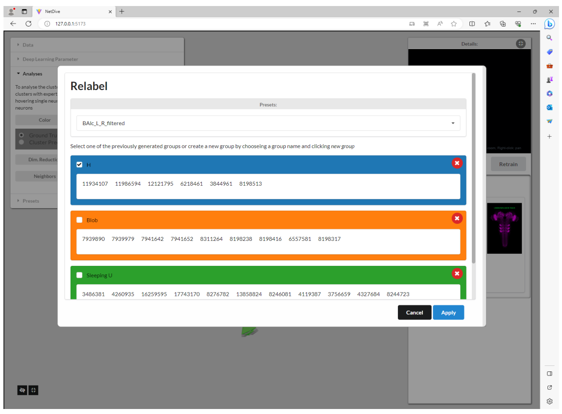
View (1), view (2) and the detail panel are linked, implementing the concept of multiple linked views (MLVs), displayed in juxtaposition. When a user selects single or multiple latent embeddings, the corresponding datapoint in the other view is highlighted, if present, and the detail panel depicts a scrollable list of tiles depicting information about the selected node(s). The tile headers show datapoint identifier, a button to select the according datapoint for a 3D graph rendering, which is displayed on top of the detail panel, and a button to add or update the datapoint label. The tile content shows pre-rendered images of the selected datapoint.

3.2.2 Relabeling and Retraining

After initially activating the relabeling feature using the *Relabeling* slider in the details panel, all datapoints are rendered gray and the color coding is disabled.

The user can then select embedded points and relabel them in the relabeling modal. The modal, i.e. the overlay, is depicted in Figure 5. The user can either create a new cluster group and add the selected id for the embedded graph to that group or they can add it to an existing cluster group.

The new labels are forwarded to the network training. The training is triggered within NetDive and processed using a *subprocess call*. The default hyperparameters correspond to hyperparameters of the currently loaded model and the user can update the network hyperparameters for the training within NetDive in the retraining modal. After


Relabel

Presets:

BAlc_L_R_filtered

Select one of the previously generated groups or create a new group by chooseing a group name and clicking *new group*

[✓] H

11934107 11986594 12121795 6218461 3844961 8198513

[ ] Blob

7939890 7939979 7941642 7941652 8311264 8198238 8198416 6557581 8198317

[ ] Sleeping U

3486381 4260935 16259595 17743170 8276782 13858824 8246081 4119387 3756659 4327684 8244723

Figure 5: Relabeling neurons in NetDive.

the training is completed, the embeddings generated with the new model for the chosen inference set will be available in NetDive.

NetDive aims to make the analyses and the iterative training fast and intuitive.

4 Experiments

We conduct a series of experiments to evaluate our dataset (Section 4.1) on the self-supervised SOTA architecture GraphDINO (Section 4.2) and on our architecture Graph-PAWS (Section 4.3).

4.1 Data

The drosophila melanogaster neuron graphs extracted from CATMAID, a platform for a collaborative reconstruction and annotation of data, are represented as undirected, acyclic graphs in three dimensions with the root node representing the soma, i.e. the cell body of the neuron.

We obtained the set of all available 7297 drosophila melanogaster larval neurons from CATMAID and restricted our analysis to a subset of 2970 neurons that was annotated by Michael Winding [23], as this subset contains more reliable neuron traces. We reduced this subset to 2541 neurons by removing all neurons that do not contain exactly one node annotated as *soma* and by removing all neurons with less than 200 nodes. CATMAID provides the neuron graphs as SWC files and we keep the (x,y,z) for each node of the neuron graph.

For the ground truth we generated a file that stores multiple ground truth cell type labels for each neuron id. The annotation files includes manually labeled cell types that we hand-crafted based on visual inspection. Furthermore it includes expert annotation by Dr. Volker Hartenstein [16]. Dr. Volker Hartenstein analyzed lineages, that describe neurons deriving from the same stem cells called *neuropblasts*. He states, that neurons within a lineage do not only share the same stem cell, but are also alike regarding the morphology. The datasets we define in out experiments are based on these lineages annotated by Dr. Hartenstein. Lineage BAlc neurons are located in the lateral surface of the antennal lobe and lineage CM4 in the postero-medial brain cortex.

We conduct our experiments on two subsets of the drosophila melanogaster dataset. We use the lineages BAlc and CM4 that Dr. Volker Hartenstein specified and divide these lineages in visually similar subgroups. Some lineages are visually coherent, whilst the lineages BAlc and CM4 fall into visually distinguishable groups. Lineage BAlc consists of 26 neurons, 13 in each brain hemisphere, and divides in three cluster groups. CM4 contains 66 neurons, 33 in each brain hemisphere, and divides in four cluster groups.

