Department of Mathematics
TUM School of Computation, Information and Technology
Technical University of Munich

Improving Topologically-Regularized Multiple Instance Learning on Single Cell Images

Milad Bassil

Thesis for the attainment of the academic degree

Master of Mathematics in Science and Engineering

at the TUM School of Computation, Information and Technology of the Technical University of Munich

Supervisor:
Dr. Carsten Marr, Dr. Bastian Rieck

Advisors:
Salome Kazeminia

Submitted:
Munich, 15. December 2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 15. December 2024 Milad Bassil

Zusammenfassung

In der biomedizinischen Datenanalyse ist der Mangel an annotierten Daten ein häufiges Problem, und Multiple Instance Learning (MIL) hat sich als vielversprechender Ansatz erwiesen, um Herausforderungen wie begrenzte Annotationen oder grobe Beschriftungen zu bewältigen. Trotz der Stärken von MIL gibt es weiterhin Herausforderungen wie datenintensive Anforderungen, Instabilität im Training und eine Anfälligkeit für Overfitting bei begrenzten Daten.

Topologische Regularisierung ist eine potenzielle Lösung, die das Training hin zu einer generalisierbareren und robusteren Merkmalsextraktion lenkt. Um topologische Informationen von Datenräumen zu erfassen, müssen diese metrische Räume sein, was bedeutet, dass eine Distanzfunktion über sie definiert werden muss. Eine robuste Distanzfunktion, die in der Lage ist, informative Abstände zu erzeugen und dabei semantisch invariante Transformationen zu berücksichtigen, ist entscheidend für die Leistung. In den meisten relevanten Arbeiten wird eine Minkowski-Distanzfunktion über dem analysierten Raum definiert. Bei der Untersuchung der Topologie von Bildräumen könnte eine Minkowski-Distanzfunktion jedoch nicht ausreichen. Um die topologische Regularisierung auf Eingabebildräumen zu verbessern, untersuchen wir verschiedene Distanzfunktionen, die repräsentativere Abstände zwischen den Instanzen liefern können und dadurch eine robustere Regularisierung der Eingabe- und Latenträume ermöglichen.

Die Effektivität dieser Distanzfunktionen wird auf mehreren synthetischen und realen Datensätzen getestet. Unsere realweltliche Anwendung konzentriert sich auf Bilder einzelner Blutzellen und bewertet die Auswirkungen dieser Funktionen auf ein topologisch reguliertes Modell zur Identifikation von Leukämiesubtypen aus Blutausstrichbildern, genannt SCEMILA.

Abstract

In biomedical data analysis, label scarcity is a common issue, and Multiple Instance Learning (MIL) has emerged as a promising approach to handle limited annotation or coarse labeling challenges. Despite the strengths of MIL, it still faces issues such as data-intensive requirements, training instability, and susceptibility to overfitting in cases with limited data.

Topological regularization is a potential solution that guides training towards more generalizable and robust feature extraction. Capturing topological information of data spaces requires them to be metric spaces, meaning that a distance function must be defined over them. Therefore, a robust distance function capable of producing informative distances, unaffected by semantically invariant transformations, is crucial for performance. In most relevant literature, a Minkowski distance function is defined over the space being analyzed. However, when studying the topology of image spaces, a Minkowski distance function may not suffice. To improve topological regularization on image input spaces, we explore various distance functions that may yield more representative inter-bag distances, leading to a more robust regularization of input and latent spaces.

The effectiveness of these distance functions is tested on several synthetic and real-world datasets. Our real-world application focuses on single blood cell images, evaluating the impact of these functions on a topologically regularized model for Leukemia subtype identification from blood smear images, called SCEMILA.

Contents

1 Introduction

Machine learning applications in the biomedical field are rapidly expanding, offering innovative solutions to complex problems, however, one of the major challenges in this domain is data scarcity[1]. A specific application of interest in this work is automating the detection of Acute Myeloid Leukemia (AML) subtypes using blood smear single-cell images. Automating this process has the potential to significantly reduce the resources required for diagnosis, such as genetic testing, which is traditionally used to determine AML subtypes by analyzing chromosomal abnormalities and genetic mutations[2].

In the proposed approaches of [3, 4, 5], blood smear images of a patient are segmented into individual single-cell images. A prediction is then made based on a single label for the collection of segmented images, framing the task as a Multiple Instance Learning (MIL) problem. MIL is a self-supervised machine learning paradigm designed to handle data with a set-like structure, where the input consists of bags (collections of instances) and each bag is associated with a label . Unlike traditional supervised learning, MIL does not require instance-level labels; instead, it learns to predict the bag's label based on the instances within it[6].

Typically, an MIL classification model consists of three main components: an instance encoder to extract features from individual instances, an instance aggregator to combine features from all instances within a bag, and a bag classifier to predict the label. This architecture has been successfully applied in diverse areas, such as drug activity prediction, document classification, and computer vision tasks[7][8]. However, training MIL models is often challenging due to the ambiguity between bag labels and their instances, placing significant demands on the encoder to produce discriminative representations and on the aggregator to identify meaningful patterns[9][10]. As a result, MIL models are prone to overfitting, particularly when faced with limited datasets, necessitating the use of additional regularization strategies.

One promising regularization technique is Topological Regularization (TR), inspired by methods from Topological Data Analysis (TDA). TR uses multi-scale topological encodings of metric spaces to compute differentiable losses, guiding models to produce latent representations with structures that mirror the input data's topology at multiple scales. This technique has been shown to enhance generalization, robustness, and interpretability, while also mitigating overfitting risks. By providing a regularization signal from the data itself, TR helps ensure that the latent spaces are structured meaningfully[11].

To compute topological encodings, pairwise distances between data points in the latent space are required, meaning the quality of the encoding depends on the quality of these distances. Prior work [12] has demonstrated that using simple Euclidean distances can improve performance in low-resolution, centered datasets like MNIST[13] and FashionMNIST[14]. However, as will be shown in this work, when applied to high-dimensional, non-centered image datasets, the limitations of Euclidean distance become evident, as it struggles to correlate with semantic labels effectively. This thesis explores these challenges in the context of high-dimensional, non-centered blood smear images for AML subtype detection, integrating topological regularization into the MIL framework to address the limitations of traditional approaches.

The main contributions of this work are as follows:

1. We present metrics that quantify the quality of a distance function over an image space and demonstrate their effectiveness.

2. We offer a collection of distance functions that are better suited for single cell image spaces and quantify their quality.

3. We propose a new topological loss function, based on [15], to improve both computational efficiency and model performance.

2 Background

2.1 Multiple Instance Learning

Multiple Instance Learning (MIL) was introduced in 1997 by Dietterich et al. [16], initially as a binary classification problem where all instances within a bag were either positive or negative, and the presence of at least one positive instance implied a positive bag label. Since its inception, MIL has gained attention due to its ability to address scenarios where labeled data is scarce or difficult to obtain, effectively shifting the task of recognizing intermediate or hidden labels to the model itself. Over time, MIL has evolved to encompass more complex problem settings, such as multi-class, multi-membership tasks and cases where the relationship between instance and bag labels is indirect or unknown—i.e., a positive instance does not necessarily imply a positive bag label, but instead requires the model to learn this relationship [6]. Despite significant shifts in the MIL paradigm, its core components have largely remained the same, functioning as follows:

Let $B = \{x_1, x_2, \ldots, x_n\}$ represent a bag B containing n instances. In MIL, each bag has a single label y_B, where $y_B \in \{0, 1\}$ for binary classification tasks. Each instance x_i in bag B is encoded using an encoder function f, which maps each instance x_i to an embedding in a feature space:

$$h_i = f(x_i), \quad h_i \in \mathbb{R}^d$$

where h_i is the encoded representation of instance x_i in a d-dimensional space. The set of embeddings $\{h_1, h_2, \ldots, h_n\}$ is then combined using an aggregation function g, producing a single bag-level embedding h_B for the entire bag:

$$h_B = g(h_1, h_2, \ldots, h_n), \quad h_B \in \mathbb{R}^{\mathrm{d'}}$$

where h_B is the encoded representation of the bag B in a d' dimensional space. Typically, d=d'.

The aggregation function g varies depending on the MIL strategy; common choices include max or mean pooling. Finally, the bag-level embedding h_B is passed through a classifier c to predict the bag label $\hat{y}_B$:

$$\hat{y}_B = c(h_B), \quad h_i \in \mathbb{R}^{\mathrm{cl}}$$

where cl is the number of classification classes.

In a simple binary classification setting, c could be a sigmoid function applied to a linear transformation of h_B:

$$\hat{y}_B = \sigma(w^\top h_B + b)$$

where σ is the sigmoid function, and w and b are parameters learned during training.

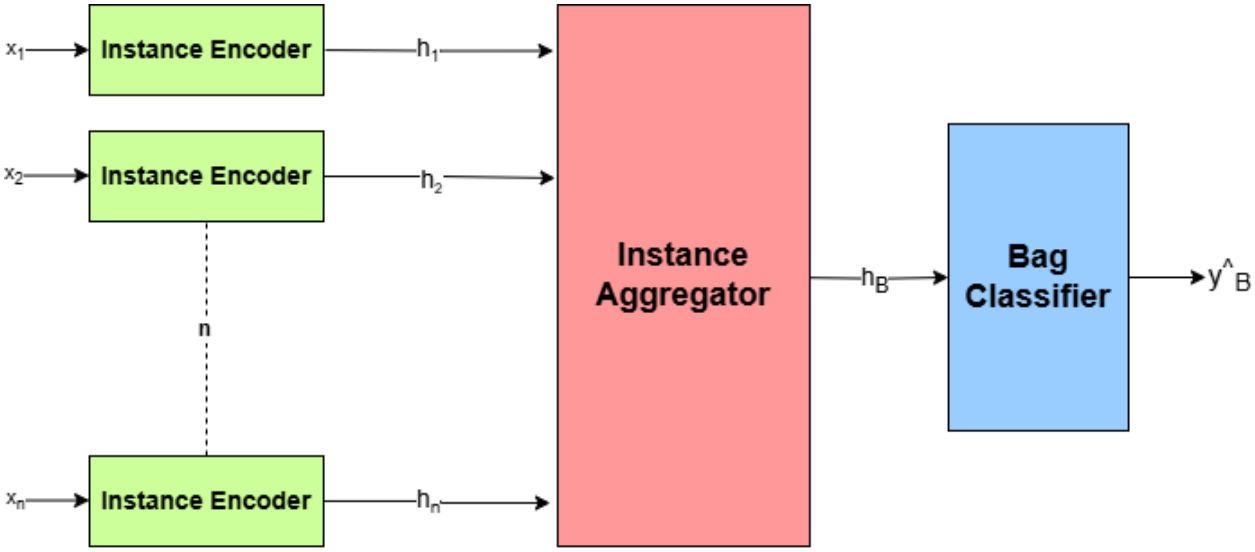


Figure 2.1 This is a classic MIL architecture, where B ,n, x_i, h_i, h_B and $\hat{y}_B$ are bag, bag size, input instance, instance latent representation vector, bag latent representation vector and bag level class probability distribution, respectively. A detailed description is given in section 2.1.

2.2 Manifold Hypothesis in Machine Learning

In machine learning, the manifold hypothesis suggests that high-dimensional data—such as images, audio, and text—can be effectively represented within a lower-dimensional space [17]. This hypothesis is based on the idea that such data has a lower intrinsic dimensionality than it's ambient space's dimensionality. For instance, consider images: the set of coherent images that can be taken with a camera at a given resolution R represents only a tiny subset of all possible pixel configurations at that resolution. To illustrate that, think of randomly generated images of resolution R. The probability of a random valued image as seen bellow has an extremely low probability of being coherent, illustrating the sparse nature of meaningful data within the high-dimensional space [17]. To demonstrate that, we randomly generate 64 images in fig 2.2, with not one looking remotely coherent, thus showing that coherent images occupy a small subspace of the set of all images.

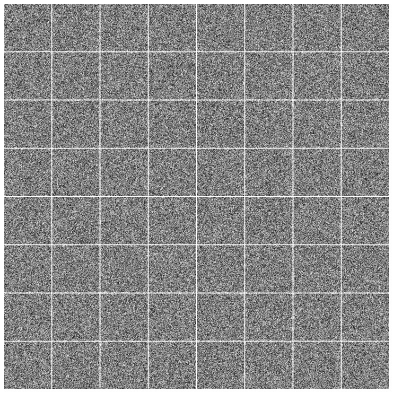


Figure 2.2 64 100x100 grayscale images generated by a uniform distribution over possible images. We can see all 64 images are incoherent, meaning an image without anything meaningfully in it, demonstrating that set of coherent images occupy a very small subspace of all possible images.

We consider a ball of radius ϵ and of dimension equal to that of the image, centered around a coherent image x. This region, for a small enough ϵ, will consist entirely of coherent images. In practice, the ball around the coherent image corresponds to the image with additive uniformly generated noise of magnitude $\sigma = \epsilon$.

As an example, see the figure below, where x represents the original image of the parrot, and all other images are samples within the ball for varying ϵ values. This demonstrates that the subspace of coherent images is, at least piece-wise, continuous [18].

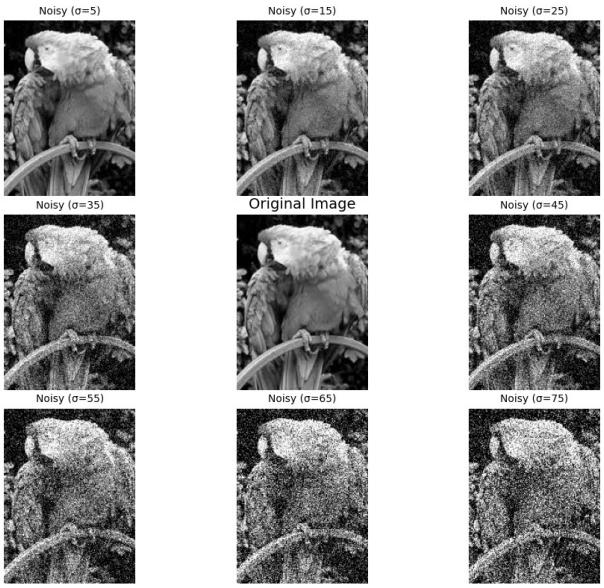


Figure 2.3 Center image is the original and the rest are the original image but varying magnitudes of added noise, demonstrating that the subspace of coherent images is piece-wise continuous.

2.3 Topology

Topology is a branch of mathematics that studies the properties of spaces that are preserved under continuous transformations [19]. Congruently we call T a topological feature extractor if it satisfies the following:

For any topological space X and for any continuous transformation $f : X \to X'$ (where X' is another topological space), the feature extractor must yield the same result for all continuous transformations applied to the space. Formally, we can express this property as:

$$T(X) = T(f(X)) \quad \forall f : X \to X' \text{ continuous}$$

This means that the extracted features remain invariant under continuous deformations of the space X. Thus, a topological feature extractor captures the essential topological properties of the space, independent of how it is continuously transformed.

From a geometric perspective, continuous transformations can be thought of as actions like stretching, twisting, and bending, but without tearing or gluing. Topology focuses on properties that remain unchanged under these transformations. A classic example used to illustrate topology is the equivalence of a mug and a doughnut, as well as a cube and a sphere, as shown below.

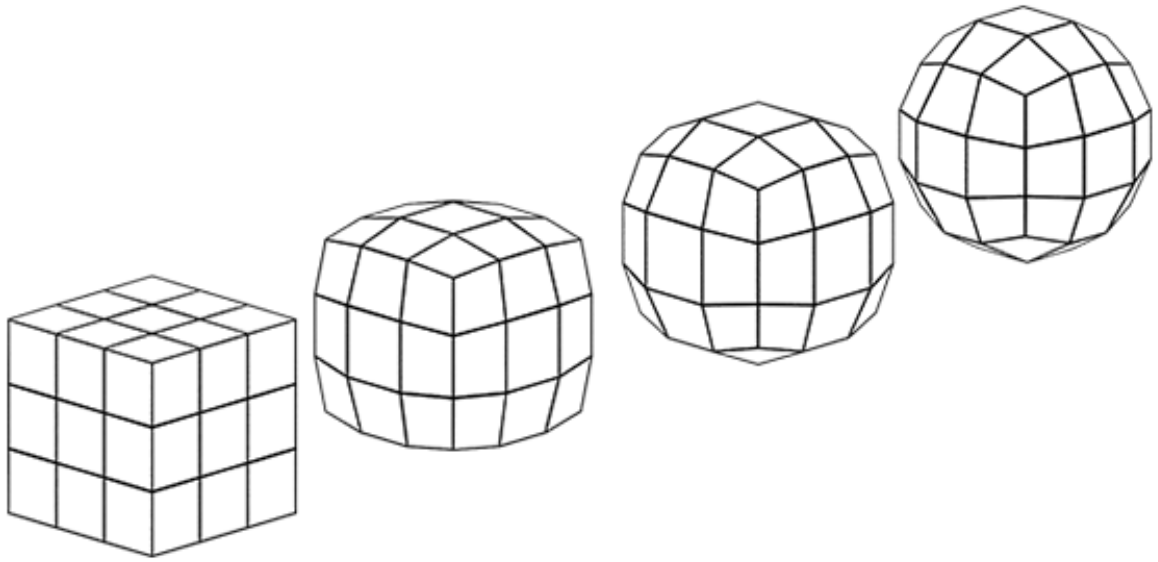


Figure 2.4 It is possible to continuously deform a cube into a sphere. This implies, by the definition of a topological feature, that a sphere and a cube have the same topological features. An example of a topological feature of these two shapes is the absence of a hole. This image was taken from [20].

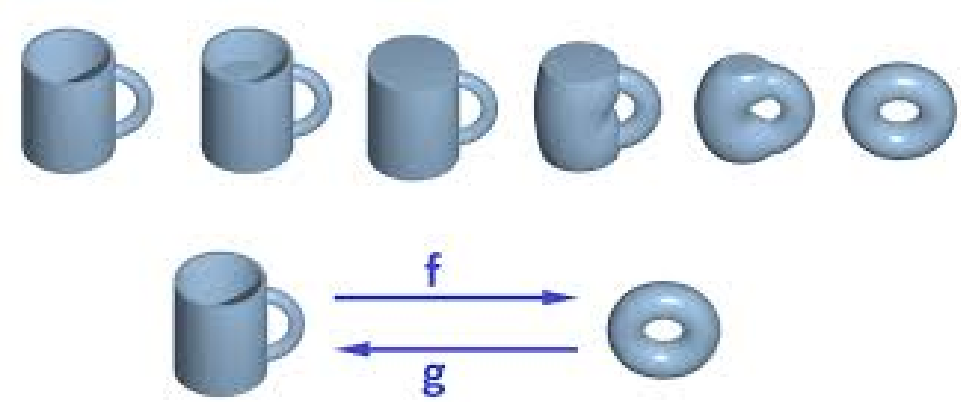


Figure 2.5 It is possible to continuously deform a mug into a doughnut. Therefore, by definition, a doughnut and a mug are topologically similar. This image was taken from [20].

Based on this definition of topology, one would expect the mug and doughnut to have similar topological features, which are distinct from those of the cube and sphere, which are themselves similar. From this, we can surmise that a function which counts the number of "holes" in a topological space is a type of topological feature extractor, as no matter how one continuously deforms a mug, the hole in the handle will not disappear [21].

