RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences
Media Informatics

Master's Thesis

Development of a Source Library for Transfer Learning: Leveraging Clustering and Contrastive Learning Techniques

Sneha Banerjee
Matriculation Number: **430069**

Wednesday 28th August, 2024

1st Examiner: Prof. Dr. Christian Bauckhage, University of Bonn
2nd Examiner: Prof. Dr.-Ing. Alexander Verl, ISW, University of Stuttgart
Supervisor: Daniel Schiller, Fraunhofer IPA

Abstract

This master's thesis aims at training a deep neural network for the selection of "good" objects which collectively represent a very large dataset. It is essential to choose some objects as the representative members of the dataset which can act as a source library. This source library is then utilized to evaluate the performance of the transfer learning approaches in bin picking applications. The goal of this thesis is to train a deep neural network that is complex enough to accurately learn the necessary features of an object to be able to classify them later on into cluster and evaluate different clustering algorithm for this purpose. During the course of the thesis, the challenges and motivation of this work is discussed. Existing work in the literature has been reviewed thereafter. The fundamental concepts that are essential for the development of this whole pipeline of training a neural network to get the latent space representations of the objects in the dataset and then clustering them have been discussed next. Then comes an elaborate documentation of the neural network used for this purpose and the different clustering algorithms prevalent in the domain. Subsequently, the idea of a new clustering algorithm has been proposed keeping in mind the constraints and limitations in this particular use-case. The performance of the different clustering algorithms has been evaluated in detail. An extension study on fine-tuning the model hyperparameters has been performed to improve the performance of the deep neural network and the clustering algorithms. The existing deep neural network at Fraunhofer Institute for Manufacturing Engineering and Automation (IPA) has been used as a baseline to compare the performance of the proposed method. Finally, a critical analysis of the results has been performed keeping in mind the goals, limitations and the scope for further development.

Keywords: Feature representations, deep neural network, autoencoder, clustering algorithms, bin picking

Contents

1 Introduction

1.1 Motivation

Robot-based bin picking process automation offers enormous potential to completely transform bin picking tasks by boosting productivity, efficiency and cost-effectiveness. However, robot's capacity to adapt and learn from a variety of dynamic situations is the deciding factor in the success of bin picking applications. Transfer learning is a potent machine learning technique that could help with this problem by allowing robots to swiftly adapt to new tasks and situations by using the knowledge from pre-trained models [84]. Transfer learning seeks to use information from one domain or task to enhance performance on a related domain or task. Conventional machine learning paradigms involve training models on a given dataset for a certain problem, starting from scratch. This involves a huge computational overhead and time and cost associated with it. But with transfer learning, even when the datasets or domains are distinct, the information obtained from one task can be helpful for completing a related task.

As the demand for efficient and cost effective machine learning solutions are on a rise in multiple sectors like robotic, computer vision, natural language processing and healthcare, more and more is the call for a source library in transfer learning. A handful number of reasons why a source library is necessary for transfer learning is elaborated below.

1. **Faster Research and development** The entire pipeline for developing and testing a deep neural network from scratch can be intensely difficult and time-consuming. The process of research and development are sped up by the pool of tested models, datasets and algorithms that a source library provides.

2. **Scalability to Diverse Domains** Transfer learning is used in numerous fields like robotics, time series analysis, image recognition and natural language processing. A source library provides frameworks that are adaptable to different tasks and datasets, as well as modular components that can be integrated with existing networks to be used in different applications.

3. **Reproducibility and Accessibility** A collection of models, dataset and algorithms are available on using a source library in the transfer learning paradigm. Hence it facilitates cooperation, repeatability and transparency for researchers, developers and other stakeholders.

4. **Minimization of Redundant Efforts** When developing similar algorithms or models, researchers frequently have to start from scratch in order to provide transfer learning solutions. A source library can reduce the effort in duplication of work and free up developers to work on new ideas and applications by offering pre-built components and reference implementations. Researchers can more efficiently compare results, assess performance and build upon each other's work by embracing standard frameworks, interfaces and evaluation criteria.

Overall, a transfer learning source library fosters new research, increases access to state-of-the-art techniques and expedites the development of practical applications across a range of disciplines. Academics and researches are made possible by a centralized information repository and an ecology of collaboration using artificial intelligence and machine learning to tackle difficult challenges and find new opportunities for progress.

1.2 Task and Goal

Bin picking is fundamentally a pose estimation problem, which makes it a computer vision task. The solution to the bin picking problem is greatly influenced by the advancements in deep learning techniques for computer vision in recent years. At the moment, there are two available methods: model-free solutions and model-based ones. The model free methods rely on an image to find appropriate grasp poses, whereas the model-based solutions use Computer-Aided Design (CAD) models of items to detect them in a scene[92]. The bin picking application at IPA utilizes an advanced model-based method initially based on PQ-Net++ from [45]. The dataset used for this task is the ABC dataset[47] which is a collection of one million CAD

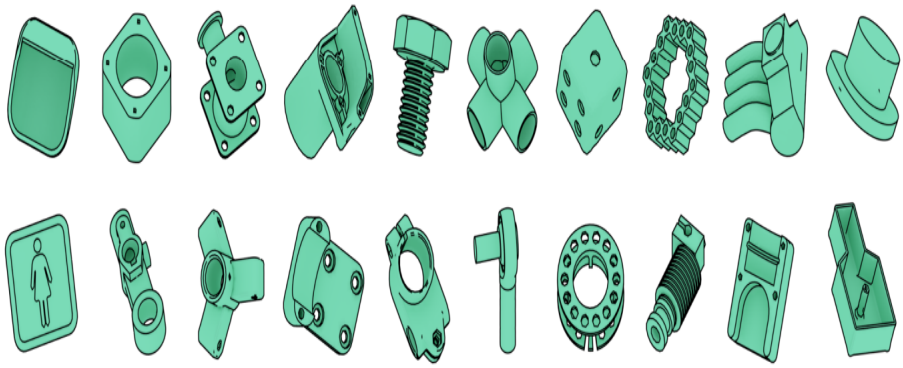

Figure 1.2.1: Some random objects from the ABC dataset.[47]

models without any ground truth labels. It contains a large variety of objects, a handful of which are shown in Fig. 1.2.1.

The goal of this work is to develop a source library consisting of "good" source data which can be used as the data from the source domain in the further steps in a transfer learning pipeline. Therefore, the focus is to identify which features prevalent in an object makes it a "good" source data. A diverse collection of source data would be the objects that approximately captures the overall variance of the entire 1 million ABC dataset. Thus, it is trivial to think that more objects in the source library could better capture the overall variance of the dataset. But since the bin application is a model-based approach, it would be required to formulate the CAD models of all these objects. Thus, having a large number of objects in the source library would entail high computational time for the generation of all the object models. Therefore, it is crucial to determine a permissible number of objects in the source library to facilitate its usage down the pipeline. To further elaborate on this, if two objects are very similar to one-another than in spite of having a large similarity which just one other object, it loses the generalization capability of being nearly similar to a lot of other objects in the dataset. Hence, it would require a large amount of objects in the source library to capture the entire variance in the dataset. Thus, there is a trade-off between being very similar to a very small subset of objects and being approximately similar to a larger subset of objects in the dataset. Moreover, it is not universally defined which features in an object qualify as an important features and how it contributes in deciding whether an object is capable of becoming a viable source data.

1.3 Approach

An end-to-end autoencoder proposed by [59] is used to get the feature representations of the objects in the dataset[47]. The autoencoder learns both the local and the global features of an object. Once the feature representation of the objects are obtained, they are grouped into clusters utilizing multiple clustering algorithm. A suitable clustering algorithm is utilized to get the representing members of the clusters. A high level overview of the entire method is shown in Fig. 1.3.1. The clustering algorithm would not just consider the Euclidean distance between the datapoints but also other intrinsic features of the dataset such as the density because real world datasets are not necessarily globular in structure. A suitable evaluation metric such as Density Based Clustering Validation (DBCV)[61] is used to compare the quality of the clustering algorithms which is robust to the size and shape of individual clusters as well as the outliers and noise of the dataset. Since, computing the DBCV index on the entire dataset is extremely computationally expensive, the evaluation is performed on random batches of the subset. Moreover, the DBCV is sensitive to the number of features in the dataset and fails to provide any tangible result in the presence of a large number of features (e.g. over 150). Thus, a dimensionality reduction is necessary for the evaluation of the clustering algorithm. Once, the clustering algorithm with best results in the evaluation metrics is achieved the entire dataset is categorized into a number of clusters, where the medoid of the cluster is chosen as the representative member of the cluster. The similarity of the test dataset with the representative members of

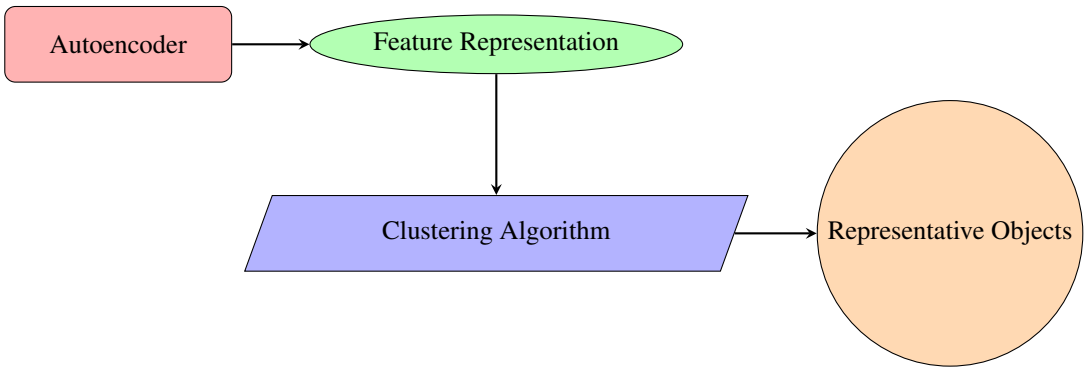

Figure 1.3.1: Overview of the entire pipeline for development of source library.

the datasets shows the generalization quality of the source library. The approximate time for each step of the work is shown in Tab. 1.

Step	Time
Training the model	28 hours
Dimensionality reduction	4 hours
Clustering Algorithm	3 hours
Similarity evaluation	11 minutes

Table 1: Overview of the approximate time taken for the creation of the source library

1.4 Contribution

Although density-based clustering algorithm like Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) shows promising results when the clusters in real-world datasets aren't globular in nature, yet the current existing implementations in the literature doesn't put any restrictions on the number of clusters returned by the clusters. In case of very large datasets like the ABC dataset[47], these algorithms return a large number of clusters (over multiple thousands). The model generation of such thousands of objects aren't feasible or efficient in business situations such as in IPA. Thus, novel contribution in this work is stated as follows:

- The idea of multi-level HDBSCAN algorithm is proposed in this work to limit the number of clusters to a feasible number but also retain the properties of a density based clustering algorithm.
- An extensive experimentation on the optimal hyperparameter values during model training has been carried on for the development of a high performing network used for the generation of feature representations of the objects in the dataset.

2 Related Work

This chapter provides a quick overview of the literature on unsupervised learning approaches for point clouds as well as variety of local feature extraction techniques on point clouds.

Since deep learning methods can directly process point clouds, feature learning for 3D objects has advanced significantly in the recent years. By independently learning every point in the point cloud and combining point characteristics with permutation invariant operations, PointNet [76] and Deep Sets [116] were the pioneers in the field to handle unordered and unstructured 3D points. Both [76] and [116] were important developments in deep learning for handling unordered data. [76] was designed particularly for 3D point cloud analysis, while [116] provides a broader framework for feature learning that is permutation invariant and could be extrapolated to graphs and sequences. These methods were capable to generate promising results. But it had limitations in learning the local features and hence needs further improvement to make them more scalable across different applications. PointNet++ [77] enhances [76] by introducing a hierarchical feature learning technique that functions directly in a metric space. It has the ability to acquire knowledge at varying levels, the hierarchical arrangement of point clouds into nested partitions and combining sampling techniques that capture complex spatial relationships while maintaining permutation invariance.

[104] designed a Dynamic Graph Convolutional Neural Network (DGCNN) architecture specifically for point cloud data. Its main contribution is to enable more efficient feature learning by dynamically building graph structures from point clouds based on local geometric correlations. But on the other hand, dynamic graph creation adds more computational complexity and might necessitate precise hyperparameter tuning. Although PointCNN [54] demonstrated state-of-the-art performance on a range of point cloud applications, the x-transformation adds to computational overhead and may restrict scalability, particularly when working with large-scale datasets. But it provided a permutation-invariant procedure that changes the orientation of point clouds to a canonical orientation enabling the use of standard convolutional operations. A deep learning architecture that directly operates on three-dimensional point clouds was introduced in [110].

The development of techniques for unsupervised representation learning on point clouds has been the focus of several works in the literature. These methods can be broadly classified as discriminative [31] or generative [83]. The majority of techniques include autoencoding [113] or Generative Adversarial Network (GAN)s [83]. Their primary method of operation is using an encoder to convert an input point cloud into a global latent representation [79], [88], or a latent distribution in the variational case [34], [32], after which a decoder tries to reconstruct the input. High-level structural features can be effectively modeled by generative models. Nevertheless, a lot of these methods often work under the premise that every 3D item in a category has the same pose [81]. Because of this assumption, these methods are sensitive to translation and rotation operations. Discriminative models, unlike generative methods, are trained to distinguish between data augmentations. It is seen that these methods produce good quality latent representations. Contrastive methods [79], [81] have demonstrated impressive outcomes in recent unsupervised point cloud representation learning approaches, among discriminative models. Through data augmentation and comparison, contrastive approaches also enable the generation of rotation invariant representations [74]. Predicting a representation that is distant from the negative examples and closer to the positive ones is a crucial concept in contrastive learning [14]. But in order to perform better, these algorithms need a large number of negative examples to compare and rely largely on the selection of the negative samples and how they are paired alongside the positives [31], [15]. Such an unsupervised method is typically computationally expensive and requires careful handling of the negative pairs, which can be achieved by relying on memory banks, huge batch sizes, or tailored mining strategies.[31]

[57] presented a Convolutional Neural Network (CNN) that incorporates relation-aware convolutional procedures which capture both local and global geometric relationships inside point clouds. Position-adaptive feature learning was the primary innovation of PAConv[112], which is the dynamic assembly of convolutional kernels based on local geometric structures. By using this method, the network performed better in segmentation and classification tasks by improving its capacity to detect intricate geometric patterns and

variances within point clouds. The main contribution of AdaptConv [118] was the inclusion of adaptive graph convolutional operations, which dynamically modify convolutional kernels receptive fields according to the point clouds local geometric characteristics. Because of the flexibility, the network can better execute tasks like segmentation and classification by capturing local and global geometric data. Convolutional operations adaptive nature might add more computational complexity. Moreover, the quality of local geometric information and intricacy of the point cloud may have a significant influence on the adaptive graph convolutional approach's efficacy, which may affect how well it performs in various datasets and applications.

In [93], a novel approach to encoding point clouds in three-dimensional (3D) space was presented. This method makes use of quadratic terms to more accurately describe local surface geometry. The capacity of the model to identify surface curvature and shape fluctuations is enhanced by the addition of quadratic components. As a result of including higher-order geometric information in the feature representations the discriminative ability of deep learning models were improved for point cloud segmentation and object classification tasks. [23] described how perturbation learning can improve the resolution of point cloud data. Its ability to generate high-resolution point clouds with improved geometric details is useful for 3D reconstruction and object detection applications. By using perturbation learning, the method can effectively capture fine-grained features and surface details, improving the overall quality of the unsampled point clouds. Also, the method can be applied to a variety of point cloud datasets because it is flexible and does not require any particular surface representations. A new technique for geometric feature preservation in Light Detection and Ranging (LiDAR) point cloud compression was reported by [91]. This framework uses a layer-wise geometry aggregation approach to achieve lossless compression, which ensures that no information is lost across multiple layers of the point cloud. The computational complexity of combining and evaluating geometric data across multiple levels could be a drawback. As more input point cloud data becomes available, this complexity might increase. [117] provided an innovative technique for improving the performance of 3D object detection by using transformer networks to refine votes. It combines and refines votes from different parts of the point cloud to enhance object recognition. This method increased accuracy and robustness in 3D object identification tasks. While these approaches were remarkably successful, in order to do feature learning, they need supervised data. Point cloud models are difficult to implement in novel real-world scenarios where labelled data is hard to come by due to their reliance on annotation. Therefore, its critical to devise strategies for lowering the quality of annotated samples needed to complete deep learning-based point cloud interpretation tasks with the necessary performance[59].

3 Fundamentals

3.1 Bin Picking

3.1.1 Definition and Application

Bin picking is a robotics and automation technique used in manufacturing and logistics, automotive industry, particularly in warehouses and assembly lines. It involves locating, grabbing and manipulating objects from a bin or container utilizing robotic devices equipped with sensors, cameras and algorithms. Thus, it is a crucial area of interest for research in computer vision and robotics. There are usually multiple steps in the process: sensing and perception, planning and path calculation, grasping and manipulation, ultimately transportation and placement which will be further elaborated in the following passages.[8]

Sensing and Perception To determine the kind position orientation and the kind of objects inside a bin or container, the robot uses its sensors and cameras to scan it. For the purpose of identifying the items, especially ones that are partially occluded or organized randomly or have entanglement in between them, advanced computer vision algorithms are used for the purpose.[8]

Planning and Path Calculation Following the identification of the items in the bin, the robots control system determines the best course for the arm or gripper to take in order to reach and retrieve the intended item in the bin. Avoiding collision with other objects in the bin and calculating the most efficient path for carrying the objects from one point to another are the two crucial parts in this stage. [8]

Grasping and Manipulation The robot can places its arm in a suitable location and employs its gripper to firmly hold the object of interest. Firm and control over the grippers is crucial for this stage to prevent damage and unwanted slippage.[8]

Transportation and Placement Once the robot has a firm grip on the object it moves the item to the intended spot, like a shipping container, assembly line or conveyor belt. Before settings the object down, the robot occasionally needs to realign or orient it.[8]
There are numerous benefits of bin picking in manufacturing and logistics.

- **Enhanced Efficiency** Businesses can greatly boost productivity and throughput, particularly in high-volume operations, by automating the process of selecting goods from bins.
- **Flexibility** Bin picking applications are appropriate for a variety of manufacturing situations because they can handle a wide range of items based on their shapes, size and materials.
- **Savings** Automating manual labor-intensive jobs, such as bin picking, can save labor costs and lower the chance of accidents that come with physical handling.
- **Health of Employees** Picking heavy objects for a longer period of time can often be detrimental for the worker's health. Thus, using automated bin picking solutions can be extremely helpful in such cases.

Keeping everything in mind, bin picking applications are essential to contemporary manufacturing and logistic operations, helping businesses to increase productivity and streamline workflows.

3.1.2 Gripper Types

Grippers are crucial parts of robotic systems used in bin picking applications that allow them to grab and move objects out of the bins or containers. There are different kinds of grippers and each is appropriate for a particular purpose depending on the shape, size, weight and substance of the objects being handled. The following are a few typical gripper kinds used for bin picking.

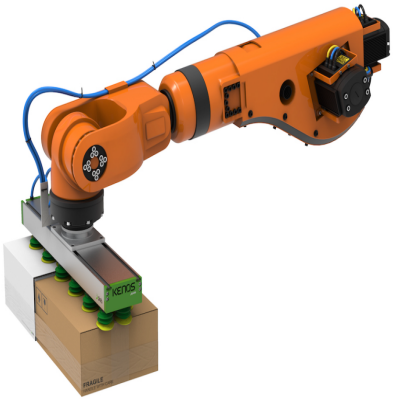

Figure 3.1.1: Vacuum Grippers[73]

Figure 3.1.2: Mechanical Grippers[29]

Figure 3.1.3: Magnetic Grippers[87]

Figure 3.1.4: Adaptive Grippers[63]

Vacuum Grippers These grippers cling to smooth or flat surfaces using suction as shown in Fig. 3.1.1. When handling huge, flat good like sheets of plastic, glass or metal, they are especially useful. By using interchangeable suction cups or pads, vacuum grippers can be made to handle objects of various sizes and shapes. The primary advantage of these type of grippers is the need for one-surface accessibility. This makes handling cluttered bins easier. The drawback however is that a vacuum gripper can still generate traction even if the intended grasping position is not met. This results in a possibly erroneous placement and an imprecise position of the gripper.[92]

Mechanical Grippers To grasp objects, mechanical grippers use mechanical jaws, fingers or claws. They may be tailored with various jaw configurations to meet particular object geometries and are appropriate for a broad variety of shapes and sizes. Mechanical grippers are useful for picking up objects with uneven shapes, such as bottles, crates and components, as they offer a solid hold as shown in Fig. 3.1.2. But because of the tough and rigid mechanical nature of the gripper, it might cause damage while handling delicate and sensitive objects.[27]

Magnetic Grippers These devices draw in and hold onto ferromagnetic materials like steel using electromagnetic forces as shown in Fig. 3.1.3. As such, the benefits and drawbacks are similar to those of vacuum grippers. They are appropriate for applications where cleanliness or delicate surfaces are a concern since they are perfect for handling metallic objects without the need for direct contact. Because of its magnetic properties, it can induce magnetic behavior to the objects being handled which is not desirable and can have serious implication in certain applications[92]. Furthermore, it is possible that the gripper might grab multiple objects at a time instead of a single object which is not desirable.[92]

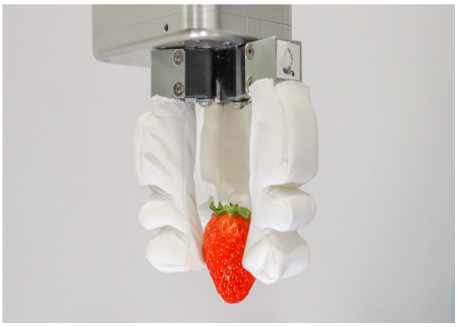

Figure 3.1.5: Soft Grippers[65]

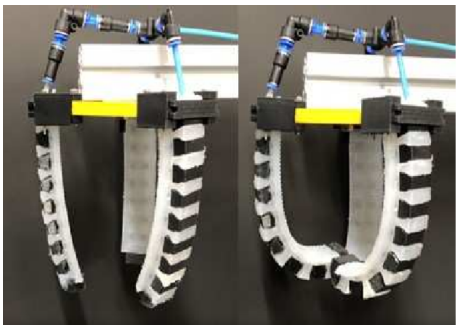

Figure 3.1.6: Hybrid Grippers with soft and rigid structures[70]

Adaptive Grippers These grippers can handle a broad range of sizes and forms without the need for exact placement since they are made to conform to the shape of the object being held as shown in Fig. 3.1.4. These grippers frequently modify their shape to fit the curves of the object by using inflated bladders or flexible polymers. However, to facilitate its adaptive property, it demands for complex control algorithms for precise grasping.[70]

Soft Grippers Designed to softly grab delicate or unusually shaped objects without causing damage, soft grippers are created from responsive materials like silicone or rubber as shown in Fig. 3.1.5. They are frequently employed in tasks like picking up food or electrical components, where dexterity and delicate handling are crucial. However, they have lower gripping force as compared rigid grippers, thus, limiting suitability for heavier objects.[70]

Hybrid Grippers To improve performance and versatility, hybrid grippers integrate many grasping techniques such as vacuum suctions and mechanical clamping. These grippers are frequently employed in intricate bin picking scenarios where a solitary gripping technique would not be adequate. It also requires more calibration and maintenance due to the integration of multiple systems. It also leads to greater power consumption and computational overhead for executing hybrid gripping mechanism. Thus, the characteristics of the objects being handled, the capabilities of the robotic system and the particular requirements of the application all play a role in the gripper selection. Stakeholders can maximize the effectiveness, dependability and precision of their bin picking processes by choosing the right kind of gripper.[70]

3.1.3 Model-Based Approaches

Model-based bin picking approaches use a digital model or representation, of the items in the bin as a guide for organizing and carrying out the bin picking procedure. To enable accurate perception and manipulation, these methods make use of CAD data or 3-D models of the items. Fig. 3.1.7 gives an overview of model-based bin picking approach, an elaboration of which is provided below.

Object Modelling Creating precise digital models of the objects to be picked is the first stage in a model-based bin-picking strategy. CAD software or 3D point cloud generated from real objects through scanning methods like laser scanning or photogrammetry can be used for this. The model ought to be accurately represent the objects surface features, size and geometry.[55]

Bin Perception The robots sensors, including depth cameras or laser scanners, record information about the contents of the bin once the object models are accessible. Based on the shape, size and orientation of each object in the bin, this data is processed to identify and segment each of them. In order to match and align the perceived objects with their corresponding models, the digital models are used as reference templates.[55]

Pose Estimation It is the process of figuring out each recognized object's exact location and orientation with

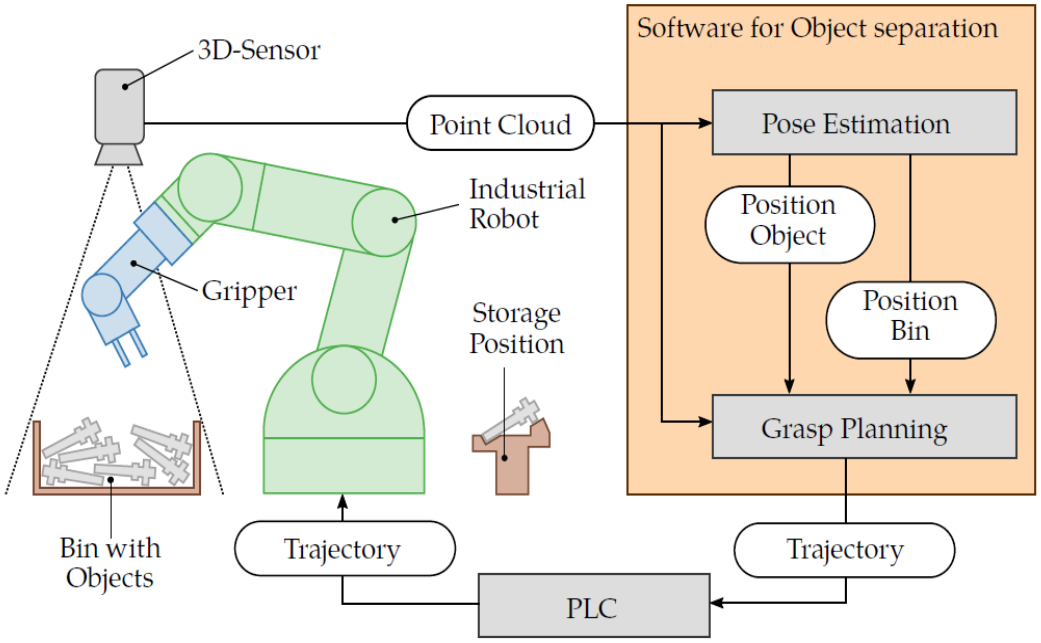

Figure 3.1.7: Overview of model-based bin picking approach.[92]

respect to the gripper or end effector of the robot. Planning precise gripping trajectories and guaranteeing successful object manipulation depend on this phase. Pose estimation methods include geometric fitting algorithms, point cloud registration and feature mapping.[55]

Grasping Planning Once the posture of the object is determined, the robot arranges its arm to grasp the object from the bin. In order for the robot's arm or gripper to approach, grasp and safely lift each object, collision-free motion path needs to be calculated. To enable successful gripping, planning algorithms take into account variables including gripper kinematics, object geometry and workspace limitations.[55]

Feedback Control To monitor and modify the robot's movements in real-time while it is grasping, feedbacks from the sensors such as force/torque sensors or vision systems is utilized. By using feedback control approaches, the robot may adjust its grip in response to unforeseen disruptions or changes in the surroundings, guaranteeing dependable and durable picking performance.[55]

Validation and Verification The system compares the actual results with the predicted result based on model-based planning to determine whether the grasp was successful after selecting each object. Verifying grasp stability, item alignment and adherence to quality standards are a few examples of validation tasks. The robot can rectify any mistakes by trying to pick the object again or modifying its approach.[55]

3.1.4 Model-Free Approaches

The goal of model-free bin picking approach is to pick objects from a bin without using CAD data or predefined object models. To identify and handle objects based solely on sensor data, these methods use advanced machine learning algorithms and complex perception techniques. Model-free approaches can be broadly categorized into two groups: deep-learning based approach[9], reinforcement learning based approach[53] and hybrid perception-based approach[41].

Deep Learning-Based Approach Without explicitly modelling the objects, deep learning techniques like CNNs are trained on massive amount of sensor data to learn object recognition and grasping policies on

their own. Labelled object instances and corresponding grasping actions are collected from large datasets of sensor data (e.g. Red Green Blue (RGB) images, depth maps). Using end-to-end optimization, CNNs are trained on those data to acquire grasping policies and object representations. Based on sensor input, robotic systems with trained models are able to grasp objects and identify them in real-time. To enhance grip execution and boost performance, feedback from sensors (such as force/torque sensors) are employed.[9]

Reinforcement Learning-Based Approach Without explicit object models, robots can acquire grasping policies through trial and error interactions with the environment by using Reinforcement Learning (RL) techniques. By performing grasping movements and getting feedback (like reward signals) when a task is completed successfully, the robot interacts with its surroundings(e.g. a bin). RL algorithms learn grasping policies by optimizing the total rewards earned from the accomplished grasps and interactions with the surroundings. In order to maximize the performance, RL agents strike a balance between exploring novel grasping tactics and utilizing previously learnt rules. Robotic systems are equipped with capability to perform bin-picking tasks in real-time.[53]

Hybrid Perception-Based Approach Robots can learn grasping rules from data by combining perception and action primitives in hybrid techniques, which also incorporate domain knowledge or limitations. Using deep learning models or conventional computer vision approaches, sensor data (e.g. RGB images) is processed to extract object representation or features. A combination of imitation learning and reinforcement learning is used to learn grasping policies. Using the learnt policies and the sensor data, perception and action modules work together to optimize the grasping performance. The learnt policies are improved over time by adjusting the grip execution based real-time data from sensors.[41]

3.1.5 Robot System

For pick and place tasks, delta robots and parallel kinematics are the usual system. For bin-picking applications, however, serial kinematics and articulated robots are the more popular options. The advantages include, a much bigger workspace, more degrees of freedom for the end-effector or gripper, allowing a wider range of gripping poses and less limitations sue to internal collisions. The robot system's main task is to match various positions. The following are the three crucial ones for bin picking systems.[92]

- The position of the object $\mathbf{W_{P_O}}$ characterizes the object's position in relation to the world coordinates.
- The position of the gripper $\mathbf{W_{P_G}}$ characterizes the end-effector's position in relation to the world coordinates.
- The grasping position $\mathbf{G_{P_O}}$ characterizes the position of the object in relation to the position of the gripper. This is particularly crucial foe the object's precise placement.

These three positions constitute the closed transformation chain. In other words, the computation of the third pose is easy if the first two are known. These three poses are shown in Fig. 3.1.8 A robot can place its end-effector using one of the two movement techniques: linear (LIN) motion and point-to-point (PTP) motion. Every robot motor moves synchronously when in PTP motion. Generally speaking, figuring out the trajectory of the Tool Center Point (TCP) is difficult when in this mode. The benefit of this motion, however, is its potential to achieve maximum speed. Because of this, this mode is employed to travel great distances between certain poses. The TCP is a crucial point in the LIN motion. In this motion, the TCP travels linearly from one place to another in Cartesian space. Its speed is much slower than in PTP, which is a drawback for confined workspaces. Inverse kinematics calculation and singularity avoidance are the two components of the LIN mode operation. These configurations are typically detected by robot controller, which will stop the robot.[92]

3.2 Deep learning-based 3D Image processing

3.2.1 Convolution Neural Network

Inspired from the visual cortex of humans, (CNN or ConvNet) is a type of deep neural network designed for processing data which appear in a grid like manner like images, videos, etc. It was introduced by LeCun

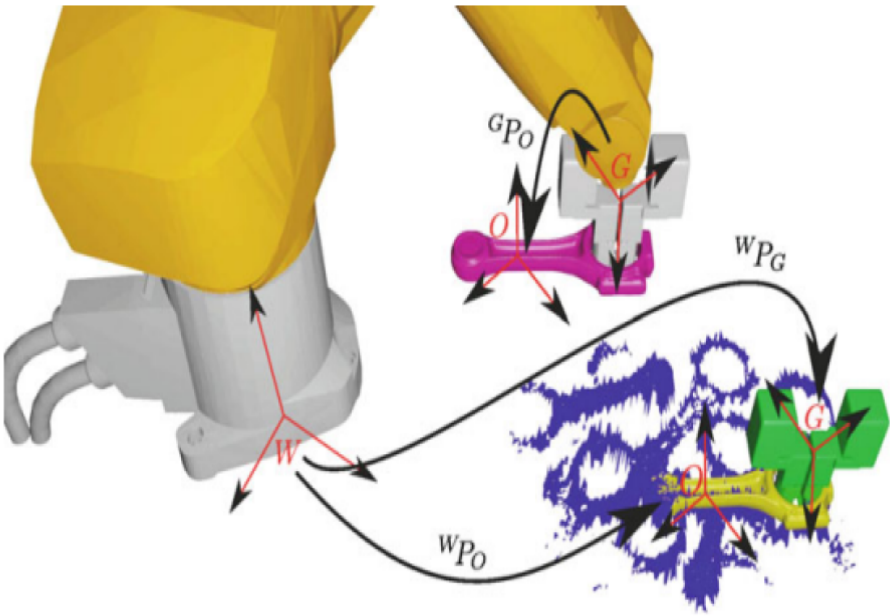

Figure 3.1.8: Important positions of the workspace in a bin-picking application. *W* is the world coordinate system with its origin at the centre of the base of the robot. *O* is the object coordinate system with its origin at the center of the object. *G* is the gripper coordinate system with its origin at the center of the gripper.[11]

et. al in [50] in 1998. It gets its name because of the usage of a special kind of linear mathematical operation called the convolution instead of using matrix multiplication as prevalent in the pre-existing neural networks. The key components of a CNN are - Convolution layer, activation function, pooling layer, loss function, output layer. The principle building block of a CNN is the convolution layer. It consists of a number of learnable filters (kernels) which can be visualized like a cubic block. The success of CNNs can be attributed to three major concepts: sparse interactions, parameter sharing and equivariant representations.[30]

Sparse interactions In traditional fully connected network, matrix multiplication is performed which involves a parameter matrix. The interaction between input and output units are captured by a distinct parameter in the parameter matrix. But CNNs have sparse interactions, i.e. only a subset of units or neurons in a layer is connected to a local region in the preceding layer. This is done by using kernels that are significantly smaller than the input. This implies, less number of parameters need to be stored which reduces the memory consumption and also it is computationally efficient because it has to perform fewer operations. For example, if there are m inputs and n outputs, the fully connected network would need to store m x n parameters and have a runtime of $O(m \text{ x } n)$ time complexity per input. On the other hand, if the number of connections for each unit is restricted to be k, then there would be k x n parameters and have a runtime of $O(k \text{ x } n)$ time complexity per input, where k is quite some fold lesser in magnitude as compared to m. Moreover, since convolution layers are stacked upon one another in a deep convolution network, units in the deeper layers have a larger receptive field, because of its indirect interaction with a larger region in the input.