4.2 Self-supervised Training

We train GraphDINO for the learning rates $\in \{0.001, 0.0001, 0.00001\}$. While Weis et al. train on batch size $\in \{32, 64, 128\}$ we train on batch size $\in \{16, 32\}$ due to the smaller training dataset. We train with a 60-20-20 training-validation-test split on dataset *BAlc*.

We evaluate with 4 fold cross-validation for k-means and for GMM on the validation data by averaging over 100 k-means / GMM adjusted random index (ARI) scores per fold. ARI measures the similarity between two clusterings. We use the ARI computation of the python library *sklearn*, which outputs values between -0.5 for especially discordant clusterings and 1.0 for identical clusterings.

4.3 Semi-supervised Training

We trained 896 models using a grid-search on Graph-PAWS for the dimensions loss function, ME-MAX influence λ, One-Hot-Enforcement influence γ, batch size and learning rate. We used the values ['cross_entropy', 'mse'] for the loss, the values [0, 0.1, 0.5, 1] for λ and γ, the values [0.001, 0.003, 0.006, 0.0001, 0.00006, 0.00003, 0.00001] for the learning rate and the values [4, 8, 16, 32] for the batch size. We ran the hyperparameter search for 100 epochs. Accordingly to the self-supervised GraphDINO training, we train with a 60-20-20 training-validation-test split on dataset *BAlc*.

We evaluate with 4 fold cross-validation for k-means and for GMM on the validation data by averaging over 100 k-means / GMM ARI scores per fold. We list the top performing models and eliminate the models that suffer from node collapsing and from an incapability to learn. We therefore analyzed the feature distributions of the latent embeddings and the loss curve. Figure 6 depicts a loss curve with downwards trend on the top right side, indicating that the model learns, while the loss curve on the top left side does not decrease. On the bottom left Figure 6 depicts the feature distributions of the latent embeddings that have marginal standard deviations, indicating that all latent embeddings collapse to the same representation. On the bottom right side the features are well distributed.

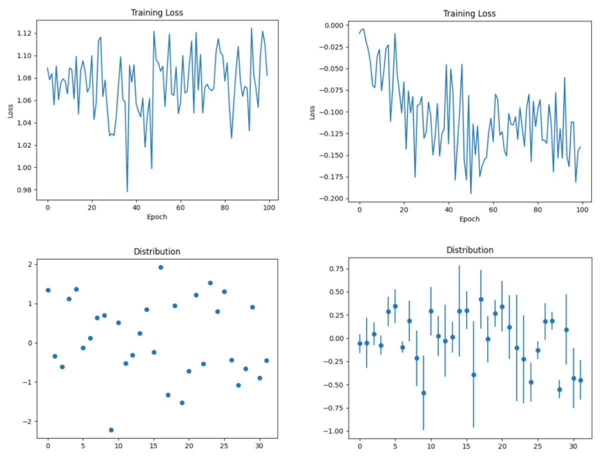


Figure 6: Example loss curves and feature distribution plots of a model suffering from node collapsing (left) and a model showing decreasing loss trend and distributed features (right).

5 Results and Evaluation

The GraphDINO model with learning rate 0.0001 and batch size 16 has the overall best ARI performance on k-means clustering with a score of 0.495.

The GraphPAWS model with learning rate 3e-05, batch size 32, gamma 1 and lambda 1 has the overall best ARI performance on GMM clustering with a score of 0.527. We repeated the training for these hyperparameters five times. The resulting ARI scores based on GMM clustering varied between 0.379 and 0.591. We have to consider this variance of performance when we evaluate the results.

Table 1 compares the ARI scores of the optimal self-supervised trained model with the optimal semi-supervised trained model.

	Self-Supervised Training	**Semi-Supervised Training**
Loss	cross entropy	mse
Learning Rate	0.0001	3e-05
Batch Size	16	32
Gamma	-	1
Lambda	-	1
ARI	0.495 (k-means)	0.527 (GMM)

Table 1: Results of optimal self-supervised trained model and semi-supervised trained model. The models are trained on lineage BAlc.

After determining the optimal GraphPAWS model we use the same model to train on a different lineage, i.e. lineage CM4, to demonstrate the usage of NetDive to iteratively explore the data and to feed new knowledge into the training.