2.4 Persistent Homology

Persistent Homology is a topological data analysis method that studies the shape of data points in a metric space by identifying and tracking topological features that emerge and disappear at different scales. This technique encodes structure as simplicial complexes, which are combinatorial elements representing shapes in different dimensions: a 0D simplex corresponds to an isolated points, a 1D simplex to an edge, a 2D simplex to a triangle, a 3D simplex to a tetrahedrons and extending to higher dimensions, as shown in fig 2.6. A simplicial complex is a collection of simplices.

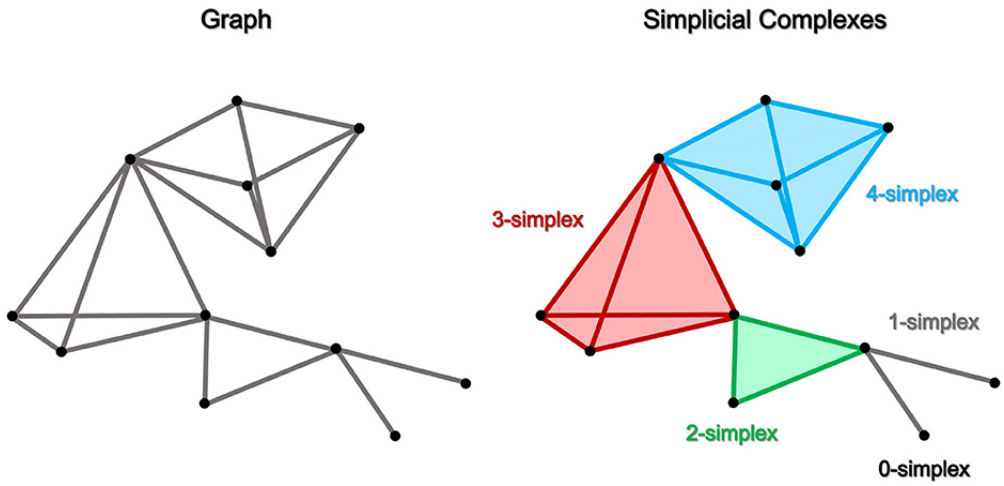


Figure 2.6 On the left we have the input structure as a non-directional graph and on the right we see examples of different degree simiplicial complexes that are contained in the graph. This image was taken from [22].

To give a more rigorous definition, a *simplicial complex* K is a collection of simplices that satisfies the following two conditions:

1. If $\sigma \in K$, then every face of σ is also in K.
2. If $\sigma, \tau \in K$, then $\sigma \cap \tau$ is either empty or a face of both σ and τ.

Here, a *simplex* is defined as the convex hull of a set of affinely independent points in some Euclidean space, and its *faces* are the convex hulls of subsets of these points, meaning we get some vector representation of the shapes.

The advantage of encoding topological spaces as simplicial complexes, is that we can calculate the topological invariant on the simplices instead of calculating it directly on the structure, which is not necessarily feasible on a computer. This is done by finding the homology groups in that represent these topological feature or those features which are not affected by continuous transformation.

Bellow we describe how we can calculated and find these homology groups.

Let X be a topological space. The **homology groups** of X, denoted by $H_n(X)$ for $n \geq 0$, are defined as follows[11]:

1. Construct a sequence of abelian groups (or modules) called *chains*:

$$\cdots \to C_{n+1}(X) \to C_n(X) \to C_{n-1}(X) \to \cdots \to C_0(X) \to 0$$

where each $C_n(X)$ is the group of n-**chains**, consisting of formal sums of n-simplices in topological space X.

2. Define the *boundary map* $\partial_n : C_n(X) \to C_{n-1}(X)$ for each n which has the property that $\partial_{n-1} \circ \partial_n = 0$. This ensures that the image of ∂_n (the *boundaries*) is contained within the kernel of ∂_{n-1} (the *cycles*).

3. The n-th homology group, $H_n(X)$, is defined as the quotient:

$$H_n(X) = \frac{\ker(\partial_n)}{\text{im}(\partial_{n+1})}$$

where $\ker(\partial_n)$ is the group of n-cycles (chains with no boundary) and $\text{im}(\partial_{n+1})$ is the group of n-boundaries (boundaries of $(n+1)$-chains).

The homology group $H_n(X)$ provides an algebraic invariant that captures information about the n-dimensional holes in X which are the topological features of interest:

- $H_0(X)$ represents the connected components of X.
- $H_1(X)$ represents loops or 1-dimensional holes.

- $H_2(X)$ represents 2-dimensional voids, and so on.

To capture changes across scales, Persistent Homology uses a filtration process that incrementally builds simplicial complexes by connecting points within a threshold distance, starting from isolated points and adding connections as the scale parameter increases. The filtration used in this work is the Vietoris-Rips complex, which adds edges and higher-dimensional simplices depending on the distance between points, as shown in the figure below. The filtration produces a sequence of complexes, each containing the preceding ones as subsets. Persistent Homology then tracks when features, such as loops or clusters, appear (birth) and merge into larger structures or vanish (death) as the scale increases, effectively encoding the data's topological characteristics across multiple scales. To recapitulate, the word **persistent** refers to the fact that this method studies or tracks the persistence of topological features as some value is changed; in this case, that value is scale. The word **Homology** refers to the fact that we are mapping topological structures in the space to algebraic simplicial complexes that are homologous to the topological structures. This means that we can perform operations on the simplicial complexes that respect the topological structures they represent.

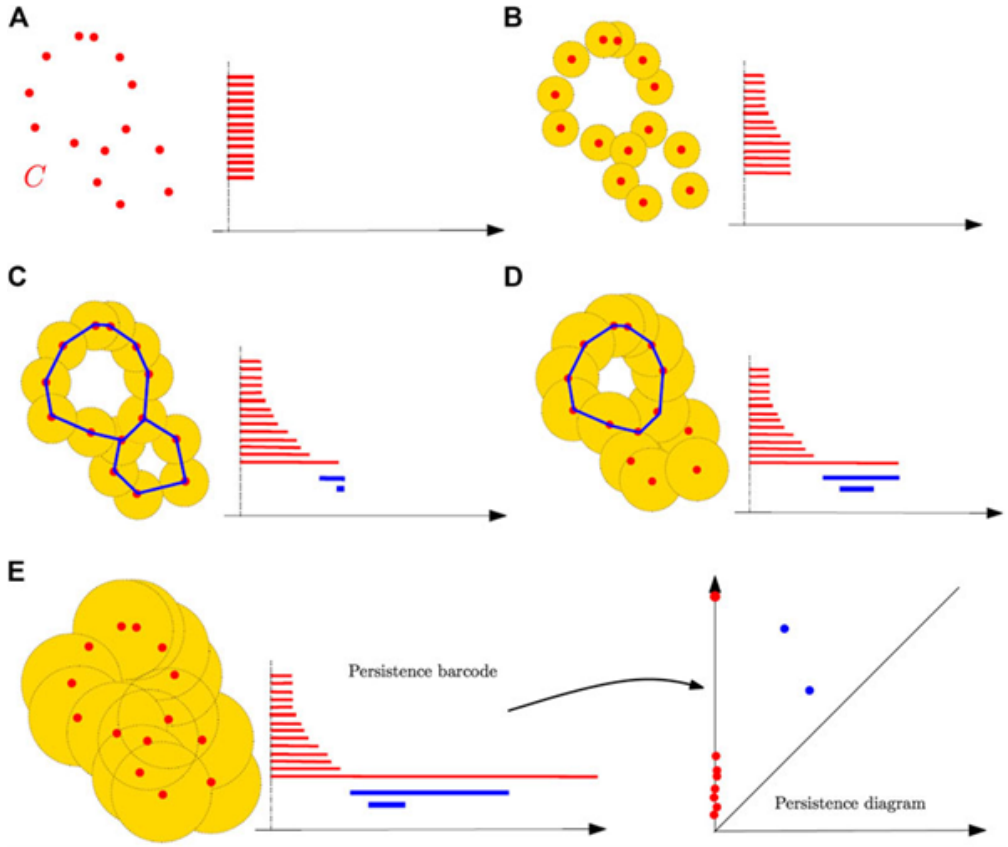


Figure 2.7 Topology Calculation and Encoding Approach; **A** Start with a point cloud with a distance function defined on space, as the red bars represent the first order topo-features at scale = 0 we can see the number of features is equal to the number of points since every point is an island not connected to anything else . **B** Increase scale and connect the points whose distance is below the scale threshold, connecting points causes one of them to die, hence the bar stops. **C** Here we observe the birth of 2 first order topological features, holes, 2 blue bars are drawn to represent them. **D** Stop one of the blue bars as the cycle disappeared at a high enough scale. **E** From the persistence barcode calculate the persistence diagram. This image was taken from [20].

2.5 Topological Regularization

Topological regularization is an emerging technique in machine learning aimed at preserving and exploiting the topological properties of data. Traditional regularization methods, such as L2 regularization, focus

on penalizing the magnitude of model parameters to prevent overfitting; however, these approaches do not explicitly consider the structural relationships within the data. Topological regularization addresses this gap by incorporating information about the "shape" or intrinsic structure of the data manifold.

In the work [23], topological regularization was used to improve the performance of a binary classifier. Using a piece wise linear approximation of the classification level set (the domain for which the classifier predicts the same probability for both classes), the authors calculate an approximate topological signature for the class boundary. They then attempt to topologically regularize the network by adjusting the critical points in the boundary approximation, which leads to the disappearance of low-persistence topological features, effectively reducing the complexity of the boundary between the classes. In the works [24] and [25], topological regularization is used to inject a topological prior into the training of models. For example, if the number of clusters to be expected is known, this information can be used to bias the network or model by injecting the topological prior into the latent space. In [25], the embedding of human cells at different stages of the cell cycle, where timestamps or ages of cells are available, is presented as an example of an application with a topological prior. Using the timestamps, one can bias the embedding model to respect the connectivity (a topological feature) of cells of similar age, thereby preserving the continuous nature of cells in the embedding.

All of the aforementioned works require a differentiable topological loss function to calculate gradients for backpropagation in the network. The implementation of topological loss functions while retaining auto-differentiability is one of the bigger challenges in this field.

In [7], the topological difference or loss between multiple networks with different modalities is calculated to guide the training of a multi-modal network. The topological loss acts as a guide that helps the model selectively leverage informative and cohesive modalities while disregarding redundancies. In this application, the need to calculate an explicit gradient is avoided by using the loss to simply guide the training and not update the parameters themselves. However, it still acts as a regularizer, as it controls the complexity of the model by limiting its inputs, instead of influencing its parameters directly.

In [15], a differentiable topological loss function is defined between an autoencoder's latent space and its input space. The intuition behind this is to let the model be informed by the input space's topology. The hope is to prevent it from creating fake topological features by overgeneralizing or from destroying potentially informative topological features in the input space as they propagate through the layers. This work showed competitive autoencoder reconstruction and latent space distribution performance; however, it was only tested on relatively small-dimensional data, ranging from 3-dimensional synthetic data to CIFAR-10, which is 32x32 RGB. Additionally, a Euclidean metric space was assumed on both the input and latent space, severely limiting the expressiveness of the topological encoding. In a follow-up work [12], different distance schemes were attempted, namely random convolutions and image perceptual distance, with no marked improvement over classic Euclidean distance.

Our work challenges the conclusion of [12], citing the following reasons;

1 In [15] only low resolution images were used, where euclidean distance retains some of its expressiveness.

2 All datasets that were used MNIST, FMNIST and CIFAR10 are centered, rotationally static image datasets, thus euclidean distance's downsides are mitigated.

3 The topological loss, as formulated in that work, is not expressive enough to use the improved distance functions proposed in [12].

3 Methods

This chapter will go over the methods used in the development of the baseline MIL model, the topologically regularized MIL model and the distance function evaluation techniques. All described methods have been implemented in the following repository: https://github.com/sewerrenegade/Master_Thesis_Code

3.1 Multiple Instance Learning Model

The deep model used in this work follows the classic MIL structure with an encoder, aggregator and classifier.

3.1.1 Instance Encoder

The encoder is based on the ResNet18 [26] deep convolutional neural network, which employs 3x3 and 1x1 convolution kernels across 17 layers. It has about 11.7 million trainable parameters and was developed for RGB images of dimension 224x224. For our application, we remove the last two layers of the network—the classification head and the average pooling layer—and replace them with two convolutions followed by one fully connected layer. As an additional modification, we add a dropout layer to the encoder at four different depths. The resultant encoder network has 12.5 million trainable parameters. Further variants of this encoder were implemented, such as:

1 A version of ResNet18 whose input layers were modified to work on grayscale images.

2 Using DinoBloomV2 S [27] as an instance encoder, DinoBloom is a foundation model trained on blood cell images using contrastive learning. Gradients are not propagated through DinoBloom, it is treated a frozen encoder.

3 ResNet32 network trained on single cell level labels [28], no gradients were propagated through this encoder since the network weights are not available, instead saved latent codes used.

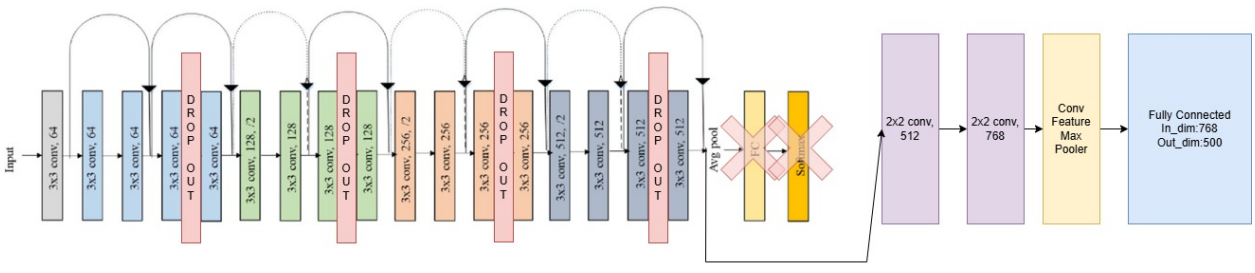


Figure 3.1 This is the our modified ResNet18 implementation, we added 4 dropout layers and replace the last two layer with more convolutions, fully connected and pooling layers as described in section 3.1.1.

3.1.2 Instance Aggregator

The instance aggregator is responsible for producing a bag level encoding using the encoded instances in the bag. In out implementation we use the same aggregator structure as [3] and [4]. Multihead attention pooling is an extension of the attention mechanism that aggregates input representations by learning multiple distinct attention heads. Each attention head computes attention scores based on learnable query,

key, and value projections, enabling the model to capture diverse aspects of the input data. The outputs of the heads are concatenated and linearly projected to form a summary representation. This approach is particularly effective for tasks such as text classification and multiple instance learning, where it aggregates information from a set of inputs into a compact, task-specific embedding. In out case we use the pooling mechnism to pool the instance level codes into a bag level code.

3.1.3 Classifier Head

The classifier, which finally outputs a predicted label, has two FC layers separated by a normal ReLU function, with the output of the first FC layer being 64 and the output head being 5-dimensional.

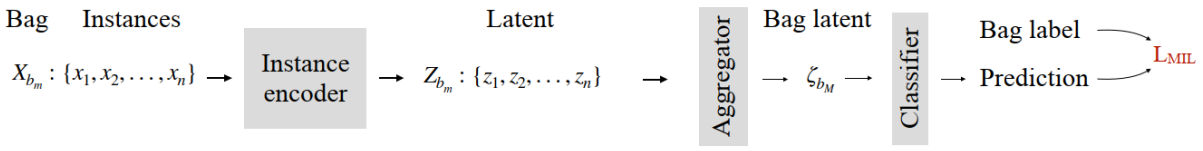


Figure 3.2 This figure shows an overview of our baseline multiple instance learning pipeline without any topological regularization. It is training only on classification loss.

3.2 Baseline Multiple Instance Learning Approach

To measure the impact of topological regularization we setup a structured PyTorch Lightning based pipeline, that is externally configurable to run multiple approaches with different settings. To establish a baseline for comparison we trained our model without any topological regularization, carefully optimizing many hyper-parameters, which will be detailed later. For a fair comparison, the same optimized hyper-parameters were maintained in the subsequent experiments, including topologically regularized approaches.

3.2.1 Data Preperation and Preporcessing

To train our model we used the AML Cytomorphology MLL Helmholtz dataset [29], which is comprised of patients diagnosed with one of four prevalent subtypes of acute myeloid leukemia (AML). The subtypes were identified based on genetic abnormalities and morphological features, and are named: NPM1, PML::Para, RUNX1::RUNX1T1 and CBFB::MYH11 of which there are 45, 51, 38 and 47 patients respectively, additionally the dataset includes 61 control patients. Each patient in our dataset has a 99 to 500 single cell images, with the average number of instances of the bags being 431 and a standard deviation of 107. The individual single cell images are 144x144 pixels with RGB channels encoded in a tif format, with them all existing on the same magnification scale of x40, making each pixel equivalent to 0.172μm. In the referenced dataset, 53 patients are not included as their data is exclusively shared with researchers at Helmholtz, however our observations indicate the impact of this is marginal. In our experiments, we allocated 21% of the data to the test set, while applying a fixed 4-fold cross-validation on the remaining 79%. As a result, each training run approximately used a 20%/20%/60% split for test, validation, and training data, respectively.

The private Helmholtz version of the dataset includes a 5-patient, fully annotated meta file with both patient-level and single-cell-level annotations. This data was not used for training but was instead employed to visualize the quality of the single-cell latent space.

3.2.2 Data Augmentaiton and Resampling

To improve our model's ability to generalize, we applied randomized augmentations to our single-cell images. The augmentations considered in our pipeline include color jitter, horizontal and vertical flips,

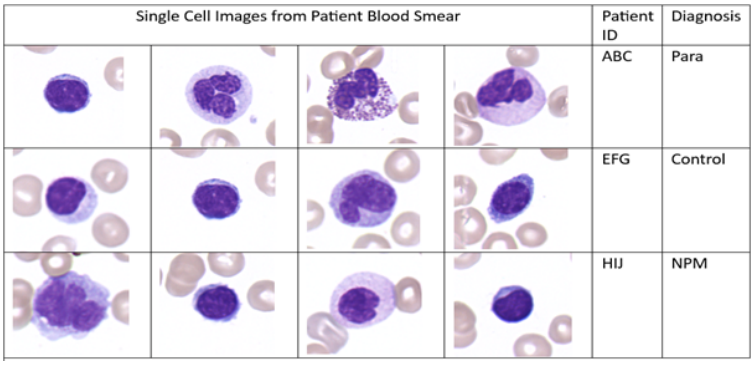


Figure 3.3 Our data is organized as such, we have a patient level AML subtype diagnosis and a corresponding set of single cell images, with images per bag ranging from 99 to 500.

sharpness adjustments, rotation, translation, Gaussian blur, and Gaussian noise. Each augmentation was applied independently with a binomial distribution, such that approximately 50% of all images were augmented by one or more transformations. These augmentation settings were validated by an expert pathologist to ensure they do not destroy important morphological features relevant to diagnosing acute myeloid leukemia.

In addition to other strategies, we resampled underrepresented classes, such as RUNX1::RUNX1T1, which has 25% fewer samples than the control group, to mitigate the negative effects of class imbalance in the dataset. It should be noted that resampled bags or patients have different augmentations applied, so the resampled data points vary.

The test set was excluded from all augmentation and resampling strategies to maintain consistency and ensure reliable evaluation of model performance.

