Deep Visual Representation Learning for Classification and Retrieval: Uncertainty, Geometry, and Applications

by

Tyler R. Scott

B.S., University of Colorado, Boulder, 2019

M.S., University of Colorado, Boulder, 2019

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2023

Committee Members:

Elizabeth Bradley, Chair

Michael Mozer

Danna Gurari

Bo Waggoner

Maziar Raissi

Scott, Tyler R. (Ph.D., Computer Science)

Deep Visual Representation Learning for Classification and Retrieval: Uncertainty, Geometry, and Applications

Thesis directed by Prof. Elizabeth Bradley

Deep visual representation learning is the process by which deep neural networks discover a low-dimensional latent feature space, or embedding space, of visual data such that distance serves as a proxy for semantic dissimilarity. We consider deep visual representation learning tailored for classification and retrieval applications, that is, the representation is trained to discriminate between inputs belonging to different classes. In particular, we explore two facets of these visual representations: their *stochasticity* and their *geometry*.

The vast majority of losses, or methods, used to discover visual representations operate on *deterministic* embeddings where an input projects to a single point in the embedding space. Methods that produce *stochastic* embeddings, in contrast, project an input to a random variable whose distribution reflects its uncertainty in the semantic space. Capturing uncertainty in the embedding space is useful for robust classification and retrieval, informing downstream applications, and interpreting representations. Our primary focus is designing novel loss functions for discovering stochastic visual representations that perform equivalently or better than deterministic alternatives, are efficient and tractable, and are more robust.

The secondary focus is on the *geometry* of the representation. The three geometries are Euclidean, spherical, and hyperbolic, and each induce constraints on the latent space. In conjunction with designing stochastic embedding methods, we empirically explore the three geometries. We propose two novel stochastic methods: (1) the Stochastic Prototype Embedding using Gaussians in Euclidean space and (2) the von Mises–Fisher loss using von Mises–Fisher distributions in spherical space (i.e., on the unit hypersphere). While each of the three geometries has benefits, we find that spherical methods produce the strongest discrimination between classes and thus are well-suited for the downstream retrieval and classification applications that act on the learned representations.

Our tertiary focus involves the application of discriminative visual representations, appealing to practitioners via two large-scale empirical studies. The first unifies few- and zero-shot egocentric action recognition—and more generally, few- and zero-shot classification—verifying that the same representation can be used jointly for both tasks without degrading generalization. The second explores clustering pretrained embeddings with results that emphasize (1) the benefit of spherical representations, (2) the value of shallow, unsupervised clustering methods, for example hierarchical agglomerative clustering, when carefully tuned and benchmarked, and (3) the fragility of recent supervised, deep clustering methods operating on embeddings with more uncertainty (i.e., less discrimination).

Dedication

To my dad, Kerry, for his limitless support.

Acknowledgements

I'd like to begin by thanking Mike Mozer for advising me throughout my collegiate journey, beginning in my third year of undergrad through graduate school. I believe many students strive to mimic their advisors, and Mike could not have been a better mentor to imitate. Mike taught me not only how to construct and execute a research agenda, but how to effectively articulate it.

Next, I'd like to thank my committee for their support, feedback, and guidance, in particular, Liz Bradley, for graciously taking over as the chair of my committee ad hoc while Mike and I worked together at Google.

Much of my research would not have been possible without support from Sensory, Meta, and Google, where I interned and worked part-time during the process. Thank you to David Sterling, Paul Mullowney, Gordon Haupt, and Todd Mozer at Sensory; Karl Ridgeway, Michael Shvartsman, Ruta Desai, Kevin Carlberg, James Hillis, and Brenden Lake at Meta; Andy Gallagher and Ting Liu at Google; and Michael Iuzzolino—my peer, collaborator, and friend throughout the Ph.D.

Finally, I'd like to thank my partner, Taylor, and my parents, Kerry and Lisa, for their support and encouragement along the way.

Contents

Tables

Figures

Chapter 1

Introduction

Until the recent advancements with deep neural networks, feature engineering was a critical step in the successful application of machine learning. The input *representation* to a learning algorithm was typically handcrafted features from raw data deemed relevant for a task. While these input representations could lead to strong task performance, manually constructing them was problematic, as they were challenging to discover, expensive, and might have required subject-matter experts. In contrast, deep neural networks operate with near-raw data as input (e.g., a 2D grid of normalized pixels), and progressively refine their internal latent representation through stacked nonlinear operations. Internal representations nearing the output of the network reflect the task-relevant features of the data, elicited by the *loss function*. Deep networks can thus be interpreted as systems for automatically discovering useful encodings of data, relieving humans of manually constructing features*.

We consider deep neural networks that discover visual *representations* or *embeddings*, that is, networks which accept visual data as input, such as videos or static images, and project the data into a low-dimensional embedding space where distance is a proxy for semantic dissimilarity. Specifically, our focus is on visual representations discovered for *classification* or *retrieval applications*. Classification applications map an embedding to a categorical distribution over a set of classes. For example, in optical character recognition (e.g., MNIST [70]) the set of classes may be the set of

*This is not to understate the value of human intervention in data preprocessing, data cleaning, hyperparameter tuning, encoding inductive biases into the network's architecture, and designing better loss functions.

possible characters. Retrieval applications, commonly used in recommendation systems, find and return stored inputs most similar, in the embedding space, to a query input. For example, the retrieved images most similar to a particular optical character should belong to the same character class or be a visually-similar character. Intuitively, representations well-suited for classification are also well-suited for retrieval, and vice-versa. To perform well on both, same-class embeddings must be separable—and in many cases are encouraged to be linearly separable—from different-class embeddings. Increased separation among different-class embeddings leads to increased effectiveness of a linear classifier (e.g., support vector machines operate on similar maximum-margin principles) and increased likelihood of retrieved nearest neighbors of a query embedding belonging to the same class. Experimentation in coming chapters verify the above intuition.

Discovering visual representations for classification and retrieval rely on *supervision*, which indicates to the loss function either the class an input belongs to or the set of inputs that should project near one another, and thus the representations are considered *discriminative*. Our research is centered around two facets of these visual representations: their *stochasticity* and their *geometry*, and argues that the representations can benefit from consideration of both.

The vast majority of losses, or methods, operate on *deterministic* embeddings, where an input projects to a single point in the embedding space. Deterministic embeddings fail to capture uncertainty related to either input corruption or class ambiguity. Representing uncertainty can be important for many reasons including robust classification and retrieval, informing downstream models or applications, and interpreting representations. Additionally, loss functions that encourage the network to marginalize over uncertainty in the embedding space can assist in discovering representations that better discriminate among classes. Our primary focus is on loss functions which operate on *stochastic* embeddings, where the embedding is not a point, but a random variable whose distribution reflects uncertainty in the semantic space. We show that stochastic embedding methods can perform equivalently or better than deterministic alternatives, are efficient and tractable, and are more robust.

Our secondary focus is on the geometry of the representation. The three possible geometries are Euclidean, spherical, and hyperbolic. The choice of the geometry has implications on the structure of the embedding space, the manifestation of the loss function, and the inductive biases one might consider given a particular domain, dataset, or setting. Along with our research on stochastic embedding methods, we empirically explore these three geometries. We propose two novel stochastic embedding methods: (1) the Stochastic Prototype Embedding using Gaussians in Euclidean space and (2) the von Mises–Fisher loss using von Mises–Fisher distributions in spherical space (i.e., on the unit hypersphere). For visual domains, the spherical geometry leads to the strongest discrimination between classes and thus is well-suited for many of the downstream applications that act on the learned representations.

The tertiary focus of our research appeals to practitioners of discriminative visual representations through two large-scale empirical studies. The first proposes a unification of few- and zero-shot classification based on data from egocentric video and explores several candidate spherical loss functions. Due to the unification, our results are the first to indicate that one can jointly perform few- and zero-shot classification using the same representation without degrading generalization. To support our research, as well as future research on open-set egocentric video classification, we developed new splits of the popular EPIC-KITCHENS dataset [23]. The second empirical study is based on clustering pretrained embeddings—an alternative approach to zero-shot classification when compared to the former egocentric video study. We investigate numerous clustering methods, varying from classical k-means to deep graph convolutional networks, on representations discovered for three large-scale datasets. Our results contend with recent research, arguing that the success of deep clustering methods depends on the discriminability of the visual representation, with additional findings corroborating the benefits of spherical embeddings.

This thesis is organized into the following chapters:

- Chapter 2 surveys deep visual representation learning by introducing the background on losses for discovering representations, the three embedding geometries, and the various types of classification and retrieval applications.

- Chapter 3 proposes a novel stochastic embedding method, namely *Stochastic Prototype Embeddings* (SPE). Extending a popular deterministic method, Prototypical Networks [112], SPE supposes the existence of a class prototype around which class embeddings are Gaussian distributed. The prototype posterior is a product distribution over supervised embeddings, and a query embedding is classified by marginalizing relative prototype proximity over embedding uncertainty. We describe an efficient sampler for approximate inference that allows us to train the model at roughly the same space and time cost as its deterministic sibling. Incorporating uncertainty improves performance on fixed- and open-set classification, specifically few-shot classification, and gracefully handles input noise and label ambiguity.

- Chapter 4 introduces a second stochastic-embedding loss: the *von Mises–Fisher loss*. In light of research supporting losses that operate on embeddings normalized to the unit-hypersphere, we propose a novel loss based on the von Mises–Fisher distribution, and we show that it is competitive with the state-of-the-art while producing significantly improved out-of-the-box calibration. Compared to SPE in Chapter 3, the von Mises–Fisher loss can scale to much higher dimensional spaces and does not suffer from the curse of dimensionality that commonly afflicts Gaussians.

- Chapter 5 shifts to empirical research on few- and zero-shot classification with egocentric video data[†]. We propose a unification of few- and zero-shot classification by reframing the latter as *cross-modal few-shot classification*. We introduce a new set of splits derived from the EPIC-KITCHENS dataset [23] to facilitate open-set classification and investigate several candidate spherical losses. Our results indicate that losses designed to jointly perform the two classification tasks can improve zero-shot performance by as much as 10% without sacrificing few-shot performance.