Parameter sharing Another important focus behind using the convolution layer was to reduce the number of parameters in a Fully Connected Network (FCN). For example in a 1024*1024 image, a FCN would have over 1 million hidden units, which means it would have over 1 trillion trainable parameters. But the pixels in an image are only locally correlated. So, CNNs make use of the kernels to limit the focus on smaller regions on the image at a time known as the receptive field. This significantly reduces the number of trainable parameters, thus, reducing the memory consumption [16]. These layers perform a convolution

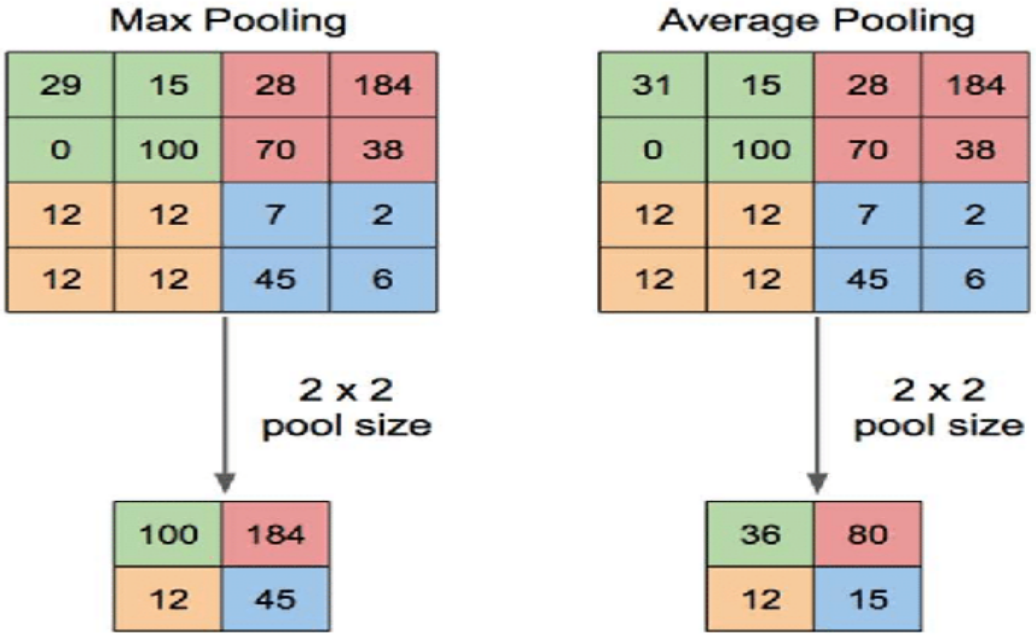

Figure 3.2.1: Max pooling operation on the left and average pooling operation on the right.[114]

operation between the input (eg. image) I and the kernel K to produce an output S known as the feature map as in Eq. 1.[30]

Equivariant Representations A function is said to be equivariant if the input to a function is changed, then the output changes in the similar way. Because of the parameter sharing mechanism, convolutions operations are translation equivariant. When the kernels are applied on an input image, the convolution layers generate a 2D map of where the particular feature occurs in the particular image. Furthermore, a pixel is related to its neighboring pixels to form a meaningful context(create a feature e.g. an edge in an image) but it is not limited to where it can occur throughout the image. Thus, it creates multiple filters each of which look for the same feature throughout the image. But it is also to be kept in mind that convolution is not equivariant to other geometric and affine transformations like rotation, scaling, etc. Since convolution operations are commutative in nature, Eq. 1 can also be written as Eq. 2. Typically, Eq. 2 is easier to incorporate into a machine learning library since values for both $'m'$ and $'n'$ varies within a small range.[30]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n). \tag{1}$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n). \tag{2}$$

Activation function The next step in a CNN is to apply an activation function. the purpose of using a non linear activation function is to capture the non-linearity in the data. Moreover if non-linearity is not used in between the multiple layers of a neural network, the network is effectively only one layer deep which is not capable even to capture the non-linearity in real world datasets. Rectified Linear Unit (ReLU) is the most commonly used non-linear activation function used in CNNs because unlike sigmoid function or tanh function it does not penalise "too correct" data points. Another remarkable benefit of using ReLU is its ability to propagate gradient through deep networks with a constant factor. Also it is more memory efficient to use ReLU as it doesn't require to store the ReLU outputs separately as compared to tanh outputs.[30]

Pooling The third step of a CNN is to use the pooling layer. The main purpose of this operation is to make the detection of the features robust to the exact location of the eye (i.e. invariant to small translations). The

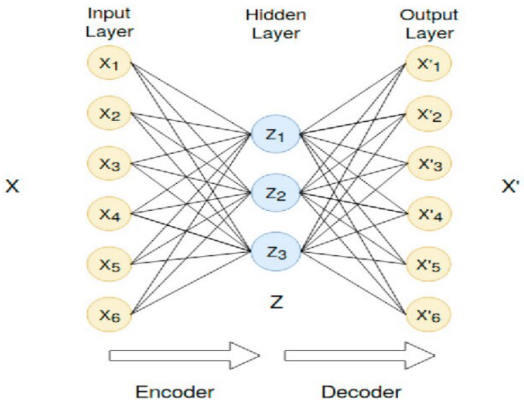

Figure 3.2.2: Architecture of an undercomplete autoencoder.[60]

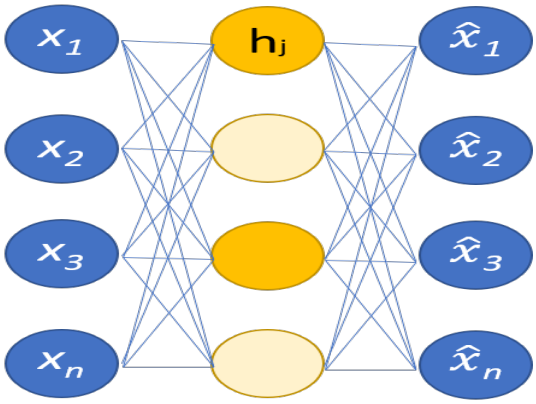

Figure 3.2.3: Architecture of a single layer sparse encoder. The hidden nodes in bright yellow are active, while the ones in light yellow are set to zero, hence inactive[5]

different types of pooling operations are - max pooling and average pooling. It also helps in reducing the dimensionality of the input without losing too much information. This is done to make the computations faster down the deeper layers of the network. If the pooling operations are performed after every k pixels, then the next layer processes inputs that are k times lesser as shown in Fig. 3.2.1. As the name suggest, the maximum value within the kernel is taken during max-pooling and the average value during average pooling. Since the number of parameters in a layer are dependent on the size of the input to the layer, it significantly reduces the computational overhead on using pooling operations.

Batch Normalization Another very important aspect in CNN is batch normalization. In this step, it takes the output of the convolution layer and reassigns the pictures in the feature maps to a new mean and standard deviation. At first, the pixels values are normalized by calculating the z-score of all the pixels. It is then multiplied by a learnable parameter commonly referred as alpha which denotes the range and is then added to another learnable parameter beta which controls the offset value as shown in Eq. 3.

$$x' = \left(\frac{(x - mean)}{standard\,deviation} \times alpha\right) + beta, \tag{3}$$

where x' is the normalized value after each batch, x is the value of the pixel in the feature map. Batch normalization method is crucial because it handles the exploring gradient problem.

3.2.2 Autoencoders

Autoencoders are a special type of feedforward neural network in which the output tries to reconstruct the input. It is predominantly used in unsupervised learning for the tasks if dimensionality reduction, learning feature representations and for data compression. It tries the encode the input data into a more compact representation with lower dimensions called the "code"[30] or "latent space". The idea is that this latent space representation should capture the most vital aspects of the input data. This is done by the first part of the network called the encoder. The second part of the network, the decoder then tries to decode this compressed representation of the input data to reconstruct the original input data as accurately as possible. During the training of an autoencoder, the goal is to minimize this reconstruction loss, i.e. the difference between the original input data and the output of the decoder. There are different types of autoencoders - undercomplete autoencoders, convolutional autoencoders, regularized autoencoders and variational autoencoders.

Undercomplete autoencoders ensure that the dimension of the latent space representation is less than the dimension of the original input data as show in Fig.3.2.2. The output of the decoder to be the exact copy of

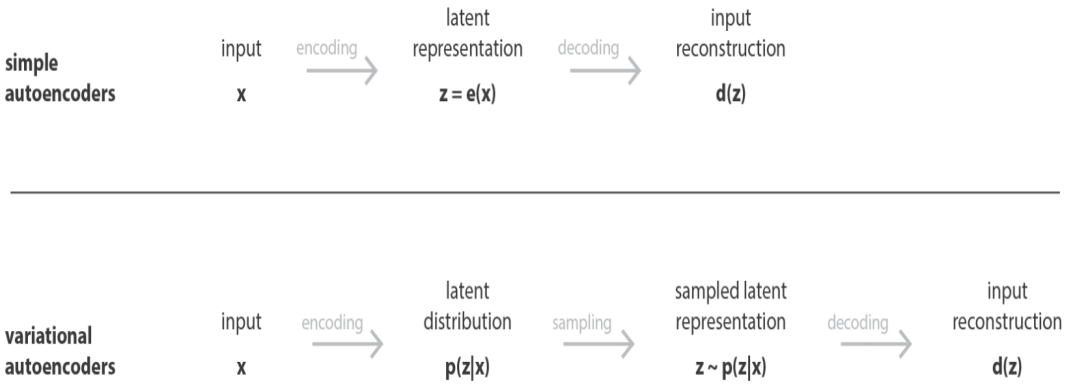

Figure 3.2.4: Difference between a vanilla autoencoder and a Variational Autoencoders (VAE).[98]

the input is of no use. Rather, if the dimension of the code is less than that of the input, then it ensures that the autoencoder learns those representative features of the input data which are most salient[30]. Convolutional autoencoders are an extension of traditional autoencoders where convolution layers are used as building blocks in both the encoder and the decoder part of the autoencoder. After training the network, the encoder part is used for extracting the features of the data and the decoder part is used for the reconstruction of the input data.

Regularized autoencoders are a special type of autoencoders that employs one or more regularization techniques to prevent the model from overfitting. The different types of regularizations used are L1 or L2 Regularization which add a penalty to the loss function depending on L1 or L2 norm of the model weights. As a result of this, the model is capable of learning sparse representation of the training data where many weights are set to a very small value or zero as shown in Fig3.2.3. Dropout is also used as a regularization technique where a percentage of the model's nodes are set to zero during training so that the model doesn't rely heavily on the weight of any particular node, thus, preventing overfitting and improving generalization on unseen data.

The previously mentioned regularization techniques give rise to a variation of regularized autoencoders called the sparse autoencoders[64]. Sometimes noise is also added to the training data or the hidden layers of the model to increase the robustness the model giving rise to denoizing autoencoders[101]. One more regularization technique is to apply a contractive regularization term based on Frobenius norm of the Jacobian matrix of the model's hidden layer activations with respect to the input data[80, 5]. This ensures that a small neighborhood of the input data corresponds to a small neighborhood in the latent space representation, which means small perturbations in the input data leads to small or zero variation in the latent space representation[80, 5]. By doing so, it makes the model more robust to small changes in the input data, thus, preventing overfitting.

VAEs are a distinct type of autoencoders which produces latent space representations that are continuous. This allows random sampling and interpolation for the generation of new datapoints. The difference between a vanilla autoencoder and a VAE is shown in Fig.3.2.4. Instead of producing a single vector for the latent space representation, it generates two vectors: a vector of means μ and a vector of standard deviations γ for all datapoints. An encoding is then sampled from a distribution with mean μ and standard deviation γ. Thus, even for the same input, i.e. when the mean and the standard deviation are the same, the sampled encoding would vary because of the involvement of the sampling procedure. The mean vector controls the position where the encoding of the input should have its centre and the standard deviation controls the area over which the sampled encoding is allowed to vary from the mean. As a result of this, the decoder learns to decode not just the encoding of a single point but also the points in the neighborhood and hence a continuous latent space representation is obtained. In order to ensure that the latent space representation satisfies that the nearby encoding are similar to each other on a local scale while also facilitating interpolation on a

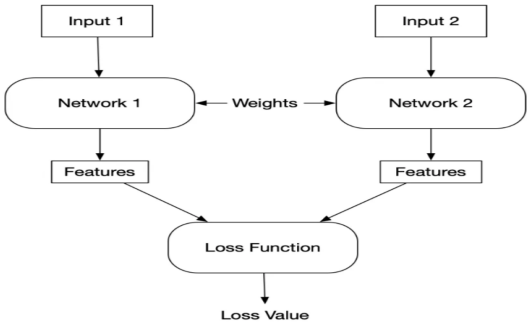

Figure 3.2.5: A generic Siamese Network.[1]

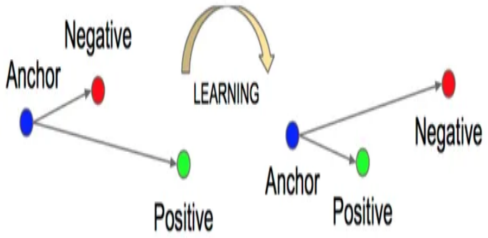

Figure 3.2.6: Before (left) and after (right) minimizing triplet loss function[96]

global scale VAEs jointly optimizes the reconstruction loss and the Kullback–Leibler divergence (KL)[49] loss.[44, 39]

3.2.3 Siamese Network

Traditional deep learning models have shown ground breaking results in a multitude of applications like image recognition and classification, web scraping, speech recognition and caption generation. But it has a limitation that the models tend to perform well if the training and testing data points are from the same distribution. If the model is to be made capable to predict datapoints belonging to an unknown class, it needs to be re-trained with an abundant number of training samples from that class and then predict it with unseen samples of that class. This technique can often be computationally expensive and can increase expotentially if the number of classes increases. Also generating a huge number of samples for a new class of objects can be tedious and quite challenging in multiple scenarios. The Siamese Network[46, 10] is one such way to mitigate the above mentioned problem. The idea behind Siamese Networks is that two identical copies of the neural network with shared weights process two distinct datapoints as shown in Fig.3.2.5. Because the networks have shared parameters, it ensures that two very similar datapoints cannot have two different locations in the feature maps as both the networks evaluate the same function. In Siamese Networks, the model learns a similarity function, the loss function, which plays the pivotal role in determining how similar or dissimilar the datapoints are to one another. The two main loss functions used in Siamese Networks are contrastive loss function[10] and triplet loss function[6]. Since the goal of a Siamese Network is not to perform classification task on datapoints, rather to compute the similarity between them, contrative loss functions are more suited for this task as compared to cross-entropy loss functions, as it is for classification tasks.[46]

Contrastive loss function evaluates how capable a Siamese Network is in deciding the similarity between two given datapoints as is given by Eq. 4[1, 46]

$$\mathcal{L}_{\text{con}}(Y, D_w) = (1-Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}max(0, m - D_w)^2, \tag{4}$$

where $\mathcal{L}_{\text{con}}$ is the calculated contrastive loss, D_w is the Euclidean distance between the outputs of the twin networks[1]. Y takes the value 0 or 1. If the two datapoints belong to the same class then Y takes the value 0, otherwise it takes the value 1[1]. max() is a function to choose the higher of the two values, 0 or $m - D_w$ where m is the margin and has a value greater than 0[1]. The usage of a margin in this equation ensures that the pair of datapoints which the dissimilar beyond this margin value are excluded while calculating the loss function[1]. The euclidean distance between the embeddings of the two datapoints are given by the Eq. 5.[1, 46]

$$D_w(X_1, X_2) = \sqrt[2]{\{G_w(X_1) - G_w(X_2)\}^2}, \tag{5}$$

where $G_w(X_i)$ is the output of the twin network such that $i \in \{1, 2\}$ as show in Fig.3.2.5.

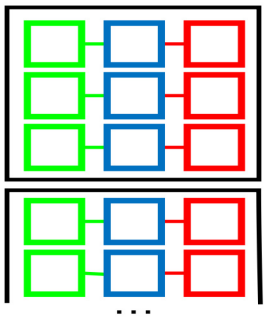

Figure 3.2.7: Minibatches in offline triplet mining. The green boxes are positive datapoints, the blue boxes are anchor datapoints and the red boxes are negative datapoints.[51]

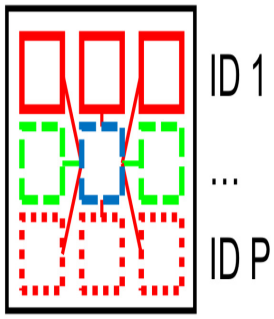

Figure 3.2.8: Triplets constructed only within the minibatch. There are P object classes in the minibatch. The green boxes are positive datapoints, the blue boxe is the anchor datapoint and the red boxes are negative datapoints.[51]

Triplet loss function[106] trains the Siamese Network in a way that a triplet of datapoints are presented to the Siamese Network. By applying triplet loss the model tries to group the datapoints similar to the anchor datapoint, close to one another while increasing the distance between the anchor datapoint and the datapoints dissimilar to it as shown in Fig.3.2.6. In order to do so, a positive value called margin is used which increases the distance between dissimilar datapoints and also eliminates the output of any trivial solution. Euclidean distance or cosine distance is used as the distance metric while calculating the triplet loss and shown in Eq. 6. [36, 1]

$$\mathcal{L}_{\text{tri}}(a,p,n) = \sum_{\substack{a,p,n \\ y_a = y_p \,!= y_n}} max(0, d(a,p) - d(a,n) + margin), \tag{6}$$

where y_x is the class label for the datapoint x for $x \in \{a,p,n\}$, a is the anchor datapoint, p is the positive example having the same class label as the anchor datapoint and n is the negative example having a different class label as compared to the anchor datapoint. But there are some practical problems related to the use of triplet loss function. The number of possible triplets grow cubically with the increase in the size of the dataset. Also most of the triplets are often uninformative. Thus, mining "hard triplets" or the ones crucial for the learning of the model becomes a challenging task. As per the definition of the triplet loss function, the triplets can be of three types: easy triplets, hard triplets and medium hard triplets. The triplets which satisfy the condition $d(a,p) + margin < d(a,n)$ results to the loss value being 0, hence are called the easy triplets[96]. For the triplet for which the negative datapoint is closer to the anchor as compared to the positive datapoint i.e. $d(a,n) < d(a,p)$ are the examples of hard triplets.[96] And the medium hard triplets are the ones where the negative datapoints are not closer to the anchor datapoint as compared to the positive datapoint but still results in positive loss i.e. $d(a,p) < d(a,n) < d(a,p) + margin$.[96] Thus, the medium hard triplets are the most essential for the model training as they constitute for the most useful information. Two triplet mining procedures are used to mitigate the issue of excessive computational overhead with increase in dataset size. They are offline triplet mining and online triplet mining. In offline hard triplet mining, at first the dataset is manually processed to find hard triplets and then they are used for training the model. But manually finding the hard triplets can be quite tedious. Online triplet mining is a better approach as compared to the one mentioned above. The model is trained in minibatches of the dataset as shown in Fig.3.2.7. Using only the triplets that were mined from the dataset could be a wasteful design choice in such scenario. So in a minibatch, it contains way more potential triplets than the ones that were mined. So each member of another triplet becomes an additional negative datapoint example. But both hard positive and hard negative datapoints are required for training. An even better design to mitigate that would be to choose let's say c datapoints from P different classes in the dataset, where the k positive examples serve as hard positives while the rest of the datapoints in the minibatch serve as hard negatives as shown in Fig.3.2.8.[36]

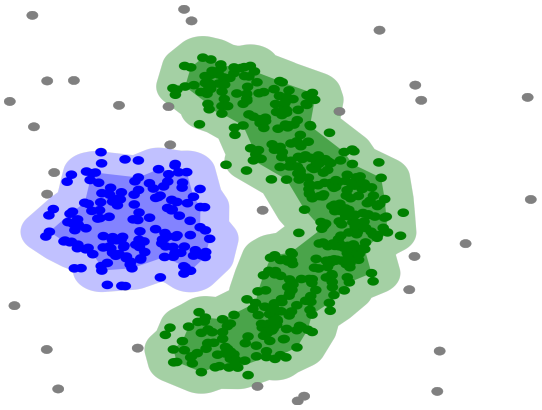

Figure 3.3.1: DBSCAN can find non-linearly separable non-convex clusters.[20]

Thus, Siamese Networks are very robust to the class imbalance in dataset which is very common in real datasets. Since it depends on very few datapoints of a particular object class, it has good generalization capabilities for unseen datapoints in the future. They are also very robust to small perturbations in the datapoints due to difference in lighting conditions orientation or background, as they keep similar objects close to each other even if the datapoints are slightly different. Since the Siamese Networks places similar datapoints together while increasing the distance between dissimilar datapoints, they are often useful for learning semantic similarity within the datapoints. However despite its benefit, training a Siamese Network takes longer as compared to traditional neural networks because it involves learning from quadratic pair of datapoints.[1, 46]

3.3 Clustering Algorithms

When the dataset does not have any labelled data, then it is said to be unsupervised learning. In this case, there is no ground truth available to measure the correctness of the outputs generated by the Machine Learning (ML) models. The primary focus of unsupervised learning is to find hidden and interesting pattern in the data. Unsupervised learning is of utmost importance in the Artificial Intelligence (AI) world as several real world datasets do not have available annotations, which requires a lot of human effort. Unsupervised learning algorithms can be broadly categorized into the following: domains-clustering, dimensionality reduction and association analysis. Clustering algorithms aim at grouping unlabelled data into groups or clusters based on how similar or dissimilar the datapoints are to one another. It can reveal underlying hidden pattern in the data and is used in applications like image segmentation, fraud detection, etc.

Dimensionality reduction reduces the number of irrelavant features or dimensions of the dataset. The inclusion of more features does help in the better representation of the dataset but it also significantly increases the memory consumption and complexity to work with it. Also it is often difficult to visualize real-life datasets with too many features. Association analysis is a rule-based unsupervised learning method that reveals the relationship between attributes in the dataset. It is used in applications like market analysis, intrusion detection, etc. Clustering algorithms can be broadly classified into the following categories: density-based, distribution-based, graph-theory based and hierarchical-based.

3.3.1 Density-Based Clustering

In density-based clustering, the algorithm looks for areas of high concentration of the datapoints and groups them as a cluster. The benefit of this algorithm is that the shape of a cluster is not limited and hence doesn't necessarily have to be convex in nature as show in Fig. 3.3.1. Moreover, these algorithms are also very robust to outliers as they do not force the outliers to belong to any category but are rather ignored as it is unlikely that outliers can form an area of high concentration of datapoints. The experiments on this thesis

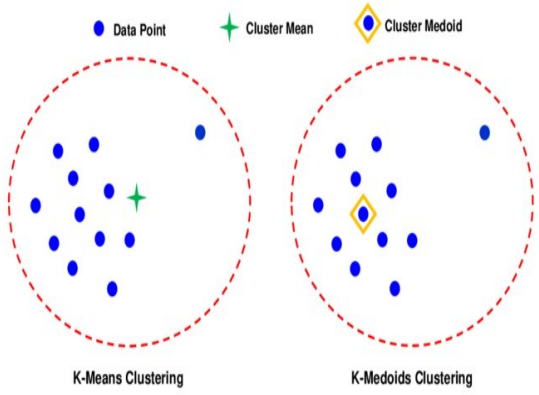

Figure 3.3.2: Difference between k-means and k-medoids.[25]

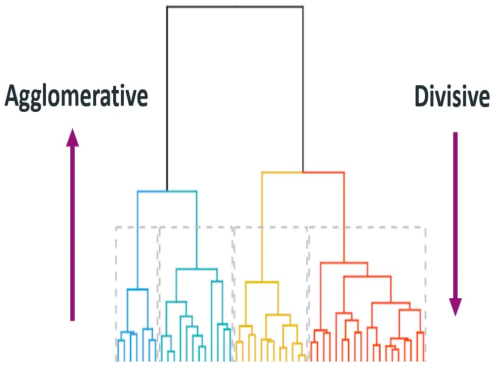

Figure 3.3.3: A dendogram in hierarchical clustering.[37]

have been carried on two such density-based clustering algorithms called DBSCAN[26] and HDBSCAN algorithms[58]. The available scikit-learn python library implementations were used for this purpose[71]. The DBSCAN[26] algorithm does not require the users to define the number of clusters to be generated which doesn't force dissimilar datapoints to belong to the same cluster. However, this algorithm was still sensitive to two parameters ε, the maximum distance between the datapoints to be considered in the same cluster and the minimum number of samples in the cluster which the user needs to define [71]. But finding an optimum value for these parameters often require domain expertise and are dependent on the data. The HDBSCAN[58] algorithm mitigates this issue and thus, does not require the user to define these two parameters. It is a hierarchical density-based clustering that performs the DBSCAN algorithm over multiple ε values to find the most stable result.

3.3.2 Distribution-Based Clustering

In distribution-based algorithms, a datapoint is said to be a member of the cluster depending on the probability of its membership to the cluster. The more the distance of a point increases from the centre of the cluster, the less is its probability of belonging to that cluster. Centroid-based algorithms groups the datapoints based on some initial cluster centres. Once all the datapoints are softly assigned to some cluster membership, the cluster centres are recalculated and this process is iterated until convergence. These algorithms are sensitive to the initial parameters like the cluster centres chosen in the first step. Another major disadvantage of these clustering algorithms are that they always form spherical clusters. The user also needs to define the number of clusters the dataset is to be grouped in, which makes it sensitive to outliers. However, these algorithms can be executed very fast and two such algorithms were used during the experiments in this thesis- k-means[42]and k-medoids[43]. In k-means, the mean of the datapoints of the clusters is assigned as the cluster-centroid. It might not be an actual datapoint in the dataset, rather a blurred, noisy average of a datapoints in the cluster. On the contrary, the k-medoids algorithm assigns an actual datapoint of the dataset, that is most centrally located as the cluster centroid as shown in Fig.3.3.2. Thus, k-medoids is more robust to outliers and noises as compared to k-means.[3]

3.3.3 Hierarchical-Based Clustering

Hierarchical-based clustering algorithms form a hierarchy of clusters. Datapoints in a cluster are more similar to each other as compared to other groups. This hierarchy of clusters is visualized by a hierarchy tree called dendograms. Hierarchical clustering algorithms can be of two types - agglomerative and divisive. In agglomerative clustering, each datapoint is considered as a separate cluster in the first step. Then these clusters are merged into one another until only one cluster remains. Thus, at the end, the last level cluster consists of all the datapoints in the dataset. The divisive method is the reverse procedure of the agglomerative method. In the beginning, all the datapoints are considered to be in a single cluster and gradually the cluster is broken into smaller clusters, until each cluster consists of only one datapoint. A visual

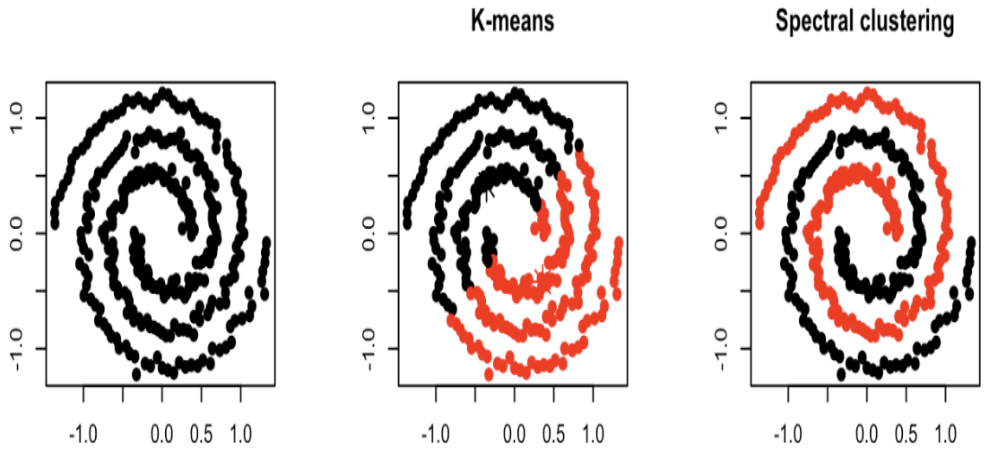

Figure 3.3.4: A comparison between the results of k-means clustering and spectral clustering. The image in the left shows the points that need to be clustered.[67]

representation of a dendogram has been shown in Fig.3.3.3.[62]

3.3.4 Graph Theory-Based Clustering

Spectral clustering is a powerful machine learning tool which divides datapoints into meaningful groups based on their similarity. The concept of spectral clustering has its root in the graph theory. It tries to determine the node communities in a graph by looking at the edges that connect them. It is specially useful when the dataset cannot be linearly-separated. The affinity matrix of the datapoints, which gauges the pairwise similarity between points, is the basis for spectral clustering. It works by creating a graph representation of the data, in which datapoints are represented by nodes and the degree of similarity between the points indicated by edges. After that the eigenvectors of a matrix is obtained from this graph which is used to create the embeddings of the datapoints in a lower dimensional space. The embedded points are then divided into clusters by using a common clustering algorithm, usually the distributed based clustering algorithm k-means, where each entry in the embedding matrix is the representation of the datapoints in the lower dimensional space. Spectral clustering has shown promising results with datasts that have complex structures and non-linear separability, which makes it a useful tool in varity of fields, such as social network analysis, image segmentation and dimensionality reduction[103]. A comparison between the results of k-means and spectral clustering in the presence of non-globular clusters in shown in Fig. 3.3.4.[21]

3.4 Evaluation Metrics for Clustering Algorithms

The ground truth or labels are available in supervised learning. So to determine the quality of the clustering algorithm the level of accuracy is calculated by comparing the labels predicted by a model to the ground truth. But most real world datasets do not have a ground truth. Therefore, it makes it challenging in unsupervised learning paradigm to determine the quality of the clustering algorithm. A good clustering algorithm is still expected to have little difference between the objects in a cluster i.e. small within-cluster variance and more difference between objects from different clusters i.e. large between-cluster variance. Thus, the evaluation metrics used in the literature could be categorized into two broad groups: extrinsic measures and intrinsic measures, depending on the availability of the ground truth. Extrinsic measures, when the ground truth is available uses metrices like Rand Index, Mutual Information, V-measure and Fowlkes-Mallows Scores. Intrinsic measures, when the ground truth is not available mostly uses Silhouette Score, Calinski Harabaz index and David Bouldin index.

3.4.1 Silhouette Score

Silhouette Score is used to calculate the distance between points within a cluster as compared to its distance from the points in the neighbouring cluster and is given by Eq. 7.

$$S_c = \frac{(n_c - i_c)}{max(n_c, i_c)}, \tag{7}$$

where S_c is the Silhouette Coefficient, n_c is the average of the nearest-cluster distance of each datapoint and i_c is the average distance between the datapoint within the cluster. The Silhouette coefficient lies within the range $[-1, 1]$, where the the more closer the coefficient is to 1, the better is the clustering results. When the Silhouette Coefficient is close to 0, it implies that the datapoints are on or very close to the decision boundary between adjacent clusters, whereas negative values imply that the datapoints have been wrongly classified. But the drawbacks of using this metric is, that is tends to favour dense and convex clusters. Since it takes into account the average of the distance within a cluster as well as between adjacent clusters, it doesn't perform well if different clusters have a significant difference in their respective sizes, which is quite a common phenomenon in real world datasets. The chosen distance metric for the calculation of the Silhouette score also has a significant effect on it and can lead to very different results.

3.4.2 Calinski-Harabasz Index

Calinski-Harabasz Index(C-H Index) or the Variance Ratio Criterion is used to measure the dispersion between the datapoints within a cluster as compared to in-between clusters as in given by Eq. 8.

$$S_{CH} = \frac{tr(B_k)}{tr(W_k)} \times \frac{(n_E - k)}{k - 1}, \tag{8}$$

where S_{CH} is the C-H Index, $tr(B_k)$ is the trace of the in-between cluster dispersion matrix. B_k is in Eq. 9 and $tr(W_k)$ is the trace of the within-cluster dispersion matrix. W_k is shown in Eq. 10.

