ConstellationNet: Reinventing Spatial Clustering through GNNs

Aidan Gao, Junhong Lin

Abstract

Spatial clustering is a crucial field, finding universal use across criminology, pathology, and urban planning. However, most spatial clustering algorithms cannot pull information from nearby nodes and suffer performance drops when dealing with higher dimensionality and large datasets, making them suboptimal for large-scale clustering. To improve upon this, we develop ConstellationNet, a convolution neural network(CNN)-graph neural network(GNN) framework that leverages the embedding power of a CNN, the neighbor aggregation of a GNN, and a neural network's ability to deal with batched data to improve spatial clustering and classification with graph augmented predictions. ConstellationNet achieves state-of-the-art performance on both supervised classification and unsupervised clustering across several datasets, outperforming state-of-the-art classification and clustering while reducing model size and training time by up to tenfold and improving baselines by 10 times. Because of its fast training and powerful nature, ConstellationNet holds promise in fields like epidemiology and medical imaging, able to quickly train on new and limited data to develop robust responses.

1 Introduction

Clustering, the grouping of data points based on similarity, has remained a crucial topic across many fields, finding applications in search engines[23, 29], medical imaging[31, 27], and anomaly detection[19, 24]. While clustering can be conducted on almost any dataset, spatial clustering, which focuses on clustering data points in space, acts as the primary and most crucial clustering method due to most datatypes, images, or three-dimensional molecular structures being able to be vectorized into spatial contexts. Traditional spatial clustering methods, like K-means or dbscan, have been widely used for processing and classifying large-scale spatial data, and recent advancements have improved density-based clustering while introducing spectral and covariance-based clustering[3, 51]. While these methods have improved clustering robustness significantly, most methods still often face three significant challenges when dealing with high-dimensional data and large datasets:

(1) **The Curse of Dimensionality.** Since data becomes sparser in higher dimensions, models have a harder time clustering these points based on distance.[2, 41]

(2) **Computational Inefficiency.** Due to most models having around $O(n^2)$ time complexity and being superpolynomial in worse-case scenarios, larger datasets become much harder for clustering methods to handle.[12]

(3) **Effetive Use of Neighborhood Information.** Since most spatial clustering models are based on the distance between points alone, they do not consider points' relations with each other, especially in dense groups, which can weaken their power when dealing with irregular data.

In a different realm, with the recent rise in machine learning, many different types of neural networks have been developed on various mediums, like images or languages, with some work being done into unsupervised classification, which can be analogized to clustering[36, 45]. However, due to only operating on the features of a single data point, these don't act similarly to clustering and suffer from the same limited scope. Graph Neural Networks (GNNs) have emerged as a powerful tool for learning from graph-structured data by leveraging information from both nodes and edges[6], showing promising results in various applications such as social network analysis[17], drug discovery[8], and human object interaction[37]. Due to their ability to aggregate information from neighbors and edges, GNNs have been applied to graph clustering tasks and have the potential to be used for spatial clustering tasks. However, Their primary strength of extrapolating information from edge and neighbor data restricts their application to graph-structured data, constrained primarily by the absence of edges between nodes in spatial data.

To address these challenges, this paper introduces ConstellationNet, a novel CNN-GNN spatial clustering framework designed to effectively and quickly cluster high-dimensional and large-scale spatial data. By integrating Convolutional Neural Networks (CNNs) with Graph Neural Networks (GNNs), ConstellationNet leverages the local feature extraction of CNNs and the neighborhood aggregation of GNNs, allowing the model to effectively extrapolate clusters of arbitrary shape and density. By constructing a weighted K-Nearest Neighbors (KNN) graph from spatial data, the model creates graph data from spatial data, using the GNN's unique aggregation to counteract the curse of dimensionality on created edges. To better distinguish features, ConstellationNet introduces an innovative CNN-GNN data-passing mechanism, passing the CNN's output to the

GNN, which gives the GNN more distinguishable features to cluster on, improving performance and power. Additionally, ConstellationNet incorporates the Deep Modularity Network (DMoN) operator and minibatching as further enhancements[42][20]. The DMoN operator provides an unsupervised pooling operator that optimizes cluster assignments and a loss, facilitating end-to-end clustering without requiring labels and mimicking traditional spatial clustering behavior. Furthermore, by processing subsets of the graph, ConstellationNet maintains high clustering accuracy while enabling scalability to larger datasets via more computational efficiency. In both supervised and unsupervised contexts, ConstellationNet achieves superior clustering performance compared to state-of-the-art methods, improving baselines by up to 10 times while beating state-of-the-art with up to tenfold reductions in parameters and training time.

This paper's contributions are summarized as follows:

- **Algorithm:** We propose ConstellationNet, a new framework that integrates CNNs and GNNs to perform spatial clustering on high-dimensional, large-scale datasets. We demonstrate the methodology for combining edge construction techniques with GNNs, enabling the application of graph-based learning methods to non-graph spatial data.
- **Extension:** We extend several graph-based and image-based methods to a spatial context, proving their viability and potential for future use in the area.
- **Evaluation:** We perform extensive experiments across several datasets and ablation studies to demonstrate the performance of each part of the framework, achieving state-of-the-art results in both supervised and unsupervised contexts without lower investments in memory and time.

