

Non-Euclidean Geometries for Prototype-Based Image Classification: a Reality Check

Samuele Fonio^{1,†}, Silvia Grosso^{1,2,†} and Sara Bouchenak²

¹University of Turin – Dept. of Computer Science, Turin, Italy

²INSA Lyon – LIRIS, Lyon, France

Abstract

Identifying the most appropriate metric to capture similarities between embeddings remains a longstanding challenge in Machine Learning (ML). Recent research has highlighted the potential of non-Euclidean geometries for modeling embedding distances, particularly in datasets with latent hierarchical structures. However, selecting an optimal geometry is non-trivial, and the literature lacks a comprehensive analysis of the advantages and limitations associated with different metric spaces. In this paper, we aim to address this gap by focusing on a setting where the choice of geometry plays a crucial role in the optimization process: Prototype Learning (PL). Through extensive comparisons across a diverse range of datasets, we uncover key insights into the behavior of non-Euclidean spaces, showcasing their limitation and possible future developments.

Keywords

Prototype Learning, Metric Learning, Non-Euclidean Geometries

1. Introduction

Deep Learning (DL) has emerged as the main tool for detecting complex patterns and solving very challenging tasks in Machine Learning (ML), achieving its best results in Computer Vision (CV) and Natural Language Processing (NLP). It relies on Deep Neural Networks (DNNs), which are known to be very effective, but also considered as black boxes, since they usually lack interpretability and explainability by design.

In particular, DL relies on compressing data representations in a so-called *latent space*, and then using these representations to accomplish specific tasks. Especially for classification tasks, it is crucial to find the best way to detect similarities between embeddings, since drawing decision boundaries heavily depends on it.

Recently, non-Euclidean geometries have garnered much interest in the research community. Specifically, the key point of leveraging non-Euclidean geometries is to embed data onto specific manifolds equipped with different metric spaces, to better shape the underlying relationships between data. Among these, the hyperspherical and hyperbolic geometries showed promising research directions.

Hyperspherical geometries, which mainly involve normalization of the embeddings and the use of cosine similarity, have been well studied in contrastive learning and face recognition [1, 2]. This geometry has the main benefit of setting a bound on the distance magnitude (since cosine similarity is bounded) possibly reflecting the underlying geometry of the data.

Hyperbolic geometries were introduced to explicitly handle hierarchical data, such as text [3] and graphs [4]. In these cases, the underlying geometry of the data is the hyperbolic one, since the data are explicitly hierarchical and this geometry is able to embed hierarchical structures with arbitrarily low distortion [5]. However, recent advancements have shown the benefits of hyperbolic geometries also in CV tasks [6, 7, 8], where the hierarchy is somewhat hidden. While this information is crucial to justify the use of these geometries, it is often overlooked, showing the leading performance of non-Euclidean geometries without a clear explanation. On the other hand, it is possible to find CV tasks where a

ITADATA2025: The 4th Italian Conference on Big Data and Data Science, September 9–11, 2025, Turin, Italy

[†]These authors contributed equally.

✉ samuele.fonio@unito.it (S. Fonio); silvia.grosso@insa-lyon.fr (S. Grosso); sara.bouchenak@insa-lyon.fr (S. Bouchenak)

ORCID 0000-0002-0877-7063 (S. Fonio); 0009-0007-7697-1566 (S. Grosso); 0000-0003-0558-0123 (S. Bouchenak)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

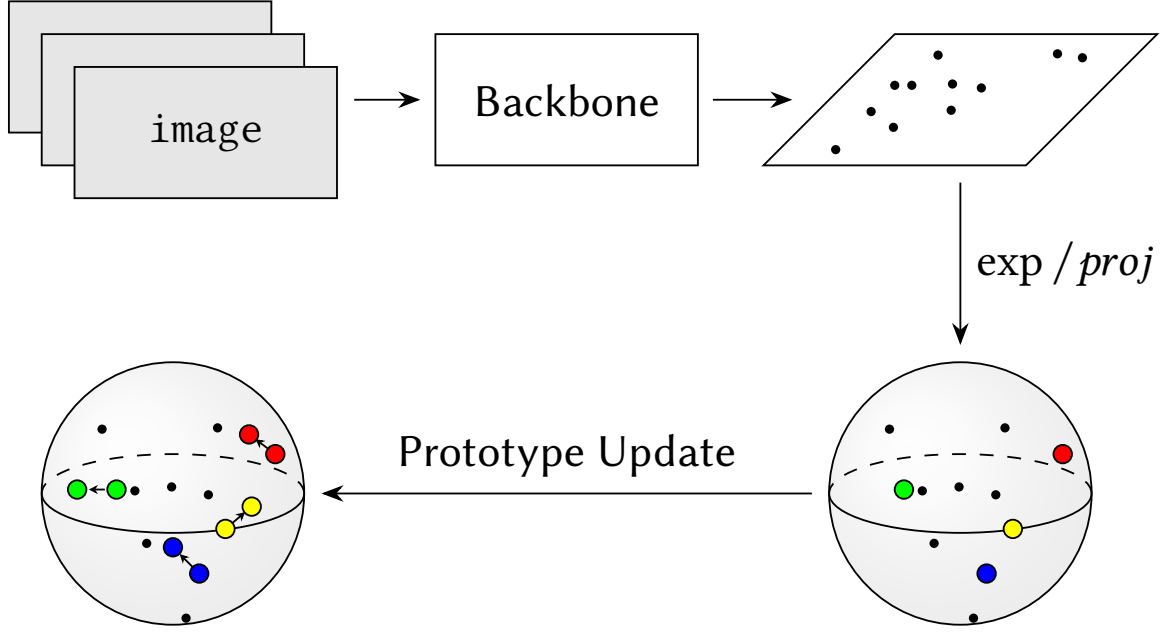


Figure 1: Illustration of Prototypical Networks on a generic manifold.

hierarchy is present but implicit. For example, in fine-grained image classification the labels are usually arranged according to a hierarchy, making hyperbolic spaces particularly suitable. Another example is remote sensing, where data can show specific hierarchical structures [9].

Many works have shown the discrepancy between Euclidean and hyperbolic performances, especially in few-shot learning [10], and also raised some concerns about the numerical stability of these spaces [11]. As a consequence, we aim to thoroughly investigate the impact of non-Euclidean geometries in a context where metric elements are crucial: Prototype Learning [12] (PL). PL was originally proposed as a simple yet efficient method for few-shot learning. The main idea behind PL is to learn a metric rather than a distribution by minimizing some metric elements (*e.g.*, a distance) between embeddings and their true class representation, *i.e.*, the *prototypes*. This approach led to good generalization properties, while leveraging pure metrics in the embedding space.

In this context, the geometry of the embedding space plays a crucial role. The employment of PL in image classification is often overlooked, but recent advancements introduce non-Euclidean geometries for Image classification [13, 7, 14], highlighting the potential for this paradigm to play an important role in this domain. In fact, PL allows for smooth integration of regularization terms directly in the embedding space, and is known to be more robust to Out-Of-Distribution (OOD) datasets [15]. However, the impact of the embedding geometry in this context has not been properly studied.

In this work, we want to shed light on this aspect, by considering PL for image classification in both a full-data setting (with a standard amount of data) and in few-data setting (with a small amount of data). To do so, we leverage a framework in which the prototypes are parametric (See Figure 1) and updated directly on the manifold. We thoroughly define the mathematical elements behind such optimization process and show some possible drawbacks for Hyperspherical and Poincaré geometries. In particular, since hyperbolic spaces are claimed to be more effective in robustness, we also evaluate the different geometries with Out-Of-Distribution Detection (common evaluation in PL for image classification [15]) and under Projected Gradient Descent (PGD) attacks. To provide a complete comparison also between hyperbolic geometries, we leverage for the first time the Lorentz geometry for Image classification in a PL setting. This is the first study comparing in a broad and fair manner all the geometries for PL in image classification.

To summarize, our contribution is threefold: *i)* we provide a comprehensive comparison of different geometries in PL for image classification; *ii)* we introduce Lorentzian prototypical networks for image

classification; *iii*) we conduct extensive experiments to validate whether or not the non-Euclidean geometries can benefit the learning process.

2. Related Works

2.1. Prototype Learning

Prototypical networks are the deep generalization of learning Vector Quantization machines [16] and nearest centroid classifiers [17].

In most of the approaches, the prototypes are defined as centroids of the representations [12], positioned a priori [13, 18, 8] or used as parameters and updated alongside the training [15, 19]. In particular, [15] highlights the importance and benefit of the latter choice, providing interesting results of parametric prototypes in terms of several metrics (*e.g.*, robustness and out-of-distribution detection). An additional benefit of using parametric prototypes is the possibility to add a regularization on the prototypes, rather than on the embeddings. For example, [19] shows the benefit of using parametric prototypes and adding hierarchical information to the learning process, provided a priori through a hierarchy of the labels.