3.2.3 Loss Function

A cross-entropy (CE) loss function, also referred to as MIL loss, was used for the multi-class classification task. The cross-entropy loss function was enhanced with exponentially decaying label smoothing. The equations for the standard and smoothed cross-entropy loss functions are as follows:

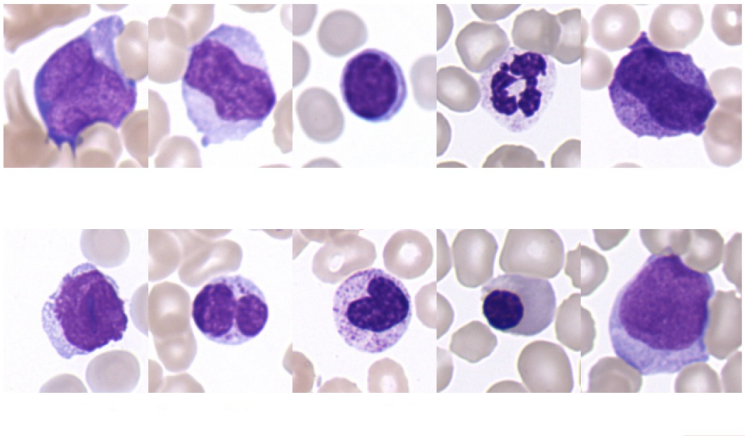
$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(p_i)$$

$$\mathcal{L}_{SCE} = -\sum_{i=1}^{C} \left((1-s)y_i + \frac{s}{C}\right) \log(p_i)$$

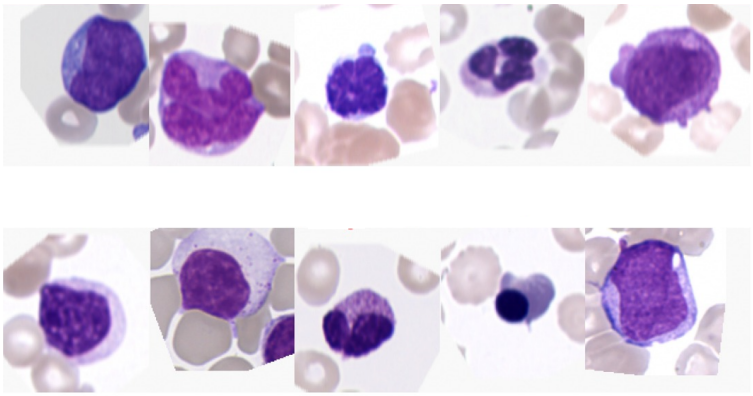
Where C is the class count, y_i is the index of a one-hot encoded vector corresponding to the label, p_i is the predicted probability for the class indexed with i, and s is the smoothing factor.

The use of smoothing helps counteract the bias where the model was inclined to predict leukemia cases with constant low certainty while showing high certainty for healthy controls. We hypothesize that the model gets stuck in a local minimum because the ratio between rewards for certainty and penalties for uncertainty is too low. Consequently, the smoothing reduces the reward for high-certainty predictions. Examining the smoothed cross-entropy equation reveals that smoothing diminishes the reward for high-certainty predictions. The smoothing factor decays exponentially to zero, ensuring it does not interfere with the convergence of the loss function. The cross-entropy loss function was additionally modified to allow for class weighting. Class weighting scales the loss function according to the rarity or frequency of occurrence of a class in the dataset, it follows this equation normal and smoothed cross entropy are respectively:

$$\mathcal{L}_{\text{CEweighted}} = -\sum_{i=1}^{C} w_i y_i \log(p_i)$$



(a) Samples before augmentation.



(b) Samples after augmentation.

Figure 3.4 Data samples before and after the application of image augmentations, shown as two separate parts.

$$\mathcal{L}_{\text{SCEweighted}} = -\sum_{i=1}^{C} w_i \left((1-s)y_i + \frac{s}{C}\right) \log(p_i)$$

where w_i is defined by:

$$w_i = (1 - \text{weighting\_factor}) + \text{weighting\_factor} \times \frac{N_{\text{total}}}{C \cdot n_i}$$

where N_{total} is the dataset size, C is the class count and n_i is number of data samples belonging to class i and the weighting_factor is a value between 0 and 1 that determines the strength of the class weighting. A value of 0 means no class weighting is applied, while a value of 1 signifies full class weighting.

3.2.4 Epochs

A maximum of 250 epochs was set for training. However, to mitigate the risk of overfitting and save on computational time, an early stopping mechanism was incorporated. The mechanism monitors the validation loss, terminating training if no improvement is observed over a window of 50 epochs. For implementing early stopping, we used the native PyTorch Lightning functionality.

3.2.5 Batch Size

A batch size of 1 was used during training, but we compensated for this by accumulating 32 gradients before performing an optimization step. Although this approach is computationally slower and does not

fully leverage the parallelism of GPUs, it was necessary due to the large and variable input size. The large input size made it challenging to train on GPUs with less than 13GB of memory, especially with 4 instances of data loaders. Furthermore, the variable input size of the model prevents PyTorch from fully utilizing its computational optimizations, potentially leading to suboptimal performance during both training and inference. It is important to note that there are no significant differences in model performance when using different batch sizes or gradient accumulation, aside from the impact on computational speed.

3.2.6 Learning Rate Control

An Adaptive Moment Estimation (Adam) optimizer, with an initial learning rate of 0.00005, was employed. Adam combines gradient momentum and the variance of the gradient to compute an adaptive gradient magnitude. To further control the learning rate, we used PyTorch's *ReduceLROnPlateau* scheduler, which monitors validation accuracy. If the validation accuracy plateaus for more than 20 epochs, the scheduler reduces the learning rate by a factor of 2. This strategy helps the model adjust its learning rate to achieve better loss convergence.

3.2.7 Checkpointing and Model Selection

A custom checkpointing and model selection function was developed to optimize model parameter checkpointing based on dual metrics: validation accuracy and validation CE loss. The method tracks these metrics at each epoch and selectively saves up to k model versions, chosen based on accuracy as the primary criterion. The function stores the k models that achieve the highest validation accuracy. For most observed cases, there were multiple checkpoints that share the same accuracy; the function uses validation CE loss as a tie-breaking metric, picking models with lower loss values among those with equal accuracy. At the end of training, the top-ranked model among the saved checkpoints is chosen as the final model. This strategy ensures that both high accuracy and stability are considered in selecting the optimal model configuration.

3.2.8 Other Regularization Techniques

Dropout was added into the encoder model at four distinct depths, as discussed in Section 3.1.1, using a 10% dropout rate as a regularization technique. Dropout is commonly used in neural networks to prevent overfitting by randomly ignoring a proportion of neurons during the training process. This forces the model to learn redundant and distributed representations of the input by preventing dependence on certain limited features, which makes the network more robust and generalizable. By adding dropout at multiple levels of the ResNet18 based encoder, we enhance the network's capacity to capture diverse features across depths while reducing the likelihood of overfitting to the training set.

Additionally, we incorporated an L2 regularization on the model parameters in the form of weight decay. It penalizes large parameter values, enhancing the model's generalization capabilities. Weight Decay is equivalent to adding the L2 norm of the parameters to the loss function; however, it is more efficient as no gradient calculation from the loss function is required. The gradient is simply calculated as shown below:

$$\mathcal{L}_{\text{L2reg}}(\mathbf{w}) = \mathcal{L}_{\text{CE}}(\mathbf{w}) + \frac{\lambda}{2}\sum_i w_i^2 \qquad \frac{\partial \mathcal{L}_{\text{L2reg}}}{\partial w_i} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_i} + \lambda w_i$$

Using gradient descent with a learning rate η, the weight update with L_2 regularization becomes:

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{\partial \mathcal{L}_{\text{reg}}}{\partial w_i} = w_i^{(t)} - \eta \left(\frac{\partial \mathcal{L}}{\partial w_i} + \lambda w_i^{(t)} \right) = (1 - \eta\lambda)\, w_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w_i}$$

Weight Decay is simply the strategy of not adding an L2 term to the loss function but instead multiplying the old weights $w_i^{(t)}$ by a factor of $1 - \eta\lambda$, as the equation above shows equivalency. Setting λ to 0 shows what would happen without L2 regularization.

3.2.9 Logged & Tracked Metrics

Weights and Biases (wandb) was used as the primary platform for logging all relevant training and experimental information, providing wide and modular tracking of both quantitative metrics and system performance data. The platform facilitated efficient monitoring and visualization throughout our development by logging such elements as:

1 **Quantitative Metrics:** Per step and epoch tracking of key performance indicators such as training and validation accuracy and loss.

2 **Printed Messages:** Systematic logging of custom messages generated during the training process, capturing important checkpoints and debugging information.

3 **Image Samples:** Recording visual data outputs, such as validation and training per epoch confusion matrices. Additionally, images were used to visualize 2D distance preserving PHATE[30] and UMAP[31] down-projections of both instance level and bag level latent code representations, the points were shaped and colored according to their label.

4 **Hardware Information:** Detailed tracking of of system hardware specifications, including CPU and GPU model, memory capacity and bandwidth helping with reproducibility and optimization.

5 **Resource Utilization:** Real-time logging hardware utilization like memory I/O, CPU usage and frequency, GPU and GRAM utilization to name a few.

6 **Training Status:** Current epoch, run health, creation and run time as well as checkpointing status.

7 **Results:** Final training outcomes, including best performing model checkpoints, test accuracies, recalls and precisions

8 **Model & Training Configurations:** Systematic logging of custom messages generated during the training process, capturing important checkpoints and debugging information.

3.2.10 Variability Control

To ensure reproducibility and consistency across experiments, a random seed was fixed for all random number generation processes involved in the training and evaluation of the model. A seed of 42 was set for Torch, Numpy, and Python's inbuilt random module. This ensures that operations relying on randomization yield the same results. Furthermore, other random processes, such as the order of the data points in the data loader, were randomized in a controlled way. Similarly, dimensionality reduction techniques were performed with a fixed seed.

PyTorch's cuDNN backend settings were not configured to maintain consistent behavior. Unlike the previous approaches, cudnn.deterministic was set to false, since its impact on computational performance was deemed too big.

3.2.11 Configuration Method

The experimental configuration for this project was managed using Facebook's Hydra library, which allows for the writing of modular and hierarchical configurations using YAML files. By using Hydra's inheritance functionality, a structured hierarchy of configuration files was built, allowing for clear organization and modification of parameters across different versions or variants of a base experiment. This integration into the pipeline is extensive, encompassing the configuration of all the mentioned hyperparameters in this section. It facilitates robust control over the experiments, without the need to change the codebase. This flexibility allows for streamlined and reproducible experimentation.

3.2.12 Hyperparameter Optimization

For hyperparameter optimization, the `wandb` sweeper was utilized to implement a Bayesian hyperparameter optimization strategy. This approach allows for efficient exploration of the hyperparameter space by iteratively refining the sampling strategy based on observed performance metrics, such as validation accuracy.

The optimization process begins with a prior probability distribution, $p(\theta)$, over a subset of the hyperparameter space, where θ represents the hyperparameters of the model. As training runs are executed with different hyperparameter settings, the performance outcomes are used to refine this distribution into a posterior probability distribution, $p(\theta \mid D)$, conditioned on the observed data D. This refinement is based on Bayes' theorem:

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)},$$

where:

- $p(\theta \mid D)$: The posterior distribution, reflecting updated beliefs about the likelihood of the hyperparameters given the observed data.
- $p(D \mid \theta)$: The likelihood, representing the probability of observing the data D given specific hyperparameters θ.
- $p(\theta)$: The prior distribution, encoding initial beliefs about the hyperparameter space.
- $p(D)$: The evidence or marginal likelihood, ensuring normalization.

In this project, the prior distribution $p(\theta)$ was defined based on general ranges and domain-specific heuristics. For multiplicative hyperparameters, such as learning rate or weight decay coefficients, log-scaled prior distributions were employed to better capture the variations in scale.

The `wandb` sweeper iteratively updates the sampling strategy using the posterior distribution, guiding the search toward hyperparameter regions with higher likelihoods of optimal performance. This efficiency is achieved by focusing on promising areas of the hyperparameter space, rather than sampling randomly or exhaustively.

To facilitate this process, wrapper scripts were developed to interface the `wandb` sweeping API with Hydra configuration files. This integration allowed the sweeper to access and optimize virtually any parameter in the model training pipeline, ensuring a seamless and flexible hyperparameter optimization process.

3.2.13 Compute Hardware

Experiments were conducted using a combination of personal computing resources and the high-performance computing (HPC) system at Helmholtz Munich. The primary development environment was set up on a laptop equipped with an Intel Core i7 processor and 16 GB of RAM. A configuration file set up to run the pipeline without a GPU was used there, which facilitated initial experimentation and debugging. Additionally, a desktop PC with an i7 processor, 32 GB of RAM, and a GTX1080 GPU was utilized for more resource-intensive tasks and GPU-related debugging. For large-scale experiments, the HPC system was leveraged, providing access to multiple GPU nodes with a wide range of GPUs and substantial memory resources. This heterogeneous computing environment enabled the execution of extensive hyperparameter sweeps and model training sessions while optimizing resource utilization across different platforms. The exact hardware specifications of the HPC are detailed here [32].

3.3 Topologically Regularized Multiple Instance Learning Approach

To measure the effectiveness of the different topological regularization strategies, we designed topologically regularized variants of the aforementioned baseline experiment. This section will describe the methodology used to produce these experiments. Unless explicitly stated, these experiments described here were run with the same methods outlined in Section 3.2.

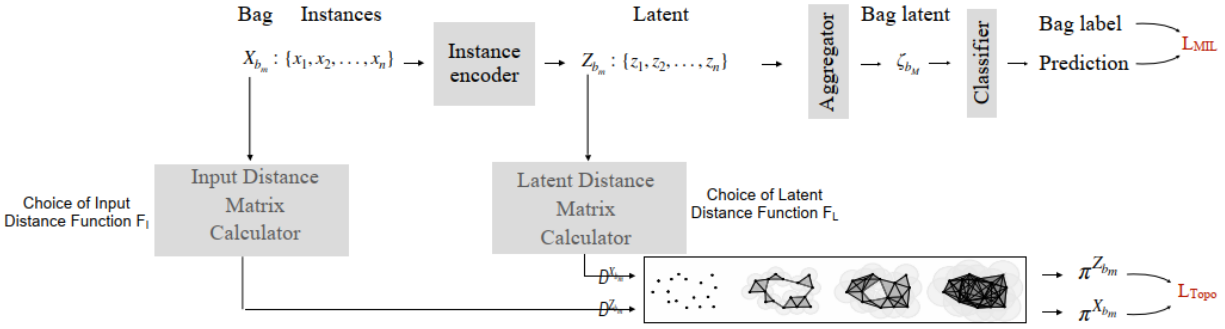


Figure 3.5 A complete overview of the topological regularization process: it starts with picking 2 distances functions F_I and F_L to transform the input and latent spaces into metrics spaces and calculate a distance matrix for each. Then a topological signature is calculated for each space, called $\pi^{X_{bm}}$ and $\pi^{Z_{bm}}$. Using $\pi^{X_{bm}}$ and $\pi^{Z_{bm}}$ we calculate a differentiable topological loss.

3.3.1 Data Preperation and Preporcessing

To apply topological regularization between two feature spaces, as discussed in Section 2.5, we must compute a topological encoding for each space. Creating this encoding requires the pairwise distances between all points in the space, also known as the "distance matrix." As described in Section 2.5, this matrix is constructed by calculating distances across all points in the dataset using a specified distance function.

In this study, we explored five different datasets, each potentially combined with various sampling strategies, transformations, and variations, alongside several distance functions, some of which are parametric. Each unique combination of dataset and distance function influences the resulting topological encoding, ultimately impacting the regularization applied during training. Given the computational intensity of certain distance matrix construction methods, we found that real-time calculation during training was infeasible in some cases, significantly slowing down training or even making it impossible.

To overcome this limitation, we chose to precalculate the necessary distance matrices before the start of training. This collection of precomputed data, referred to as the "topological dataset," is generated once per topology configuration and is stored for future use. To create a topological dataset, one requires 2 things:

1 A fully configured dataset which takes an index as an input and outputs a bag-like structures with at least 2 instances inside. This dataset should have all its configurations set, like sampling strategy, augmentations and transformations.

2 A fully configured distance function d that takes 2 instances and outputs a positive scalar. The other 2 criteria defining a distance function are required as well, namely $d(x,x) = 0$ for all possible values of x and $d(x,y) = d(y,x)$

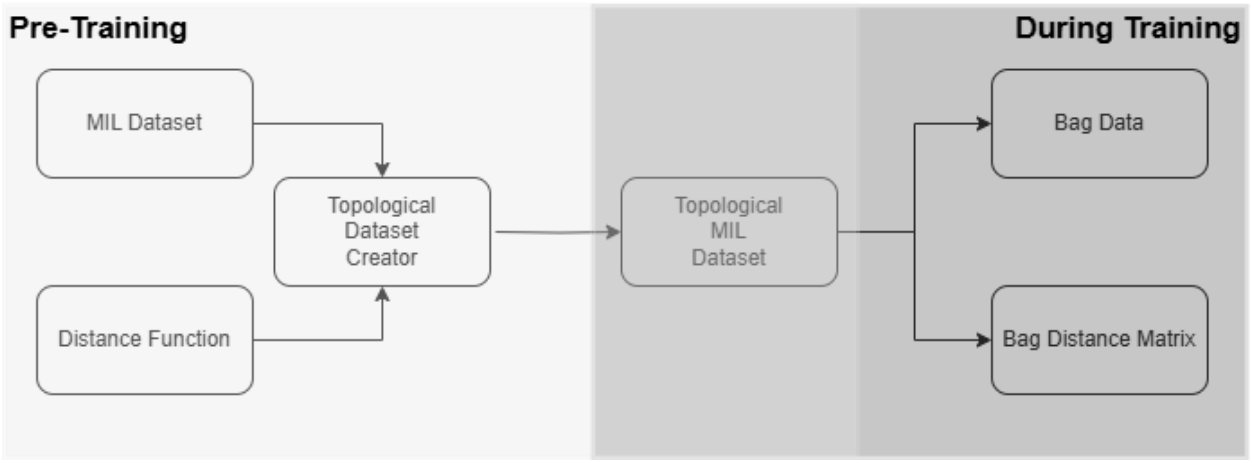


Figure 3.6 This figure illustrates how topological datasets are handled in our pipeline. The output of a topological set is bag data, a set of n image, while the bag distance matrix is a nxn symmetric zero-diagonal square matrix, where $a_{i,j}$ is the distance between the i^{th} and j^{th} image in the bag.

Each topological dataset is associated with a unique hash based on relevant experimental parameters, ensuring that previously computed datasets are reused if the settings remain the same, while new configurations prompt a fresh calculation.

Furthermore, the topological dataset maintains the order of instances within each bag, preserving consistency between the distance matrices of the input space and the latent space. To avoid dimensionality or scale variations in the distance function from impacting the topology calculations, we normalize all distance matrices within the topological dataset. Once generated, this dataset is appended to the standard training dataset of the base model, allowing the topological regularizer to access precomputed distance matrices for efficient, differentiable loss calculation during training.

This approach to precalculation and careful dataset management enables the topological regularization process to operate seamlessly, ensuring that training remains efficient without compromising the complexity of the topological encoding.

3.3.2 Distance Functions

Topological regularization experiments were implemented with the distances bellow for comparison.

Minkowski Distances

The Minkowski distances are the most well known family of distance functions, they are defined as such:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

this distance is called the Euclidean distance when p=2 and is called Manhattan distance when p=1.