2 Background

2.1 Spatial Clustering

Spatial clustering aims to group spatial data points based on location and distance, uncovering structures within spatial datasets. Traditional methods have been broadly categorized into centroid-based, density-based, and distribution-based approaches [21]. Centroid-based methods, such as K-means [33], partition data into a predefined number of clusters by minimizing the distance between data points and cluster centroids. Density-based methods like DBSCAN [16] identify clusters as areas of high point density. Distribution-based methods assume data are generated from a mixture of underlying probability distributions and clusters as such[38].

Recent research in spatial clustering has focused on enhancing traditional methods and developing new algorithms. New algorithms include spectral Clustering [34], which uses eigenvectors of a similarity matrix to perform dimensionality reduction before clustering, and Deep learning approaches like Deep Embedded Clustering (DEC) [46] which learns feature representations and cluster assignments. Improved baselines include OPTICS[4] and ST-DBSCAN [7], density-based methods that address the limitation of DBSCAN in detecting clusters with varying density. However, these new methods still ultimately suffer from the same curse of dimensionality due to the inherent lack of separation that high dimensional data creates, leading to worse performances on datasets with higher dimensionality, like STL and CIFAR.

2.2 Graph Neural Networks

Graph neural networks (GNNs) are a class of neural networks specifically designed to perform inference on data described by graphs. Unlike traditional neural networks, which use a series of transforms on the information expressed through a vector, GNNs instead process similar vector data that describes graph information, using neural networks to transform their features, as in fig. 1.

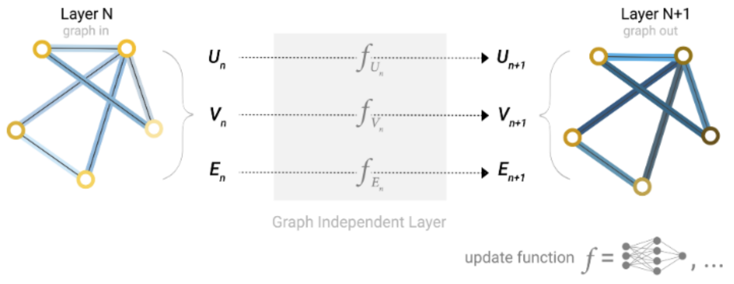

Figure 1: A visualization of a GNN transform function[6].

This transformation happens to node features, edge features, and graph information, acting similarly to other neural networks in highlighting features and predicting properties. However, due to the connected nature of a graph, GNNs are much more versatile and able to aggregate information from different parts to update representations. This can be done in predictions, where message passing allows edge embeddings to predict node classifications and vice versa, or in training, where each node's representation is updated by combining its features with the aggregated information from its neighbors, allowing the network to learn localized patterns within the graph [25].

Besides the basic GNN, several GNN architectures have been proposed, including Graph Convolutional Networks (GCNs) [25], Graph Attention Networks (GATs) [43], and Graph Isometric Networks [48]. These models differ primarily in how they aggregate information from neighboring nodes. GCNs use a localized approach to aggregate information with convolutional operations based on a convolution neural network, GATs introduce attention mechanisms that factor in edge weights to assign weights to neighbors and create selected aggregations, and GINs use a Weisfeiler-Lehman graph isomorphism test and a separate multilayer perception to aggregate data.

In spatial clustering, GNNs have had somewhat limited applications to the topic. Currently, most research into applying GNNs to spatial clustering has been focused on cell transcriptomics, with most methods establishing the construction of a graph through distance-based edge generation. We highlight two papers discussing the topic of applying GNNs toward spatial clustering: Learning Hierarchical Graph Neural Networks for Image Clustering and Cell Clustering for Spatial Transcriptomics Data with Graph Neural Networks[47][30]. Graph Neural Networks for Image Clustering establishes several precedents for this paper, mainly using K-Nearest Neighbors for graph creation. Additionally, this paper demonstrates the use of graph neural networks in spatial clustering, presenting a concrete base to build on top of. Cell Clustering for Spatial Transcriptomics and other papers further demonstrate the potential of using GNNs within spatial data and introducing dimensionality reduction to enhance clustering capability.

3 Methods

This section describes the Preliminaries and architecture of ConstellationNet. The main libraries used are PyTorch and PyTorch Geometric.

3.1 Preliminaries

3.1.1 Dataset Construction

To transform a spatial dataset into a graph for clustering, all images from the dataset are extracted and turned into one-dimensional vectors of values based on the process used in PECANN[50]. For instance, a single image in MNIST is transformed into a shape $(1, 728)$ vector with the 728 values corresponding to the pixel values concatenated. Each data point is then treated as a node, and the entire dataset is treated as a graph, allowing the node feature array to be created by stacking all images. Once a node feature array is obtained, a K-nearest neighbors algorithm is run on the data with a varying number of neighbors, denoted as the K value and an edge index is created based on the indexes returned from the K-nearest neighbors. An additional edge weight array is then created by inverting the distance between each edge created by the K-nearest neighbors. The data transformation process is seen in fig. 2.