[†]The research was conducted during an internship at Meta Reality Labs and their interest was in egocentric video classification.

- Chapter 6 explores clustering of pretrained embeddings as a means of zero-shot classification. Recent research has leveraged clustering for pseudo-supervising data via zero-shot cluster assignments and has shown that novel, deep clustering approaches significantly and unequivocally outperform shallow methods such as k-means [78] and hierarchical agglomerative clustering [111]. We propose benchmarks on three large-scale datasets and find that (1) clustering approaches operating on spherical embeddings robustly outperform those operating on Euclidean embeddings and (2) the deep methods are fragile when the embedding space exhibits uncertainty (i.e., is less discriminative), reverting to or underperforming shallow methods. Our findings contend with recent, state-of-the-art research regarding the applicability of deep clustering methods and provide a suite of benchmarks to support further improvements.

Chapter 2

Background

A deep visual representation or embedding, hereafter $\boldsymbol{z}$, is a low-dimensional output of a multilayered, nonlinear neural network, $f_{\boldsymbol{\phi}}(\boldsymbol{x})$, parameterized by $\boldsymbol{\phi}$, and acting on visual stimulus, $\boldsymbol{x}$:

$$\boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{x}), \text{ where } \boldsymbol{z} \in \mathbb{R}^d, \ \boldsymbol{x} \in \mathbb{R}^D, \ d \ll D. \tag{2.1}$$

Along with the training input, we assume the existence of *categorical supervision*. Let y_i be the supervisory signal for training input, $\boldsymbol{x}_i$, where y_i belongs to a set, Y, containing an element for each of C classes (i.e., $y_i \in Y = \{1, 2, \ldots, C\}$). Even though most datasets for visual representation learning contain—or can synthesize—categorical supervision, tasks exist where supervisory signals come in weaker forms. One such task is modeling human-similarity judgments [5, 137] where inputs are presented as triplets of items containing a query and two references and the supervision, as judged by humans, is which of the two references is most similar to the query.

Table 2.1: Supervisory constraints from Ridgeway [96] where $s(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the similarity between $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ and $d(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the distance; sorted from strongest to weakest. Grayed constraints are not widely used in losses for deep visual representation learning.

Type	Example Constraint
Direct	$\boldsymbol{z}_i = \boldsymbol{\alpha}_i$
Equality	$d(\boldsymbol{z}_i, \boldsymbol{z}_j) = 0 \quad \text{or} \quad s(\boldsymbol{z}_i, \boldsymbol{z}_j) \to \infty$
Similarity/Distance	$s(\boldsymbol{z}_i, \boldsymbol{z}_j) = \delta_{ij} \quad \text{or} \quad d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \delta_{ij}$
Inequality	$d(\boldsymbol{z}_i, \boldsymbol{z}_j) < d(\boldsymbol{z}_i, \boldsymbol{z}_k) \quad \text{or} \quad s(\boldsymbol{z}_i, \boldsymbol{z}_j) > s(\boldsymbol{z}_i, \boldsymbol{z}_k)$
Analogy	$\boldsymbol{z}_i - \boldsymbol{z}_j = \boldsymbol{z}_k - \boldsymbol{z}_l$

To train the network, a loss function is used, denoted $\mathcal{L}(\boldsymbol{z}_i, y_i)$, which is differentiable and designed to encourage the network to produce outputs or make predictions acceptable for a task. The loss is minimized via gradient-based optimization of network parameters using algorithms such as stochastic gradient descent. We use a purposely-generic definition of the loss as it can vary in arguments, as well as learnable parameters, however it operates on at least one embedding and its paired supervisory signal.

Visual representation learning losses utilize categorical supervision by constraining embeddings. Ridgeway [96] presents a comprehensive set of supervisory constraints which are listed in Table 2.1. The common constraints in losses for visual representation learning are equality and inequality constraints, specifically because they can be applied in the presence of categorical supervision. Direct constraints are typically too inflexible and both similarity and analogical constraints require supervision that is too difficult to obtain.

Given tuples of embeddings and their paired supervisory signals from the training set, $\{(\boldsymbol{z}_i, y_i)\}_{i=1:m}$, the coming sections describe a broad subset of loss functions for discovering visual representations. At the highest level, the losses are categorized as either equality-constraint losses or inequality-constraint losses—although the latter contains far more. We further subcategorize losses based on additional factors such as:

- If the loss is *classification-based* where a categorical distribution over classes is estimated directly or *similarity-based* where the loss is evaluated based on the similarity or distance to other embeddings.

- The number of embeddings necessary to compute the loss (e.g., single-instance losses, pairwise losses, triplet losses, etc.).

After detailing losses, we introduce downstream classification and retrieval applications that act on the learned, discriminative representations, including fixed-set classification, open-set classification (i.e., few- and zero-shot), clustering, retrieval, verification, and inductive transfer learning.

2.1 Losses for Discovering Deep Visual Representations

Research on losses for visual representation learning is vast and expanding, and thus, we emphasize that the following survey is necessarily incomplete. Each year, tens, if not hundreds, of new losses are proposed and it would be infeasible to consider them all. To distill the losses to a manageable subset, we chose those that are either heavily cited, commonly used as baselines for comparison, or provide intuitions useful for generalizing to more complex losses in the literature. We acknowledge many areas of important work omitted such as meta-learning, ensembling, distillation, regularization, data augmentation, and specialized distance metrics. These areas all help discover stronger representations and should be carefully considered in conjunction with loss functions.

2.1.1 Equality-Constraint Losses

The primary equality-constraint loss is similarity-based and simply minimizes the distance between a pair of embeddings directly:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = d(\boldsymbol{z}_i, \boldsymbol{z}_j) \text{ where } y_i = y_j, \tag{2.2}$$

and one popular instantiation of the loss is squared-error (SE) in which $d(\cdot, \cdot)$ is squared Euclidean distance:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2 \text{ where } y_i = y_j. \tag{2.3}$$

The loss in Equation 2.2 should naturally cluster same-class embeddings, a desirable property of visual representations, when applied over a dataset, but networks can easily exploit the loss during training. Imagine a network that simply learned a constant function (i.e., $f_{\boldsymbol{\phi}}(\boldsymbol{x}_i) = \boldsymbol{\alpha}$). Such a network would minimize the loss while producing a useless representation. To force networks to learn useful representations, two approaches have been used.

The first is common in multi-modal prediction—for example, zero-shot classification to be described in Section 2.2—where one seeks to align two independent representations, each trained on data from different modalities. Let $f^v_{\boldsymbol{\phi}}$ be a network that operates on *visual* data, $\boldsymbol{x}^v_i$, such as images, indicated by the superscript v, and $f_{\boldsymbol{\nu}}^{\ell}$ be an independently-parameterized network operating on *natural language* data, $\boldsymbol{x}_i^{\ell}$, for example a word-embedding model, indicated by the superscript ℓ. Networks trained with natural language are able to leverage far larger amounts of training data compared to networks trained with visual stimuli, so it's common to use a pretrained word-embedding model that is *frozen* (i.e., the representation is fixed), and use Equation 2.2 to discover a visual representation aligned to the natural-language counterpart [113, 151]:

$$\mathcal{L}\left(\boldsymbol{x}_i^{v}, \boldsymbol{x}_i^{\ell}\right) = d\left(f_{\boldsymbol{\phi}}^{v}\left(\boldsymbol{x}_i^{v}\right), f_{\boldsymbol{\nu}}^{\ell}\left(\boldsymbol{x}_i^{\ell}\right)\right). \tag{2.4}$$

Aligning a learnable representation with a frozen one forces the network to learn a semantically-meaningful space, since a constant function will result in a large loss.

The second approach uses a more common solution: *contrastive learning.* Losses that employ contrastive learning do so by using an *attractive* force in conjunction with a *repulsive* force. The attractive force (e.g., Equation 2.2) clusters same-class embeddings while the repulsive force pushes different-class embeddings apart. The repulsive force is critical in preventing embedding collapse. The pairwise contrastive loss was introduced in Chopra et al. [21] and Hadsell et al. [42], and has the following standard formulation:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = \begin{cases} d(\boldsymbol{z}_i, \boldsymbol{z}_j), & \text{if } y_i = y_j \\ \max(0, m - d(\boldsymbol{z}_i, \boldsymbol{z}_j)), & \text{otherwise.} \end{cases} \tag{2.5}$$

Note that the loss in Equation 2.5 combines both an equality constraint and an inequality constraint. When the pair of embeddings belong to the same class, the loss simply minimizes their distance via an equality constraint. However, when the embeddings belong to different classes, the loss encourages their distance to be greater than a margin hyperparameter, $m > 0$.

The purpose of m is to prevent overfitting. Different-class or inter-class embeddings sufficiently far apart need not be separated further. When m is too large, the network is continually penalized, even when different-class embeddings are discriminable. However, m must be large enough such that inter-class pairs are not confused with same-class or intra-class pairs. When m is too small, the network can underfit by minimizing the loss without forcing discrimination among embeddings of different classes. One should not expect the network to project all instances from the same class to one point in the representational space. More likely, the intra-class instances will approximate some distribution with unknown *intra-class variance*. For example, imagine a face verification task where each true identity represents a class. The same face may be seen in various conditions, for example with a mask or sunglasses, after a haircut, in different lighting conditions, or at different poses. We cannot expect the network to be perfectly invariant to all such conditions, thus some of these features will be represented in the embedding. The margin should be set such that the network can still represent some intra-class variance while ensuring inter-class pairs are discriminable. One downside of the pairwise contrastive loss even when m is optimally-chosen, however, is that it is constant throughout the embedding space. A constant margin works well assuming the intra-class variance is the same across all identities, which is likely not true. We discuss losses in coming sections that overcome this limitation.