Random Convolutions Distance

Convolutional neural networks (CNNs) have been an essential component of the success of deep networks in computer vision applications. Due to their local extraction properties, CNNs provide a strong inductive bias for working with image spaces. The following work demonstrates that even randomly initialized convolutional neural networks can serve as rich feature extractors [33][34]. In line with this approach [12], we use random convolutions to transform our input into a lower-dimensional space, which is then flattened, and the Manhattan distance is computed, as seen in this equation:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\text{flatten}(F(\mathbf{x}_i)) - \text{flatten}F(\mathbf{x}_j))\|_1$$

Where F represents a randomly initialized three-layer CNN that employs ReLU activations between the convolutional layers. The architecture features a kernel size of 4, with the number of channels progressively increasing from 1 or 3 to 12, then to 24, and finally to 48.

Learned Perceptual Distance

In the work of [35], a similarity metric was developed based on the latent space embeddings of images, demonstrating that VGG networks [36] effectively predict human perceptual similarity. The authors provided evidence that their metric outperforms traditional handcrafted feature extraction methods. Similarly, we adopt the Learned Perceptual Image Patch Similarity (LPIPS) metric [35] as an inter-image distance function, following the approach in [12]. For further details on the implementation of this distance metric, the reader is referred to the original work.

Manifold Learning Distances

For our experiments, we used UMAP (Uniform Manifold Approximation and Projection) and PHATE [30] (Potential of Heat-diffusion for Affinity-based Transition Embedding) as geodesic distance functions. UMAP [31] is a dimensionality reduction technique that preserves both local and global structures by learning a low-dimensional embedding that respects manifold structure. PHATE, on the other hand, excels at capturing continuous data trajectories and complex relationships by modeling data with heat diffusion processes.

One advantage of these geodesic distance functions is that they are quasi-parametric: the distance between two points is influenced not only by those two points but also by the structure of the entire input space. This feature allows us to group not just from a single bag, but from multiple bags when calculating an embedding, resulting in intra-instance distances that respect a manifold learned from a larger number of points. To calculate the distance matrix for bag B_i, while factoring in g - 1 other bags:

$$\mathbf{B} = \bigcup_{k \in I_j} \mathbf{B}_k$$

where $\mathbf{B}_k$ represents the set of input instances for the k-th bag, I_j is a set of indices containing i and g - 1 other bag indices.

A joint embedding function f is applied to all instances in $\mathbf{B}$:

$$\mathbf{Z} = f(\mathbf{B}) \in \mathbb{R}^{N \times d}$$

where $\mathbf{Z}$ is the lower-dimensional embedding, with N as the total number of instances across g bags, and d the embedding dimension.

The distance matrix D_i of bag B_i ,is generated from the embedding Z as such:

$$D_i = \|\mathbf{z}_m - \mathbf{z}_n\|_2, \quad \mathbf{z}_m, \mathbf{z}_n \in \mathbf{Z}_i$$

where $\mathbf{Z}_i \subset \mathbf{Z}$ is the subset of the embedding corresponding to the instances in B_i.

This produces a distance matrix D_i for each bag individually, capturing the intra-bag distances in the learned lower-dimensional space.

These methods are also known to be noise-resistant in pairwise distances, providing stable embeddings even in noisy data environments. Each method has hyperparameters that control aspects such as neighborhood size and diffusion rate, allowing fine-tuning of the embedding's sensitivity to both local and global structure.

Due to their computational complexity, we precomputed these geodesic learning distances before the main training phase. Similar to the topological datasets described in Section 3.3.1, we hash and save these embeddings for reuse across different runs. For consistency, we applied a constant downprojection dimension across all embeddings and grouped bags in sets of 10 to optimize pairwise distance calculations. This preprocessing ensured stable and computationally manageable embeddings throughout our experiments.

Foundation Model Generated Distance

We implemented a distance function derived from the output of a foundation model called DinoBLOOM [27]. DinoBLOOM is a self-supervised contrastive model trained exclusively on single-cell hematological data, using a tailored DINOv2 [37] pipeline. This model was developed using data from 13 hematology datasets, resulting in embeddings that are well-suited for capturing hematological patterns.

DinoBLOOM is available in four versions of varying model sizes; for computational efficiency, we used the smallest version in our experiments, which outputs a 384-dimensional feature vector per single-cell image. By leveraging this model, we obtained distances that not only reflect direct feature differences but also capture insights from the extensive training data, providing robust representations of hematological variations.

To calculate the distance between two single-cell images, we perform a forward pass on both images, then use the resulting vectorized embeddings to define a distance function. In our experiments, we explored multiple distance metrics, including those described in Sections 3.3.2, 3.3.2, as well as cosine similarity.

$$d(\text{image}_1, \text{image}_2) = f(\text{embedding}_{\text{DinoBLOOM}}(\text{image}_1), \text{embedding}_{\text{DinoBLOOM}}(\text{image}_2))$$

Where f can be one of the aforementioned distance functions.

Figure 3.7 This is a 2D embedding of Dinobloom-B features of over 80,000 cells. On the image extremities we can see the cells that belong to certain Leukemia subtypes highlighted. This figure was taken from [3].

Cubical Complex Distance

A cubical complex is a mathematical structure used to analyze shapes and spaces by decomposing them into connected "cubes" of varying dimensions. For 2D images, this means representing the image as a collection of connected pixels (0-dimensional points), edges (1-dimensional lines between pixels), and squares (2-dimensional areas formed by pixels). Constructing a cubical complex on an image captures both local and global spatial patterns, including connectivity and topological features, such as holes and connected components. This is particularly useful for defining distance functions that take into account the underlying topological properties of images, which simpler pixel-wise comparisons might overlook [38].

Similar to persistent homology, cubical complexes are tracked as a threshold value changes. In this context, the filtration is defined by setting pixels below the threshold to zero and those above it to one, which results in a persistence diagram for each degree of topological feature, with features "born" and "died" at specific brightness thresholds. In our application, we use the GUDHI [39] package to compute the persistence diagrams.

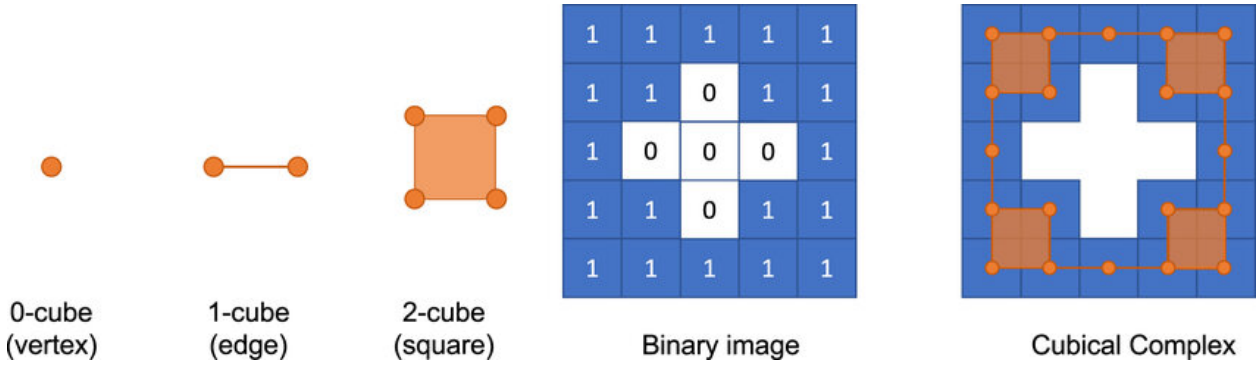


Figure 3.8 This figure illustrates how the cubical complex features are calculated on an image. The first three images show the 0^{th}, 1^{st}, and 2^{nd} order topological features. Notice that the 2^{nd} order features differ from the 2^{nd} order features of triangle-based topology used in regularization. The fourth object represents the image **after** the filtration is applied, and the fifth object shows the cubical complexes calculated for this particular threshold.This image was taken from [38]

Comparing the resultant persistence diagrams to produce a loss or distance between two images, as done in [40], allows us to calculate the bag distance matrix. Their approach involves using the Earth Mover's Distance (EMD) on the cubical complex persistence diagrams of the two images to calculate a loss across input images. This approach is summarized as follows:

Given a cubical complex C derived from an image I, the persistence diagram $\mathrm{PD}(C)$ is calculated by tracking homological features across a filtration:

$$\mathrm{PD}(C) = \{(b_i, d_i) \mid i \in \{1, \ldots, n\}\}$$

where b_i and d_i represent the birth and death times of the i-th feature in the cubical complex.

For two persistence diagrams $\mathrm{PD}(C_1)$ and $\mathrm{PD}(C_2)$, the Wasserstein distance W_p (with p-norm) is defined as:

$$W_p(\mathrm{PD}(C_1), \mathrm{PD}(C_2)) = \left(\inf_{\gamma:\mathrm{PD}(C_1)\to\mathrm{PD}(C_2)} \sum_{x\in\mathrm{PD}(C_1)} \|x - \gamma(x)\|^p \right)^{\frac{1}{p}}$$

where γ is a bijection between the points in $\mathrm{PD}(C_1)$ and $\mathrm{PD}(C_2)$ that minimizes the total p-th power of the distances between corresponding points.

The Wasserstein distance was chosen here because we require a distance that compares two distributions with different number of samples. Thus the Wasserstein distance is used to compare the persistence diagrams produced by two images.

3.3.3 Topological Loss

This work was initially inspired by the paper Topological Autoencoders [15] by M. Moor et al. In that work, an autoencoder is implemented with a special loss function containing both a reconstruction term and a topological loss term. A novel topological loss function was developed by Moor et al. for this purpose. In this work, we implement the loss function proposed in [15] and also introduce some new approaches.

Topological Autoencoder Approach

In the approach implemented in [15], the topological regularization proceeds as follows:

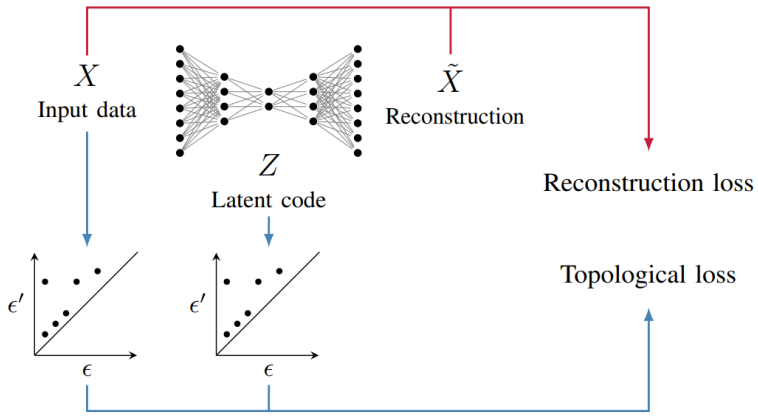


Figure 3.9 In the Topological Autoencoders approach a topological signature is calculated at the autoencoder latent space and at the input space, this topological signature is encoded as a persistence diagram, this figure was taken from [15].

1- The 0-order topological features (connectivity) of both input and latent space are calculated, however, the end result of this calculation is not the persistence diagram, but the persistence edges. Persistence edges are defined as edges in the point cloud which led to a change in its 0 order topological features, or in other words caused a 0 order feature to be born or to die. Calculating for and saving the persistence edges is a way to encode the 0 order topology across scales. It is not difficult to show that one can deterministically construct the 0 order components at any scale, given the persistence pair, additionally, one can also show that any point cloud of n points, has exactly n-1 persistence pairs to encode the 0 order topology for all scales.

2- Once the persistence pairs are calculated, we have a compact representation for the 0-order topology for both spaces, now it is time to calculate a loss. In their work Moor et al. formulated 3 variants of the same approach, here only the most successful will be discussed. With the persistence edges at hand, the topological differentiable loss of one space on the other is formulated as such:

Let,

$$E^1_{\text{persistent}} = \{e_i = (v^i_j, v^i_k) \mid 1 \leq i \leq n-1,\ 1 \leq j, j \leq k,\ k \leq n\}$$

Where:

- $E^1_{\text{persistent}}$ is the ordered set of persistent edges generated by space S1.
- $b > a \Leftrightarrow D_1(v^a_j, v^a_k) < D_1(v^b_j, v^b_k)$ where D_1 is the distance matrix of S1.
- e_i represents the i-th persistence edge.
- v_j and v_k are the vertices of the i-th edge.
- n is the total number of vertices.

Then the loss is calculated using the following equation:

$$L^{1\text{ on }2}_{\text{Topo}} = \sum_{e_i \text{ in } E^1_{\text{persistent}}} \left(D_1(v^i_j, v^i_k) - D_2(v^i_j, v^i_k)\right)^2$$

3- Now we add $L^{1\text{ on }2}_{\text{Topo}}$ and $L^{2\text{ on }1}_{\text{Topo}}$ to obtain $L^{\text{total}}_{\text{Topo}}$ with is differentiable with respect to the model parameters.

Our Topological Autoencoder Amendments

In this work, we propose 2 amendments to the Moor et al. differentiable topological loss approaches.

1- The first modification to the approach is to ensure that the topological encoding of both spaces are scaled to a common standard. In the literature when comparing the persistent homology of spaces the scale dimension is always scaled so it ranges from 0 to 1. This is intuitively clear, as a cube and scaled up or down version of it should have the same persistent homology. This fact was overlooked in the previous loss function, the modification looks as such:

$$L_{\text{Topo}}^{1 \text{ on } 2} = \sum_{e_i \text{ in } E^1_{\text{persistent}}} \left(\frac{D_1(v_j^i, v_k^i)}{\max(D_1)} - \frac{D_2(v_j^i, v_k^i)}{\max(D_2)} \right)^2$$

Where $\max(D_1)$ and $\max(D_2)$ are non-gradient tracking scalars, representing the highest value pairwise distance entry in in space 1 and 2 respectively.

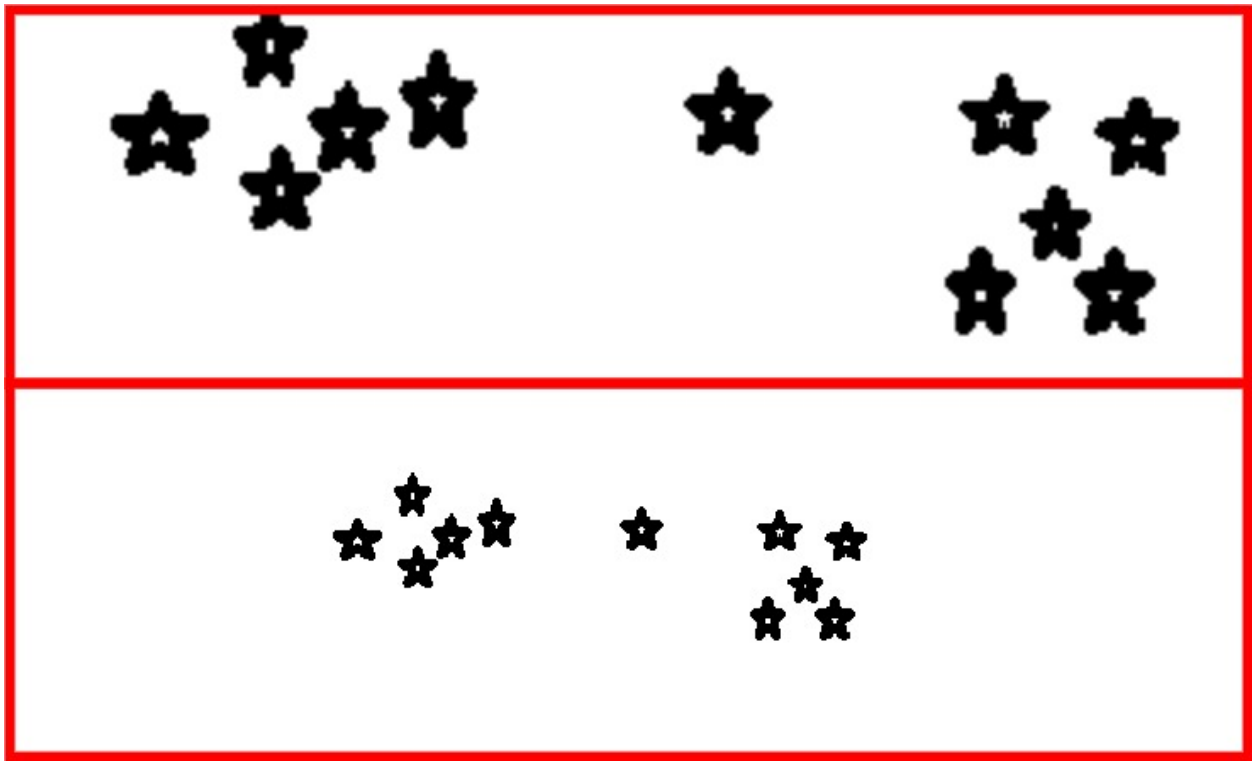


Figure 3.10 Two spaces defined by 2 pointclouds, one is a scaled version of the other, theoretically the topological loss between them should be 0, but it is not. Both spaces have the same persistence edges, but have different distances between them. Applying the topological loss defined in [15] would yield a nonzero loss, which is erroneous.

2- The second amendment is that the loss if formulated as such:

$$L_{\text{Topo}}^{1 \text{ on } 2} = \sum_{e_i \text{ in } E^1_{\text{persistent}}} \left(D_1(v_j^i, v_k^i) - D_2(h_{S2}(v_j^i, v_k^i)) \right)^2$$

where h_{S2} is a function that takes as input the indices (v_j^i, v_k^i) and searches space 2 for the shortest possible edge that places v_j^i and v_k^i in the same connected component. The idea behind this, is that topologically speaking when 2 components are connected it should not matter what edge connected them, as long as these 2 components are merged at the same scale. The idea is illustrated in the figure below.

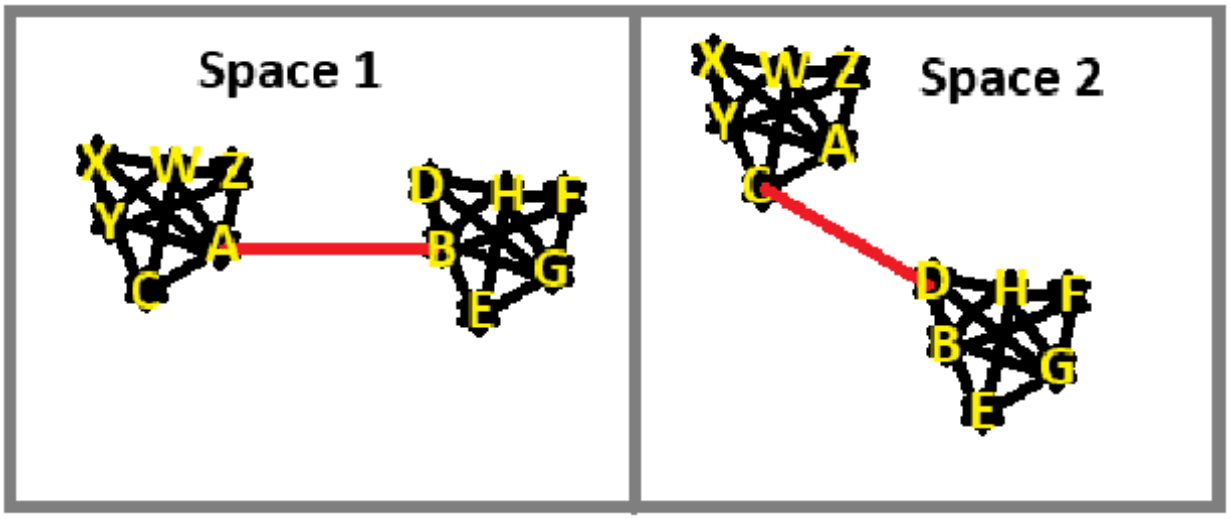


Figure 3.11 Assuming edge AB in space 1 has the same scale as edge CD in space 2, then there should be no 0 order topological difference between these 2 spaces, hence it does not matter which edge connected these components. Instead what matters is at which scale they were connected. In this example $h_{S1}(A, B) = h_{S2}(C, D)$, so if these 2 clusters joined at a smaller scale in space 1 than 2, then edge CD in space 2 needs to shrink and not edge AB in space 2. In other words, one should be able to rotate the clusters around their centers without impacting the topological loss.