2.2. Non-Euclidean Prototype Learning

Hyperspherical PL The work presented in [13] introduces a non-Euclidean geometry in PL, *i.e.*, the hyperspherical prototypical networks, keeping the prototypes fixed on the hypersphere and maximizing the cosine separation between them. The addition of hierarchical information is also investigated, by using a triplet-loss while positioning the prototypes. The importance of incorporating hierarchical information in this non-Euclidean context was further explored by [18]. It is worth mentioning that in these frameworks, the prototypes are kept fixed throughout the training, while our approach maintains the effort of updating the prototypes, by treating them as learnable parameters of the network. Similarly to our approach, [1] tackles the specific challenge of face recognition and exploits the utility of a normalization layer for the embeddings, leveraging the benefits of using the cosine similarity. Differently, we tackle a broader task with image classification, and enrich the comparison with other non-Euclidean geometries.

Hyperbolic PL Hyperbolic manifolds are claimed to represent hierarchical data with arbitrarily low distortion [5]. Regarding hyperbolic models, five different ones have been defined: the Poincaré ball, the Lorentz model (Hyperboloid), the Poincaré half-plane model, the hemisphere model, and the Beltrami-Klein disk. The two most popular are the Poincaré ball and the Lorentz model.

Among the works that operate mainly with the Poincaré ball there are [20, 6, 21, 8], covering various computer vision tasks such as few-shot learning, image classification, and action recognition. More specifically, some works highlighted that models built on hyperbolic spaces can outperform the state-of-the-art Euclidean counterparts [6, 9]. Among these, [6] gained particular relevance in the recent literature, as it presented the first hyperbolic prototypical framework. However, [10] showed that hyperbolic representations do not outperform well-structured Euclidean methods, raising some questions about the supposed advantages typically attributed to these spaces.

Our work can be considered a further exploration in this direction, setting up a benchmark for various non-Euclidean geometries in the context of image classification. In fact, few works have tackled the problem of exploiting hyperbolic geometries for image classification. [7] pioneered this approach, while [22] explored the impact of changing the temperature parameter in a contrastive loss when exploring a hyperbolic space. [7] introduces a method with fixed ideal prototypes positioned at the boundary of the Poincaré ball, which is conceptually at an infinite distance from the center of the hyperbolic space. A different approach is taken by [14], which showed that using hyperbolic entailment cones as metric elements is particularly beneficial, especially in scenarios with a large number of classes. However, in both these scenarios, the prototypes are fixed.

Several works have been based on the Lorentz model [23, 3], but Computer Vision is usually overlooked in the main literature. For example, [24] proposes a fully hyperbolic CNN based on Lorentz operation, achieving better results than the Euclidean version and the Poincaré version [21]. Some studies comparing the Poincaré ball and Lorentz models [11, 25] report that Lorentz operations are numerically more stable. Our work involves this comparison, but focuses more on the performance on which the hyperbolic spaces usually outperform the Euclidean ones: accuracy, robustness and OOD detection.

3. Background

In this section, we are going to provide the background related to each geometry: Euclidean, Hyper-spherical, Lorentz and Poincaré ball. Comparing different geometries implies comparing different manifolds, equipped with their own metric elements and operations. However, each manifold has its own definition, impacting both on the metric elements and on their usage. In the following, we are going to define the key elements to be used, and we are providing their formulation for each geometry.

Definition 1. A **manifold** \mathcal{M} of dimension n is a topological space such that each point's neighborhood can be locally approximated by the Euclidean space \mathbb{R}^n .

Definition 2. Given a point $x \in \mathcal{M}$, the **tangent space** $\mathcal{T}_x\mathcal{M}$ of \mathcal{M} at x is the n -dimensional vector-space, omeomorphic to \mathbb{R}^n , built as the first order approximation of \mathcal{M} around x .

Definition 3. The **Riemannian metric** is the metric tensor that gives a local notion of angle, length of curves, surface and volume. For a manifold \mathcal{M} , the Riemannian metric g_x is a smooth collection of inner products on the associated tangent space: $g_x : \mathcal{T}_x\mathcal{M} \times \mathcal{T}_x\mathcal{M} \rightarrow \mathbb{R}$. A **Riemannian manifold** is defined as a manifold equipped with a Riemannian metric g , and is written (\mathcal{M}, g) .

Definition 4. A geodesics γ is the shortest path between two points on the manifold. It can be seen as the generalization of the straight line in Euclidean spaces. Given $x, y \in \mathcal{M}$ the **distance** $d(x, y)$ is defined by measuring the length of the geodesic segment connecting the two points.

Definition 5. Given a point $x \in \mathcal{M}$ and a vector $v \in \mathcal{T}_x\mathcal{M}$, the **exponential map** projects v to the manifold \mathcal{M} , $\exp_x^{\mathcal{K}}(v) : \mathcal{T}_x\mathcal{M} \rightarrow \mathcal{M}$. The projection is obtained by moving the point along the geodesic $\gamma : [0, 1] \rightarrow \mathcal{M}$ uniquely defined by $\gamma(0) = x$ and $\gamma'(0) = v$. The projection is defined to be $\exp_x^{\mathcal{K}}(v) = \gamma(1)$. The precise definition of the exponential map depends on the manifold; its inverse function is called the **logarithmic map**, $\log_x^{\mathcal{K}}(\cdot)$. See Figure 2 to visualize the exponential map.

Definition 6. Given a point $z \in \mathbb{R}^n$ in the ambient space, the **projection onto the manifold** \mathcal{M} is the operation that maps z to the closest point on \mathcal{M} with respect to a given metric (typically the Euclidean metric), denoted by $\text{proj}_{\mathcal{M}}(z) : \mathbb{R}^n \rightarrow \mathcal{M}$.

Given these definitions, we can now describe the respective elements for each geometry.

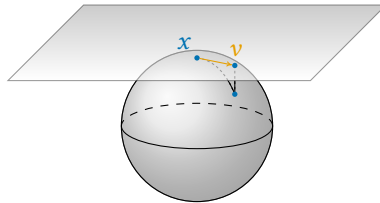


Figure 2: Illustration of the exponential map (Definition 5) applied onto a generic manifold \mathcal{M} (shown as a sphere in the picture).

Euclidean The Euclidean setting is the standard one, as the embedding space of a Neural Network is usually assumed to be equipped with this geometry. In this setting, there is no need of projection on the manifold, as we assume the embedding to lie in a Euclidean space. The distance used is the standard Euclidean distance:

$$d(x, y) = \|x - y\|.$$

It is worth mentioning that we did not use the squared norm as it yields low performances [19].

Hyperspherical The manifold in this case is defined as the hypersphere in n dimensions:

$$\mathbb{S}^{n-1} = \{x \in \mathbb{R}^n : \|x\|^2 = 1\}.$$

This manifold is usually equipped with the standard cosine distance:

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}.$$

Let $x \in \mathbb{S}^{n-1} \subset \mathbb{R}^n$ and $v \in \mathcal{T}_x \mathbb{S}^{n-1} = \{v \in \mathbb{R}^n : \langle v, x \rangle = 0\}$. The exponential map at x is given by:

$$\exp_x(v) = \begin{cases} \cos(\|v\|)x + \sin(\|v\|)\frac{v}{\|v\|}, & \text{if } v \neq 0, \\ x, & \text{if } v = 0. \end{cases}$$

Given a point $z \in \mathbb{R}^n \setminus \{0\}$, the projection onto the unit hypersphere \mathbb{S}^{n-1} is:

$$proj_{\mathbb{S}^{n-1}}(z) = \frac{z}{\|z\|}. \quad (1)$$

Lorentz The n -dimensional Lorentz model of hyperbolic space, denoted by \mathbb{H}^n , is defined as:

$$\mathbb{H}^n = \{x \in \mathbb{R}^{n+1} : \langle x, x \rangle_{\mathcal{L}} = -1, x_0 > 0\}, \quad (2)$$

where $\langle \cdot, \cdot \rangle_{\mathcal{L}}$ is the Lorentzian inner product \mathbb{R}^{n+1} is defined as:

$$\langle x, y \rangle_{\mathcal{L}} = -x_0 y_0 + \sum_{i=1}^n x_i y_i,$$

for any $x, y \in \mathbb{R}^{n+1}$.

