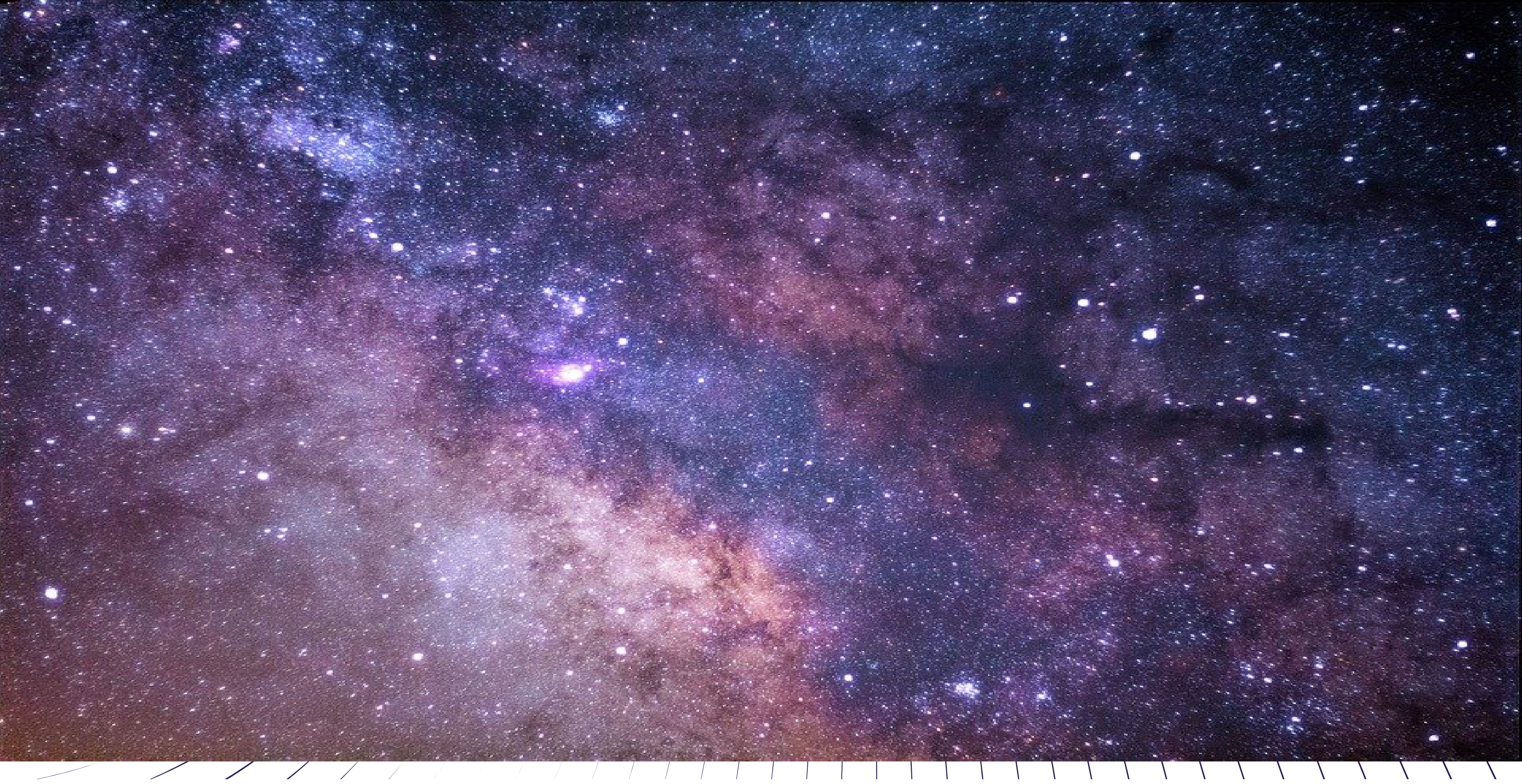

Hyperbolic Generalized Category Discovery

Hyperbolic visual learning and clustering for generalized category discovery

Mohamad Dalal

AI, Vison and Sound, AVS10-1045, 2025

Master's Thesis

AALBORG UNIVERSITY
STUDENT REPORT

Electronic Systems
Aalborg University
http://es.aau.dk

Title:
Hyperbolic Generalized Category Discovery

Theme:
Artificial Intelligence

Project Period:
Fall Semester 2025

Project Group:
1045

Participant(s):
Mohamad Dalal

Supervisor(s):
Thomas B. Moeslund
Joakim Bruslund Haurum

Copies: 1

Page Numbers: 56

Date of Completion:
June 4, 2025

Abstract:

This thesis investigates the use of hyperbolic visual learning in Generalized Category Discovery (GCD), an open-world classification problem in which a model is tasked with classifying both known and novel classes.
To that end, two GCD methods are adapted to learn embeddings in the Lorentz model of hyperbolic geometry. Furthermore, the K-Means algorithm is modified to perform non-parametric clustering on hyperbolic embeddings.
The experiments showcase that the non-parametric method benefits from learning in Lorentz space, while the parametric method does not, with its best results in Euclidean geometry.
Furthermore, experiments with the hyperbolic K-Means demonstrate increased accuracy when clustering embeddings directly in Lorentz space. However, K-Means in the Poincaré model of hyperbolic geometry suffers from instability, failing to generate valid clusters depending on the initialization of the cluster prototypes.

Contents

Preface

This report was written as part of a thesis project for the computer engineering with specialization in AI, Vision and Sound (AVS) masters degree at Aalborg University (AAU).

On the usage of AI: Throughout this thesis the only generative AI tool used was Copilot, which was used for only two purposes. It was mainly used for auto-completion within Visual Studio Code, and any code generated by Copilot was thoroughly checked and tested to insure that it performs as expected and is mathematically correct. The other purpose was for generating figures using matplotlib. The code was generated using prompts such as "Generate a 3D plot of a hyperboloid with equation $x^2 + y^2 - z^2 = -1$" or selecting a piece of code and prompting "Draw arrows between the points in xy and xy2". The generated code is later adjusted and modified personally to generate the final figures.

Lastly I would like to thank my supervisors Thomas B. Moeslund and Joakim Bruslund Haurum for offering their constant guidance and supervision throughout this project. I would also like to thank Alex Pujol Vidal from the Visual Analysis of Perception lab at AAU for sparring and sharing ideas when it came to hyperbolic geometry.

Aalborg University, June 4, 2025

[illegible]

Mohamad Dalal
<mdalal20@student.aau.dk>

Acronyms

BSSK Balanced Semi-Supervised K-Means. 3, 7, 49, 50

CUB Caltech-UCSD Birds. 12, 39, 44, 45, 48

FC Fully Connected. 3, 4, 21, 23–25, 29, 36, 42, 43, 45, 49, 51

GCD Generalized Category Discovery. 3–7, 9, 11, 12, 14, 22, 24, 25, 27–29, 39–52

GELU Gaussian Error Linear Unit. 40

HSSK Hierarchical Semi-Supervised K-Means. 3, 7, 27, 50

MLP Multi-Layer Perceptron. 6, 10, 27, 29

SGD Stochastic Gradient Descent. 41, 43, 48, 49, 52

SOTA State of The Art. 3, 25, 27, 50

SPT Spatial Prompt Tuning. 11

SSB Semantic Shift Benchmark. 13, 39, 44, 45, 50

ViT Vision Transformer. 6, 10, 11, 25, 27, 29, 40, 41, 47

VPT Visual Prompt Tuning. 11

Chapter 1

Introduction

Image classification in the real world is a tricky problem. Traditionally, a classification model is trained with fully labeled datasets, seeing only a subset of all classes that it can encounter in the wild, which is categorized as closed-world learning. While semi-supervised learning approaches also incorporate unlabeled data, they still assume that the unlabeled data belongs to the same set of classes as the labeled data. These scenarios struggle in the wild when a new sample from an unseen class is encountered, also known as open-world learning, in which case the model would classify the new sample into one of the seen classes.

Some approaches try to train the model to recognize samples from unseen classes and categorize them as "unknown"[8, 28]. However, this ignores the potential of learning from the new objects. The Generalized Category Discovery (GCD) task aims to tackle this problem by requiring the model to be able to correctly classify already seen classes, while also being able to cluster new samples into unseen classes[31]. Another advantage to the GCD setup is that it no longer assumes that unlabeled data comes from the same set of classes as the labeled data, extending the amount of data that can be used in self-supervised learning.

The clustering of seen and unseen classes in GCD can be split into two main categories. Non-parametric methods[26, 32] utilize a non-parametric algorithm, such as K-Means, to cluster the samples, while parametric methods[30, 34, 35, 37] utilize parametric Fully Connected (FC) layers to cluster the samples.

The State of The Art (SOTA) GCD method by Rastegar et al. [26] is a non-parametric method that learns data embeddings through contrastive learning using hierarchical pseudo-labels. They use Balanced Semi-Supervised K-Means (BSSK) and define the Hierarchical Semi-Supervised K-Means (HSSK) algorithms to generate clusters of different levels of hierarchies. Rastegar et al. [26]'s method showcases the benefits of utilizing the latent hierarchies present in the datasets to solve the GCD task.

Concurrently, work in the field of hyperbolic machine learning has shown that learning embeddings in hyperbolic geometry leads to increased performance in hierarchical tasks[12, 24], as hyperbolic geometry is more suited for representing hierarchies than traditional Euclidean geometry[10, 17, 23]. With works showcasing success by learning embeddings in the Poincaré and Lorentz models of hyperbolic geometry[5, 12, 24].

The potential of hyperbolic geometry for the GCD task was concurrently explored in a recently published paper by Liu et al. [20]. They adapt three GCD models, two non-parametric and one parametric, to learn embeddings in the Poincaré model of hyperbolic geometry. For the parametric method, they utilize the hyperbolic FC layer developed by Ganea et al. [6] to cluster the data, while for non-parametric methods, they perform K-Means on the Euclidean embeddings.

Liu et al. [20] leave the potential of using hyperbolic embeddings directly for non-parametric clustering uninvestigated. Furthermore, it is worth investigating another model for hyperbolic space, as the Lorentz model is known to be more numerically stable, and has achieved successes in large data representation learning as shown by Desai et al. [5]. This lays the foundation for this thesis, in which the effects of using Lorentz hyperbolic learning and hyperbolic clustering algorithms for GCD are explored.

Chapter 2 presents a detailed analysis of the existing literature within GCD and hyperbolic visual learning. Chapter 3 fully fleshes out the research questions based on the analyzed literature. Chapter 4 details the novel implementations and modifications used throughout this project. Chapter 5 presents the experiment setups and the results of these experiments, followed by numerous ablation studies. Lastly, Chapter 6 highlights some potential future directions, followed by the conclusion in Chapter 7.

Chapter 2

Technical Review

It is important to understand the existing literature within Generalized Category Discovery (GCD) and hyperbolic visual learning to combine the two fields. This chapter describes the GCD task and some of the methods used to solve it, followed by a review of the hyperbolic visual learning literature, before finally highlighting the work done by Liu et al. [20] within hyperbolic GCD.

2.1 Generalized Category Discovery

Generalized Category Discovery (GCD) is an open world classification problem, where only a subset of a dataset is labeled, and the labeled subset does not necessarily contain all considered classes in the dataset. An example would be training a model on partially labeled datasets, such as animal datasets from camera traps. This results in a labeled subset, which includes labels from a set known classes $\mathcal{Y}_{\mathcal{L}}$, and an unlabeled subset including labels from both known and unknown classes $\mathcal{Y}_{\mathcal{U}}$. The model is then tasked with classifying images from both the known (seen) and novel (unseen) classes.

Therefore, when running GCD experiments, two data subsets are used:

- $\mathcal{D}_{\mathcal{L}} = \{x_i, y_i\} \in \mathcal{X}, \mathcal{Y}_{\mathcal{L}}$ A data subset whose labels are available to the model during training. This dataset only includes known classes $\mathcal{Y}_{\mathcal{L}}$ and is only used for training.
- $\mathcal{D}_{\mathcal{U}} = \{x_i, y_i\} \in \mathcal{X}, \mathcal{Y}_{\mathcal{U}}$ A data subset whose labels are not available to the model during training. This dataset includes both known and novel classes $\mathcal{Y}_{\mathcal{L}} \subset \mathcal{Y}_{\mathcal{U}}$ and is used for training and testing.

The goal is to classify the samples in $\mathcal{D}_{\mathcal{U}}$, regardless of which set of classes the samples belong to. The model is trained using samples from both $\mathcal{D}_{\mathcal{L}}$ and $\mathcal{D}_{\mathcal{U}}$.

However, since only labels from $\mathcal{D}_{\mathcal{L}}$ are available, the training method usually includes both supervised and self-supervised learning.

GCD has typically been approached via either non-parametric or parametric methods. Non-parametric methods cluster input data based on proximity to class prototypes (e.g. K-Means), whereas parametric methods classify based on proximity to classification hyperplanes (e.g., cross-entropy classification).

While the class assignment methods differ, both methods use self-supervised learning to learn the feature embeddings of the samples. The following section will list some of the training techniques used in the GCD literature.

2.1.1 Non-Parametric Methods

Vanilla GCD

Vaze et al. [31] introduces the GCD task and presents a method to solve it based on contrastive learning to learn embeddings and K-Means to cluster the classes and find class prototypes.

When learning the embeddings, they use a Vision Transformer (ViT) pre-trained with DINO[2] and ImageNet as the backbone and a Multi-Layer Perceptron (MLP) projector. They fine-tune the projector and the last layer of the ViT using self-supervised contrastive learning and supervised contrastive learning. The function for the self-supervised contrastive learning is:

$$\mathcal{L}_i^u = -\log\left(\frac{\exp(\mathbf{z}_i \cdot \mathbf{z'}_i/\tau)}{\sum_n \mathbb{1}_{[n\neq i]} \exp(\mathbf{z}_i \cdot \mathbf{z}_n/\tau)}\right) \tag{2.1}$$

where $\mathbf{z}$ and $\mathbf{z}'$ are two augmentations of one image, τ is a temperature constant and $\mathbb{1}_{[n\neq i]}$ is an indicator function that evaluates to 1 if and only if $n \neq i$. The function for the supervised contrastive learning is:

$$\mathcal{L}_i^s = -\frac{1}{\mathcal{N}(i)} \sum_{q\in\mathcal{N}(i)} \log\left(\frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_q/\tau)}{\sum_n \mathbb{1}_{[n\neq i]} \exp(\mathbf{z}_i \cdot \mathbf{z}_n/\tau)}\right) \tag{2.2}$$

where $\mathcal{N}(i)$ is a set of indices of features with the same label as $\mathbf{z}_i$ in the mini-batch B, excluding $\mathbf{z}_i$ itself. These two losses are combined into:

$$\mathcal{L}^t = (1-\lambda)\sum_{i\in B} \mathcal{L}_i^u + \lambda \sum_{i\in B_L} \mathcal{L}_i^s \tag{2.3}$$

Where λ is a weighting hyperparameter, B is the entire mini-batch, and B_L is the labeled part of the mini-batch, as labels are needed for the supervised contrastive loss.

After training is finished, the features from the training set are extracted and used to find class prototypes using a semi-supervised K-Means algorithm, which takes the labels of the labeled part of the training dataset into account. This is done by fixing the labeled features to the centers corresponding to their labels when iteratively finding new prototypes. This algorithm assumes prior knowledge of the number of classes in the dataset. Vaze et al. [31] introduced a method for finding for finding the number of classes by performing binary search for the number of classes that maximizes the clustering accuracy. However, another way to do this is to use the true number of classes.

SelEx

Rastegar et al. [26] introduce the concept of "expertise", i.e., the ability to generalize across abstraction layers (e.g., hierarchies). This is achieved by combining pseudo-labels and contrastive learning into self-expertise losses. The pseudo-labels are created using the Balanced Semi-Supervised K-Means (BSSK) and Hierarchical Semi-Supervised K-Means (HSSK) algorithms, which are then used to calculate the unsupervised and supervised self-expertise losses.

The BSSK algorithm builds upon the semi-supervised K-Means algorithm used in Vanilla GCD by ensuring that the generated clusters have a minimum number of samples. BSSK is first used to generate the cluster centers $\mu^0 = \{\mu_1^0, \mu_2^0, \cdots, \mu_{|\mathcal{Y}_\mathcal{U}|}^0\}$ corresponding to the most granular hierarchy. Thereafter, coarser hierarchies are generated by halving the number of clusters using HSSK. In HSSK, the clusters corresponding to the seen classes $\mathcal{Y}_\mathcal{L}$ are combined, and the remaining unseen clusters are generated using BSSK with half the number of clusters and double the minimum cluster sizes, resulting in $\mu^1 = \{\mu_1^1, \mu_2^1, \cdots, \mu_{|\mathcal{Y}_\mathcal{U}|/2}^1\}$. In the end, a total of $\lceil \log_2 (|\mathcal{Y}_\mathcal{U}|) + 1 \rceil$ hierarchies are generated.