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_o)(c_q - c_o)^T, \tag{9}$$

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T, \tag{10}$$

where, C_q is the collection of points in cluster q, c_q is the centroid of the cluster q and c_o is the overall centroid of the datapoints. B_k calculates the quality of the separation between two clusters, i.e. higher the value of B_k, better it is. W_k calculates the distance of a datapoint (x) from the centroid of the cluster and thus, measures the cohesiveness or compactness of the datapoints within a cluster, i.e smaller the value of W_k the better it is. Distribution-based clustering algorithms like k-means tries to minimize the value of W_k. Thus, according to the equation of C-H Index in Eq. 8, the higher the C-H index, the better is the clustering results. Because sum-of squared Euclidean distance is used to calculate the distance of the datapoints from their respective cluster centroids, C-H Index tends to favour convex clusters, which is a major drawback of using this metric similar to that in Silhouette Coefficient.[12]

3.4.3 David-Bouldin Index

David-Bouldin Index is another metric used for evaluating clustering algorithms, which uses only the intrinsic features of the dataset and doesn't require any ground truth labels. David and Bouldin proposed R,a measure to calculate cluster similarity that satisfied the following conditions[100]:

- $R(S_i, S_j, M_{ij}) \geq 0$
- $R(S_i, S_j, M_{ij}) = R(Sj, S_i, M_{ij})$
- $R(S_i, S_j, M_{ij}) = 0$ iff $S_i = S_j$

- if $S_i = S_j$ and $M_{ij} < M_{ik}$, then $R(S_i, S_j, M_{ij}) > R(S_i, S_k, M_{ik})$
- if $Mij = M_{ik}$ and $Sj > Si$, then $R(Si, Sj, M_{ij}) > R(Si, Sk, M_{ik})$

where M_{ij} is the distance between the centroids of the clusters i and j and S_i and S_j are their respective dispersion measures. The conditions enforce that the cluster similarity measure R is non-negative and symmetrical in nature. It becomes 0 if and only if the dispersions of the two clusters disappear. If the distance M_{ij} increases even if S_i, S_j are constant, then the cluster similarity measure R decreases and vice versa. Also if M_{ij} remains constant and S_i, S_j increases, then R increases. Thus, the David-Bouldin Index was defined as the average of the similarities of each cluster as in Eq. 11, using the same notations as the authors.

$$\bar{R} = \frac{1}{C} \sum_{i=1}^{C} R_i, \tag{11}$$

where C is the total number of clusters and R_i is the $max_{j \neq i} R_{i,j}$. Thus, $\bar{R}$ or the David-Bouldin Index is the average of the similarity measures R of each cluster with its most similar cluster. Thus, David-Bouldin Index has a lower bound of 0, while a higher value means bad clustering. In other words, the less is the average similarity, the better is the clustering results. Because of this, it acts as a good measure to decide on the number of clusters that actually exists in the dataset. Therefore, this index can be used to determine the number of clusters in distribution-based clustering algorithms like k-means. $R_{i,j}$ is more formally defined as in Eq. 12.

$$R_{i,j} = \frac{S_i + S_j}{M_{ij}}. \tag{12}$$

The dispersion measure of a cluster i is defined by S_I as in Eq. 13.

$$S_i = \left\{ \frac{1}{N_i} \sum_{j=1}^{N_i} \|X_j - A_i\|^q \right\}^{\frac{1}{q}}, \tag{13}$$

where N_i are the number of datapoints in the cluster i and X_j are the respective vector representations of the datapoints in the cluster i. For $q = 1$, The dispersion measure S_i is the average Euclidean distance of the vector representation of the datapoints from the centroid of the cluster i, while for $q = 2$, it is the standard deviation of the metric about the datapoints in the cluster with respect to its centroid [100]. It is different as compared to the distance defined between the centroids of the clusters which are defined as in Eq. 14.

$$M_{i,j} = \left\{ \sum_{k=1}^{N} \|a_{ki} - a_{kj},\|^p \right\}^{\frac{1}{p}} \tag{14}$$

where a_{ki} is the k-th component of the vector representation of a_i which is the centroid of the cluster i. $M_{i,j}$ is the Minkowski metric of the centroids of the clusters i and j, i.e. when $p = 1$, it is the Manhattan distance and when $p = 2$ it is the euclidean distance between the centroids. When both p and q are 2, then $R_{i,j}$ is the Fisher similarity measure between the two clusters [100]. But this metric too is not without its limitations. It is not robust to noise and outliers, thus, giving an incorrect sense of good clustering. This index too is biased towards spherical clusters with similar sizes, thereby making it inappropriate for many real world applications. Moreover, since it is a metric of the clustering results on a global level and gives no information about the individual clusters, even a single "too good" or "too bad" cluster can heavily influence the final result.

3.5 Transfer Learning

Transfer learning is an improvement in machine learning techniques where a model trained for a particular task. Let's say task 1 is different from another task, task 2. It uses the knowledge learnt by the model in the previous task to be adapted for the new task. Transfer learning has its benefits in a multitude of applications because real world problems doesn't necessarily always have an abundant amount of labelled data which can be used to train a model from scratch [107]. It can be used for tasks which intend to solve different but related problems [107]. In other words, if the domain of the data for task 2 is similar to that

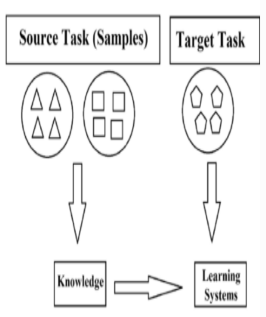

Figure 3.5.1: Model trained for a task in the similar domain is re-used for a new task.[38]

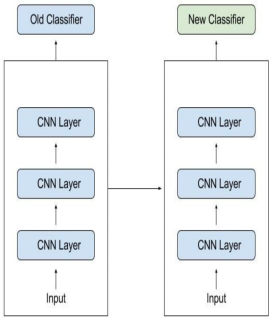

Figure 3.5.2: A model trained on a different task is adapted to be used for a new task.[109]

of task 1, transfer learning can be used as shown in Fig. 3.5.1. Several approaches have been adapted to use transfer learning. Let's say one wants to solve task T_1 but don't have adequate data to train a deep neural network for it. But there is ample data to solve a similar task T_2. So one can train a model for task T_2 and then use it for task T_1. It might be needed to re-train only the later layers or all the layers of the model but that heavily depends on the problem at hand. Another approach that could be adapted is to use an existing pre-trained network. There are a lot of available open-source models which can be re-trained and re-used depending on the problem. Some of the popular models that are used as pre-trained models in transfer learning are Oxford VGG Model [90],Google Inception model [94] and Microsoft ResNet model [35]. Another approach could be to use transfer learning for the purpose of feature extraction. Transfer learning has found a lot of application in the domain of computer vision and natural language processing because it requires a lot of data for training such complex models from scratch. In the domain of computer vision, it is known that the early layers of a deep neural network focuses on learning the low level features like the detection of edges, while the middle layers focus on learning the mid level features like the detection of shapes. In transfer learning, these pre-trained early and middle layers generally referred to as backbones could be used verbatim and only the latter layers which focus on the high level features could be re-trained for the new task as shown in fig 3.5.2. Transfer learning has numerous advantages like significant reduction in the amount of training time[56], improvement in performance for some neural networks[69] and less reliance on a huge amount of data[24, 48, 119].

3.5.1 Definition

According to [107, 68], transfer learning can be formally defined as the following. A domain $\mathcal{D}$ of the input data, with the feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$ such that $X = \{x_1, ..., x_n\} \in \mathcal{X}$, where n is the number of feature vectors in X. In the given input data domain $\mathcal{D}$, the task $\mathcal{T}$ has a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$ which is learned from the feature vector and label pairs (x_i, y_i) such that $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and $f(x)$ is the learner that predicts the label value for x. As per the above definition, the source domain is defined as $\mathcal{D}_\mathcal{S}$ and the corresponding source task as $\mathcal{T}_\mathcal{S}$.The target domain is defined as $\mathcal{D}_\mathcal{T}$ and the corresponding target task as $\mathcal{T}_\mathcal{T}$. Transfer learning aims at improving the target predictive function $f_\mathcal{T}(\cdot)$ by using the related information from $\mathcal{D}_\mathcal{S}$ and $\mathcal{T}_\mathcal{S}$ where $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$ or $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$. Since $\mathcal{D}_\mathcal{S} = \{\mathcal{X}_\mathcal{S}, P(X_\mathcal{S})\}$ and $\mathcal{D}_\mathcal{T} = \{\mathcal{X}_\mathcal{T}, P(X_\mathcal{T})\}$, here the condition $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$ means that $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ and/or $P(X_\mathcal{S}) \neq P(X_\mathcal{T})$. The scenario where $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ in the context of transfer learning is defined as heterogeneous transfer learning. The scenario where $\mathcal{X}_\mathcal{S} = \mathcal{X}_\mathcal{T}$ is defined as homogeneous transfer learning. The scenario where $P(X_\mathcal{S}) \neq P(X_\mathcal{T})$ means the source and the target data are from different domains, i.e. they belong to different marginal distributions. Transfer learning algorithms are not expected to give optimal results in ths case. Refering back to the definition of transfer learning, when $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$ and it was defined that $\mathcal{T} = \{(Y), P(Y|X)\}$. This scenario is possible when $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ and/or $P(Y_\mathcal{S}|X_\mathcal{S}) \neq P(Y_\mathcal{T}|X_\mathcal{T})$. The scenario where $P(Y_\mathcal{S}|X_\mathcal{S}) \neq P(Y_\mathcal{T}|X_\mathcal{T})$ implies that the conditional probability distributions between the source and target domains are different. The case $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ means the label space of the source and target domain are different.[107]

Transfer Learning Settings	Related Areas	Source Domain Labels	Target Domain Labels	Tasks
Inductive Transfer Learning	Multi-task learning	Available	Available	Regression, Classification
	Self-taught Learning	Unavailable	Available	
Transductive Transfer Learning	Domain Adaptation, Simple Selection bias, Co-variate Shift	Available	Unavailable	
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering, Dimensionality Reduction

Table 2: Different settings of transfer learning.[68]

3.5.2 Categories

As per the definition of transfer learning mentioned above, it can be broadly categorized into 3 groups: inductive transfer learning, transductive transfer learning and unsupervised transfer learning as mentioned in [68]. Depending on the availability of labels of the source and target tasks, these 3 categories are possible as shown in Tab. 2. Regardless of whether the source and target domains are the same or not, the target task in an inductive transfer learning situation is distinct from the source task. In this instance, in order to generate an objective predictive function $f_{\mathcal{T}}(\cdot)$ for usage in the target domain, some labeled data from the target domain are needed. Furthermore, inductive transfer learning can be broken down into two groups depending on the availability of source and target domain labels. When a lot of labeled data is available for the source domain, it is similar to multi-task learning. But the difference lies in the fact that the goal of inductive transfer learning is to attain good results in the target task only by using the knowledge obtained from the source task while multi-task learning endeavors to learn both the target and source tasks concurrently [68]. If the labels of the source data are not available, then it is similar to the self-taught learning as proposed by [78]. In such a scenario, the label space of the source data and the target data can vary which means that the knowledge obtained from the source domain cannot be utilized directly. This implies that basically the labels of the source domain are not available.[68]

On the other hand, in transductive transfer learning, the target task and the source task are same but the source domain and the target domain are different. Depending on the similarity of the feature space and the marginal probability distribution between the source and the target domain, transductive transfer learning can be classified into two groups. One scenario can be that the feature space of the source domain and that of the target domain are different i.e. $\mathcal{X}_{\mathcal{S}} \neq \mathcal{X}_{\mathcal{T}}$. Again, the feature space of the two domains can be same but the marginal probability distributions of the input data are different i.e. $P(X_{\mathcal{S}}) \neq P(X_{\mathcal{T}})$. Similar assumptions underlie the latter instance of the transductive transfer learning, which is related to sample selection bias [115], covariate shift [89] and domain adaptation for knowledge transfer in text categorization [19, 115].[68]

Lastly, in unsupervised transfer learning, the target task is different from the source task as it was in inductive transfer learning but the labels of neither the source domain nor the target domain are available. The goal of unsupervised transfer learning is to solve unsupervised learning problems in the target domain like density estimation, dimensionality reduction and clustering[18, 105]. The categorisation of transfer learning has been summarised in Fig.3.5.3.

3.6 Dimensionality Reduction

Reducing the number of features (i.e. dimensions) in a dataset without sacrificing its relevant attributes is known as dimensionality reduction [22]. This is equivalent to eliminating features that are superfluous, redundant or just noisy data, in order to build a model with fewer variables. Dimensionality reduction refers

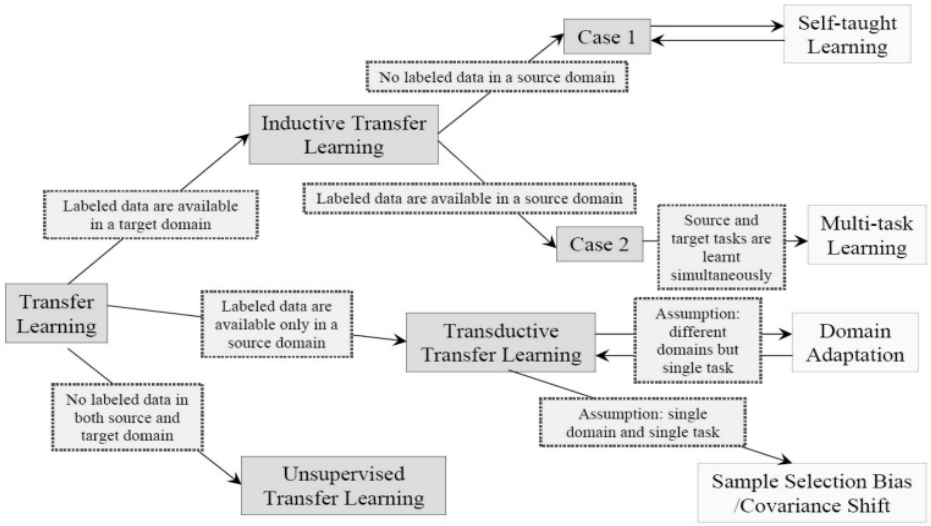

Figure 3.5.3: Overview of the different types of transfer learning.[68]

to a variety of preprocessing techniques for feature selection and data compression. Although dimensionality reduction techniques operate differently, they all use variable extraction or combination to convert high-dimensional spaces into low-dimensional spaces.[22]

Dimensions, often known as features or input variables, are predictor variables in machine learning that control a model's output. Any dataset having a significant number of predictor variables is regarded as high-dimensional data. Machine learning algorithms face several practical challenges when dealing with high-dimensional datasets, including longer calculation times and larger data storage requirements. Perhaps the greatest worry though, is that predictive models are becoming less accurate. Models in machine learning that are trained on high-dimensional datasets frequently have poor generalization.[22]

Curse of dimensionality The inverse link between growing model dimensions and declining generalizability is known as the "curse of dimensionality". The model space rises with the amount of model input variables. Conversely, sparse data results from a constant amount of data points. This indicates that much of the feature space of the model is empty or devoid of observable data points. Data points get so different from one another as data sparsity rises that prediction models are less successful in finding explanatory patterns [30]. Models may overfit on training data in order to explain patterns in sparse data sufficiently. Thus, poor generalizability may result from increases in dimensionality. By causing multicollinearity, high dimensionality might further impede model interpretability [7]. The effect of the number of model parameters on the performance of the model is shown in Fig. 3.6.1. There is a chance that certain variables are unnecessary or associated as the number of model variables rises. The curse of dimensionality can be mitigated by increasing the collection of data by reducing data sparsity. However, the number of data points required to counteract the curse of dimensionality grows exponentially with the number of dimensions in a model. Its not always possible of course, to gather enough data. Therefore, dimensionality reduction is necessary to enhance data analysis. Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are dimensionality reduction algorithms that improve machine learning models.[22]

3.6.1 Linear Discriminant Analysis

In supervised machine learning, LDA is a method used to resolve multi-class classification problems. It reduces the dimensionality of the data to enable the separation of several classes with numerous features. A generative model framework is used in LDA, which is often referred to as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA). In other words, LDA algorithms estimate the distribu-

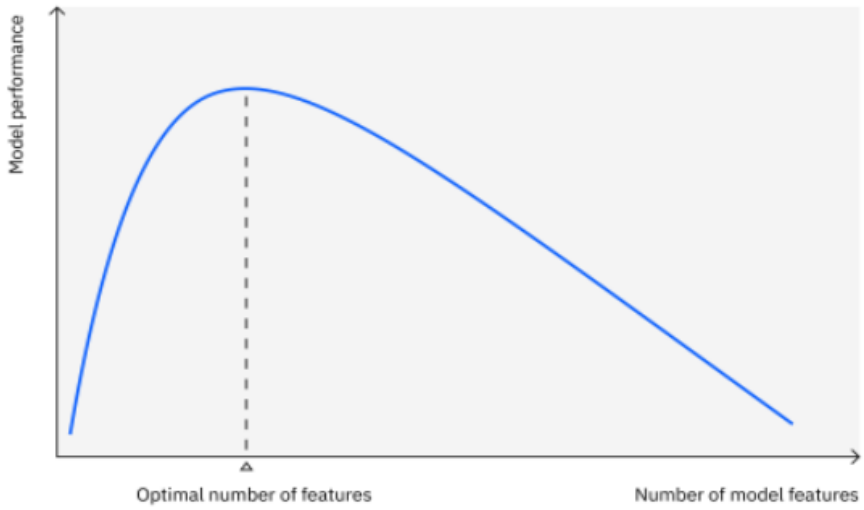

Figure 3.6.1: The effect of the number of model parameters on the model performance.[108]

tion of data for each class and classify additional data points by applying Bayes' Theorem [7]. Conditional probabilities or the likelihood of an event given the occurrence of another event are computed by Bayes Theorem. It is used by LDA algorithms to determine the likelihood that a particular data item will be included in a specific output. Finding a linear feature combination that distinguishes any number of categories of objects or events is how LDA operates. To facilitate easier classification, LDA projects data with at least two dimensions into a single dimension as shown in Fig. 3.6.2. As a result, the method is occasionally

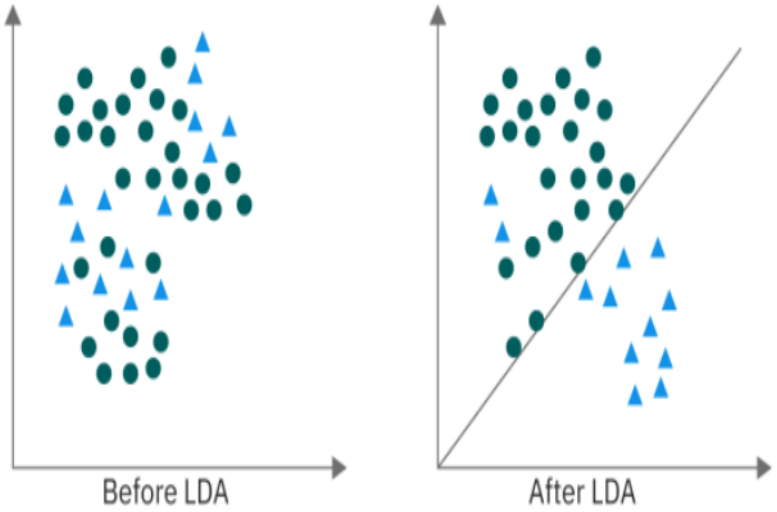

Figure 3.6.2: Projection of the data before and after applying LDA.[108]

called "dimensionality reduction". Unlike logistic regression for example, which is restricted to binary classification, LDA's adaptability guarantees that it may be applied to multi-class data classification issues. For this reason, it is frequently used to improve the performance of other machine learning classification methods, such as Support Vector Machines (SVM), random forests and decision trees.[111]

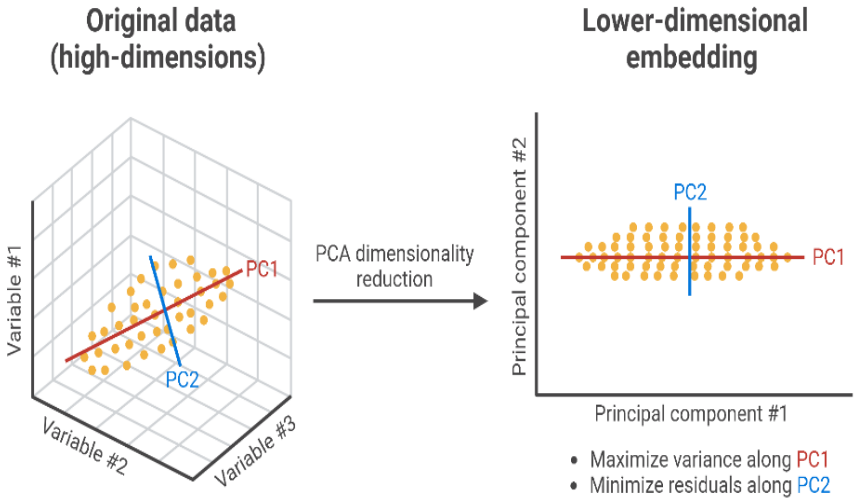

Figure 3.6.3: Projection of the data from higher dimensional space to lower dimensional space on applying PCA[75]

3.6.2 Principal Component Analysis

Similar to LDA, PCA is also a dimension reduction method. Unlike LDA, it is not restricted to problems involving supervised learning. In other words, PCA can reduce dimensions for unsupervised learning tasks without taking class labels or categories into account. Mathematician Karl Pearson first presented the PCA approach in 1901 [82]. It functions under the requirement that the variance of the data in the space of fewer dimensions should be maximal even when data in a space with more dimensions is transformed to data in a lower dimensional space as shown in Fig. 3.6.3. In an unsupervised learning paradigm it can find the correlation between different features in the dataset. A linear combination of the the variables in the original dataset gives rise to the principal components. By combining the variables in this manner, the majority of the information found in the original variables is condensed into the first components and these new variables or principal components are left uncorrelated. The premise is that ten-dimensional data sets provide ten principal components. PCA then attempts to assign the maximum amount of information to the first principal component, the maximum amount of information left to the second principal component and so on, until the result is somewhat like the Fig. 3.6.4. By removing the components with little information and using the ones that remain as the new variables, the dimensionality can be minimized without compromising much information when the data is organized using principle components. Its crucial to understand that the primary components are created as linear combinations of the original variables, they are less interpretable and have no true significance. From a geometric perspective, the principal components are the data's directions that account for the greatest amount of variance or the lines that encompass the majority of the data's information. Here, variation and information are related in that a line's variance indicates how dispersed the data points are along it. The more dispersion a line has, the more data it contains. To put all of this into perspective, principle components are new vectors that offer the optimal viewpoint for examining and assessing the data, making the variations between the observations easier to observe.[2]

Working priniciple for PCA

1. **Normalization** This step's objective is to normalize the continuous original variable's range to ensure each one makes an equal contribution to the analysis. More precisely, the reason PCA is extremely sensitive to the variances of the original variables is because normalization must be done first. In other words, if the ranges of the initial variables differ significantly, the variables with greater ranges will take precedence over the variables with smaller ranges, which will produce biased results. Therefore, this issue can be avoided by translating the data to similar scales. The mathematical procedure for

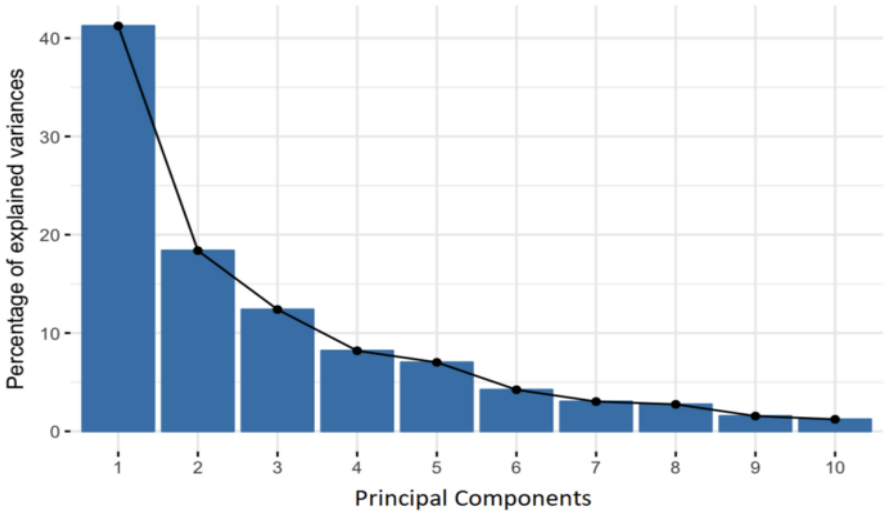

Figure 3.6.4: The percentage of explained variance for each principal component

achieving this is to take each variable's value, subtract the mean and then divide the result by the standard deviation as shown in Eq. 15.

$$z = \frac{value - mean}{standarddeviation}. \tag{15}$$

All of these variables will be converted to the same scale after standardization is complete.

2. **Computation of covariance matrix** This step's objective is to determine whether there is a correlation between the variables in the dataset being analyzed and how they differ from the mean in relation to one another. Because variables might occasionally include redundant information due to strong correlations, the matrix of covariance is calculated to find these correlations. The correlation matrix of a 3-D data with variables a, b, c are given by the covariance matrix shown below.[2]

$$\begin{bmatrix} Cov(a,a) & Cov(a,b) & Cov(a,c) \\ Cov(b,a) & Cov(a,b) & Cov(a,c) \\ Cov(c,a) & Cov(a,b) & Cov(a,c) \end{bmatrix} \tag{16}$$

The covariance matrix has all possible pairs of initial variables as entries and is a p × p symmetric matrix (where p is the number of dimensions). The variances of each starting variable are along the major diagonal (top left to bottom right), as the variance of a variable with itself is its covariance ($Cov(a,a) = Var(a)$). Furthermore, because the covariance is commutative ($Cov(a,b) = Cov(b,a)$), the covariance matrix's entries are symmetric with regard to the main diagonal, meaning that the top and lower triangular sections are equal. When two variables increase or decrease at the same time, i.e. they are correlated, then the value in the covariance matrix is positive. Whereas, if the two variables doesn't follow the same trend, i.e. one increases while the other decreases, then the variables are uncorrelated.[2]

3. **Computation of eigenvectors and eigenvalues** The two necessary mathematical concepts that are essential for the computation of the principal components are eigenvectors and eigenvalues. Every eigenvector has an associated eigenvalue with it. The number of eigenvectors or eigenvalues for the data is equivalent to the dimension of the data. The eigenvector gives the direction of the axis along which most of the variance of the original dataset is preserved or in other words the direction of the principal components. The eigenvalue associated with the eigenvector is the coefficient of the eigenvector, i.e. the amount of variance that particular principal component is responsible for in the total variance. The principal components are obtained by arranging the eigenvectors from highest to lowest of their corresponsing eigenvalues.[2]

4. **Creation of feature vector** Creating the feature vector involves selecting which elements to keep and which to remove. Generally speaking, components with low eigenvalues won't be as important. A plot is generated to display the cumulative proportion of variation as well as the percentage of total variance. Typically, the number of PCA components that is to be included is indicated by the point at which the Y axis of eigenvalues or total variance forms a "elbow".[2]

5. **Data transformation along the axis of principal components** With the exception of normalization, none of the data is altered in the preceding steps. Instead, the input data set is always expressed in terms of the original axes or the original variables. The primary components are chosen and the feature vector is created. The goal of this final step is to reorient the data from the original axis to that of the principal components, thus, the name Principal Components Analysis represent by utilizing the feature vector that was created using the eigenvectors of the covariance matrix. To accomplish this, the feature vector's transpose is multiplied by the original data set's transpose as shown in Eq. 17.[2]

$$TransformedDataSet = FeatureVector^T \times NormalizedOriginalDataSet^T. \tag{17}$$

4 Method

This chapter gives the details of the network, clustering algorithms and the evaluation metrics that are utilized and examined in this thesis.

4.1 Network

4.1.1 Selection of network

The unsupervised feature representation learning approach proposed by Guofeng Mei *et. al.*[59] is adapted to be used in the existing scenario at IPA. The primary motivation behind using an unsupervised learning methodology is that annotation of point clouds can be really challenging. To really focus on the point level features of a point cloud, it is required to have a high number of points in the point cloud. And labelling such high number of points in each point cloud in large real world datasets can be really expensive and severely inefficient. Unsupervised learning facilitates high quality feature representation without the need for humans in the loop. Moreover, sporadic and non-uniform distribution of the point clouds can also add further challenges in point-level annotation of the point clouds. Thus, in the absence of such annotations, it is impossible to get supervised feature representations for computer vision tasks.[59]

An unsupervised learning approach "ConClu" is proposed in [59] that learns both the local features as well as the global features of the point clouds. It is suitable to be used in this scenario, as it would require no point-level annotation of the dataset, which is unavailable in this case. To do so, it performed point-level clustering of the points clouds to generate semantically related regions within the point cloud and performed instance-level contrasting to make the network robust to its global appearance which would be explained in detail in the following subsections.[59]. Other related works like [120] also worked in the domain of learning object feature representations using autoencoders but depth images were used in such cases. [59] on the other hand worked with point clouds which is similar to the ABC dataset used at IPA. Furthermore, as mentioned in the Ch. 2, discriminative models have the capability of distinguishing between different data augmentations. Amongst them [79] and [81] is seen to have promising results because of the use of contrastive learning techniques. But it is computationally expensive generate negative samples required in contrastive learning in unsupervised learning. On the other hand, the mechanism of learning the global features which encapsulate the high-level semantic information reduces the necessity to rely on negative samples of contrastive learning. Therefore, the network used in [59] is deemed suitable for the task in hand.[59]

4.1.2 Architecture

Analysing how humans perceive an object, they do not focus merely on the individual points but rather semantically related points that form a group and act as the fundamental block for the entire object as a whole. The global features focus on the entire shape of the object while the local features attains to the more detailed information in the different parts of the object. In this work, the 3D point cloud $\mathbf{P} = \{\mathbf{p}_i \in \mathbb{R}^3 | i = 1,2,3,....,N\}$, where N is the number of points in each point cloud and $\mathbf{p}_i$ is each point in the point cloud representated in 3D cartesian coordinate system. The point cloud $\mathbf{P}$ is fed to an encoder backbone f_φ which generates a point-wise feature matrix $\mathbf{F} = \{f_{p_i}\}_{i=1}^{N}$ where f_{p_i} is the point-wise feature vector. The objective is to train a feature encoder f_φ, PointNet in this case, having parameters φ in an unsupervised learning paradigm. The entire pipeline of the process is elaborated in Fig.4.1.1. As it can be seen that the architecture has two main parts: the point-level clustering for learning the local features and the instance-level contrasting for learning the global features. Before a detailed analysis of each module is given, here is a brief overview of the entire process. The augmented views $\mathbf{P}^a$ and $\mathbf{P}^b$ are obtained for each point cloud $\mathbf{P}$ and are fed to the encoder f_φ such that it generates the feature matrices $\mathbf{F}^a$ and $\mathbf{F}^b$ respectively. Thus, the inputs to the point-level clustering are $\mathbf{P}^a$ and $\mathbf{F}^a$. It is to be noted here that the feature encoder f_φ shares the weights between the two augmented views. f_φ is trained in a way such that it minimizes the point-level clustering and the instance-level contrasting loss together. It is also to be noted here, that the encoder receives the gradient only from the top branch. After the training is completed, both the loses are cast aside and only the encoder is needed for further tasks.[59]

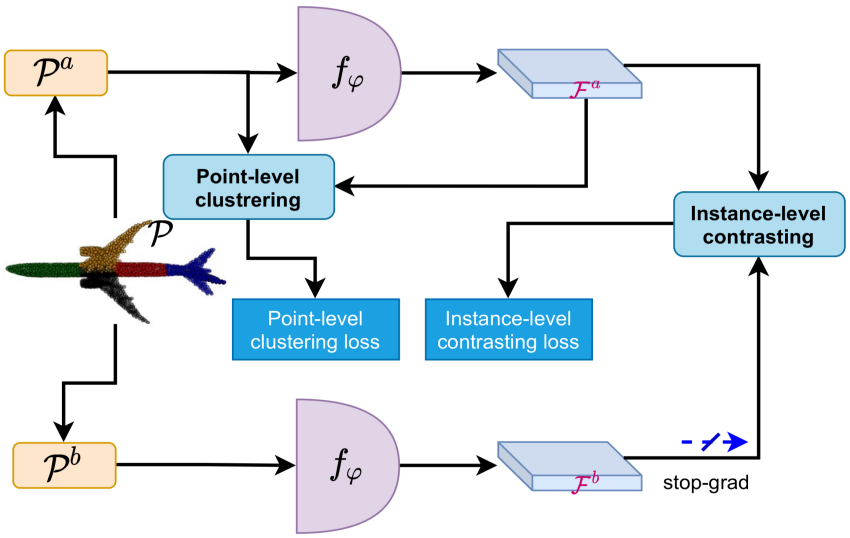

Figure 4.1.1: The ConClu model pipeline including the two modules: point -level clustering and instance-level contrasting.[59]

4.1.3 Encoder Backbone

Keeping in line with [59], two feature representation learning backbones are used for the experiments, the PointNet [76] and the DGCNN [104] backbone. The mechanism and the architecture of the feature extraction backbones are elaborated further in the following sections.

PointNet Autoencoder

For the purpose of weight sharing and other kernel optimization methods, traditional convolution networks need extremely regular formats for the input data as prevalent in RGB or depth images and 3D voxels. But point clouds are not regular data structures. It is just a set of 3D points which is permutation invariant to its constituent points. Or in other words, the order in which the individual points of a point cloud are processed is inconsequential. In [76] the authors suggested an end-to-end architecture that accepts the point clouds as input and generates the class label of the entire object or a class label for each of the individual points in the point cloud. The PointNet architecture is shown in Fig. 4.1.2. It consists of three important modules:

- a symmetric function to aggregate information,
- a module that integrates the local and the global features,
- an alignment network that orients the points of the point cloud and their features.

A further detailed overview of each of the building blocks are given in the following passages.

Symmetry function The points in the point cloud are an unordered set of points, which means the order in which they appear is irrelevant. Therefore, to make the network invariant to permutation, the following approaches could be adopted.