2.1.2 Inequality-Constraint Losses

Inequality-constraint losses, either implicitly or explicitly, encourage same-class embeddings to be closer to one another than different-class embeddings. Rather than forcing same-class inputs to project to the same embedding (e.g., the SE loss in Equation 2.3), many of the inequality-constraint losses allow the representation to maintain intra-class variance, so long as it is distinguishable from embeddings belonging to other classes. The means by which discrimination among categories is achieved varies greatly across losses. We dichotomize inequality-constraint losses into two groups: classification-based and similarity-based. Classification-based losses attempt to predict the class an instance belongs to, while similarity-based losses are evaluated directly on similarities or distances between embeddings.

2.1.2.1 Classification-Based Losses

We focus on classification-based losses that utilize softmax cross-entropy and refer the reader to Kornblith et al. [61] for other variants such as sigmoid cross-entropy. Softmax cross-entropy losses map an embedding through the softmax function to a categorical distribution over classes. The categorical distribution is then evaluated with cross-entropy against a one-hot target vector*. The generic form of softmax cross-entropy [11, 12] is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{z}_i, y_i, C) &= -\sum_{j=1}^{C} \mathbb{1}[j = y_i] \log p(j|\boldsymbol{z}_i) = -\log p(y_i|\boldsymbol{z}_i) \\ &= -\log(\text{softmax}(\boldsymbol{z}_i, y_i, C)) = -\log \frac{\exp(h(\boldsymbol{z}_i, y_i))}{\sum_{j=1}^{C} \exp(h(\boldsymbol{z}_i, j))},\end{aligned} \tag{2.6}$$

where $C = |Y_{\text{train}}|$ is the number of classes in the training dataset and $h(\boldsymbol{z}_i, j)$ is a function that computes a logit for class j. The loss encourages the network to produce a large logit value for the correct class, y_i, relative to the logit values for competing classes, $j \in \{1, 2, \ldots, C \,|\, j \neq y_i\}$. As will become clear in Section 2.1.2.1, the logit function, $h(\boldsymbol{z}_i, j)$, can be interpreted as a similarity function, typically between $\boldsymbol{z}_i$ and a prototypical representation for class j, denoted by $\boldsymbol{w}_j$. Thus, the loss is minimized when $h(\boldsymbol{z}_i, y_i) >> h(\boldsymbol{z}_i, j) \;\forall\; j \neq y_i$, which is why softmax cross-entropy losses are implicitly inequality based.

Many variants of softmax exist, each employing a unique logit function, h, and can be categorized according to the *embedding geometry*. The embedding geometry has implications on the similarity structure of the embedding space and determines the specific mapping from embedding to class posterior, $p(y|\boldsymbol{z}_i)$. The three possible embedding geometries—Euclidean, spherical, and hyperbolic—are described below along with the softmax variants operating within them.

Euclidean Softmax Variants. Euclidean embeddings lie in a d-dimensional real-valued space (i.e., $\boldsymbol{z}_i \in \mathbb{R}^d$). The *standard* softmax formulation uses the following dot-product logit function:

$$h_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C,b_1,\ldots,b_C}(\boldsymbol{z}_i, j) = \boldsymbol{w}_j^{\mathrm{T}} \boldsymbol{z}_i + b_j = \|\boldsymbol{w}_j\| \, \|\boldsymbol{z}_i\| \cos\theta_j + b_j, \tag{2.7}$$

*A one-hot vector refers to a vector of all zeros except for one element with a value of one.

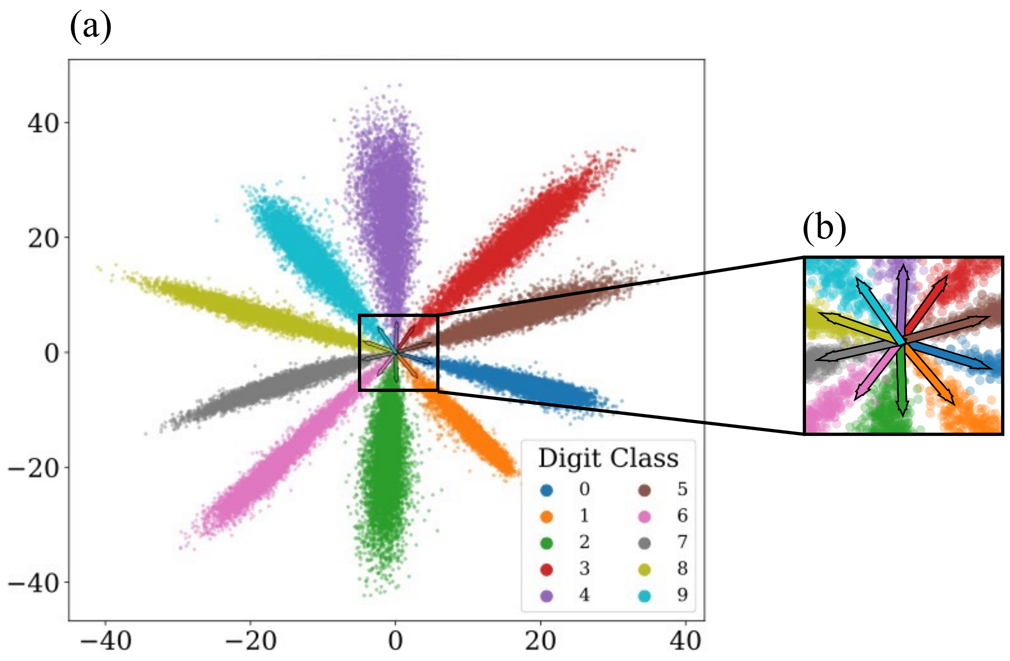


Figure 2.1: (a) A 2D embedding space learned with standard softmax cross-entropy on MNIST [70]. Each point is a test-set embedding colored by the ground-truth digit class. (b) A zoom of the learned class weight vectors, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{10}$ for the 10 digit classes. Each weight vector is drawn with an arrow outlined in black, colored by the class it represents.

where $\boldsymbol{w}_j \in \mathbb{R}^d$ is a learnable weight vector for class j, θ_j is the angle between $\boldsymbol{w}_j$ and $\boldsymbol{z}_i$, and $b_j \in \mathbb{R}$ is a learnable scalar bias for class j.

Figure 2.1(a) shows a 2D embedding learned on MNIST [70] using standard softmax cross-entropy. Each point represents the embedding of a test input and the drawn arrows that are zoomed in Figure 2.1(b) show the learned class weight vectors, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{10}$. When used in the softmax function, the magnitude of the class weight, $\|\boldsymbol{w}_j\|$, along with the biases intuitively behave as class priors. For datasets like MNIST where the classes are roughly balanced in terms of the number of assigned instances, the class weight vectors maintain similar magnitudes, as seen in Figure 2.1(b). Additionally, note that the class weight vectors live in the same representational space as the embeddings and further, they can be interpreted as “prototypical” representations for their respective class—at least prototypical in the sense that they are approximately radially aligned with their associated embeddings. Interpreting the weight vectors as class prototypes and h as a similarity function motivates the categorization of softmax cross-entropy as an inequality-constraint

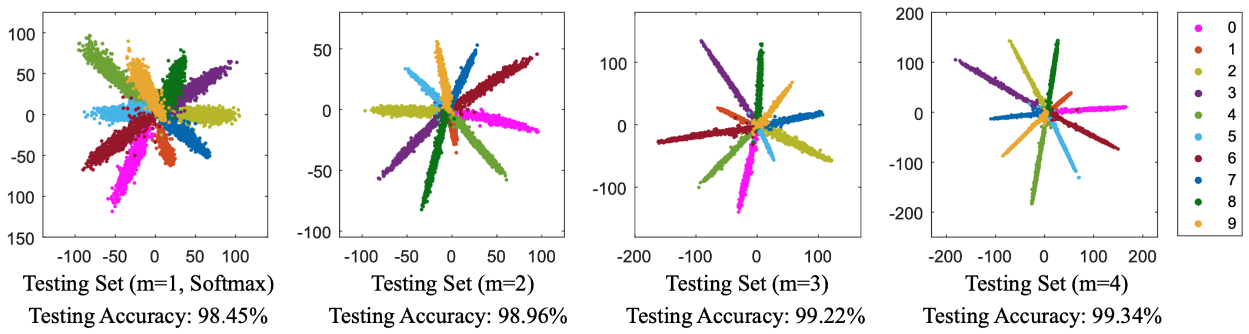


Figure 2.2: 2D embeddings of the test set from MNIST learned with L-Softmax [76]. Plots from left to right show embeddings trained with increasing values of the margin, m. Taken directly from Liu et al. [76].

loss, even though it predicts class association directly.

To minimize softmax cross-entropy when Equation 2.7 is used as the logit function, the network will first attempt to radially align $\boldsymbol{z}_i$ and $\boldsymbol{w}_{y_i}$, then it will focus on increasing $\|\boldsymbol{z}_i\|$. In practice and verified in Figure 2.1(a), the network tends to focus more on enlarging the embedding norm rather than continued alignment of embeddings with their respective class weight vectors, as the norm is unbounded unlike cosine similarity [18]. However, increasing the embedding norm as a strategy for minimizing the loss only applies to inputs already classified correctly. As pointed out in Chen et al. [18], the most difficult examples are those not well-aligned with their ground-truth class weight vector. This observation has motivated recent softmax variants, particularly those using a spherical embedding geometry—which is discussed in Section 2.1.2.1 below.