For the amended approach, we developed a custom zero-order topological signature calculator. This modification was crucial for extending the functionality of the existing topological computation tools, enabling us to handle multi-scale encoding and compute the associated loss more efficiently. Our custom implementation simplified navigation across varying scales, ensuring that topological information was accurately captured and utilized within our framework. For a detailed understanding of the implementation, we refer the reader to the code repository. A key feature of our approach is the ability of the topological encoder to determine which persistent pair connects two queried points. This function is described in the formulation above as h_{S2}.

3.3.4 Topological Loss vs MIL Loss Balancing

In the topologically regularized experiments, we introduce an additional hyperparameter, λ, which controls the balance between the topological loss and the MIL classification loss. This λ weighting factor adjusts the contribution of the topological loss relative to the classification objective. Unlike SGD, where scaling factors might interact with the learning rate, λ here only affects the gradient's composition, as we use the Adam optimizer for all experiments. Specifically, λ modulates the portion of the gradient attributed to the topological loss without impacting the learning rate directly. The loss is formulated as such:

$$L_{\text{total}} = L_{\text{MIL}} + \lambda \cdot L_{\text{topo}}$$

Multiple custom schedulers were implemented to control λ throughout the training,namely:

- **Constant Scheduler** has a constant weighting throughout training.
- **Exponential Scheduler** has an exponentially increasing or decreasing weight w.r.t to the epoch.
- **Trigger Scheduler** has two weighting values, high and low, it switches between states based on configurable boolean function that has access to all the training metrics being logged to wandb.
- **Match MIL and Topo Loss** calculates the weight such that the ratio of the MIL and Topology loss contributions to the gradient are some configurable value.

3.3.5 Logged & Tracked Metrics

Weights and Biases was was also used to log the relevant metrics to our experiments. In addition to the values mentioned in 3.2.9, we monitored the training and validation topological loss.

3.4 Distance Function Evaluation Methods

Different distance functions 3.3.2 are proposed in this work, this section covers the evaluation approaches and criteria upon which distance functions are rated.

3.4.1 Instance Level Distance Metrics

Several of the proposed distance functions (see Section 3.3.2) are parametric, necessitating lightweight heuristic evaluations to compare the quality of the different approaches. To facilitate these comparisons, computationally efficient metrics were developed to guide further, more computationally intensive evaluations.

Instance-level distance metrics are used to evaluate image distance functions based on instance-level labels. To compute an instance-level metric, a fully labeled image dataset and a distance function are required. These metrics provide a score that reflects how effectively the distance function distinguishes data points according to their labels. While these instance-level metrics are well-established in the fields of data science and machine learning, they have been slightly modified in this work to assess the quality of the distance function itself, rather than the embedding strategy or approach.

Average and Per Class Inter to Intra Class Distance Ratio: The inter-to-intra class distance ratio is a simple metric that calculates the expected ratio of the distance between two points within the same class to the distance between two points from different classes. This metric reflects the global effectiveness of the distance function by assessing the overall separation between classes. Additionally, it can be evaluated on a per-class basis to measure how well the distance function distinguishes between points within each individual class.

$$R_{\text{inter-intra}}(i) = \frac{\frac{1}{|X_i|(|X_i|-1)} \sum_{\substack{x,y \in X_i \\ x \neq y}} d(x,y)}{\frac{1}{C-1} \sum_{\substack{j=1 \\ j \neq i}}^{C} \frac{1}{|X_i||X_j|} \sum_{x \in X_i} \sum_{y \in X_j} d(x,y)}$$

$$R^{\text{overall}}_{\text{inter-intra}} = \frac{\frac{1}{C} \sum_{i=1}^{C} \frac{1}{|X_i|(|X_i|-1)} \sum_{\substack{x,y \in X_i \\ x \neq y}} d(x,y)}{\frac{1}{C(C-1)} \sum_{i=1}^{C} \sum_{\substack{j=1 \\ j \neq i}}^{C} \frac{1}{|X_i||X_j|} \sum_{x \in X_i} \sum_{y \in X_j} d(x,y)}$$

Where d is the distance function and X_i is the set of data points whose labels are i and C is the class size.

Inter and Intra Class Distance Matrix Average and Standard Deviation: The inter and intra class distance average and standard deviation matrix is a metric that comes in the form of two square matrices of size equal to the class count. Each entry in the normalized average matrix represents the average pairwise distance between samples in each class, similarly the standard deviation matrix represents the expected variance when comparing images from similar or differing classes. These matrices are a good way of visualizing the global separability of the data points by the distance function, as well as giving an indication on the certainty of the separation.

$$D^{\text{avg}}_{\text{intra}}(i,i) = \frac{1}{|X_i|(|X_i|-1)} \sum_{\substack{x,y \in X_i \\ x \neq y}} d(x,y) \quad D^{\text{std}}_{\text{intra}}(i,i) = \sqrt{\frac{1}{|X_i|(|X_i|-1)} \sum_{\substack{x,y \in X_i \\ x \neq y}} \left(d(x,y) - D^{\text{avg}}_{\text{intra}}(i,i)\right)^2}$$

$$D^{\text{avg}}_{\text{inter}}(i,j) = \frac{1}{|X_i||X_j|} \sum_{x \in X_i} \sum_{y \in X_j} d(x,y) \quad D^{\text{std}}_{\text{inter}}(i,j) = \sqrt{\frac{1}{|X_i||X_j|} \sum_{x \in X_i} \sum_{y \in X_j} \left(d(x,y) - D^{\text{avg}}_{\text{inter}}(i,j)\right)^2}$$

Where d is the distance function and X_i is the set of data points whose labels are i.

Triplet Loss: Triplet loss is a well-known loss function in machine learning. It takes three data points as input: an anchor, a positive sample, and a negative sample. As the name implies, the positive sample is in the same class as the anchor, while the negative sample is from a different class. The triplet loss is evaluated by comparing the distance between the anchor and the positive sample with the distance between the anchor and the negative sample, as shown in the equation:

$$\mathcal{L} = \sum_{i=1}^{N} \left[d(x_i^a, x_i^p) - d(x_i^a, x_i^n) + \alpha \right]_+$$

Where α alpha is the margin and the upper indices a, b and n represent the anchor, positive and negative points respectively and d represents the distance function being tested.

Silhouette Score: The Silhouette score is typically a metric used to measure the quality of clustering algorithms. It assesses cohesion, which is how similar an object is to its own cluster, and compares it to separation, which is how similar it is to the nearest cluster different from its own. Typically, the similarity is calculated based on Euclidean distance, so the metric effectively evaluates the embedding method rather than the distance function.

In our case, we do it differently: by fixing the embeddings to be the images, we can change the distance function defining similarity. This way, the Silhouette score reflects the performance of the distance function instead of the embedding algorithm. The Silhouette score is determined using this equation:

$$s = \frac{1}{N} \sum_{i=1}^{N} \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Where a(i) is the average distance between a point i and all other points in the same class and b(i) is the average distance between i and all points in the nearest different class.

K-Nearest Neighbors Classification Accuracy, Precision and Recall: K-Nearest Neighbors is a supervised learning algorithm. Typically, it uses Euclidean distance to find the K nearest points to the point to be classified, and based on the majority label of the K neighbors, a classification is determined for the new point. KNN is one of the simplest classification algorithms, relying on one hyperparameter, K, to create a classifier. In our case, we replace the Euclidean distance with the distance function to be tested, so that the accuracy, precision, and recall metrics of the classifier reflect the quality of the distance function. In our experiments we used k=3.

Leave One out cross validation K- Nearest Neighbors Accuracy, Precision and Recall: Leave one out cross validation of the KNN classifier is a technique where the point to be predicted is left out the learning process, thus preventing a data leak between testing and training. The result of this is a more accurate all be it more pessimistic quantification of the quality of the distance function. In our experiments we used k=3.

3.4.2 Data Preparation and Preprocessing

For the heuristic evaluation of our distance functions, we utilized 5 diverse datasets that cover a broad range of domains and applications:

- **MNIST** [13] is a widely recognized dataset of handwritten single digits. It consists of 70,000 annotated grayscale images. The images are 28x28 making MNSIT images relatively small dimensional objects compared hematological images.
- **FashionMNSIT** [14] has a similar composition as MNIST, containing also 70,000 images of 28x28 images divided in 10 classes of different clothing articles. While similar in structure to MNIST, FashionMNIST offers a more complex computer vision task due to the high variability in appearance of items in the same class, thus requiring better distance functions.
- **CIFAR-10** [41] is a image dataset, consisting of 60,000 32x32 RGB images of animals and transportation vehicles. CIFAR is the most complex of the mentioned datasets, involving very fine differences between classes that are often hard for even humans to distinguish.

- **ACEVEDO** [42] also known as BloodMNIST, is a hematology dataset containing 17,000 labeled single blood cell images, taken from only healthy people. The images are 360x363 pixels originally, however, in practice, we found some images in the dataset that were in fact 360x360, for that reason everything was resized to 360x360. All images in ACEVEDO fall into 8 classes; neutrophils, eosinophils, basophils, lymphocytes, monocytes, immature granulocytes (promyelocytes, myelocytes, and metamyelocytes), erythroblasts and platelets or thrombocytes.
- **AML Cytomorphology MLL Helmholtz** [29] dataset is the dataset used for the training of SCEMILA 3.2.1. Although primarily a MIL dataset with labels provided only at the bag or patient level, we were able to obtain a fully annotated subset of it (<4%) for evaluating our distance functions using our instance level tests. These instance level labels belong to one of 25 very imbalanced classes, with "ambiguous" and "other" classes. For clarity and interpretability of the results "ambiguous" and "other" classes, as well as any class containing less than 30 members were removed, leaving us with 13 final classes.

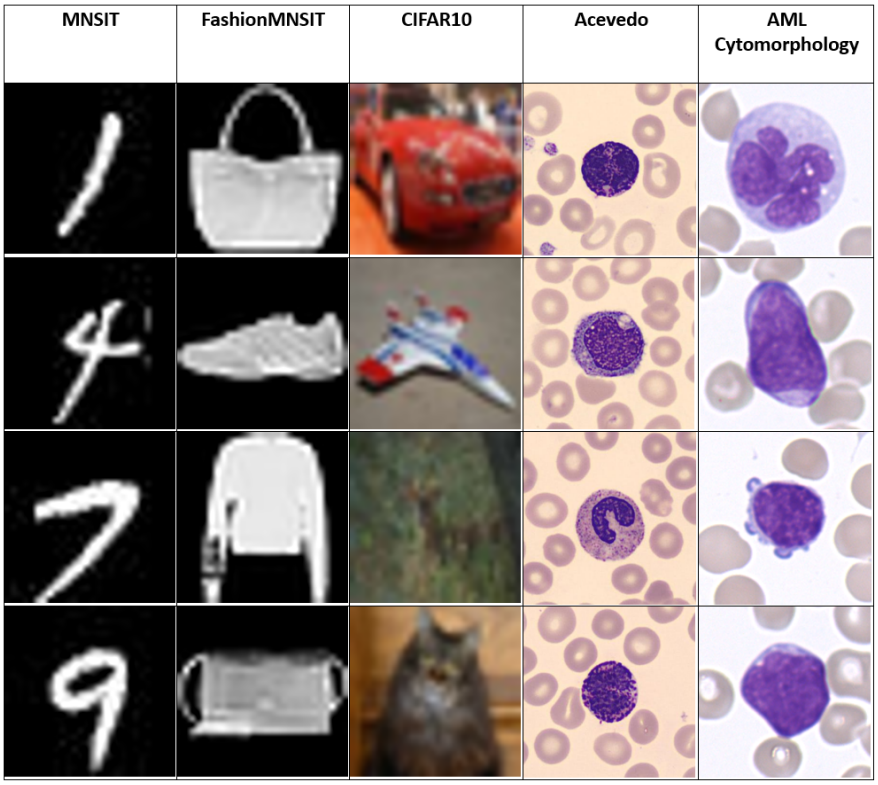


Figure 3.12 Examples of samples from the datasets used in the experiments, showcasing diverse image data types, including handwritten digits (MNIST), fashion items (FashionMNIST), natural images (CIFAR10), blood smear cell images (Acevedo), and cytomorphology of Acute Myeloid Leukemia (AML).

3.4.3 Experiment Configuration Method

Similarly to the machine learning configuration process in 3.2.11, we used a persistent experiment configuration approach. From these configuration files one can set:

- Which dataset to use for the experiment
- Sampling strategy to use on the dataset for metric evaluation
- Augmentation strategy settings
- Distance function to use and its associated settings

3.4.4 Results Logging Method

We used a PDF creation package in Python called ReportLab to visualize these results. This was done to save the experiment results along with the settings that generated them, as well as visuals and graphs to aid with visualization and analysis. Additionally, samples of the transformed or augmented images are visualized in the PDF output.

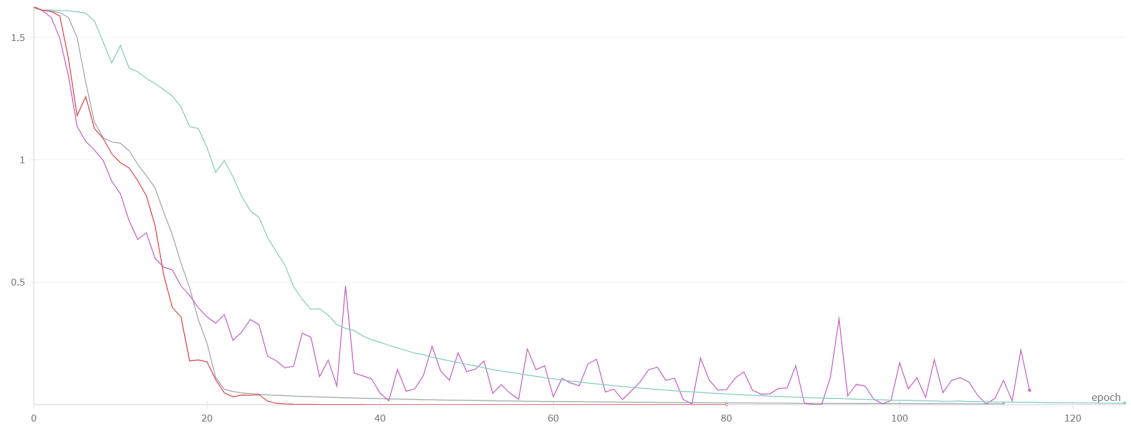
4 Experiments and Results

4.1 Baseline Multiple Instance Learning Model

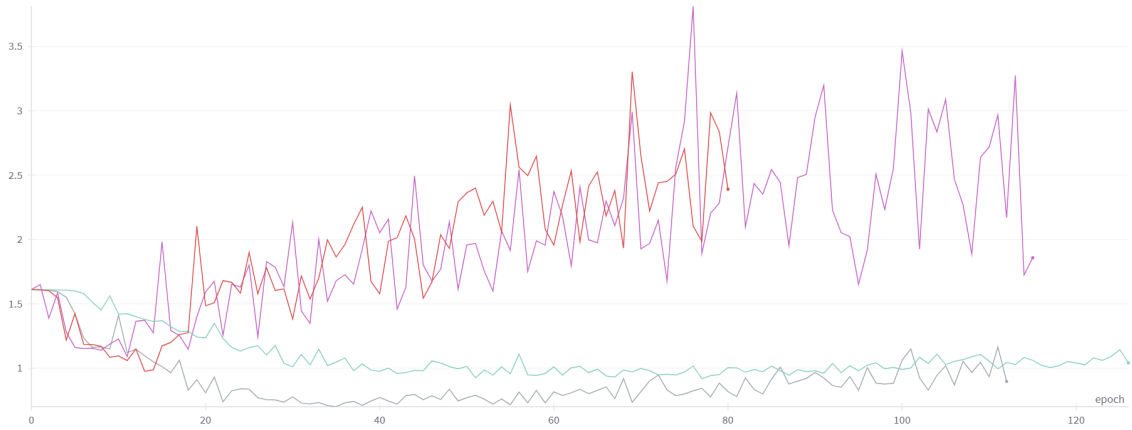
As mentioned previously, we are operating in a data scarce setting, where overfitting is a constant problem. To setup a stable baseline to test our topological regularization approaches, we tested and added many training methods detailed in 3.2, achieving a 67%±6.6 test accuracy. Detailed here are the experiments conducted to reach this performance.

4.1.1 Optimizing Regularization Techniques

Since overfitting is a big risk in this problem setting, special attention was paid to the regularization approaches. The risk of overfitting was apparent throughout the work as exemplified by the following examples:



(a) Training loss over epochs.

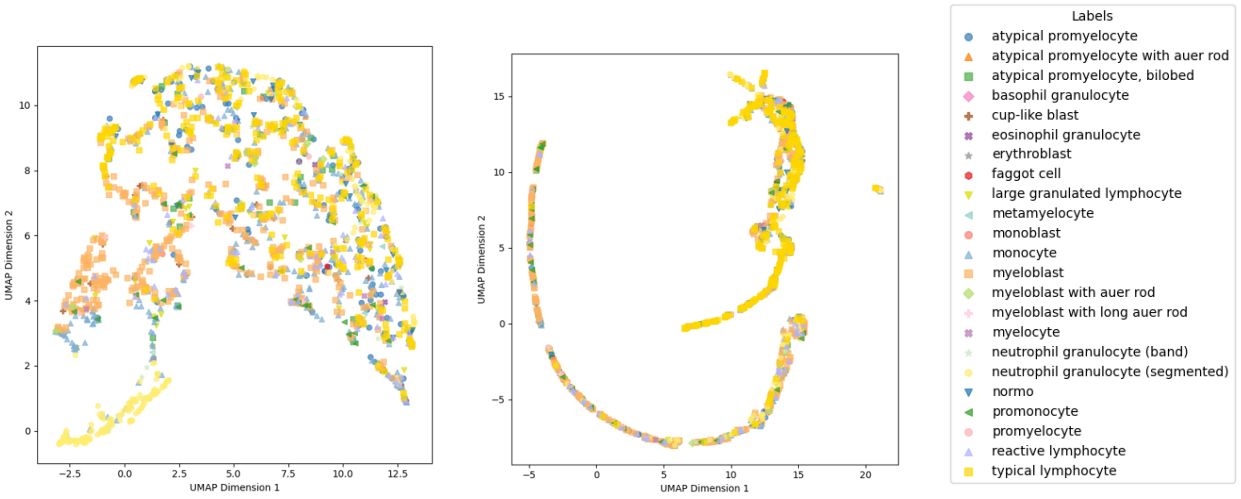


(b) Validation loss over epochs.