Unsupervised self-expertise builds upon unsupervised contrastive learning by integrating the hierarchical pseudo-labels. This is done by assigning weights to the negative samples depending on how far up the hierarchy a common cluster between the anchor and negative sample exists. This is done by constructing the objective matrix as:

$$\mathbf{Y} = y_{ij} = \sum_{k=0}^{\lceil |\mathcal{Y}_\mathcal{U}| \rceil} \frac{\mathbb{1}(c_i^k \neq c_j^k)}{2^k}, y_{ii} = 1 \tag{2.4}$$

Where c_i^k is the pseudo-label of i at the kth hierarchy. This objective matrix leads to samples closer semantically being treated as harder negatives, with samples with the same pseudo-label being the hardest. Counter-intuitively, this leads to samples

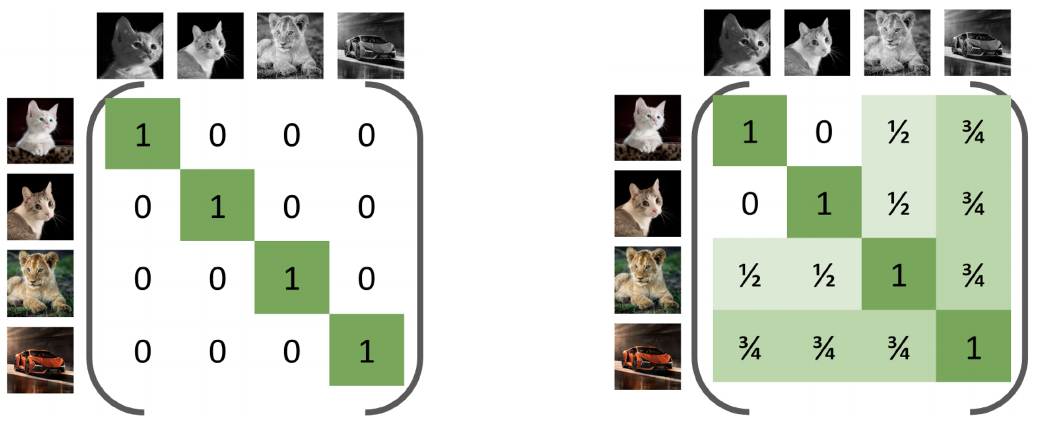

Figure 2.1: The objective matrix in traditional unsupervised contrastive learning (left) compared to unsupervised self-expertise learning (right). In unsupervised contrastive learning, only the augmentation is treated as a positive, while all other samples are hard negatives. In unsupervised self-expertise, the weights of the negative samples are dependent on the hierarchical pseudo-labels, with semantically closer samples being treated as harder negatives[26].

in the same cluster being split apart the most, which Rastegar et al. [26] argue "*can significantly enhance the purity of the cluster*". It is also possible to combine the expertise and contrastive objective matrices:

$$\mathbf{O} = \alpha \mathbf{Y} + (1-\alpha)\mathbf{I} \tag{2.5}$$

Where $\mathbf{I}$ is the identity matrix and α is a weighting hyperparameter. A lower α leads to sharper logits, reducing to the original contrastive loss objective matrix when $\alpha = 0$. The unsupervised self-expertise loss is then computed as:

$$\mathcal{L}^{u}_{SE} = \mathcal{L}_{BCE}(\mathbf{P}, \mathbf{O}) \tag{2.6}$$

Where $\mathbf{P}$ is the logits matrix produced using cosine similarity. Figure 2.1 shows an example of the objective matrix in unsupervised contrastive learning vs unsupervised self-expertise learning.

The hierarchical pseudo-labels are also used in the supervised self-expertise loss:

$$\mathcal{L}^{s}_{SE} = \frac{1}{2} \sum_{k=0}^{\log_2(|\mathcal{Y}_{\mathcal{U}}|)} \frac{\mathcal{L}^{k}_{s}|\frac{D}{2^k}}{2^k} \tag{2.7}$$

Where $\mathcal{L}^{k}_{s}$ is supervised contrastive learning applied to the pseudo-labels at hierarchical level k, and $\mathcal{L}^{k}_{s}|\frac{D}{2^k}$ means that the loss is only applied to the first $\frac{D}{2^k}$ elements out of the embedding vector of dimension D. This leads to higher hierarchies utilizing only a sub-set of the full embedding dimension, as they have a lower number of clusters and a larger number of positive samples. When working with the lowest hierarchy level $k = 0$, only labeled samples $\mathcal{D}_{\mathcal{L}}$ are used alongside ground-truth labels.

When training the SelEx model, the hierarchical pseudo-labels are recomputed every epoch, and the full loss is computed as:

$$\mathcal{L}_{SE} = (1 - \lambda)\mathcal{L}^{u}_{SE} + \lambda\mathcal{L}^{s}_{SE} \tag{2.8}$$

2.1.2 Parametric Methods

SimGCD

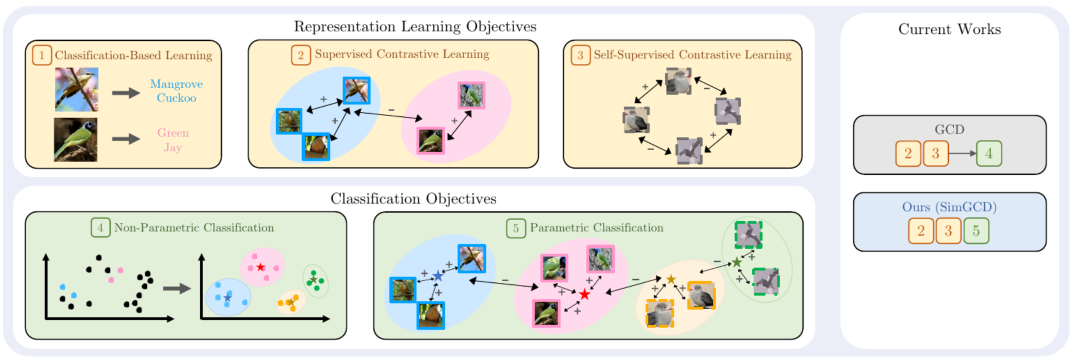
Representation Learning Objectives
1 Classification-Based Learning
Mangrove Cuckoo
Green Jay
2 Supervised Contrastive Learning
3 Self-Supervised Contrastive Learning
Current Works
GCD
2 3 4
Ours (SimGCD)
2 3 5
Classification Objectives
4 Non-Parametric Classification
5 Parametric Classification

Figure 2.2: Figure from the SimGCD paper showing the different building blocks on the left, and how they are utilized in the different models on the right. The arrow $\rightarrow$ signifies a separation of learning between the steps[35]

.

Wen et al. [35] investigate and find that previous parametric methods have a bias towards old labels, where they misclassify many new instances as old classes. They conclude that this is due to the poor quality of the pseudo-labels used in training. Therefore, they utilize self-distillation to generate the pseudo-labels. They also show that joint training with contrastive learning and classification objectives provides improved performance with the improved pseudo-labels. Figure 2.2 shows the difference between the training procedure of Vanilla GCD and SimGCD.

They follow Vanilla GCD's formula for representation learning, where they use supervised contrastive loss $\mathcal{L}^{s}_{rep,i}, i \in B_L$ on labeled data, and self-supervised contrastive loss on all the data $\mathcal{L}^{u}_{rep,i}, i \in B$. Combining the two losses with a weight hyperparameter λ:

$$\mathcal{L}_{rep} = (1 - \lambda)\sum_{i \in B}\mathcal{L}^{u}_{rep,i} + \lambda\sum_{i \in B_L}\mathcal{L}^{s}_{rep,i} \tag{2.9}$$

They generate their pseudo-labels using a student/teacher setup. The student model is a model comprised of a Vision Transformer (ViT) pre-trained with DINO[2] and ImageNet as the backbone and a Multi-Layer Perceptron (MLP) projector to perform the classification. The teacher model is a replica of the student model, but with sharper predictions. If the student model is $f(x)$ then the teacher model is $h(x) = f(x)/\tau$, where τ is a sharpening variable. A smaller value for τ leads to sharper distributions with narrower peaks, making the teacher more confident in its predictions.

For classification training, they utilize a simple cross-entropy loss between student predictions and ground truth labels for the supervised loss:

$$\mathcal{L}^s_{cls,i} = -\sum_{k=1}^{|\mathcal{Y}_\mathcal{U}|} y_{i,k} \log(f(\mathbf{x})_{i,k}) \tag{2.10}$$

$$\mathcal{L}^s_{cls} = \frac{1}{|B_L|} \sum_{i=1}^{|B_L|} \mathcal{L}^s_{cls,i} \tag{2.11}$$

Where $|\mathcal{Y}_\mathcal{U}|$ is the number of classes, both seen and unseen, and the subscript k corresponds to the classification score of class k in the output classification probabilities vector and ground truth vector.

For the unsupervised loss, the cross-entropy between the student's output of an image x and the teacher's output of another view of the same image x' is utilized:

$$\mathcal{L}^u_{cls,i} = -\sum_{k=1}^{|\mathcal{Y}_\mathcal{U}|} h(\mathbf{x}')_{i,k} \log(f(\mathbf{x})_{i,k}) \tag{2.12}$$

The unsupervised loss also utilizes a mean-entropy maximization regularizer:

$$H(\bar{x}) = -\sum_{k=1}^{|\mathcal{Y}_\mathcal{U}|} \bar{x}_k \log(\bar{x}_k) \tag{2.13}$$

$$\bar{\mathbf{x}} = \frac{1}{2|B|} \sum_{i=1}^{|B|} f(\mathbf{x})_i + f(\mathbf{x}')_i \tag{2.14}$$

Where $\bar{x}_k$ is the k'th element of the vector $\bar{\mathbf{x}}$. The full unsupervised clustering loss is then:

$$\mathcal{L}^u_{cls} = \frac{1}{|B|} \sum_{i=1}^{|B|} \mathcal{L}^u_{cls,i} - \varepsilon H(\bar{\mathbf{x}}) \tag{2.15}$$

With ε being a weighting factor for the entropy regulariser. The classification losses are combined as:

$$\mathcal{L}_{cls} = (1-\lambda)\mathcal{L}^{u}_{cls} + \lambda\mathcal{L}^{s}_{cls} \tag{2.16}$$

Using the same weight λ that is used in the representation objective. Lastly the full loss for the learning objective is the sum of the representation and classification losses:

$$\mathcal{L} = \mathcal{L}_{rep} + \mathcal{L}_{cls} \tag{2.17}$$

This way the model performs joint learning on both the representation and classification tasks, unlike in Vanilla GCD, which separates the representation task from the clustering task.

SPTNet

Wang et al. [34] build upon SimGCD by using Visual Prompt Tuning (VPT)[13]. VPT works by introducing learnable parameters $\mathbf{P}_s = \{\mathbf{p}^j; j = 1, \cdots, n\}$ to the input images as they are passed through the model. They introduce Spatial Prompt Tuning (SPT), where the parameters are added to the borders of the individual image patches used by the ViT. Given the image patches $\phi(\mathbf{X}) = \{\mathbf{x}^i; i = 1, \cdots, n\}$, where $\phi(\mathbf{X})$ is the patchify operation, SPT provides $\phi(\mathbf{X}) + \mathbf{P}_s = \{\mathbf{x}^1 + \mathbf{p}^1, \cdots, \mathbf{x}^n + \mathbf{p}^n\}$ as the patch inputs to the ViT backbone.

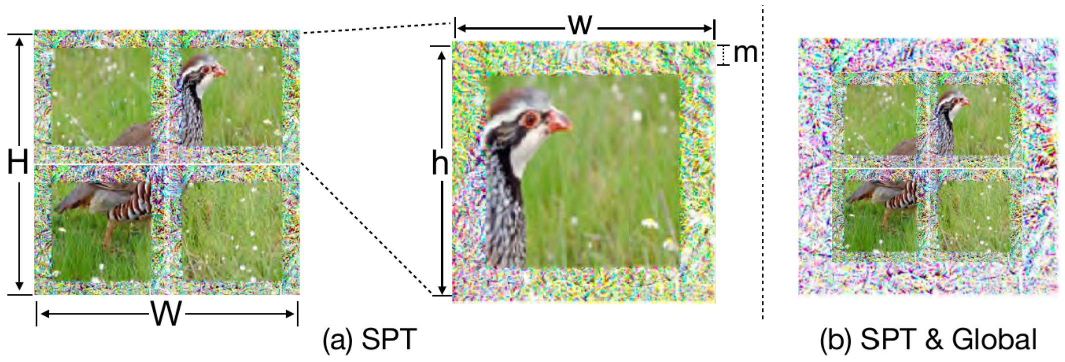

Figure 2.3: Spatial prompts applied to image patches. Compared to global prompts applied to the entire image[34].

SPTNet uses spatial prompts of width $m = 1$ applied to each patch, with global prompts of width $m = 30$ applied to the entire image. Figure 2.3 shows how the prompts are applied to the edge of an image. As they build upon SimGCD, they utilize the same training loss functions, which are a mix of supervised and unsupervised representation and classification learning losses:

$$\mathcal{L} = (1-\lambda)\mathcal{L}^{u}_{rep} + \lambda\mathcal{L}^{s}_{rep} + (1-\lambda)\mathcal{L}^{u}_{cls} + \lambda\mathcal{L}^{s}_{cls} \tag{2.18}$$

Training is done in two alternating stages:

- In the first stage, the model parameters are frozen, while the prompt parameters are trained.
- In the second stage, the prompt parameters are frozen, while the model parameters are trained.

This is due to training being unstable when both model and prompt parameters are trained concurrently. Therefore, they alternate the training stages every 20 iterations. Lastly, classification is performed in the same way as SimGCD, using the output of the projector to classify the input samples into clusters.

2.1.3 Datasets

	CIFAR10	CIFAR100	ImageNet-100	CUB	SCars	Aircraft
$\lvert\mathcal{Y}_{\mathcal{L}}\rvert$	5	80	50	100	98	50
$\lvert\mathcal{Y}_{\mathcal{U}}\rvert$	10	100	100	200	196	100
$\lvert\mathcal{D}_{\mathcal{L}}\rvert$	12.5K	20K	31.9K	1.5K	2.0K	1.7K
$\lvert\mathcal{D}_{\mathcal{U}}\rvert$	37.5K	30K	95.3K	4.5K	6.1K	3.3K

Table 2.1: Number of seen classes $|\mathcal{Y}_{\mathcal{L}}|$ and total classes $|\mathcal{Y}_{\mathcal{U}}|$ and the sample sizes in the labeled $|\mathcal{D}_{\mathcal{L}}|$ and unlabeled $|\mathcal{D}_{\mathcal{U}}|$ data subsets for each of the datasets commonly used for evaluating GCD methods[31, 32].

GCD methods are most commonly evaluated on a combination of generic classification datasets and fine-grained classification datasets. The generic classification datasets are CIFAR10[18], CIFAR100[18], and ImageNet-100[4], which is a subset of the ImageNet dataset with only 100 of its classes sub-sampled. The fine-grained classification datasets are:

- Caltech-UCSD Birds (CUB): A fine-grained dataset featuring 200 different species of birds with 312 binary attributes describing the bird's visual appearance [33].
- Stanford Cars: Featuring 196 classes of different car makes. The fine-grained nature comes from having to classify car type, brand, make, and year[16].

- FGVC-Aircraft: Containing 100 different variants of aircrafts as separate classes. Furthermore, it contains three hierarchy levels being manufacturer, family, and variant. For example, the variant Boeing 737-700 would also have labels 737 as family and Boeing as manufacturer[21].

Figure 2.4 shows example images from the fine-grained datasets, which highlights the difficulty in performing fine-grained classification between semantically similar classes.

The generic classification datasets are split by taking the first classes to be the seen classes $\mathcal{Y}_{\mathcal{L}}$, while the fine-grained datasets follow the splits provided by the Semantic Shift Benchmark (SSB), which uses the hierarchy present in the class labels to split the classes in a semantically coherent manner[31, 32]. The labeled data subset is created by taking 50% of the samples from seen classes, while the unlabeled data subset is generated by the remaining 50% plus all samples from unseen classes $\mathcal{Y}_{\mathcal{U}}/\mathcal{Y}_{\mathcal{L}}$. Table 2.1 shows the numerical statistics for each dataset.