Both the large-margin softmax (L-Softmax) [76] and angular softmax (A-Softmax) [75] are Euclidean variants that attempt to force the network to focus on angular discrimination between classes. L-Softmax uses the following conditional logit function:

$$h_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C}(\boldsymbol{z}_i, j) = \begin{cases} \|\boldsymbol{w}_j\| \, \|\boldsymbol{z}_i\| \left((-1)^k \cos(m\theta_j) - 2k\right), & \text{if } j = y_i \\ \|\boldsymbol{w}_j\| \, \|\boldsymbol{z}_i\| \cos(\theta_j), & \text{otherwise,} \end{cases} \tag{2.8}$$

where $m \in \mathbb{N}^+$ is a fixed margin hyperparameter, $\theta_j \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right]$, and $k \in \mathbb{N}$ with $k \leq m-1$. Increasing m applies a greater penalty to the logit corresponding only to the ground-truth class, y_i, forcing the network to further minimize θ_{y_i}. As seen from left to right in Figure 2.2, increasing

m leads to improved angular discrimination among embeddings from different classes. A-Softmax [75] is identical except that it ℓ_2-normalizes all weight vectors such that $\|\boldsymbol{w}_j\| = 1 \; \forall \; j$.

Other Euclidean softmax variants compare embeddings to each other rather than to learnable weight vectors. One such loss is from Prototypical Networks (ProtoNets) [112]:

$$h(\boldsymbol{z}_i, \boldsymbol{\rho}_j) = -\left\|\boldsymbol{z}_i - \boldsymbol{\rho}_j\right\|^2 \text{ where } \boldsymbol{\rho}_j = \frac{1}{|S_j|} \sum_{\boldsymbol{z} \in S_j} \boldsymbol{z}. \tag{2.9}$$

For the ProtoNets loss, it is assumed that a held-out *support set* of embeddings exists that is used solely to support classification. When the network is presented with a *query embedding*, $\boldsymbol{z}_i$, it is compared to each of the *class prototypes*, $\boldsymbol{\rho}_j$, computed as the mean of embeddings in the support set, S_j, for class j. The loss is motivated by the setup used in *few-shot classification* which we describe in Section 2.2 with details on support and query sets. Functionally, the logit is computed with a squared Euclidean distance and compares embeddings to class prototypes estimated from other embeddings rather than class prototypes learned via gradient descent.

One downside of the ProtoNets loss is that it operates at a relatively coarse granularity. When used with softmax, Equation 2.9 corresponds to each prototype being represented as an isotropic Gaussian. To model more complex similarity structures in the embedding space, some losses operate at an embedding level rather than a prototype level. For example, the loss from Koch et al. [60] uses binary cross-entropy (as opposed to multi-class cross-entropy) to determine if a pair of embeddings belong to the same class or not with the following logit function:

$$h_{\boldsymbol{\beta}}(\boldsymbol{z}_i, \boldsymbol{z}_j) = \sum_k \boldsymbol{\beta}_k |\boldsymbol{z}_{ik} - \boldsymbol{z}_{jk}|, \tag{2.10}$$

where $\boldsymbol{z}_{ik}$ is the kth element of $\boldsymbol{z}_i$, $\boldsymbol{\beta} \in \mathbb{R}^d$ is a learnable vector that weighs the various dimensions of the embedding, and $\boldsymbol{\beta}_k$ is the kth element of $\boldsymbol{\beta}$. Rather than using categorical supervision, Equation 2.10 uses a weaker form of supervision where $y_{ij} = 1$ if $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ belong to the same class and $y_{ij} = 0$ otherwise. Two alternative approaches used by Ba et al. [6] and Liu et al. [73] apply the dot-product logit function (Equation 2.7) directly between $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ and to the concatenation of $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ and learned weight vectors, respectively, followed by binary cross-entropy.

The n-pairs loss [56, 114, 124] is a generalization of the losses from [6, 60, 73] to higher-order sets of embeddings. In particular, assume we have the following mini-batch of embeddings:

$$\{\boldsymbol{z}_i, \boldsymbol{z}^+, \boldsymbol{z}_1^-, \ldots, \boldsymbol{z}_{n-1}^-\}, \tag{2.11}$$

where $\boldsymbol{z}_i$ is a query embedding, $\boldsymbol{z}^+$ is an embedding from the same class as the query, and $\boldsymbol{z}_1^-, \ldots, \boldsymbol{z}_{n-1}^-$ are $n-1$ embeddings from different classes. The goal of the n-pairs loss is to ensure the similarity between $\boldsymbol{z}_i$ and $\boldsymbol{z}^+$, is larger than the similarity between the query and any of the different-class or negative embeddings. The loss uses the dot-product logit function specified in Equation 2.7 between $\boldsymbol{z}_i$ and all other embeddings in the mini-batch, except with $b_j = 0 \; \forall \; j$. The benefit of the n-pairs loss is that it is able to learn complex similarity structures by being presented many unique combinations of embeddings, however, finding batches of embedding with informative pairs (i.e., pairs not already well separated) can be difficult and costly compared to simply learning a prototype for each class.

Spherical Softmax Variants. Spherical embeddings lie on the surface of a d-dimensional unit-hypersphere (i.e., $\boldsymbol{z}_i \in \mathbb{S}^{d-1}$). The simplest spherical loss [61, 129, 149], *cosine softmax*, uses the dot-product logit function where the embedding and all class weight vectors are ℓ_2-normalized (i.e., $\|\boldsymbol{z}_i\| = \|\boldsymbol{w}_j\| = 1 \; \forall \; i, j$) and $b_j = 0 \; \forall \; j$:

$$h_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C}(\boldsymbol{z}_i, j) = \frac{1}{\tau} \cos\theta_j, \tag{2.12}$$

where $\tau > 0$ is a temperature parameter that controls the peakedness of softmax. As $\tau \to 0$, the output distribution of softmax is aggressively peaked toward $\arg\max_j h(\boldsymbol{z}_i, j)$ and as $\tau \to \infty$, the output distribution approaches uniformity. Since $\cos\theta_j \in [-1, 1]$ is bounded, τ is necessary to scale the dynamic range of softmax. Without τ, the loss reaches a minima far earlier than desired resulting in underfitting and poor model calibration [39, 65]. Additionally, we acknowledge several minor variants of Equation 2.12. The loss from Ranjan et al. [95] does not constrain class weight vectors to lie on the unit-hypersphere and includes a learnable bias vector. Teh et al. [120] uses negative squared Euclidean distance between ℓ_2-normalized embeddings and class weight vectors

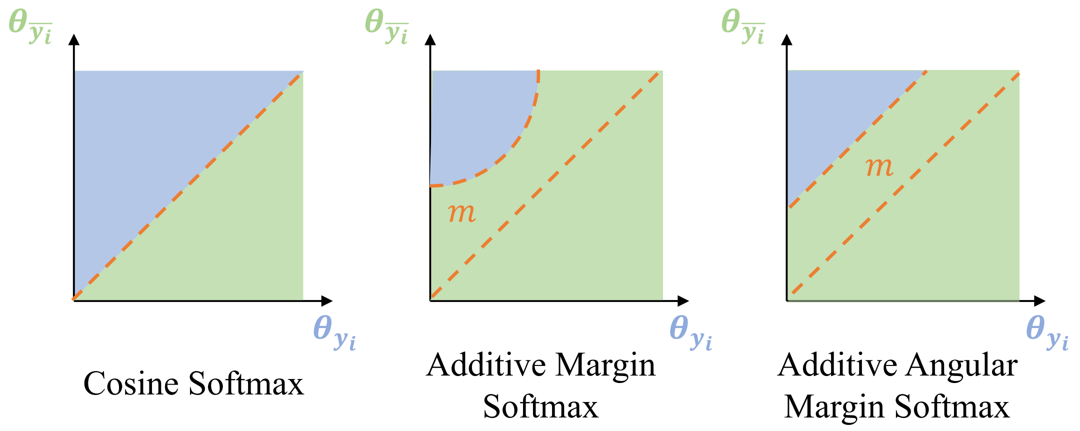


Figure 2.3: Visualization of the decision boundaries (dashed orange lines) induced by (left) cosine softmax [129, 149] (Equation 2.12), (middle) additive margin softmax [128, 130] (Equation 2.14), and (right) additive angular margin softmax [28] (Equation 2.15) for a binary classification task. θ_{y_i} represents the angle between the embedding and ground-truth class weight vector and $\theta_{\bar{y}_i}$ represents the angle between the embedding and the incorrect class's weight vector. For additive margin softmax and additive angular margin softmax, the second dashed orange line shows the updated decision boundary after imposing the margin, m.

instead of cosine similarity, which is functionally equivalent to doubling the temperature value, assuming the temperature is fixed. Movshovitz-Attias et al. [83] remove the temperature and minimize the negative log-odds instead of cross-entropy resulting in the following loss:

$$\mathcal{L}_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C}(\boldsymbol{z}_i, y_i, C) = -\log \frac{\exp\left(-d\left(\frac{\boldsymbol{z}_i}{\|\boldsymbol{z}_i\|}, \frac{\boldsymbol{w}_{y_i}}{\|\boldsymbol{w}_{y_i}\|}\right)\right)}{\sum_{j=1:j\neq y_i}^{C} \exp\left(-d\left(\frac{\boldsymbol{z}_i}{\|\boldsymbol{z}_i\|}, \frac{\boldsymbol{w}_j}{\|\boldsymbol{w}_j\|}\right)\right)}, \tag{2.13}$$

where d is squared Euclidean distance.

As discussed in Section 2.1.2.1, many softmax variants encourage the network to attend to angular separation because that improves class discrimination amongst the most-challenging inputs to classify. Spherical losses (e.g., cosine softmax) naturally accomplish this as the network cannot exploit the magnitudes of the embedding or class weight vectors. However, more can be done and one approach used by several losses is to incorporate margin hyperparameters similar to L-Softmax [76] and A-Softmax [75] above. Both additive margin softmax [128] and large margin cosine softmax

(CosFace) [130] propose the following logit function:

$$h_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C}(\boldsymbol{z}_i, j) = \begin{cases} \frac{1}{\tau}(\cos(\theta_j) - m), & \text{if } j = y_i \\ \frac{1}{\tau}\cos(\theta_j), & \text{otherwise,} \end{cases} \tag{2.14}$$

where $m \geq 0$ is a fixed margin hyperparameter. Again, as m increases, the network endures a stronger force to align $\boldsymbol{z}_i$ and $\boldsymbol{w}_{y_i}$. A recent alternative—currently considered as close, if not state-of-the-art—is additive angular margin softmax (ArcFace) [28]:

$$h_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C}(\boldsymbol{z}_i, j) = \begin{cases} \frac{1}{\tau}\cos(\theta_j + m), & \text{if } j = y_i \\ \frac{1}{\tau}\cos(\theta_j), & \text{otherwise.} \end{cases} \tag{2.15}$$

Figure 2.3 shows the decision boundaries induced by Equations 2.12, 2.14, and 2.15, respectively. Additive angular margin softmax argues superiority because it imposes a constant angular margin regardless of θ_{y_i}. In contrast, additive margin softmax imposes a larger margin when θ_{y_i} approaches its minimum and maximum values, which may lead to unstable training and overfitting.