The tangent space at a point $x \in \mathbb{H}^n$ is the subspace of \mathbb{R}^{n+1} given by:

$$\mathcal{T}_x \mathbb{H}^n = \{v \in \mathbb{R}^{n+1} : \langle v, x \rangle_{\mathcal{L}} = 0\}. \quad (3)$$

The Riemannian metric on \mathbb{H}^n is induced by the Lorentzian inner product restricted to the tangent space. For tangent vectors $u, v \in \mathcal{T}_x \mathbb{H}^n$, it is given by:

$$g_x(u, v) = \langle u, v \rangle_{\mathcal{L}}.$$

The distance between two points $x, y \in \mathbb{H}^n$ is given by:

$$d_{\mathbb{H}}(x, y) = \operatorname{arccosh}(-\langle x, y \rangle_{\mathcal{L}}).$$

Given a point $x \in \mathbb{H}^n$ and a tangent vector $v \in \mathcal{T}_x \mathbb{H}^n$, the exponential map is defined as:

$$\exp_x(v) = \cosh(\|v\|_{\mathcal{L}})x + \sinh(\|v\|_{\mathcal{L}})\frac{v}{\|v\|_{\mathcal{L}}},$$

where $\|v\|_{\mathcal{L}} = \sqrt{\langle v, v \rangle_{\mathcal{L}}}$.

Given a point $z \in \mathbb{R}^{n+1}$ such that $\langle z, z \rangle_{\mathcal{L}} < 0$ and $z_0 > 0$, the projection onto the hyperboloid is:

$$proj_{\mathbb{H}^n}(z) = \frac{z}{\sqrt{-\langle z, z \rangle_{\mathcal{L}}}}. \quad (4)$$

Poincaré ball The n -dimensional Poincaré ball model of hyperbolic space with curvature κ , denoted by \mathbb{B}^n , is the open unit ball in \mathbb{R}^n :

$$\mathbb{B}_\kappa^n = \{x \in \mathbb{R}^n : \kappa\|x\| < 1\}.$$

If not otherwise stated, we assume $\kappa = 1$. At a point $x \in \mathbb{B}^n$, the tangent space is identified with \mathbb{R}^n :

$$\mathcal{T}_x \mathbb{B}^n \cong \mathbb{R}^n.$$

The Riemannian metric on the Poincaré ball is conformally equivalent to the Euclidean metric, which means that it preserves angles but not lengths. It is defined as:

$$g_x(u, v) = \lambda_x^2 \langle u, v \rangle, \quad \text{where} \quad \lambda_x = \frac{2}{1 - \|x\|^2},$$

and $\langle \cdot, \cdot \rangle$ is the standard Euclidean inner product.

The distance between two points $x, y \in \mathbb{B}^n$ is given by:

$$d_{\mathbb{B}}(x, y) = \text{arcosh} \left(1 + 2 \frac{\|x - y\|^2}{(1 - \|x\|^2)(1 - \|y\|^2)} \right).$$

Given a point $x \in \mathbb{B}^n$ and a tangent vector $v \in \mathcal{T}_x \mathbb{B}^n \cong \mathbb{R}^n$, the exponential map is defined as:

$$\exp_x(v) = x \oplus \left(\tanh \left(\frac{\lambda_x \|v\|}{2} \right) \frac{v}{\|v\|} \right), \quad \text{for } v \neq 0,$$

where $\lambda_x = \frac{2}{1 - \|x\|^2}$ and \oplus denotes **Möbius addition**. For $v = \mathbf{0}$, we define $\exp_x(\mathbf{0}) = x$, while for $x = \mathbf{0}$:

$$\exp_{\mathbf{0}}(v) = \frac{1}{\sqrt{\kappa}} \tanh(\lambda_x \|v\|/2) \frac{v}{\|v\|}. \quad (5)$$

The **Möbius addition** addition of $x, y \in \mathbb{B}^n$ is defined as:

$$x \oplus y = \frac{(1 + 2\langle x, y \rangle + \|y\|^2)x + (1 - \|x\|^2)y}{1 + 2\langle x, y \rangle + \|x\|^2\|y\|^2}.$$

Given a point $z \in \mathbb{R}^n \setminus \mathbb{B}^n$, the projection onto the Poincaré ball is given by:

$$\text{proj}_{\mathbb{B}^n}(z) = \frac{z}{\max(1, \|z\| - \varepsilon)},$$

where $\varepsilon > 0$ is a small numerical tolerance to avoid projecting exactly onto the boundary $\|z\| = 1$.

4. Methodology

In this section, we are going to explain the methodology we employed.

4.1. Metric Learning

Standard PL relies on centroid prototypes, which means that the prototypes are defined as centroid of the representations of each class. Alternatively, [15] and [19] introduced the idea of extending deep networks to learn prototypes by embedding the prototype representations as network parameters.

Our methodology involves extracting the output from a backbone network, such as a ResNet18, and projecting it onto the manifold \mathcal{M} . Subsequently, distances from class prototypes are computed. Each geometry will have its own distance, that might be able or not to effectively represent the distances between embeddings. These distances are interpreted as a probability distribution using softmax activation, which is then employed in a cross-entropy loss function for learning purposes.

Formally, we assume to be given a dataset $X = \{(x_i, y_i)\}_{i=1}^N$, with x_i taking values in a sample space \mathcal{X} , $y_i \in \mathcal{C} = \{1 \dots C\}$, and $|X| = N$. A backbone network $f(\cdot, \theta) : \mathcal{X} \rightarrow \mathbb{R}^d$ is augmented with parameters $\Pi = \{\pi_j, j \in \mathcal{C}\}$ representing the prototypes, with $\pi_j \in \mathcal{M}$. We attach to the backbone a projection layer $h : \mathbb{R}^n \rightarrow \mathcal{M}$, which maps the embeddings onto the manifold, enabling the use of different metric elements, obtaining $z = g(x) = h \circ f(x, \theta)$.

We train a Prototypical Network, by solving:

$$\arg \min_{\theta, \Pi} \mathcal{L}(\theta, \Pi; \gamma),$$

i.e., finding the parameters θ and Π minimizing over the training set the distance based cross-entropy loss [15]:

$$\mathcal{L}(\theta, \Pi; \gamma) = \frac{1}{N} \sum_{(x_i, y_i) \in X} -\log \frac{e^{-d(z_i, \pi_{y_i})/\tau}}{\sum_{j \in \mathcal{C}} e^{-d(z_i, \pi_j)/\tau}}, \quad (6)$$

where τ is the *temperature* parameter [15], and $d(\cdot, \cdot)$ is the distance defined on the manifold.

The temperature is used to modulate the sharpness or smoothness of distances. Lower temperatures tend to sharpen distributions, leading to more confident, peaked outputs, while higher temperatures flatten distributions, allowing for greater exploration or uncertainty. This tuning plays a crucial role in model behavior, especially in non-Euclidean geometries such as hyperbolic or spherical spaces, where distances and similarity measures differ fundamentally from flat Euclidean space. In these geometries, the curvature affects how features are clustered and how separation boundaries form, making them more sensitive to the scale at which similarities are interpreted. Improper temperature tuning can distort learning dynamics by over- or under-emphasizing these curved-distance relationships, potentially degrading the quality of embeddings or classification performance. Therefore, understanding and adjusting the temperature parameter becomes even more critical when models operate in non-Euclidean latent spaces.

4.2. Projection layer

A key module of our methodology is the projection onto the manifold \mathcal{M} , i.e., $h : \mathbb{R}^n \rightarrow \mathcal{M}$. For the Euclidean geometry, we assume the embedding space to be already equipped with the Euclidean metrics and operations, and as a consequence, there is no need of explicit projection.

For Non-Euclidean geometries, we need to consider case by case. In particular, for the Poincaré ball, our projection is the exponential map (5), which is the standard adopted by the main works adopting the Poincaré ball [7, 8, 14]. However, this operation has an inherent assumption: $f(x, \theta) \in \mathcal{T}_0 \mathbb{B}^n$. This assumption is plausible for this geometry, while being incoherent for the hyperspherical and the Lorentz case for different reasons.

In particular, this is due to the role of the origin, for which $\mathbf{0} \in \mathbb{B}^n$, while $\mathbf{0} \notin \mathbb{S}^{n-1}$, which does not allow us to use the exponential map to project the points onto the hypersphere. It is however reasonable to use the projection (1).

For what it concerns the Lorentz geometry, we still cannot use the exponential math based in $\mathbf{0}$ since, again, the origin $\mathbf{0} \notin \mathbb{H}^n$ according to (2). As a consequence, we cannot consider the tangent plane to a point that is not on the manifold. For this geometry, as a projection layer we need to use (4).

4.3. Prototypes' update

It is important to notice that prototypes within this context possess a dual nature: they function as parameters of the architecture, while existing as entities within the embedding space. Computing the gradient involves operating within the parameter space, but it is instead crucial to maneuver the prototypes within the embedding space, for which the update rule must be coherent with the chosen geometry. Specifically, an operation that avoids moving the prototypes' from the manifold is needed. It is worth mentioning that this operation involves only the prototypes, while the parameter of the backbone θ can be updated by the standard SGD.