- The input points could be sorted in canonical manner. Sorting the points isn't an efficient solution because in high dimension space there isn't actually any stable ordering w.r.t the point perturbations. Contradiction is a simple way to demonstrate that. If such an ordering existed, it would mean there is a bijection mapping between the higher dimension space and a 1d real line. For this ordering to be stable, the bijection mapping would need to maintain spatial proximity even if the dimensions decreased in the event of point perturbations. This is not feasible in most cases. As a result, sorting only partially fixes the ordering problem and it brings additional challenges for the network to learn a consistent bijection mapping between the input and the output.[59]

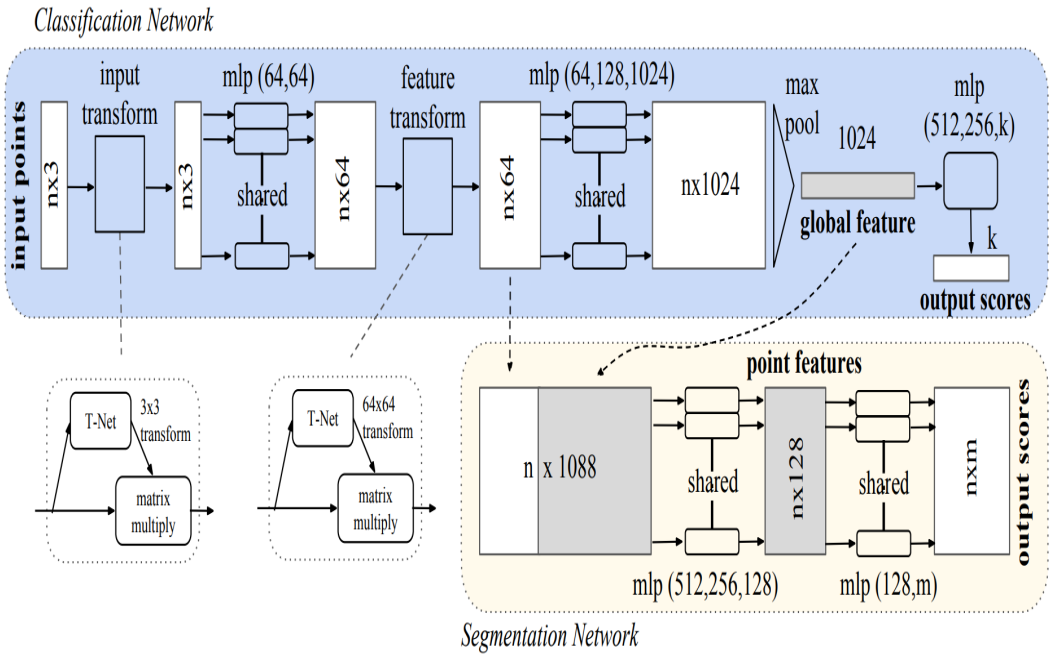

Figure 4.1.2: The PointNet Architecture.[76]

- The second approach is to treat the points as a sequential data and training a Recurrent Neural Network (RNN) with different randomly permuted input sequence in order to make it invariant to the order in which the input appears. But in [102] it is proved that the order indeed matters and cannot be overlooked completely. Moreover, even if a RNN is robust to the order of the input with small length(e.g. dozens), it suffers from scalibility issues. It can't be scaled to inputs with high number of elements as it is in point clouds.[59]
- The data from each point could be combined using a symmetric function. Binary addition and multiplication are some examples of symmetric functions. Hence, the idea proposed by the authors in [76] is to devise a symmetric function on the transformed set of points to approximate the general function defined on the points as in Eq. 18.[59]

$$f(\{x_1,....,x_n\}) \sim g(h(x_1),....,h(x_n)), \tag{18}$$

where $f : 2_{\mathbb{R}_N \to \mathbb{R}}, \mathbb{R}^N \to \mathbb{R}^K$ and $g : \underbrace{\mathbb{R}^K \times ... \times \mathbb{R}^K}_{\text{n}} \to \mathbb{R}$ is the symmetric function. Here h is a multi-layer perceptron network and g is a combination of a single variable function and a max pooling function. Even if this module is fairly simple, it has some of the very important implications. If C_S is the set of critical points, the points which contributed to the max pooled feature, the PointNet architecture is seen to be quite robust to the non-critical points. In other words, discarding the non-critical points don't lead to any loss of information about the global shape of the object as shown in Fig. 4.1.3.

Local and Global Information Aggregation The output of the symmetry function is a vector $[f_1,...,f_K]$ which is the global feature representation of the point cloud. But to get a point-wise segmentation, information about both the local features and the global features are necessary. The local features are learnt by the network in the *Segmentation Network* module as shown in Fig. 4.1.2. Once the global feature vector of the point cloud is calculated, it is concatenated with each of the point features and is fed into the network. This enables the network to learn new point-level features when it is informed about both the local and the global information. Because of this module, the network is able to predict the class probability of individual point based on both the local geometry of the object as well as the semantic region it belongs to on a global-level.[76]

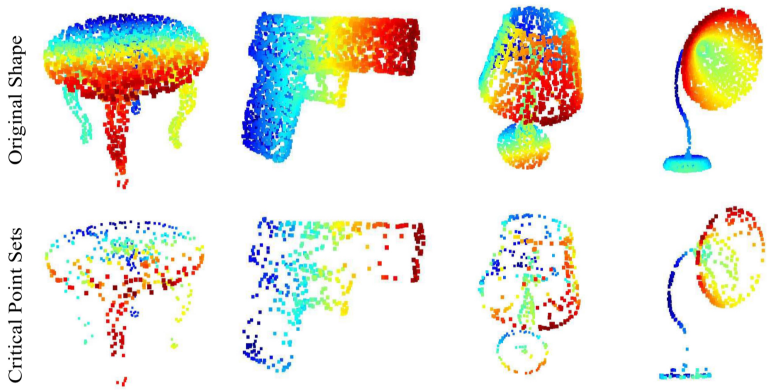

Figure 4.1.3: Critical points set and the original shape.[76]

Joint Alignment Network The network should be invariant to certain geometric transformations like rigid body transformations. A trivial solution to this issue is aligning every input set to a canonical order prior to feature extraction. But this would require an additional layers for aligning all input sets. On the contrary, the authors in [76] utilized a "mini-network" (*T-net*) as shown in Fig. 4.1.2. This predicts a transformation matrix which is then applied to the coordinates of the points in the point cloud. The mini-network takes after the main network and is made up of point independent feature extraction module, max pooling layers and the fully connected layers. To align the point-features from different point cloud sets, another alignment network could be introduced. It would predict the transformation matrix required for the alignment. But that would significantly increase the difficulty of optimization since the transformation matrix has much higher dimensions as compared to the spatial transformation matrix. To circumvent this issue, a regularization term is used in the softmax training loss. The constraint enforced in this case is that the feature transformation matrix is to be approximated by an orthogonal matrix as given in Eq. 19.

$$L_{reg} = \|I - AA_T\|_F^2, \tag{19}$$

where $\|.\|_F$ is the Frobenius-norm, I is the Identity matrix and A is the feature alignment matrix predicted by the mini-network. The feature transformation matrix has been approximated by an orthogonal matrix because orthogonal transformations ensure that no information about the input is lost.

Dynamic Graph Convolutional Neural Network

Another more general feature encoder is used in this work. It is seen that the PointNet architecture treated each point of the point cloud independently. The model is made invariant to the different possible permutations by applying a symmetric function to aggregate the features. But a limitation to this method is that it doesn't take into consideration the topology of the points. It ignores the connectedness in between the points and thus, cannot learn the local features. This limitation is addressed by Y. Wang *et al.* in [104]. The authors proposed the EdgeConv operation which is able to learn the local features of the point cloud while still meeting the requirement of being permutation invariant. EdgeConv generated the edge features which are capable of capturing the topological information of the points i.e. the connection between a point and its neighbors. In other words, the EdgeConv operation computes a local graph for each of the points in the point cloud which is capable of learning the edge embeddings for the edges in the graph. So the model group points based on both the Euclidean distance between the points and also how they are semantically related. The EdgeConv is a plug-and-play module that would be used combined into any existing model architecture to further enhance its performance. The approach has been further elaborated in the following passages.

Unlike PointNet, the DGCNN network captures the topological information by generating a local neighborhood graph and then applying an operation analogous to the convolution operation on the edges connecting a pair of points. It is in line to the methodology used in graph neural networks. But unlike a traditional

graph CNN, the local neighborhood graph is not fixed in this case. On the contrary, it is updated after every layer in the network. This is because the k-nearest neighbors of a point doesn't remain the same after every layer of the network. This happens as the k-nearest neighbors are based on the Euclidean distance between the points but rather of the proximity of the points based on the learnt embeddings. In other words, the proximity of the points in the feature space are different from the proximity of the input points in the Euclidean space. This is responsible for the non-local flow of information through the point cloud.

Edge Convolution In this work, the 3D point cloud $P = p_i \in \mathbb{R}^F | i = 1,2,3,....,N$ where N is the number of points in each point cloud and p_i is each point in the point cloud represented in 3D cartesian coordinate system where $p_i = (x_i, y_i, z_i)$. In this work $F = 3$ but it is also possible to include additional information about the coordinates such as surface normals, color, etc. Like most traditional deep neural networks, each layer acts upon the outputs of the previous layer. So generally speaking, F is the dimension of the features of a particular layer in the network. The architecture of the model used in this work is shown in Fig. 4.1.4. The top branch is used for the classification of the point cloud and the bottom branch is for the point-wise segmentation task. The classification task requires the n points in point cloud as input. k is pre-defined as the number of nearest neighbors that are to be considered for generating the edge features. The EdgeConv layer computes an edge feature set of size k for each point p_i in the point cloud. An aggregation operation is performed on each of these sets to get the EdgeConv response for the respective points. A global aggregation function is applied on the output of the last EdgeConv layer to generate a 1D global feature descriptor. This is used to compute the classification score for c classes. The point-wise segmentation branch further extends the classification branch by concatenating the 1D global descriptor with all EdgeConv responses (acts as local descriptors) for each point in the point cloud. Thus, it generates a point-wise classification score for p semantic classes that it can belong to. $\oplus$ is the concatenation operation. The spatial transform block is the module used to align the input point cloud to a canonical space. It is done by applying an estimated 3×3 transformation matrix. The coordinates of each of the points in the point cloud are concatenated with the difference in the coordinates with the k neighboring points giving rise to a tensor. This tensor is used in computing above-mentioned transformation matrix. The input point cloud of shape $n \times f$, here $(f = 3)$ is fed to the EdgeConv block which computes the edge features for each of the points in the point cloud. A Multi Layer Perceptron (MLP) is used for this purpose where the number of neurons in a layer is defined as $\{a_1, a_2, ..., a_n\}$. The output of this block is a tensor with shape $n \times a_n$ after performing a pooling operation on the neighboring edge features.

A directed graph $G = (V, E)$ which captures the topology of the point cloud where $V = \{p_i | i = \{1,2,...,n\}\}$ is the set of vertices and $E \subset V \times V$ is the set of edges. The graph G is the k-Nearest Neighbor (k-NN) graph of $\mathcal{P}$ in $\mathbb{R}^F$. The graph has self-loops which means, each point in the point cloud has an edge to itself. The edge features computed by the EdgeConv block is defined as $e_{ij} = h_\theta(p_i, p_j)$ where $h_\theta : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$ is a non-linear function with a set of trainable parameters θ. The EdgeConv operation is defined as a symmetric aggregation operation $\square$ which is applied on a channel wise manner. Summation or max are some of the examples of aggregation functions. It is applied on the edge features associated with all the edges that orginate from each connected vertex. Therefore, the output of the EdgeConv layer for i-th vertex is given by Eq. 20

$$p'_i = \underset{j:(i,j)\in E}{\square} h_\theta(p_i, p_j). \tag{20}$$

Drawing similarity with convolution on images, p_i is the central pixel and $\{p_j : (i,j) \in E\}$ is a patch around the central pixel as shown in Fig. 4.1.6. Therefore, for a point cloud with F dimensions and n points in it, the EdgeConv layer generates a point cloud with F' dimensions and n points in it.

Deciding the non-linear function h_θ and the symmetric aggregation function $\square$ plays a pivotal role in deciding the properties of the EdgeConv layer. To further illustrate on this point, if $p_1, ..., p_n$ are the points in the point cloud and the graph G encapsulates the topological information of patches of fixed size around each central pixel, then if $\theta_m \cdot p_j$ is chosen as the non-linear function and sum is chosen as the aggregation operation, then it would behave as a standard convolution operation as in Eq. 21.

$$p'_{im} = \sum_{j:(i,j)\in E} \theta_m \cdot p_j, \tag{21}$$

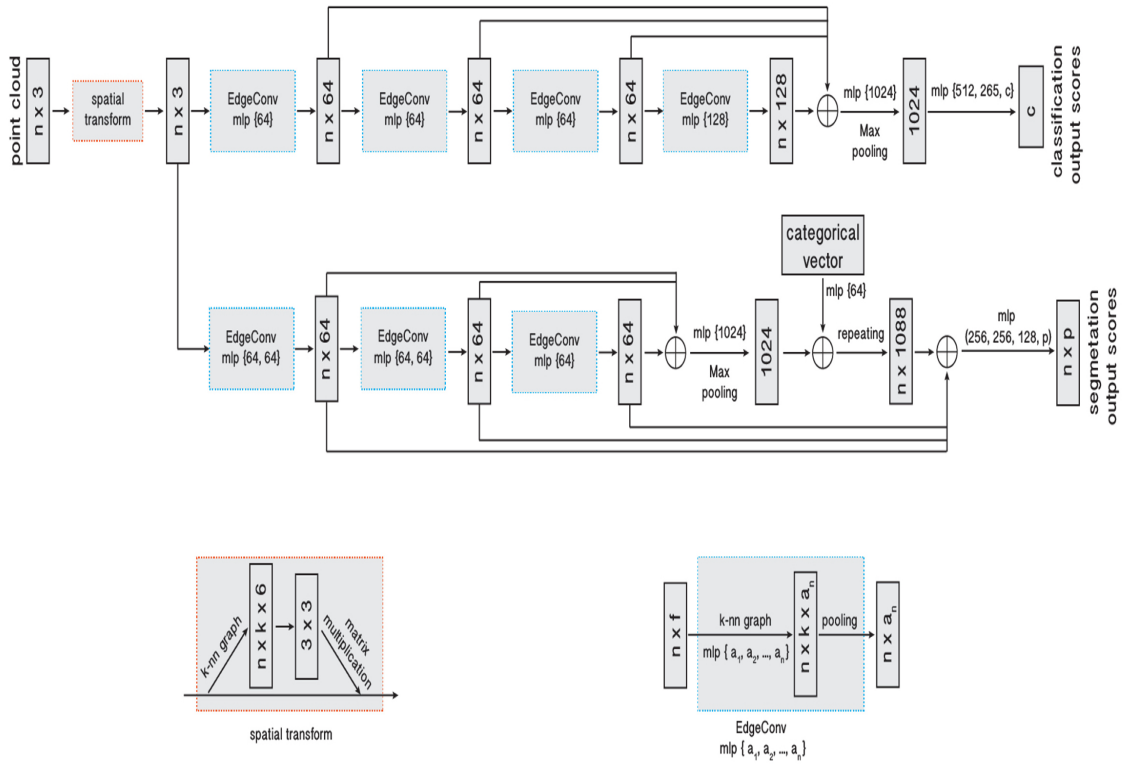

Figure 4.1.4: The DGCNN Architecture.[104]

where $\theta = (\theta_1, \theta_2, ..., \theta_M)$ are the weights of M different filters used in the convolution operation. Each entry in θ_m has the dimensionality as p_i and $\cdot$ is the Euclidean inner product. Another variation of h could be Eq. 22.

$$h_\theta(p_i, p_j) = h_\theta(p_i). \tag{22}$$

Here only the global embedding is taken into consideration and the non-linear function is unaware of the local features. This happens in PointNet and thus, it could be considered as a special case of EdgeConv. Another variation of h as adapted by Atzmon *et al.* in [4] is Eq. 23 and Eq. 24.

$$h_\theta(p_i, p_j) = h_\theta(p_j). \tag{23}$$

and

$$p'_{im} = \sum_{j \in V} (h_{\theta(p_j)}) g(u(p_i, p_j)). \tag{24}$$

Here g is a Gaussian distribution kernel and u calculates the pairwise Euclidean distance between the points in the point cloud. Another variation of h could be to only capture the local features and ignore the global features based on the fact that the global shape is a collection of the local patches as shown in the following Eq. 25.

$$h_\theta(p_i, p_j) = h_\theta(p_j - p_i). \tag{25}$$

The final variation of h which is utilized by the authors in [104] is an asymmetric edge function as shown in Eq. 26.

$$h_\theta(p_i, p_j) = \bar{h_\theta}(p_i, p_j - p_i). \tag{26}$$

This explicitly takes into account the global feature information as in Eq.22 captured by the coordinates of the central pixel of the patch p_i and the information of the local neighborhood as in Eq. 25 and is captured by $p_j - p_i$. The EdgeConv operator is defined as

$$e'_{ijm} = \text{ReLU}(\theta_m \cdot (p_j - p_i) + \phi_m \cdot p_i). \tag{27}$$

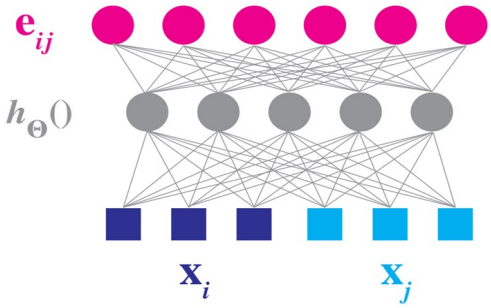

Figure 4.1.5: The Computation of edge feature e_{ij} from a pair of points p_i and p_j. Here, h_θ is a fully connected layer and the weights associated with it are the trainable parameters θ.[104]

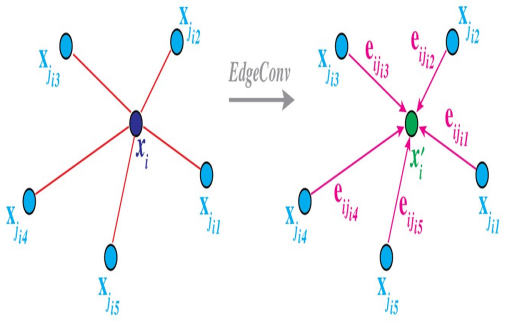

Figure 4.1.6: The EdgeConv operation. The output of EdgeConv is computed by aggregating the features associated with all the edges that orginate from each connected vertex.[104]

This operator could be implemented as a shared MLP and then performing a max-pooling operation as in Eq. 28.

$$p'_{im} = \max_{j:(i,j)\in E} e'_{ijm}, \tag{28}$$

where $\theta = (\theta_1, \theta_2, ..., \theta_M)$ are the weights of M different filters used in the convolution operation.

Dynamic Graph Update A crucial point in this network is to regenerate the graph using k-NN in the feature space after each layer. This is the main difference of this network as compared to traditional graph CNN for which there is a fixed graph for all the layers of the network. Because the graph is updated after every layer, the architecture gets its name the DGCNN. With dynamically updating the graph after each layer, the receptive field increases to as large as diameter of the point cloud, even if the points in the point cloud are sparse. Thus, after each layer a different graph is computed $G^{(l)} = (V^{(l)}, E^{(l)})$, where $G^{(l)}$ is a graph in the l-th layer with vertices $V^{(l)}$ and edges $E^{(l)}$ which can be denoted as $(i, j_{i1}), ..., (i, j_{ik_l})$ such that $p^{(l)}_{j_{i1}}, ..., p^{(l)}_{j_{ik_l}}$ are the k-nearest points closest to point p_i at layer l. In other words, the model constructs a different graph for each point in the point cloud after every layer where the nearest points and the edges emerging from a vertex(point in the point cloud in the case) changes at every layer. A pairwise distance matrix in the feature space is used to calculate the k-nearest points of each point in the point cloud. As a result of this, the network has the following properties,

- *Invariant to permutations*. The output of an EdgeConv layer is given by

$$p'_i = \max_{j:(i,j)\in E} h_\theta(p_i, p_j). \tag{29}$$

  Because of the usage of a max operator which is symmetric in nature, the output of the layer p'_i is permutation invariant to the order in which p_i appears in the input point cloud. Other functions which are symmetric (eg. $\sum$) in nature can also be applied here instead of max. The global max pooling operator used to aggregate point features is a symmetric function and thus, gives permutation invariant results.

- *Invariant to translation.* The EdgeConv operator satisfies "partial" translation invariance. If a translation T is applied to the points p_i and p_j of the point cloud $\mathcal{P}$, then applying translation to Eq. 27, the translated point cloud would be

$$\begin{aligned} e'_{ijm} &= \theta_m \cdot (p_j + T - (p_i + T)) + \phi_m \cdot (p_i + T), \\ &= \theta_m \cdot (p_j - p_i) + \phi_m \cdot (p_i + T). \end{aligned} \tag{30}$$

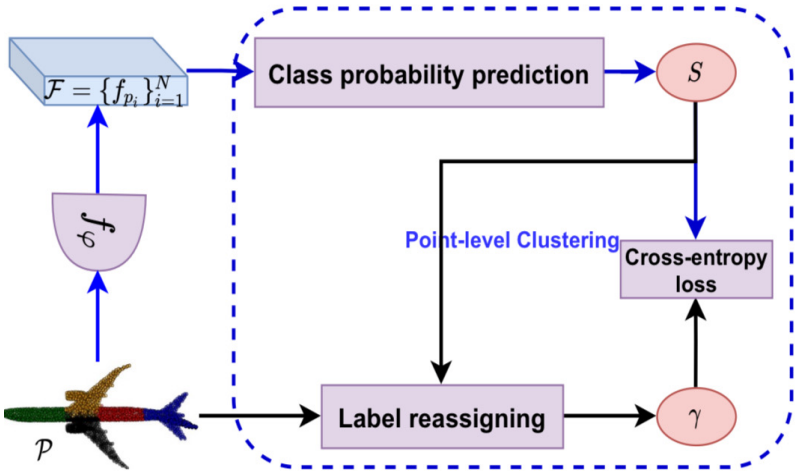

Figure 4.1.7: The architecture of the point-level clustering based unsupervised feature learning.[59]

Therefore, if $\phi_m = 0$, then EdgeConv operator is completely translation invariant. But this however restricts the model to recognize an object depending on only the unordered set of patches, i.e. it disregards the orientation and the exact location of the patches. On the other hand, with both $p_j - p_i$ and p_i, the model considers both the local orientation of the patches as well as the global information about the shape of the object.

4.1.4 Point-level Clustering Module

Now, the model already has the feature representation from the feature encoder backbone (eg. PointNet or DGCNN). The next step is the point-level clustering module. Fig.4.1.7 gives a detailed visualization of the point-level clustering module. It can be seen that it is made up of 2 entities, the "class probability prediction" function and the "label reassigning". The point-level clustering module is analogous to the semantic segmentation task. Each point p_i of the point cloud $\mathcal{P}$ is assigned to one of the J predetermined possible subgroups or semantic categories. Essentially, it is achieved by processing each of the point cloud $\mathbf{P}$ followed by a point-level class probability prediction operator that yields a matrix for class probability $\mathbf{S} = \{s_{ij} \in [0,1]\}_{i,j}^{N,J}$. The label reassigning operator thus, takes the point cloud $\mathbf{P}$ and the class probability matrix S and generates a pseudo-label matrix $\gamma = \{\gamma_{ij} \in \{0,1\}\}_{i,j}^{N,J}$. The weights for the encoder f_φ are learnt such that it minimizes the average cross-entropy loss between the pseudo-label γ and the predicted class probability $\mathbf{S}$. The average cross entropy loss is calculated by the formula given in Eq.31.

$$H(\gamma, \mathbf{S}) = -\frac{1}{N}\langle \gamma, \log \mathbf{S} \rangle = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{J}\gamma_{ij}\log s_{ij}. \tag{31}$$

But to train a network with a cross-entropy loss, the ground truth labels of the points in the point cloud are required. Since that is not available, one needs to assign the label γ automatically, which is explained in details in the following passage.

Class Probability Prediction

As mentioned in the last passage, the output of the encoder f_φ is fed as input to a classification head ϕ_α and it generates a logit score for each of the feature vector as shown in Fig.4.1.8. The vector for the logit score is given by $\mathbf{g}_i = (g_{i1}, g_{i2}, ..., g_{iJ}), where \mathbf{g}_i = \gamma_\alpha(f_{p_i})$. The classification head has 2 fully connected layers where each layer is a linear layer followed by a batch normalization. The leakyReLU activation function is used in all the layers except the last layer. As the output of the last layer N vectors for each of the N points in the point cloud is received where each of the output vector has J dimensions, the number of predetermined subcategories each point cloud is to be segmented into. Thus, each entry of the feature vector

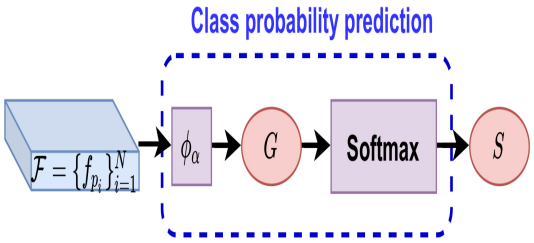

Figure 4.1.8: Architecture of class probability prediction.[59]

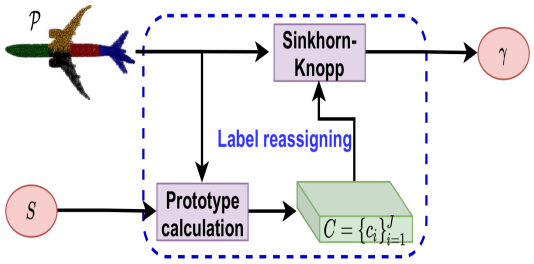

Figure 4.1.9: Architecture of label reassigning [59]

is the probability of that point belonging to j-th category. Therefore, the logit score matrix $\mathbf{G} = \{g_{ij}\}_{i,j}^{N,J}$ has the dimension $N \times J$. The prediction that the point p_i belongs to the category j is computed by the application of a row-wise softmax operation on $\mathbf{G}$ given by Eq.32.

$$s_{ij} = \frac{exp(g_{ij})}{\sum_{l=1}^{J} exp(g_{ij})}. \tag{32}$$

Label reassigning

In order to calculate the labels shown in Fig.4.1.9, the representative members of each of the semantic subgroups is needed. It can be done by selecting the cluster mean or cluster centers of each of these subgroups as the representative member of the cluster. The class probability matrix S can be infered as the soft assignment of each point p_i in the point cloud to J semantic partitions. So the softly weighted mean of each of these partitions can be calculated as in Eq.33.

$$c_j = \frac{1}{\sum_{i=1}^{N} s_{ij}} \sum_{i=1}^{N} s_{ij}\mathbf{p}_i, j = 1, 2, ..., J, \tag{33}$$

or in another way $C = \{c_j\}_{j_1}^{J}$ is a matrix with dimensions $J \times 3$. In a completely unsupervised learning setup, $\gamma \in [0,1]$ can be visualized as the posterior probability of a p_i belong to the $j-th$ partition. γ is optimized in the same method as in Eq.32. This results in a degenerate solution where each point is assigned to a single category. Thus, two assumptions are very pivotal for the reassigning of the labels. Firstly, the points in the point cloud ought to be partitioned into "equally-sized" partitions. In other words it can be formulated as $\sum_{i=1}^{N} \gamma_{ij} = \frac{N}{J}$. This is extremely importance in order to pretend all the points in the point cloud to be assigned to a single partition. Secondly, since the idea of using cluster centers as representative members of the partitions is inspired from the k-means algorithm, it means that if a point p_i of the point cloud belongs to the j^*-th partition, the distance from p_i to c_j should be the shortest as compared to the distance of p_i to all other cluster centers. In other words $\|\mathbf{p}_i - c_{j^*}\|_2 \leq \|\mathbf{p}_i - c_j\|_2, j \neq j^*$. This is also equivalent to $min_\gamma \sum_{i=1}^{N} \sum_{j=1}^{J} \gamma_{ij} \|\mathbf{p}_i - c_j\|_2^2$. This objective function is optimized to get γ. As per the rules of probability, the sum of all possible outcomes of an event is equal to 1, therefore $\sum_{j=1}^{J} \gamma_{ij} = 1$. Thus, the matrix of joint probability can be formulated as Γ as $\Gamma = \frac{\gamma}{N}$ which has the elements $\Gamma_{ij} = \frac{\gamma_{ij}}{N}$. The distance matrix $D = \{d_{ij}\}_{i,j}^{N,J}$ has dimensions $N \times J$ where each element d_{ij} is defined as $d_{ij} = \|\mathbf{p}_i - c_j\|_2^2$. In accordance to the assumptions stated above, the objective function D could be minimized with respect to Γ as in Eq.34.

$$\begin{aligned} &\min_{\Gamma} \langle \Gamma, D \rangle, \\ &s.t., \Gamma^T \vec{1}_N = \frac{1}{J}\vec{1}_J =, \Gamma \vec{1}_J = \frac{1}{N}\vec{1}_N, \end{aligned} \tag{34}$$

where $\vec{1}_J$ is the vector of ones having dimension J. The above condition enforces that each of the cluster representative is selected atleast $\frac{N}{J}$ times for each point cloud and $\sum_{j=1}^{J} \gamma_{ij} = 1$. The objective function as defined in Eq.34 is a typical case of an optimal transport problem [72]. Such problems can be solved by

the Sinkhorn-Knopp algorithm [17]. In other words, it means solving the entropic regularized objective function in Eq. 35.

$$\begin{aligned} &\min_{\Gamma} \langle \Gamma, D \rangle - \lambda H(\Gamma), \\ &s.t., \Gamma^T \vec{1}_N = \frac{1}{J}\vec{1}_J =, \Gamma \vec{1}_J = \frac{1}{N}\vec{1}_N, \end{aligned} \tag{35}$$

where, $H(\Gamma) = \langle \Gamma, \log\Gamma - 1 \rangle$ is the entropy of Γ and λ is the regularization parameter. As per [17], on solving Eq. 35, the normalized exponential matrix in Eq. 36 is attained.

$$\Gamma = diag(\mu) exp(D/\lambda) diag(\nu), \tag{36}$$

where μ and ν are renormalized vectors in $\mathbb{R}^N$ and $\mathbb{R}^J$ respectively and $exp()$ is the exponential function. These vectors are computed on utilizing the iterative Sinkhorn-Knopp algorithm[17] with the constraints $\mu = \frac{1}{N}\vec{1}_N$ and $\nu = \frac{1}{J}\vec{1}_J$. The pseudo-code for the Sinkhorn-Knopp algorithm is elaborated in Algorithm 1.

Algorithm 1 Pseudo-code for Sinkhorn-Knopp algorithm

def sinkhorn($D, \varepsilon, niters = 3$)

$\Gamma = exp(D/\varepsilon)^T$
$\gamma/ = sum(\Gamma)$
$N, J = \Gamma.shape$
$u, r, c = zeros(N), ones(J)/J, ones(N)/N$
for $inrange(0,\ niters)$ **do**
    $u = sum(\Gamma, dim = 1)$
    $\Gamma* = (r/u).unsqueeze(1)$
    $\Gamma* = (c/sum(\Gamma, dim = 0)).unsqueeze(0)$
end for
return Γ

Thus, the overall point-level clustering can be encapsulated by an Expectation Maximization (EM) like algorithm. The network learns the parameters f_φ and ϕ_α by optimization of Eq.31 and get a label reassigning matrix γ by solving the optimization problem in Eq. 36 with respect to Γ. It is executed by performing the following steps in repetition.

- Step 1: representation learning. Having the current probability matrix γ, the weights of the model are updated by optimizing Eq. 31 with respect to the parameters f_φ and ϕ_α. It is in line with the supervised case where the model uses the cross entropy loss for classification.

- Step 2: label reassigning. On having the updated set of f_φ and ϕ_α, Γ is calculated by using Eq. 36. The posterior probability matrix is then computed by $\gamma = N \cdot \Gamma$.

Thus, each update step requires a single matrix-vector multiplication which has the time complexity of $O(N \times J)$ which means the computation time increases linearly with the increase in the number of points in the point cloud. Therefore, scalability of this method is quite high and the process remains relatively quick even for a million samples in the dataset. Moreover, the orthogonal regularization is introduced to circumvent the issue of having the same output vector for all cluster representatives. It is computed by

$$\mathcal{L}_{orth}(C) = \|C_*^T C_* - I\|_1, \tag{37}$$

where $\|\cdot\|_1$ is the l_1-norm or the Manhattan distance and $C_* = [\frac{c_1}{\|c_1\|_2}, \frac{c_2}{\|c_2\|_2}, ..., \frac{c_J}{\|c_J\|_2}]$. The pseudo-code for the point-level clustering module in elaborated in Alg. 2.