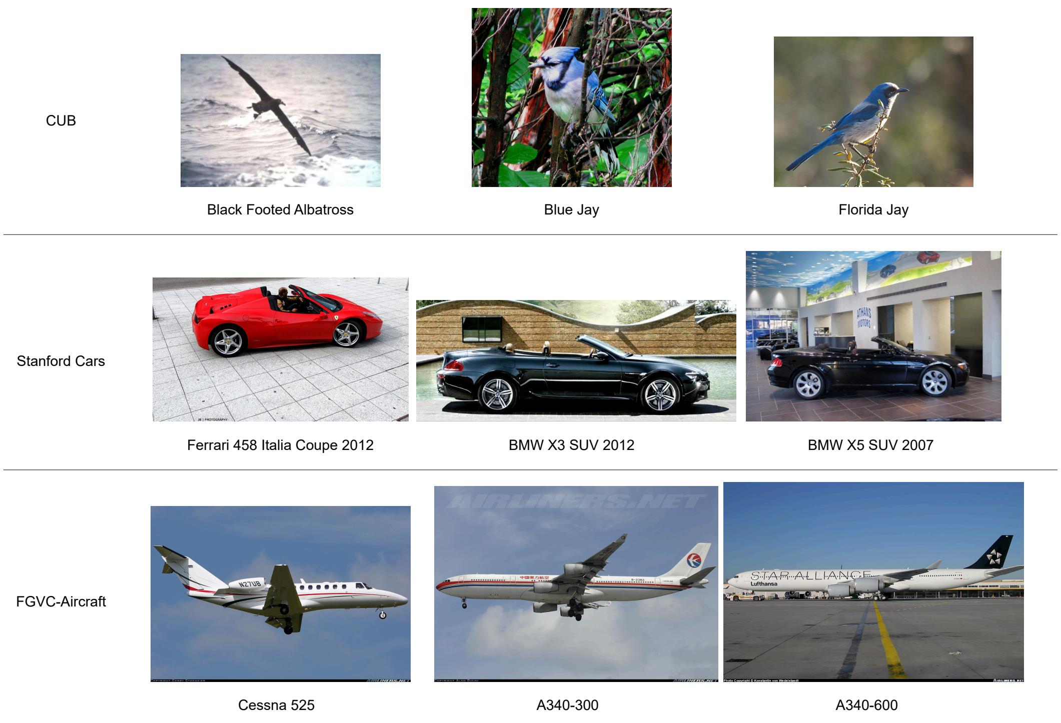

Figure 2.4: Example images from the SSB datasets. Three different classes are shown from each dataset, with the middle and right classes being semantically similar. This is to show the difficulty present in doing classification on fine-grained datasets.

2.1.4 Evaluation of GCD

The performance of a method solving the GCD task is evaluated using the clustering accuracy metric introduced by Vaze et al. [31]. The metric is mathematically formulated as the following:

$$ACC = \max_{p \in \mathcal{P}(\mathcal{Y}_{\mathcal{U}})} \frac{1}{M} \sum_{i=1}^{|\mathcal{D}_{\mathcal{U}}|} \mathbb{1}\{y_i = p(\hat{y}_i)\} \tag{2.19}$$

Where $\mathcal{P}(\mathcal{Y}_{\mathcal{U}})$ is the set of all permutations of clusters in the unlabeled set, and $\hat{y}_i$ is the cluster assigned to the sample i. The maximizing cluster permutation is found using the Hungarian algorithm, assigning labels to the unlabeled clusters that can lead to maximum accuracy. After the permutation is found and the overall accuracy is calculated, two further metrics are reported:

- Old ACC: The accuracy calculated only using data belonging to the seen classes $\mathcal{Y}_{\mathcal{L}}$
- New ACC: The accuracy calculated only using data belonging to the unseen classes $\mathcal{Y}_{\mathcal{U}}/\mathcal{Y}_{\mathcal{L}}$

These two accuracies are used to evaluate the model's ability to cluster data belonging to seen and unseen classes separately, aiding in the analysis of the method.

Pre-training	Model	CUB			Stanford Cars			FGVC-Aircraft			Cifar10			Cifar100		
		All	Old	New	All	Old	New	All	Old	New	All	Old	New	All	Old	New
DINO	Vanilla GCD	51.3	56.6	48.7	39.0	57.6	29.9	45.0	41.1	46.9	91.5	97.9	88.2	73.0	76.2	66.5
	SimGCD (P)	60.3	65.6	57.7	53.8	71.9	45.0	54.2	59.1	51.8	97.1	95.1	98.1	80.1	81.2	77.8
	SPTNet (P)	65.8	68.8	65.1	**59.0**	**79.2**	49.3	**59.3**	61.8	**58.1**	**97.3**	95.0	**98.6**	81.3	84.3	75.6
	SelEx	**73.6**	**75.3**	**72.8**	58.5	75.6	**50.3**	57.1	**64.7**	53.3	95.9	**98.1**	94.8	**82.3**	**85.3**	**76.3**
DINOv2	Vanilla GCD	71.9	71.2	72.3	65.7	67.8	64.7	55.4	47.9	59.2	97.8	99.0	97.1	79.6	84.5	69.9
	SimGCD (P)	71.5	78.1	68.3	71.5	81.9	66.6	63.9	69.9	60.9	**98.7**	96.7	**99.7**	**88.5**	89.2	**87.2**
	SPTNet (P)	76.3	79.5	74.6	-	-	-	-	-	-	-	-	-	-	-	-
	SelEx	**87.4**	**85.1**	**88.5**	**82.2**	**93.7**	**76.7**	**79.8**	**82.3**	**78.6**	98.5	**98.8**	98.5	87.7	**90.8**	81.5

Table 2.2: Performance of the introduced GCD methods on the most common GCD datasets[20]. (P) denotes that a model performs parametric clustering. The highest accuracy under every column for each backbone is highlighted.

Table 2.2 shows the accuracy of the different methods solving GCD. SelEx achieves the best performance on most datasets using both DINO and DINOv2 pre-training. Regardless, SPTNet performs better on three out of the 6 datasets when using DINO, showing that there is no clear advantage to non-parametric methods, with both parametric and non-parametric methods having potential.

2.2 Hyperbolic Machine Learning

Many computer vision problems are hierarchical in nature[7, 12, 23]. Hierarchies can exist in the classes of a dataset, either explicitly where different levels of hierarchies are labeled, or implicitly where classes such as car, plane, boat can all be considered to share the parent class "vehicle" even when not explicitly labeled as such[32]. Hierarchies can also exist between objects and scenes in an image[12] or between different-sized patches[7].

Machine learning is traditionally performed in Euclidean geometry, due to its intuitive grid-like structure and pre-existing concepts and formulas developed for it. However, in Euclidean geometry, the volumes of hyperspheres grow polynomially, which makes it unsuitable for representing hierarchical structures that grow exponentially the deeper into the hierarchy one goes[10, 23].

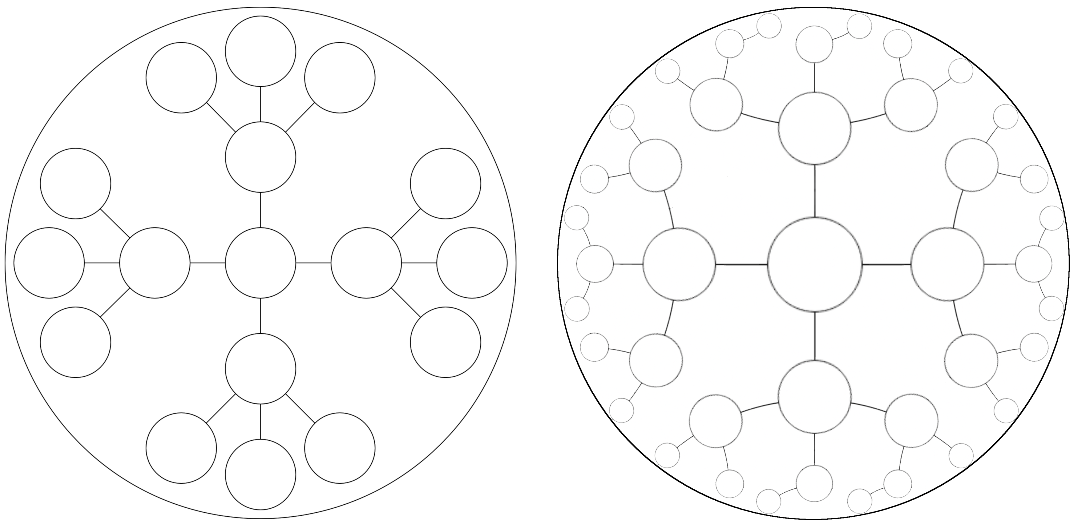
Figure 2.5: Comparison between representing a tree-like hierarchical structure in Euclidean space (left) vs Hyperbolic space (right)

Hyperbolic geometry is a non-Euclidean geometry where the parallel postulate behaves differently. In Euclidean geometry, for different points A and B, and a line L passing through A but not B, there exists only one line passing through B parallel to L. On the other hand, in hyperbolic geometry, there exist at least two lines passing through B parallel to L[27]. This leads to many differing properties, with one important property being the exponential growth of volumes of hyperspheres compared to the polynomial growth in Euclidean space[17, 23]. Furthermore, hyperbolic geometry has a more tree-like structure compared to the grid structure of Euclidean geometry, making it more suitable for representing hierarchies[10]. Figure 2.5 shows a tree contained within a circular subspace in Euclidean geometry vs hyperbolic geometry. The exponential growth in hyperbolic geometry allows for deeper trees with more nodes to be represented in a circle with the same radius[27].

Another property of hyperbolic geometry is that it has a constant negative curvature $-\kappa$, which has the effect that every point in hyperbolic geometry is a saddle point. This is in contrast to Euclidean geometry with constant zero curvature, and spherical geometry with constant positive curvature.

2.2.1 Representing Hyperbolic Space

There exist multiple isometric models to represent hyperbolic geometry, with each its own metrics and properties. In this section, three models will be described, namely the Lorentz hyperboloid[5, 27], the Poincaré[15, 23, 27], and the Klein[22, 27] models. The models will be defined, and relevant properties and formulas will be introduced, such as the distance between two points and the mapping function from Euclidean geometry to the respective model.

The Lorentz Hyperboloid Model

Hyperbolic geometry in the Lorentz hyperboloid model $\mathbb{L}^n$ is represented as the upper sheet of a hyperboloid in $\mathbb{R}^{n+1}$:

$$\mathbb{L}^n_\kappa = \{\mathbf{x} \in \mathbb{R}^{n+1} | \langle \mathbf{x}, \mathbf{x} \rangle_\mathbb{L} = -\frac{1}{\kappa}, x_0 > 0\} \tag{2.20}$$

Where $\langle \mathbf{x}, \mathbf{x} \rangle_\mathbb{L} = -x_0^2 + x_1^2 + \cdots + x_n^2$ is the Lorentz inner product. Borrowing from special relativity theory, a point in the Lorentz hyperboloid model can be written as $\mathbf{x}_\mathbb{L} = [x_{time}; \mathbf{x}_{space}]$. Representing the axis of symmetry of the hyperboloid as the time dimension and simplifying the Lorentz inner product to:

$$\langle \mathbf{x}_\mathbb{L}, \mathbf{y}_\mathbb{L} \rangle_\mathbb{L} = -x_{time} y_{time} + \langle \mathbf{x}_{space}, \mathbf{y}_{space} \rangle \tag{2.21}$$

Where, $\langle \mathbf{x}, \mathbf{y} \rangle$ is the Euclidean dot product. Additionally, due to the constraint $\langle \mathbf{x}, \mathbf{x} \rangle_\mathbb{L} = -\frac{1}{\kappa}$ it is possible to calculate the value of x_{time} from $\mathbf{x}_{space}$:

$$x_{time} = \sqrt{1/\kappa + \langle \mathbf{x}_{space}, \mathbf{x}_{space} \rangle} = \sqrt{1/\kappa + ||\mathbf{x}_{space}||^2} \tag{2.22}$$

This leads to only needing to store $\mathbf{x}_{space}$ when working with vectors in the Lorentz hyperboloid model. Furthermore, the distance between two points on the Lorentz hyperboloid is given as:

$$d_\mathbb{L}(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{1}{\kappa} \cosh^{-1}(-\kappa \langle \mathbf{x}, \mathbf{y} \rangle_\mathbb{L})} \tag{2.23}$$

And lastly, the exponential map [1]can be used to map vectors from the Euclidean space to the Lorentz hyperboloid:

$$\mathbf{x}_{space} = expm(\mathbf{v}) = \frac{\sinh(\sqrt{\kappa}\|\mathbf{v}\|)}{\sqrt{\kappa}\|\mathbf{v}\|}\mathbf{v} \tag{2.24}$$

Where $\mathbf{v} \in \mathbb{R}^n$ is a point in Euclidean geometry. For convenience, the Lorentz hyperboloid model will be called the Lorentz model in the rest of the thesis.

The Poincaré Model

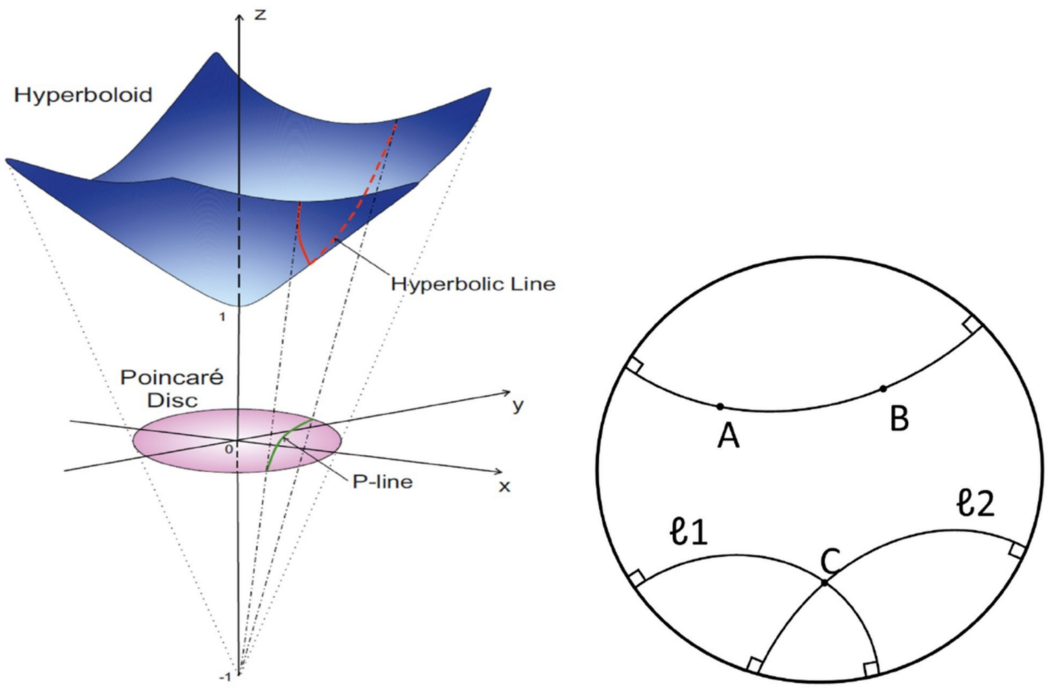

Figure 2.6: Figure showing the projection mapping from the Lorentz hyperboloid $\mathbb{L}^2$ to the Poincaré disc $\mathbb{P}^2$ in $\mathbb{R}^3$. Also highlighting how the projected geodesics end up becoming arcs in the Poincaré disc that are perpendicular to the disc's edge[25].

The Poincaré model is isometric to the Lorentz model, and can be constructed by projecting the Lorentz hyperboloid in $\mathbb{R}^{n+1}$ into an n-dimensional hypersphere of

[1]This exponential map works only when $\langle \mathbf{O}_{\mathbb{L}}, \mathbf{v} \rangle_{\mathbb{L}} = 0$, or in other words, when $\mathbf{v}$ is in the tangent space of $\mathbf{O}_{\mathbb{L}}$, where $\mathbf{O}_{\mathbb{L}}$ is origo of the hyperboloid. This is always the case for vectors in $\mathbb{R}^n$ and a hyperboloid in $\mathbb{R}^{n+1}$. There exists a more general exponential map[5], but it requires explaining projections, which are not relevant to the project. The exponential map for the other models will also be only valid at origo.

radius $1/\sqrt{\kappa}$ at the origin of $\mathbb{R}^{n+1}$:

$$\mathbf{x}_{\mathbb{P}} = \frac{\mathbf{x}_{space}}{x_{time} + \sqrt{\kappa}} \tag{2.25}$$

Where $\mathbf{x}_{\mathbb{P}}$ is a point in the Poincaré model, and an example of this projection is shown in Figure 2.6. Due to this projection, the set of points in the Poincaré model are:

$$\mathbb{P}^n_\kappa = \{\mathbf{x} \in \mathbb{R}^n; \|\mathbf{x}\|^2 < \frac{1}{\kappa}\} \tag{2.26}$$

and the distance between two points in the Poincaré hypersphere is derived as:

$$d_{\mathbb{P}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{\kappa}} \cosh^{-1}\left(1 + \frac{2\|\mathbf{x} - \mathbf{y}\|^2}{(1 - \|\mathbf{x}\|^2)(1 - \|\mathbf{y}\|^2)}\right) \tag{2.27}$$

One property of the Poincaré model is that its geodesics, or lines of shortest distance, are circular arcs perpendicular to the boundary of the hypersphere. Figure 2.6 shows an example on the Poincaré disc $\mathbb{P}^2$. Another property is that the Poincaré model is conformal, meaning that the hyperbolic angles in the Poincaré model are measured as one would normally measure angles in Euclidean space:

$$\angle(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}\right) \tag{2.28}$$

This makes it is possible to utilize cosine similarity in the Poincaré model. Lastly, the exponential map in the Poincaré model is:

$$\mathbf{x}_{\mathbb{P}} = expm(\mathbf{v}) = \frac{\tanh(\sqrt{\kappa}\|\mathbf{v}\|)}{\sqrt{\kappa}\|\mathbf{v}\|}\mathbf{v}, \mathbf{v} \in \mathbb{R}^n \tag{2.29}$$