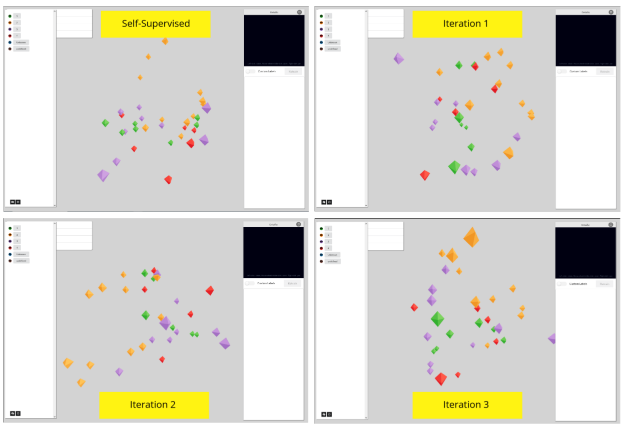


Figure 7: Neuron graph embeddings after each iteration denoted in Table 2.

For the NetDive evaluation instead of performing cross-validation, we train on the whole dataset and evaluate using the manually annotated CM4 samples that were partly also used as support samples during the semi-supervised training.

Initially we simulate the case that no labeled data is available and therefore train with the self-supervised GraphDINO model. We load the dimensionality reduced latent representations of the CM4 neuron graphs into NetDive and analyze the embeddings.

We explore how the ground truth clusters differ from the predicted clusters and label samples that are most distant from the visual cluster centroids. The ground truth we generated is for evaluation purposes only and is not available in a real use case. We then retrain a model on the GraphPAWS architecture with the annotated samples. We do this in three iterations and we add additional support samples in each iteration.

Figure 7 depicts the embeddings after each iteration, colored based on the CM4 ground truth. We cannot recognize a clear subdivision into clusters. We must therefore be cautious in assessing the positive trend in the improvement of ARI scores, reported in Table 2. Table 2 denoted the ARI scores based on k-means and GMM clustering after each iteration and lists the neuron ids of the support samples used for each iteration.

6 Discussion and Conclusion

In this paper, we established a workflow to address the problem of clustering graph data without initially having a ground truth for training whilst giving the user the possibility to guide the training process with minimal effort.

After the grid search that we performed in order to find the optimal GraphPAWS hyperparameters, we had to elim-

	Self-Supervised [22]	**Iteration 1**	**Iteration 2**	**Iteration 3**
ARI: **GMM**	0.152	0.145	0.143	0.225
ARI: **kmeans**	0.117	0.211	0.191	0.317
#Support Samples	-	4: 1 per class	8: 2 per class	12: 3 per class

Table 2: ARI scores of incremental training with NetDive. The ARI computation is based on manual ground truth for evaluation purposes only. The training is performed onlineage CM4.

inate models that had good ARI scores but which suffered from node collapsing and a lack of learning capability. These models sometimes had high ARI scores per coincidence.

While we used feature distribution visualizations and the loss curve plots to evaluate the models, these effects should also be visible in NetDive, as the embedding space would not divide in distinct clusters.

The results we achieved with GraphPAWS are not yet convincing. As documented in Section 5, we see an improvement reflected in the ARI scores (Table 2), but this effect is not clearly reflected in the NetDive clustering (Figure 7).

In order to address this, it would be an interesting future work, to further investigate the model optimization of GraphPAWS using NetDive, as we see indicators, that the pipeline that involves labeling support samples and restarting the training is intuitive and effective. We suspect that training on bigger datasets would eliminate outlier models and reduce the variance of performance for models trained on identical hyperparameters, reported in Section 4.3. Furthermore we want to experiment with fine-tuning the model after adding new support samples, instead of training new randomly initialized models, and therefore reduce training times. We also want to employ alternative subsampling strategies to reduce the input graphs to a fixed amount of nodes by evenly distributing the resampled nodes.

The NetDive user interface can be improved by adding simulations that visualize the cluster changes over time during training with color updates. It is also possible to add more characteristics of the neurons in the details Section and provide interaction techniques like brushing and linking over a feature space visualization for neurons to understand correlations between clusters and the cluster contents. We can extend the spatial representations and use the properties size and opacity of each data point to encode additional information besides the cluster label, e.g. the certainty of the cluster assignment in the opacity and the variance over a sequence of models in the size of the data point.

Regarding the evaluation we want to perform user studies with experts in the field of neuroscience to see how users outside the domain of deep learning can use visual analytics to refine pre-trained models and which features they are missing in the current NetDive setup.

Acknowledgement

This work was funded by the Austrian Science Fund (FWF): I4836-B“ and enabled by the Competence Centre VRVis. VRVis is funded by BMK, BMAW, Styria, SFG, Tyrol and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (879730) which is managed by FFG.

References

[1] Merriam-Webster neuroscience. www.merriam-webster.com/dictionary/neuroscience. Accessed: 2023-04-12.