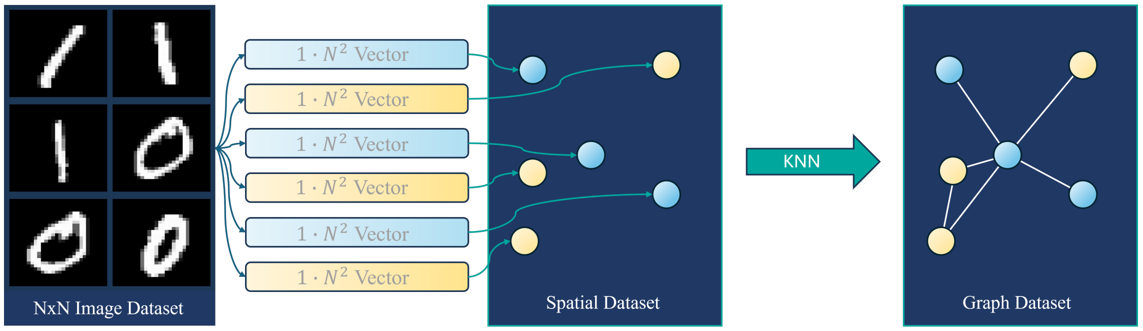

Figure 2: The dataset transformation pipeline visualized on six samples from the MNIST dataset, moving from images to nodes on a graph[14].[1]

3.1.2 Deep Modularity Network operator

To define a loss that ConstellationNet can train on during unsupervised clustering, the Deep Modularity Network (DMON) operator introduced by Google Research is extended to a spatial application, defined as the following[42]:

$$\mathcal{L}_{\text{DMon}}(\mathbf{C}; \mathbf{A}) = -\frac{1}{2m} \operatorname{Tr}(\mathbf{C}^\top \mathbf{B} \mathbf{C}) + \frac{\sqrt{k}}{n} \left\| \sum_i \mathbf{C}_i^\top \right\|_F - 1$$

Functioning as a pooling mechanism, the DMON operator integrates a differentiable clustering module that optimizes clustering assignments within the GNN framework through spectral modularity maximization, which seeks to maximize modularity based off of edge distribution differences between clusters and random distributions. The DMoN operator uses collapse regularization, a relaxed constraint on the soft clustering assignments, to prevent trivial assignments without compromising the clustering objective.

Besides acting as a clustering and pooling function, the DMoN operator provides the main unsupervised losses that guide the framework during unsupervised training, being the spectral loss and collapse regularization loss that the operator returns. Besides these losses, ConstellationNet employs two additional auxiliary losses to improve model performance: orthogonality loss and

[1]Self-made figure, citation is for MNIST images.

clustering loss. Orthogonality loss maximizes orthogonality between rows and columns in weight matrices, improving model uniqueness, while clustering loss attempts to balance cluster sizes.

3.1.3 CNN Embedding

Enabling GNNs and baselines to better extrapolate features and grouping clusters closer together, CNN embedding functions as a feature extractor through magnifying certain spatial features[28]. The extracted features are then pooled and projected into a lower-dimensional space via a fully connected layer, resulting in embeddings that serve as more distinct node features in the graph.

CNN embedding can be done in two ways: training a CNN to classify and then removing its fully connected layer to create an embedding model, or specifically training an embedding model via projection losses like triple loss[15]. A benefit of CNN embedding is the fact that it can be done in both supervised and unsupervised manners, with triple loss frameworks being an example of supervised frameworks and DINO and Simclr being unsupervised frameworks. [9, 11]

3.2 ConstellationNet

To address the issues with large-scale image clustering, We present a novel framework, ConstellationNet, that can be used both as a spatial embedding technique and as a standalone end-to-end clustering pipeline. ConstellationNet consists of two similar frameworks: a dynamic framework for classification and a static framework for clustering.

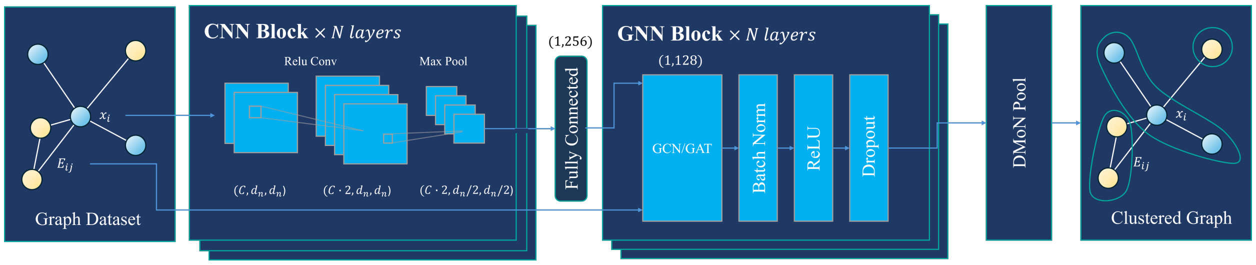

Figure 3: The Supervised CNN-GNN pipeline, acting on the end product of dataset construction. x_i Represents a node's attribute, and E_{ij} denotes an edge between node x_i and x_j. C represents the number of channels an image has, and D_n represents the size of the image.[2]