The Klein Model

The Klein model is also isometric to the Lorentz model, and is constructed by projecting the Lorentz hyperboloid in $\mathbb{R}^{n+1}$ into an n-dimensional hypersphere of radius $1/\sqrt{\kappa}$ with center at the origo of the hyperboloid $\mathbf{O}_{\mathbb{L}} = [1/\sqrt{\kappa}; \mathbf{0}]$:

$$\mathbf{x}_{\mathbb{K}} = \frac{\mathbf{x}_{space}}{\sqrt{\kappa} x_{time}} \tag{2.30}$$

Where $\mathbf{x}_{\mathbb{K}}$ is a point in the Klein model, and an example of this projection can be seen in Figure 2.7. As the Klein model is also a hypersphere of radius $1/\sqrt{\kappa}$, its set of points is the same as the Poincaré model:

$$\mathbb{K}^n_\kappa = \{\mathbf{x} \in \mathbb{R}^n; \|\mathbf{x}\|^2 < \frac{1}{\kappa}\} \tag{2.31}$$

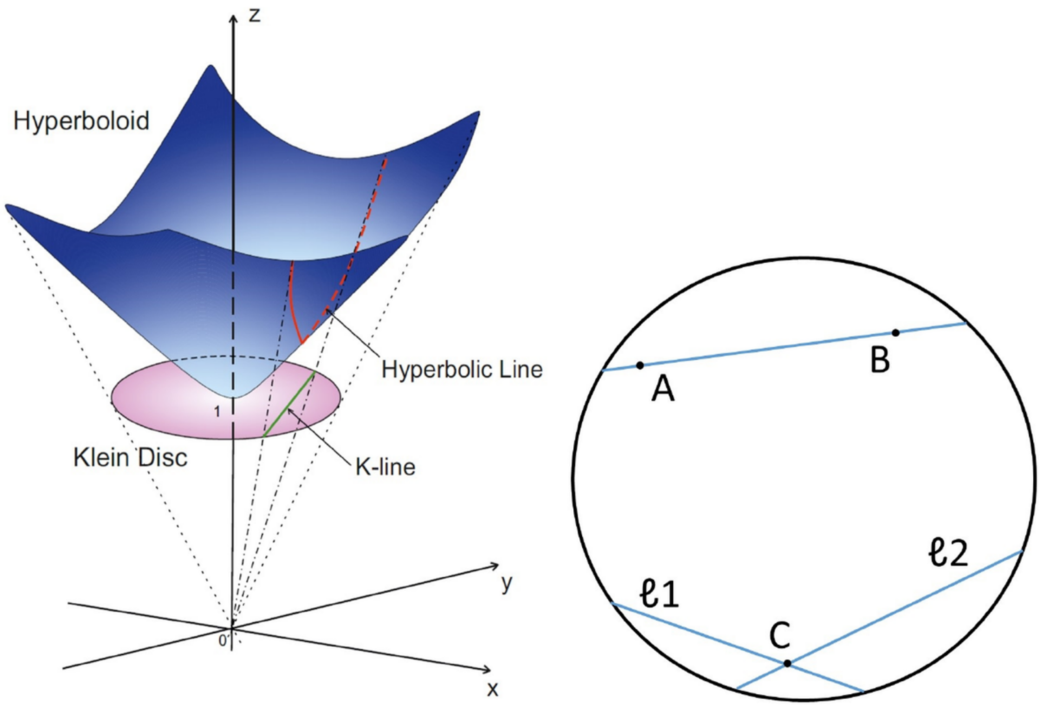

Figure 2.7: Figure showing the projection mapping from the Lorentz hyperboloid $\mathbb{L}^2$ to the Klein disc $\mathbb{K}^2$ in $\mathbb{R}^3$. Also highlighting how the projected geodesics end up as straight lines in the Klein disc[25].

However, since it is mapped differently from the Lorentz hyperboloid, its distance function between two points is different from the Poincaré model, and is derived as:

$$d_{\mathbb{K}}(\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{\kappa}} \cosh^{-1}\left(\frac{1-\langle \mathbf{x},\mathbf{y}\rangle}{\sqrt{1-\|\mathbf{x}\|^2}\sqrt{1-\|\mathbf{y}\|^2}}\right) \tag{2.32}$$

and geodesics in the Klein model are straight lines in $\mathbb{R}^n$, as seen in Figure 2.7 for the Klein disc $\mathbb{K}^2$. This makes it straightforward to define a mid-point in the Klein model[23], with one such midpoint being the Einstein mid-point:

$$\mu_{\mathbb{K}} = \frac{\sum_{i=1} \gamma_i \mathbf{x}_{\mathbb{K},i}}{\sum_{i=1} \gamma_i} \tag{2.33}$$

Where γ_i are the Lorentz factors:

$$\gamma_i = \frac{1}{\sqrt{1-\kappa\|\mathbf{x}_{\mathbb{K},i}\|^2}} \tag{2.34}$$

Lastly, the exponential map in the Klein model[2] is:

$$\mathbf{x}_{\mathbb{K}} = expm(\mathbf{v}) = \frac{\tanh(\sqrt{\kappa}\|\mathbf{v}\|)}{\sqrt{\kappa}\|\mathbf{v}\|}\mathbf{v}, \mathbf{v} \in \mathbb{R}^n \tag{2.35}$$

2.2.2 Hyperbolic Visual Learning

Hyperbolic geometry has been used in many computer vision models, ranging from simple classification to generative models, and used in both supervised and unsupervised learning scenarios[23]. This section will describe methods utilizing hyperbolic machine learning to construct feature space embeddings of visual data.

Hyperbolic Self-Supervised Learning

Most hyperbolic learning methods utilize a Euclidean backbone network to generate Euclidean embeddings. This is followed by an exponential map to map the Euclidean embeddings into one of the hyperbolic models, usually the Poincaré model[23]. Therefore, when performing unsupervised or self-supervised learning, it is important to modify the loss functions to work with the hyperbolic embeddings. This can be done by replacing the original distance or similarity function with the hyperbolic distance. For example, the hyperbolic triplet loss proposed by Hsu et al. [12] is:

$$\mathcal{L}_i = \max(0, d_{\mathbb{H}}(\mathbf{z}_i, \mathbf{z'}_i) - d_{\mathbb{H}}(\mathbf{z}_i, \mathbf{z}_n) + \alpha) \tag{2.36}$$

Where $d_{\mathbb{H}}$ is a hyperbolic distance function, $\mathbf{z'}_i$ is an augmentation of $\mathbf{z}_i$, and $\mathbf{z}_n$ is a negative sample. On the other hand, the hyperbolic contrastive loss proposed Yue et al. [36] is:

$$\mathcal{L}_i = -\log\left(\frac{exp(-d_{\mathbb{H}}(\mathbf{z}_i, \mathbf{z'}_i)/\tau)}{\sum_n \mathbb{1}_{[n \neq i]} \exp(-d_{\mathbb{H}}(\mathbf{z}_i \mathbf{z}_n)/\tau)}\right) \tag{2.37}$$

Replacing the dot product similarity with the negative hyperbolic distance $d_{\mathbb{H}}$.

Furthermore, to take advantage of hyperbolic geometry's ability to represent hierarchies. Hsu et al. [12] argue that there exist hierarchies within medical images and create multiple levels of patches in 3D voxel-grid biomedical images. They then use triplet loss to learn hierarchies between parent patches, and smaller child batches within them:

$$\mathcal{L}_i = \max(0, d_{\mathbb{H}}(\mathbf{z}_p, \mathbf{z}_c) - d_{\mathbb{H}}(\mathbf{z}_p, \mathbf{z}_n + \alpha) \tag{2.38}$$

[2]While this is identical to the Poincaré exponential map, this is only the case for points in the tangent space of the origo.

Where $\mathbf{z}_p$ are the parent patches, $\mathbf{z}_c$ are the child patches within the parent patches, and $\mathbf{z}_n$ are negative patches that do not overlap with the parent patches.

Ge et al. [7] utilize the hierarchy present between a scene and the objects within it. They define a scene as a bounding box region with multiple objects in it. Then they utilize the contrastive loss function:

$$\mathcal{L}_{\text{hyp}} = -\log \frac{\exp\left(-\frac{d_{\mathbb{H}}(z_1, z_2)}{\tau}\right)}{\exp\left(-\frac{d_{\mathbb{H}}(z_1, z_2)}{\tau}\right) + \sum_n \exp\left(-\frac{d_{\mathbb{H}}(z_1, z_n)}{\tau}\right)} \tag{2.39}$$

Where z_1 is the scene, z_2 is an object in the scene, and z_n are negative objects. They show that this loss is minimized when the objects are moved closer to the center of the Poincaré hypersphere. Furthermore, it ensures that a scene's embedding is close to the embeddings of the objects inside it.

Desai et al. [5] train MERU, a large-scale vision-language model utilizing hyperbolic contrastive learning and entailment loss to learn the hierarchy present between an image and its textual description. They utilize the contrastive loss function in Eq. 2.37 to learn embeddings for the images $\mathbf{y}$ and their text descriptions $\mathbf{x}$ in the Lorentz model and define the entailment loss as:

$$\mathcal{L}_{entail}(\mathbf{x}, \mathbf{y}) = \max(0, ext(\mathbf{x}, \mathbf{y}) - aper(\mathbf{x})) \tag{2.40}$$

Where $ext(\mathbf{x}, \mathbf{y})$ is the exterior angle between two points and $aper(\mathbf{x})$ is the aperture of a point's entailment cone. This loss penalizes the model only if the image embedding is outside its text embeddings' entailment cone. However, in order to calculate this loss, they derive equations for the aperture of an embedding and the exterior angle between two embeddings in the Lorentz model. The derived aperture is:

$$aper(\mathbf{x}) = \sin^{-1}\left(\frac{2K}{\sqrt{\kappa}\|\mathbf{x}_{space}\|}\right) \tag{2.41}$$

Where κ is the curvature and K is a constant, setting boundary conditions near the origin. While the derived exterior angle between the image and text embeddings is:

$$ext(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\frac{y_{time} + x_{time}\kappa\langle\mathbf{x}, \mathbf{y}\rangle_{\mathbb{L}}}{\|\mathbf{x}_{space}\|\sqrt{(\kappa\langle\mathbf{x}, \mathbf{y}\rangle_{\mathbb{L}})^2 - 1}}\right) \tag{2.42}$$

2.2.3 Hyperbolic Classification

Classification in hyperbolic geometry requires using different algorithms than its Euclidean counterpart. This section will describe some of the algorithms used to perform hyperbolic classification using either class prototypes or Fully Connected (FC) layers to find class logits.

Prototype-Based Classification

Prototype-based classification aims at learning class prototypes or representatives, which can be used under inference to classify new samples[9]. One example of a method performing prototypical classification is Vanilla GCD by Vaze et al. [31], as they utilize k-Means to find class prototypes, and assign a new sample to the class whose prototype is closest to the sample's embedding.

K-Means is also utilized in hyperbolic learning. Khrulkov et al. [14] create a hyperbolic prototypical network using the Einstein midpoint to calculate class prototypes. In order to find the Einstein midpoint for embeddings in the Poincaré model, they first map the points to the Klein model, calculate the Einstein midpoint, and then map the midpoint back to the Poincaré model. On the other hand, Hsu et al. [12] compute the prototypes using the weighted Lorentz centroid proposed by Law et al. [19]:

$$\mu_{\mathbb{L}} = \frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i \mathbf{x}_i}{|\|\sum_{i=1} w_i \mathbf{x}_i\|_{\mathbb{L}}|} \tag{2.43}$$

Where w_i are arbitrary sample weights and $|\|\mathbf{x}\|_{\mathbb{L}}| = |\sqrt{\langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}}}|$ is the modulus of the imaginary Lorentz norm, as the Lorentz inner product can be negative. To compute this centroid, they first map the Poincaré embeddings into the Lorentz hyperboloid, compute the Lorentz centroid, and then map the centroid back to the Poincaré model.

Ghadimi Atigh et al. [9] use ideal points, or points at infinity, as prototypes. In the Poincaré model, the ideal points are the points at the edge of the hypersphere $\|\mathbf{x}\|^2 = 1/\kappa$. First, they pre-initialize a prototype for each class, then they train using a loss based on the Busemann function. The Busemann function in a Poincaré model of curvature $\kappa = 1$ is:

$$b_{\mathbf{p}}(\mathbf{z}) = \log\left(\frac{\|\mathbf{p} - \mathbf{z}\|^2}{1 - \|\mathbf{z}\|^2}\right) \tag{2.44}$$

Where $\mathbf{p}$ is the ideal prototype. The Busemann function is used to measure distance to the prototype, and the Busemann loss they use is:

$$\mathcal{L}(\mathbf{z}, \mathbf{p}) = b_{\mathbf{p}}(\mathbf{z}) - \phi(d) log(1 - \|\mathbf{z}\|^2) \tag{2.45}$$

Where $\phi(d)$ is a scaling factor for the regularization term $log(1 - \|\mathbf{z}\|^2)$ based on the dimension of the Poincaré ball d. Using the Busemann function as a loss teaches the model to get embeddings closer to their prototypes (the numerator), while penalizing embeddings too close to the edge of the hypersphere (the denominator). The regularization term is used to increase the penalty of the denominator and force embeddings farther away from the edge of the hypersphere.

This makes it possible to initialize prototypes without prior knowledge, proving helpful in few-shot learning scenarios.

Logits-Based Classification

Logit-based classification aims at computing a probability vector containing the probability of a sample belonging to a class k:

$$l_n = p(y = k|\mathbf{x}) \tag{2.46}$$

This is commonly accomplished by using a Fully Connected (FC) layer $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b} : \mathbb{R}^m \to \mathbb{R}^n$ with $\mathbf{W}$ being a weights matrix and $\mathbf{b}$ being the bias vector. The FC layer can also be understood as evaluating the distance between the vector $\mathbf{x}$ and n hyperplanes, one for each class k.

Ganea et al. [6] define the hyperbolic FC layer in the Poincaré model as:

$$f_{\mathbb{P}}(\mathbf{x}) = \mathbf{W} \otimes_\kappa \mathbf{x} \oplus_\kappa \mathbf{b} \tag{2.47}$$

Where $\mathbf{W} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{P}^m$, $\mathbf{b} \in \mathbb{P}^n$, $\otimes_\kappa$ is the gyrovector-matrix product defined as:

$$\mathbf{W} \otimes_\kappa \mathbf{x} = \frac{1}{\sqrt{\kappa}} \tanh\left(\frac{\|\mathbf{W}\mathbf{x}\|}{\|\mathbf{x}\|} \tanh^{-1}(\sqrt{\kappa}\|\mathbf{x}\|)\right) \frac{\mathbf{W}\mathbf{x}}{\|\mathbf{W}\mathbf{x}\|} \tag{2.48}$$

And $\oplus_\kappa$ is the gyrovector addition operator:

$$\mathbf{x} \oplus_\kappa \mathbf{y} := \frac{(1 + 2\kappa\langle\mathbf{x},\mathbf{y}\rangle) + \kappa\|\mathbf{y}\|^2)\mathbf{x} + (1 - \kappa\|\mathbf{x}\|^2)\mathbf{y}}{1 + 2\kappa\langle\mathbf{x},\mathbf{y}\rangle + \kappa^2\|\mathbf{x}\|^2\|\mathbf{y}\|^2} \tag{2.49}$$

However, their definition of the gyrovector-matrix multiplication first maps the Poincaré vector back to Euclidean space, then computes the Euclidean vector-matrix multiplication, then transforms the produced vector back to the Poincaré model. This is only equivalent to computing the distance to Poincaré hyperplanes if $\mathbf{b} = \mathbf{0}$[29]. That is why Shimizu et al. [29] define their hyperbolic FC layer as:

$$f_{\mathbb{P}}(\mathbf{x}) = \mathbf{v}(1 + \sqrt{1 + \kappa\|\mathbf{v}\|^2})^{-1} \tag{2.50}$$

$$\mathbf{v} := [\kappa^{-1/2}\sinh(\sqrt{\kappa}(v_k(\mathbf{x}))]_{k=1}^n \tag{2.51}$$

Where $v_k(\mathbf{x})$ is the hyperbolic multinomial logistic regression they define as:

$$v_k(\mathbf{x}) = 2\kappa^{-\frac{1}{2}}\|\mathbf{w}_k\| \sinh^{-1}\left(\lambda_{\mathbf{x}}^\kappa\langle\sqrt{\kappa}\mathbf{x},\mathbf{w}_k\rangle\right)\cosh\left(2\sqrt{\kappa}b_k\right) - (\lambda_{\mathbf{x}}^\kappa - 1)\sinh\left(2\sqrt{\kappa}b_k\right) \tag{2.52}$$

Where $\mathbf{w}_k$ are the rows of the weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ and b_k are elements in the bias vector $\mathbf{b} \in \mathbb{P}^n$. This formulation of the FC layer produces Poincaré hyperplanes for any bias vector $\mathbf{b}$.