The standard SGD update rule assumes implicitly a Euclidean geometry:

$$\pi \leftarrow \pi - \mu \cdot \nabla_{\pi} \mathcal{L}, \quad (7)$$

where ∇_x is the standard Euclidean gradient with respect to the prototypes and μ is the learning rate.

On the other hand, if we operate on a Non-Euclidean manifold, the traditional SGD rule is no more valid. In particular, using Eq. (7) is not consistent with the manifold as it does not assure that $\pi \in \mathcal{M}$ after its update.

One possible solution for this is to apply SGD regardless of the geometry and then apply the projection layer on the obtained prototypes, ensuring that the prototypes lie on the manifold. However, this procedure violates the geometry of the embedding space.

In a more elegant way, it is possible to use a Riemannian version of SGD [26] (shown in Figure 3), which depends on the manifold. In particular, the Euclidean gradient is not constrained on the tangent space of the manifold. As a consequence, it is needed to project the calculated Euclidean gradient (which is efficient to be calculated) onto the tangent space of the manifold. Denoting $\nabla = \nabla_{\pi}$ if not stated otherwise, for the Hypersphere we have the following formulation of the Riemannian gradient:

$$\nabla^S \mathcal{L} = \nabla \mathcal{L} - \langle \nabla \mathcal{L}, \pi \rangle \pi. \quad (8)$$

For the Lorentz Model:

$$\nabla^H \mathcal{L} = \nabla \mathcal{L} + \langle \nabla \mathcal{L}, \pi \rangle \pi. \quad (9)$$

For the Poincaré ball:

$$\nabla^B \mathcal{L} = \frac{1}{\lambda_x^2} \nabla \mathcal{L}. \quad (10)$$

Once the gradient is projected onto the right space, we still need a proper update rule to actually move the prototypes. To accomplish this, we use RSGD [26] In its most general form:

$$\pi \leftarrow \exp_{\pi}(-\mu \cdot \nabla_{\pi}^{\mathcal{M}} \mathcal{L}), \quad (11)$$

where $\nabla_x^{\mathcal{M}}$ is the Riemannian gradient relative to manifold \mathcal{M} . The intuition about how the update rule operates is that the exponential map folds the gradient vector on the tangent space onto the Manifold, moving the prototype accordingly. In recent research, also the Riemannian version of other optimizers (e.g., ADAM) have also been studied [27], but, to the best of our knowledge, never in a PL setting, which actually represent a nice study case for this type of optimization.

Poincaré ball In the case of the Poincaré ball, it is important to notice that embeddings close to the boundary negatively impact the learning, causing possible vanishing gradients behaviors [20]. As a consequence, we use two techniques to prevent this scenario. Prototypes π_j are initialized by sampling randomly from $[-0.1, 0.1]$ and then projecting the sampled points onto the Poincaré ball via the exponential map. Secondly, we clip the features to be at most 1 before applying the exponential map.

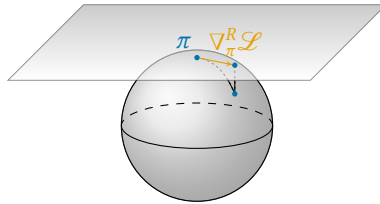


Figure 3: Illustration of the RSGD update rule.

5. Experiments and Results

5.1. Experimental setting

Datasets We compare the performances of the algorithms on four public datasets: *CIFAR-10* [28] 10 classes, 50000/10000 examples (train/test); *CIFAR-100* [28] 100 classes, 50000/10000 examples (train/test); *CUB* [29] 200 classes, 5994/5794 examples (train/test); *Aircraft* [30] 100 classes, 6667/3333 examples (train/test); The datasets were selected because they provide a fine-grained hierarchy over the classes (Aircraft, CUB) or because they show low δ -hyperbolicity (CIFAR-100, CIFAR-10) [6].

We consider two experimental settings: the *full-data setting*, where the full training set is used for model training, and the *few-data setting*, where we simulate low-data regimes by limiting the training set to only m examples per class, with $m \in \{5, 15, 30\}$, allowing us to analyze the scaling behavior of the different geometries under data scarcity.

Baselines We compared the embedding spaces equipped with the different geometries: Euclidean (ECL), Hyperspherical (HPS), Lorentz (LOR), Poincaré (POI). Each method is equipped with a ResNet18 [31] backbone network. It is worth mentioning that, due to the small sizes of the CIFAR datasets (images are 32×32 pixels) a standard procedure is to replace the first layer of the ResNet with a 3×3 kernel, rather than a 7×7 , and we employed this technique both in full-data setting and few-data setting. The training setting for the traditional scenario was SGD with a learning rate of 0.1, momentum of 0.9, and weight decay of 0.0005, for 200 epochs. Additionally, we used a learning rate scheduler which divided the learning rate by 5 at epoch 60, 120, 160. In case of non-Euclidean geometry (except for Hyperspherical), we used RSGD [26], with the same hyperparameter setting. We used geoopt [32] to implement the non-Euclidean operations. To test the best model, we kept a validation set extracted from the training set (10%) and picked the best model according to the validation accuracy. We employed an early stopping with a patience of 10 steps activated after the third learning rate scheduler’s step (epoch 160). Further details about the optimization are provided in the discussion.

In the few-data setting, due to the lack of data, we employed a finetuning setting on a pretrained ResNet18 backbone, using RADAM [27] with learning rate 0.0003, weight decay 0.0003 and generally small batch size for the different dataset, since the number of the examples is impacted by the number of classes. Specifically, we picked batch size 2, 2, 4 for CIFAR-10 for respectively m equal to 5, 15, 30; batch size 8, 16, 16 for CIFAR-100 and Aircraft, and batch size 16 and 32 for CUB with m equal to 5 and 15, while 30 was not possible due to lack of examples.

For the temperature, we performed an hyperparameter tuning, which is discussed in the Results subsection 5.2. In particular, if not otherwise stated we picked $\tau = 1$ for the Euclidean geometry, except for CIFAR-10, for which we used $\tau = 0.1$. For the Hyperspherical geometry we used $\tau = 0.1$. For Lorentz we employed $\tau = 0.1$ for all the datasets except CIFAR-10, for which we used $\tau = 1$. Finally, for Poincaré, we used $\tau = 0.1$ for CIFAR-10 and CIFAR-100, and $\tau = 0.02$ for Aircraft and CUB.

The embedding dimension was set equal to the number of classes for each dataset. Hyperbolic embeddings have shown good performances in low dimensional learning [7], but we keep this investigation for future works.

The experiments were run on HPC4AI [33] on a cluster with 4 nodes, each having 2 CPU - 2x Intel® Xeon® Processor E5-2680 v3, 12 core 2.1Ghz, RAM - 128GB/2133 (8 x 16Gb) DDR4, DISK - 1 x SSD , 800GB sas 6 Gbps 2.5, NET - IB 56Gb + 2x10Gb, GPU - 2 x NVIDIA T4 (Tesla) su PCI-E Gen3 x16.

The code is publicly available at this [link](#).

Metrics Our evaluation employed a range of metrics, including accuracy, robustness, and out-of-distribution (OOD) detection. Notably, all datasets used are balanced in class distribution, making standard accuracy a reliable and meaningful metric for performance assessment. For what it concerns the Robustness, we provide the results under PGD attacks [34] with $\epsilon \in 0, 0.8/255, 1.6/255, 3.2/255$, similar to [21]. The OOD detection metric is calculated as the gap between the confidence on the trained dataset and the confidence on an OOD dataset, similarly to [15]. In particular, if we denote with

$\hat{y}(x) = \max_{c \in \{0, \dots, C\}} \sigma(g(x, \theta))$, where C is the number of classes, g is the network as previously defined and σ is the SoftMax function, the OOD is calculated as:

$$OOD(f) = \frac{1}{|\mathcal{D}_{in}|} \sum_{x \in \mathcal{D}_{in}} \hat{y}(x) - \frac{1}{|\mathcal{D}_{out}|} \sum_{x \in \mathcal{D}_{out}} \hat{y}(x), \quad (12)$$

where \mathcal{D}_{in} is the in-distribution dataset (the one on which the model was trained) and \mathcal{D}_{out} is the OOD dataset.

Ideally, we would like a model to have high confidence on its training data, while showing low confidence on OOD samples. Due to a minor modification in the first convolutional layer of the network, necessary to accommodate the smaller image size of 32×32 used in CIFAR-10 and CIFAR-100, we restrict OOD evaluations to compatible architectures. Specifically, we use CIFAR-10 as the OOD dataset for models trained on CIFAR-100 (and vice versa), and CUB as the OOD dataset for models trained on Aircraft (and vice versa), since the latter pair uses the standard ResNet18 without architectural changes.