Supervised ConstellationNet is illustrated in fig. 3 above, consisting of a dynamic CNN-GNN framework with a final DMoN pool. Given a constructed graph dataset with spatial features $X = (x_1, x_2, x_3, \cdots, x_n) \in \mathbb{R}^{n \times d_n^2}$, ConstellationNet iteratively samples large neighborhoods to train on given a random cluster of nodes $X_{rand} \in \mathbb{R}^{b \times d_n^2}$ where b is the batch size, resulting a final feature array $X_{rand} \in \mathbb{R}^{b \cdot d \cdot b_n \times d_n^2}$ where b_n is the neighborhood size and d is the neighborhood hop depth. This feature array is then passed through the CNN embedder that reduces dimensionality down to variable size, transforming the subsampled feature array into $X_{CNN} = (x_1^2 x_2^2, x_3^2, \cdots, x_n^2) \in \mathbb{R}^{b \cdot d \cdot b_n \times G}$ where G is the final dimension of the CNN and the starting dimension of the GNN. This feature array and the subsampled portion of the edge index $E \in \mathbb{R}^{2 \times K \cdot n}$ are then passed into the GNN, which aggregates information before the DMoN pool finally clusters. We choose to use a residual edge index connection instead of building the edge index after reducing dimensionality with the CNN to maintain more information, allow the GNN to better aggregate distilled features based on original connections, and allow for dynamic updates. The main difference of this framework is its dynamic nature, functioning as one collective model. Because this framework operates in a supervised manner, both the CNN and GNN can be trained on supervised loss, allowing for abstract representations of data that can enhance the functionality of overall model more than separately training. To counteract the lack of connections between test data, we introduce graph-augmented classification, allowing ConstellationNet to take advantage of its training to augment predictions further. By connecting a test data point to the created train graph during prediction, Constellation establishes links to known data, building upon the training process and utilizing its data like never before.

[2]Self made figure

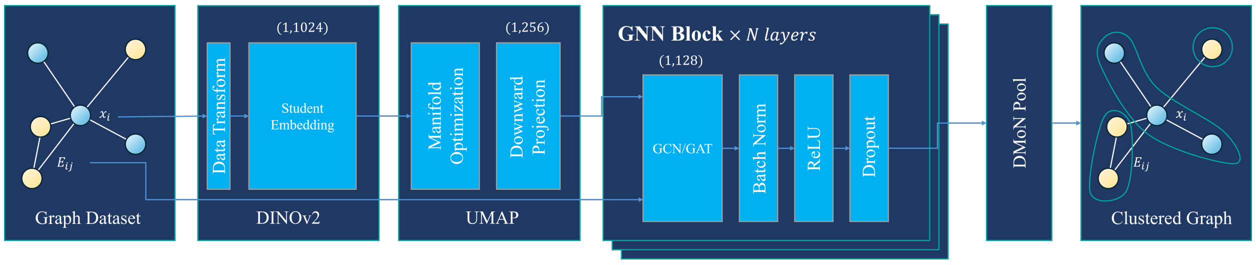

Figure 4: The Unsupervised CNN-GNN pipeline, acting on the end product of dataset construction. Data is passed through two transforms before the GNN, which are static in this case.[3]

The unsupervised ConstellationNet, seen in fig. 4 above, acts similarly to the supervised ConstellationNet, with a few changes. The same neighborhood sampling and residual edge connections are kept, leading the same arrays $X_{rand} \in \mathbb{R}^{b \cdot db_n \times d_n^2}$ and $E \in \mathbb{R}^{2 \times K \cdot n}$. However, unsupervised ConstellationNet is not a singular model but a pipeline through which data is passed because the DMoN operator and its losses are graph-focused, meaning that combined unsupervised training doesn't yield good results. Since an untrained CNN isn't good at feature separation to enhance clustering quality, spatial features are instead passed without neighborhoods through a large DINO v2 model, an unsupervised model pretrained on imagenet via a student and teacher framework, to transfer embed data into shape $X_{DINO} \in \mathbb{R}^{n \times 1024}$ [9]. To further improve the separation of clusters, Uniform Manifold Approximation and Projection(UMAP) is run through the embedded features to produce $X_{UMAP} \in \mathbb{R}^{n \times G}$. While empirically like other clustering algorithms, UMAP has demonstrated the ability to further enhance clusters once DINOv2 embeds features. Finally, after two embeddings, data is passed into the GNN and clustered via DMoN.

One notable feature of both supervised and unsupervised ConstellationNet is the interchangeability of all parts and the ability for the entire framework to be used as a transform. Since the end classifier is the GNN with a DMoN cluster optimizer, by removing the DMoN clustering assignment, the entire framework functions as a data embedding, reducing dimensionality down to the variable hidden dimension of the GNN and creating a new dataset that any spatial clustering baseline can quickly cluster. Additionally, the CNN and UMAP embeddings can be used as a separate standalone transform, which allows the final GNN to be switched out for any spatial clustering method.