Figure 4.1 This figure shows the generalization problems of our baseline model in this problem setting. This illustrates the generalization problem, where we find that the training and validation losses are not well coupled together. The differently colored curves are runs on the same splits of the data but with different regularization settings. The red and purple curves have unstable generalization, while grey and green have stable but bad generalization.

The figure above 4.1 shows 4 runs with different regularization settings, each having generalization problems. Their respective training loss converges to near 0, while their validation losses are either unstable or inadequately converging. We observed that the model is always able to fit the data, even under extreme regularization, which is explained by data scarcity, meaning our model can always memorize the data because it is too little. This pose a difficulty, as we were not able to clearly identify helpful regularization techniques, as usually the training curve's convergence is very helpful in diagnosing over and under fitting.

The negative effects of bad visualization are also perceivable on the encoder output level. Improper regularization cause then encoder to lose it's differentiation capabilities, causing different cell classes to collapse on each other as seen in the figure below.



(a) Good Regularization including data augmentation, dropout, L2 and batching

(b) Bad Regularization including only drouput and L2

(c) Single Cell Labels

Figure 4.2 This figure illustrates how the encoder loses its expressiveness when trained under bad regularization.

In order to find adequate regularization settings, we set up a sweep optimizing over 6 regularization techniques namely, L2 weight decay, dropout, image augmentation, model parameter reduction, label smoothing and gradient accumulation. The sweeper was configured to optimize the expected value of the test accuracy, using a Bayesian optimization method. Please find the sweep configuration file, detailing the parameter ranges and sweep strategy here.

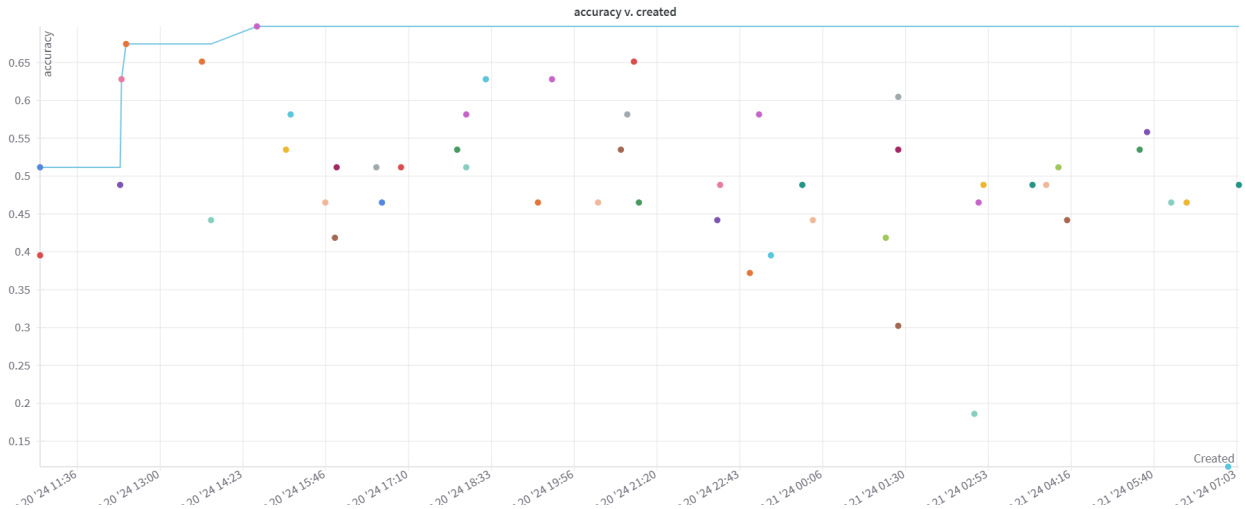


Figure 4.3 Accuracy of different runs with different regularization settings. The search for hyperparameters here is guided by Bayesian Optimization. The best of these was picked and its hyper parameters were tuned further.

The results for our optimal regularization settings showed, that model regularization (i.e. L2, dropout and model reduction) is not as effective as data regularization (i.e. data augmentation and label smoothing), as increasing the model regularization simply caused the validation loss to converge at higher values while the training remained unaffected. On the other hand regularizing the model via the data showed big generalization improvements. The final regularization settings can be found in this configuration file.

After fixing the regularization hyper parameters settings, another sweep was run over the learning rate scheduler only, controlling such parameters, like patience, LR drop factor and initial learning rate of the "ReduceLROnPlateaux". This step showed very marginal improvement, as the results showed that the performance was stable for a large range of the LR hyperparameter settings. The final learning rate settings can be found in this configuration file.

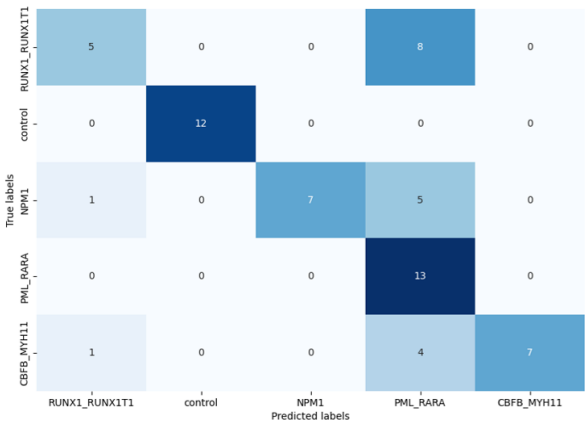
Finally a sweep was run over such settings that control computation time and speed. This included such settings as float precision, max epoch and early stopping. While early stopping is considered a regularization method, we observed that our model results were not affected with early stopping (within a reasonable range).

metric	Split **1**	Split **2**	Split **3**	Split **4**	**All** Splits
accuracy	0.729 ± 0.042	0.627 ± 0.055	0.659 ± 0.074	0.655 ± 0.042	0.668 ± 0.067
auroc	0.923 ± 0.016	0.857 ± 0.017	0.871 ± 0.046	0.897 ± 0.018	0.887 ± 0.037
f1 macro	0.701 ± 0.044	0.588 ± 0.056	0.624 ± 0.077	0.619 ± 0.041	0.633 ± 0.070
precision macro	0.715 ± 0.043	0.611 ± 0.051	0.658 ± 0.060	0.657 ± 0.047	0.660 ± 0.063
recall macro	0.710 ± 0.043	0.595 ± 0.055	0.634 ± 0.071	0.629 ± 0.040	0.642 ± 0.068

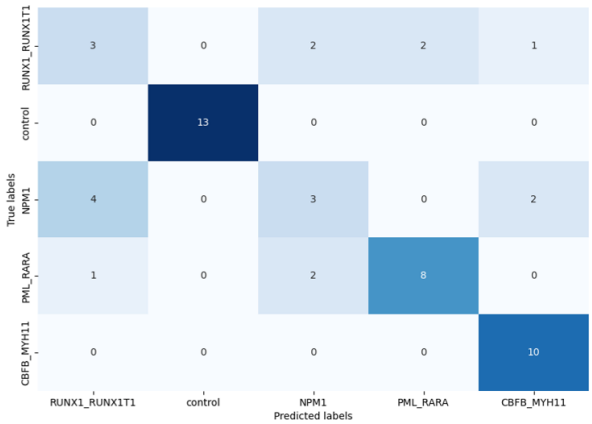
Table 4.1 This table shows the average performance of our baseline on a per split level and on the whole experiment level.

Performing ANOVA analysis on the results of the different splits from table 4.1, we could **not** reject the hypothesis that the generating averages of each split are the same. This means that there is a statistically significant dependence of the results on the split, demonstrating a strong dependence on the input data, which is consistent with data scarce applications.

Examining the training, validation, and testing confusion matrices, we observe that our encoder performs well in answering the question: "Does this patient have leukemia?" However, accurately identifying specific subtypes remains a significant challenge.



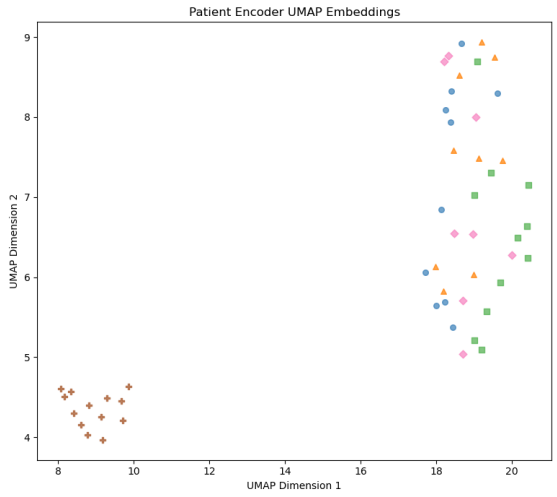
(a) Validation confusion matrix of the baseline.



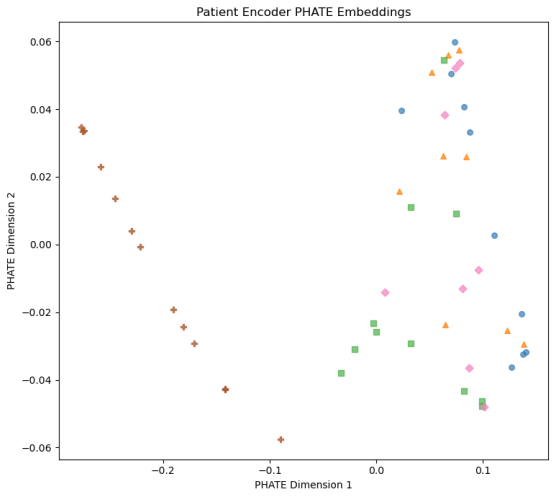
(b) Testing confusion matrix of the baseline.

Figure 4.4 This figure shows the testing and validation confusion matrices of our baseline. The second row and column in the confusion matrix indicate that there are no false positives or negatives in the binary classification task (has leukemia or not). However, this is not the case for leukemia subtypes.

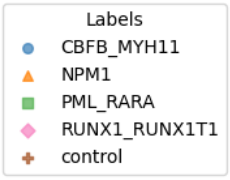
This implies our data is sufficient for the binary classification problem of detecting leukemia, however our model struggles to differentiate subtypes. This is also reflected in the bag latent space visualization.



(a) UMAP downprojection of patient-level latent space.



(b) PHATE downprojection of patient-level latent space.



(c) Legend for the latent space visualizations.

Figure 4.5 This figure shows the 2D downprojections of our test set bag-level latent space embeddings. The UMAP and PHATE plots (top) highlight that the binary leukemia detection problem is easier for the classifier than the subtype classification problem. The legend (bottom) provides the label color mapping.

4.2 Distance Function Evaluation

We conducted multiple experiments to evaluate and optimize our different distance functions approaches.

4.2.1 Baseline Experiment

To set a baseline we conducted the following experiment; Across 5 different data sets we sample 100 points from every class and evaluated a suit of distance functions on the metrics detailed in 3.4.1, over the samples. The evaluations of three of the most indicative metrics can be seen below.

Table 4.2 Baseline Intra-to-Inter Class Distance Overall Ratio (↓)

Method	SCEMILA	ACEVEDO	FMNSIT	CIFAR	MNSIT
Isomap	0.802	0.727	0.580	0.924	0.623
Manhattan	0.910	0.938	0.668	0.967	0.786
PCA	0.814	0.754	0.605	0.953	0.688
PHATE	**0.474**	**0.463**	**0.346**	0.918	**0.409**
TSNE	0.966	0.982	0.956	0.971	0.984
UMAP	0.601	0.570	0.440	**0.919**	0.518
cubical_complex_distance	0.935	0.864	0.963	0.971	0.686
euclidean_distance	0.914	0.926	0.744	0.970	0.851

Table 4.2 shows that UMAP and PHATE are best performers across all datasets in the inter to intra distance metric. This metric is good at measuring the global class distance quality, unlike KNNs it favors a single cluster per class type, where as KNNs don't mind it as long as the different clusters are of the same class. As expected the MNIST and FashionMNIST data sets performed the best due to their simplicity, with the blood datasets following shortly behind and with CIFAR performing the worst.

Table 4.3 k-NN Accuracy (↑)

Method	SCEMILA	ACEVEDO	FMNSIT	CIFAR	MNSIT
Isomap	0.58 ± 0.49	0.65 ± 0.48	0.78 ± 0.41	0.46 ± 0.50	0.81 ± 0.39
Manhattan	0.54 ± 0.50	0.63 ± 0.48	0.81 ± 0.39	0.53 ± 0.50	0.82 ± 0.38
PCA	0.62 ± 0.49	0.65 ± 0.48	0.78 ± 0.41	0.47 ± 0.50	**0.88 ± 0.33**
PHATE	0.62 ± 0.49	0.56 ± 0.50	0.72 ± 0.45	0.47 ± 0.50	0.72 ± 0.45
TSNE	**0.65 ± 0.48**	0.43 ± 0.49	0.66 ± 0.47	0.45 ± 0.50	0.66 ± 0.47
UMAP	0.59 ± 0.49	0.64 ± 0.48	0.79 ± 0.41	0.41 ± 0.49	0.83 ± 0.38
cubical_complex_distance	0.60 ± 0.49	**0.78 ± 0.42**	0.46 ± 0.50	**0.76 ± 0.43**	0.62 ± 0.49
euclidean_distance	0.58 ± 0.49	0.60 ± 0.49	**0.83 ± 0.38**	0.50 ± 0.50	0.82 ± 0.38

Table 4.3 summarizes the performance of the distance function, using the k-NN classification accuracy. While it favors some distances like PCA and Cubical Complex over others, no clear winners come out. We also note that UMAP and Phate, while not the best, are still competitive in performance.

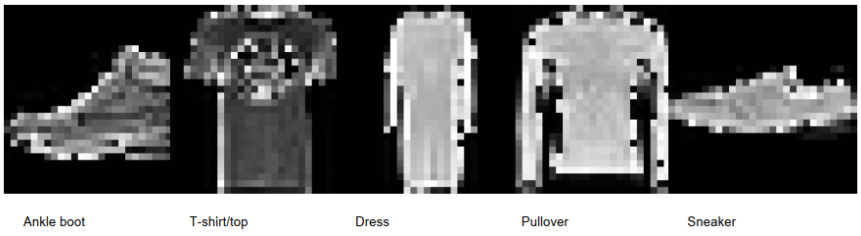
Table 4.4 Baseline **Silhouette** Score (↑)

Method	SCEMILA	ACEVEDO	FMNSIT	CIFAR	MNSIT
Isomap	-0.066	-0.013	0.1	-0.124	0.117
Manhattan	**-0.03**	-0.006	0.057	-0.095	0.048
PCA	-0.06	**0.01**	**0.105**	-0.122	0.118
PHATE	-0.179	-0.175	-0.004	-0.298	-0.048
TSNE	-0.133	-0.231	-0.184	-0.22	-0.253
UMAP	-0.087	-0.061	0.098	-0.202	**0.144**
cubical_complex_distance	-0.068	-0.023	-0.175	-0.138	-0.179
euclidean_distance	-0.025	0	-0.175	**-0.093**	0.048

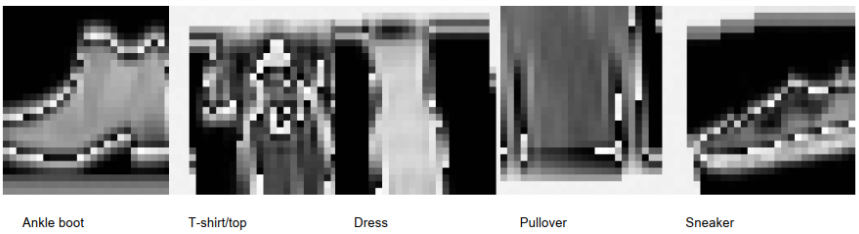
In the table above 4.4 we see the silhouette scores, where the linear decomposition method PCA is best performing across the 5 datasets. From these base experiments we deduce that UMAP and PHATE are the best performing when consider the over performance over all the metrics. Additionally we consider the euclidean distance and PCA as overall good performers for this non-augmented experiment.

4.2.2 Augmentation Sensitivity Experiment

As way to test the distance function robustness to perturbations, we ran experiments comparing performances of augmented vs non-augmented datasets. As a first experiment we wanted to quantify the euclidean distance's robustness to augmentation, as that is the distance function used in [15] and its supplementary work [12], on which this work builds on. As detailed in the baseline experiments we ran the same experiment setup on twin datasets, one augmented the other not. All datasets have been augmented in a tailored manner as to preserve image meaning and integrity as shown bellow.

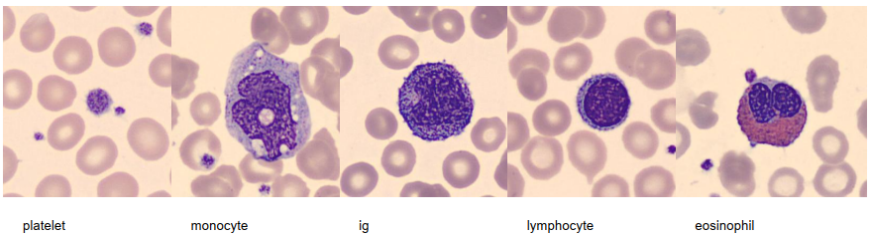


(a) FashionMNIST without augmentation.

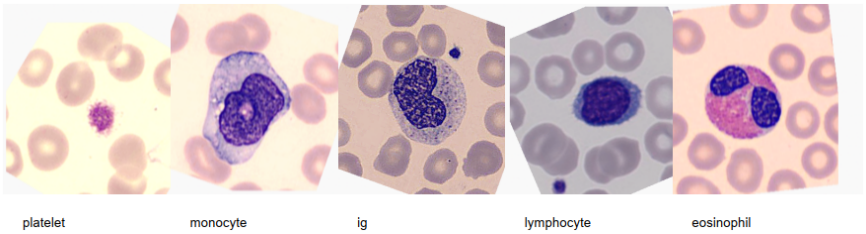


(b) FashionMNIST with augmentation.

Figure 4.6 Comparison of FashionMNIST images with and without augmentation. The first row represents original images, followed by their augmented versions.



(a) ACEVEDO without augmentation.



(b) ACEVEDO with augmentation.

Figure 4.7 Comparison of ACEVEDO images with and without augmentation. The first row represents original images, followed by their augmented versions.

Dataset	KNN Accuracy ↑		Inter-to-Intra Distance Ratio↓	
	w/o Augmentations	with Augmentations	w/o Augmentations	with Augmentations
MNIST	0.82	0.52	0.851	0.985
FMNSIT	0.83	0.60	0.744	0.927
CIFAR	0.50	0.41	0.970	0.995
ACEVEDO	0.60	0.537	0.926	0.966
SCEMILA	0.58	0.42	0.914	0.978

Table 4.5 Comparison of metrics with and without augmentations across datasets.

In table 4.5, we observe that the Euclidean distance evaluated using two metrics across the five datasets shows a clear deterioration in quality upon augmentation. However, we note that the Euclidean distance retains some descriptive capabilities, implying that a certain level of generalization is still possible with this distance function. Additionally, we see a strong drop in performance for MNIST and FMNIST, which consist of very centered and similarly oriented images, while more diverse datasets do not suffer as much from this degradation.

To further investigate the sensitivity of our distance functions, we devised an experiment in which we decompose the augmentation process into its constituent components—namely, Gaussian noise and translational shifting—to examine their individual contributions to the observed performance degradation.degradation.

Table 4.6 Leave One Out KNN **Accuracy** (LOOCV) ↑ Across Different Augmentation Types on **FashionMNSIT**.