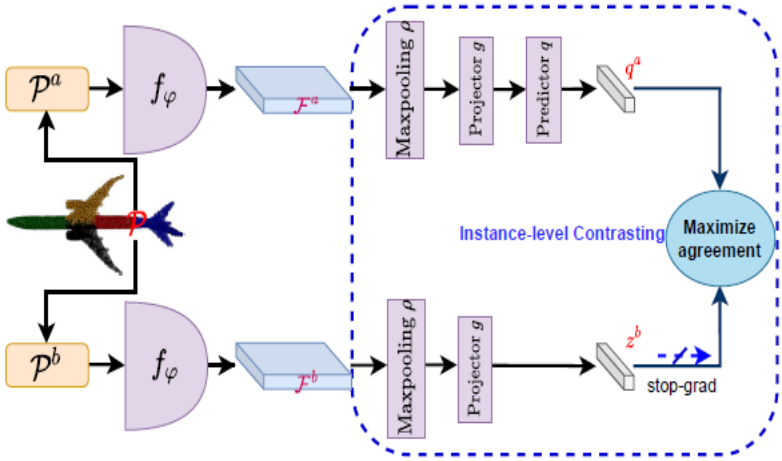

Figure 4.1.10: The architecture of the instance-level Contrasting.[59]

Algorithm 2 Pseudo-code for point-level clustering

Input: $\{\mathcal{P}\}$: is a set of 3D point clouds with N number of points in it; K: number of epochs.
Output: backbone f_φ.

for i in range $(0, K)$ **do**
  $\mathcal{L} = 0$
  for $\mathcal{P} \in \{\mathcal{P}\}$ **do**
    # Computes Class probability
    $S = softmax(\phi_\alpha(f_\varphi(\mathcal{P})))$
    # Computes cluster representatives of partitions
    $C = \left\{ \frac{1}{\sum_{i=1}^{N} s_{ij}} \sum_{i=1}^{N} s_{ij} \mathbf{p}_i \right\}_{j=1}^{N}$
    # Computes mean-squared error
    $D = \left\{ \|\mathbf{p}_i - c_j\|_2^2 \right\}_{i,j}^{N,J}$
    # Computes posterior probability matrix
    $\gamma = sinkhorn(stop-gradient(\text{stop-grad})(D), 1e-3, 20)$
    # Computes Loss Function
    $\mathcal{L} = H(\gamma, S) + \eta \mathcal{L}_{orth}(C)$
  end for
  # Updates backbone, projector ad predictor
  $f_\varphi, \phi_\alpha \leftarrow optimize(\frac{\mathcal{L}}{N})$
end for
return $\mathbf{f}_\varphi$

4.1.5 Instance-Level Contrasting Module

This module plays a crucial role in learning the global features of the point clouds. The architecture of this module is shown in Fig. 4.1.10. The network is fed with two augmented views $\mathcal{P}^a$, $\mathcal{P}^b$ of the same point cloud $\mathcal{P}$. The feature representation obtained as the output of the encoder backbone f_φ, are fed to a MaxPooling layer ρ, the output of which is fed to a projection MLP head g [15]. The encoder backbone f_φ and the projection head g shares the weights for both the augmented views. The output of the projection head of one of the augmented views (the top branch) is fed to the prediction head q. The purpose of this unit is to transform the output of one of the augmented views to match the other. The operation is applied to only one augmented view to make the network asymmetric [31]. In other words, the output vectors can be realized as $q^a \triangleq \mathbf{q}(g(\rho(\mathcal{F}^a)))$ and $\mathbf{z}^b \triangleq g(\rho(\mathcal{F}^b))$ where $\mathcal{F}^a = f_\phi(\mathcal{P}^a)$ and $\mathcal{F}^b = f_\phi(\mathcal{P}^b)$. Keeping in line with [15], stop-grad is applied on the second branch z^b to resist the network from generating a constant

mapping without using negative samples for contrastive learning. The network is constructed such that the similarity between q^a and z^b is maximized. Mathematically speaking, the mean-squared error between the l_2-normalized prediction q^a and projection z_b is minimized as shown in Eq. 38.

$$\mathcal{D}(q^a, z^b) \triangleq \| \frac{q^a}{\|q^a\|_2} - \frac{z^b}{\|z^b\|_2} \|_2^2 = 2 - \frac{2q^{aT} z^b}{\|q^a\|_2 \cdot \|z^b\|_2}. \tag{38}$$

In other words, it is the negative cosine similarity, up to a scale of 2. Since, stop-grad is enforced on z^b, Eq. 38 is modified by the following Eq. 39.

$$\mathcal{D}(q^a, \text{stop-grad}(z^b)), \tag{39}$$

which means z^b acts as a constant vector in this equation. Keeping in line with [15], a global symmetrized loss is defined as in Eq. 40.

$$\mathcal{L}_{global} = \mathcal{D}(q^a, \text{stop-grad}(z^b)) + \mathcal{D}(q^b, \text{stop-grad}(z^a)). \tag{40}$$

The minimum value the above equation can attain is 0. This is because, in the first term of the equation, the encoder backbone q^a gets no gradient flow from z^b. But it gets gradients from q^b in the second term. In the same way, in the second term of the equation, the encoder backbone q^b gets no gradient flow from z^a. But it gets gradients from q^a in the first term. if the stop-grad mechanism is not applied, then in spite of having a loss of zero during training, the representations learnt by the network would be futile i.e. all the point clouds would be mapped to the same feature representation. Or in other words, the network would crumble to constant mapping [15]. The pseudo-code for this module is shown in Alg. 3.

Algorithm 3 Pseudo-code for instance-level clustering

Input: $\{\mathcal{P}\}$: is a set of 3D point clouds with N number of points in it; K: number of epochs.
Output: backbone f_φ.

for i in range $(0, K)$ **do**
  $\mathcal{L} = 0$
  for $\mathcal{P} \in \{\mathcal{P}\}$ **do**
    # Generate random augmentations
    $\mathcal{P}^a = aug(\mathcal{P})$
    $\mathcal{P}^b = aug(\mathcal{P})$
    # Compute projections
    $z^a = g(\rho(f_\varphi(\mathcal{P}^a)))$
    $z^b = g(\rho(f_\varphi(\mathcal{P}^b)))$
    # Compute predictions
    $q^a, q^b = q(z^a), q(z^b)$
    # Calculate loss
    $\mathcal{L} += \mathcal{D}(q^a, \text{stop-grad}(z^b)) + \mathcal{D}(q^b, \text{stop-grad}(z^a))$
  end for
  # Updates backbone, projector ad predictor
  $f_\varphi, g, q \leftarrow optimize(\frac{\mathcal{L}}{N})$
end for
return $\mathbf{f}_\varphi$

4.1.6 Loss Function

The objective of the network is to simultaneously minimize both the point-level clustering loss and the instance level contrasting loss. If the two augmented views of the point cloud $\mathcal{P}$ are $\mathcal{P}^a$ and $\mathcal{P}^b$, then the point-level clustering loss can be calculated by Eq. 41.

$$\mathcal{L}_{local} = H(\gamma^a, S^a) + H(\gamma^b, S^b) + \eta(\mathcal{L}_{orth}(C^a) + \mathcal{L}_{orth}(C^b)) \tag{41}$$

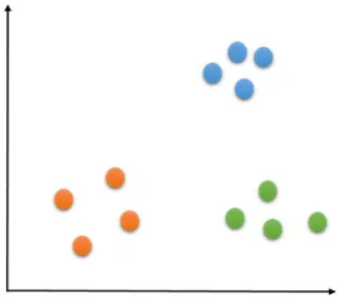

Figure 4.2.1: The original data in 2-D space [95].

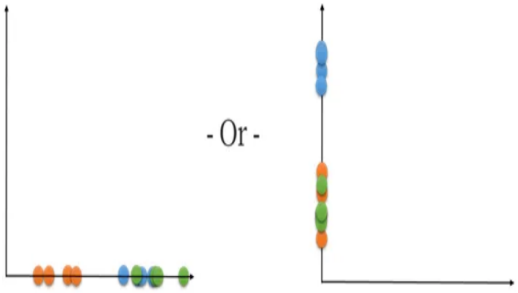

Figure 4.2.2: Data projected to 1-D space [95].

where $\eta > 0$ is a regularization parameter. Here γ^a is the posterior class probability of the point belonging to one of the subcategories, S^a is the predicted class probability and C^a is the normalized center of the augmented view of the point cloud $\mathcal{P}^a$. In the same way, the corresponding terms for the other augmented view $\mathcal{P}^b$ are γ^b, S^b and C^b. Therefore, the total loss could be calculated as the linear combination of $\mathcal{L}^{global}$ and $\mathcal{L}^{local}$ as in Eq. 42.

$$\mathcal{L}_{total} = \mathcal{L}_{local} + \mathcal{L}_{global}. \tag{42}$$

4.2 Dimensionality Reduction

4.2.1 Reasons for Dimension Reduction

The goal of this thesis is to find a diverse set of objects that collectively represent the entire ABC dataset [47]. This set of objects is to be then used as the source library for the different transfer learning approaches to be used in bin picking applications. Therefore, clustering algorithms are necessary to find the representative members of the dataset. In order to facilitate the usage of the DBCV Index as the evaluation metric for the clustering algorithm as mentioned in Ch. 4.4.1, it is necessary to reduce the dimensions of the latent space representations. Furthermore, it would be beneficial to find lesser but denser clusters. This means that fewer objects could collectively represent the entire dataset and each representative member would be similar to a greater number of other objects in the dataset. Thus, reducing the dimensions of the latent space representations by dimensionality reduction would entail more "crowded" clusters in case of density-based clustering algorithms.

4.2.2 T-distributed Stochastic Neighbor Embedding

A technique for reducing dimensionality, t-SNE is mostly used for data visualization in 2D and 3D maps. This approach has become well-liked since it can identify non-linear relationships in the data. If the data has more than two or three features, it might be necessary to consider looking for clusters in the data. To better understand the data and determine how many clusters to use in clustering models like k-means, if necessary. To further grasp what one hopes to obtain, the following example is used for illustration. It is assumed that the data is to be converted from a 2D space to a 1D dimension as shown in Fig. 4.2.1 and Fig. 4.2.2 respectively.[99]
Every color in this illustration denotes a cluster. It is evident that the densities of each cluster differ. It can be observed that at least two of the clusters overlap when a simple projection of the data onto one of its dimensions is performed. This highlights that a better method is necessary for the dimension reduction task. This problem is handled by the t-SNE algorithm and its effectiveness is described in the following steps.

- Determining a joint probability distribution that captures the commonalities amongst the data points.[99]
- Constructing a dataset of the points with the target dimension and thereafter computing the joint probability distribution of those points as well.[99]

Figure 4.2.3: A random dataset created in 1D space [95].

Figure 4.2.4: Visualization of the dataset in 1D space after applying t-SNE [95].

- Gradient descent is used to transform the low-dimensional dataset into a joint probability distribution that is as close as feasible to the high-dimensional dataset.[99]

Working principle of t-SNE

1. **The likelihood of points to belong to the same neighborhood**
   Finding each point's Euclidean distance from every other point is the initial step in the process. Next, these distances are converted into conditional probabilities, which express how similar one pair of points is to the other. This means that it is required to determine the degree of similarity—that is the likelihood—between any two points in the data. A Gaussian distribution with a standard deviation of σ_i and a center at x_i represents the conditional probability of point x_j to be next to point x_i. The conditional probability is defined mathematically as shown in Eq. 43. The factors that effect σ_i are discussed later.

$$p_{j|i} = \frac{exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}, \tag{43}$$

   where $p_{j|i}$ is the probability that the point x_i has x_j as its neighbor. Since the clusters can have varying densities, the division by the total of all the other points positioned at the Gaussian distribution centered at x_i is necessary. Looking back to Fig. 4.2.1 as an example to clarify, the orange cluster has a lesser density than the blue. As a result, there will be less similarity between the orange and blue points if the Gaussian distribution is used to compute the commonalities between each pair of points. The normalization is performed because it is only necessary to view the clusters as such in the final output and if any of the clusters have differing densities need not be visualized. The joint probability distribution can be computed from the conditional distributions as shown in Eq. 44.[99]

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}. \tag{44}$$

   One of the ways that the t-SNE approach differs from the previous SNE is that it uses the joint probability distribution instead of the conditional probability. The computation in the third stage of the algorithm is made simpler by the symmetric property of the pairwise similarities ($p_{ij} = p_{ji}$).

2. **Transformation of the data to a lower dimension**
   In this step, a joint probability distribution for the points in the low-dimensional space is computed. In order to achieve that, a new dataset is created at random that has K features (K being the desired dimension) and the same amount of points as the original dataset. As the dimension reduction is performed for visualization, K will often be two or three. Returning to the example, the t-SNE algorithm now creates a random dataset of 1D points as shown in Fig. 4.2.3. Similar to the original dataset, the joint probability distribution for this set of points is generated using the t-distribution rather than the Gaussian distribution. The points in this case are denoted as y and the probabilities as q and thus, the joint probability distribution is computed as shown in Eq. 45.

$$q_{ij} = \frac{(1 + \|y_i - y_j\|)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|)^{-1}}. \tag{45}$$

The heavy tails attribute of the t-distribution is the rationale behind selecting it over the Gaussian distribution. This property helps minimize "crowding" of the points in the lower dimension by making moderate gaps between points in the high-dimensional space. An additional benefit of employing the t-distribution is that it enhances the algorithm's optimization procedure in the third stage.[99]

3. **Modification of the dataset in the low-dimensional space for the best possible data visualization** Now, the Kullback–Leibler divergence (KL) divergence is utilized to maximize the similarity between the joint probability distribution of the data points in the low dimensional space and the higher dimensional space. The KL divergence serves as a gauge for the degree of difference between two distributions. It is defined for distributions P and Q in the probability space of χ by Eq. 46.

$$D_{\mathrm{KL}}(P||Q) = \sum_{x \in \chi} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \tag{46}$$

In instances where the distributions are identical, the KL divergence value approaches to 0, indicating an increase in similarity between the distributions. The lower dimension dataset is modified so that, in the joint probability distribution, it resembles the original data as soon as possible. To do this, gradient descent is used. The KL divergence between the joint probability distributions P from the high-dimensional space and Q from the low-dimensional space is the cost function that the gradient descent algorithm seeks to minimize as defined in Eq. 47.

$$C = \mathrm{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{47}$$

The low dimension dataset's point values are obtained from this optimization, which is then utilized for visualizing the dataset in the lower dimensional space. Refering to the example before, the clusters in the low-dimensional space appears as in Fig. 4.2.4. The t-SNE model has a number of hyperparameters that can be fine-tuned depending on the use-case. A few of these have to do with gradient descent. The two most crucial ones are the number of iterations, the learning rate and perplexity. It serves to select the Gaussian distribution expressing the conditional distribution in high-dimensional space's standard deviation σ_i. It can be defined as the count of each point's effective neighbors. For perplexities ranging from 5 to 50, the model is fairly robust.[99]

4.3 Clustering Algorithms

4.3.1 Requirements and Conditions

Once the feature representations are obtained, multiple clustering algorithms are utilized to group the objects in the datasets into clusters. For this, one algorithm from each type of clustering algorithms as mentioned in 3.3 are implemented to analyze the behavior of different clustering algorithms in this use case. Since the ultimate goal of the task in hand is to find representative members of the dataset, it is more robust to find the medoid of the cluster instead of the arithmetic mean. As mentioned in 3.3.2, the medoids are actual datapoints of the dataset which is more practical to use in this use case instead of using a rather blurred, noisy arithmetic mean of the clusters.

4.3.2 Selected Clustering Algorithms

The scikit-learn implementations of the k-medoids algorithm [85], spectral clustering algorithm [71] and agglomerative clustering algorithm [71] are used for this purpose. Since the spectral clustering algorithm and agglomerative clustering algorithm implementations provide the cluster centres and not the cluster medoids at the end, an additional step is necessary to compute the cluster medoids. Another clustering algorithm that is suggested for this task is the "multi-level HDBSCAN" clustering. The traditional implementation of the HDBSCAN algorithm available in scikit-learn can provide hierarchical density based clusters. However, this algorithm does not provide any control on the number of clusters that can be generated. Thus, in the presence of very large datasets like the ABC [47] dataset, it can potentially generate more than a hundred thousand clusters and thus, that many number of cluster representative members. Generating that many

data for the source library in the current infrastructure present at IPA is extremely computationally expensive and thus, technically infeasible. Increasing the minimum cluster size to be considered as a cluster and increasing the minimum number of samples around a point to be considered as a core point can decrease the overall number of cluster but that would also mean that the algorithm would label more datapoints as outliers. Losing too much information about the objects as outliers can jeopardize the purpose of a source library which is expected to generalize the feature of maximum possible objects in the dataset.

Working principle of multi-level HDBSCAN algorithm

1. At first HDBSCAN algorithm is applied on the whole dataset. After the clustering algorithm, p medoids are obtained. These p datapoints are the relevant points for the next step.
2. The HDBSCAN algorithm is applied again on those p datapoints. As a result of it, m clusters are obtained and n datapoints are labelled as outliers. Thus, there would be m medoids which are the representative members of each of the clusters. From this level onwards, no new datapoints would be discarded as outliers. Thus, each of the datapoints labelled as outliers would be assigned to a new cluster along with all the fellow cluster member which is a part of the cluster this particular datapoint is a member of in the previous level. The pseudo-code for this part of the algorithm is shown in Alg. 5.
3. Thus, the relevant datapoints for the next level are the m medoids and all the n outliers relabelled as medoids of new clusters. Step 2 and 3 are repeated until convergence, i.e. when $m+n \leq k$ where k is the maximum number of permissible clusters as shown in Alg. 4.

After the convergence of the algorithm, c medoid points are obtained which are the representative members of the c clusters where $c \leq k$. Similar to the k-medoids algorithm, the Euclidean distance metric is used to compute the medoids. The "*min_cluster_size*" hyperparameter controls the minimum number of datapoints to be contained in a cluster. Increasing this value would mean clusters with more number of elements. But this would also mean that groups smaller than this size would be treated as outliers. Thus, increasing this value could lead to significant loss of information about the features of different objects in the cluster. To circumvent this issue, the value of this parameter is kept at its default value which is 2. This is the reason why HDBSCAN algorithm returns a significantly high number of clusters. Thus, the proposal of multi-level HDBSCAN algorithm is deemed necessary for this use case.

Algorithm 4 Pseudo-code for hierarchical clustering

Input: $\{\mathcal{E}\}$: is the set of feature representations for all objects in the dataset, K is the number of permissible clusters.
Output: $\{\mathcal{P}_R\}$ is the set of representative members of the dataset.

$\mathcal{E}_R \leftarrow \{\mathcal{E}\}$
$classifier,\ predictions,\ num\_clusters = \text{HDBSCAN}(\mathcal{E}_R)$
$count \leftarrow 0$
$C \leftarrow predictions$
$cluster\_members = get\_cluster\_members(C, \{\mathcal{E}\})$
$clusters = list(unique(C))$
while $True$ **do**
    $\mathcal{E}_R$, C, $cluster\_members$, $num\_clusters\ classifier = get\_hier\_clusters(\mathcal{E},\ \mathcal{E}_R,\ C,\ cluster\_members, count)$
    $count += 1$
    $total\_num\_clusters = len(list(unique(C)))$
    if $total\_num\_clusters \leq K$ **or** $n\_clusters\ is\ 0$ **then**
        break
    end if
end while
$medoid\_indices = get\_medoid\_indices(\mathcal{E}_R, \mathcal{E}\ )$
$\{\mathcal{P}_R\} = \{\mathcal{P}_i,\ \mathcal{P}_i \in \{\mathcal{P}\}\ and\ i \in medoid\_indices\}$
return $\{\mathcal{P}_R\}$

Algorithm 5 Pseudo-code for multi-level HDBSCAN clustering

Input: $\{\mathcal{P}\}$: is a set of 3D point clouds with N number of points in it, $\{\mathcal{E}\}$: is the set of feature representations for all objects in the dataset, $\{\mathcal{E}_R\}$ is the set of feature representations for relevant points objects in the dataset, C is the cluster membership of all the points clouds in the dataset, *clusters* is the list of all clusters, *clusters_members* is the list of all the point clouds in the dataset, grouped by their cluster membership, *count* is the current level of multi-level HDBSCAN

Output: $\mathcal{E}_R$, C, $cluster\_members$, $num\_clusters$, $classifier$

$classifier$, $predictions$, $num\_clusters$ = HDBSCAN($\mathcal{E}_R$)
$medoids = classifier.medoids$
$cluster\_pointer \leftarrow num\_clusters$
Assign Empty list to $outlier\_list$
for $\forall\ obj\_idx, object \in enumerate(\mathcal{E}_R)$ **do**
    if $prediction[obj\_idx] < 0$ **then**
        $AddItem(outlier\_list,\ obj\_idx)$
        $C[cluster\_members[obj\_idx]] = cluster\_pointer$
    else
        $C[cluster\_members[obj\_idx]] = prediction[obj\_idx]$
    end if
end for
$\{O\} = \{\mathcal{E}_i,\ \mathcal{P}_i \in \{\mathcal{E}_R\}\ and\ i \in outlier\_list\}$
$cluster\_members = get\_cluster\_members(C, \{\mathcal{E}\})$
$medoid\_indices = get\_medoid\_indices(medoids, \mathcal{E}_R)$
$\mathcal{E}_R = \{\mathcal{E}_i,\ \mathcal{E}_i \in \{\mathcal{E}_R\}\ and\ i \in medoid\_indices\}$
if $count \neq 0$ **then**
    $\mathcal{E}_R = concatenate(\mathcal{E}_R,\ O)$
end if
if *count is* 0 **then**
    $num\_clusters = len(list(unique(C)))$
end if
return $\mathcal{E}_R$, C, $cluster\_members$, $num\_clusters$, $classifier$

The HDBSCAN algorithm performs the DBSCAN algorithm for different epsilon values and hence provides a more robust clustering algorithm. Moreover, the optimum value of epsilon is case-dependent and no single value is suitable for clustering different datasets. This also means that expert domain knowledge is essential for determining the optimum epsilon value and is difficult to obtain in this use case. Thus, it is logical to use the HDBSCAN algorithm in this use case as it incorporates the result of clustering for different epsilon value and finds the clustering solution that provides the optimum stability over all epsilon. The HDBSCAN algorithm can opt for different algorithms like K-Dimensional (KD)-trees, Ball Tree or brute-force algorithm.

KD-tree algorithm The KD-tree is a data structure that is used for the purpose of effectively organizing and searching for points in multi-dimensional spaces. Specifically, KD-trees excel at solving the nearest neighbor search problem, which involves finding the closest data points to a given query point. The structure of a KD-Tree resembles a binary tree, where each node can have up to two children. It operates by recursively dividing data points along different dimensions, creating a hierarchical arrangement that facilitates efficient search and retrieval as shown in Fig. 4.3.1. At each level, the tree selects a dimension and splits the data points into two groups based on their values in that dimension. This process continues until all data points are assigned to leaf nodes. One of the key advantages of KD-Trees lies in their ability to handle high-dimensional data in static dataset—a common challenge in machine learning applications like computer vision, natural language processing and bioinformatics. High-dimensional data often suffers from the "curse of dimensionality," where the search space grows exponentially with the number of dimensions, making nearest neighbor queries time-consuming. As the ABC [47] dataset is an immensely high dimensional dataset, the KD-Tree algorithm is used in this use case. KD-Trees address this by narrowing

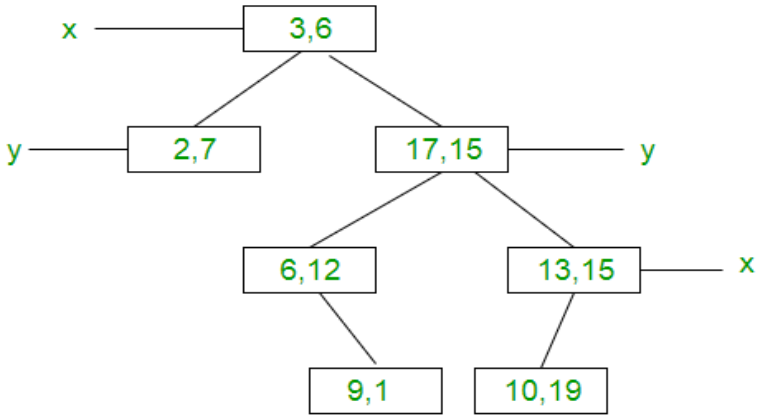

Figure 4.3.1: A demonstration of the KD-tree algorithm with 7 datapoints having 2 dimensions x and y.[86]

down the search space at each tree level, resulting in faster queries. Despite their advantages, KD-Trees face limitations and challenges. One concern arises when the number of dimensions increases, particularly with non-uniformly distributed data points. In such cases, the tree may become unbalanced, resulting in inefficient search times. But since the datapoints are reduced to 2-dimensional datapoints, the HDBSCAN algorithm doesn't result to inefficient search times in this case. Also KD-trees are sensitive to outliers. The steps for KD-tree algorithm are:

1. **Dimension Selection**: A dimension is chosen along which the data is split [28].
2. **Sorting**: The data points are sorted and arranged.
3. **Median Point**: The median point (root of the current subtree) is selected, splitting the data into two subsets [28].
4. **Recursive Process**: The process is repeated for each subset, with a new dimension selected at each level [28].

Ball-Tree algorithm The ball tree algorithm is designed for organizing and efficiently querying multidimensional data in computational geometry and machine learning. Formally, subsets of points are enclosed within hyperspheres and the structure is constructed as a binary tree. Each non-leaf node represents a hypersphere containing a subset of the data and each leaf node corresponds to a small subset of points. The partitioning process involves choosing a 'pivotal' point within the subset and constructing a hypersphere centered at this point to enclose the data as shown in Fig. 4.3.2. Fast nearest neighbor searches are facilitated by this hierarchical arrangement, allowing for the rapid elimination of entire subtrees during the search process. Ball trees are particularly useful in scenarios with high-dimensional data, offering improved efficiency over exhaustive search methods in applications such as clustering, classification and nearest neighbor queries. The steps for ball-tree algorithm are:

1. **Point Selection** : A random data point is chosen as the center of the hypersphere [66].
2. **Radius Calculation**: The radius of the hypersphere is calculated using distance metrics such as Euclidean distance or Manhattan distance [66].
3. **Datapoint Assignment**: Data points are assigned to the current node [66].
4. **Recursive Process**: The process is repeated by selecting new centers and radii [66].

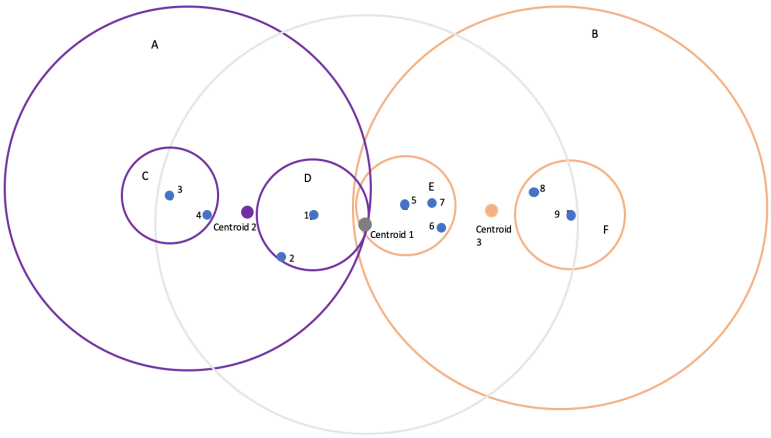

Figure 4.3.2: A demonstration of the Ball-tree algorithm with 9 datapoints having 2 dimensions x and y.[86]

Thus, the memory consumption for the ball-tree algorithm is way more in comparison to the KD-tree algorithm. Furthermore, it fails to provide good results in low-dimensional data. It results to inefficient search time for low dimensional queries. This KD-tree is suitable for low dimensional static use cases like image recognition and nearest neighbor search in low to medium dimensional dataset. Whereas, the ball tree algorithm is suitable for data clustering in high dimensional spaces and nearest neighbor search in high dimensional datasets.

Brute Force Algorithm A brute force algorithm is a straightforward and exhaustive search strategy that systematically examines all possible options until it finds the solution to a problem. It's a general approach used for problem-solving when the problem size allows for detailed investigation. In the brute force method, it is iterated through each data point and the density is calculated by considering the distances to other points. However, due to their high time complexity, brute force methods are inefficient for large-scale problems.

4.4 Evaluation Metrices

This section gives an overview of the evaluation metrices used to compare the results of the different clustering algorithms evaluated in this thesis. It also provides the details of the metric used to compare how similar the objects in the test dataset are to the representative members of the entire dataset.

4.4.1 Metric for Clustering Algorithms

It is evident how even the evaluation metrics using just the intrinsic features of the dataset are still not efficient in the presence of noise, outliers or non-convex clusters. As pointed out by the authors of [40], "without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage". The fact that this statement holds even today, 35 years after he said it is even more astonishing. To tackle the problem of inefficient evaluation metrics for non-convex clusters or where the different clusters have a difference in size Moulavi *et. al.* proposed the DBCV Index [61].

Density Based Cluster Validition Index

The authors have utilised the theory of Hartigan's model of density-contour trees [33]. The least dense region within a cluster and the most dense region outside a cluster is calculated with it which is analogous to computing the within cluster and between cluster density connectedness. As per the definition of Density Contour Trees [33] by Hartigan, density based clusters are regions of high density which are separated by low density regions. Having such model as a backdrop, it is expected that a good clustering algorithm

which generate clusters such that the least dense region within a cluster is still more dense as compared to regions surrounding a cluster. Traditional intrinsic measure based evaluation metrics uses distance between the datapoints to compute cluster-variance and then incorporates it with the separation between the clusters to eventually compute the quality of the clustering algorithm. But this is not the target for DBCV. So a metric is proposed which is defined by densities instead of distances. DBCV wants to take into account both the density and the shape of the cluster. So at first, the authors defined the concept of "all-points-core-distance"($a_{pts}coredist$). It is the inverse of the density of each object with respect to all other objects inside its cluster as given in Eq. 48.

$$a_{pts}coredist(\mathbf{o}) = \left\{ \frac{\sum_{i=2}^{n_i} \left(\frac{1}{KNN(\mathbf{o},i)} \right)^d}{n_i - 1} \right\}^{-\frac{1}{d}}. \tag{48}$$

Using the above-mentioned equation, a symmetric reachability distance is defined which is then utilized to generate the Minimum Spanning Trees (MST) inside each clusters. Because the MSTs are built on the transformed space of symmetric reachabilty distances, it is capable of capturing both the density and shape of the cluster. On using these MSTs, it is possible to find the least dense region within each of the clusters and the most dense regions between the pair of clusters. As per the definitions provided by the authors, $\mathbf{O} = o_1, o_2, ..., o_n$ is the dataset containing n objects in the $\mathbb{R}^d$ feature space. **Dist** is an $n \times n$ matrix of pairwise distance $d(o_p, o_q)$ between the objects of the dataset, such that $o_p, o_q \in \mathbf{O}$. $KNN(\mathbf{o}, i)$ is the distance between the object o and its i^{th} nearest neighbor. $C = (C_i, N)1 \leq i \leq l$ is the results returned by the clustering algorithms having l clusters and a set of N noise objects. n_i is the size of the i^{th} cluster and n_N is the number of objects in that particular cluster. To get an estimate of the density of an object within a cluster, traditional approaches use the notion of taking the inverse of the threshold distance required to find K objects within this threshold [33, 13]. But a drawback of this method would be that it is required to decide the value of the parameter K, which is not desirable.

Moreover, with this approach, the density of the object is dependent to its distance from only one other object i.e. the k^th nearest neighbor. This approach is not as robust as compared to considering more objects in its neighborhood as done in Gaussian distribution kernel density estimation. Thus, in order to propose a more robust density estimation, the use of a new parameterless core distance is proposed which can be elucidated as the inverse of a density estimate and thus, can be used in defining the Mutual Reachability Distance (MRD). So now instead of using just one point in the cluster, all the points are taken into account such that the objects closer to it contribute to the density more as compared to the objects further away. It can be seen in Eq. 48, that the inverse of the *KNN* distances raised to the power of dimensions is calculated such that closer objects gets a higher weight as compared to further objects. This effect can further be increased by using squared Euclidean distance instead of Euclidean distances. Also, as per the definition of MRD[52], the core distance of an object is compared to the distances of the object to other objects in the cluster. Hence, the core distance needs to be comparable to these distances. For each object **o** in the cluster, the "all-points-core-distance" $a_{pts}coredist(\mathbf{o})$ lies between the second and the last nearest neighbor distance of the object[61] as in Eq. 49.

$$KNN(\mathbf{o}, 2) \leq a_{pts}coredist(\mathbf{o}) \leq KNN(\mathbf{o}, n). \tag{49}$$

Moreover, the core distance is approximated to the distance of the object to its k^{th} nearest neighbor such that k is not too large. This means, that only a small neighborhood of the object is to be considered. Furthermore, let n objects be uniformly distributed random variables in a d-dimensional unit hypersphere. Then the core distance of object **o** is given by Eq. 50.[61]

$$a_{pts}coredist(\mathbf{o}) = \ln(n)^{-\frac{1}{d}}. \tag{50}$$

The core distance of the object **o** is approximately equal to the same nearest neighbor distance and is independent of the dimension of the data space, if the objects are uniformly distributed. If the dataset doesn't have a uniform distribution, which is the case for most real world datasets, only the first property holds i.e. the density of the object doesn't depend on just one other object but on all objects in the cluster.[61]

Now the $a_{pts}coredist$ is used to compute the symmetric reachability distance which is delved into deeper in the following paragraphs. As per the authors, the MRD between two objects o_i and o_j as in Eq. 51.

$$d_{mreach}(\mathbf{o}_i, \mathbf{o}_j) = max\{a_{pts}coredist(\mathbf{o}_i), a_{pts}coredist(\mathbf{o}_j), d(\mathbf{o}_i, \mathbf{o}_j)\}, \tag{51}$$

where $d(\mathbf{o}_i, \mathbf{o}_j)$ is the Euclidean distance between object $\mathbf{o}_i$ and $\mathbf{o}_j$. Once the MRD is computed, it could now be used to generate the mutual reachability graph where the objects are the different vertices and the graph and the MRD calculated above are the edge weights between the pair of vertices. Now with the set of objects $\mathbf{O}$ and the mutual reachability graph G, the MST of graph G is computed as MST_{MRD}. So a quick overview of the process of DBCV would be that for a single cluster C_i having objects O, the $a_{pts}coredist$ of each object in the cluster is calculated. This is then used to computer the MRD between all pair of objects in the cluster. Depending on the MRD, the MST is generated. This process is then performed for all the clusters generated by the clustering algorithm, eventually giving rise to l MSTs if the clustering algorithms returns l clusters in total. Thus, it can noticed that the number of MSTs increase linearly with the increase in the number of clusters. Also, the size of the tree depends on the number of objects in the clusters and thus, can blow up exponentially in case of very large datasets having a lot of elements in each cluster.

From the MSTs generated above, a new measure to validate the quality of the clustering algorithm depending on the density sparseness and density separation is crafted as explained below. The Density Sparseness of a Cluster (DSC) C_i can be visualized as the least dense region within the cluster and thus, can be obtained from the maximum edge weight of the MST_{MRD} of that particular cluster. On the other hand, Density Separation of a Pair of Clusters (DSPC) C_i and C_j where $1 \leq i, j \leq l, i \neq j$ can be visualized as the region of highest density in between adjacent clusters and can be formulated as the minimum MRD between the objects of a cluster to the objects of its neighboring cluster. The DSC and the DSPC together then give the validity index of a cluster as in Eq. 52.

$$V_C(C_i) = \frac{\min\limits_{1 \leq j \leq l, i \neq j} (\text{DSPC}(C_i, C_j)) - \text{DSC}(C_i)}{max\left(\min\limits_{1 \leq j \leq l, i \neq j} (\text{DSPC}(C_i, C_j)) - \text{DSC}(C_i)\right)}. \tag{52}$$

Once the validity index of each cluster is calculated, the final DBCV score of the clustering algorithm can be defined as the weighted average of the validity index of all the clusters in C generated by the clustering algorithm as in Eq. 53. It is to be noted that even if noisy elements aren't explicitly present in the formula as a separate cluster but it does however implicitly affects the overall score as the weighted average takes into consideration both the cardinality of individual clusters as well as the total number of elements in the dataset, which includes noise and outliers.

$$\text{DBCV}(C) = \sum_{i=1}^{l} \frac{|C_i|}{|O|} V_C(C_i). \tag{53}$$

If a cluster has better density compactness within the cluster as compared to density separation in between adjacent clusters, than the validity index of the cluster would take a positive value. On the other hand, if the density outside the cluster separting it from adjacent clusters is more than density compactedness inside the cluster than it would take a negative value. Thus, the DBCV score can vary in the range $[-1, 1]$ where greater DBCV score indicates to a better density-based clustering algorithm.