Chen et al. [3] developed a FC layer for the Lorentz model. First they define the matrix parameters $\mathbf{M} = \begin{bmatrix}\mathbf{v}^\intercal \\ \mathbf{W}\end{bmatrix} \in \mathbb{R}^{(m+1)\times(n+1)}$. Then they define the function

$$f_{\mathbf{x}}(\mathbf{M}) = \begin{bmatrix}\frac{\sqrt{\|\mathbf{M}\mathbf{x}\|^2 + 1/\kappa}}{\mathbf{v}^\intercal\mathbf{x}}\mathbf{v}^\intercal \\ \mathbf{W}\end{bmatrix} \tag{2.53}$$

Ensuring that the product produces a vector on the Lorentz hyperboloid $f_{\mathbf{x}}(\mathbf{M})\mathbf{x} \in \mathbb{L}^n$ and $\langle f_{\mathbf{x}}(\mathbf{M})\mathbf{x}, f_{\mathbf{x}}(\mathbf{M})\mathbf{x}\rangle_{\mathbb{L}} = -1/\kappa$. They show that the matrix $\mathbf{M}$ can represent any Lorentz transformation. Furthermore, they generalize the FC layer to adding activation, bias and normalization:

$$f(\mathbf{x}) = \begin{bmatrix} \sqrt{\|\phi(\mathbf{W}\mathbf{x}, \mathbf{v})\|^2 + 1/\kappa} \\ \phi(\mathbf{W}\mathbf{x}, \mathbf{v}) \end{bmatrix} \tag{2.54}$$

Where:

$$\phi(\mathbf{W}\mathbf{x}, \mathbf{v}) = \frac{\lambda\sigma(\mathbf{v}^{\intercal}\mathbf{x} + b')}{\|\mathbf{W}h(\mathbf{x}) + \mathbf{b}\|}(\mathbf{W}h(\mathbf{x}) + \mathbf{b}) \tag{2.55}$$

This includes the activation $h(\mathbf{x})$, biases $\mathbf{b} \in \mathbb{R}^n, b' \in \mathbb{R}$ and normalization $\|\mathbf{W}h(\mathbf{x}) + \mathbf{b}\|$. Furthermore, $\sigma(\mathbf{x})$ is the sigmoid function and $\lambda > 0$ controls the scaling range. However, despite the output still being on the Lorentz hyperboloid, this relaxation is no longer fully hyperbolic and incorporates Euclidean operations, introducing a trade-off between adding extra functionality to the FC layer and performing all operations in hyperbolic geometry.

2.3 Hyperbolic Generalized Category Discovery

Pre-training	Model	CUB			Stanford Cars			FGVC-Aircraft			Cifar10			Cifar100		
		All	Old	New	All	Old	New	All	Old	New	All	Old	New	All	Old	New
DINO	Vanilla GCD	51.3	56.6	48.7	39.0	57.6	29.9	45.0	41.1	46.9	91.5	97.9	88.2	73.0	76.2	66.5
	Hyp-GCD	61.0	67.0	58.0	50.8	60.9	45.8	48.2	43.6	50.5	92.9	97.5	90.6	74.0	80.0	62.0
	SimGCD	60.3	65.6	57.7	53.8	71.9	45.0	54.2	59.1	51.8	**97.1**	95.1	98.1	80.1	81.2	77.8
	Hyp-SimGCD	64.8	65.8	64.2	62.8	73.4	**57.7**	58.7	58.9	58.5	96.8	95.9	**97.2**	**82.4**	83.1	**81.2**
	SelEx	73.6	75.3	72.8	58.5	75.6	50.3	57.1	64.7	53.3	95.9	**98.1**	94.8	82.3	85.3	76.3
	Hyp-SelEx	**79.8**	**75.8**	**81.8**	**62.9**	**80.0**	54.7	**65.9**	**67.3**	**65.1**	96.7	97.6	96.3	**82.4**	**85.1**	77.0
DINOv2	Vanilla GCD	71.9	71.2	72.3	65.7	67.8	64.7	55.4	47.9	59.2	97.8	99.0	97.1	79.6	84.5	69.9
	Hyp-GCD	75.6	75.1	75.9	72.8	80.4	69.1	62.7	70.0	59.0	97.5	98.9	96.7	84.5	87.5	78.5
	SimGCD	71.5	78.1	68.3	71.5	81.9	66.6	63.9	69.9	60.9	98.7	96.7	**99.7**	88.5	89.2	87.2
	Hyp-SimGCD	77.6	77.9	77.4	82.5	85.8	**81.0**	76.4	70.3	79.4	**98.9**	97.7	99.5	**91.5**	90.0	**94.6**
	SelEx	87.4	85.1	88.5	82.2	**93.7**	76.7	79.8	**82.3**	78.6	98.5	**98.8**	98.5	87.7	90.8	81.5
	Hyp-SelEx	**90.7**	**85.3**	**93.4**	**83.8**	93.3	79.2	**83.4**	82.0	**84.1**	98.6	98.1	98.9	88.6	**91.5**	82.8

Table 2.3: Performance of the hyperbolic GCD methods compared to their Euclidean counterparts[20]. The highest accuracy under every column for each backbone is highlighted.

In parallel with this thesis, Liu et al. [20] released a paper titled "Hyperbolic Category Discovery", which analyzes the performance of different GCD models when trained with Hyperbolic machine learning algorithms.

Liu et al. [20] adapt three GCD method, namely the Vanilla GCD, SimGCD and SelEx. For all three methods, they adapt the representation learning by doing joint contrastive learning based on hyperbolic distance and angles. Since they use the Poincaré model, which is conformal to Euclidean geometry, the angle-based contrastive loss becomes the usual contrastive loss that uses dot product similarity. The full loss function is:

$$\mathcal{L} = (1-\lambda)((1-\alpha)\mathcal{L}^{s}_{dis} + \alpha\mathcal{L}^{s}_{ang}) + \lambda((1-\alpha)\mathcal{L}^{u}_{dis} + \alpha\mathcal{L}^{u}_{ang}) \tag{2.56}$$

Where λ balances between supervised and self-supervised contrastive learning, and α balances between distance and angle losses. They also choose to linearly decrease α from 1 to 0 during training.

To avoid vanishing gradients[11], they clip their Euclidean embeddings before mapping them to the Poincaré model using the function:

$$\mathcal{C}(\mathbf{z}) = \min\left(1, \frac{r}{\|\mathbf{z}\|}\right)\mathbf{z} \tag{2.57}$$

Where $\mathbf{z}$ is the embedding and r is a hyperparameter for the maximum embedding norm. Then, following the exponential map $\mathbf{z}_{\mathbb{P}} = expm_{\mathbb{P}}(\mathcal{C}(\mathbf{z}))$, they clip the Poincaré embedding to avoid having embeddings close to the edge of the hypersphere, as they cause numerical instabilities:

$$\mathcal{C}_{\mathbb{P}}(\mathbf{z}_{\mathbb{P}}) = \min\left(1, \frac{1-10^{-3}}{\sqrt{\kappa}\|\mathbf{z}_{\mathbb{P}}\|}\right)\mathbf{z}_{\mathbb{P}} \tag{2.58}$$

For non-parametric clustering, they remove the projector after training, and perform Euclidean K-Means on the embeddings outputted by the ViT backbone. For parametric clustering, they implement the Poincaré FC layer developed by Ganea et al. [6], utilizing a Riemannian Adam optimizer[1] and the original classification losses from SimGCD to learn its parameters.

They achieve State of The Art (SOTA) results, showing the benefits of utilizing hyperbolic machine learning to solve the GCD task. Table 2.3 shows the results they report in their paper.

Chapter 3

Problem Statement

The GCD task was described in Section 2.1, alongside four different methods that tried to solve it. One notable method was the SOTA SelEx method, which uses Hierarchical Semi-Supervised K-Means (HSSK) to generate hierarchical clusters to use as pseudo-labels. The SelEx method demonstrated the importance of taking hierarchies present in the data into account.

Section 2.2 highlighted Euclidean geometry's sub-optimal ability to represent hierarchies and presented hyperbolic geometry, which is much more suited for representing hierarchies. This makes hyperbolic geometry a more suitable candidate for solving the GCD task. This is later demonstrated to be true in Section 2.3 in the recent work by Liu et al. [20], who adapt numerous GCD methods to hyperbolic geometry and show improved performance.

Liu et al. [20] only adapt the GCD tasks to the Poincaré model, while it was shown by Desai et al. [5] that the Lorentz model can be used to train on large text-image datasets and is suitable for learning hierarchies between text and images. This makes it worth investigating if it can model the hierarchies that are beneficial for the GCD task.

Another choice made by Liu et al. [20] is to perform non-parametric clustering in Euclidean geometry, by utilizing the output embeddings from the ViT backbone instead of the MLP projector. This could potentially lead to a loss of performance, as removing the projector and utilizing Euclidean geometry can lead to a loss of some of the learned hierarchical representation. This makes it interesting to investigate the performance of non-parametric clustering in hyperbolic geometry instead of Euclidean geometry, for both the Lorentz and Poincaré models.

With these facts in mind, this thesis aims to answer the following two research questions:

"Does the GCD task benefit from learning representations in the Lorentz Hyperboloid model of hyperbolic geometry?

"Does using a hyperbolic non-parametric clustering algorithm increase clustering accuracy?"

Chapter 4

Methods

Following the problem statement, this thesis will adapt one non-parametric and one parametric GCD method to use hyperbolic learning in the Lorentz model to analyze their performance in comparison to using Euclidean learning. The non-parametric method will be Vanilla GCD and the parametric method will be SimGCD. Furthermore, the HypCD methods presented by Liu et al. [20] for Vanilla GCD and SimGCD will also be adapted to act as hyperbolic baselines, and to test the accuracy of hyperbolic K-Means in both Lorentz and Poincaré space.

Section 4.1 will present the hyperbolic representation learning setup that will be used to learn representations in hyperbolic geometry. Section 4.2 will formalize the hyperbolic K-Means algorithms for the Lorentz and Poincaré models. Lastly, Section 4.3 will detail how parametric clustering in hyperbolic geometry will be performed using the different hyperbolic FC layers.

4.1 Hyperbolic Representation Learning

Following previous GCD methods, a pre-trained ViT backbone with a MLP projector is utilized, with the projector and the last layer of the ViT being fine-tuned. Passing an image through the backbone and projector produces the Euclidean embedding $\mathbf{z}$, which is then clipped and mapped into the Lorentz model:

$$\mathbf{z}_{space} = exmp_{\mathbb{L}}(\mathcal{C}(\mathbf{z})) \tag{4.1}$$

Where $expm_{\mathbb{L}}(\mathbf{x})$ is the Lorentz exponential map defined in Eq. 2.24, $\mathcal{C}(\mathbf{x})$ is the Euclidean clipping function defined in Eq. 2.57, and $\mathbf{z}_{space}$ is the space component of the embedding in the Lorentz model.

Following HypCD by Liu et al. [20], hyperbolic representation learning will be done using a combination of distance-based and angle-based contrastive loss, as this setup was empirically shown to produce the best results in Section 5.3.2 The contrastive losses from Eq. 2.1 and Eq. 2.2 are adapted to the Lorentz model by using the distance for the Lorentz model defined in Eq. 2.23:

$$\mathcal{L}^u_{dist,i} = -\log\left(\frac{\exp(-d_{\mathbb{L}}(\mathbf{z}_{\mathbb{L},i}\mathbf{z}'_{\mathbb{L},i})/\tau)}{\sum_n \mathbb{1}_{[n\neq i]}\exp(-d_{\mathbb{L}}(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},n})/\tau)}\right) \tag{4.2}$$

$$\mathcal{L}^s_{dist,i} = -\frac{1}{\mathcal{N}(i)}\sum_{q\in\mathcal{N}(i)}\log\left(\frac{\exp(-d_{\mathbb{L}}(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},q})/\tau)}{\sum_n \mathbb{1}_{[n\neq i]}\exp(-d_{\mathbb{L}}(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},n})/\tau)}\right) \tag{4.3}$$

And the exterior angle derived by Desai et al. [5] and found in this thesis report at Eq. 2.42:

$$\mathcal{L}^u_{ang,i} = -\log\left(\frac{\exp(ext(\mathbf{z}_{\mathbb{L},i}\mathbf{z}'_{\mathbb{L},i})/\tau)}{\sum_n \mathbb{1}_{[n\neq i]}\exp(ext(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},n})/\tau)}\right) \tag{4.4}$$

$$\mathcal{L}^s_{ang,i} = -\frac{1}{\mathcal{N}(i)}\sum_{q\in\mathcal{N}(i)}\log\left(\frac{\exp(ext(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},q})/\tau)}{\sum_n \mathbb{1}_{[n\neq i]}\exp(ext(\mathbf{z}_{\mathbb{L},i}\mathbf{z}_{\mathbb{L},n})/\tau)}\right) \tag{4.5}$$

Similar to HypCD, the four losses are combined as:

$$\mathcal{L} = (1-\lambda)((1-\alpha)\mathcal{L}^s_{dis} + \alpha\mathcal{L}^s_{ang}) + \lambda((1-\alpha)\mathcal{L}^u_{dis} + \alpha\mathcal{L}^u_{ang})$$

With λ being the weighting hyperparameter between self-supervised and supervised contrastive loss, and α being the weighting hyperparameter between distance and angle-based losses, whith the α weight being linearly decayed from 1 to 0 as training progresses.

4.2 Hyperbolic K-Means

The K-Means algorithm is an iterative algorithm that aims to find K prototypes to represent a dataset. K-Means iteratively performs the following two steps:

- Assign every datapoint to one of the K prototypes
- Update prototype K by moving it to the center of all datapoints assigned to it

In order to adapt this algorithm to hyperbolic geometry, a distance function and a centroid calculation function are needed for both the Poincaré and Lorentz models.

4.2.1 Computing Centroid in Hyperbolic Geometry

Two closed-form solutions to finding centroids in hyperbolic geometry are the Einstein midpoint[23] in the Klein model $\mu_{\mathbb{K}}$ and the Lorentz centroid[19] in the Lorentz model $\mu_{\mathbb{L}}$:

$$\mu_{\mathbb{K}} = \frac{\sum_{i=1} \gamma_i \mathbf{x}_{\mathbb{K},i}}{\sum_{i=1} \gamma_i}$$
$$\mu_{\mathbb{L}} = \frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}}{|\|\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}\|_{\mathbb{L}}|}$$

Where $\gamma_i = \frac{1}{\sqrt{1-\kappa\|x_{\mathbb{K}}\|^2}}$ are the Lorentz factors and w_i are arbitrary sample weights. While neither of these two centroid functions can be directly computed in the Poincaré model, they can still be computed by mapping the points from Poincaré to Klein or Lorentz, computing the centroid there, and then mapping the centroid back to Poincaré. However, despite the two equations finding midpoints in two different hyperbolic models, it can be proven that mapping the Lorentz centroid to the Klein model produces the Einstein midpoint.

In order to prove that the Lorentz centroid maps to the Einstein midpoint, it is necessary to derive the reverse mapping from points in the Klein model to points on the Lorentz hyperboloid:

Theorem 1. *The function mapping points from the Klein model to points on the Lorentz hyperboloid is:*

$$\pi_{\mathbb{K}\to\mathbb{L}}(\mathbf{x}_{\mathbb{K}}) = \frac{1}{\sqrt{\kappa - \kappa^2\|\mathbf{x}_{\mathbb{K}}\|^2}}[1; \sqrt{\kappa}\mathbf{x}_{\mathbb{K}}] \tag{4.6}$$

Proof. Using the property $x_{time} = \sqrt{1/\sqrt{\kappa} + \|\mathbf{x}_{space}\|^2}$, the Function 2.22 mapping points from the Lorentz hyperboloid to the Klein model can be written as:

$$\mathbf{x}_{\mathbb{K}} = \pi_{\mathbb{L}\to\mathbb{K}}(\mathbf{x}_{\mathbb{L}}) = \frac{\mathbf{x}_{space}}{\sqrt{\kappa}x_{time}} = \frac{\mathbf{x}_{space}}{\sqrt{\kappa}\sqrt{1/\kappa + \|\mathbf{x}_{space}\|^2}} \tag{4.7}$$

On the other hand, by rearranging the same mapping function, one gets:

$$\mathbf{x}_{space} = \sqrt{\kappa}x_{time}\mathbf{x}_{\mathbb{K}} \tag{4.8}$$

By replacing $\mathbf{x}_{space}$ in Eq. 4.7 with the value in Eq. 4.8:

$$\mathbf{x}_{\mathbb{K}} = \frac{\sqrt{\kappa}x_{time}}{\sqrt{\kappa}\sqrt{1/\kappa + \|\sqrt{\kappa}x_{time}\mathbf{x}_{\mathbb{K}}\|^2}}\mathbf{x}_{\mathbb{K}} = \frac{x_{time}}{\sqrt{1/\kappa + \kappa x_{time}^2\|\mathbf{x}_{\mathbb{K}}\|^2}}\mathbf{x}_{\mathbb{K}} \tag{4.9}$$

Meaning that:

$$\begin{aligned}1 &= \frac{x_{time}}{\sqrt{1/\kappa + \kappa x_{time}^2 ||\mathbf{x}_{\mathbb{K}}||^2}} \\ 1/\kappa + \kappa x_{time}^2 ||\mathbf{x}_{\mathbb{K}}||^2 &= x_{time}^2 \\ 1/\kappa &= (1 - \kappa ||\mathbf{x}_{\mathbb{K}}||^2) x_{time}^2 \\ \frac{1}{\kappa(1 - \kappa ||\mathbf{x}_{\mathbb{K}}||^2)} &= x_{time}^2 \\ \frac{1}{\sqrt{\kappa - \kappa^2 ||\mathbf{x}_{\mathbb{K}}||^2}} &= x_{time}\end{aligned}$$