Distance Function	All	Translation Aug	Rotation Aug	Gaussian Noise Aug	None
Isomap	0.14	0.25	0.54	0.64	0.62
Manhattan	0.42	0.33	0.52	0.61	0.71
PCA	0.23	0.24	0.42	0.58	0.63
PHATE	0.05	0.22	0.37	0.66	0.60
TSNE	0.28	0.28	0.40	0.66	0.68
UMAP	0.22	0.30	0.41	0.68	0.63
Cubical Complex Distance	0.16	0.17	0.19	0.23	0.19
Euclidean Distance	0.29	0.28	0.51	0.60	0.61

We observed in the table above 4.6 and consistently across other datasets and metrics that the ranking of impact across the studied augmentation is in decreasing order translation, rotation, guassian noise. The above explains why topological regularization using euclidean distance works well on datasets like MNIST and FashionMNIST [12], as they have very minor translational and rotational variation across them, generally well centered and oriented, however, the same can not be said about datasets like ACEVEDO or SCEMILA.

Table 4.7 Leave One Out KNN **Accuracy** ↑ (LOOCV) on **ACEVEDO** dataset.

Input Type	Distance Function	All	Translation Aug	Rotation Aug	Gaussian Noise Aug	None
dino	Isomap	0.875	0.925	0.887	0.887	0.887
	Manhattan	0.812	0.838	0.825	0.700	0.775
	PCA	0.863	0.838	0.787	0.800	0.850
	PHATE	0.887	0.925	0.838	0.875	0.912
	TSNE	0.912	0.900	0.900	0.900	0.938
	UMAP	0.887	0.887	0.938	0.912	0.850
	Euclidean Distance	0.812	0.887	0.725	0.900	0.825
image	Isomap	0.212	0.237	0.312	0.338	0.400
	Manhattan	0.188	0.237	0.362	0.300	0.300
	PCA	0.200	0.287	0.362	0.388	0.487
	PHATE	0.163	0.212	0.338	0.263	0.350
	TSNE	0.225	0.287	0.300	0.375	0.263
	UMAP	0.188	0.312	0.388	0.525	0.425
	Cubical Complex	0.237	0.650	0.463	0.625	0.562
	Euclidean Distance	0.237	0.312	0.388	0.388	0.425

Table 4.8 Leave One Out KNN **Accuracy** ↑ (LOOCV) on **SCEMILA** dataset.

Input Type	Distance Function	All	Translation Aug	Rotation Aug	Gaussian Noise Aug	None
dino	Isomap	0.52	0.63	0.57	0.52	0.56
	Manhattan	0.58	0.51	0.62	0.63	0.56
	PCA	0.63	0.50	0.45	0.53	0.51
	PHATE	0.54	0.57	0.67	0.62	0.66
	TSNE	0.45	0.49	0.43	0.58	0.59
	UMAP	0.63	0.61	0.60	0.56	0.48
	Euclidean Distance	0.56	0.52	0.57	0.62	0.49
image	Isomap	0.13	0.10	0.19	0.32	0.27
	Manhattan	0.17	0.21	0.15	0.25	0.36
	PCA	0.13	0.16	0.32	0.18	0.39
	PHATE	0.10	0.21	0.36	0.22	0.36
	TSNE	0.17	0.16	0.20	0.32	0.19
	UMAP	0.15	0.21	0.28	0.30	0.33
	Cubical	0.22	0.33	0.27	0.28	0.26
	Euclidean Distance	0.13	0.21	0.34	0.26	0.36

The above tables 4.7 & 4.8 display the leave-one-out cross-validation accuracy of various distance function approaches applied to the ACEVEDO and SCEMILA datasets under different augmentation settings. When examining the experiments marked as image, where distances are calculated directly in the image space, we observe that the already poor performance without augmentations deteriorates further to being uninformative once augmentations are applied. This highlights the need for more robust distance functions to achieve generalizable topological regularization.

The entries marked as dino correspond to experiments performed on single-cell images after passing them through the foundation model DinoBloom [27]. Using DinoBloom, we aimed to generate better distance matrices for topological regularization. The tables confirm that DinoBloom produces good and stable performance across all augmentation settings. The discrepancy in performance between the SCEMILA and ACEVEDO datasets is attributed to their inherent differences: SCEMILA's fully labeled subset is highly imbalanced, whereas ACEVEDO is well-balanced and designed to be a cell-level labeled dataset. Below, we observe similar trends in the inter- to intra-class distance ratio metric.

4.2.3 Manifold Learning Distance Functions Experiments

We aimed to further investigate the effectiveness of manifold learning distance functions. First, we tested how the performance of these functions depends on the number of samples provided. Theoretically, increasing the number of samples should enable better evaluation metrics, as manifold learning methods can benefit from more data to model the underlying structure accurately.

If significant improvements are observed, this approach could be leveraged to compute bag distance matrices jointly with other bags, potentially improving the loss generated by these jointly calculated distances. To test this hypothesis, we conducted experiments where manifold learning methods were applied to increasingly larger input sizes. As a baseline for comparison, we included Euclidean distance to determine whether this approach offers any tangible benefits.

Table 4.9 Leave One Out KNN **Accuracy** ↑ (LOOCV) comparison of SCEMILA and ACEVEDO datasets using different distance functions and input sizes.

Dataset	Input Type	Distance Function	10 Images	20 Images	50 Images	100 Images
SCEMILA	dino	Isomap	0.45	0.44	**0.6**	**0.6**
		PCA	0.41	0.54	**0.56**	0.55
		PHATE	0.46	0.51	**0.61**	0.53
		TSNE	0.23	0.53	**0.54**	0.46
		UMAP	0.56	0.51	0.66	0.57
		euclidean_distance	0.57	0.57	0.61	0.62
	image	Isomap	0.1	**0.18**	0.09	0.11
		PCA	0.12	0.15	0.14	**0.17**
		PHATE	**0.22**	0.1	0.17	0.17
		TSNE	0.12	0.14	0.1	**0.17**
		UMAP	0.14	0.13	0.17	**0.18**
		euclidean_distance	0.13	0.2	0.13	0.19
ACEVEDO	dino	Isomap	0.65	0.775	0.85	**0.863**
		PCA	0.8	**0.838**	0.762	0.8
		PHATE	0.7	0.738	0.838	**0.925**
		TSNE	0.175	0.775	0.688	**0.875**
		UMAP	0.787	0.9	**0.912**	0.9
		euclidean_distance	0.85	0.887	0.787	0.812
	image	Isomap	0.175	0.212	**0.25**	0.212
		PCA	0.212	**0.287**	0.125	0.212
		PHATE	0.188	0.113	**0.212**	0.188
		TSNE	0.1	0.175	**0.225**	0.175
		UMAP	0.212	0.163	0.125	**0.263**
		euclidean_distance	0.2	0.338	0.237	0.325

Table 4.10 Inter-to-Intra Distance Ratio ↓ comparison of SCEMILA and ACEVEDO datasets using different distance functions and input sizes.

Dataset	Input Type	Distance Function	10 Images	20 Images	50 Images	100 Images
SCEMILA	dino	Isomap	0.725	0.653	**0.595**	0.617
		PCA	0.781	0.736	**0.701**	0.746
		PHATE	0.380	0.391	**0.375**	0.393
		TSNE	0.985	0.800	**0.740**	0.753
		UMAP	0.559	0.498	**0.283**	0.322
		euclidean_distance	0.931	0.928	0.924	**0.921**
	image	Isomap	0.971	**0.942**	0.983	0.954
		PCA	0.948	0.973	**0.945**	0.954
		PHATE	**0.838**	0.995	0.957	0.948
		TSNE	0.995	0.960	0.941	**0.939**
		UMAP	0.976	0.978	0.979	**0.913**
		euclidean_distance	0.978	0.972	0.985	0.983
ACEVEDO	dino	Isomap	0.610	0.565	0.533	**0.471**
		PCA	0.622	0.593	**0.591**	0.619
		PHATE	0.303	0.212	0.326	**0.245**
		TSNE	1.001	0.772	0.728	**0.460**
		UMAP	0.372	0.211	0.226	**0.185**
		euclidean_distance	0.868	0.850	0.858	0.857
	image	Isomap	0.972	1.000	**0.931**	0.951
		PCA	0.970	0.948	0.948	**0.941**
		PHATE	0.982	0.987	**0.949**	0.979
		TSNE	0.997	0.995	0.974	**0.967**
		UMAP	0.987	**0.955**	0.987	0.968
		euclidean_distance	0.966	0.970	0.966	0.968

Studying the results from the tables above 4.9 & 4.10, we observe similar trends across both datasets, suggesting that the metrics align. When encoding the images using DinoBloom, there is a clear advantage to embedding them jointly in increasingly larger batches. In the table, the bold values represent the best-performing results for a particular distance function approach, while the underlined metric indicates the top metric for each dataset and input type. While we observed improvements when applying these manifold methods to larger portions of small datasets, such as MNIST and FashionMNIST, we did not see significant gains for high-resolution blood cell datasets. The conclusion drawn from these results is that manifold learning on DinoBloom embeddings can provide notable benefits. This approach could be particularly useful in small-bag-size applications where topological regularization is desired. In such cases, other bags can be leveraged to construct reliable bag distance matrices, enabling more effective topological regularization.

4.2.4 Cubical Complex Approach Optimization

As detailed in 3.3.2, we employed the cubical complex loss as implemented in [40] as an inter-image distance. However, since the cubical complex implementation in that work was intended for 3D structures, we had to make some modifications to adapt and optimize it. In our case, the input is also 3D, but the third dimension represents color channels in the case of RGB datasets. This left us with three possible approaches:

1 **Treat Color Channels Separately:** This is the most straightforward strategy. The cubical complex of each channel is calculated for both images, and a loss is calculated across the equivalent channels in the input images using the cubical complex distance function. This approach is the most computationally demanding among the proposed methods, as it requires six cubical complex signature calculations and three Earth Mover's Distance (EMD) calculations, one for each channel.

2 **Join Channels' Output, Treating Them as One:** In this approach, we also calculate the cubical complex signature of each channel separately (implying six cubical complex calculations). However, instead of calculating the loss for each channel individually, we combine all the cubical complex birth-death features into a single persistence diagram and calculate the Earth Mover's Distance as if the features of all three channels (red, green, and blue) existed on one channel.

3 **Transform the Image to Grayscale Before Calculating Loss:** The simplest approach involves transforming the image to grayscale, reducing it to a single-channel input, and then calculating the cubical complex distance as we normally would. This involves converting both images to grayscale, computing their cubical complex signatures, and calculating the Wasserstein Distance between them. This is the computationally lightest method, requiring a grayscale conversion, two cubical complex signature calculations, and a single Wasserstein Distance calculation.

To measure the effectiveness of these approaches, we set up experiments on the CIFAR and ACEVEDO datasets. Using the same data sample, we calculated the cubical complex distance three times, once with each approach. The results are presented in the table below.

Table 4.11 Comparison of intra-to-inter-class distance ratio and LOOCV KNN accuracy for **CIFAR10** with different cubical complex calculation methods.

Augmentation	Distance Function	Intra-to-Inter Distance Ratio	LOOCV KNN Accuracy
all	euclidean_distance	0.985	0.14
	grayscale_cub_complex	**0.975**	0.11
	merged_pd_channels_cub_complex	0.980	**0.12**
	normal_cub_complex	0.980	**0.12**
none	euclidean_distance	0.937	0.18
	grayscale_cub_complex	0.962	**0.15**
	merged_pd_channels_cub_complex	**0.945**	**0.15**
	normal_cub_complex	0.949	0.13

Table 4.12 Comparison of intra-to-inter-class distance ratio and LOOCV KNN accuracy for **ACEVEDO** with different cubical complex calculation methods.

Augmentation	Distance Function	Intra-to-Inter Distance Ratio	LOOCV KNN Accuracy
all	euclidean_distance	0.978	0.237
	grayscale_cub_complex	**0.954**	**0.287**
	merged_pd_channels_cub_complex	0.962	0.200
	normal_cub_complex	0.969	0.237
none	euclidean_distance	0.924	0.425
	grayscale_cub_complex	**0.834**	**0.562**
	merged_pd_channels_cub_complex	0.850	**0.562**
	normal_cub_complex	0.843	**0.562**

From the tables 4.11 and 4.12, we observe that the cubical complex distance demonstrates consistent performance across different approaches. Consequently, we adopt the grayscale cubical complex approach, as it is computationally the lightest and yields the best performance based on our experiments.

4.3 Topologically Regularized Model

In this section, we present results from the topologically regularized multiple instance learning leukemia subtype detection model described in 3.3.

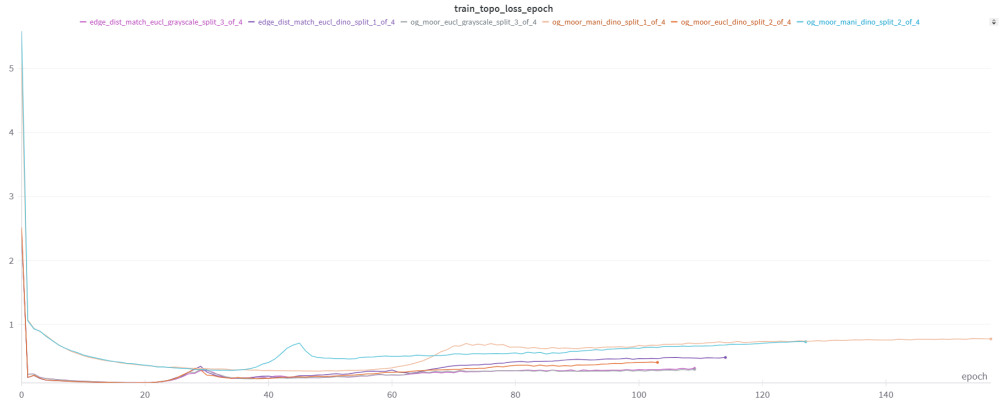
4.3.1 Baseline Experiment

As a baseline, we set up an experiment that inherited all hyperparameter configurations from the established baseline described in 3.2. To this, we added a constant topological loss weighting factor such that the average topological loss accounted for approximately 10% of the classification loss. Below, we provide various results related to these experiments.

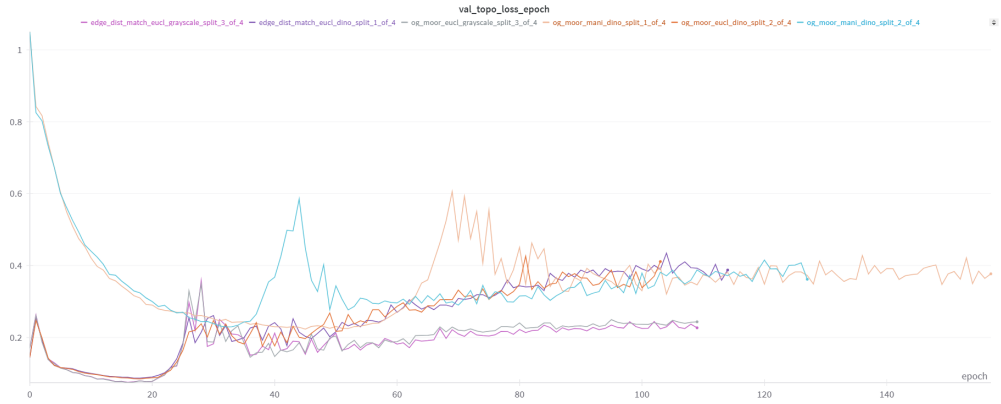
Table 4.13 Metrics comparison of topological regularization with different distance functions

Run Type	Accuracy	Recall Macro	Precision Macro	F1 Macro	AUROC
Original approach UMAP Grayscale	0.37 ± 0.10	0.34 ± 0.07	0.22 ± 0.10	0.24 ± 0.08	0.73 ± 0.04
Original approach UMAP Dino	0.43 ± 0.05	0.39 ± 0.04	0.27 ± 0.07	0.29 ± 0.04	0.74 ± 0.02
Original approach Eucl Grayscale	0.47 ± 0.05	0.43 ± 0.05	0.35 ± 0.08	0.36 ± 0.05	0.74 ± 0.02
Original approach Eucl Dino	0.47 ± 0.02	0.44 ± 0.03	0.37 ± 0.08	0.37 ± 0.06	0.74 ± 0.02
Amended approach Eucl Grayscale	0.43 ± 0.03	0.40 ± 0.03	0.29 ± 0.05	0.31 ± 0.04	0.74 ± 0.03
Amended approach Eucl Dino	0.46 ± 0.04	0.43 ± 0.05	0.34 ± 0.07	0.34 ± 0.04	0.73 ± 0.02
baseline	0.67 ± 0.07	0.64 ± 0.07	0.66 ± 0.06	0.63 ± 0.07	0.89 ± 0.04

The metrics above are collected from three k=4 cross-fold validation runs, except for the baseline, which was run five times. The results include both Euclidean and UMAP distance functions applied to Dinobloom and the image space. Additionally, we present results for the original algorithm proposed by Moor et al. (3.3.3) and for the amended approach detailed in this work (3.3.3). The results clearly favor the baseline approach without topological regularization, with the best-performing regularized approach scoring 20 points below the baseline's accuracy. Furthermore, no significant performance differences between the distance functions are observed. Using ANOVA, we could not reject the null hypothesis that the same Gaussian distribution generates the accuracies of the different distance function approaches. This suggests no significant performance variation among the approaches. Below, we provide examples of learning curves from some of the runs that contributed to the results shown in the table above.



(a) Training topological loss of some of the runs from Table 4.13.



(b) Validation topological loss of some of the runs from Table 4.13.

Figure 4.8 Learning curves showing training and validation loss from some of the runs in Table 4.13. The top figure corresponds to training topological loss, and the bottom figure corresponds to validation topological loss.

From the training curves above 4.8, we can see that while the validation loss is higher than the training loss, the curves still mirror each other in shape, implying some generalization of the topological signature to the rest of the dataset. However, when comparing the classification validation loss with the baseline, it becomes clear that topological regularization is negatively affecting the generalizability of the classification performance. To address this issue while keeping the topological weighting constant, we attempted sweeping over reasonable hyperparameter ranges but found no success. Based on our observations, the topological loss was primarily detrimental to the model's performance, particularly its generalization capabilities. This was evident as the topological loss only impacted the training loss of our model under very extreme hyperparameter configurations. Below, we include the validation classification loss for both the baseline and the topologically regularized approaches.

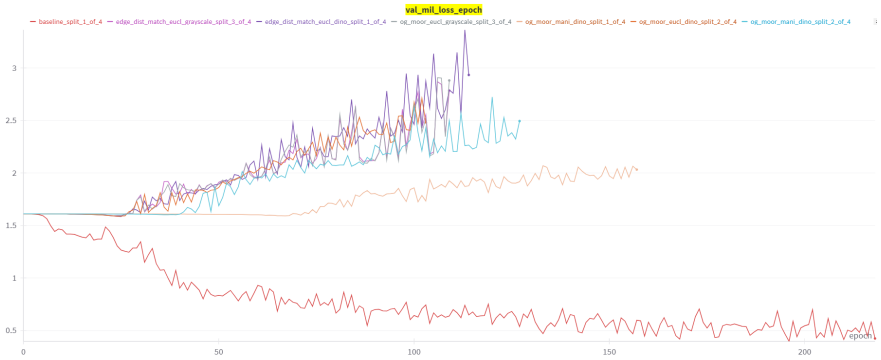
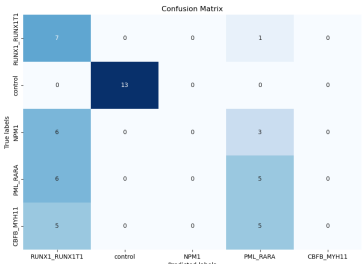
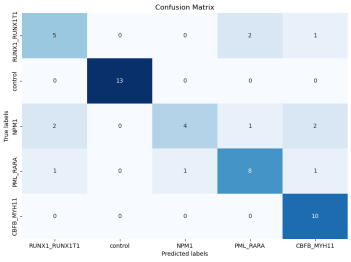


Figure 4.9 The figure above shows the baseline model (red curve), from which all topologically regularized approaches inherit their hyperparameters. The baseline achieves an acceptable convergence of validation loss. In contrast, the validation loss of the topologically regularized approaches converges to a value higher than that of the untrained network, indicating significant generalization challenges.