As with the Euclidean variants, several spherical softmax cross-entropy losses perform embedding-to-embedding comparisons in the logit function which is more granular than comparing embeddings to learned or estimated class prototypes. Chen et al. [19] and Vinyals et al. [126] use the exact analog of the n-pairs loss [114, 124] above, but with cosine similarity instead of the unnormalized dot-product. Unlike the prior losses which use ground-truth supervision, Chen et al. [19] synthesizes supervision in a *self-supervised* manner (i.e., the data supervises itself). Self supervision is typically generated by applying a transformation or augmentation to an input such that the augmented input shares the same class. For example, in an image-based task, the set of transformations may be random cropping, color jittering, or rotation. The intuition is that an image of a golden retriever is still an image of a golden retriever after these minor image transformations, however, the augmented input contains enough intra-class variance for the pair of embeddings to be useful for training. After generating the positive pair via augmentation, one can train with an n-pairs-like loss by randomly selecting other embeddings in the dataset to form negative pairs.

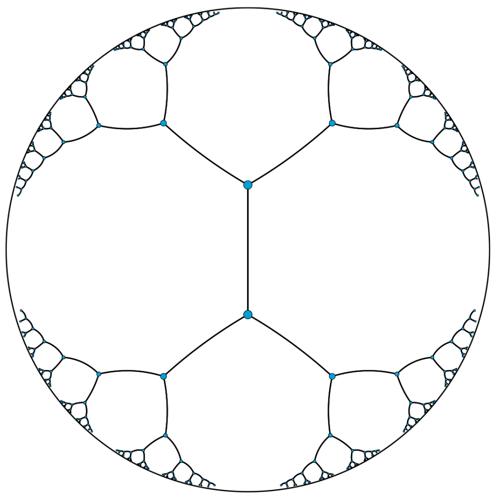


Figure 2.4: Visualization of $\mathbb{D}_1^2$. Each edge between two nodes has identical hyperbolic distance. The Euclidean distance between connected nodes decreases exponentially relative to the hyperbolic distance the closer the nodes are to the surface of the circle. The result is a tree-like representation where the origin represents the root and nodes near the surface of the circle represent leaves. Taken directly from Nickel and Kiela [87].

While the n-pairs loss allows the network to capture complex similarity structures by comparing embeddings to themselves rather than class prototypes, it suffers from scalability issues as the number of training instances increases. The n-pairs loss compares embeddings within a batch by forming positive and negative pairs and it requires the negative pairs to be informative. What tends to happen in practice is that most negative pairs of embeddings are sufficiently far apart and do not provide useful training signals. Finding informative negative pairs is a topic of research and many heuristics have been developed to help (e.g., semi-hard negative mining [103]).

The SoftTriple loss [93] takes an alternative approach that can be interpreted as an intermediary between the embedding-to-embedding losses and the embedding-to-prototype losses: allow each class to be represented by multiple prototypes. The SoftTriple loss uses the following conditional logit function:

$$h_{\boldsymbol{w}_1^{1:P},\dots,\boldsymbol{w}_C^{1:P}}(\boldsymbol{z}_i, j) = \begin{cases} \frac{1}{\tau}(S_{i,j} - m), & \text{if } j = y_i \\ \frac{1}{\tau} S_{i,j}, & \text{otherwise,} \end{cases}$$
$$\text{where } S_{i,j} = \sum_p \frac{\exp\left(\frac{1}{\gamma}\cos\theta_j^p\right)}{\sum_{p'}\exp\left(\frac{1}{\gamma}\cos\theta_j^{p'}\right)}\cos\theta_j^p. \tag{2.16}$$

The logit function uses an additive margin, m, to penalize the logit of the ground-truth class similar to Wang et al. [128] and Wang et al. [130], however, the similarity between the embedding and class j, $S_{i,j}$, is a smooth approximation to the max function across all P prototypes representing class j. The similarity, $S_{i,j}$, uses its own temperature parameter, denoted by γ, and θ_j^p denotes the angle between $\boldsymbol{z}_i$ and $\boldsymbol{w}_j^p$, the pth prototype for class j.

Hyperbolic Softmax Variants. Hyperbolic softmax variants assume the Poincaré ball model of hyperbolic geometry: $\mathbb{D}_c^d = \{\boldsymbol{z}_i \in \mathbb{R}^d \ : \ c\left\|\boldsymbol{z}_i\right\|^2 < 1, c \geq 0\}$, where c is a hyperparameter controlling the curvature of the ball. Ganea et al. [36] note that the geometry reverts to Euclidean when $c = 0$. Hyperbolic geometries exhibit tree-like properties, visualized in Figure 2.4, which is a desirable inductive bias when discovering representations of structured data. While the original motivation for hyperbolic softmax variants is to learn representations for structured data such as natural language, recent research has found benefits of hierarchical structure in visual tasks, as well [57].

Ganea et al. [36] derive the hyperbolic equivalent to standard Euclidean softmax, which results in the following logit function:

$$h_{\boldsymbol{a}_1,\dots,\boldsymbol{a}_C,\boldsymbol{p}_1,\dots,\boldsymbol{p}_C}(\boldsymbol{z}_i, j) = \frac{\lambda_{\boldsymbol{p}_j}^c \left\|\boldsymbol{a}_j\right\|}{\sqrt{c}} \sinh^{-1}\left(\frac{2\sqrt{c}\langle -\boldsymbol{p}_j \oplus_c \boldsymbol{z}_i, \boldsymbol{a}_j\rangle}{\left(1 - c\left\|-\boldsymbol{p}_j \oplus_c \boldsymbol{z}_i\right\|^2\right)\left\|\boldsymbol{a}_j\right\|}\right), \tag{2.17}$$

where $\boldsymbol{p}_j \in \mathbb{D}_c^d$ and $\boldsymbol{a}_j \in \mathrm{T}_{\boldsymbol{p}_j}\mathbb{D}_c^d \setminus \{\boldsymbol{0}\}^\dagger$ are learnable parameters for class j, $\lambda_{\boldsymbol{p}_j}^c$ is the conformal factor of $\boldsymbol{p}_j$, $\langle\cdot\rangle$ is the dot product, and $\oplus_c$ is the Möbius addition operator. Note that when $c = 0$, the logit function reduces to $\langle -\boldsymbol{p}_j + \boldsymbol{z}_i, \boldsymbol{a}_j\rangle = \langle \boldsymbol{a}_j, \boldsymbol{z}_i\rangle - \langle \boldsymbol{p}_j, \boldsymbol{a}_j\rangle$ which is exactly the standard Euclidean softmax logit function with $\boldsymbol{w}_j = \boldsymbol{a}_j$ and $b_j = -\langle \boldsymbol{a}_j, \boldsymbol{p}_j\rangle$.

$^\dagger\mathrm{T}_{\boldsymbol{z}}\mathbb{D}_c^d$ denotes the tangent space of $\mathbb{D}_c^d$ at $\boldsymbol{z}$.

Nickel and Kiela [87] and Khrulkov et al. [57] employ the hyperbolic equivalents of n-pairs loss [114, 124] and ProtoNets loss [112], respectively. Both rely on the geodesic distance induced by the Poincaré ball model:

$$d_c(\boldsymbol{z}_1, \boldsymbol{z}_2) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c}\,\|-\boldsymbol{z}_1 \oplus_c \boldsymbol{z}_2\|), \tag{2.18}$$

except Nickel and Kiela [87] assume $c = 1$.

2.1.2.2 Similarity-Based Losses

Similarity-based losses also rely on inequality constraints, but do so by directly optimizing distance relationships between embeddings rather than predicting some form of class association, as seen previously with the classification-based losses. Standard classification losses map an embedding to a distribution over C classes. If we assume N training embeddings and C class prototypes, the result is $O(NC)$ comparisons, where $C << N$. In contrast, similarity-based losses compare pairs, triplets, or higher-order sets of embeddings, resulting in exponential growth in the number of possible comparisons. Sampling informative sets of embeddings to compare is a highly-active topic of research and critical to the success of these losses. In this survey, we only focus on the loss functions, but refer the reader to other research regarding the sampling process (e.g., [103, 140]).

As expected, the chronology of similarity-based losses is correlated with their performance and ability to discover strong representations. Additionally, we noticed a correlation between the chronology of the losses and how many embeddings are used in them. The recently proposed similarity-based losses tend to incorporate higher-order sets of embeddings beyond just pairs or triplets. This choice is intuitive, as comparing larger sets of embeddings provides more information to the network of the complex similarity structures, both locally and globally, in the representation. Thus, we group losses according to the number of embeddings necessary to compute the loss, which roughly follows chronological order of when the losses were proposed. We start with pairwise losses, then triplet losses, and finish with losses that operate on entire mini-batches of embeddings.

We note that the majority of similarity-based losses described below use a generic distance or similarity function. While one could, in principle, use a distance function induced by any of the three embedding geometries, many have found cosine similarity or cosine distance—induced by a spherical geometry—results in the strongest discrimination. We discuss this choice and its implications further in Chapter 4.