[3]Self made figure

4 Experimentation

4.1 Experimental Setup

4.1.1 Dataset

I primarily conduct experiments on well-known image datasets, as images are a widely used form of spatial data. In a supervised setting, models are tested on the MNIST, CIFAR-10, and Imagnet image datasets, with MNIST being a black-and-white collection of handwritten digits while CIFAR-10 and Imagenet are both colored datasets of various objects[14, 26, 13]. In an unsupervised setting, models are tested on MNIST, CIFAR-10, CIFAR-100, and Oxford flowers[14, 26, 35]. Due to Imagenet's large size, testing is somewhat inconclusive in a supervised setting, using only 100 thousand data points, and thus, the dataset is not used in unsupervised testing.

4.1.2 Models

Several GNNs are used, with the main models mentioned in the background being the GCN, GAT, and GIN, as well as the SuperGAT, a version of the GAT with self-supervised attention. All models are built on PyTorch Geometric, and most models use existing convolutions within the library, except for the NAGphormer, a graph transformer that uses the source code provided by the paper[10]. For the GCN, GAT, and GIN, each model is built through stacking a variable number of the layers consisting of the following: the chosen convolution followed by a batch norm, a Relu activation function, and a dropout layer of 30%. The multilayer perception used for the GIN followed the MLP used in the paper, comprising two linear layers with a batch norm and Relu function between. During experiments, the two main properties that varied were the number of layers and the hidden dimension of the convolution and batch norms. After the final layer that transforms the data to the output dimension, the edge index is converted into a dense adjacency matrix and passed into the DMoN operator for clustering.

4.1.3 Metrics

To evaluate models, we use accuracy, NMI, and ARI throughout unsupervised testing to match with standard metrics. We only used top-1 accuracy during supervised training to match state-of-the-art image classification. Note the accuracy used during supervised and unsupervised training are different types of accuracy, with the supervised being a strict top 1 accuracy while the unsupervised uses a clustering accuracy determined via a contingency matrix of checking if clusters with the same class are in the same cluster and if clusters of different classes are in different clusters.

4.1.4 Supervised Constants

Training on all three vision datasets involved creating the graph as described in section 3.1.1 with a K of 50 and minibatching through Pytorch Geometric's neighbor-loader, sampling 30 neighbors over two hops during training and all neighbors during testing with a starting batch size of 64. For the imagenet dataset, a ConvNet embeds data into 1024 dimensions before feeding into the model[32]. Two different CNN architectures were used, one for the black and white images from MNIST and one for the RGB images from CIFAR 10. Both architectures expand from the input channel to 32, 64, and 128 channels over three convolutions, followed by a Relu activation and 2d maxpool on the later two convolutions. The MNIST CNN embeds the output down to 256, while the RGB model first embeds to 512 before embedding down to 256. The GNN model consists of the standard GNN architecture with a GCN convolution and a hidden size of 128 with five layers. Models were trained a variable number of epochs on the training portion of the dataset with negative log-likelihood, optimized by an Adam optimizer with a 0.001 learning rate and reduce learning rate on plateau scheduler with a patience of 5 epochs and a multiplication factor of 0.75 and tested using metrics described above.

4.1.5 Unsupervised Constants

Unsupervised training involves the same graph creation process and K value of 50, with the same minibatching happening as in the supervised constants. A DINO v2 model is used to universally embed down to 1024 dimensions, and the UMAP then further embeds down to either 512 or 256 dimensions, which is the starting point of the GNN. Either a GAT or GCN is used as the GNN, and the hidden dimension is half of the starting dimension. Models were trained for up to 100 epochs on the entire dataset with DMoN losses and collapse regularization with 0.1 multiplication factor, optimized by an Adam optimizer with a 0.001 learning rate and reduce learning rate on plateau scheduler with a patience of 5 epochs and a multiplication factor of 0.75 and tested using metrics described above.

4.1.6 Baselines

Model	Accuracy	NMI
K-Means 10 cluster	0.906	0.492
DBScan 3eps 2 samples	0.836	0.256
Optic 2 samples 1k	0.854	0.226
Birch 10 clusters 10k	0.925	0.685
Ward 10 clusters	0.914	0.693
Agglomerative 10 clusters	0.915	0.693
HDBscan	0.839	0.277

Table 1: Baseline Accuracies and NMI for MNIST.

A collection of spatial clustering processes and results from current state-of-the-art clustering are collected to provide a point of comparison between models and traditional clustering processes. Baselines use most clustering methods from sci-kit learn's clustering page, as they comprise the most popular and powerful spatial clustering methods[39]. These models include K-means, DBScan and HDBScan, OPTICS, ward and agglomerative clustering, and BIRCH. Out of these, Kmeans and Aggolmerative are typically used when comparing against GNNs, Kmeans due to its simplicity and Aggolmerative due to its being the best-performing baseline tested, as seen in table 1. Apart from baselines, state-of-the-art is taken from the top performing model on papers with code pages for each of the datasets used, with supervised state-of-the-art coming from the image classification leaderboards, while the unsupervised state-of-the-art comes from image clustering leaderboards.