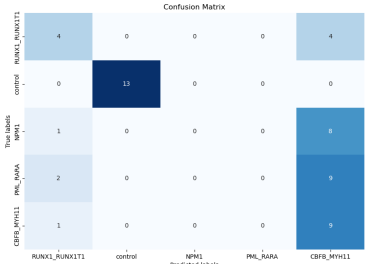
Further detailed investigation of the results, particularly the confusion matrices, show that the topologically regularized model retains the baseline's basic leukemia detection capabilities, however loses its leukemia subtype differentiation power, as seen bellow.



(a) Original approach with Euclidean DinoBloom distance.



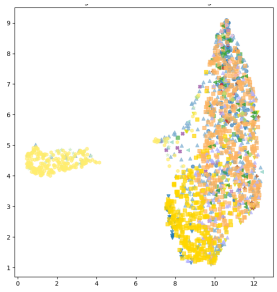
(b) Baseline model.



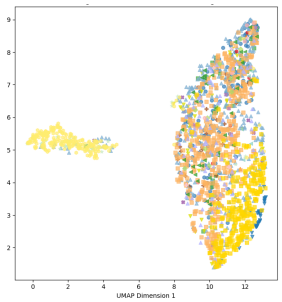
(c) Amended approach with Euclidean DinoBloom distance.

Figure 4.10 Confusion matrices for the baseline model (center) and comparison models (left and right). The baseline serves as a reference for evaluating the two topological regularization approaches. One can see that in all matrices the binary classification problem of leukemia detection is perfectly solved by all models, since the 2nd row and column contain only entries in the [2,2] index for all matrices. The labels are, top to bottom, left to right; **RUNX1, control, NPM1, PML_PARA** and **CBFB**.

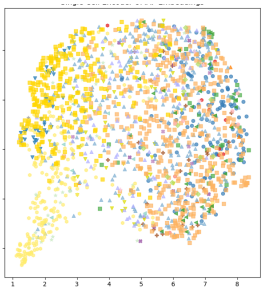
Despite the difficulty in generalizing, the approach is not without its successes. Observing the 2D embeddings of the single-cell space, we can see clear and distinct differences between the baseline, the topologically regularized model, and the various topological regularization approaches. Below 4.11, we include single-cell latent space visualizations of some of the distance functions alongside the baseline for comparison.



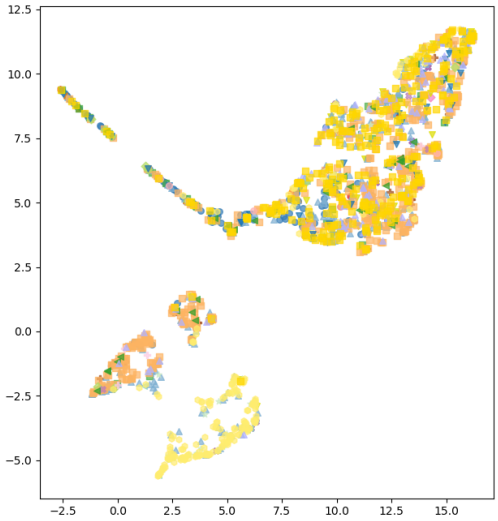
Original approach using Euclidean distance on Dinobloom latent space.



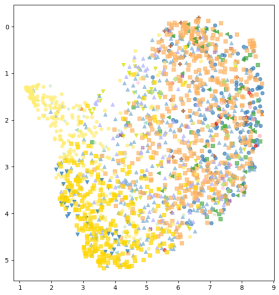
Modified approach using Euclidean distance on Dinobloom latent space.



Original approach using Euclidean distance on Grayscale image space.



Non-Topologically Regularized Baseline.



Modified approach using Euclidean distance on Grayscale image space.

Figure 4.11 Visualization of 2D Latent Space Embedding. The baseline is shown larger in the center, with alternative approaches stacked vertically on both sides.

4.3.2 Topological Regularization Loss Weighting Experiments

After obtaining the results of the baseline, we aimed to study the impact of the loss weighting factor in improving the results. To that end, we set up a series of experiments where we gradually varied the weight of topological regularization, or its contribution to the gradient updates, and observed the performance as we progressed. As we conducted our experiments, we noticed that performance degraded as we increased the effect of topological regularization and improved as we decreased it. Consequently, we continued to reduce the topological regularization until we obtained these results.

Method	Accuracy	Recall (macro)	Precision (macro)	F1 (macro)	AUROC
Image Input Experiment	0.66 ± 0.07	0.63 ± 0.07	0.65 ± 0.06	0.62 ± 0.07	0.87 ± 0.04
Topo Modified App UMAP Dino	0.57 ± 0.09	0.54 ± 0.09	0.57 ± 0.10	0.53 ± 0.10	0.85 ± 0.06
Topo Modified App Eucl Dino	0.63 ± 0.09	0.60 ± 0.09	0.62 ± 0.10	0.59 ± 0.10	0.85 ± 0.06
Topo Modified Eucl Gray	0.59 ± 0.08	0.56 ± 0.09	0.60 ± 0.11	0.55 ± 0.10	0.84 ± 0.05
Topo Modified Gray Image	0.64 ± 0.12	0.60 ± 0.13	0.61 ± 0.14	0.58 ± 0.14	0.87 ± 0.05

Table 4.14 Comparison of metrics across different experimental setups, but with low topological regularization weighting. Each value is represented as mean ± standard deviation.

The table above shows competitive results with the baseline, with one of the approaches outperforming the benchmark in the AUROC metric. However, as with the previous test set performance table, when performing ANOVA analysis on the accuracy metric of the above methods, one cannot dismiss the null hypothesis, which states that all approaches are generated by the same Gaussian distribution. From this, we deduce that we have reached a point in topological weighting where the topological loss no longer impacts training.

Examining the training and validation loss curves and confusion matrices, we observe that all these approaches deviate from each other in minor ways. If one inspects the single-cell latent spaces of these approaches, they also appear very similar and are not strongly differentiated, as shown in 4.11. This negative result is not proof of the approach's ineffectiveness, as this failure can likely be attributed to other factors that may be hindering the topological regularization.

4.3.3 Direct Dinobloom Experiments

After obtaining negative or non-positive results when regularizing with Dinobloom, we set out to test Dinobloom's effectiveness on this problem. To evaluate Dinobloom, we replaced our instance encoder (ResNet18) with the pretrained Dinobloom model while keeping all other hyperparameters and experimental parameters the same as the baseline. The test set result metrics can be seen below.

Method	Accuracy	Recall (macro)	Precision (macro)	F1 (macro)	AUROC
Image Input Experiment	0.66 ± 0.07	0.63 ± 0.07	0.65 ± 0.06	0.62 ± 0.07	0.87 ± 0.04
Dinobloom Input Experiment	0.83 ± 0.04	0.80 ± 0.04	0.83 ± 0.05	0.79 ± 0.05	0.97 ± 0.01

Table 4.15 Comparison of metrics across the baseline and the Dinobloom encoder model. Each value is represented as mean ± standard deviation.

We can see from the table above 4.15 that the Dinobloom model is indeed a robust and expressive single-cell feature extractor, providing better accuracy and performance stability than the baseline. Examining the latent space representation of Dinobloom 4.12 with respect to the fully labeled subset, we also observe significant positive patterns.

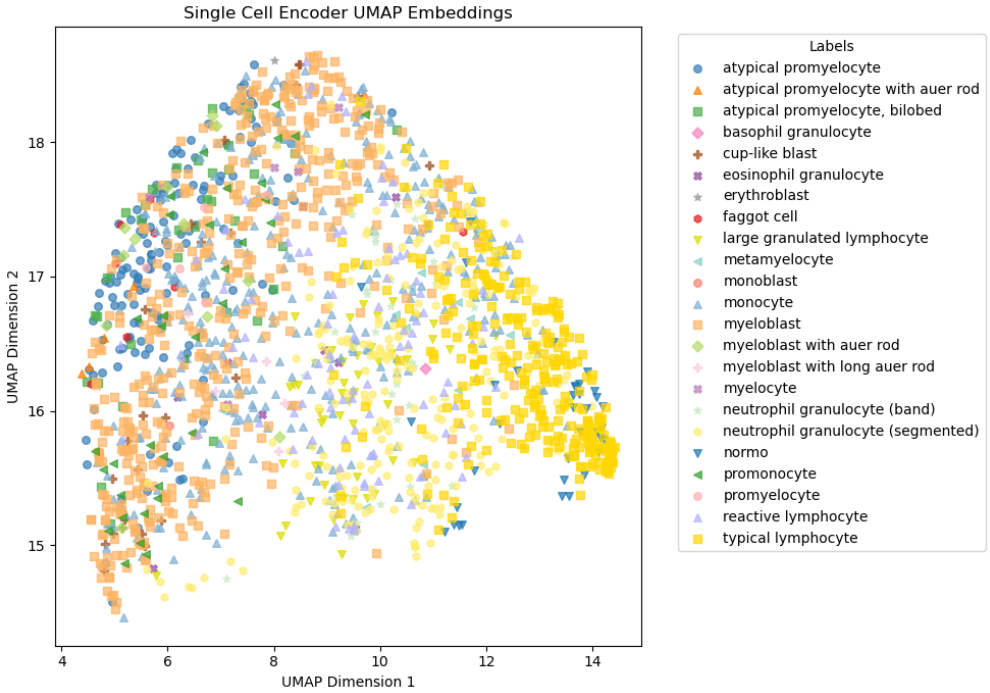


Figure 4.12 This figure shows 2 dimensional embedding of the Dinobloom small single cell encoder.

These results indicate that Dinobloom is a strong foundation model for the leukemia subtype detection problem, and our inability to improve on the baseline with topological regularization based on Dinobloom is not due to the inexpressiveness of our distance function. This leads us to believe that further investigations and adjustments to the topological regularization technique itself are needed to effectively leverage and match the topological signatures of the two spaces.

5 Conclusion

In this thesis, our contributions can be summarized into three key points:

1. We introduced metrics to quantify the quality of a distance function in image spaces and demonstrated their effectiveness.

2. We proposed a collection of distance functions better suited for single cell image spaces and quantified their quality using these metrics.

3. We developed a novel topological loss function, inspired by [15], to enhance computational efficiency and model performance.

The metrics introduced in the first contribution were applied to evaluate the proposed distance functions. Our results consistently highlighted the superiority of certain distances over others across different datasets and conditions. This indicates that there is room for improvement in topological regularization through better-designed distance functions tailored to metric spaces. As expected, we observed that leveraging foundation models significantly enhanced the quality and resilience of distance functions, particularly against augmentations. Additionally, manifold learning methods showed promise, suggesting the potential for creating more noise-resilient distance functions.

However, when testing the impact of these distance functions on the performance of topological regularization, we could not achieve statistically significant improvements. Regardless of the distance function employed, leukemia classification accuracy remained unaffected to a degree that could decisively prove the advantage of one distance function over another. Furthermore, none of the topological regularization approaches outperformed the baseline, which consistently delivered the best results—except in scenarios where Dinobloom was used directly as an encoder.

We believe a significant limitation in evaluating the efficacy of topological regularization in this context is data scarcity. The current training setup on AML data presents an ambiguous scenario where too many unknowns impede clear analysis and diagnosis of the observed training outcomes. To fully explore the potential of topological regularization as a method to combat data scarcity, future research should focus on artificially data-scarce scenarios. This controlled setting could allow for systematic development and testing, helping isolate the variables and better understand the true capabilities of topological regularization.

Additionally, another major limitation of the topological regularization research in this context is MIL. For future research, a classical model should be considered instead of a multiple instance learning model, where the topological loss is applied on a batch instead of a bag. This helps in limiting the sources of variance and uncertainty when developing and improve the topological regularization technique. We also propose that topological regularization be studied as a dimensionality reduction tool like PHATE and UMAP before it's effect on training models is explored. This is because we believe that even the amended version of the topological loss proposed in [15], the loss can still be improved to be more informative. Quick tests conducted by us using topological regularization loss as a guide to downproject embedding show that the current form of the loss is not very expressive, being easily surpassed by the likes of PHATE. By implementing it as dimensionality reduction tool we can quickly iterate on it improving and debugging implementation errors.

We would like to conclude that our findings indicate a strong effect of the metric space's distance function on the quality of the topological encoding and regularization. We proposed distance functions, metrics to judge them and topological loss to enforce them however our results have come out a mixture of positive, negative and inconclusive. The results are negative in the sense that no improvements were found, they

were positive in the sense that potentially promising observations were made relating the distance functions and performance and finally inconclusive since the reason for the lack of observable improvements is still not found.

Bibliography

[1] P. Rajpurkar et al. “AI in health and medicine”. In: *Nature medicine* 28.1 (2022), pp. 31–38.

[2] M. H. Harris et al. “Genetic testing in the diagnosis and biology of acute leukemia: 2017 society for hematopathology/European association for haematopathology workshop report”. In: *American Journal of Clinical Pathology* 152.3 (2019), pp. 322–346.

[3] M. Hehr et al. “Explainable AI identifies diagnostic cells of genetic AML subtypes”. In: *PLOS Digital Health* 2.3 (2023), e0000187.

[4] A. Sadafi et al. “Attention based Multiple Instance Learning for Classification of Blood Cell Disorders”. In: *CoRR* abs/2007.11641 (2020). arXiv: 2007.11641.

[5] S. Kazeminia et al. “Self-Supervised Multiple Instance Learning for Acute Myeloid Leukemia Classification”. In: *arXiv preprint arXiv:2403.05379* (2024).

[6] M. Waqas et al. “Exploring Multiple Instance Learning (MIL): A brief survey”. In: *Expert Systems with Applications* (2024), p. 123893.

[7] Y. Ohnuki, M. Akiyama, and Y. Sakakibara. “Deep learning of multimodal networks with topological regularization for drug repositioning”. In: *Journal of Cheminformatics* 16.1 (2024), p. 103.

[8] G. Quellec et al. “Multiple-Instance Learning for Medical Image and Video Analysis”. In: *IEEE Reviews in Biomedical Engineering* PP (Jan. 2017).

[9] W. Zhu et al. *PDL: Regularizing Multiple Instance Learning with Progressive Dropout Layers*. 2024. arXiv: 2308.10112 [cs.CV].

[10] Y. Zhang et al. *AEM: Attention Entropy Maximization for Multiple Instance Learning based Whole Slide Image Classification*. 2024. arXiv: 2406.15303 [cs.CV].

[11] F. Hensel, M. Moor, and B. Rieck. “A survey of topological machine learning methods”. In: *Frontiers in Artificial Intelligence* 4 (2021), p. 681108.

[12] M. Moor et al. “Challenging euclidean topological autoencoders”. In: *TDA {\&} Beyond*. 2020.

[13] L. Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[14] H. Xiao, K. Rasul, and R. Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG].

[15] M. Moor et al. “Topological autoencoders”. In: *International conference on machine learning*. PMLR. 2020, pp. 7045–7054.

[16] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. “Solving the multiple instance problem with axis-parallel rectangles”. In: *Artificial intelligence* 89.1-2 (1997), pp. 31–71.

[17] L. Melas-Kyriazi. *The Mathematical Foundations of Manifold Learning*. 2020. arXiv: 2011.01307 [cs.LG].

[18] G. Carlsson et al. “On the local behavior of spaces of natural images”. In: *International journal of computer vision* 76 (2008), pp. 1–12.

[19] University of Waterloo. *What is Topology?* Accessed: 2024-11-21. n.d.

[20] A. D. T. Barba. *Topological Data Analysis with Persistent Homology*. Ed. by medium.com. Aug. 2024.

[21] F. Fan. *Computing Topological Features of Data and Shapes*. The Ohio State University, 2013.

[22] M. Mehrabbeik, S. Jafari, and M. Perc. “Synchronization in simplicial complexes of memristive Rulkov neurons”. In: *Frontiers in Computational Neuroscience* 17 (2023).

[23] C. Chen et al. *A Topological Regularizer for Classifiers via Persistent Homology*. 2018. arXiv: 1806.10714 [cs.LG].

[24] E. Heiter et al. “Incorporating Topological Priors Into Low-Dimensional Visualizations Through Topological Regularization”. In: *IEEE Access* (2024).

[25] R. Vandaele et al. *Topologically Regularized Data Embeddings*. 2022. arXiv: 2110.09193 [cs.LG].

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

[27] V. Koch et al. “DinoBloom: A Foundation Model for Generalizable Cell Embeddings in Hematology”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2024, pp. 520–530.

[28] C. Pohlkamp et al. “Machine learning (ML) can successfully support microscopic differential counts of peripheral blood smears in a high throughput hematology laboratory”. In: *Blood* 136 (2020), pp. 45–46.

[29] M. Hehr et al. *A morphological dataset of white blood cells from patients with four different genetic AML entities and non-malignant controls (AML-Cytomorphology_MLL_Helmholtz) (Version 1)*. Data set. 2023.

[30] K. R. Moon et al. “Visualizing structure and transitions in high-dimensional biological data”. In: *Nature biotechnology* 37.12 (2019), pp. 1482–1492.

[31] L. McInnes, J. Healy, and J. Melville. “Umap: Uniform manifold approximation and projection for dimension reduction”. In: *arXiv preprint arXiv:1802.03426* (2018).

[32] H. B. Core. *General Information on the Helmholtz Munich HPC Cluster*. Accessed: 10.10.2024.

[33] A. M. Saxe et al. “On random weights and unsupervised feature learning.” In: *Icml*. Vol. 2. 3. 2011, p. 6.

[34] Z. Xu et al. “Robust and generalizable visual representation learning via random convolutions”. In: *arXiv preprint arXiv:2007.13003* (2020).

[35] R. Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[36] K. Simonyan. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[37] M. Oquab et al. “Dinov2: Learning robust visual features without supervision”. In: *arXiv preprint arXiv:2304.07193* (2023).

[38] L. Dongjin, S.-H. Lee, and J.-H. Jung. “The Effects of Topological Features on Convolutional Neural Networks – An explanatory analysis via Grad-CAM”. In: *Machine Learning: Science and Technology* 4 (Aug. 2023).

[39] C. Maria. “Filtered Complexes”. In: *GUDHI User and Reference Manual*. 3.1.1. GUDHI Editorial Board, 2020.

[40] D. J. Waibel et al. “Capturing shape information with multi-scale topological loss terms for 3d reconstruction”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2022, pp. 150–159.

[41] A. Krizhevsky. *Learning multiple layers of features from tiny images*. 2009.

[42] A. Acevedo et al. “A dataset of microscopic peripheral blood cell images for development of automatic recognition systems”. In: *Data in Brief* 30 (2020), p. 105474. ISSN: 2352-3409.