5.2. Results

In this section, we are going to comment on the results we obtained by comparing the different geometries.

Table 1

Average test accuracy and OOD detection over 3 runs with their standard deviation in full-data setting. In bold we indicate the best result. HPS* indicates the use of the retraction-based SGD optimizer.

Accuracy				
	CIFAR-10	CIFAR-100	Aircraft	CUB
ECL	94.76 \pm 0.16	75.11 \pm 0.17	75.90 \pm 0.63	48.59 \pm 0.25
HPS	93.59 \pm 0.15	72.88 \pm 0.07	64.95 \pm 0.43	35.07 \pm 0.69
HPS*	94.68 \pm 0.1	74.09 \pm 0.13	74.42 \pm 1.0	39.12 \pm 0.65
LOR	93.69 \pm 0.06	72.83 \pm 0.18	64.42 \pm 0.1	39.65 \pm 0.33
POI	94.59 \pm 0.17	75.69 \pm 0.30	70.63 \pm 1.65	40.64 \pm 0.99
OOD				
	CIFAR-10	CIFAR-100	Aircraft	CUB
ECL	0.14 \pm 0.01	0.31 \pm 0.00	0.54 \pm 0.02	0.10 \pm 0.0
HPS	0.10 \pm 0.0	0.19 \pm 0.0	0.28 \pm 0.02	0.11 \pm 0.01
HPS*	0.10 \pm 0.00	0.19 \pm 0.00	0.34 \pm 0.02	0.14 \pm 0.01
LOR	0.14 \pm 0.0	0.27 \pm 0.01	0.33 \pm 0.01	0.19 \pm 0.01
POI	0.13 \pm 0.00	0.24 \pm 0.00	0.20 \pm 0.02	0.11 \pm 0.01

Full-Data Setting Our comparisons cover a diverse set of datasets, providing insights across different number of classes and types of data. In particular, CIFAR-10 and CIFAR-100 are well-studied benchmark datasets in CV tasks. They have also been used in works that leverage hierarchical information [19], but their latent hierarchy lacks formal structure [24]. CIFAR-10 is primarily used for experiments involving a small number of classes, rather than for its hierarchical structure. Table 1 shows the image classification performance across the different geometries in the full-data setting. We can observe that in the full-data setting, the Euclidean geometry achieves slightly better performances for CIFAR-10, while the Poincaré geometry achieves better performances for CIFAR-100, in terms of accuracy. This is in line with other works that show hyperbolic spaces achieving good results with a large number of classes [14].

On the other hand, Aircraft and CUB are usually employed to benchmark fine-grained image classification tasks, as their labels follow an explicit hierarchy. In particular, CUB serves as an interesting testbed due to its large number of classes (200). We can see from Table 1 that the Euclidean geometry achieves remarkable performances on both datasets. This likely suggests that, without explicit hierarchical regularization, non-Euclidean geometries fail to leverage the underlying structure, resulting in lower overall accuracy. As for the comparison between non-Euclidean geometries, Poincaré achieves good results on CIFAR-100 and CUB, while Lorentz geometry demonstrates promising performance in OOD detection.

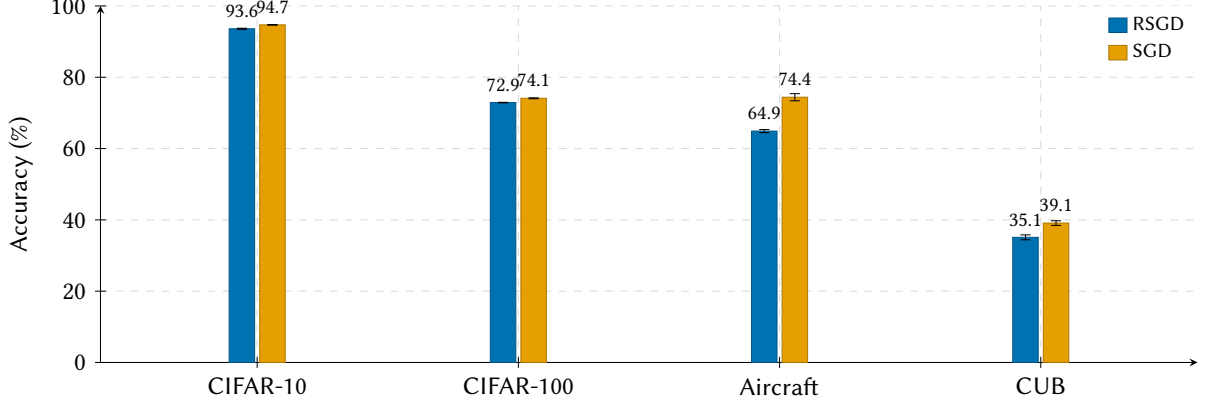
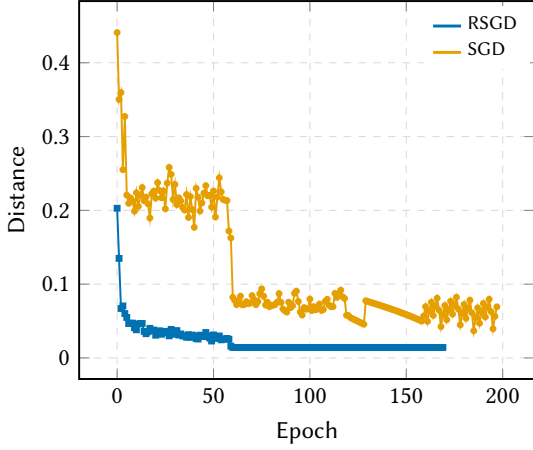


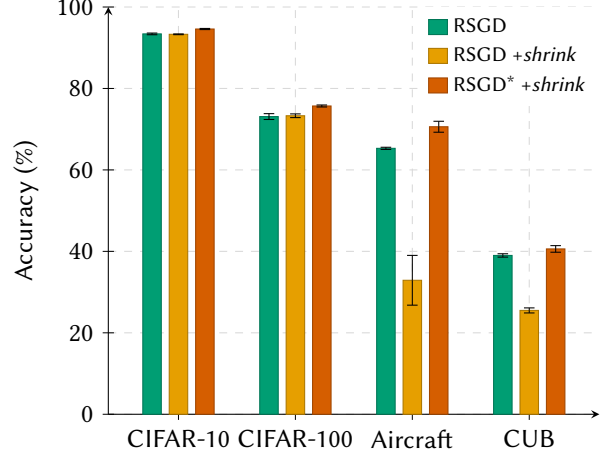
Figure 4: Comparison between RSGD and SGD optimizers in Hyperspherical geometry.

Ablation studies Since the prototypes play a crucial role, we believe it is important to take a deeper look into their optimization process. In particular, for the Hyperspherical geometry, we conducted experiments using both the Riemannian optimizer RSGD (11), which performs gradient steps via the exponential map on the manifold, and the standard SGD optimizer. In the latter case, we adopt a common alternative that replaces the exponential map with a retraction, a smooth mapping $R : \mathcal{T}_\pi \mathcal{M} \rightarrow \mathcal{M}$ that provides a first-order approximation of the exponential map. In this case, the retraction is the projection (1). As shown in Figure 4, in our experiments, the retraction-based SGD consistently outperforms RSGD across all four datasets, achieving up to a 10% improvement on Aircraft. This performance gap offers an insightful comparison between the two optimization strategies, particularly highlighting the importance of properly updating the prototypes, as discussed in detail in subsection 4.3. Although the two methods share the same step size, the key difference lies in the operation used to project the gradient back onto the manifold. While this may appear to be a minor variation, we observed that the prototypes tended to move more when using the retraction, potentially enabling a more flexible adaptation of the data distribution in the embedding space. In Figure 5a we show the average distance of consequent prototypes at different epochs. We clearly see a decreasing trend, common to every geometry, but we also observe that with RSGD the prototypes converge faster, while SGD shows higher values until the end, suggesting that the prototypes are still moving while the model has converged. Our evaluation provide interesting insights on the practical impact of this choice; however, further investigation is required to draw more definitive conclusions.

As a further investigation, we examined more closely the effect of using a separate optimizer for the prototypes across all geometries. In particular, in our experiments, we usually employed RSGD with the same parameters used for the neural network (e.g., learning rate 0.1). However, the prototypes may require more careful updates. Therefore, we also tried a separate optimizer (RSGD) with smaller magnitude of the updates (learning rate 0.001, weight decay 0.0001). This investigation resulted in better performance only for the Euclidean geometry applied to the CUB dataset and for the Poincaré geometry across all datasets, suggesting that the optimization in the case of the Poincaré ball requires



(a) Cosine distance between prototypes of consequent epochs.