Pairwise Losses. Pairwise losses are motivated by a simple goal: all pairs of same-class embeddings should be close together and all pairs of different-class embeddings should be far. The classic pairwise contrastive loss [42], presented in Section 2.1.1, accomplishes the goal using a conditional loss function, combining both an equality and inequality constraint:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = \begin{cases} d(\boldsymbol{z}_i, \boldsymbol{z}_j), & \text{if } y_i = y_j \\ \max(0, m - d(\boldsymbol{z}_i, \boldsymbol{z}_j)), & \text{otherwise.} \end{cases} \tag{2.19}$$

Equation 2.19 encourages same-class inputs to project to the same embedding and different-class inputs to project at least a distance m apart, where $m > 0$ is a margin hyperparameter that is fixed.

The pairwise contrastive loss can also be interpreted as a *hard-margin* or *hinge loss*. Once a different-class pair has distance m, the loss becomes zero. Yi et al. [145] proposed a soft-margin version of the contrastive loss called Binomial Deviance, utilizing a softplus-like nonlinearity:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = \begin{cases} \log\left[1 + e^{\alpha(\lambda - s(\boldsymbol{z}_i, \boldsymbol{z}_j))}\right], & \text{if } y_i = y_j \\ \log\left[1 + e^{\beta(s(\boldsymbol{z}_i, \boldsymbol{z}_j) - \lambda)}\right], & \text{otherwise,} \end{cases} \tag{2.20}$$

where $s(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the similarity between $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ and $\alpha > 0$, $\beta > 0$, and $\lambda > 0$ are fixed hyperparameters. For $y_i \neq y_j$ and $\beta = 1$, Equations 2.19 and 2.20 are nearly identical except that the latter smoothly approaches its minimum. One important benefit of Binomial Deviance over the pairwise contrastive loss is that it applies a greater penalty to pairs farther from satisfying their inequality constraint. The partial derivative of Equation 2.20 with respect to the similarity function depends on the similarity itself, as well as α, β, and λ. When same-class pairs have small similarity, a larger penalty is applied to them, likewise with different-class pairs and a large similarity. In contrast, the partial derivative of the pairwise contrastive loss with respect to the distance function is constant, regardless of the distance itself. Importance-weighted gradients are attributed to improved generalization [133]. Hu et al. [50] proposed a very similar loss to Binomial Deviance except they use a true softplus nonlinearity and a constant offset applied to the margin for same-class and different-class pairs:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j) = \begin{cases} \frac{1}{\beta} \log \left[1 + e^{\beta(1-\lambda+d(\boldsymbol{z}_i,\boldsymbol{z}_j))}\right], & \text{if } y_i = y_j \\ \frac{1}{\beta} \log \left[1 + e^{\beta(1+\lambda-d(\boldsymbol{z}_i,\boldsymbol{z}_j))}\right], & \text{otherwise,} \end{cases} \tag{2.21}$$

where $\beta > 0$ and $\lambda > 1$ are fixed hyperparameters.

Regardless of if the loss uses a hard or soft margin, the margin is fixed and can lead to two potential issues. First, classes may exhibit different amounts of intra-class variance, and a fixed, absolute margin may over-separate some classes while not separating others enough. Second, the embedding space evolves throughout training and one may want to anneal m as the representation better discriminates.

The next group of losses attempt to fix the issue of an absolute margin by leveraging triplet constraints with a relative margin that provides the network flexibility in representing classes with differing amounts of intra-class variance.

Triplet Losses. The well-known triplet loss [35, 103, 136] is defined as:

$$\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j, \boldsymbol{z}_k, y_k) = \max(0, d(\boldsymbol{z}_i, \boldsymbol{z}_j) + m - d(\boldsymbol{z}_i, \boldsymbol{z}_k)) \text{ where } y_i = y_j \neq y_k, \tag{2.22}$$

and $m > 0$ is a fixed, relative margin hyperparameter. The triplet loss has a similar formulation to the pairwise contrastive loss, except it merges the attractive and repulsive forces into a single term with an anchor embedding, $\boldsymbol{z}_i$. Additionally, it employs a relative margin. Note that the loss is zero when $d(\boldsymbol{z}_i, \boldsymbol{z}_j) + m \leq d(\boldsymbol{z}_i, \boldsymbol{z}_k)$. Regardless of the intra-class variance among embeddings in class y_i, impostor embeddings should be pushed an additional margin away. The triplet loss thus allows the network to flexibly represent intra-class variance, particularly when it differs among classes, without sacrificing discriminability. The downside of the triplet loss, like many of the losses directly comparing embeddings, is the training dataset contains $O(N^3)$ possible triplets and after a short amount of training, the majority of the triplets have a loss of zero (i.e., informative triplets are sparse). Selecting triplets with non-zero loss is critical to discovering strong representations [103].

Yuan et al. [148] rectify the poor sample complexity of triplets by relaxing the loss. Instead of comparing a triplet of embeddings, they compare an embeddings to empirically-estimated centroids for each class, $\boldsymbol{c}_j \,\forall\, j \in \{1, \ldots, C\}$. They define the relaxed triplet loss as:

$$\mathcal{L}(\boldsymbol{z}_i, \boldsymbol{c}_{y_i}, \boldsymbol{c}_j) = \max(0, d(\boldsymbol{z}_i, \boldsymbol{c}_{y_i}) + m - d(\boldsymbol{z}_i, \boldsymbol{c}_j)) + R(\boldsymbol{c}_{y_i}) + R(\boldsymbol{c}_j) \text{ where } y_i \neq y_j, \tag{2.23}$$

and $R(\boldsymbol{c}_j)$ is the radius of the cluster with centroid $\boldsymbol{c}_j$. Equation 2.23 is shown to be an upper bound on the standard triplet loss. By using cluster centroids to compare against an embedding, the sample complexity reduces to $O(CK^2)$ where K is the average number of embeddings belonging to a class.

Another notable variant of the triplet loss is the angular loss [129]. Wang et al. [129] observe that a triplet of embeddings consisting of a pair of same-class embeddings, $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$, as well as an impostor embedding, $\boldsymbol{z}_k$, form a triangle. The triplet loss only considers two edges of that triangle, $\|\boldsymbol{z}_i - \boldsymbol{z}_j\|$ and $\|\boldsymbol{z}_i - \boldsymbol{z}_k\|$, whereas the third edge, $\|\boldsymbol{z}_j - \boldsymbol{z}_k\|$, provides additional, unused information. They derive a symmetrical triplet constraint with $\boldsymbol{z}_j$ as the anchor (i.e., $\|\boldsymbol{z}_i - \boldsymbol{z}_j\| + m \leq \|\boldsymbol{z}_j - \boldsymbol{z}_k\|$ and note that when both triplet constraints (i.e., one with $\boldsymbol{z}_i$ as the anchor and one with $\boldsymbol{z}_j$ as the anchor) are satisfied, the angle at the vertex of the impostor embedding, $\angle\boldsymbol{z}_k$, must be the smallest (i.e., $\angle\boldsymbol{z}_k \leq \min(\angle\boldsymbol{z}_i, \angle\boldsymbol{z}_j)$). They design a loss that attempts to minimize $\angle\boldsymbol{z}_k$ directly. Specifically, they try to satisfy the inequality constraint $\tan\angle\boldsymbol{z}_k \leq \tan m$ where $m > 0$ is an angular margin. The angular loss takes the following final form:

$$\begin{aligned}\mathcal{L}(\boldsymbol{z}_i, y_i, \boldsymbol{z}_j, y_j, \boldsymbol{z}_k, y_k) = \max\left(0, \|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2 - 4\tan^2 m \left\|\boldsymbol{z}_k - \frac{\boldsymbol{z}_i + \boldsymbol{z}_j}{2}\right\|^2\right) \\ \text{where } y_i = y_j \neq y_k.\end{aligned} \tag{2.24}$$

The angular loss has several benefits over the triplet loss: (1) it's scale invariant, (2) it incorporates all three edges of the triplet triangle, and (3) m has an interpretable value as an angle, unlike previous margins which lack a meaningful reference point.

Higher-Order Losses. Higher-order loss functions leverage all embeddings in a mini-batch rather than treating pairs or triplets as independent. The first two losses, lifted structure loss [48, 116] and multi-similarity loss [133], consider all possible triplets within a mini-batch and combine them nonlinearly. The lifted structure loss is defined as:

$$\begin{aligned}&\mathcal{L}((\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_n, y_n)) = \\ &\qquad \frac{1}{n}\sum_{i=1}^{n} \max\left(0, \left[\log \sum_{j\,:\,y_j=y_i} e^{\lambda - s(\boldsymbol{z}_i,\boldsymbol{z}_j)} + \log \sum_{k\,:\,y_k\neq y_i} e^{s(\boldsymbol{z}_i,\boldsymbol{z}_k)}\right]\right),\end{aligned} \tag{2.25}$$

where n is the mini-batch size and $\lambda > 0$ is a fixed hyperparameter. The log-sum-exponential terms are smooth approximations to the max function. Thus, if each of the sums are dominated by only one exponential term, Equation 2.25 simplifies to $\max(0, s(\boldsymbol{z}_i, \boldsymbol{z}_k) + \lambda - s(\boldsymbol{z}_i, \boldsymbol{z}_j))$, the standard triplet loss. In general, the lifted structure loss focuses on penalizing the triplet farthest from satisfying the inequality constraint, but considers all other triplets in the mini-batch which can provide additional error signal to the network during training. The lifted structure loss still employs a hard margin via the max operator, however.