Inserting the value of x_{time} into Eq. 4.8:

$$\mathbf{x}_{space} = \sqrt{\kappa} \frac{1}{\sqrt{\kappa - \kappa^2 ||\mathbf{x}_{\mathbb{K}}||^2}} \mathbf{x}_{\mathbb{K}} \tag{4.10}$$

Hence:

$$\pi_{\mathbb{K} \to \mathbb{L}}(\mathbf{x}_{\mathbb{K}}) = [x_{time}; \mathbf{x}_{space}] = \frac{1}{\sqrt{\kappa - \kappa^2 ||\mathbf{x}_{\mathbb{K}}||^2}} [1; \sqrt{\kappa} \mathbf{x}_{\mathbb{K}}] \qquad \square$$

With this proof, it is possible to compute the Einstein midpoint directly from points on the Lorentz hyperboloid:

Corollary 1.1. *The Lorentz factors in the Einstein midpoint can be written as:*

$$\gamma_i = \frac{\sqrt{\kappa}}{\sqrt{\kappa - \kappa^2 ||\mathbf{x}_{\mathbb{K}}||^2}} = \sqrt{\kappa} x_{time} \tag{4.11}$$

Hence, in combination with Eq. 4.7, the Einstein midpoint can be written as:

$$\mu_{\mathbb{K}} = \frac{\sum_{i=1} \sqrt{\kappa} x_{time,i} \frac{\mathbf{x}_{space,i}}{\sqrt{\kappa} x_{time,i}}}{\sum_{i=1} \sqrt{\kappa} x_{time,i}} = \frac{\sum_{i=1} \mathbf{x}_{space,i}}{\sqrt{\kappa} \sum_{i=1} x_{time,i}} \tag{4.12}$$

Lastly, proving that the Einstein midpoint is the Lorentz centroid mapped to the Klein model:

Theorem 2. *The Einstein midpoint $\mu_{\mathbb{K}}$ can be found by passing the Lorentz Centroid $\mu_{\mathbb{L}}$ through the map $\pi_{\mathbb{L} \to \mathbb{K}}$:*

$$\mu_{\mathbb{K}} = \pi_{\mathbb{L} \to \mathbb{K}}(\mu_{\mathbb{L}}) \tag{4.13}$$

Proof. The Lorentz centroid can be split into its separate space and time components as:

$$\mu_{time} = \frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i x_{time}}{|||\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}||_{\mathbb{L}}|} \tag{4.14}$$

$$\mu_{space} = \frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i \mathbf{x}_{space,i}}{|||\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}||_{\mathbb{L}}|} \tag{4.15}$$

Using these components the mapping $\pi_{\mathbb{L}\to\mathbb{K}}(\mu_{\mathbb{L}})$ evaluates to:

$$\pi_{\mathbb{L}\to\mathbb{K}}(\mu_{\mathbb{L}}) = \frac{\mu_{space}}{\sqrt{\kappa}\mu_{time}} = \frac{\frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i \mathbf{x}_{space,i}}{|\|\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}\|_{\mathbb{L}}|}}{\sqrt{\kappa}\frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i x_{time}}{|\|\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}\|_{\mathbb{L}}|}} = \frac{\sum_{i=1} w_i \mathbf{x}_{space,i}}{\sqrt{\kappa}\sum_{i=1} w_i x_{time}} \tag{4.16}$$

Which is equal to the Einstein midpoint μ_κ if the weights w_i are all equal. $\square$

With this proof, it can be concluded that choosing either of the two midpoints is equivalent. The Lorentz centroid is the obvious choice for finding centroids in the Lorentz model, as no mappings between models are needed. However, it is not possible to avoid mapping between models to find a centroid in the Poincaré model. Therefore, the Einstein midpoint will be used to compute centroids in the Poincaré model, as mapping between Klein and Poincaré model is more numerically stable than mapping between Lorentz and Poincaré spaces[3]. The mapping functions between Klein and Poincaré are[23]:

$$\pi_{\mathbb{P}\to\mathbb{K}} = \frac{2\mathbf{x}_{\mathbb{P}}}{1+\kappa\|\mathbf{x}_{\mathbb{P}}\|^2} \tag{4.17}$$

$$\pi_{\mathbb{K}\to\mathbb{P}} = \frac{\mathbf{x}_{\mathbb{K}}}{1+\sqrt{1-\kappa\|\mathbf{x}_{\mathbb{K}}\|^2}} \tag{4.18}$$

4.2.2 Adapting the K-Means Algorithm to Hyperbolic Geometry

The K-Means algorithm can be seen in Algorithm 1. As input, it takes the input vectors $\mathbf{X}$, the number of prototypes or centroids to find K, and a tolerance value to stop iterating ε. It starts by initializing the prototypes, then every iteration it assigns labels to the data samples according to the prototype with the minimum distance. Thereafter, for each label, it computes the centroid of the points assigned to it and assigns it as the new prototype. The distance shift of all prototypes is saved in δ and the algorithm stops when $\delta < \varepsilon$, where it returns a matrix μ of K prototypes.

Algorithm 1: General K-Means algorithm

```
   Input: Data: $\mathbf{X}$
   Number of centroids: K
   Center shift tolerance: $\varepsilon$
   Output: Matrix of K centroid prototypes: $\mu$
1  Initialize the matrix $\mu$ with K vectors
2  $\delta \leftarrow 0$
3  while $\delta > \varepsilon$ do
4      $\delta \leftarrow 0$
5      Compute pairwise distance between $\mathbf{X}$ and $\mu$
6      Assign each point in $\mathbf{X}$ to $\mu_k$ with lowest distance, $d$
7      for $k \leftarrow 1$ to $K$ do
8          $\mu_{k,old} \leftarrow \mu_k$
9          $\mathbf{X}_k \leftarrow$ Points in $\mathbf{X}$ with labels $\mu_k$
10         $\mu_k \leftarrow$ centroid($\mathbf{X}_k$)
11         $\delta \leftarrow \delta + d(\mu_k, \mu_{old,k})$
12 return $\mu$
```

It is possible to modify the K-Means algorithm to work in any geometry by simply using an appropriate distance and centroid functions. For example, Euclidean K-Means is done by using the Euclidean distance function $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ and assigning the centroid as the mean of all the points in matrix $\mathbf{X}$. Similarly, for the Lorentz model of hyperbolic geometry, the Lorentz hyperboloid distance and the Lorentz centroid can be used as distance and centroid functions:

$$d_{\mathbb{L}}(\mathbf{x}, \mathbf{y}) = \sqrt{\frac{1}{\kappa} \cosh^{-1}(-\kappa \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbb{L}})}$$
$$\mu_{\mathbb{L}} = \frac{1}{\sqrt{\kappa}} \frac{\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}}{|\|\sum_{i=1} w_i \mathbf{x}_{\mathbb{L},i}\|_{\mathbb{L}}|}$$

While for the Poincaré model of hyperbolic geometry, the Poincaré distance and

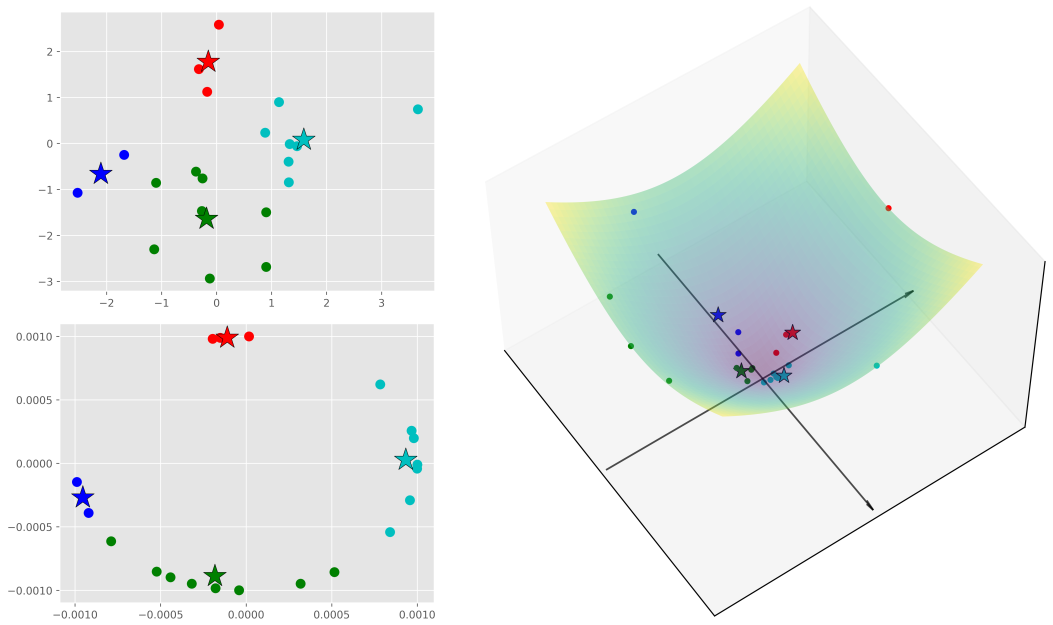

Figure 4.1: Figure showing the K-Means algorithm performed in Eulcidean (Top left), Poincaré (Bottom left), and Lorentz (Right) spaces on a set of points and four cluster.

Einstein midpoint can be used:

$$d_{\mathbb{P}}(\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{\kappa}}\cosh^{-1}\left(1+\frac{2\|\mathbf{x}-\mathbf{y}\|^2}{(1-\|\mathbf{x}\|^2)(1-\|\mathbf{y}\|^2)}\right)$$

$$\mu_{\mathbb{P}} = \pi_{\mathbb{K}\to\mathbb{P}}(\mu_{\mathbb{K}}) = \pi_{\mathbb{K}\to\mathbb{P}}\left(\frac{\sum_{i=1}\gamma_i\pi_{\mathbb{P}\to\mathbb{K}}(\mathbf{x}_{\mathbb{P},i})}{\sum_{i=1}\gamma_i}\right)$$

With the points being mapped to the Klein model to find the Einstein midpoint, and then mapping the Einstein midpoint back to the Poincaré model. An example of K-Means in each of the three spaces can be seen in Figure 4.1, where K-Means is performed on 25 points with 4 prototypes.

Lastly, the semi-supervised K-Means algorithm is shown in Algorithm 2. In addition to the classic K-Means algorithm, it works with a labeled data subset $\mathbf{X}_L$, where the prototypes for the points in the labeled subset are assigned according to the ground truth labels $\mathbf{y}_L$ instead of nearest distance. Everything else remains the same, including the adjustments needed for each of the three spaces.

Algorithm 2: Semi-supervised K-Means algorithm

```
   Input: Labeled data: $\mathbf{X}_L$
   Data labels: $\mathbf{y}_K$
   Unlabeled data: $\mathbf{X}_U$
   Number of centroids: K
   Center shift tolerance: $\varepsilon$
   Output: Matrix of K centroid prototypes: $\mu$
1  Initialize the matrix $\mu$ with K vectors
2  $\delta \leftarrow 0$
3  while $\delta > \varepsilon$ do
4      $\delta \leftarrow 0$
5      Assign each point in $\mathbf{X}_L$ to the centroid $\mu_k$, where $k$ is the
         corresponding label value in $\mathbf{y}_k$
6      Compute pairwise distance between $\mathbf{X}_U$ and $\mu$, $d$
7      Assign each point in $\mathbf{X}$ to $\mu_k$ with lowest distance
8      for $k \leftarrow 1$ to $K$ do
9          $\mu_{k,old} \leftarrow \mu_k$
10         $\mathbf{X}_k \leftarrow$ Points in $\mathbf{X}_L$ and $\mathbf{X}_U$ with labels $\mu_k$
11         $\mu_k \leftarrow$ centroid($\mathbf{X}_k$)
12         $\delta \leftarrow \delta + d(\mu_k, \mu_{old,k})$
13 return $\mu$
```

4.3 Parametric Clustering

To adapt SimGCD to hyperbolic geometry, it is necessary to calculate logit probabilities directly from hyperbolic embeddings. This can be done with the FC layers introduced in Section 2.2.3.

For embeddings in the Lorentz model, the FC layer proposed by Chen et al. [3] can be used:

$$\hat{\mathbf{y}}_{\mathbb{L}} = f_{\mathbf{x}_{\mathbb{L}}}(\mathbf{M})\mathbf{x}_{\mathbb{L}} = \begin{bmatrix} \frac{\sqrt{\|\mathbf{M}\mathbf{x}\|^2+1/\kappa}}{\mathbf{v}^{\mathsf{T}}\mathbf{x}}\mathbf{v}^{\mathsf{T}} \\ \mathbf{W} \end{bmatrix} \mathbf{x}_{\mathbb{L}}$$

Where $\mathbf{M} \in \mathbb{R}^{(n+1)\times(m+1)}$ is the weights matrix and $\hat{\mathbf{y}}_{\mathbb{L}}$ is the logits vector in the Lorentz model. This layer is used instead of the relaxed layer, as computing logits does not require a bias term or an activation function. Furthermore, the layer is simplified to:

$$\hat{\mathbf{y}}_{space} = \mathbf{W}\mathbf{x}_{\mathbb{L}} \tag{4.19}$$

As the time component of the logit vector is not needed. This reduces the need for the vector $\mathbf{v}$ and the function $f_{\mathbf{x}_{\mathbb{L}}}(\mathbf{M})$, only needing to optimize the weights matrix

$\mathbf{W} \in \mathbb{R}^{n\times(m+1)}$.

For embeddings in the Poincaré model, both the layer proposed by Ganea et al. [6]:

$$\hat{\mathbf{y}}_{\mathbb{P}} = \mathbf{W} \otimes_\kappa \mathbf{x} = \frac{1}{\sqrt{\kappa}} \tanh\left(\frac{\|\mathbf{W}\mathbf{x}\|}{\|\mathbf{x}\|} \tanh^{-1}(\sqrt{\kappa}\mathbf{x})\right) \frac{\mathbf{W}\mathbf{x}}{\|\mathbf{W}\mathbf{x}\|}$$

And the layer proposed by Shimizu et al. [29]:

$$\begin{gathered} \hat{\mathbf{y}}_{\mathbb{P}} = f_{\mathbb{P}}(\mathbf{x}) = \mathbf{v}(1 + \sqrt{1 + \kappa\|\mathbf{v}\|^2})^{-1} \\ \mathbf{v} := [\kappa^{-1/2} \sinh(\sqrt{\kappa}(v_k(\mathbf{x}))]_{k=1}^{n} \end{gathered}$$

Can be used, as both calculate the distance to valid hyperplanes in the Poincaré model when the bias term is removed. $\hat{\mathbf{y}}_{\mathbb{P}}$ is the logits vector in the Poincaré model and $v_k(\mathbf{x})$ is the multinomial logistic regression from Eq. 2.52.

Chapter 5

Results

This chapter describes the training and testing setup used for the hyperbolic Vanilla GCD and SimGCD implementations. Presenting the main results using Euclidean, Poincaré and Lorentz based clustering methods. Followed by several ablation studies on the Lorentz space Vanilla GCD model.

5.1 Experiment Setup

In order to present comparable results, it was necessary to train all models on the same machine. Therefore, the training code for HypCD was written from scratch following the details from the paper by Liu et al. [20], as no code for their experiments had been released by the time this thesis was written. This potentially leads to different performance results being reported in this thesis compared to what Liu et al. [20] have in their paper.[1]

5.1.1 Dataset Setup

The datasets used are the three SSB fine-grained datasets (CUB, Stanford Cars and FGVC-Aircraft), alongside CIFAR-10 and CIFAR-100 as generic datasets. The splitting of the classes and data subsets follows the splitting by Vaze et al. [31] detailed in Section 2.1.3. The data augmentation procedure during training is:

- Resize image to a size of 224×224
- Random horizontal flip with 50% chance

[1]The code for the hyperbolic Vanilla GCD can be found at `https://github.com/MohamadDalal/hyperbolic-generalized-category-discovery`, while the code for the hyperbolic SimGCD can be found at `https://github.com/MohamadDalal/Hyperbolic-SimGCD`.

- Normalize image following ImageNet normalization parameters. With mean $\mu = [0.485, 0.456, 0.406]$ and standard deviation $\sigma = [0.229, 0.224, 0.225]$. Each value corresponds to height, width and depth, respectively.

During testing, the same augmentations are used with the exception of the random horizontal flip.