4.4.2 Metric for Similarity in Point Clouds

Hausdorff distance is commonly used to calculate the similarity between two point clouds. So if the set of representative members generalizes the entire dataset well enough, then each object in the test dataset would be very similar to atleast one object from the set of representative members. Hausdorff distance calculates how different one point cloud is to another. So, lower the Hausdorff distance, more similar are the point clouds to one another. Thus, for the a set of "good" representative members, the average distance from all objects in the test dataset to their corresponding nearest object in the representative set is minimum. Mathematically, Hausdorff distance is the measure of how far two subsets in a metric space. Intuitively

speaking, the Hausdorff distance between two sets of points is lower when every point in one set is close to some point in the other set. In other words Hausdorff distance is the longest distance that needs to be covered when traveling from any point in one of the subsets to the closest point in the other subset. Let (M,d) be a d-dimensional metric space and $X \subset M$, $Y \subset M$ are the two non-empty subsets of points, then the Hausdorff distance between X and Y is defined as in Eq. 54.

$$d_H(X,Y) := \left\{ max(\sup_{x \in X} d(x,Y), \sup_{y \in Y} d(X,y)) \right\}, \tag{54}$$

where sup is the supremum operator and $d(a,B) := \inf_{b \in B} d(a,b)$ is the distance from a point $a \in X$ to the subset $B \subseteq X$, where inf is the infimum operator. Thus, in the context of point clouds the Hausdorff distance is defined as in Eq. 55. If $P_1 = \{x_i \in \mathbb{R}^3\}_{i=1}^{n}$ and $P_2 = \{x_j \in \mathbb{R}^3\}_{j=1}^{m}$, then the maximum distance between any pair of nearest neighbors between the two point clouds is defined as in eq. 55.

$$hausdorff(P_1,P_2) := \frac{1}{2} \underset{x \in P_1}{max} |x - NN(x,P_2)| + \frac{1}{2} \underset{x' \in P_2}{max} |x' - NN(x',P_1)|, \tag{55}$$

4.5 Important Aspects of the Studies

4.5.1 Focus and Restrictions

The aim of this thesis is to find "good" representative members of the dataset which can be used as source library for transfer learning approaches to be used in bin picking applications. in order to do so, it is necessary to evaluate the performance of the clustering algorithms. But calculating the cluster evaluation metric like DBCV for a very large high dimensional data can be extremely time consuming and inefficient. In order to circumvent this issue, the evaluation is carried on smaller subset of the dataset. But in order to ensure that the results can be interpolated to the entire dataset, subsets of varying sizes of the dataset have been evaluated on to confirm that the conclusions are consistent. The dimensions of the latent space representations play a very crucial role when encoding the features of the objects in a dataset in lower dimensional space. But in order to accurately evaluate the performance of the clustering algorithms, the number of dimensions has to be reduced further to be able to utilize the DBCV metric for evaluation of the clustering results. Moreover, to facilitate larger but denser clusters t-SNE dimensionality reduction is utilized. In the event of a rapid reduction of dimensions for example 1024 to 2, it is advised in [97] to perform multi-step dimensionality reduction instead of reducing it in a single step. This helps in better handling of the noise and outliers [71]. To assure that the theoretical advice is coherent with the practical results, evaluations are carried on in this area, the results of which are documented in Ch. 5.1.1 and 5.2.2. The data generation procedure currently used at IPA requires 24 hours for completely generating a single object. It was decided to use 3 available servers for this purpose of data generation. A practical and feasible choice of roughly two months is made for this purpose. Two months nearly equals to 1400 hours. Thus, the underlying derivation shows how the maximum possible number of representative objects is determined.

$$\begin{aligned} Total\ time &= \frac{Number\ of\ clusters}{Number\ of\ servers} * Time\ for\ each\ object \\ Number\ of\ clusters &= \frac{Total\ time \times Number\ of\ server}{Time\ for\ each\ object} \\ &= \frac{1400 \times 3}{24} = 175 \end{aligned} \tag{56}$$

4.5.2 Hardware and Environment Setup

All the aspects of this thesis work has been carried on into a single setup, a powerful workstation computer, ML-Lab-2 at IPA. It is an Ubuntu server. The Central Processing Unit (CPU) are two ”Quad Core Intel Xeon Silver 4112 @ 2.60 GHz” with a total installed Random Access Memory (RAM) of 275 GB. The system has two Graphics Processing Unit (GPU) which are "Tesla V100-PCIE-32GB" and "NVIDIA GeForce RTX 3090" with 32 Giga Bytes (GB) and 24 GB memory respectively. Each of the individual model training is executed on a single GPU.

5 Experiments

The two different backbone architecture for the generation of the latent space representation of the objects and the performance of the different clustering algorithms are evaluated by hyperparameter optimization of the different parameters along the entire pipeline of the procedure is evaluated in this section.

5.1 Baseline Algorithm

The already existing PointNet autoencoder currently used at IPA is used as a baseline for evaluating the performance of the proposed algorithms in this thesis. The dimension of the output of the latent space representation of the baseline algorithm is 64. As mentioned in the restrictions in Section. 4.5.1, usually a two-step dimensionality reduction is performed before the datasets are grouped into clusters for the two networks evaluated in this thesis because the dimension of the output of the latent space representations is too high. This is done to suppress some noise and also to escalate the procedure by faster calculation of the pairwise distance between the datapoints. But since an output dimension of 64 is not very high, two-step dimensionality reduction is not necessary. However, to make the baseline algorithm comparable with the proposed networks, one-step dimensionality reduction is performed in some cases.

5.1.1 Basic Study without Dimensionality Reduction

Tab. 3 shows the results of the baseline algorithm when no dimensionality reduction technique is used. It shows the results of the basic algorithm for the clustering of 1000 datapoints. The unit of time across all the tables in this section is in seconds unless otherwise mentioned. The four clustering algorithms HDBSCAN, k-means clustering, spectral clustering and agglomerative clustering are evaluated. The number of clusters generated by the HDBSCAN algorithm cannot be fixed by the user and hence it is determined by the algorithm automatically. Thus, the number of clusters after the first level of the multi-level HDBSCAN algorithm as proposed in Section. 4.3.1 is 232 in this case. As mentioned in Ch. 4.5.1 the number of clusters for the rest of the clustering algorithms is set to 175 throughout all the experiments in this section. The DBCV score caries in the range [-1,1] with more positive results denoting better clustering of the datapoints. The reason why HDBSCAN algorithm shoes better results as compared to other clustering algorithms is HDBSCAN algorithm is a density based clustering algorithm. DBCV being a density based clustering validation index is likely to give better results for density based clustering algorithms which is proven to be true in this experiment. Furthermore, theoretically HDBSCAN is better capable of handling noises and outliers in the dataset which can also contribute to the better results as exhibited in this experiment.

No. of datapoints	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
1000	HDBSCAN	232	0.182	-0.189	297.031
	k-medoids	175	3.960	-0.332	237.640
	Spectral	175	2.396	-0.339	161.604
	Agglomerative	175	0.115	-0.327	157.584

Table 3: Results of the baseline algorithm on 1000 datapoints and without any dimensionality reduction technique.

5.1.2 Basic Study with T-distributed Stochastic Neighbor Embedding

Tab. 4 shows the results of the baseline algorithm when the t-SNE dimensionality reduction is applied. The number of components for t-SNE could be 1, 2 or 3 as per the available scikit-learn implementation [71]. It is set to be 2 in these set of experimentations for better and easier visualization. The DBCV score caries in the range [-1,1] with more positive results denoting better clustering of the datapoints. The results are seen to be significantly better as compared to Tab. 3 when dimensionality reduction technique is not applied. This is because the t-SNE algorithm tends to keep datapoints that are distant from one another in the higher dimensional space also distant datapoints even in the lower-dimensional space. This gives rise to more dense and distinct clusters thus, improving the DBCV score. As opposed to the result in Tab.3, where the DBCV score is negative, meaning the datapoints are mostly wrongly classification, in this table it can be seen that the datapoints are mostly correctly classified. It is also to be note that the number of clusters generated by the HDBSCAN algorithm is 286 which is a bit more as compared to the previous case. This further hints to the direction that as a result of using t-SNE, more dense and non-overlapping clusters are formed in the dataset.

No. of datapoints	t-SNE dim	t-SNE time (seconds)	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
1000	2	3.030	HDBSCAN	286	0.110	0.0975	275.039
	2	2.777	k-means	175	0.191	-0.052	94.004
	2	3.318	Spectral	175	3.604	-0.064	126.854
	2	3.190	Agglomerative	175	0.087	-0.076	113.937

Table 4: Results of the baseline algorithm on 1000 datapoints after applying t-SNE dimensionality reduction.

5.1.3 Basic Study with Increasing Datapoints

No. of datapoints	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2000	HDBSCAN	436	0.444	-0.234	2115.777
	k-medoids	175	0.793	-0.462	764.222
	Spectral	175	20.607	-0.617	749.487
	Agglomerative	175	0.205	-0.502	756.480

Table 5: Results of the baseline algorithm on 2000 datapoints and without any dimensionality reduction technique.

As mentioned in Ch. 4.5.1, it is extremely inefficient to compute the DBCV for all the datapoints in the ABC datasets [47] as once. It is necessary to verify that the results obtained for a smaller subset of the dataset can be interpolated for the entire dataset. In other words, the results obtained for a smaller subset of the dataset generalizes the behavior of the entire dataset well. Tab. 5 shows the results of the baseline algorithm on 2000 datapoints without applying dimensionality reduction and Tab. 6 shows the results of the baseline algorithm on 2000 datapoints when t-SNE dimensionality reduction is applied. It is observed that roughly 10% of the datapoints are classified as outliers by the HDBSCAN algorithm for all the experiments on 1000 and 2000 datapoints.

No. of data-points	t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2000	2	6.689	HDBSCAN	626	0.237	0.172	4497.151
	2	7.450	k-medoids	175	0.155	-0.226	932.509
	2	157.017	Spectral	175	91.684	-0.125	755.647
	2	5.155	Agglomerative	175	0.112	-0.192	363.605

Table 6: Results of the baseline algorithm on 2000 datapoints after applying t-SNE dimensionality reduction.

5.2 ConClu Approach with PointNet as Backbone

As mentioned in Ch. 4, two feature representation learning backbones are used for the experiments. Experiments with the PointNet autoencoder as the backbone is recorded here. The performance of all the clustering algorithm is evaluated. Different hyperparameter optimizations are performed which will be elaborated gradually through this subsection.

5.2.1 Evaluation of Increasing Datapoints

No. of data-points	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
1000	HDBSCAN	288	0.228	0.137	437.956
	Agglomerative	175	0.063	-0.057	191.887
	k-medoids	175	0.763	-0.120	192.580
	Spectral	175	7.831	-0.051	186.879

Table 7: Results of the ConClu approach on 1000 datapoints.

As mentioned in Ch. 4.5.1, it is extremely inefficient to compute the DBCV for all the datapoints in the ABC datasets [47] as once. It is necessary to verify that the results obtained for a smaller subset of the dataset can be interpolated for the entire dataset. In other words, the results obtained for a smaller subset of the dataset generalizes the behavior of the entire dataset well. Tab. 7 shows the results of the ConClu approach on 1000 datapoints and Tab. 9 and 10 shows the results of the ConClu approach on 2000 datapoints and 5000 datapoints respectively. It is observed that roughly 10% of the datapoints are classified as outliers by the HDBSCAN algorithm for all the experiments on 1000, 2000 and 5000 datapoints. The dimension of the latent space representations for table 7, 9 and 10 is 1024.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV Time (seconds)
Level 1	HDBSCAN	288	94	0.137	437.956
Level 2	HDBSCAN	111	33	-0.023	113.242

Table 8: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm.

Tab. 8 shows the performance of the multi-level HDBSCAN algorithm for 1000 datapoints. The performance of this algorithm is seen to deteriorate as compare to the traditional HDBSCAN algorithm. This is because, clusters are combined to form bigger clusters in this algorithm to limit the number of maximum clusters returned by it. But it is still seen to perform better as compared to other traditional clustering algorithm like k-medoids, spectral clustering and agglomerative clustering. The next best clustering algorithm in this case is spectral clustering. It is also to be noted that the clustering time even for multiple levels in multi-level HDBSCAN algorithm is less than that of spectral clustering algorithm by quite some factors. The number of outliers after the first level denotes the number of datapoints that is newly classified as outlier which is not classified as outliers in the previous layer. These new outliers and its associated cluster members are then re-assigned as a new cluster. Thus, outliers of the dataset remains constant after the first iteration of the multi-level HDBSCAN clustering algorithm. This means, more information is not lost and no new datapoint is classified as an outlier. This is necessary to ensure that not too many objects in the dataset is discarded as outliers and the representative members at the end is actually a general representative of almost the entire dataset.

No. of datapoints	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2000	HDBSCAN	605	1.030	0.159	8278.394
	Agglomerative	175	0.196	-0.212	786.866
	k-medoids	175	6.322	-0.209	1079.195
	Spectral	175	80.008	-0.145	1187.477

Table 9: Results of the ConClu approach on 2000 datapoints.

It can be noticed that the DBCV score for the HDBSCAN algorithm increases consistently from 1000 datapoints to 2000 datapoints and so on. This could be attributed to the fact that more datapoints would

quite naturally give rise to denser clusters. More dense the inside of the cluster is as compared to its outside, the better is the DBCV score. This further reassures the fact that denser clusters would mean more number of elements in individual clusters. In other words, the representative member of the cluster would represent a more diverse amount of other cluster members.

No. of datapoints	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
5000	HDBSCAN	1.586	1.904	0.210	165884.854
	Agglomerative	175	1.345	-0.380	4945.410
	k-medoids	175	9.142	-0.427	8289.148
	Spectral	175	641.189	-0.359	5069.388

Table 10: Results of the ConClu approach on 5000 datapoints.

Also it is noteworthy how the time to calculate the DBCV score doesn't increase linearly on increasing the number of datapoints. Rather it increases in an exponential rate. This shows how it would take immensely long to compute the DBCV score of the entire dataset in the presence of one million datapoints in it. But since the results obtained on a smaller subset of the dataset can be interpolated to the entire dataset, evaluation hyperparameter optimization for the rest of the parameters are performed on 1000 datapoints. However, after deciding the final values for all the hyperparameters, the cluster representatives are generated taking into account all the datapoints under consideration as that no longer involves calculating the DBCV score anymore for evaluation.

5.2.2 Evaluation of Different Dimensionality Reduction Techniques

As mentioned in Ch. 4.5.1, it is essential to perform dimensionality reduction before clustering the objects in the dataset into groups. Two different dimensionality techniques are evaluated for this purpose, the PCA method and the t-SNE method.

PCA dim	PCA time (seconds)	PCA dim	PCA time (seconds)	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
512	32.921	154	11.249	HDBSCAN	169	0.459	-0.410	234.172
	41.230	154	14.891	Agglomerative	175	0.216	-0.151	209.954
	38.408	154	13.032	k-medoids	175	7.001	-0.184	212.482
	44.171	154	13.897	Spectral	175	46.703	-0.481	221.064

Table 11: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints with PCA as dimensionality reduction technique in both steps.

Tab. 11 shows the results on the ConClu approach for 1000 datapoints and applying PCA method as the dimensionality reduction technique in both steps. As the DBCV score depends on the number of nearest neighbor raised to the power of the negative of one divided by the number of features in the latent space representation as shown in Eq. 48 on Pg. 48, it is important to determine the to the maximal possible dimension of the representation after the second step of dimensionality reduction. Experiments are performed and the value is found to be 154 in this use case. Also keeping the first step of dimensionality reduction to be PCA, experiments are also conducted with t-SNE method as the second step. Tab. 12 shows the results on the ConClu approach for 1000 datapoints under such settings. The dimensions of the latent space representations in both the cases is set to 1024. it is to be noted that the results for all the clustering algorithms have better performance when t-SNE is used as the dimensionality reduction technique in the second step as compared to PCA. It can be attributed to the fact that the t-SNE algorithm preserves the spatial relationship between the datapoints. In other words, if datapoints are close to one another in a higher dimensional space, t-SNE tends to preserve this spatial closeness between points even in the lower dimensional space.

PCA dim	PCA time (seconds)	t-SNE dim	t-SNE time (seconds)	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
512	32.530	2	143.024	HDBSCAN	288	0.228	0.137	437.956
	38.832	2	141.566	Agglomerative	175	0.063	-0.057	191.887
	39.326	2	162.957	k-medoids	175	7.763	-0.120	192.580
	71.552	2	177.427	Spectral	175	28.831	-0.051	186.879

Table 12: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints with PCA as dimensionality reduction technique in the first step and t-SNE as the second.

Also it is to be noted that the HDBSCAN algorithm performs significantly bad in presence of PCA as compared to t-SNE as the second step of dimensionality reduction. This is also because of the property of t-SNE mentioned above which gives rise to denser clusters as compared to PCA.

No. of datapoints	t-SNE dim	t-SNE time (seconds)	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
1000	2	6.769	HDBSCAN	314	0.189	-0.120	337.313
	2	4.872	k-medoids	175	0.337	-0.132	131.622
	2	5.831	Spectral	175	7.592	-0.122	130.678
	2	4.478	Agglomerative	175	0.033	-0.085	128.358

Table 13: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints with only t-SNE as dimensionality reduction technique as the only step.

The effect of two step dimensionality reduction is also compared against only one step dimensionality reduction as shown in Tab. 13. A PCA followed by a t-SNE shows better results as compared to directly applying t-SNE just once. This is because of the fact that a multi-step dimensionality reduction helps in reducing the noises during the procedure. Thus, for the rest of the experiments the dimensionality reduction with PCA having number of components to be 512 in the first step is performed followed by applying the t-SNE method with 2 components as the next step.

5.2.3 Evaluation of Dimension of Feature Space Representations

Latent space representations are lower dimensional representations of the dataset. It is expensive to work with very high dimensional data. Thus, the dimension of the latent space representations is immensely important. It is necessary that the original high dimensional data is compressed to a lower dimensional latent space without severe loss of information in the process. Since the original dataset has the point clouds with 2048, for the sake of completeness of the experimentation, the evaluation is performed for all dimensions on a logarithmic scale of 2 from 2048 to down until 4. Tab. 14 shows the experiments when the dimension of the latent space representations is set to be 2048. As mentioned in Ch. 4.5.1, because of the very high number of features in the output of the network, a two-step dimensionality reduction is performed to suppress noise during the process of dimensionality reduction and also to facilitate faster calculation of the pairwise distance between the points. In line with the results of the previous experiments, the HDBSCAN clustering algorithm produces the best results. The learning rate is rate set to 0.001 with a decay factor of 0.1 throughout all the experiments in this subsection. A early stopping criteria is used to naturally terminate the training of the models on convergence. When the validation loss no long reduced for atleast 5 epochs, then the training is terminated. The number of epochs is set to a very high number of 200 for all the experiments so that the training is only stopped when early stopping criteria is triggered on model convergence.

PCA dim	PCA time (seconds)	t-SNE dim	t-SNE time (seconds)	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
512	18.507	2	3.594	HDBSCAN	289	0.127	0.189	243.297
	15.525	2	3.368	Agglomerative	175	0.036	-0.013	109.668
	16.595	2	3.709	k-medoids	175	0.222	-0.040	102.518
	18.593	2	3.918	Spectral	175	9.257	0.020	101.925

Table 14: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimensions of the latent space representations is 2048.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	289	105	0.189	243.297
Level 2	HDBSCAN	127	53	0.021	65.340

Table 15: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimensions of the latent space representations is 2048.

As observed in Ch. 5.2.1, in Tab. 15 too it can be seen that the performance of multi-level HDBSCAN algorithm deteriorates as compared to HDBSCAN but it is still comparable to spectral clustering and better than the rest. But larger dimensions of the latent space representations mean longer training time. When it is set to 2048, the model converges after the 50^{th} epoch according to the early stopping criteria mentioned above with a total training time of 137.4 hours which is quite high. So to evaluate the performance of the model with fewer dimensions of the latent space representations further experiments are conducted. Tab. 12 and 16 shows the experiments when the dimension of the latent space representations is set to be 1024 and 512 respectively. The number of semantic subgroups or parts in the objects is set to 64 in line with [59] to find the optimum number of dimensions of the latent space representations. Experiments on the number of semantic subgroups are included in Ch. 5.2.4.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	145.809	HDBSCAN	307	0.187	0.186	489.133
	6.575	k-medoids	175	0.532	-0.023	129.956
	4.989	Spectral	175	9.226	0.024	131.239
	6.823	Agglomerative	175	0.053	0.010	131.956

Table 16: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	307	93	0.186	489.133
Level 2	HDBSCAN	126	38	0.024	129.881

Table 17: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimensions of the latent space representations is 512.

Similarly, Tab. 17 records the performance of the multi-level HDBSCAN algorithm. In line with the previous observations, the results direct to similar conclusions in this case well with HDBSCAN algorithm having the best result followed by multi-level HDBSCAN and spectral clustering algorithm having comparable results with a large difference in the clustering time. The training time for the model with 1024 dimensions is 73 hours for 65 epochs. The training time for the model with 512 dimensions is 64 hours for 54 epochs. The performance for the model with dimension 512 is comparable to the model with dimension 2048 for a much lower training time. If a good enough result could be achieved with a significant decrease in the training time, it would mean a more cost efficient model. It is to be noted here that the performance of the HDBSCAN algorithm is worse for dimension 1024 as compared to both dimensions 2048 and 512. To achieve a conclusive result as to whether the performance of the HDBSCAN algorithm on the model with dimension 1024 is an outlier, further experiments are conducted to train models with fewer dimensions. Tab. 18 and Tab. 20 are the experiments for the model with dimensions 256 and 128 respectively with rest of the parameters being same as mentioned above. Tab. 19 shows the performance of the multi-level HDBSCAN algorithm when the model ad dimension 256. The training time for the model with dimension 256 is 42 hours for 50 epochs.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	5.901	HDBSCAN	293	0.725	0.164	718.808
	6.206	k-medoids	175	0.471	-0.078	136.173
	6.106	Spectral	175	11.077	0.003	133.593
	5.539	Agglomerative	175	0.031	-0.020	132.193

Table 18: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 256.

Whereas, the training time for the model with dimension 256 is 32 hours for 40 epochs. It is observed that there is no significant change in the training time as compared to training time of the model with dimension of 512 to 256 as it is observed when the dimension changed from 2048 to 512. In the latter, when the dimension changed by a factor of 4, the training time reduced by almost a factor of 2.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	293	95	0.164	718.808
Level 2	HDBSCAN	122	41	0.045	198.859

Table 19: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimensions of the latent space representations is 256.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.036	HDBSCAN	307	0.121	0.174	354.460
	3.689	k-medoids	175	5.255	-0.037	177.251
	3.295	Spectral	175	5.511	0.024	116.696
	3.531	Agglomerative	175	0.057	-0.033	94.799

Table 20: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 128.

Whereas, when the dimension changed from 512 to 128, there is a change in the dimension by a factor of 4 but the training time does not reduce comparably. Also, the performance of the HDBSCAN algorithm is better with the model of dimension 512 by 2% and 1% as compared to models with dimension 256

and 128 respectively. Until now, the number of semantic subgroups in the objects are set to 64. In other words, each dimension of the feature representation of the point cloud is to belong to 1 of the 64 semantic subgroups or parts. Thus, when the dimension of of the latent space representation of the objects is atmost 64, it is not realistic to have 64 semantic subgroups. So to train the models with dimensions of latent space representation to be less than equal to 64, the number of semantic subgroups is to chosen to be half of the number of dimensions. So the model of dimension 64 has K set to be 32, the model of dimensions 32 has K set to be 16 and so on. Thus, Tab. 21 and Tab. 22 are the experiments for the model with dimensions 256 and 128 respectively with rest of the parameters being same as mentioned above. The training time for the model with 64 dimensions is 45 hours for 48 epochs. The training time for the model with 32 dimensions is 23 hours for 39 epochs.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.059	HDBSCAN	286	0.538	0.148	639.097
	3.086	k-medoids	175	3.143	-0.068	145.711
	3.189	Spectral	175	8.760	-0.020	152.836
	3.914	Agglomerative	175	0.032	-0.017	157.152

Table 21: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 64.

As expected, the performance of these models have decreased by 4% and 3% respectively as compared to the model with dimensions 512. When the number of dimensions have decreased by a factor of 16, the amount of training time has only reduced by a factor of 3 when the dimensions have change from 512 to 32. The learning rate for all the experiments in this subsection is set to 0.001 with a decay factor of 0.1. A batch size of 128 is chosen for these experiments as that is the maximum possible batch size that could be executed without running into out of memory issues in the given hardware setup.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	2.748	HDBSCAN	273	0.222	0.158	562.223
	2.917	k-medoids	175	1.434	-0.021	141.890
	2.985	Spectral	175	6.190	0.016	107.808
	2.501	Agglomerative	175	0.031	0.023	93.092

Table 22: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 32.

Experiments are also conducted for training the models with dimensions 16, 8 and 4 for the completeness

of the experiments. Since the dimensions of the latent space representations are to be categorized into subgroups, the minimum number of possible subgroups is 2. Therefore, a model with the dimension of the latent space representations as 2 is not realistic. So the minimum number of dimensions of the latent space representations is 4.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	2.673	HDBSCAN	287	0.419	0.168	621.601
	3.240	k-medoids	175	2.276	-0.023	147.403
	3.005	Spectral	175	6.064	0.027	109.908
	3.469	Agglomerative	175	0.031	0.005	92.398

Table 23: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 16.

Tab. 23, 24 and Tab. 25 are the experiments for the model with dimensions 16, 8 and 4 respectively with rest of the parameters being same as mentioned above.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	2.851	HDBSCAN	290	0.187	0.135	551.979
	2.529	k-medoids	175	7.996	-0.028	146.129
	2.693	Spectral	175	3.949	-0.026	115.756
	2.714	Agglomerative	175	0.033	-0.020	93.977

Table 24: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 8.

The training time for the model with 16 dimensions is 16 hours for 29 epochs. The number of subgroups for this model is set to be 8 as explained earlier. The training time for the model with 8 dimensions is 12 hours for 28 epochs. The number of semantic groups is set to be 4. The training time for the model with 4 dimensions is 13 hours for 31 epochs. The number of semantic groups is set to be 2. The performance is seen to decline rapidly as the number of dimensions decrease. This could also be attributed to the fact that models with such low dimensions aren't powerful enough to capture the correct representation of such large datasets accurately. Thus, weighing the performance of the models, their validation loss and the training time, the model with dimensions 512 is seen to have the best performance with a practical training time to carry on experiments for the rest of the parameters. An overview of the training and validation loss of the models is shown in Fig. 5.2.1. The validation loss for models with dimensions 4 and 8 are quite high as compared to the rest of the models. This further proves that the models with such low dimensions aren't

complex enough to accurately represent the objects in the dataset.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	2.876	HDBSCAN	296	0.550	0.101	526.278
	3.102	k-medoids	175	0.555	-0.102	193.242
	2.432	Spectral	175	1.952	-0.071	92.449
	2.517	Agglomerative	175	0.030	-0.116	92.347

Table 25: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 4.

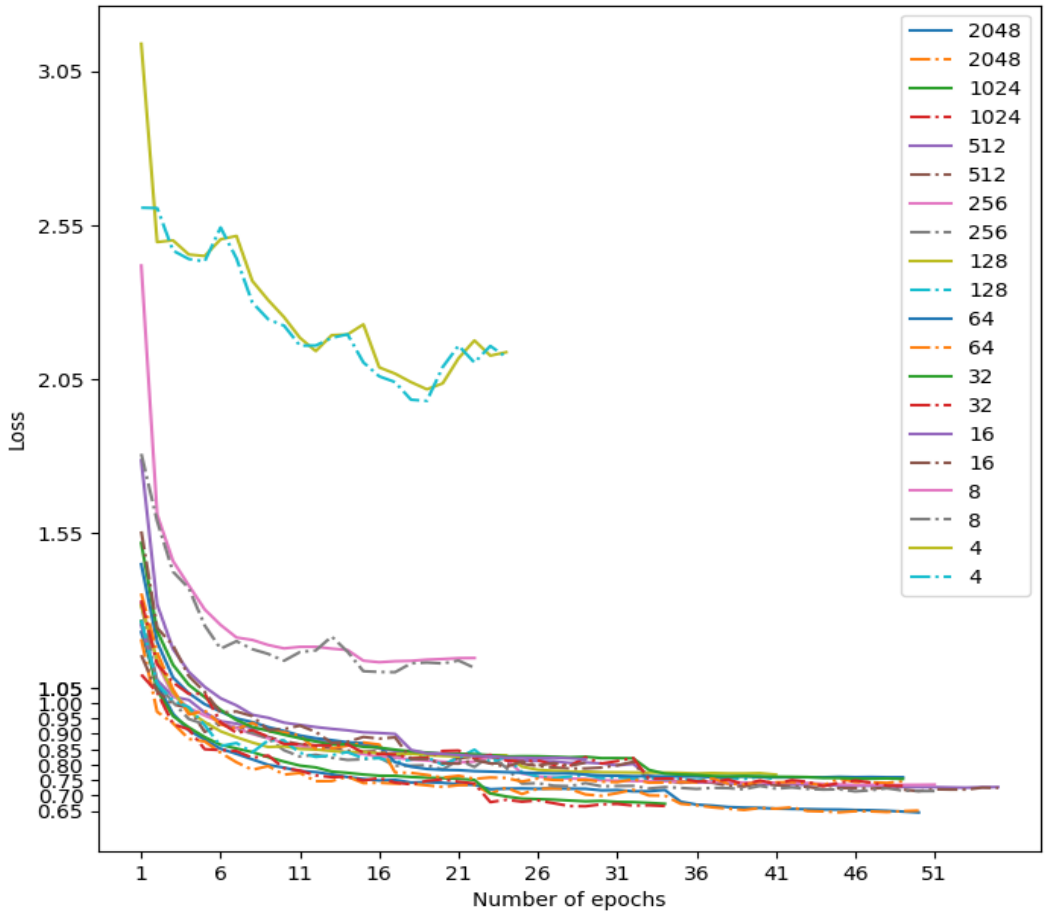

Figure 5.2.1: Overview of the validation loss of the models trained with different dimensions. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent the dimension of the latent space representations of the models.

5.2.4 Evaluation of the Number of Semantic Subgroups

Once the number of dimensions of the latent space representations is set to 512, next the optimum number of semantic subgroups is evaluated. Since the dataset consists of a large variety of objects, it is not possible to determine a fixed number of categories the objects are to be partitioned into. Thus, the evaluation of the number of subcategories is essential so that the ultimate goal of achieving a set of representative members of the dataset which generalizes the entire dataset quite well is achieved.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.555	HDBSCAN	279	0.620	0.185	689.203
	3.731	k-medoids	175	0.176	-0.029	101.171
	3.349	Spectral	175	7.221	0.042	100.748
	3.663	Agglomerative	175	0.031	-0.007	102.203

Table 26: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 32.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	279	96	0.185	689.203
Level 2	HDBSCAN	122	51	0.043	203.029

Table 27: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representations is 512 and the number of semantic subgroups is 32.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.797	HDBSCAN	291	0.140	0.182	244.248
	3.557	k-medoids	175	0.257	-0.50	100.699
	3.480	Spectral	175	6.331	0.030	103.005
	3.731	Agglomerative	175	0.033	-0.007	104.203

Table 28: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 16.

Tab. 16 on Pg. 58 already shows the performance of the model when the dimensions of the latent space representation is 512 and the number of semantic subgroups is 64. Tab. 26 records the performance of the model when the number of semantic subgroups is set to 32. Tab. 27 shows the performance of the multi-level HDBSCAN algorithm on this model. The rest of the parameters for the experiments remains constant throughout this subsection. It is seen that the performance of the model with $k = 32$ is comparable to the model with $k = 64$. The DBCV score for the model with $k = 32$ is 0.185 while that of the model with $k = 64$ is 0.186. In other words, even with fewer semantic subgroups the model could achieve comparable results. Moreover, the model for $k = 32$ requires a training time of 28 hours for 39 epochs. Whereas, the model for $k = 64$ requires a training time of 64 hours for 54 epochs. Thus, reducing the value of k by a factor of 2 also led t the decrease in the training time by a factor of 2.29. This also re-ensures the ultimate goal of the task. Fewer subcategories in the objects means the model will learn more "general" features of the objects and not the extremely intricate and instance-specific object features. This would help in objects of the dataset which actually capture the overall variance in the dataset. Tab. 28 and Tab. 30 shows the performance of the model when the dimension of the latent space representation is 512 and the number of semantic subgroups is 16 and 8 respectively. Tab. 29 records the performance of multi-level HDBSCAN algorithm when the number of semantic subgroups is set to 16. The dimension of the latent space representations of all these models are 512. a batch size of 128 is used to train the models as mentioned earlier. The model for $k = 16$ requires a training time of 37 hours for 63 epochs until convergence. Whereas, the model for $k = 8$ requires a training time of 31 hours for 40 epochs. It is seen that the model with $k = 16$ gives almost comparable results to that of the model with $k = 32$. But it is observed that the model with $k = 16$ takes longer training time for convergence as compared to the model with $k = 32$. It is further noted that the performance of the model with $k = 8$ degrades by a factor of 4.3% as compared to that of the model with $k = 8$.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	291	110	0.182	244.248
Level 2	HDBSCAN	117	38	0.033	57.618

Table 29: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representations is 512 and the number of semantic subgroups is 16.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.018	HDBSCAN	291	0.140	0.139	235.023
	3.091	k-medoids	175	0.191	-0.092	98.415
	3.101	Spectral	175	6.810	-0.027	96.898
	3.177	Agglomerative	175	0.038	-0.030	97.490

Table 30: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 8.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.259	HDBSCAN	301	0.115	0.186	332.309
	4.356	k-medoids	175	0.172	-0.049	205.339
	3.344	Spectral	175	11.716	0.024	166.501
	5.815	Agglomerative	175	0.049	-0.041	170.280

Table 31: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 128.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.295	HDBSCAN	296	0.122	0.103	269.228
	3.577	k-medoids	175	0.308	-0.045	139.550
	3.751	Spectral	175	11.623	-0.004	167.611
	5.013	Agglomerative	175	0.032	-0.005	169.509

Table 32: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 4.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	4.469	HDBSCAN	289	0.126	0.085	320.217
	3.364	k-medoids	175	5.662	-0.118	155.136
	3.349	Spectral	175	5.662	-0.089	115.978
	3.029	Agglomerative	175	0.038	-0.100	98.683

Table 33: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512 and the number of semantic subgroups is 2.