(b) Impact of shrink initialization and separate prototype optimization in Poincaré geometry.

Figure 5: Ablation study on non-Euclidean methods.

careful design and tuning.

For this reason, as mentioned in subsection 4.3, we also evaluated the the impact of shrinking the initialization of the prototypes for the Poincaré ball. In Figure 5b, we present our ablation study, showing the effectiveness of shrinking the prototypes and using a separate optimizer in the case of the Poincaré geometry. Interestingly, the shrinkage initialization affects the datasets differently: it has a positive impact on CIFAR-10 and CIFAR-100, but a negative one on Aircraft and CUB. In contrast, the additional separate optimization step (denoted as *RSGD**) consistently yields the best performance across all datasets. This further highlights the sensitive and crucial role of prototype optimization in non-Euclidean geometries. The results shown in Table 1 for Poincaré include shrinking initialization and a separate optimizer for the prototypes.

To conclude our discussion on the key factors affecting the learning of Euclidean and non-Euclidean geometries, we analyzed the impact of temperature scaling across the four geometries and the different datasets. This parameter is particularly crucial for non-Euclidean geometries, where distances can grow exponentially [20]. We performed a fine-tuning procedure over four temperature values, $\tau \in \{1, 0.1, 0.05, 0.01\}$, evaluating the behavior of each geometry across all datasets.

As an illustrative example, we report in Figure 7 the distinct impact of temperature on the four geometries for the CUB dataset. Overall, we observe significant instability at $\tau = 0.01$, and an interesting opposite behavior between Euclidean and non-Euclidean geometries, especially pronounced for the Hyperspherical geometry. Except for CIFAR-10, where results remain relatively robust across diverse temperatures, for all other datasets, lowering the temperature leads to worse performance in Euclidean space, while Lorentz, Poincaré, and Hyperspherical geometries tend to benefit from sharper softmax outputs.

Robustness As previously stated, we have also tested the geometries in terms of robustness, with respect to OOD and adversarial robustness. In Table 1, we report the global average performance in terms of OOD detection.

On CIFAR-10 and CUB the results in terms of OOD detection show limited performance, despite the fact that they represent very different multiclass classification tasks, with 10 and 200 classes respectively. Interestingly, when comparing datasets with the same number of classes (e.g., Aircraft and CIFAR-100, both with 100 classes), we can notice that the highest OOD performance are on Aircraft, showcasing a good confidence gap on CUB. This suggests that the quality of OOD detection does not depend solely on the number of classes, and thus on the overall complexity of the classification task, but is also strongly influenced by the intrinsic nature of the data, such as the type of images, visual characteristics,

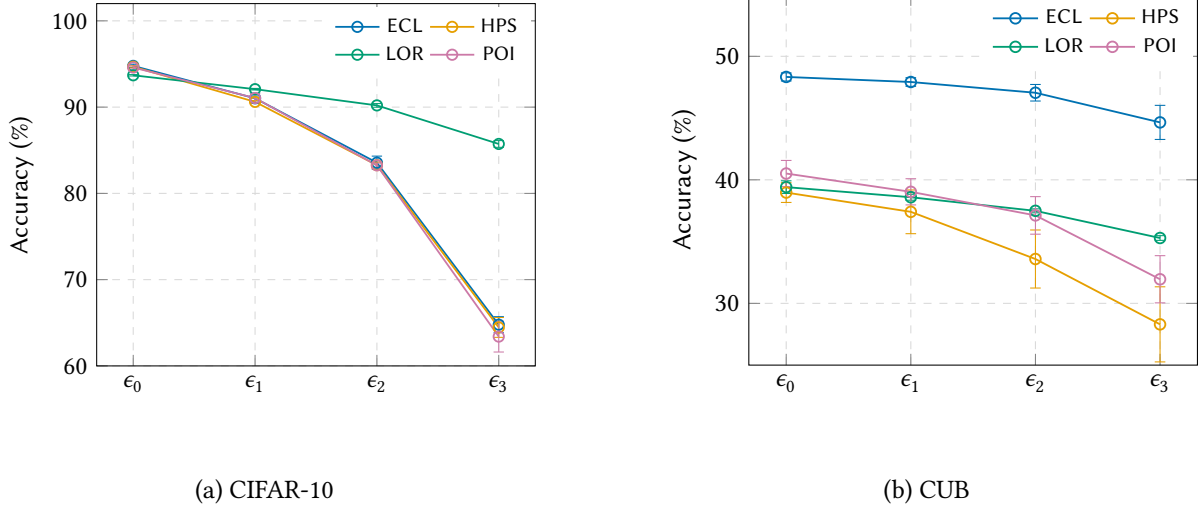


Figure 6: PGD robustness across different geometries (full-data setting).

intra-class complexity, and other structural factors.

Regarding the comparison between geometries, Hyperbolic geometries are usually indicated as strong OOD detectors [21]. However, this does not seem to be the case for CIFAR-100 and Aircraft, while the leading performance of Lorentz on CIFAR-10 and CUB, as well as generally among non-Euclidean geometries, is particularly interesting. Since this is the first study introducing the Lorentz geometry as a competitor, this represents a noteworthy result. The magnitude of Lorentzian distances, combined with the higher numerical stability of the Lorentz geometry, likely enables better performance in terms of OOD detection.

An additional metric used to compare the different geometries is adversarial robustness, *i.e.*, a model’s ability to maintain its performance even when subjected to adversarial examples. Adversarial examples are inputs that have been slightly perturbed in ways that are often imperceptible to humans but can lead the model to make incorrect predictions. The *Projected Gradient Descent* (PGD) attack is one of the most common methods for generating adversarial examples and evaluating the robustness of machine learning model. PGD is widely recognized as a strong attack; if a model is robust against PGD, it is generally considered robust against other types of adversarial attacks as well [34]. Figure 6a and Figure 6b illustrate the varying degrees of robustness across geometries on CIFAR-10 and CUB. The different values ϵ_i on the x -axis represent the increasing magnitude of the perturbation applied to the input data. The smaller the impact on model accuracy as the perturbation increases, the greater the robustness of the model. We can see from Figure 6a that the Lorentz geometry clearly outperforms all other geometries on CIFAR-10, while achieving comparable results in CUB (see Figure 6b). This shows that the Lorentz geometry is a possible robust competitor with respect to the Euclidean geometry when the number of classes is low. Further investigations will be conducted in future works to assess the properties offered by this geometry.

2. Few-Data Setting Table 2 presents the classification performance across different geometries in the few-data setting, considering 5, 15, and 30 training examples per class. For consistency, we retained the optimal temperature values determined in the full-data setting for each geometry and dataset. Additionally, we preserved shrink initialization toward the origin for the experiments using the Poincaré geometry. Given the increased difficulty of the few-data scenario, we employed a ResNet18 model pretrained on ImageNet, fine-tuning the entire network.

Even under this challenging setup, the Euclidean geometry continues to outperform the other geometries in nearly all scenarios. Overall, CIFAR-10 and CIFAR-100 show comparable performance across geometries for all the three few-data settings. In contrast, for the more fine-grained datasets Aircraft and CUB, especially in the most constrained settings with only 5 or 15 examples per class,

non-Euclidean geometries, particularly Lorentz and Poincaré, suffer from a more pronounced drop in accuracy.

Training with a small amount of data is, of course, a more challenging scenario. In this context, the magnitude of the non-Euclidean geometries might help by sharpening the data distributions, adding stronger penalties, and potentially improving generalization. It is worth mentioning that our setting is different from few-shot learning, although it leads to similar conclusions. In particular, we adopted the setting from [18], and we can observe that the Euclidean geometry outperforms the other geometries. As shown in Table 2, performance increases with the number of examples, as expected. In this context, the number of classes is particularly crucial, as it directly affects the total number of examples. In fact, performance on CUB remains comparable to the full dataset, and even improves when using 15 examples per class, thanks to the fine-tuning of the network on ImageNet. The only case where the best geometry is not the Euclidean one is with CIFAR-10, with 15 examples per class. However, the gain in performance is modest.