5.1.2 Vanilla GCD Setup

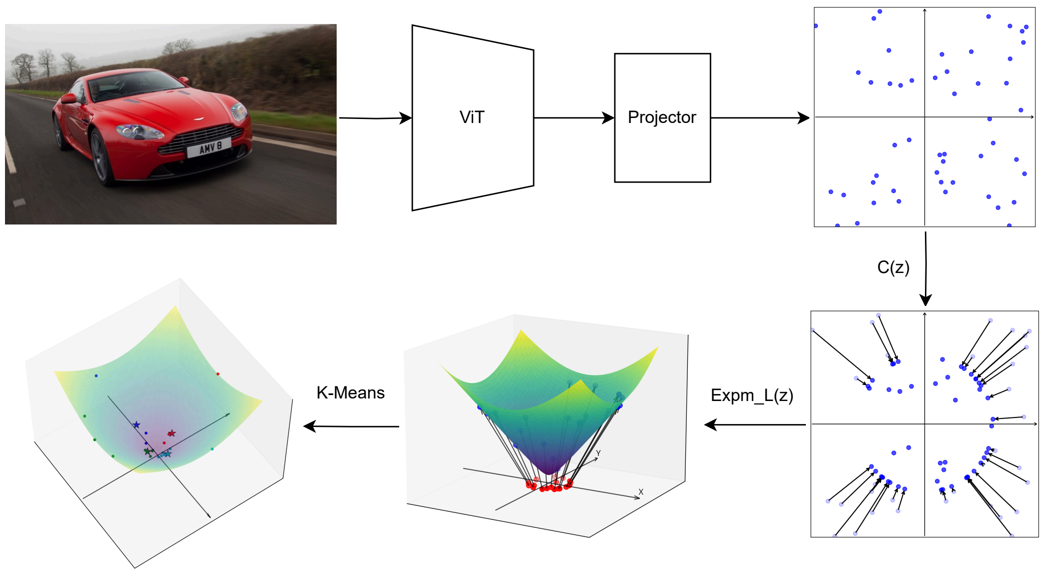

Figure 5.1: The full inference pipeline for the Vanilla GCD model with Lorentz representation learning and clustering. During training, the last clustering step is removed, and contrastive learning is performed on the embeddings on the Lorentz hyperboloid.

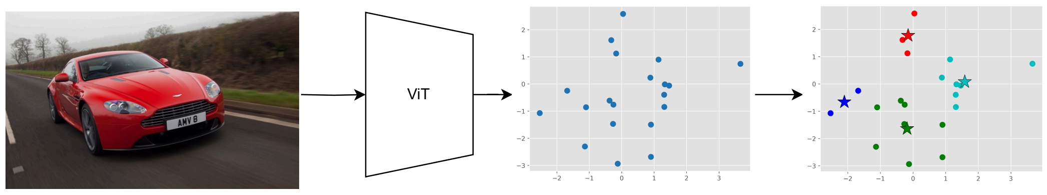

Figure 5.2: The inference pipeline for the Vanilla GCD model when using Euclidean K-Means. The embeddings directly from the ViT backbone are used to perform clustering.

Three models were trained, each performing contrastive learning in a different geometry or hyperbolic model. All models use a ViT-b-14 backbone pre-trained with DINOv2, and a projector with 4 linear layer and Gaussian Error Linear Unit

(GELU) activations in-between. The last layer is special in that it has no bias or activation.

The model performing contrastive learning in Euclidean geometry follows Vanilla GCD[31] by using embeddings of 65536 dimensions, leading to a projector with the following setup:

$$\mathbb{R}^I \to \mathbb{R}^{2048} \to \mathbb{R}^{2048} \to \mathbb{R}^{256} \to \mathbb{R}^{65536}$$

Where $I = 768$ is the embedding dimension from the ViT backbone. The supervised and self-supervised contrastive learning in Euclidean geometry only use cosine similarity as the similarity function.

The models performing contrastive learning in hyperbolic geometry follow the HypCD training setup[20] by using embeddings of 256 dimensions:

$$\mathbb{R}^I \to \mathbb{R}^{2048} \to \mathbb{R}^{2048} \to \mathbb{R}^{256} \to \mathbb{R}^{256}$$

As hyperbolic learning benefits from embeddings of lower dimensions[20, 23]. Similar to HypCD, these output embeddings from the projector are then clipped with a clipping value of $r = 2.3$ and mapped into their respective hyperbolic models. As an example, for the Lorentz model, this mapping can be represented as:

$$\mathbb{R}^{256} \xrightarrow{expm(\mathcal{C}(\mathbf{z}))} \mathbb{L}^{256}$$

The supervised and self-supervised contrastive learning in hyperbolic geometry uses a combination of distance and angle-based similarity, as described in Section 4.1. Furthermore, the weight of the angle-based contrastive loss is decreased linearly from 1 to 0 throughout training. Lastly, the curvature is frozen to a value of $\kappa = 0.05$, as is the case in HypCD.

All experiments are run with a batch size of 128. The learning rate is initialized as 0.1 with a cosine annealing scheduler reducing it to 0.0 at the end of training and Stochastic Gradient Descent (SGD) is used to optimize all models. The hyperbolic models utilize gradient clipping, where all gradients are first clipped to a max of 1.0, and then the mean value of all gradients is clipped to 0.25. This is due to hyperbolic representation learning being unstable, especially in the Lorentz model, where exploding gradients were frequently encountered. The Euclidean model does not utilize any gradient clipping.

After the models are trained, K-Means is used to find cluster prototypes. When using Euclidean K-Means, the projector is removed, and the embeddings directly from the backbone are used as shown in Figure 5.2. On the other hand, the hyperbolic K-Means algorithms use the embeddings in hyperbolic space, which do not require removing the projector, as the exponential map is performed on the clipped projector outputs. Figure 5.1 shows the full pipeline for a Lorentz model with Lorentz K-Means.

During testing, the trained model is used to compute embeddings for the test samples. Then the embeddings are assigned to the closest prototype, and the assignment is evaluated by the protocol described in Section 2.1.4.

5.1.3 SimGCD Setup

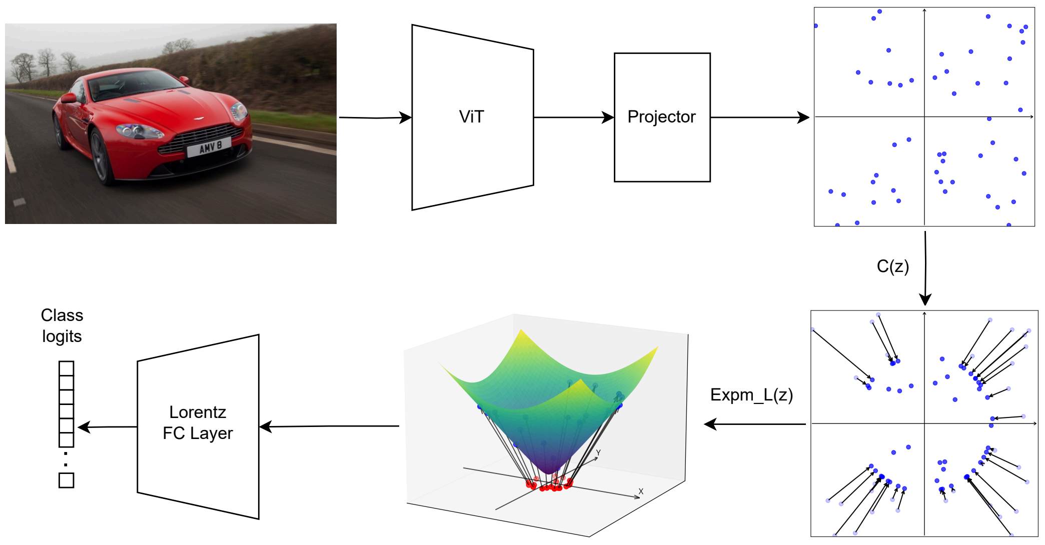

Figure 5.3: The full pipeline for the hyperbolic SimGCD model with embeddings and classification in Lorentz space. During training, the embeddings from the Lorentz hyperboloid are used for the contrastive losses before being passed to the Lorentz FC layer, while the class logits from the FC layer are used for the classification losses.

Much of the setup for SimGCD is identical to that for Vanilla GCD. Therefore, only the differences will be highlighted in this chapter, and anything omitted is identical to Vanilla GCD's setup.

The first difference is in the projector's last layer, as it is now used to output class logits. For the Euclidean model, the last layer has an output of $|\mathcal{Y}_{\mathcal{U}}|$:

$$\mathbb{R}^{I} \to \mathbb{R}^{2048} \to \mathbb{R}^{2048} \to \mathbb{R}^{256} \to \mathbb{R}^{|\mathcal{Y}_{\mathcal{U}}|}$$

While for the hyperbolic models, the embedding is clipped and mapped to hyperbolic space before being passed through a hyperbolic FC layer for classification. For example, in Lorentz space:

$$\mathbb{R}^{I} \to \mathbb{R}^{2048} \to \mathbb{R}^{2048} \to \mathbb{R}^{256} \xrightarrow{expm(\mathcal{C}(\mathbf{z}))} \mathbb{L}^{256} \to \mathbb{L}^{|\mathcal{Y}_{\mathcal{U}}|}$$

All three hyperbolic FC layers shown in Section 4.3 are implemented, and an example of the SimGCD pipeline with Lorentz embeddings can be seen in Figure 5.3.

The input embeddings to the last layer in the projector are used in the contrastive losses. Otherwise, the contrastive loss setup for SimGCD is identical to that of Vanilla GCD.

For classification loss, the same setup used in the original SimGCD by Wen et al. [35] is used. As described in Section 2.1.2, for labeled data, cross-entropy loss is computed between ground truth values and computed logits. For unlabeled data, a pseudo-label is generated using another view of the same image and with sharper logit values. Then cross-entropy loss is computed between the pseudo-labels and the computed logits minus a mean-entropy maximization regularizer.

Identical to the Vanilla GCD setup, all experiments are run with a batch size of 128. The learning rate is initialized as 0.1 with a cosine annealing scheduler, and Stochastic Gradient Descent (SGD) is used to optimize all models. The hyperbolic models utilize gradient clipping, with max clipping of 1.0 and mean clipping of 0.25, while the Euclidean model does not utilize any gradient clipping.

During testing, the trained model is used to generate class logits, which are used to assign each sample to a class. Then the assignment is evaluated by the protocol described in Section 2.1.4.

5.2 Results

The results from the different Vanilla GCD and SimGCD setups are presented and discussed in the following section. All experiments were run using a single Nvidia L4 GPU, and the model with the least overall loss was used to perform clustering and testing.

5.2.1 Vanilla GCD

The results from the Vanilla GCD experiments can be seen in Tables 5.1 and 5.2. The experiments aim to investigate the performance of representation learning in the Lorentz model for hyperbolic geometry and whether using a hyperbolic K-Means algorithm to perform clustering can benefit the GCD task.

Focusing on the first investigation point, Table 5.1 highlights the benefits of training using embeddings in the Lorentz model, compared to the Poincaré model and Euclidean geometry. The models trained on embeddings in the Lorentz model achieve the highest accuracy in the fine-grained datasets, with the exception of FGVC-Aircraft, where they only outperform in the accuracy of unseen classes. As

Contrastive Training	K-Means Algorithm	CUB			Stanford Cars			FGVC-Aircraft		
		All	Old	New	All	Old	New	All	Old	New
Euclidean	Euclidean	55.20	69.66	40.86	65.20	82.22	48.79	45.48	58.57	32.37
Poincaré	Poincaré	00.52	01.04	00.00	00.85	00.00	01.66	59.11	**74.46**	43.72
Poincaré	Euclidean	70.47	78.92	62.10	72.62	88.45	57.34	**64.36**	73.32	55.38
Lorentz	Lorentz	**73.04**	**83.70**	62.47	**76.18**	**90.98**	**61.91**	61.69	66.13	**57.25**
Lorentz	Euclidean	72.95	82.07	**63.92**	72.50	86.07	58.66	60.97	70.32	51.59

Table 5.1: Accuracy of the Vanilla GCD models on the fine-grained SSB datasets. For the hyperbolic models, performance is reported using both the Euclidean and hyperbolic K-Means algorithms. The highest accuracy for each column is highlighted.

Contrastive Training	K-Means Algorithm	CIFAR-10			CIFAR-100		
		All	Old	New	All	Old	New
Euclidean	Euclidean	**88.97**	88.16	**78.78**	81.70	88.31	55.25
Poincaré	Poincaré	10.00	20.00	00.00	01.00	00.00	05.00
Poincaré	Euclidean	88.17	**98.99**	78.46	83.23	83.84	**80.80**
Lorentz	Lorentz	72.16	98.26	46.06	**88.62**	91.45	77.30
Lorentz	Euclidean	88.04	98.04	78.04	85.50	**91.99**	62.75

Table 5.2: Accuracy of the Vanilla GCD models on the generic CIFAR datasets. For the hyperbolic models, performance is reported using both the Euclidean and hyperbolic K-Means algorithms. The highest accuracy for each column is highlighted.

for the accuracy on the generic dataset in Table 5.2, the Lorentz models achieve much higher accuracy on CIFAR100, while achieving mixed results on CIFAR10. This might suggest that learning embeddings in the Lorentz model is better when more classes are involved. However, this requires further testing to be conclusive.

For the second investigation point, only performance between models using the same embeddings is considered. One striking result is the poor performance of the Poincaré K-Means algorithm, achieving very low accuracy in almost all datasets. Upon further investigation, it was discovered that the Poincaré K-Means algorithm is very sensitive to the initialization of the prototypes. Most of the time, all unlabeled points end up being assigned to a single prototype, which leads to non of the other prototypes being updated. This then leads to all test samples being assigned to a single cluster, resulting in very low accuracies.

While it is not fully known why the Poincaré K-Means algorithm is unstable, it is safer to use Euclidean K-Means when training embeddings in the Poincaré model. On the other hand, the Lorentz K-Means algorithm leads to clear improvements in accuracy for all fine-grained datasets and the CIFAR100 generic dataset. This showcases the importance of performing non-parametric clustering in hyperbolic geometry when performing representation learning in the Lorentz space.

5.2.2 SimGCD

Contrastive	FC	CUB			Stanford Cars			FGVC-Aircraft		
Training	Layer	All	Old	New	All	Old	New	All	Old	New
Euclidean	Euclidean	64.70	**79.99**	49.55	**74.24**	**87.31**	61.64	**67.18**	**68.94**	**65.41**
Poincaré	Poincaré	59.73	67.79	51.75	56.40	66.03	47.10	48.33	48.68	47.99
Poincaré	Poincaré++	**66.47**	61.27	**71.62**	64.83	65.40	**64.28**	54.13	52.58	55.68
Lorentz	Lorentz	51.38	77.77	25.22	55.69	78.44	33.74	54.94	66.13	43.72

Table 5.3: Accuracy of the SimGCD models on the fine-grained SSB datasets. The FC layer column highlights the layer used to calculate logits for parametric clustering. Euclidean is a simple matrix multiplication, Poincaré corresponds to the layer by Ganea et al. [6], Poincaré++ corresponds to the layer by Shimizu et al. [29], and Lorentz corresponds to the layer by Chen et al. [3]. The highest accuracy for each column is highlighted.

The SimGCD experiments highlight new results from using the Hyperbolic Neural Networks++ layer by Shimizu et al. [29] for the Poincaré model and the Lorentz FC layer by Chen et al. [3] for the Lorentz model. These are compared to the Euclidean SimGCD by Wen et al. [35] and the hyperbolic SimGCD by Liu et al. [20]. Only results for the fine-grained SSB datasets are shown due to the long training time needed to train the SimGCD models.

The results for all 4 models are shown in Table 5.3. In contrast to the results produced by Liu et al. [20], which are shown in Table 2.3, the Euclidean SimGCD performs the best in almost all datasets, while the Lorentz and Poincaré SimGCD produce much lower performance. These results show that SimGCD would not benefit from hyperbolic learning. However, further experimentation and analysis would be needed to understand the contrast to the results provided by Liu et al. [20].

The results in Table 5.3 do showcase a much improved performance when using the Poincaré layer defined by Shimizu et al. [29] (denoted as Poincaré++ in the table).

5.3 Ablations

Several ablation studies are carried out to investigate the effect of different hyper-parameter values and training regimes. All ablations were done using the Vanilla GCD setup with Lorentz representation learning and only on the fine-grained SSB datasets.

5.3.1 Euclidean Clipping

Euclidean	K-Means	CUB			Stanford Cars			FGVC-Aircraft		
Clipping	Algorithm	All	Old	New	All	Old	New	All	Old	New
✓	Lorentz	73.04	83.70	62.47	**76.18**	**90.98**	**61.91**	61.69	66.13	57.25
✗	Lorentz	**73.82**	**82.73**	**65.05**	74.93	89.31	61.06	**65.92**	**80.10**	51.71
✓	Euclidean	72.95	82.07	63.92	72.50	86.07	58.66	60.97	70.32	51.59
✗	Euclidean	71.49	80.86	62.20	70.87	86.75	55.56	65.41	72.30	**58.60**

Table 5.4: Accuracy of the Lorentz embeddings Vanilla GCD model when trained with a Euclidean clipping of $r = 2.3$ and without any Euclidean clipping. The highest accuracy for each column is highlighted.

The main issues encountered while training the models during this thesis were exploding gradients rather than vanishing gradients. Therefore, the need for the Euclidean clipping function $\mathcal{C}(\mathbf{z})$ is investigated.

Table 5.4 presents the results of this ablation study. From these results, it can be seen that clipping the Euclidean embeddings is not needed when training in the Lorentz model and clustering using Lorentz K-Means, with a large gain in performance for the FGVC-Aircraft dataset. However, removing the clipping does have a slight impact when clustering with the Euclidean K-Means algorithm.