[2] Mahmoud Assran, Mathilde Caron, Ishan Misra, Piotr Bojanowski, Armand Joulin, Nicolas Ballas, and Michael Rabbat. Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, October 2021.

[3] Angie Boggust, Brandon Carter, and Arvind Satyanarayan. Embedding Comparator: Visualizing Differences in Global Structure and Local Neighborhoods via Small Multiples. *27th International Conference on Intelligent User Interfaces*, pages 746–766, March 2022.

[4] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, pages 9912–9924, Red Hook, NY, USA, December 2020. Curran Associates Inc.

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Herve Jegou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging Properties in Self-Supervised Vision Transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9630–9640, October 2021.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *ICML'20*, pages 1597–1607. JMLR.org, July 2020.

[7] Yi Chen, Qinghui Zhang, Zeli Guan, Ying Zhao, and Wei Chen. GEMvis: a visual analysis method for the comparison and refinement of graph embedding models. *The Visual Computer*, 38(9-10):3449–3462, September 2022.

[8] Marta Costa, James D. Manton, Aaron D. Ostrovsky, Steffen Prohaska, and Gregory S.X.E. Jefferis. NBLAST: Rapid, sensitive comparison of neuronal structure and construction of neuron family databases. *Neuron*, 91(2):293–311, July 2016.

[9] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent a new approach to self-supervised learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, pages 21271–21284, Red Hook, NY, USA, December 2020. Curran Associates Inc.

[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9726–9735. Computer Vision Foundation / IEEE, 2020.

[11] Florian Heimerl, Christoph Kralj, Torsten Moller, and Michael Gleicher. embcomp : Visual Interactive Comparison of Vector Embeddings. *IEEE Transactions on Visualization and Computer Graphics*, 28(8):2953–2969, August 2022.

[12] Phuc Le-Khac, Graham Healy, and Alan Smeaton. *Contrastive Representation Learning: A Framework and Review*. October 2020.

[13] Quan Li, Kristanto Sean Njotoprawiro, Hammad Haleem, Qiaoan Chen, Chris Yi, and Xiaojuan Ma. EmbeddingVis: A Visual Analytics Approach to Comparative Network Embedding Inspection. *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 48–59, October 2018.

[14] Xu Liu, Yuxuan Liang, Chao Huang, Yu Zheng, Bryan Hooi, and Roger Zimmermann. When do contrastive learning signals help spatio-temporal graph forecasting? In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. ACM, November 2022.

[15] Zipeng Liu, Yang Wang, Jurgen Bernard, and Tamara Munzner. Visualizing Graph Neural Networks with CorGIE: Corresponding a Graph to Its Embedding. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2022.

[16] J. K. Lovick, K. T. Ngo, J. J. Omoto, D. C. Wong, J. D. Nguyen, and V. Hartenstein. Postembryonic lineages of the drosophila brain: I. development of the lineage-associated fiber tracts. *Dev Biol.*, 2013.

[17] National Research Council (US) Committee on Research Opportunities in Biology. *Opportunities in Biology*. National Academies Press (US), Washington (DC), 1989.

[18] Agapi Rissaki, Bruno Scarone, David Liu, Aditeya Pandey, Brennan Klein, Tina Eliassi-Rad, and Michelle A. Borkin. BiaScope: Visual Unfairness Diagnosis for Graph Embeddings. *2022 IEEE Visualization in Data Science (VDS)*, pages 27–36, October 2022.

[19] Venkatesh Sivaraman, Yiwei Wu, and Adam Perer. Emblaze: Illuminating Machine Learning Representations through Interactive Comparison of Embedding Spaces. *27th International Conference on Intelligent User Interfaces*, pages 418–432, March 2022.

[20] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. Technical report, 2018.

[21] Marissa Weis, Laura Hansel, Timo Lüddecke, and Alexander Ecker. *Self-supervised Representation Learning of Neuronal Morphologies*. December 2021.

[22] Marissa A. Weis, Laura Pede, Timo Lüddecke, and Alexander S Ecker. Self-supervised graph representation learning for neuronal morphologies. *Transactions on Machine Learning Research*, 2023.

[23] Michael Winding, Benjamin D. Pedigo, Christopher L. Barnes, Heather G. Patsolic, Youngser Park, Tom Kazimiers, Akira Fushiki, Ingrid V. Andrade, Avinash Khandelwal, Javier Valdes-Aleman, Feng Li, Nadine Randel, Elizabeth Barsotti, Ana Correia, Richard D. Fetter, Volker Hartenstein, Carey E. Priebe, Joshua T. Vogelstein, Albert Cardona, and Marta Zlatic. The connectome of an insect brain. *Science*, 379(6636):eadd9330, March 2023.

[24] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.