Regarding OOD detection, we can see that on CIFAR-10 all the geometries show poor performances, with very high confidence observed in both cases. This suggests that, since the model has been trained on few data and that the number of classes is smaller than the entire dataset, it tends to predict OOD samples with high confidence. However, for this dataset the Hyperspherical geometry shows leading performances. For the other datasets, with 5 examples per class the leading method is usually the Euclidean one, paired with a non-Euclidean one. In particular, for CIFAR-100 Euclidean and Poincaré perform the same, while on Aircraft Euclidean, Hyperspherical and Lorentz perform the best. Finally, on CUB, the Lorentz geometry achieves the best results. This is an interesting behavior, but the performance gains are always very modest. Among the non-Euclidean geometries, Lorentz consistently performs

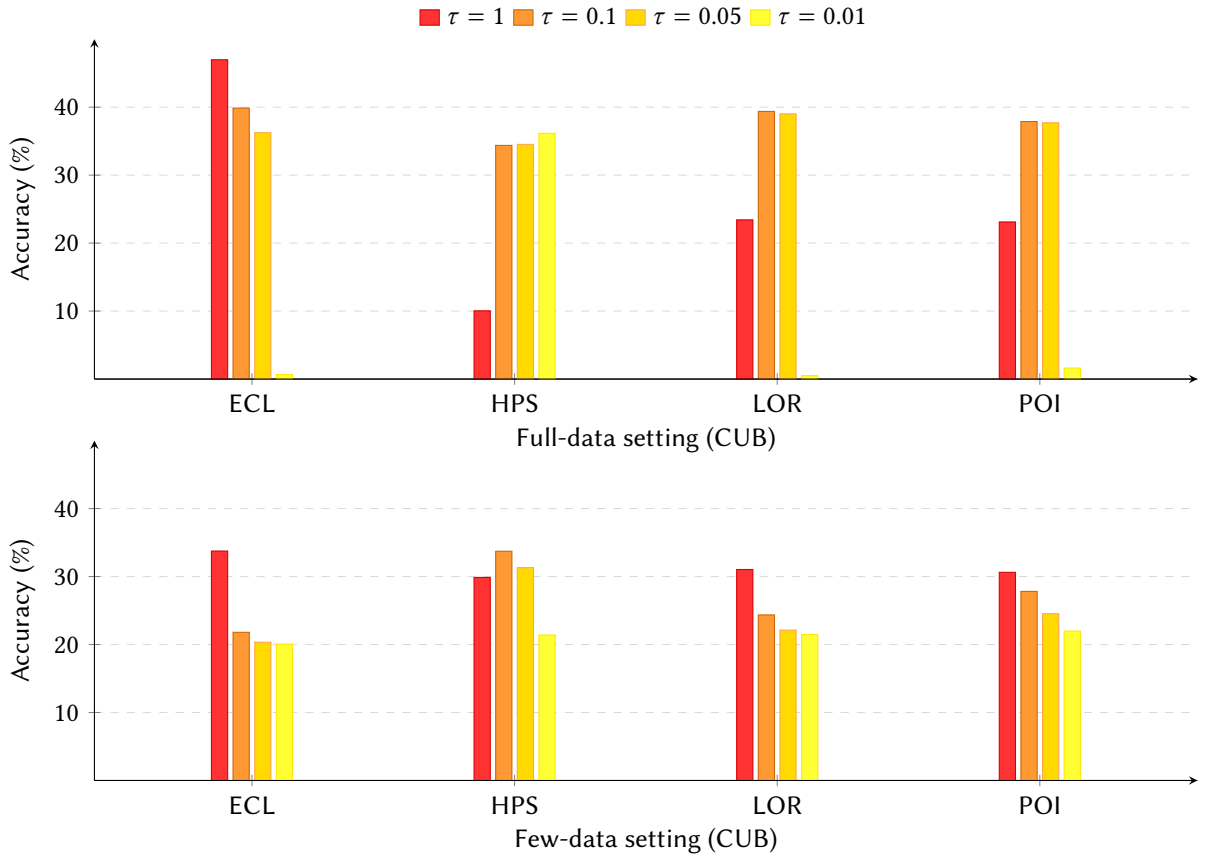


Figure 7: Impact of temperature scaling on Accuracy across geometries for CUB. The upper plot reports results in the full-data setting, while the lower shows the few-data setting, with 5 examples per class.

Table 2

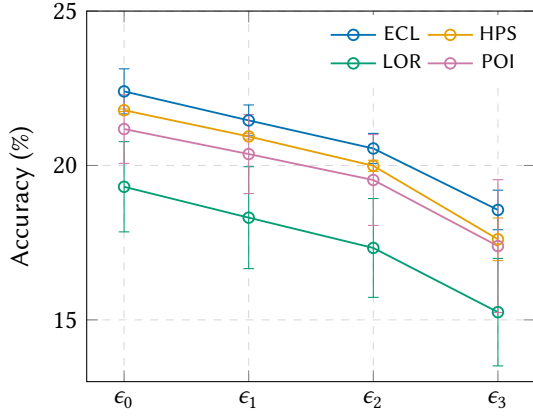
Average test accuracy and OOD detection over 3 runs with their standard deviation in few-data setting. In bold we indicate the best result.

Accuracy						
m	CIFAR-10			CIFAR-100		
	5	15	30	5	15	30
ECL	32.32 \pm 1.07	48.12 \pm 0.52	61.05 \pm 0.36	22.4 \pm 0.59	37.66 \pm 0.29	47.96 \pm 0.43
HPS	30.08 \pm 0.9	48.90 \pm 1.52	60.89 \pm 0.97	21.79 \pm 0.03	36.27 \pm 0.21	46.03 \pm 0.61
LOR	27.25 \pm 3.6	48.85 \pm 3.41	58.42 \pm 1.05	19.31 \pm 1.19	35.36 \pm 0.69	46.77 \pm 0.54
POI	31.56 \pm 0.77	49.14 \pm 1.01	60.11 \pm 0.98	21.18 \pm 0.90	35.74 \pm 0.87	46.0 \pm 0.90
m	Aircraft			CUB		
	5	15	30	5	15	30
ECL	30.31 \pm 0.79	59.49 \pm 1.51	71.69 \pm 0.74	35.04 \pm 1.5	61.9 \pm 0.54	–
HPS	28.42 \pm 0.91	56.43 \pm 1.84	67.63 \pm 1.51	34.09 \pm 1.79	61.7 \pm 1.33	–
LOR	24.36 \pm 0.98	53.11 \pm 0.95	69.92 \pm 0.94	26.97 \pm 0.68	55.48 \pm 0.57	–
POI	22.26 \pm 1.28	49.19 \pm 0.61	68.67 \pm 0.51	22.84 \pm 0.68	50.47 \pm 1.05	–
OOD						
m	CIFAR-10			CIFAR-100		
	5	15	30	5	15	30
ECL	0.01 \pm 0.01	0.05 \pm 0.01	0.08 \pm 0.0	0.08 \pm 0.0	0.17 \pm 0.01	0.19 \pm 0.02
HPS	0.02 \pm 0.01	0.07 \pm 0.01	0.08 \pm 0.01	0.07 \pm 0.01	0.10 \pm 0.01	0.11 \pm 0.02
LOR	0.0 \pm 0.0	0.03 \pm 0.01	0.06 \pm 0.01	0.07 \pm 0.0	0.12 \pm 0.0	0.14 \pm 0.0
POI	0.02 \pm 0.01	0.06 \pm 0.01	0.07 \pm 0.01	0.08 \pm 0.01	0.11 \pm 0.02	0.12 \pm 0.01
m	Aircraft			CUB		
	5	15	30	5	15	30
ECL	0.15 \pm 0.02	0.40 \pm 0.02	0.58 \pm 0.04	0.09 \pm 0.02	0.35 \pm 0.04	–
HPS	0.15 \pm 0.05	0.28 \pm 0.05	0.42 \pm 0.06	0.13 \pm 0.05	0.21 \pm 0.05	–
LOR	0.15 \pm 0.06	0.31 \pm 0.02	0.50 \pm 0.03	0.16 \pm 0.04	0.34 \pm 0.02	–
POI	0.08 \pm 0.01	0.18 \pm 0.02	0.21 \pm 0.0	0.05 \pm 0.04	0.15 \pm 0.01	–

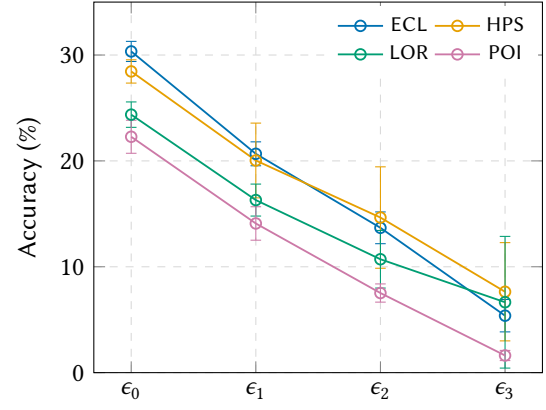
the best.