Tab. 31 shows the performance of the model when the dimension of the latent space representation is 512 and the number of semantic subgroups is 128. The model requires a training time of 58 hours for 36 epochs until convergence is reached.

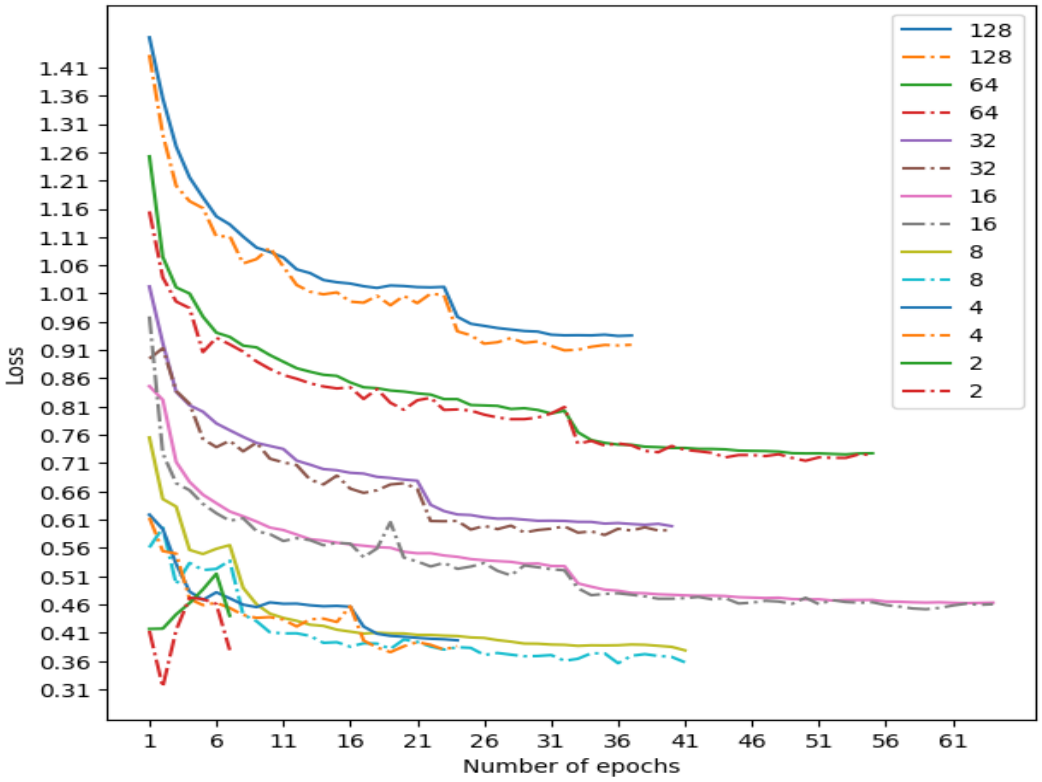

Figure 5.2.2: Overview of the validation loss of the models trained with different values of semantic subgroups when dimension of the latent space representation is fixed to 512. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent number of semantic subgroups in the objects.

It can be seen that the performance for the model with $k = 128$ is comparable to the model with $k = 32$. Also, the training time for the model with $k = 32$ is much less compared to that of the model with $k = 128$. Tab. 32 and Tab. 33 shows the performance of the model when the dimension of the latent space representation is 512 and the number of semantic subgroups is 4 and 2 respectively. The training time for the model with $k = 4$ is 17 hours for 23 epochs. Similarly, the training time for the model with $k = 2$ is 5 hours for 6 epochs. It can be observed that the performance of the models decline significantly when the number of semantic subgroups is too low. Thus, it can be concluded that the performance of the models increases with the increase in the number of semantic subgroups and then attains a saturation in the performance. In other words, the number of semantic subgroups doesn't effect the performance of the models as long as they are enough parts the objects are partitioned into. Tab. 31 shows the performance of the model when the dimension of the latent space representation is 512 and the number of semantic subgroups is 128. The model requires a training time of 58 hours for 36 epochs until convergence is reached. It can be seen that the performance for the model with $k = 128$ is comparable to the model with $k = 32$. Also, the training time for the model with $k = 32$ is much less compared to that of the model with $k = 128$.

5.2.5 Evaluation of the Batch Size for Training

One of the most crucial hyperparameters to adjust in deep learning methods is batch size. Attempting to optimize convex functions involves an intrinsic trade-off between the advantages of larger and smaller

batch sizes. One way to ensure convergence to the global optima of the objective function is to use a batch size equal to the entire dataset. But doing so comes at the expense of a slower empirical convergence to that optimal state. Also, it practically infeasible to load the entire dataset like the ABC dataset [47] in this use-case all at once. However, it has been demonstrated empirically that employing lower batch sizes leads to a faster convergence to "good" solutions. Smaller batch sizes enable the model to "start learning before having to see all the data," which provides an easy explanation for this. A smaller batch size has the drawback of not guaranteeing that the model will eventually come to the global optima. It will oscillate between the global optima, remaining outside a certain ε-ball of the optima, where ε is determined by the batch size to dataset ratio. Therefore, as mentioned in Ch. 4.5.2, the largest batch size that could be fit in the GPU with the higher capacity is observed to be 256. Thus, to facilitate the training of multiple models for experimentation in the limited time frame parallelly, the performance of the models with batch sizes 64, 128 and 256 are evaluated. The dimension of the latent space representations of the all the models in this subsection is set to 512 and $k = 32$ as discussed above. The learning rate is 0.001 and the decay factor for the learning rate is 0.1 as mentioned earlier. Tab. 26 on Pg. 63 already shows the performance of the model when batch size is 128. Tab. 34 shows the performance of the model when the batch size is 256. The training time for this model is 40 hours for 43 epochs. Tab. 35 records the performance of the multi-level HDBSCAN algorithm.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	5.855	HDBSCAN	291	0.178	0.154	382.814
	6.034	k-medoids	175	0.384	-0.040	146.189
	5.365	Spectral	175	13.660	0.026	145.207
	5.513	Agglomerative	175	0.040	-0.011	167.093

Table 34: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 256.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	291	92	0.154	382.814
Level 2	HDBSCAN	126	53	0.038	101.948

Table 35: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 256.

It is seen that this algorithm gives the second-best clustering results after HDBSCAN algorithm. And the result is slightly better than that of spectral clustering. Quite naturally, the model takes longer to converge for a larger batch size. But performance of the model does not seem to improve as expected as compared to the model with batch size 128. The DBCV of the HDBSCAN algorithm is 0.185 for the earlier model, whereas for the model with batch size it appears to suffer a downfall in performance by 3.1%.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	5.454	HDBSCAN	300	1.080	0.177	730.026
	6.732	k-medoids	175	7.606	-0.056	602.923
	6.326	Spectral	175	51.657	0.008	492.017
	6.129	Agglomerative	175	0.489	-0.007	481.838

Table 36: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 64.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	300	85	0.177	730.026
Level 2	HDBSCAN	128	41	0.035	210.596

Table 37: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the batch size for training the model is 64.

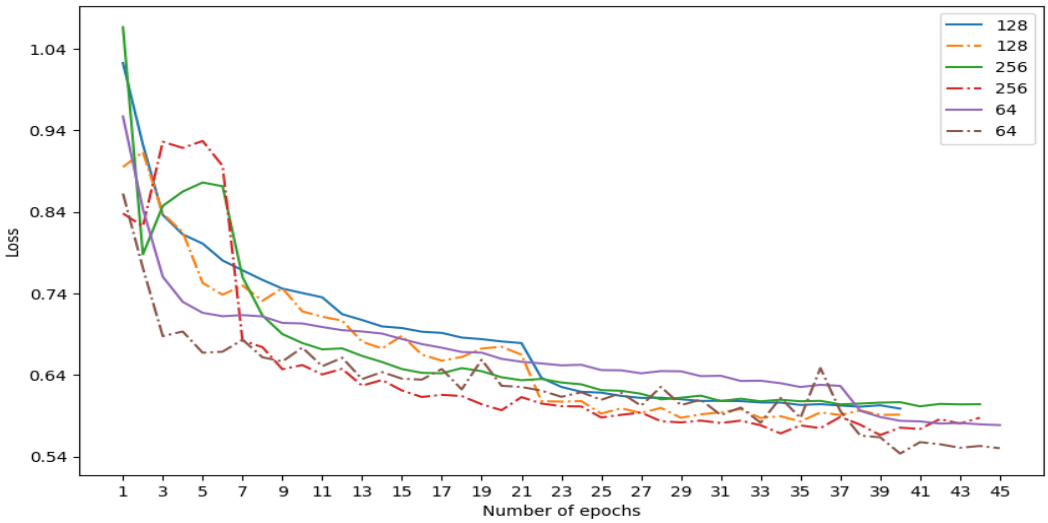

Figure 5.2.3: Overview of the validation loss of the models trained with different batch sizes when dimension of the latent space representation is fixed to 512 and the number of semantic subgroups to 32. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent the batch size used for training the models.

On the other hand, Tab. 36 shows the performance of the model when the batch size is 64. Tab. 37 documents the performance of the multi-level HDBSCAN algorithm on this model. The training time for this model is 36 hours for 44 epochs. The performance of the models is seen to improve from batch size 64 to 128 as expected. Fig. 5.2.3 gives an overview of the training and validation loss of the models with changing batch size. It can be observed that the validation loss of the models are quite comparable on convergence which can justify the reason behind no significant improvement of the models with different batch sizes.

5.2.6 Evaluation of the Effect of Early Stopping

Determining an appropriate finite number of training epochs is a major challenge while developing machine learning models. If a model has too less epochs, then it would result to an underfit model whereas if the number of epochs is too large, then the model tends to memorize the training dataset and doesn't generalize well for unseen data [30]. Therefore, the method of termination of the training when the model no longer improves its performance on the validation dataset is known as the early stopping mechanism. The patience parameter plays a very vital role in this mechanism. The model does not immediately terminate the training as soon as there is an increase in the validation loss. Rather it waits for a number of consecutive epochs defined by the patience parameter. And if the validation loss no longer decreases as compared to the least recorded validation loss, the training is then terminated.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.643	HDBSCAN	289	0.112	0.132	260.046
	3.614	k-medoids	175	0.101	-0.054	115.333
	3.535	Spectral	175	5.567	-0.008	114.561
	3.765	Agglomerative	175	0.299	-0.033	106.243

Table 38: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the patience for early stopping is 3, dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 128.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	289	98	0.132	260.046
Level 2	HDBSCAN	119	41	-0.003	63.805

Table 39: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the patience for early stopping is 3.

A patience of 3, 5, 7 and 10 have been evaluated in this subsection. The learning rate of 0.001 and a decay factor of 0.1 as mentioned earlier is used throughout the experiments recorded below. Tab. 38 shows the performance of the model with patience 3. Tab. 39 records the performance of multi-level HDBSCAN algorithm on this model. The model takes a training time of 10 hours for 10 epochs until convergence. Tab. 26 on Pg. 63 already shows the performance of the model when the patience parameter is set to 5.

It requires a training time of 28 hours for 39 epochs. As expected the model with less patience has lower training time compared to that with a high patience. But it is that the performance of the model with less patience is worse as compared to that with higher patience. There is a 5% decline in the quality of the performance.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.731	HDBSCAN	298	0.126	0.199	293.794
	3.257	k-medoids	175	0.134	-0.037	116
	3.340	Spectral	175	9.391	0.011	94.393
	2.788	Agglomerative	175	0.032	-0.018	93.149

Table 40: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the patience for early stopping is 7, dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 128.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	298	99	0.199	293.794
Level 2	HDBSCAN	117	40	0.013	64.412

Table 41: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the patience for early stopping is 7.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.338	HDBSCAN	308	0.126	0.190	330.257
	3.191	k-medoids	175	0.288	-0.051	166.315
	3.072	Spectral	175	7.785	0.013	156.930
	3.441	Agglomerative	175	0.032	-0.041	159.887

Table 42: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the patience for early stopping is 10, dimension of the latent space representations is 512, the number of semantic subgroups is 32 and the batch size for training the model is 128.

Tab. 40 and Tab. 42 shows the performance of the models with patience 7 and 10 respectively. The former model takes a training time of 46 hours for 49 epochs. As expected, the model takes more number of epochs until convergence as compared to the model with patience 5. Also, the results for this model outperform the performance shown by the model with patience 3 and 5 as expected. Tab. 41 shows the performance of multi-level HDBSCAN algorithm on this model. The DBCV score attained by the HDBSCAN algorithm is 0.199 by this model as coma pared to the next best model with patience 5 which has the DBCV score of 0.185. The model with patience 10 requires a training time of 30 hours for 42 epochs. But the performance of the model is seen to drop a bit as compared to the model with patience 7. Fig. 5.2.4 gives an overview of the training and validation loss of the models for different values of the patience parameter. As expected, the model with patience 10 has the least validation loss on convergence. Thus, weighing the training time of the models and their performance for the clustering task, the model with patience 7 is observed to have the best results.

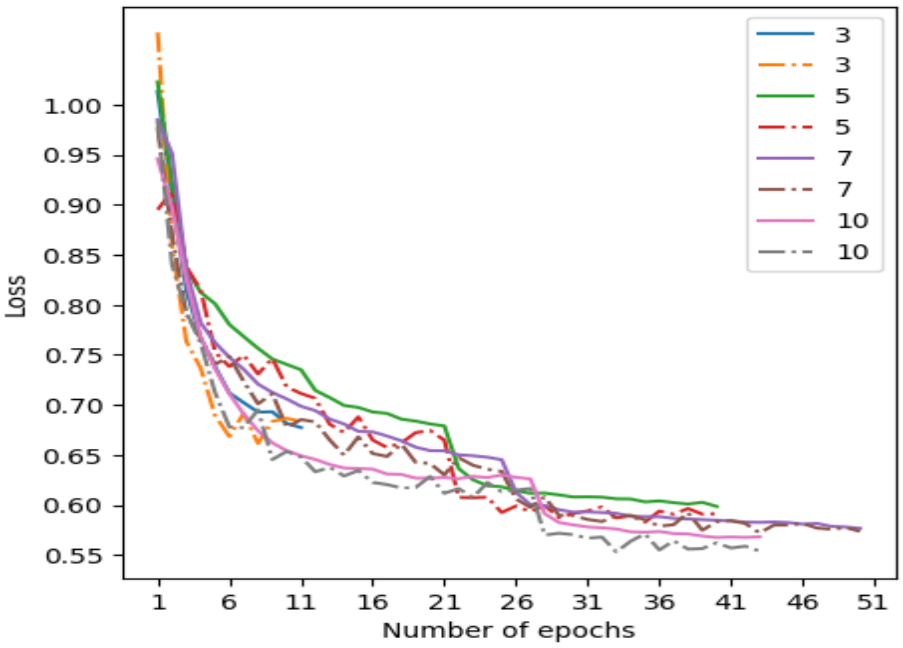

Figure 5.2.4: Overview of the validation loss of the models trained with different values of the patience parameter in early stopping when dimension of the latent space representation is fixed to 512, the number of semantic subgroups is 32 and batch size is 128. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent the value of the patience parameter.

5.2.7 Evaluation of the Effect of Learning Rate

The learning rate is one of the crucial hyperparameters when building a machine learning model. When the model weights are updated, it regulates how much the model is to be altered after calculating the estimated error. Carefully selecting the learning rate is extremely necessary because if the learning rate is too small, then the training process will be extremely low and it can get stuck at a local minima. Whereas, if the learning rate s too high, then the model can oscillate around the global optima without reaching a convergence. Tab. 22 on Pg. 60 already shows the performance of the model when the learning rate is set to 0.001. Tab. 43 documents the performance of the model when the learning rate is set to 0.01. The model requires a training time of 28 hours for 40 epochs. It takes almost the same number of epochs to converge as does the model with learning rate 0.001. Furthermore, it can be observed that the performance of the model deteriorates when the learning rate is changed to 0.01 from 0.001. There is a 6% degradation in performance when the learning rate is increased by a factor of 10. Tab. 44 shows the performance of the

multi-level HDBSCAN algorithm on this model.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.417	HDBSCAN	285	0.109	0.128	315.829
	3.455	k-medoids	175	0.246	-0.038	147.585
	3.270	Spectral	175	6.347	-0.057	156.651
	3.844	Agglomerative	175	0.037	-0.083	162.778

Table 43: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the learning rate is 0.01.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	285	107	0.128	315.829
Level 2	HDBSCAN	121	42	-0.049	62.360

Table 44: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the learning rate is 0.01.

5.2.8 Evaluation of the Effect of Decay Factor of the Learning Rate

Keeping the learning rate and the rest of the parameters constant, experiments are performed to evaluate the effect of the decay factor of the learning rate on the performance of the model. The decay factor determines the value of the new learning rate as shown in Eq. 57 when the metrices for the performance of the model stops improving.

$$new\ learning\ rate = decay\ factor \times current\ learning\ rate \tag{57}$$

Tab. 22 on Pg. 60 already shows the performance of the model when the learning rate is set to 0.001 and the decay factor is 0.1. Tab. 45 shows the performance of the model when the learning rate remains constant at 0.001 but the decay factor is change to 0.5. Tab. 46 shows the performance of multi-level HDBSCAN algorithm on this model. The performance of this model is seen to improve as compared to model with decay factor 0.1. There is almost a 3% increase in the performance of the model. The decay factor is further increased and Tab. 47 records the performance of the model when the decay factor is 0.9. All the remaining parameters of the model remains constant. The model takes a training time of 20 hours for 26 epochs until convergence. A decrease in performance of the model is observed as compared to the model with decay rate 0.5.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.351	HDBSCAN	303	0.113	0.212	334.613
	3.734	k-medoids	175	0.273	-0.033	164.001
	3.192	Spectral	175	6.865	0.003	109.385
	3.722	Agglomerative	175	0.043	-0.001	92.632

Table 45: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the learning rate is 0.001 and the decay factor is 0.5.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	303	71	0.212	334.613
Level 2	HDBSCAN	118	34	0.009	62.142

Table 46: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the learning rate is 0.001 and the decay factor is 0.5.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.265	HDBSCAN	287	0.114	0.184	225.236
	3.298	k-medoids	175	0.144	0.010	95.764
	4.107	Spectral	175	14.298	0.036	116.901
	3.297	Agglomerative	175	0.033	0.017	96.614

Table 47: Results of the ConClu approach with PointNet autoencoder on 1000 datapoints when the learning rat is 0.001 and the decay factor is 0.9.

Fig. 5.2.5 gives an overview of the training and validation loss of the models for learning rate and decay factor of the learning rate used during training the models. Even with a lower higher learning rate, the model takes almost the same number of epoch until convergence. But it can be noticed that the model reaches a comparable validation loss quite early on as compared to the model with a lower learning rate. The model with learning rate 0.001 as decay factor 0.9 takes the least training time.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	287	80	0.184	225.236
Level 2	HDBSCAN	120	46	0.044	56.283

Table 48: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the learning rate is 0.001 and the decay factor is 0.9.

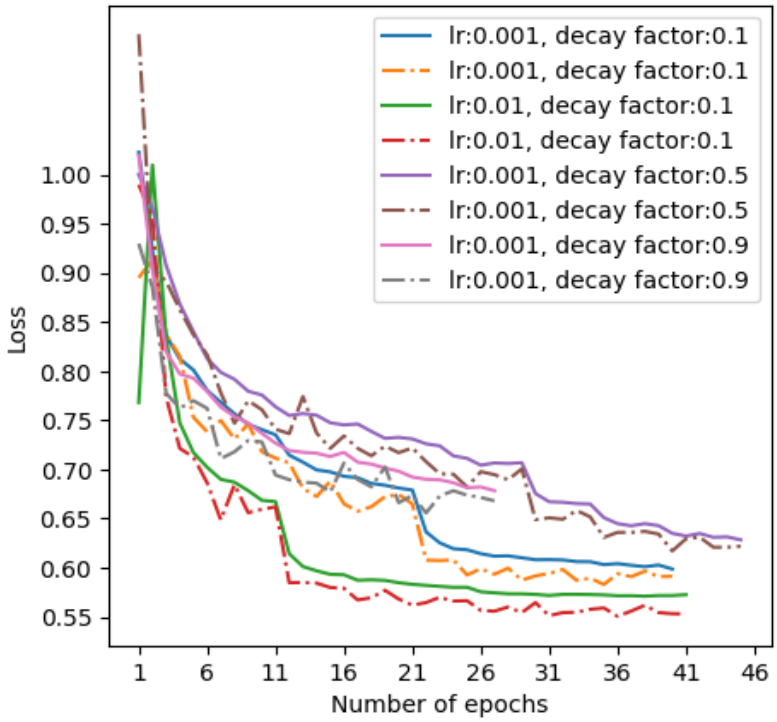

Figure 5.2.5: Overview of the validation loss of the models trained with different values of learning rate and decay factor of the learning rate when dimension of the latent space representation is fixed to 512, the number of semantic subgroups is 32 and batch size is 128. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent the learning rate and decay factor of the learning rate.

5.3 ConClu Approach with Dynamic Graph Convolutional Neural Network as Backbone

As mentioned in Ch. 4, two feature representation learning backbones are used for the experiments. Experiments with the DGCNN as the backbone is recorded here. The performance of all the clustering algorithms are evaluated.

5.3.1 Evaluation of dimension of feature space representations

As discussed earlier, the dimension of the latent space representation is an extremely crucial parameter while training the network. Since the original dataset has the point clouds with 2048 points in each of them, for the sake of completeness of the experimentation, the evaluation are performed for all dimensions on a

logarithmic scale of 2 from 1024 to down until 4. Tab. 49 shows the experiments when the dimension of the latent space representations is set to be 1024. Tab. 50 shows the performance of multi-level HDBSCAN model on this model. For all the models in this subsection, the number of nearest neighbors requires for a DGCNN graph is set to 20 based on [59].

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.022	HDBSCAN	295	0.121	0.112	228.925
	3.519	k-medoids	175	0.151	-0.067	92.043
	3.212	Spectral	175	6.712	-0.038	92.080
	3.683	Agglomerative	175	0.037	-0.089	124.850

Table 49: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 1024.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	295	89	0.112	228.925
Level 2	HDBSCAN	132	40	-0.035	86.639

Table 50: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representation is 1024.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.214	HDBSCAN	299	0.125	0.129	327.926
	3.739	k-medoids	175	0.248	-0.060	162.210
	3.212	Spectral	175	6.712	-0.038	92.080
	3.683	Agglomerative	175	0.037	-0.089	124.850

Table 51: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 512.

It is observed that training the models with the DGCNN backbone takes extremely large training time. The model with the dimension of latent space representation of 1024 takes 168 hours for 37 epochs until con-

vergence. This is more than twice the time requires for the model with PointNet autoencoder as backbone, which requires 73 hours for 65 epochs. This is because of the dynamic graph creation step involved for this method in each epoch before every convolutional layer. Also the performance of this model is a little worse as compared to the model with PointNet autoencoder as the backbone with same dimension for the latent space representation, the performance of which is recorded in Tab. 7 on Pg. 53. Tab. 51 records the performance of the model when the dimension of the latent space representation is 512.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	299	119	0.129	327.926
Level 2	HDBSCAN	126	45	-0.071	64.300

Table 52: Results of the ConClu approach on 1000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representation is 512.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.374	HDBSCAN	278	0.112	0.098	240.169
	3.673	k-medoids	175	0.171	-0.151	103.629
	3.579	Spectral	175	6.933	-0.078	105.700
	3.722	Agglomerative	175	0.044	-0.001	92.631

Table 53: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 256.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.498	HDBSCAN	289	0.118	0.103	250.010
	3.839	k-medoids	175	0.131	-0.116	106.311
	3.559	Spectral	175	6.662	-0.108	103.972
	3.590	Agglomerative	175	0.045	-0.099	104.450

Table 54: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 128.

Tab. 52 documents the performance of multi-level HDBSCAN algorithm on this model. All other parameters of the model remains constant. This model takes a training time of 231 hours for 60 epochs. This is four times the time requires by the model when PointNet autoencoder as the backbone. Even with such high training time, the performance of the model is seen to deteriorate by atleast 3% in case of HDBSCAN algorithm and atmost 5% in case of agglomerative clustering algorithm. Table. 53 records the performance of the model when the dimension is further reduced to 256. On reducing the dimension by a factor of 2, the model training time requires by the model also decreased by a factor of 2. Now the model requires a training time of 120 hours for 31 epochs. But unlike the model with the PointNet autoencoder as backbone, this model had a degradation in the performance when the dimension is reduced to 256. There is a 3% decrease in performance for the HDBSCAN algorithm and roughly 5% degradation for the k-medoids algorithm and spectral clustering algorithm. However, the performance of the agglomerative clustering algorithm remains almost similar. For the purpose of completeness of the evaluation, the dimension of the model is further reduced to 64, 32, 16, 8 and 4. Tab. 55 and Tab. 56 are the models with dimension 64 and 32 respectively.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.430	HDBSCAN	289	0.113	0.083	253.259
	3.572	k-medoids	175	0.137	-0.074	107.411
	3.384	Spectral	175	6.879	-0.090	105.169
	3.298	Agglomerative	175	0.030	-0.104	104.248

Table 55: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 64.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.390	HDBSCAN	284	0.179	0.088	308.699
	4.151	k-medoids	175	0.304	-0.098	178.198
	4.199	Spectral	175	34.680	-0.065	185.643
	3.067	Agglomerative	175	0.032	-0.110	99.183

Table 56: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 32.

As mentioned earlier, early stopping criteria is used during the training of the models to prevent overfitting. A patience of 5 is used for all the models in this subsection. Similar to the setup of the ConClu approach with PointNet autoencoder as backbone. A very high number of epochs is set for the training so that the training always terminates only when the early stopping criteria is triggered. Thus, the number of epochs is

set to 200. The model with dimension 64 takes a training time of 90 hours for 33 epochs. The performance of the model is seen to deteriorate as compared to that of the models of higher dimension. Tab. 56 records the performance of the model when the dimension of the laten space representation is set to 32.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	3.204	HDBSCAN	286	0.117	-0.078	235.378
	3.317	k-medoids	175	0.144	-0.135	100.929
	3.133	Spectral	175	3.587	-0.059	102.188
	3.225	Agglomerative	175	0.033	-0.100	99.532

Table 57: Results of the ConClu approach with DGCNN backbone on 1000 datapoints when the dimension of the latent space representation is 4.

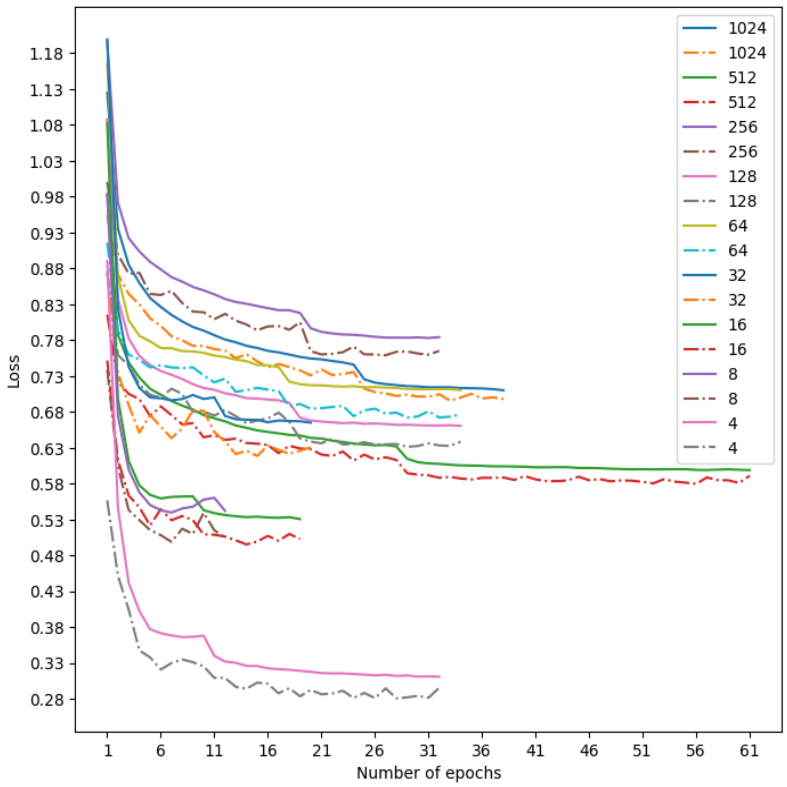

Figure 5.3.1: Overview of the training and validation loss of the models trained with different values of dimension of the latent space representation when the number of nearest neighbors is fixed to 20. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The numbers in the legend represent the dimension of the latent space representations.

The model requires a training time of 70 hours for 19 epochs until convergence. The performance of the model is quite comparable to the model with dimension 64. Further reducing the dimension of the latent space representations degrades the performance of the models even more. Fig. 5.3.1 gives an overview of all the models trained with DGCNN as the backbone of the model for different sizes of the latent space representation. It can be observed that the validation loss is least when the model has dimensionality 4. But looking into the performance of the model while clustering the dataset, it can be seen that the performance is significantly worse. This could be a result of overfitting of the model. In the absence of enough dimensions of the latent space representations, the model is not properly calibrated. It learns the noise in the training dataset which makes it more confident in the predictions in the validation set which reduces the loss. But these "confident" predictions are actually incorrect predictions. Thus, there is a significant decrease in performance while clustering the dataset. Tab. 57 records the performance of the model when the dimensionality of the latent space representations is set to 4.

5.3.2 Evaluation of the effect of increasing datapoints

It is important to check that the results obtained on a small subset of the entire dataset can be relied upon for deriving conclusions on the entire dataset. Tab. 51 on Pg. 75 already shows the results of the model for 1000 datapoints.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	7.034	HDBSCAN	591	0.225	0.182	3.596.855
	6.511	k-medoids	175	0.213	-0.202	409.159
	6.730	Spectral	175	26.729	-0.129	410.735
	6.666	Agglomerative	175	0.102	-0.17	399.792

Table 58: Results of the ConClu approach with DGCNN backbone on 2000 datapoints when the dimension of the latent space representation is 512.

Tab. 58 records the performance of the model when the number of datapoints are increased to 2000. It can be seen that the time for dimensionality reduction with t-SNE increases linearly on increasing the number of datapoints. But as mentioned earlier, the time for calculating the DBCV score increases exponentially and not linearly. Similar to the observations with the PointNet autoencoder as backbone, the performance increases for the HDBSCAN algorithm but not quite for the rest of the algorithms. This is because of the fact that more datapoints invariably mean more dense clusters when the number of clusters remain constant. Also, the DBCV score being a density based clustering validation index shows a better score for density based clustering algorithms.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV time (seconds)
Level 1	HDBSCAN	591	193	0.129	327.926
Level 2	HDBSCAN	223	73	-0.054	604.703

Table 59: Results of the ConClu approach on 2000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representation is 512.

t-SNE dim	t-SNE time	Clustering Algorithm	No. of Clusters	Clustering Time (seconds)	DBCV score	DBCV Time (seconds)
2	17.516	HDBSCAN	1.486	0.647	0.185	120280.203
	18.025	k-medoids	175	0.550	-0.409	2.696.609
	6.730	Spectral	175	26.729	-0.129	410.735
	6.666	Agglomerative	175	0.102	-0.17	399.792

Table 60: Results of the ConClu approach with DGCNN backbone on 5000 datapoints when the dimension of the latent space representation is 512.

Level	Algorithm	No. of Clusters	No. of Outliers	DBCV score	DBCV Time (seconds)
Level 1	HDBSCAN	1486	510	0.185	120280.203
Level 2	HDBSCAN	609	176	-0.026	20.460.755
Level 3	HDBSCAN	265	106	-0.205	4595.876

Table 61: Results of the ConClu approach on 5000 datapoints for multi-level HDBSCAN clustering algorithm when the dimension of the latent space representation is 512.

Tab. 59 shows the performance of multi-level HDBSCAN algorithm on this model. Tab. 60 records the performance of the model when the number of datapoints are further increased to 5000. Tab. 61 shows the performance of multi-level HDBSCAN algorithm on this model. As the number of datapoints increase, the number of iterations of the multi-level HDBSCAN algorithm requires to reach a total number of clusters less than or equal to 175 also increases. Hence, it requires 3 levels for 5000 objects. It is to be noted that the performance of the multi-level HDBSCAN algorithm improves with increase in the number of datapoints.

5.4 Comparison Between the PointNet Autoencoder Backbone and the Dynamic Graph Convolutional Neural Network backbone

As observed earlier, the training time for the model with DGCNN as the backbone is quite higher as compared to the models with PointNet autoencoder as backbone. And even with higher training time, the performance of the models are significantly worse. Amongst the models with DGCNN as the backbone, the one with dimensionality 512 has the best results. But it requires a training time of 231 hours (i.e. almost 10 days) as compared to the model with PointNet autoencoder as backbone which requires 28 hours. Thus, fine-tuning the model for the rest of the hyperparameters with the DGCNN backbone when the dimensionality is 512 is infeasible within the duration of this thesis as is possible for the model with PointNet autoencoder backbone, the results of which are recorded in Ch. 5.2. An overview of the training of the two models with different backbone but same dimensionality of the latent space representations is shown in Fig. 5.4.1. It can be observed that the ultimate validation loss for both the models are extremely comparable where the model with PointNet autoencoder backbone converges faster than the one with DGCNN backbone. Thus, it seemed to be a rather practical choice to fine-tune the former model for the ultimate goal of this thesis.