5.3.2 Angle Loss

Contrastive	K-Means	CUB			Stanford Cars			FGVC-Aircraft		
Metric	Algorithm	All	Old	New	All	Old	New	All	Old	New
Distance&Angle	Lorentz	**73.04**	**83.70**	62.47	**76.18**	**90.98**	**61.91**	**61.69**	66.13	**57.25**
Angle Only	Lorentz	54.37	73.54	35.36	56.24	78.47	36.72	44.79	58.27	31.29
Distance&Angle	Euclidean	72.95	82.07	**63.92**	72.50	86.07	58.66	60.97	**70.32**	51.59
Angle Only	Euclidean	65.43	78.88	52.10	65.92	81.31	51.09	53.20	65.17	41.20

Table 5.5: Accuracy of the Lorentz embeddings Vanilla GCD model when trained with a combination of angle-based and distance-based losses, and when trained with only angle-based loss. The highest accuracy for each column is highlighted.

While training the different hyperbolic models, it was observed that using Euclidean clipping can lead to all Euclidean embeddings having the same norm, and hence all lying on a hypersphere with radius r. Therefore, it is worth investigating the importance of using both angle-based and distance-based losses when all embeddings have the same norm.

Table 5.5 presents the results from this ablation study. It can be seen that a combination of angle and distance losses is needed to achieve high performance, despite all Euclidean embeddings lying in a hypersphere with radius r.

5.3.3 Embedding Dimension

Embedding Dimension	K-Means Algorithm	CUB			Stanford Cars			FGVC-Aircraft		
		All	Old	New	All	Old	New	All	Old	New
128	Lorentz	73.01	84.12	61.99	74.57	89.69	59.98	**65.56**	**72.12**	**58.98**
256	Lorentz	**73.04**	83.70	62.47	**76.18**	**90.98**	61.91	61.69	66.13	57.25
768	Lorentz	70.61	**87.52**	55.68	75.95	90.35	62.06	58.42	61.87	54.95
65536	Lorentz	72.56	84.36	60.86	75.54	89.06	62.50	59.32	77.40	50.15
128	Euclidean	71.14	82.14	60.24	72.09	88.22	56.54	63.91	72.00	55.80
256	Euclidean	72.95	82.07	**63.92**	72.50	86.07	58.66	60.97	70.32	51.59
768	Euclidean	70.07	79.37	60.86	71.77	85.22	**62.67**	61.99	70.38	53.57
65536	Euclidean	72.21	81.45	63.06	74.57	88.60	61.03	59.71	65.47	53.93

Table 5.6: Accuracy of the Lorentz embeddings Vanilla GCD model when trained with different embedding dimensions. The highest accuracy for each column is highlighted.

To investigate the fact that representation learning benefits from lower embedding dimensions, models with four different embedding dimensions were trained. The dimensions chosen are 128, 256, 768, and 65536. 256 is the output of the second-to-last layer of the projector and the value used by Liu et al. [20] in HypCD, while 128 is half of that value. 768 is the dimension of the outputs from the ViT backbones, and 65536 is the dimension used in the original Vanilla GCD paper for Euclidean training.

Table 5.6 presents the results of this ablation study. The accuracy scores are very mixed across all embedding dimensions, except with FGVC-Airplane, which benefits from the lowest dimension of 128. These results showcase that representation learning in the Lorentz space can perform well in both high and low embedding dimensions. However, more experiments are needed to investigate how low the dimension can be decreased before a loss in performance is incurred.

5.3.4 Learning Curvature Value

Desai et al. [5] opt to learn the curvature value when training their large vision-language model MERU. This ablation aims to see if the Lorentz Vanilla GCD model can benefit from learning the curvature instead of freezing it to a value of $\kappa = 0.05$. Similar to Desai et al. [5], the curvature is learned in logarithmic space, and is clipped between the values $[0.1\kappa_I, 10\kappa_I]$, where κ_I is the initial curvature. The initial curvature value is chosen to be 0.1 for this ablation study. Furthermore, to avoid instability in training the curvature, the gradient of the curvature value was clipped to a maximum of 0.1.

Learned Curvature	K-Means Algorithm	CUB All	CUB Old	CUB New	Stanford Cars All	Stanford Cars Old	Stanford Cars New	FGVC-Aircraft All	FGVC-Aircraft Old	FGVC-Aircraft New
✓	Lorentz	**73.04**	**83.70**	62.47	**76.18**	**90.98**	**61.91**	61.69	66.13	**57.25**
✗	Lorentz	64.07	81.90	46.39	66.93	83.69	50.77	61.36	**75.84**	46.85
✓	Euclidean	72.95	82.07	**63.92**	72.50	86.07	58.66	60.97	70.32	51.59
✗	Euclidean	67.22	81.10	53.47	67.07	81.48	53.16	**63.37**	74.82	51.89

Table 5.7: Accuracy of the Lorentz embeddings Vanilla GCD model when trained with a frozen curvature of $\kappa = 0.05$ and with learned curvature. The highest accuracy for each column is highlighted.

Table 5.7 presents the results of this ablation study. Learning the value of the curvature leads to a decrease in accuracy for the CUB and Stanford Cars datasets, while leading to an increase in accuracy for the FGVC-Aircraft dataset. However, during training on all datasets, the curvature value increases to the maximum value of 1.0 within the first 30 epochs and remains there for the rest of the training. With this information, it is safe to conclude that freezing the curvature would be more beneficial.

5.3.5 AdamW Optimizer

Optimizer	K-Means Algorithm	CUB All	CUB Old	CUB New	Stanford Cars All	Stanford Cars Old	Stanford Cars New	FGVC-Aircraft All	FGVC-Aircraft Old	FGVC-Aircraft New
SGD	Lorentz	**73.04**	**83.70**	62.47	**76.18**	**90.98**	**61.91**	**61.69**	66.13	**57.25**
AdamW	Lorentz	29.22	39.94	18.59	36.14	55.29	17.66	41.37	60.85	21.86
SGD	Euclidean	72.95	82.07	**63.92**	72.50	86.07	58.66	60.97	**70.32**	51.59
AdamW	Euclidean	30.69	40.19	21.27	34.29	46.63	22.38	42.09	48.98	35.20

Table 5.8: Accuracy of the Lorentz embeddings Vanilla GCD model when trained with a the SGD optimizer and the AdamW optimizer. The highest accuracy for each column is highlighted.

All models were optimized using the SGD optimizer. However, it is worth investigating their performance when optimized with another optimizer. Therefore, the models are trained with the AdamW optimizer to compare the performance of training between the two optimizers.

Table 5.8 presents the results of this ablation study. Using the AdamW optimizer leads to a decrease in accuracy by almost half for most of the datasets. This showcases the benefits of using the SGD for learning Lorentz embeddings in Vanilla GCD. However, it is worth investigating other optimizers or different hyperparameter setups to see if higher accuracy can be achieved.

Chapter 6

Future Work

This chapter lists some possible further work and analysis for the proposed methods in this thesis, along with some future directions for hyperbolic GCD.

Further Work

The main results of the Vanilla GCD model presented very low accuracy from using the Poincaré K-Means algorithm. This was hypothesized to be due to sensitivity to the initialization of the prototypes, where all samples were assigned to one prototype in the assignment phase of the algorithm. Therefore, further work is needed to pinpoint if this hypothesis is correct. One direction would be to adapt the BSSK algorithm to the Poincaré model, and evaluate if it resolves these issues.

Furthermore, the main results of the SimGCD model presented the superiority of the Euclidean model, which is in contrast to the results presented in the HypCD paper by Liu et al. [20]. Therefore, it is worth performing ablations on the different SimGCD models and further analyzing the models' outputs to be able to present a more robust conclusion. Another factor can also be the missing Riemannian optimizer, as in contrast to the HypCD paper, the setup in this thesis utilizes the normal SGD optimizer for all layers, while Liu et al. [20] choose to optimize the hyperbolic FC layer using Riemannian Adam.

Lastly, further analysis of the hyperbolic embeddings and clustering results could be carried out to analyze the hierarchical capabilities of the hyperbolic GCD methods. This analysis can be used to understand where the models fail and where they succeed, and also to closely analyze whether hyperbolic representation learning discovers hierarchies present in the data.

Future Directions

One future direction for hyperbolic GCD is to modify the SOTA Euclidean method, SelEx, to learn embeddings in the Lorentz model, and perform hierarchical K-Means in hyperbolic geometry by modifying the BSSK and the HSSK algorithms for the Poincaré and Lorentz models.

Furthermore, this thesis assumes that the GCD methods would implicitly learn hierarchies in hyperbolic geometry. Making it worth investigating explicitly using the hierarchical labels in the fine-grained SSB datasets to learn embeddings using entailment loss. For example, by representing variant labels as child nodes of the manufacturer labels in the FGVC-Aircraft dataset.

Chapter 7

Conclusion

This thesis focused on exploring the potential of utilizing hyperbolic visual learning for solving the Generalized Category Discovery (GCD) task. To this end, the GCD and hyperbolic visual learning literature were analyzed.

Motivated by the success of SelEx in utilizing hierarchical pseudo labels for solving GCD, and the results presented by Liu et al. [20] in their Hyperbolic Category Discovery (HypCD) paper, which was concurrently published with this thesis. It was chosen to analyze the effect of using the Lorentz model for hyperbolic geometry to learn representations of data, and the impact of performing K-Means clustering directly on hyperbolic embeddings.

To this end, the Vanilla GCD method by Vaze et al. [31] and the SimGCD method by Wen et al. [35] were adapted to learn representations in the Lorentz model. Furthermore, the K-Means algorithm was adapted for hyperbolic geometry for non-parametric clustering in Vanilla GCD, and hyperbolic Fully Connected (FC) layers were used for parametric clustering in SimGCD.

The adapted Vanilla GCD and SimGCD models were tested alongside the original and HypCD models. The results showed that utilizing the Lorentz model for Vanilla GCD does lead to increased performance in most datasets. However, it showed worse performance in comparison to all other models with SimGCD, while the original Euclidean SimGCD achieved the best performance, in contrast to the results reported by Liu et al. [20], showing the need for further experimentation.

Experiments were also run on the adapted Vanilla GCD to test the performance of performing non-parametric clustering directly on the hyperbolic embeddings using the hyperbolic K-Means algorithms. The results showed that embeddings in the Lorentz space benefit from the hyperbolic K-Means algorithm, achieving the best performance. However, the K-Means algorithm in Poincaré space was found to be too unstable, with the algorithm completely failing most of the time, depending on the prototype initialization.

Lastly, ablation studies were carried out on the Lorentz Vanilla GCD model. The conclusions from the five ablation studies were the following:

- Clipping Euclidean embeddings is not necessary for training in the Lorentz model.
- Using a combination of angle-based and distance-based contrastive learning is essential for hyperbolic learning.
- The embedding dimension of the Lorentz embeddings does not affect performance. However, further experimentation is needed for dimensions below 128.
- Learning the curvature instead of freezing it does not lead to improved performance for representation learning in the Lorentz space.
- The Stochastic Gradient Descent (SGD) is a much better fit for hyperbolic GCD than the AdamW optimizer.

Bibliography

[1] Gary Bécigneul and Octavian-Eugen Ganea. “Riemannian adaptive optimization methods”. In: *arXiv preprint arXiv:1810.00760* (2018).

[2] Mathilde Caron et al. “Emerging Properties in Self-Supervised Vision Transformers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 9650–9660.

[3] Weize Chen et al. “Fully hyperbolic neural networks”. In: *arXiv preprint arXiv:2105.14686* (2021).

[4] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[5] Karan Desai et al. “Hyperbolic Image-text Representations”. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 7694–7731. URL: https://proceedings.mlr.press/v202/desai23a.html.

[6] Octavian Ganea, Gary Becigneul, and Thomas Hofmann. “Hyperbolic Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/dbab2adc8f9d078009ee3fa810bea142-Paper.pdf.

[7] Songwei Ge et al. “Hyperbolic Contrastive Learning for Visual Representations Beyond Objects”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 6840–6849.

[8] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. “Recent Advances in Open Set Recognition: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2021), pp. 3614–3631. DOI: 10.1109/TPAMI.2020.2981604.

[9] Mina Ghadimi Atigh, Martin Keller-Ressel, and Pascal Mettes. "Hyperbolic Busemann Learning with Ideal Prototypes". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 103–115. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/01259a0cb2431834302abe2df60a1327-Paper.pdf.

[10] M. Gromov. "Hyperbolic Groups". In: *Essays in Group Theory*. Ed. by S. M. Gersten. New York, NY: Springer New York, 1987, pp. 75–263. ISBN: 978-1-4613-9586-7. DOI: 10.1007/978-1-4613-9586-7_3. URL: https://doi.org/10.1007/978-1-4613-9586-7_3.

[11] Yunhui Guo et al. "Clipped Hyperbolic Classifiers Are Super-Hyperbolic Classifiers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11–20.

[12] Joy Hsu et al. "Capturing implicit hierarchical structure in 3D biomedical images with self-supervised hyperbolic representations". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 5112–5123. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/291d43c696d8c3704cdbe0a72ade5f6c-Paper.pdf.

[13] Menglin Jia et al. "Visual Prompt Tuning". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 709–727. ISBN: 978-3-031-19827-4.

[14] Valentin Khrulkov et al. "Hyperbolic Image Embeddings". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[15] Max Kochurov, Rasul Karimov, and Serge Kozlukov. *Geoopt: Riemannian Optimization in PyTorch*. 2020. arXiv: 2005.02819 [cs.CG].

[16] Jonathan Krause et al. "3D Object Representations for Fine-Grained Categorization". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*. 2013.

[17] Dmitri Krioukov et al. "Hyperbolic geometry of complex networks". In: *Phys. Rev. E* 82 (3 2010), p. 036106. DOI: 10.1103/PhysRevE.82.036106. URL: https://link.aps.org/doi/10.1103/PhysRevE.82.036106.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[19] Marc Law et al. "Lorentzian Distance Learning for Hyperbolic Representations". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 3672–3681. URL: https://proceedings.mlr.press/v97/law19a.html.

[20] Yuanpei Liu, Zhenqi He, and Kai Han. "Hyperbolic Category Discovery". In: *arXiv preprint arXiv:2504.06120* (2025).

[21] Subhransu Maji et al. "Fine-grained visual classification of aircraft". In: *arXiv preprint arXiv:1306.5151* (2013).

[22] Yidan Mao et al. "Klein Model for Hyperbolic Neural Networks". In: *arXiv preprint arXiv:2410.16813* (2024).

[23] Pascal Mettes et al. "Hyperbolic deep learning in computer vision: A survey". In: *International Journal of Computer Vision* 132.9 (2024), pp. 3484–3508.

[24] Maximillian Nickel and Douwe Kiela. "Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3779–3788. URL: `https://proceedings.mlr.press/v80/nickel18a.html`.

[25] Wei Peng et al. "Hyperbolic Deep Neural Networks: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12 (2022), pp. 10023–10044. DOI: `10.1109/TPAMI.2021.3136921`.

[26] Sarah Rastegar et al. "SelEx: Self-expertise in Fine-Grained Generalized Category Discovery". In: *Computer Vision – ECCV 2024*. Ed. by Aleš Leonardis et al. Cham: Springer Nature Switzerland, 2025, pp. 440–458. ISBN: 978-3-031-72897-6.

[27] John G Ratcliffe, Sheldon Axler, and Kenneth A Ribet. *Foundations of hyperbolic manifolds*. Vol. 149. Springer, 1994.

[28] Walter J. Scheirer et al. "Toward Open Set Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.7 (2013), pp. 1757–1772. DOI: `10.1109/TPAMI.2012.256`.

[29] Ryohei Shimizu, Yusuke Mukuta, and Tatsuya Harada. "Hyperbolic neural networks++". In: *arXiv preprint arXiv:2006.08210* (2020).

[30] Sagar Vaze, Andrea Vedaldi, and Andrew Zisserman. "No Representation Rules Them All in Category Discovery". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 19962–19989. URL: `https://proceedings.neurips.cc/paper_files/paper/2023/file/3f52ab4322e967efd312c38a68d07f01-Paper-Conference.pdf`.

[31] Sagar Vaze et al. "Generalized Category Discovery". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 7492–7501.

[32] Sagar Vaze et al. "The Semantic Shift Benchmark". In: *ICML 2022 Shift Happens Workshop*. 2022. URL: `https://openreview.net/forum?id=2ql76f4zE3`.

[33] Catherine Wah et al. “The caltech-ucsd birds-200-2011 dataset”. In: (2011).

[34] Hongjun Wang, Sagar Vaze, and Kai Han. “Sptnet: An efficient alternative framework for generalized category discovery with spatial prompt tuning”. In: *arXiv preprint arXiv:2403.13684* (2024).

[35] Xin Wen, Bingchen Zhao, and Xiaojuan Qi. “Parametric Classification for Generalized Category Discovery: A Baseline Study”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 16590–16600.

[36] Yun Yue et al. “Hyperbolic contrastive learning”. In: *arXiv preprint arXiv:2302.01409* (2023).

[37] Bingchen Zhao, Xin Wen, and Kai Han. “Learning Semi-supervised Gaussian Mixture Models for Generalized Category Discovery”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 16623–16633.