4.2 Supervised Results

MNIST:		CIFAR 10:		Imagenet:	
Model	Accuracy	Model	Accuracy	Model	Accuracy
ConstellationNet	99.96%	ConstellationNet	99.67%	ConstellationNet*	91.10%
Merging CNN	99.87%	Efficient Adaptive Ensemble	99.61%	OmniVec	92.40%
EnsNet	99.84%	ViT-H/14	99.50%	CoCa	91%
Efficient-CapsNet	99.84%	DINOv2	99.50%	Model Soups	90.98%
SOPCNN	99.83%	µ2Net	99.49%	ViT-e	90.90%

Table 2: ConstellationNet compared against state of the art on MNIST, CIFAR-10, and Imagenet.

Table 2 lists the results of ConstellationNet across MNIST, CIFAR-10, and ImageNet, compared against state-of-the-art on each dataset. As seen across all datasets, ConstellationNet demonstrates leading performance as indicated by its top rankings amongst state-of-the-art. On MNIST, ConstellationNet achieves an accuracy of 99.96%, a near-perfect accuracy of 0.09% better than the next state-of-the-art. Similarly, on CIFAR-10, ConstellationNet attains an accuracy of 99.67%, surpassing state-of-the-art by 0.06%. While these improvements seem minimal, on two datasets where the state-of-the-art is near perfect, ConstellationNet's improvements are still significant, as evidenced by its increase of over 3 times compared to the previous state-of-the-art's improvement and being comparable to the prior state of the art's improvement on CIFAR 10.

ConstellationNet achieves a competitive accuracy of 91.10% on the ImageNet dataset, not outperforming OmniVec but still beating most other state-of-the-art. This result isn't indicative of ConstellationNet's performance on the entire dataset due to having only trained on 100 thousand samples and testing on the next 20 thousand. While the results suggest that ConstellationNet performs strongly on simpler datasets and scales worse on complex and large-scale datasets, ConstellationNet's small size and training time indicate that its potential to improve is high.

While ConstellationNet performs above state of the art on two datasets tested, its main benefit comes from its size, augmented predictions, and lower training time. On MNIST, ConstellationNet has 1.8 million parameters, similar to the Merging CNN, but only needs to train for around five epochs, whereas the Merging CNN trains for 300 epochs[22]. Additionally, on CIFAR 10, compared to the efficient adaptive ensemble, ConstellationNet's 5.8 million parameters and 20 epoch training time are almost half of its 11 million parameters and 5 times faster than its 100 epochs of training time[5]. Lastly, on Imagnet, despite using a large ConvNet[32] with 198 million parameters, meaning that ConstellationNet totals 203 million parameters, ConstellationNet still performs better than CoCa and Model Soups, both with over 2000 million parameters, while training for up to 100 epochs, over 9 times faster than OmniVec, which trains for 900 epochs in addition to pretraining for 2000 epochs[4][49, 44, 40].

Overall, the results confirm the power and efficiency of ConstellationNet in image classification tasks, outperforming state-of-the-art with significantly less training time and parameters.

[4]OmniVec size not listed due to not being state in paper.

4.3 Unsupervised Results

4.3.1 Baselines

Before discussing ConstellationNet's results across clustering datasets, we first demonstrate the GNN's ability to improve clustering, demonstrated in table 3 for MNIST and CIFAR 10:

Model	MNIST		CIFAR-10	
	NMI	ARI	NMI	ARI
GCN	0.676	0.573	0.127	0.078
GIN	**0.771**	**0.712**	0.111	0.062
GAT	0.710	0.647	0.131	0.080
SGAT	0.695	0.606	**0.137**	**0.086**
K-means	0.491	0.360	0.079	0.041
Agglomerative	0.693	0.453	0.077	0.036

Table 3: Baseline Performance for Models on MNIST and CIFAR-10

Across both datasets, when compared to the best-performing baselines, GNN models using the DMoN loss operator can perform significantly better than baselines, with top models improving NMI by 1.1x on MNIST and 1.7x on CIFAR-10 while improving ARI by 1.6x on MNIST and over 2x on CIFAR-10. Base GNN models and baselines both have somewhat good performances on MNIST due to the separated dataset and low dimensionality, but both GNN models and baselines suffer on CIFAR-10 due to the higher dimensionality of the dataset. Thus, while the GNN and DMoN operator perform better than baselines, a basic GNN still suffers similarly from the curse of dimensionality, showcasing the strength of the DMoN operator and the neighborhood aggregations of the GNN while presenting a limitation that ConstellationNet aims to resolve.

4.3.2 Ablation Study

Table 4 presents an ablation study exploring different configurations of ConstellationNet on MNIST and CIFAR-10, with both the base constellationNet and its use as a transform tested across both datasets.

Configuration	**MNIST**		**CIFAR-10**	
	NMI	ARI	NMI	ARI
Kmeans	0.49	0.36	0.08	0.04
+UMAP	0.86	0.83	0.08	0.04
+UMAP + GCN	0.91	0.92	-	-
+DINO	-	-	0.80	0.60
+DINO + UMAP	-	-	0.86	0.81
+DINO + UMAP + GCN *(ConstellationNet)*	-	-	0.90	0.86
GCN + DMoN	0.68	0.57	0.13	0.08
+UMAP	0.92	0.93	-	-
+DINO	-	-	0.72	0.72
+DINO + UMAP *(ConstellationNet)*	-	-	0.90	0.93

Table 4: ConstellationNet ablation studies across MNIST and CIFAR-10. Each component is tested using both the final GNN and Kmeans.