The multi-similarity loss [133] is to the lifted structure or triplet loss what binomial deviance is to the pairwise contrastive loss. However, unlike binomial deviance, multi-similarity considers all possible pairs of embeddings within a mini-batch. The multi-similarity loss is defined as:

$$\begin{aligned}&\mathcal{L}((\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_n, y_n)) = \\ &\qquad \frac{1}{n}\sum_{i=1}^{n} \left[\frac{1}{\alpha}\log\left[1 + \sum_{j\,:\,y_j=y_i} e^{\alpha(\lambda - s(\boldsymbol{z}_i,\boldsymbol{z}_j))}\right] + \frac{1}{\beta}\log\left[1 + \sum_{k\,:\,y_k\neq y_i} e^{\beta(s(\boldsymbol{z}_i,\boldsymbol{z}_k) - \lambda)}\right]\right],\end{aligned} \tag{2.26}$$

where $\alpha > 0$, $\beta > 0$, and $\lambda > 0$ are fixed hyperparameters. The multi-similarity loss can be interpreted as a combination of binomial deviance and the lifted structure loss. It is a soft-margin version of the lifted structure loss, but employs α and β, like binomial deviance, to weigh the importance of same-class pairs with small similarity and different-class pairs with large similarity, respectively. Wang et al. [133] motivate the multi-similarity loss by considering the importance-weighted gradients with respect to the same-class and different-class similarities of binomial deviance and the lifted structure loss.

The histogram loss [123], unlike all previous similarity-based losses, is a quadruplet loss, in that it compares two pairs of embeddings, an intra-class pair and an inter-class pair. However, the loss is extremely sample-efficient, only requiring $O(N^2)$ comparisons instead of $O(N^4)$ for all quadruplets. Histogram loss first constructs two sets of similarities, one for same-class pairs and one for different-class pairs: $S^+ = \{s(\boldsymbol{z}_i, \boldsymbol{z}_j) \ |y_i = y_j\}$ and $S^- = \{s(\boldsymbol{z}_i, \boldsymbol{z}_j) \ |y_i \neq y_j\}$. The loss simply estimates the probability of a different-class pair having larger similarity than a same-class pair (i.e., the probability of reverse):

$$\mathcal{L}(p^+, p^-) = \mathbb{E}_{s \sim p^-} \left[\int_{-\infty}^{s} p^+(\boldsymbol{z}) d\boldsymbol{z} \right], \tag{2.27}$$

where the distributions of same-class pairs and different-class pairs, p^+ and p^-, are each estimated as soft-binned, differentiable histograms using S^+ and S^-, respectively. Thus, the loss penalizes the network for any quadruplet where $s^- \in S^- \geq s^+ \in S^+$. In addition to sample efficiency, the only hyperparameter in the loss is the number of histogram bins, to which the loss was empirically determined to be insensitive.

Another class of losses similar, in principle, to the histogram loss are rank-based losses. Rank-based losses take as input a query embedding and a list of support embeddings and encourage the network to rank the support embeddings according to a scoring function. For example, support embeddings that belong to the same class as the query should score higher, and a natural scoring function is the similarity to the query. Both Triantafillou et al. [122] and Cakir et al. [13] optimize average precision (AP):

$$\text{AP} = \sum_{i}^{n} \text{Precision}(i) \Delta \text{Recall}(i) = \sum_{i}^{n} \text{Precision}(i) \left(\text{Recall}(i) - \text{Recall}(i-1) \right), \tag{2.28}$$

where Precision(i) and Recall(i) are the precision and recall evaluated at position i in the ranking of support embeddings. The challenge in optimizing AP, or other retrieval metrics, is that they rely on a discrete sorting operation that is non-differentiable because the prediction is structured. Triantafillou et al. [122] use two optimization frameworks for structured prediction: structured SVMs and direct-loss-minimization. In contrast, Cakir et al. [13] relaxes AP by quantizing similarities

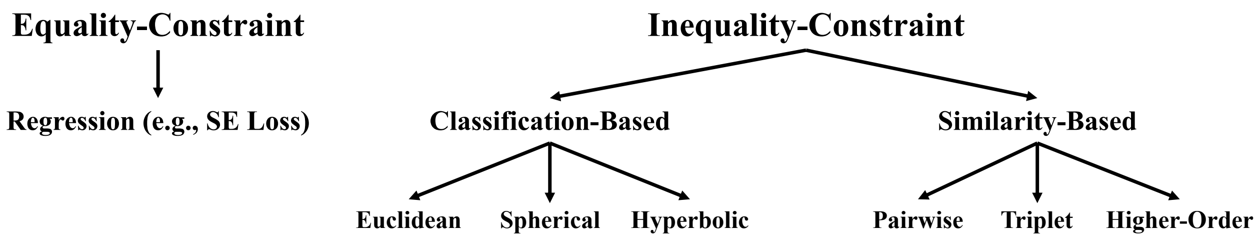


Figure 2.5: Taxonomy of losses for deep visual representation learning. While classification losses vary with the embedding geometry, similarity-based losses are defined generically. Many of the similarity-based losses can operate in any of the geometries.

into a finite set and representing them as soft-binned differentiable histograms, exactly as was done for the histogram loss [123].

2.1.3 Summary of Losses

In this section, we taxonomized losses for discovering deep visual representations, visualized in Figure 2.5. The losses were first subset based on the type of constraint employed: equality based, where a pair of inputs are encouraged to project to the same embedding or inequality based, where same-class embeddings are encouraged to have a smaller distance than different-class embeddings. Far more losses use inequality constraints, as equality-based losses are often too inflexible and may require an additional inequality-constraint term to prevent embedding collapse. The inequality-based losses were further categorized as classification based or similarity based. Classification-based losses use the combination of a logit function with softmax to map embeddings to a categorical distributions over classes. In many cases, the set of predicted classes mirror the C classes in the training set, represented by y_i, however, some predict embedding-level associations (e.g., the n-pairs loss). We detailed classification-based losses operating in each of three possible geometries—Euclidean, spherical, and hyperbolic—highlighting their respective benefits. In contrast to classification-based losses, similarity-based losses operate directly on distance relationships between embeddings where the distance relationships can be computed across pairs, triplets, or higher-order sets of embeddings. Similarity-based losses tend to provide more information to the network about both the local and global structure in the embedding space, but suffer from sample complexity (i.e., $O(N^2)$ for pairs, $O(N^3)$ for triplets, etc.).

2.2 Applications for Visual Representations

Section 2.1 introduced deep visual representations and a variety of recent loss functions for discovering them. In this section, we detail the common classification and retrieval applications that act on the visual representations once they have been learned. Applications that act on top of the representations assume access to the embedding network, f_ϕ, and use it to produce embeddings for a test dataset. Applications can vary in how the test data is used, what is being predicted from the test data, as well as assumptions about the test data itself.

2.2.1 Classification

One of the most common applications is classification where we seek to map from an embedding to a categorical distribution over a set of classes. We distinguish between two types of classification: *fixed-set* classification where the set of classes used during training is exactly the set of classes at test time and *open-set* classification where test classes are disjoint from training classes.

2.2.1.1 Fixed-Set Classification

Fixed-set classification assumes equality between the set of training classes and the set of test classes. If the set of training and test classes are denoted $\mathcal{Y}_{\text{train}}$ and $\mathcal{Y}_{\text{test}}$, respectively, then fixed-set classification enforces $\mathcal{Y}_{\text{train}} = \mathcal{Y}_{\text{test}}$. Even though the classes are identical across the train and test stages, the input instances presented to the network during the test stage are previously unseen. Given the test embeddings, there are many ways of extracting a class prediction. First, representations learned with softmax cross-entropy explicitly map to a categorical distribution over the training classes. Since training and test classes are identical, we can simply utilize the learned logit function for prediction: $\hat{y} = \arg\max_j h(\boldsymbol{z}_{\text{test}}, j)$. For representations learned with a non-softmax loss (e.g., triplet loss), one could embed the validation set and use the embeddings as a

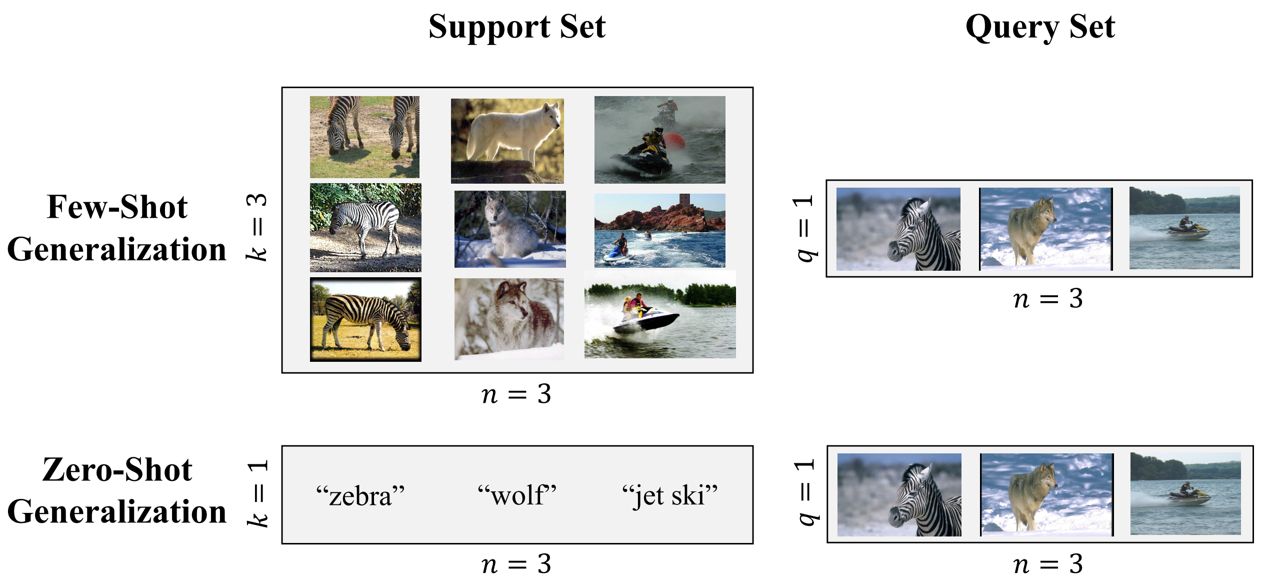


Figure 2.6: Example support set (left column) and query set (right column) for a 3-shot, 3-class few-shot episode (top row), and natural-language zero-shot episode (bottom row). All images are taken from the aPY dataset [31].

support set for k-nearest-neighbor classification of a query test embedding, z_{test}. Another common alternative to k-nearest-neighbor classification is to use the validation set to train a classification head on top of the embeddings (e.g., logistic regression).