We also analyzed the robustness of models trained in the few-data setting with only 5 examples per class. No significant differences in robustness were observed across the four geometries. However, an interesting pattern emerges when comparing the robustness of models on CIFAR-100 and Aircraft datasets, as shown in Figure 8a and Figure 8b. Despite the fact that all four geometries achieve significantly higher classification accuracy on Aircraft compared to CIFAR-100, the models trained on CIFAR-100 are noticeably more robust. In contrast, the accuracy on Aircraft drops below 10% under adversarial perturbations for all geometries. This is likely due to the nature of PGD attacks, which operate at the pixel level. On small, low-resolution images like those in CIFAR-100, even relatively strong perturbations affect fewer global visual features. On high-resolution images like those in Aircraft, the same perturbations are more spatially concentrated and thus more visually disruptive, leading to a much greater degradation in performance.

Finally, we analyzed the impact of different temperature values in the few-data setting, focusing on the most challenging scenario with only 5 examples per class. As shown in Figure 7, we observe a similar trend as in the full-data setting: Euclidean geometry behaves inversely compared to non-Euclidean ones, with the Hyperspherical geometry showing the most pronounced contrast. However, differently from the full-data setting, Lorentz and Poincaré show a decreasing trend in decreasing the temperature. This further highlights the importance of properly fine-tuning this parameter.



(a) CIFAR-100



(b) Aircraft

Figure 8: PGD robustness across different geometries (few-data setting with 5 examples per class).

6. Conclusion

In this study, we conducted a thorough evaluation of four different geometries within a PL framework for image classification, considering both standard and few-data settings. While non-Euclidean geometries demonstrated competitive performance in few scenarios, our results suggest that standard optimization techniques leveraging Euclidean geometry continue to represent the state-of-the-art in the settings we explored.

These findings align with [10], who observed that Poincaré embeddings offered only modest gains in the few-shot learning setting, and that strong results could still be achieved using conventional Euclidean approaches.

Our work extends this conclusion by evaluating a broader range of geometries and experimental settings. We find that non-Euclidean methods often require carefully tailored algorithms and data with specific latent structures to outperform Euclidean methods. This study enriches the comparative landscape of geometric methods and highlights the limitations and potential of each approach. In doing so, it provides a solid testbed for developing future non-Euclidean techniques that can surpass current Euclidean-based models.

We believe that uncovering and leveraging the latent structure of data is crucial to achieving high performance with minimal supervision. Continued exploration in this direction is essential, and we hope our results stimulate further interest in the research community.

As future work, there are several promising directions. We are considering adding a hierarchical prior to investigate whether an explicitly hierarchical arrangement of the embedding space allows hyperbolic methods to achieve better performance. Another direction we plan to explore is expanding out evaluation to few-shot learning and to other prototypical methods such as ProtoPNETs, which have shown promise not only in terms of performance but also in model interpretability.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication’s content.

References

- [1] F. Wang, X. Xiang, J. Cheng, A. L. Yuille, Normface: L2 hypersphere embedding for face verification, in: Proceedings of the 25th ACM international conference on Multimedia, 2017, pp. 1041–1049.
- [2] J. Deng, J. Guo, N. Xue, S. Zafeiriou, Arcface: Additive angular margin loss for deep face recognition, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4690–4699.
- [3] M. Nickel, D. Kiela, Learning continuous hierarchies in the lorentz model of hyperbolic geometry, in: International conference on machine learning, PMLR, 2018, pp. 3779–3788.
- [4] O. Ganea, G. Bécigneul, T. Hofmann, Hyperbolic neural networks, Advances in neural information processing systems 31 (2018).
- [5] F. Sala, C. De Sa, A. Gu, C. Ré, Representation tradeoffs for hyperbolic embeddings, in: International conference on machine learning, PMLR, 2018, pp. 4460–4469.
- [6] V. Khruikov, L. Mirvakhabova, E. Ustinova, I. Oseledets, V. Lempitsky, Hyperbolic image embeddings, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 6418–6428.
- [7] M. Ghadimi Atigh, M. Keller-Ressel, P. Mettes, Hyperbolic busermann learning with ideal prototypes, Advances in Neural Information Processing Systems 34 (2021) 103–115.
- [8] T. Long, P. Mettes, H. T. Shen, C. G. Snoek, Searching for actions on the hyperbole, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1141–1150.
- [9] M. Hamzaoui, L. Chapel, M.-T. Pham, S. Lefèvre, Hyperbolic prototypical network for few shot remote sensing scene classification, Pattern Recognition Letters 177 (2024) 151–156.
- [10] G. Moreira, M. Marques, J. P. Costeira, A. Hauptmann, Hyperbolic vs euclidean embeddings in few-shot learning: Two sides of the same coin, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 2082–2090.
- [11] G. Mishne, Z. Wan, Y. Wang, S. Yang, The numerical stability of hyperbolic representation learning, in: International Conference on Machine Learning, PMLR, 2023, pp. 24925–24949.
- [12] J. Snell, K. Swersky, R. Zemel, Prototypical networks for few-shot learning, Advances in neural information processing systems 30 (2017).
- [13] P. Mettes, E. Van der Pol, C. Snoek, Hyperspherical prototype networks, Advances in neural information processing systems 32 (2019).
- [14] S. Fonio, R. Esposito, M. Aldinucci, Hyperbolic prototypical entailment cones for image classification, in: Y. Li, S. Mandt, S. Agrawal, E. Khan (Eds.), Proceedings of The 28th International Conference on Artificial Intelligence and Statistics, volume 258 of *Proceedings of Machine Learning Research*, PMLR, 2025, pp. 3358–3366. URL: <https://proceedings.mlr.press/v258/fonio25a.html>.
- [15] H.-M. Yang, X.-Y. Zhang, F. Yin, C.-L. Liu, Robust classification with convolutional prototype learning, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 3474–3482.
- [16] P. Somervuo, T. Kohonen, Self-organizing maps and learning vector quantization for feature sequences, Neural Processing Letters 10 (1999) 151–159.
- [17] R. Tibshirani, T. Hastie, B. Narasimhan, G. Chu, Diagnosis of multiple cancer types by shrunken centroids of gene expression, Proceedings of the National Academy of Sciences 99 (2002) 6567–6572.
- [18] S. Fonio, L. Paletto, M. Cerrato, D. Ienco, R. Esposito, et al., Hierarchical priors for hyperspherical prototypical networks, in: ESANN 2023-Proceedings, ESANN, 2023, pp. 459–464.
- [19] L. Landrieu, V. S. F. Garnot, Leveraging class hierarchies with metric-guided prototype learning, in: British Machine Vision Conference (BMVC), 2021.
- [20] Y. Guo, X. Wang, Y. Chen, S. X. Yu, Clipped hyperbolic classifiers are super-hyperbolic classifiers, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11–20.
- [21] M. van Spengler, E. Berkhout, P. Mettes, Poincaré resnet, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 5419–5428.
- [22] Y. Yue, F. Lin, G. Mou, Z. Zhang, Understanding hyperbolic metric learning through hard negative

- sampling, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 1891–1903.
- [23] M. Law, R. Liao, J. Snell, R. Zemel, Lorentzian distance learning for hyperbolic representations, in: International Conference on Machine Learning, PMLR, 2019, pp. 3672–3681.
 - [24] A. Bdeir, K. Schwethelm, N. Landwehr, Fully hyperbolic convolutional neural networks for computer vision, arXiv preprint arXiv:2303.15919 (2023).
 - [25] B. Wilson, M. Leimeister, Gradient descent in hyperbolic space, arXiv preprint arXiv:1805.08207 (2018).
 - [26] S. Bonnabel, Stochastic gradient descent on riemannian manifolds, IEEE Transactions on Automatic Control 58 (2013) 2217–2229.
 - [27] G. Bécigneul, O.-E. Ganea, Riemannian adaptive optimization methods, arXiv preprint arXiv:1810.00760 (2018).
 - [28] A. Krizhevsky, Learning multiple layers of features from tiny images, <https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf> (2009).
 - [29] C. Wah, S. Branson, P. Welinder, P. Perona, S. Belongie, The Caltech-UCSD Birds-200-2011 Dataset, Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.
 - [30] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, A. Vedaldi, Fine-Grained Visual Classification of Aircraft, Technical Report, 2013. arXiv:1306.5151.
 - [31] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
 - [32] M. Kochurov, R. Karimov, S. Kozlukov, Geoopt: Riemannian optimization in pytorch, 2020. arXiv:2005.02819.
 - [33] M. Aldinucci, S. Rabellino, M. Pironti, F. Spiga, P. Viviani, M. Drocco, M. Guerzoni, G. Boella, M. Melia, P. Margara, et al., Hpc4ai: an ai-on-demand federated platform endeavour, in: Proceedings of the 15th ACM International Conference on Computing Frontiers, 2018, pp. 279–286.
 - [34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, arXiv preprint arXiv:1706.06083 (2017).