We observe that incorporating UMAP enhances performance in the case of an already separated dataset, confirming its use as a feature enhancer. On MNIST, where clusters are already somewhat distinct, UMAP offers a very concrete advantage, with the combination of UMAP and a GNN achieving an NMI of 0.92 and an ARI of 0.93, surpassing all other configurations and competing with state-of-the-art clustering. UMAP similarly improves the NMI and ARI of Kmeans on MNIST but doesn't affect its NMI or ARI on CIFAR-10, a dataset where data points are not distinctly grouped. Given that the UMAP functions by projecting manifolds down to a lower dimension, its poor performance on a dataset where no clear manifolds can be found is somewhat expected.

In addition to UMAP, using DINO significantly enhances the performance of both the Kmeans and GNN on CIFAR-10. DINO isn't used on MNIST because it is dissimilar to the training set for DINO. The DINO operator's ability to embed data points and separate clusters significantly improves clustering performance and allows UMAP to further enhance features, as seen in the best NMI of 0.90 and an ARI of 0.93.

Finally, using the final GNN as an embedding, ConstellationNet is shown to have concrete and significant improvements over a baseline like Kmeans. On MNIST, passing data through ConstellationNet before the Kmeans causes an improvement of around two times for both NMI and ARI and over 10 times for CIFAR-10, showcasing the power of ConstellationNet as a transform.

4.3.3 ConstellationNet

To evaluate ConstellationNet's clustering performance, we test it against state-of-the-art on CIFAR-10, CIFAR-100, and Oxford Flowers, as seen in table 5.

Model	CIFAR-10			CIFAR-100			Flowers		
	NMI	ARI	Accuracy	NMI	ARI	Accuracy	NMI	ARI	Accuracy
KMeans	0.08	0.04	0.06	0.021	0.142	0.213	0.048	0.195	0.203
ConstellationNet	*0.931*	*0.944*	**0.971**	*0.876*	0.124	**0.967**	0.992	0.971	**0.999**
TURTLE	**0.985**	**0.989**	*0.969*	**0.915**	**0.832**	*0.899*	0	0	*0.996*
TEMI CLIP ViT-L	0.926	0.932	0.946	0.799	*0.612*	0.737	-	-	-
DPAC	0.87	0.866	0.934	-	-	-	-	-	-
SPICE*	-	-	-	0.583	0.422	0.584	-	-	-

Table 5: Performance of Different Models on CIFAR-10, CIFAR-100, and Flowers Datasets. The best results are highlighted, and the second-best results are italicized.

ConstellationNet demonstrates leading performance on three datasets, outperforming most other state-of-the-art except unsupervised transfer, or TURTLE. On CIFAR-10 and CIFAR-100, ConstellationNet closely trails TURTLE while outperforming it in accuracy, and on the Flowers dataset, ConstellationNet surpasses all other available state-of-the-art, achieving a near-perfect accuracy of 0.999 compared to TURTLE's 0.996. Thus, ConstellationNet demonstrates its power amongst the current state of the art, being competitive across all metrics.

However, the strength of ConstellationNet again lies in its smaller model size and training time. Due to TURTLE and TEMI being the closest-performing models, we only compare ConstellationNet to them when referring to runtime and parameters[18, 1]. Both TURTLE and TEMI use a DINO v2 as a data embedding before their respective frameworks, but TURTLE notably uses a DINO v2 giant instead of a DINO v2 large, which results in an increase of around 500 million parameters compared to our framework. TURTLE also uses CLIP as another representation space, which can add another 400 million parameters to the model size. For the clustering model, ConstellationNet uses around 500 thousand trainable parameters for its GNN, while TEMI uses another vision transformer or DINO model, both several hundred million parameters, and TURTLE uses upwards of 1.9 million parameters[1, 18]. Thus, the unsupervised ConstellationNet pipeline consists of around 500 million parameters, while TURTLE uses upwards of 1.5 billion parameters, and TEMI uses around 1 billion parameters. Additionally, while ConstellationNet trains for 100 epochs, TEMI trains for 200 epoch, and TURTLE trains for 6000 iterations at a batch size of 10000, which for CIFAR-10 and 100 equates to 1000 epochs. This means that ConstellationNet competes with state-of-the-art with more a twofold reduction in parameters and training, showcasing its robust power across datasets.

Overall, these results confirm the power and efficiency of ConstellationNet in unsupervised image clustering tasks, being able to efficiently cluster both as an independent pipeline and as a transform for another clustering method with less memory and time used.