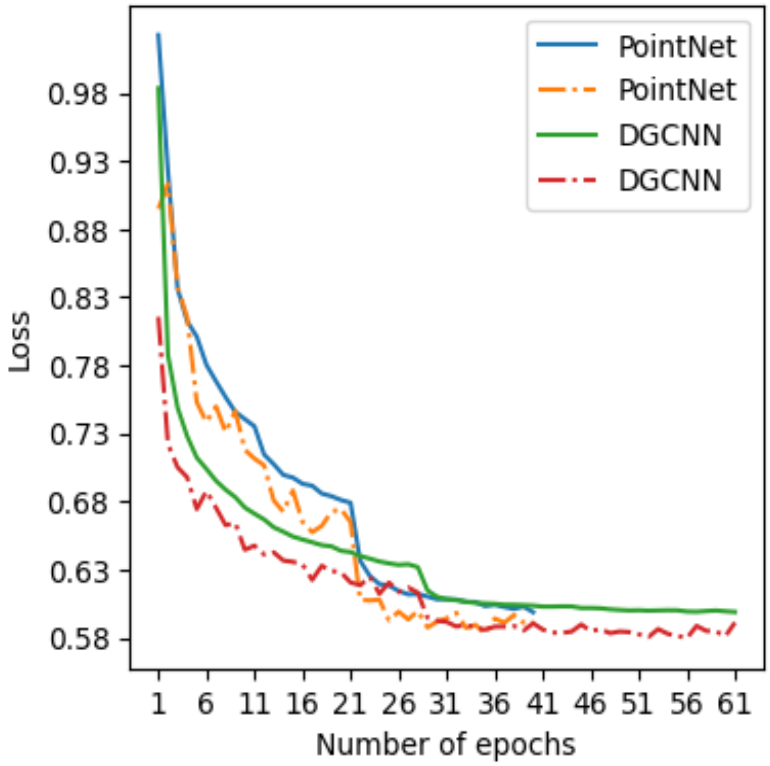

Figure 5.4.1: Overview of the training and validation loss of the models trained with different backbones - PointNet autoencoder and DGCNN when the dimension of the latent space representation is set to 512. The y-axis represents the loss. The x-axis represents the number of epochs. The solid lines represent the mean training loss after each epoch. The dash dot line represents the mean validation loss after each epoch. The legend represent the backbone of the model.

Calculating the Similarity of the Test Dataset with the Representative Objects

Once the representative objects are obtained, it is necessary to compare how similar they are compared to the objects in the test dataset. Hausdorff distance is commonly used to calculate the similarity between two point clouds.

Dataset Size(Whole)	No. of representative objects	Dataset Size(Test)	Clustering Algorithm	Average Hausdorff Distance
949647	94	49981	multi-level HDBSCAN	0.170
949647	175	49981	k-medoids	0.134
949647	175	49981	Spectral	0.155
949647	175	49981	Agglomerative	0.138

Table 62: Similarity metric for the ConClu approach with PointNet Autoencoder backbone.

Table- 62 shows the similarity metric of the ConClu approach with the PointNet Autoencoder as backbone.

It can be seen that the k-medoids has the minimum Hausdorff distance which is quite comparable to agglomerative clustering followed by spectral clustering and eventually multi-level HDBSCAN algorithm. This can be due to the fact that the traditional spectral clustering algorithm and agglomerative clustering doesn't give medoids of the cluster but cluster centres. But to ensure that the cluster centres are actual members of the dataset, Hausdorff distance is used to calculate the medoid of the clusters from the respective cluster members. Because of this usage of Hausdorff distance in calculating the medoid, it is possible that when Hausdorff distance is used to calculate the ultimate similarity of the test dataset and the representative members, it slightly favoured the clustering algorithm which already used it for calculating the cluster medoid. Also to make it consistent with spectral clustering and agglomerative clustering, Hausdorff distance is also used as the pairwise distance metric during the k-medoids. Overall, it is to be noted further that even if the average distance for multi-level HDBSCAN is a little more than the rest of the algorithm, the variation isn't quite drastic. Tab. 63 shows the similarity metric of the ConClu approach with the DGCNN as backbone. It is observed that the results for both the backbones are very similar. Hence, it is not conclusive enough to use just this similarity metric to choose a suitable network for the purpose.

Dataset Size(Whole)	No. of representative objects	Dataset Size(Test)	Clustering Algo	Average Hausdorff Distance
949647	94	49981	multi-level HDBSCAN	0.170
949647	175	49981	k-medoids	0.133
949647	175	49981	Spectral	0.152
949647	175	49981	Agglomerative	0.137

Table 63: Similarity metric for the ConClu approach with DGCNN backbone.

5.5 Results

Observing the results of the experimentations it is evident that the HDBSCAN algorithm has significantly better performance as compared to any other clustering algorithm. But it had a major drawback of not having any control on the ultimate number of clusters obtained in the end. To circumnavigate this issue, the idea of multi-level HDBSCAN is proposed. This naturally degrades the DBCV score to some extent because the least-dense region within a cluster is no longer more dense as compared to the region in-between clusters but it has 2 important benefits. It had a control on the ultimate number of representative objects of the dataset which is very crucial in the present use-case. Also often times, it is observed that even though the result of this algorithm is not up to the mark as of HDBSCAN algorithm, it is still better or at least comparable to the next best algorithm, i.e. the spectral clustering algorithm. Furthermore, another significant benefit of multi-level HDBSCAN algorithm as compared to spectral clustering is that the latter takes significantly higher time to cluster such a large dataset as compared to the former. Even in evaluations recorded in Ch. 5 it can be seen that the time taken by the spectral clustering algorithm is quite high as compared to any other algorithm. The multi-level HDBSCAN algorithm takes 0.8 seconds to cluster 1000 datapoints, whereas spectral clustering takes 7.2 seconds for the same task. This phenomenon is even intensified when the number of samples increase from 1000 datapoints to 1 million datapoints. Spectral clustering takes roughly 20 days to form 175 clusters from the entire dataset, whereas the multi-level HDBSCAN takes as less as 3 hours. This is because spectral clustering has a time complexity of $O(N^3)$ for a dataset with N objects. Also, it is observed that the multi-level HDBSCAN algorithm had better performance in most cases as compared to spectral clustering. Thus, it is inefficient to use such large clustering time for worse performance. The ultimate goal of this thesis is to identify "good" source data that represents the variance of the entire dataset.

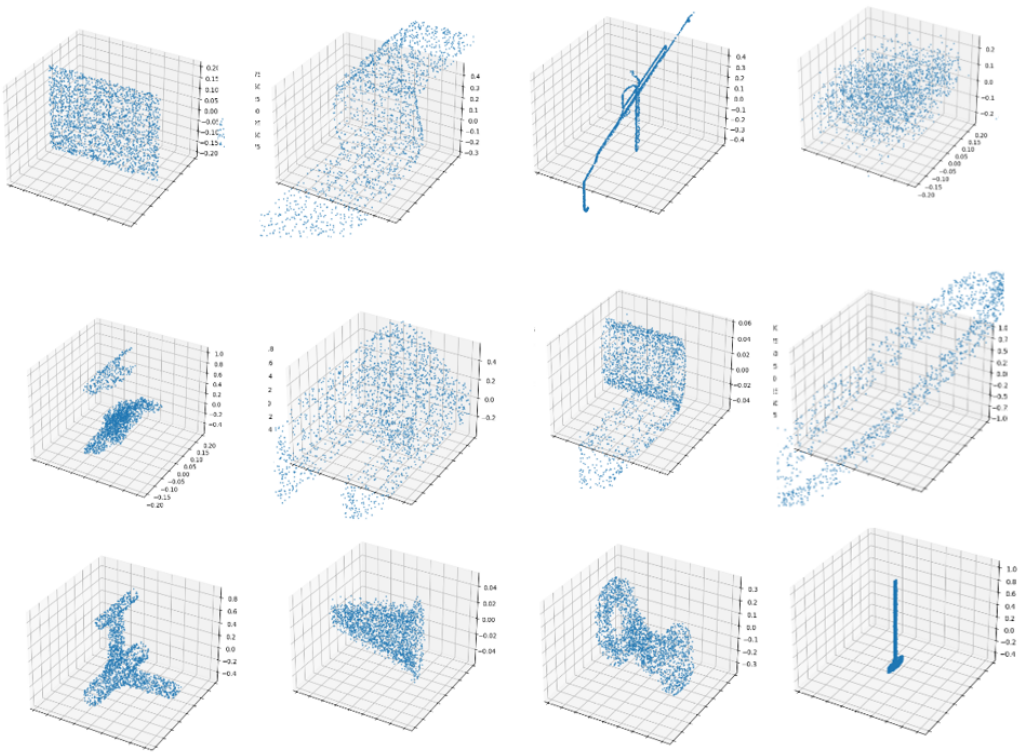

Figure 5.5.1: Some of the representative members of the ABC dataset[47] obtained on applying multi-level HDBSCAN algorithm on the feature representations obtained from the ConClu approach with PointNet Autoencoder backbone.

It is observed that the objects obtained after the applying the clustering algorithms are quite diverse in nature. Although there is no known objective way to determine what qualifies as a "good" source object in case of unsupervised learning, the diversity in the objects obtained in the end quite ensures the fulfillment of the task in hand. Fig. 5.5.1 shows some of the cluster representatives obtained after applying the multi-level HDBSCAN algorithm. Only some objects were chosen from the entire set of representative objects for the sake of visualization. It can be seen that the representative objects of the dataset are quite diverse in nature. Ranging from an ellipse shaped ring, an aircraft to a dumbbell, the objects are quite different from one another. Thus, weighing the pros and cons of the two different backbones and the 5 different clustering algorithm, it can be concluded that the ConClu approach with the PointNet Autoencoder gives the best model for the generation of the latent space representations of te objects in the dataset. Furthermore, weighing the performance of the clustering algorithm under different setup and the overall time required by the clustering algorithms, the multi-level HDBSCAN algorithm seems to be a practical choice for this scenario. The HDBSCAN algorithm has better performance compared to the multi-level HDBSCAN algorithm because more number of clusters would naturally mean more smaller but denser clusters. But since it has no control on the number of clusters it returns and mostly returns a large number of clusters in the presence of a large dataset. Having such a high number of cluster representatives isn't feasible for further tasks in the current setup at IPA. This is why the maximum number of cluster representatives is set to 175. Under this particular constraint, the multi-level HDBSCAN algorithm is the best choice for clustering algorithm in this use-case.

6 Summary

6.1 Conclusion

The primary goal of this thesis was to create a source library to be used for transfer learning approaches in bin picking applications at IPA. This is achieved at the end of the thesis work. A chosen number of objects collectively represent the entire ABC dataset [47] consisting of one million 3D point clouds. It can be qualitatively justified that the representative members of the dataset are quite diverse in nature.

A deep learning algorithm is implemented and evaluated which is capable of processing any dataset containing points clouds of the objects. The algorithm is capable of generating latent space representations of the objects with any number of dimensions as deemed fit according to the user. The two backbones used in this network are the PointNet Autoencoder and the DGCNN. This thesis also evaluates the performance of different clustering algorithms for the purpose of generating representative members of the dataset in hand which constitute as a diverse collection of source data. The clustering algorithms evaluated in this procedure are k-medoids, HDBSCAN, spectral clustering and agglomerative clustering. A new concept called multi-level HDBSCAN algorithm is also proposed in this thesis keeping the limitations in this use-case in mind which is illustrated in the next section. Thus, this thesis is able to develop a framework of the network and the data which can be used further by the transfer learning approaches.

The best performance is attained by the model with PointNet Autoencoder as backbone. Several hyperparameter tuning is performed to improve the performance of the model. Using this model has a plethora of benefits. The performance of the clustering algorithms on the output of this model is better as compared to both the existent network at IPA and also the model with the DGCNN backbone. The training time for this model is significantly lower which facilitated further hyperparameter-tuning resulting in improved results.

6.2 Limitations

There were several challenges encountered during the course of the thesis work. Each sample had 2048 points in the point cloud of the objects. This is essential for learning the features of the objects to eventually come up with accurate feature representation. But this also makes it a dataset with high dimensionality. To tackle such large dataset with individual sample having high dimensions, the concept of dimensionality reduction had to be incorporated. This resulted in additional time in the pipeline and also loss of information about the data to some extent. But this step is extremely necessary to achieve the final goal of the thesis. There is loss of information about the objects due to dimensionality reduction. But there is no way to quantize how much of the information is actually lost. The hardware available served as a challenge when dealing with large datasets like the ABC dataset [47]. There are systems available at IPA with GPUs of higher capacity. But since there are multiple ongoing projects, they were not available for use during this thesis.

Another very obvious problem already existing in these type of tasks is evaluating the performance of the clustering algorithms in unsupervised learning paradigm. Though the availability of the DBCV index helped this situation to some extent, it incorporated challenges of its own. Calculating the DBCV score depended heavily on the number of features of the datapoints. In the presence of a large number of features, it is impossible to work with such high dimensional data in the available hardware setup mentioned in Ch. 4.5.2. This is a primary reason why the concept of dimensionality reduction had to be used. Because of how the DBCV algorithm works, evaluating the performance of the clustering algorithms while considering all the datapoints together is unfeasible. So the performance of the clustering algorithm had to be evaluated for a smaller subset of the dataset. This made it crucial to verify if the performance followed a similar trend even for higher number of datapoints and the results obtained while working with a smaller subset of the dataset is enough to derive logical conclusions.

Furthermore, a unifying evaluation metric that gives the same preference to globular and non-globular clusters is not available. The prevalent metrics tend to have a bias towards either of the two. In the presence

of high dimensional dataset such as the ABC dataset [47], it is not possible to visualize the entire dataset in high dimensional space to decide whether the clusters are convex or non-convex in nature. Thus, the decision of which metric to use cannot be relied on the visualization of the representations of the actual objects. Therefore, it is completely based on the theoretical aspect and the expected outcomes of the same.

As documented in Ch. 5.3, it is seen that the training time of the ConClu approach with the DGCNN backbone is significantly high in the current hardware setup as mentioned in 4.5.2. In the absence of better hardware resources it is infeasible to conduct further studies in this limited time frame with this backbone in the search of better results.

6.3 Future Scope

The source library and the framework suggested in this thesis can be used for further transfer learning tasks down the pipeline. The performance of those approaches with the existing library at IPA can be compared to that of this newly created source library. Finding a better clustering evaluation metric while keeping the current limitations is an open point for further research.

Furthermore, it is seen that the performance of spectral clustering algorithm is often comparable to that of multi-level HDBSCAN algorithm. But it required an impractical amount of time for the spectral clustering algorithm to converge. The hyperparameters in spectral clustering could be fine-tuned to evaluate if it ameliorates the convergence time. Moreover, in the presence of better hardware resources like GPU clusters it could be possible to perform spectral clustering algorithm on such large dataset in a practical duration of time.

Also, it is noticed that the network with DGCNN backbone is quite computationally expensive. Better hardware setup might facilitate fine-tuning of its hyperparameters to further improve the results. Having better hardware setup would also mean more number of samples could be evaluated at once during calculating the DBCV score. This could further help in improving the quality of the results. The values of certain parameters are set according to [59], specially in the Sinkhorn-Knopp algorithm module. Better domain knowledge could possibly improve the performance, if the hyperparameters are fine-tuned further. Therefore, if some of the limitations can be circumnavigated, the performance can be improved even further.

List of Acronyms

AI Artificial Intelligence

CAD Computer-Aided Design

CNN Convolutional Neural Network

CPU Central Processing Unit

DBCV Density Based Clustering Validation

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DGCNN Dynamic Graph Convolutional Neural Network

DSC Density Sparseness of a Cluster

DSPC Density Separation of a Pair of Clusters

EM Expectation Maximization

FCN Fully Connected Network

GAN Generative Adversarial Network

GB Giga Bytes

GPU Graphics Processing Unit

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise

IPA Fraunhofer Institute for Manufacturing Engineering and Automation

KD K-Dimensional

KL Kullback–Leibler divergence

k-NN k-Nearest Neighbor

LDA Linear Discriminant Analysis

LiDAR Light Detection and Ranging

LIN linear

ML Machine Learning

MLP Multi Layer Perceptron

MRD Mutual Reachability Distance

MST Minimum Spanning Trees

NDA Normal Discriminant Analysis

DFA Discriminant Function Analysis

PTP point-to-point

PCA Principal Component Analysis

ReLU Rectified Linear Unit

RAM Random Access Memory

RGB Red Green Blue

RL Reinforcement Learning

RNN Recurrent Neural Network

stop-grad stop-gradient

SVM Support Vector Machines

TCP Tool Center Point

t-SNE t-distributed Stochastic Neighbor Embedding

VAE Variational Autoencoders

List of Figures

List of Tables

List of Symbols

Symbol	Unit	Description
$a_{pts}coredist$	-	The inverse of the density of each object with respect to all other objects inside its cluster
B_k	-	In-between cluster dispersion matrix
Cov	-	Covariance
C_S	-	Set of critical points
$d_H(X,Y)$	-	Hausdorff distance between two points X and Y
d_{mreach}	-	Mutual reachability distance
D_w	-	Euclidean distance between two datapoints
$\mathcal{D}_\mathcal{S}$	-	Source domain
$\mathcal{D}_\mathcal{T}$	-	Target domain
e'_{ijm}	-	EdgeConv operator
F	-	Point Wise feature matrix of point cloud P
f	-	Symmetric function
f_{p_i}	-	Point wise feature vector
f_φ	-	Encoder backbone with parameters φ
g	-	Gaussian distribution kernel
G_{PO}	-	Pose of an object with respect to the gripper
H	-	Average cross entropy loss
$G(V,E)$	-	Graph G with vertices V and edges E
$hausdorff(P_1,P_2)$	-	Hausdorff distance between two point clouds P_1 and P_2
h_θ	-	Non-linear function
KNN	-	Distance to the k-th nearest neighbor
λ	-	Regularization parameter
$\mathcal{L}_{con}$	-	Contrastive loss
L_{global}	-	GLobal symmetric loss
L_{orth}	-	Orthogonal Loss
L_{reg}	-	Loss for regularization
$\mathcal{L}_{tri}$	-	Triplet loss
M_{ij}	-	M_{ij} is the distance between the centroids of the clusters i and j
N	-	Number of datapoints
P	-	Point cloud

Continued to next page

Symbol	Unit	Description
P^a, P^b	-	Two augmented views of point cloud P
p_i	-	Each point in a point cloud
p_{ij}	-	Joint conditional probability
$p_{j\|i}$	-	Probability that the point x_i has x_j as its neighbor
$P(\mathcal{X})$	-	Marginal probability distribution of $\mathcal{X}$
q	-	Projection head
q_{ij}	-	Joint probability distribution
R	-	Measure for cluster similarity in David-Bouldin Index
$\mathbb{R}^K$	-	K-dimensional real number
$\square$	-	Symmetric aggregation function
$\mathbf{S}$	-	Class probability matrix
S_c	-	Silhouette Coefficient
S_{CH}	-	Calinski-Harabasz Index
S_i	-	Dispersion measure of cluster i
$\mathcal{T}_\mathcal{S}$	-	Source task
$\mathcal{T}_\mathcal{T}$	-	Target task
var	-	Variance
V_C	-	Validity index of cluster C
W_k	-	Within-between cluster dispersion matrix
W_{PG}	-	Goal pose of the end-effector with respect to the world coordinate system
W_{PO}	-	Pose of an object with respect to some world coordinate system
$\mathcal{X}$	-	Feature space
$\mathcal{Y}$	-	Label space

Bibliography

[1] *A Comprehensive Guide to Siamese Neural Networks*. https://medium.com/@rinkinag24/a-comprehensive-guide-to-siamese-neural-networks-3358658c0513.

[2] Hervé Abdi and Lynne J Williams. "Principal component analysis". In: *Wiley interdisciplinary reviews: computational statistics* 2.4 (2010), pp. 433–459.

[3] Preeti Arora, Shipra Varshney, et al. "Analysis of k-means and k-medoids algorithm for big data". In: *Procedia Computer Science* 78 (2016), pp. 507–512.

[4] Matan Atzmon, Haggai Maron, and Yaron Lipman. "Point convolutional neural networks by extension operators". In: *arXiv preprint arXiv:1803.10091* (2018).

[5] *Autoencoder*. https://en.wikipedia.org/wiki/Autoencoder.

[6] Vassileios Balntas et al. "Learning local feature descriptors with triplets and shallow convolutional neural networks." In: *Bmvc*. Vol. 1. 2. 2016, p. 3.

[7] R Bellman. "Adaptive control processes: a guided tour. Princeton University Press, 1961". In: *PUBLICATIONS 46 Signatureofthecandidate withReg* ().

[8] *Bin Picking in the Industry*. https://atriainnovation.com/en/blog/bin-picking-in-the-industry/.

[9] Konstantinos Bousmalis et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[10] Jane Bromley et al. "Signature verification using a" siamese" time delay neural network". In: *Advances in neural information processing systems* 6 (1993).

[11] Dirk Buchholz. *Bin-picking: new approaches for a classical problem*. Vol. 44. Springer, 2015.

[12] *Calinski-Harabasz Index*. https://en.wikipedia.org/wiki/Calinski-Harabasz_index.

[13] Ricardo JGB Campello, Davoud Moulavi, and Joerg Sander. "A simpler and more accurate AUTO-HDS framework for clustering and visualization of biological data". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9.6 (2012), pp. 1850–1852.

[14] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[15] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15750–15758.

[16] *Convolutional Neural Networks, Explained*. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[17] Marco Cuturi. "Sinkhorn distances: Lightspeed computation of optimal transport". In: *Advances in neural information processing systems* 26 (2013).

[18] Wenyuan Dai et al. "Self-taught clustering". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 200–207.

[19] Hal Daume III and Daniel Marcu. "Domain adaptation for statistical classifiers". In: *Journal of artificial Intelligence research* 26 (2006), pp. 101–126.

[20] *DBSCAN Wikipedia*. https://en.wikipedia.org/wiki/DBSCAN.

[21] J Demmel. *CS 267: Notes for Lecture 23, April 9, 1999. Graph Partitioning, Part 2*. 1999.

[22] Lih-Yuan Deng, Max Garzon, and Nirman Kumar. "What is dimensionality reduction (dr)?" In: *Dimensionality Reduction in Data Science*. Springer, 2022, pp. 67–77.

[23] Dandan Ding et al. "Point cloud upsampling via perturbation learning". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), pp. 4661–4672.

[24] Lixin Duan, Dong Xu, and Ivor Tsang. "Learning with augmented features for heterogeneous domain adaptation". In: *arXiv preprint arXiv:1206.4660* (2012).

[25] Alireza Entezami, Hassan Sarmadi, and Behzad Saeedi Razavi. "An innovative hybrid strategy for structural health monitoring by modal flexibility and clustering methods". In: *Journal of Civil Structural Health Monitoring* 10.5 (2020), pp. 845–859.

[26] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[27] Patrik Fager et al. "Gripper types and components in robotic bin picking". In: *Advances in Production Management Systems. The Path to Digital Transformation and Innovation of Production Management Systems: IFIP WG 5.7 International Conference, APMS 2020, Novi Sad, Serbia, August 30–September 3, 2020, Proceedings, Part I*. Springer. 2020, pp. 267–274.

[28] Jerome H Friedman, Jon Louis Bentley, and Raphael A Finkel. *An algorithm for finding best matches in logarithmic time*. Department of Computer Science, Stanford University, 1975.

[29] *Fundamentals of Pneumatic Grippers for Industrial Applications*. https://www.digikey.co.uk/en/articles/fundamentals-of-pneumatic-grippers-for-industrial-applications.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[31] Jean-Bastien Grill et al. "Bootstrap your own latent-a new approach to self-supervised learning". In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[32] Zhizhong Han et al. "Multi-angle point cloud-VAE: Unsupervised feature learning for 3D point clouds from multiple angles by joint self-reconstruction and half-to-half prediction". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. 2019, pp. 10441–10450.

[33] John A Hartigan. *Clustering algorithms*. John Wiley & Sons, Inc., 1975.

[34] Kaveh Hassani and Mike Haley. "Unsupervised multi-task feature learning on point clouds". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 8160–8171.

[35] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[36] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In defense of the triplet loss for person re-identification". In: *arXiv preprint arXiv:1703.07737* (2017).

[37] *Hierarchical Clustering*. https://harshsharma1091996.medium.com/hierarchical-clustering-996745fe656b.

[38] Asmaul Hosna et al. "Transfer learning: a friendly introduction". In: *Journal of Big Data* 9.1 (2022), p. 102.

[39] *Intuitively Understanding Variational Autoencoders*. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[40] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[41] Eric Jang et al. "Grasp2vec: Learning object representations from self-supervised grasping". In: *arXiv preprint arXiv:1811.06964* (2018).

[42] Xin Jin and Jiawei Han. "K-Means Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 563–564. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_425. URL: https://doi.org/10.1007/978-0-387-30164-8_425.

[43] Xin Jin and Jiawei Han. "k-medoids Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 564–565. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_426. URL: https://doi.org/10.1007/978-0-387-30164-8_426.

[44] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[45] Kilian Kleeberger et al. "Precise Object Placement with Pose Distance Estimations for Different Objects and Grippers". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 4639–4646.

[46] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. “Siamese neural networks for one-shot image recognition”. In: *ICML deep learning workshop*. Vol. 2. 1. Lille. 2015.

[47] Sebastian Koch et al. “ABC: A Big CAD Model Dataset For Geometric Deep Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[48] Brian Kulis, Kate Saenko, and Trevor Darrell. “What you saw is not what you get: Domain adaptation using asymmetric kernel transforms”. In: *CVPR 2011*. IEEE. 2011, pp. 1785–1792.

[49] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[50] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[51] Bastian Leibe. *Lecture notes in Computer Vision*. 2023.

[52] Levi Lelis and Jörg Sander. “Semi-supervised density-based clustering”. In: *2009 Ninth IEEE International Conference on Data Mining*. IEEE. 2009, pp. 842–847.

[53] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[54] Yangyan Li et al. “Pointcnn: Convolution on x-transformed points”. In: *Advances in neural information processing systems* 31 (2018).

[55] Huei-Yung Lin, Shih-Cheng Liang, and Yu-Kai Chen. “Robotic grasping with multi-view image acquisition and model-based pose estimation”. In: *IEEE Sensors Journal* 21.10 (2020), pp. 11870–11878.

[56] Tong Liu et al. “Exploring transfer learning to reduce training overhead of hpc data in machine learning”. In: *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE. 2019, pp. 1–7.

[57] Yongcheng Liu et al. “Relation-shape convolutional neural network for point cloud analysis”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 8895–8904.

[58] Claudia Malzer and Marcus Baum. “A hybrid approach to hierarchical density-based cluster selection”. In: *2020 IEEE international conference on multisensor fusion and integration for intelligent systems (MFI)*. IEEE. 2020, pp. 223–228.

[59] Guofeng Mei et al. “Unsupervised learning on 3d point clouds by clustering and contrasting”. In: *arXiv preprint arXiv:2202.02543* (2022).

[60] Sumit Misra et al. “An autoencoder based model for detecting fraudulent credit card transaction”. In: *Procedia Computer Science* 167 (2020), pp. 254–262.

[61] Davoud Moulavi et al. “Density-based clustering validation”. In: *Proceedings of the 2014 SIAM international conference on data mining*. SIAM. 2014, pp. 839–847.

[62] Fionn Murtagh and Pedro Contreras. “Algorithms for hierarchical clustering: an overview”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), pp. 86–97.

[63] *New Electric Gripper Plus Kit for Universal Robots*. `https://blog.robotiq.com/bid/73065/New-Electric-Gripper-Plus-Kit-for-Universal-Robots`.

[64] Andrew Ng et al. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[65] *NUS engineers bring a soft touch to commercial robotics*. `https://news.nus.edu.sg/nus-engineers-bring-a-soft-touch-to-commercial-robotics/`.

[66] Stephen M Omohundro. “Five balltree construction algorithms”. In: (1989).

[67] William R Palmer and Tian Zheng. “Spectral clustering for directed networks”. In: *Complex Networks & Their Applications IX: Volume 1, Proceedings of the Ninth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2020*. Springer. 2021, pp. 87–99.

[68] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[69] Weike Pan et al. "Transfer learning to predict missing ratings via heterogeneous user feedbacks". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain*. 2011, p. 2318.

[70] Wookeun Park, Seongmin Seo, and Joonbum Bae. "A hybrid gripper with soft material and rigid structures". In: *IEEE Robotics and Automation Letters* 4.1 (2018), pp. 65–72.

[71] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[72] Gabriel Peyré, Marco Cuturi, et al. "Computational optimal transport: With applications to data science". In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607.

[73] *Piab broadens scope of Kenos KVG vacuum grippers*. https://www.therobotreport.com/piab-kenos-kvg-vacuum-grippers/.

[74] Omid Poursaeed et al. "Self-supervised learning of point clouds via orientation estimation". In: *2020 International Conference on 3D Vision (3DV)*. IEEE. 2020, pp. 1018–1028.

[75] *Principal Component Analysis (PCA) Transformation*. https://www.biorender.com/template/principal-component-analysis-pca-transformation.

[76] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[77] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems* 30 (2017).

[78] Rajat Raina et al. "Self-taught learning: transfer learning from unlabeled data". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 759–766.

[79] Yongming Rao, Jiwen Lu, and Jie Zhou. "Global-local bidirectional reasoning for unsupervised representation learning of 3d point clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5376–5385.

[80] Salah Rifai et al. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th international conference on international conference on machine learning*. 2011, pp. 833–840.

[81] Aditya Sanghi. "Info3d: Representation learning on 3d objects using mutual information maximization and contrastive learning". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer. 2020, pp. 626–642.

[82] Parinya Sanguansat. *Principal component analysis*. BoD–Books on Demand, 2012.

[83] Muhammad Sarmad, Hyunjoo Jenny Lee, and Young Min Kim. "Rl-gan-net: A reinforcement learning agent controlled gan network for real-time point cloud shape completion". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5898–5907.

[84] Daniel Schiller. *Transfer Learning Approaches for Optimising Bin Picking Applications*. 2023.

[85] *scikit-learn-extra API KMedoids*. https://scikit-learn-extra.readthedocs.io/en/stable/generated/sklearn_extra.cluster.KMedoids.html.

[86] *Search and Insertion in K Dimensional tree*. https://www.geeksforgeeks.org/search-and-insertion-in-k-dimensional-tree/.

[87] *SGM-SV MAGNETIC GRIPPER*. https://www.universal-robots.com/fi/plus/products/j-schmalz/sgm-sv-magnetic-gripper/.

[88] Yi Shi et al. "Unsupervised deep shape descriptor with point distribution learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9353–9362.

[89] Hidetoshi Shimodaira. "Improving predictive inference under covariate shift by weighting the log-likelihood function". In: *Journal of statistical planning and inference* 90.2 (2000), pp. 227–244.

[90] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[91] Fei Song et al. “Layer-wise geometry aggregation framework for lossless lidar point cloud compression”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), pp. 4603–4616.

[92] Felix Spenrath. “Heuristic search method for efficient planning for gripping disorderly stored workpieces with industrial robots”. In: (2022).

[93] Tiecheng Sun et al. “Quadratic terms based point-to-surface 3D representation for deep learning of point cloud”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.5 (2021), pp. 2705–2718.

[94] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[95] *T-SNE Explained — Math and Intuition*. https://medium.com/swlh/t-sne-explained-math-and-intuition-94599ab164cf.

[96] *Triplet Loss and Siamese Neural Networks*. https://medium.com/@enoshshr/triplet-loss-and-siamese-neural-networks-5d363fdeba9b.

[97] *TSNE*. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html.

[98] *Understanding Variational Autoencoders(VAEs)*. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[99] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[100] Alberto A Vergani and Elisabetta Binaghi. “A soft davies-bouldin separation measure”. In: *2018 IEEE international conference on fuzzy systems (FUZZ-IEEE)*. IEEE. 2018, pp. 1–8.

[101] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[102] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. “Order matters: Sequence to sequence for sets”. In: *arXiv preprint arXiv:1511.06391* (2015).

[103] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17 (2007), pp. 395–416.

[104] Yue Wang et al. “Dynamic graph cnn for learning on point clouds”. In: *ACM Transactions on Graphics (tog)* 38.5 (2019), pp. 1–12.

[105] Zheng Wang, Yangqiu Song, and Changshui Zhang. “Transferred dimensionality reduction”. In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II 19*. Springer. 2008, pp. 550–565.

[106] Kilian Q Weinberger and Lawrence K Saul. “Distance metric learning for large margin nearest neighbor classification.” In: *Journal of machine learning research* 10.2 (2009).

[107] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[108] *What is dimensionality reduction?* https://www.ibm.com/topics/dimensionality-reduction.

[109] *What Is Transfer Learning? Exploring the Popular Deep Learning Approach.* https://builtin.com/data-science/transfer-learning.

[110] W Wu, Z Qi, and F Pointconv Li. “Deep convolutional networks on 3d point clouds. IEEE”. In: *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 9613–9622.

[111] Petros Xanthopoulos et al. “Linear discriminant analysis”. In: *Robust data mining* (2013), pp. 27–33.

[112] Mutian Xu et al. “Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3173–3182.

[113] Yaoqing Yang et al. “Foldingnet: Point cloud auto-encoder via deep grid deformation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 206–215.

[114] Muhamad Yani, S Irawan, and Casi Setianingsih. “Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail”. In: *Journal of Physics: Conference Series* 1201 (May 2019), p. 012052. DOI: 10.1088/1742-6596/1201/1/012052.

[115] Bianca Zadrozny. “Learning and evaluating classifiers under sample selection bias”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 114.

[116] Manzil Zaheer et al. “Deep sets”. In: *Advances in neural information processing systems* 30 (2017).

[117] Lichen Zhao et al. “Transformer3D-Det: Improving 3D object detection by vote refinement”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), pp. 4735–4746.

[118] Haoran Zhou et al. “Adaptive graph convolution for point cloud analysis”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 4965–4974.

[119] Yin Zhu et al. “Heterogeneous transfer learning for image classification”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. 1. 2011, pp. 1304–1309.

[120] Zhuotun Zhu et al. “Deep learning representation using autoencoder for 3D shape retrieval”. In: *Neurocomputing* 204 (2016), pp. 41–50.