5 Conclusion

In conclusion, this study has introduced ConstellationNet, a CNN-GNN framework that performs state-of-the-art clustering while enhancing baselines by over 10 times. Using both a convolutional neural network and a graph neural network, ConstellationNet addresses significant problems of dimensionality and local information problems that traditional and deep learning clustering methods face while using mini-batching to improve runtime and predictions. Through novel message passing and residual edge connection frameworks, ConstellationNet showcases its power as an end-to-end clustering pipeline and data embedding, outperforming state-of-the-art across several popular image datasets in accuracy, size, and training time. Due to its robust supervised and unsupervised performance, fast predictions due to smaller model size and minibatching, and data-specific knowledge, ConstellationNet holds many potential applications, able to be quickly trained and utilized in fields like epidemiology, urban planning, and medical imaging to solve crucial problems.

6 Acknowledgements

We thank Julian Shun for providing datasets during the experimental process and high-level feedback during points of the research process.

Bibliography

[1] N. Adaloglou, F. Michels, H. Kalisch, and M. Kollmann. Exploring the limits of deep image clustering using pretrained models, 2023.

[2] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metric in high-dimensional space. *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)*, 02 2002.

[3] D. T. Anh Luong and V. Chandola. A k-means approach to clustering disease progressions. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 268–274, 2017.

[4] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD Record*, volume 28, pages 49–60. ACM, 1999.

[5] B. Antonio, D. Moroni, and M. Martinelli. Efficient adaptive ensembling for image classification. *Expert Systems*, Aug. 2023.

[6] A. P. A. B. W. Benjamin Sanchez-Lengeling, Emily Reif. A gentle introduction to graph neural networks. 2021.

[7] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[8] P. Bongini, M. Bianchini, and F. Scarselli. Molecular generative graph neural networks for drug discovery. *Neurocomputing*, 450:242–252, 2021.

[9] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers, 2021.

[10] J. Chen, K. Gao, G. Li, and K. He. Nagphormer: A tokenized graph transformer for node classification in large graphs, 2023.

[11] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations, 2020.

[12] S. V. David Arthur. How slow is the k-means method? *Proceedings of the twenty-second annual symposium on Computational geometry*, 2006.

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

[14] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[15] X. Dong and J. Shen. Triplet loss in siamese network for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[17] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, page 417–426, New York, NY, USA, 2019. Association for Computing Machinery.

[18] A. Gadetsky, Y. Jiang, and M. Brbic. Let go of your labels with unsupervised transfer, 2024.

[19] G. C. Gerhard Munz, Sa Li. Traffic anomaly detection using k-means clustering. *net.in.tum.d*, 2007.

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

[21] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[22] D. HIRATA and N. TAKAHASHI. Ensemble learning in cnn augmented with fully connected subnetworks. *IEICE Transactions on Information and Systems*, E106.D(7):1258–1261, July 2023.

[23] Z. C. W.-Y. M. J. M. Hua-Jun Zeng, Qi-Cai He. Learning to cluster web search results — Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval — dl.acm.org. [Accessed 24-10-2024].

[24] G. W. Iwan Syarif, Adam Prugel-Bennett. Unsupervised clustering approach for network anomaly detection. *Networked Digital Technologies*, 2012.

[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint, 2017.

[26] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[27] C.-T. Kuo, P. B. Walker, O. Carmichael, and I. Davidson. Spectral clustering for medical imaging. In *2014 IEEE International Conference on Data Mining*, pages 887–892, 2014.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] K. W.-T. Leung, W. Ng, and D. L. Lee. Personalized concept-based clustering of search engine queries. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1505–1518, 2008.

[30] J. Li, S. Chen, X. Pan, Y. Yuan, and H.-B. Shen. Cell clustering for spatial transcriptomics data with graph neural networks, Jun 2022.

[31] T. Liu, C. Rosenberg, and H. Rowley. Clustering billions of images with large scale nearest neighbor search. page 28, 02 2007.

[32] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[33] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[34] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[35] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[36] S. Park, S. Han, S. Kim, D. Kim, S. Park, S. Hong, and M. Cha. Improving unsupervised image clustering with robust learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12278–12287, June 2021.

[37] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu. Learning human-object interactions by graph parsing neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[38] D. A. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, pages 659–663. Springer, 2009.

[39] scikit learn. clustering.

[40] S. Srivastava and G. Sharma. Omnivec: Learning robust representations with cross modal sharing, 2023.

[41] J. F. Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning*. Springer Nature Link, 2009.

[42] A. Tsitsulin, J. Palowitch, and B. Perozzi. Graph clustering with graph neural networks. 2020.

[43] P. Veličković et al. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[44] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022.

[45] S. G. M. P.-L. V. G. . Wouter Van Gansbeke, Simon Vandenhende. Scan: Learning to classify images without labels. In *ECVA*, 2020.

[46] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pages 478–487, 2016.

[47] Y. Xing, T. He, T. Xiao, Y. Wang, Y. Xiong, W. Xia, D. Wipf, Z. Zhang, and S. Soatto. Learning hierarchical graph neural networks for image clustering, 2021.

[48] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2019.

[49] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu. Coca: Contrastive captioners are image-text foundation models, 2022.

[50] S. Yu, J. Engels, Y. Huang, and J. Shun. Pecann: Parallel efficient clustering with graph-based approximate nearest neighbor search, 2023.

[51] M. Çelik, F. Dadaşer-Çelik, and A. Dokuz. Anomaly detection in temperature data using dbscan algorithm. In *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pages 91–95, 2011.