2.2.1.2 Open-Set Classification

Open-set classification refers to the setting in which the train classes are disjoint from test classes: $\mathcal{Y}_{\text{train}} \cap \mathcal{Y}_{\text{test}} = \emptyset$. Although, one might consider the more practical setting of *generalized open-set classification* where $\mathcal{Y}_{\text{train}} \subset \mathcal{Y}_{\text{test}}$. Because test classes are previously unseen by the network, the classification procedure is different from the fixed-set sibling. The goal of an open-set classifier is to leverage the representation learned with the training set to generalize to semantically similar, but novel categories. At test time, the network is presented a *support set* containing k instances—sometimes called *shots*—from each of n new classes. The network uses the support set to familiarize itself with the new classes, and then is given a *query set* of instances to classify. The query instances are associated only with the n classes present in the support set, and predicting the class of each query given the support set is commonly referred to as an *episode* or *task*. Some losses are designed specifically for open-set classification which is why they naturally incorporate support and query sets during training (e.g., [112, 126]).

There are many ways of utilizing the support set for classification. One could embed the support set and then perform k-nearest-neighbor classification on queries similar to fixed-set classification. Alternatively, the support set could be used to learn class weight vectors for the n new classes via a softmax cross-entropy loss, and then be used directly for classification of queries [121]. Scott et al. [105] show that backpropagating training signals from the support set through the embedding network (i.e., fine-tuning), particularly with histogram loss [123], can lead to significantly improved open-set classification. Snell et al. [112] construct class prototypes by averaging the support-set embeddings from each class and then classify a query based on proximity to the prototypes.

Open-set classification can be further subset into few-shot classification and zero-shot classification. Few- and zero-shot classification vary in the structure of the test episodes, as well as the modalities present in both the training and test data.

Few-Shot Classification. Let $\mathcal{Y}_{\text{test}}$ be a set of n classes unseen during training. A test episode consists of a support set, $\mathcal{S}$, and a query set, $\mathcal{Q}$:

$$\begin{aligned} \mathcal{S} &= \{(\boldsymbol{x}_{ij}, y_{ij}) | y_{ij} \in \mathcal{Y}_{\text{test}}\}_{i=1:n,\, j=1:k}, \\ \mathcal{Q} &= \{\boldsymbol{x}_{ij}\}_{i=1:n,\, j=k+1:k+q}, \end{aligned} \tag{2.29}$$

where $\boldsymbol{x}_{ij}$ is the jth instance of the ith class in the episode. As previously noted, the support set contains k instances from each of the n unseen classes in $\mathcal{Y}_{\text{test}}$, and the query set contains q instances to be classified. The top row of Figure 2.6 displays a sample 3-shot, 3-class few-shot episode.

Zero-Shot Classification. Zero-shot classification can be formalized similarly, but has a slightly different goal: generalize to query examples from novel classes without having seen any support examples drawn from the same input modality. A common approach to zero-shot classification fuses the learned representation to a representation from an alternate modality [6, 35, 113, 151]. Figure 2.6 displays a zero-shot test episode where the support modality is natural language and the query modality is static imagery. Many zero-shot episodes use natural language as the support modality because each class can typically be summarized with a class label, such as "zebra," for example. If the learned visual representation was somehow tied to a learned—or pretrained—semantic representation, one could embed the word "zebra" through the semantic branch of the model, as well as the query image through the visual branch, and perform classification in the shared space. In Chapter 5, we unify few- and zero-shot classification through the support-query set formalism. For zero-shot, the support and query sets are defined as:

$$\begin{aligned} \mathcal{S} &= \{(\boldsymbol{x}_i^{\ell}, y_i) | y_i \in \mathcal{Y}_{\text{test}}\}_{i=1:n}, \\ \mathcal{Q} &= \{\boldsymbol{x}_{ij}^{v}\}_{i=1:n,\, j=1:q}, \end{aligned} \tag{2.30}$$

where superscript ℓ denotes natural-language inputs, superscript v denotes visual inputs, and the indices match those from Equation 2.29. For the sake of simplicity, we assume support instances are semantic and query instances are visual, but other pairs exist such as class-attribute vectors for support and audio for query. In many cases, the support set contains a single instance per class, since the class can be fully specified by its natural-language description or class-attribute vector. For this reason, we drop the second subscript for support examples.

2.2.2 Clustering

Clustering is a technique commonly applied on top of a pretrained representation in which embeddings are grouped based on their proximity to other embeddings or proximity to learned centroids. It can be useful, in general, for model interpretability, compression, visualization, outlier detection, but specifically for zero-shot classification. Section 2.2.1.2 detailed the primary approach for zero-shot classification, which is fusing representations from different modalities, but another seemingly less-explored approach is clustering. Some benefits of clustering for zero-shot classification are that first, there is no functional need for the support set, and second, with no need for the support set, only the visual representation is required and it does not need to be fused to a representation from an alternate modality. One potential issue with clustering for zero-shot classification, however, is that once embeddings are grouped, it may be difficult to associate each cluster with a class label without human intervention. As is popular for face verification systems, pseudo-supervising data via zero-shot cluster assignments can be useful for enlarging labeled datasets for training larger and better-performing embedding networks (e.g., [108, 150]).

2.2.3 Retrieval

Systems such as search engines, recommendation engines, and databases, or identification tasks such as person re-identification [145], rely on content-based retrieval, that is, retrieving stored support instances that are most similar to a query instance. Such systems and tasks rely on embedding spaces where distance is a proxy for semantic dissimilarity. Retrieval is a task that directly probes how well embeddings are clustered by their content and thus is not only used in production settings, but is an informative way of evaluating the quality of visual representations. When both the query instance and support instances are supervised, one can embed the query and retrieve the nearest support embeddings, where the retrieval is deemed "correct" when the nearest neighbors are the same class as the query. Recall@R [52], R-precision [84], mean-average-precision [79], and mean-average-precision@R [84] are popular retrieval metrics, and all attempt to measure the representation's ability to either rank relevant instances or return a relevant instance in the top R retrieved.

2.2.4 Verification

Classification and retrieval applications consider a single query embedding and compare it to a set of stored embeddings from many classes (i.e., a one-to-many comparison). Verification tasks—prolific in face recognition and credit-card fraud detection, among other domains—compare a query to an *enrollment* from a single user (i.e., a one-to-one comparison). The enrollment could be a set of embeddings, an average or prototypical embedding, or an estimated distribution, all of which serve to represent a single user and its intra-user variance. As an example, consider a simplified fraud detection system that uses geographical location as a representation. A user will have many transactions within a radius of their home or place of employment, serving as an enrollment. A query transaction that is too far away from the stored enrollment instances could be flagged as fraudulent.

Verification tasks typically leverage classification or retrieval when deciding if a query belongs to the enrolled user. One approach is to train a linear classifier on top of the representation that accepts the query and enrollment as input and predicts binary association directly. A second approach is to threshold the distance to the enrollment where the threshold is estimated from held-out data. One could construct a synthetic verification task by taking a test dataset and forming pairs of embeddings, labeling them as either a *genuine* pair (i.e., from the same user) or an *impostor* pair, and quantifying false-reject and false-accept rates. Empirical analyses such as these are common in face verification, particularly (e.g., [28, 75, 129, 130]).

2.2.5 Inductive Transfer Learning

A final application is inductive transfer learning which refers to learning a representation on a *source* dataset or task that is then transferred and fine-tuned on a related *target* task. Bootstrapping the representation for a target task leads to computationally-efficient training and better generalization compared to starting from random weights [146], and the transferred representation can be successfully fine-tuned only with a small target dataset [61, 105]. For example, many networks are initialized using weights pretrained on ImageNet [27] (e.g., [10, 13, 84, 133]). In addition, recent methods based on self-supervised learning (e.g., [19, 45]) provide the ability to learn a general representation without the need for labeled source data, and the representation can be fine-tuned downstream. Once a representation has been transferred and fine-tuned, any of the above applications can be employed on the new representation. We note that few-shot classification could be categorized as inductive transfer learning, particularly when one fine-tunes on the support set.

2.2.6 Summary of Applications

Deep visual representations have led to significant improvements on downstream classification and retrieval applications that use the embeddings including fixed-set classification, open-set classification, clustering, retrieval, verification, and inductive transfer learning. Classification is where an embedding is mapped to a categorical distribution over classes. When using softmax cross-entropy, classification comes for free through the loss function, assuming a fixed set of classes for training and testing. Open-set classification, including few- and zero-shot classification, attempt to generalize beyond the training classes to novel, but semantically-similar test classes. Clustering can be viewed as a form of zero-shot classification where embeddings are grouped typically via proximity to each other or proximity to estimated centroids. Retrieval and verification tasks also rely on distance relationships to other embeddings. Retrieval systems want to find semantically-similar embeddings to a query, for example as recommendations, while verification systems attempt to extract a binary signal: does the query embedding belong to the enrolled user? Finally, inductive transfer learning is a general application where a learned representation—typically trained on a very large dataset—is transferred and fine-tuned on a smaller target dataset for improved training efficiency and generalization.

Chapter 3

Stochastic Prototype Embeddings[*]

Nearly all previous losses, or methods, for discovering deep visual representations operate on deterministic embeddings, where an input projects to a single point in the embedding space. Deterministic embeddings fail to capture uncertainty due either to data corruption or class ambiguity. Representing uncertainty is important for many reasons, including robust classification and retrieval, informing downstream models, and interpreting representations. In this chapter, we propose a novel Euclidean method for discovering stochastic embeddings, where each embedded instance is a random variable whose distribution reflects its uncertainty in the embedding space.

Our proposed method, the *Stochastic Prototype Embedding* (SPE), is an extension of the *Prototypical Network* (PN) [112]. PN computes a prototypical embedding, $\boldsymbol{\rho}_y$, for class, y, via an arithmetic mean of a set of support embeddings belonging to it, and ensures that query embeddings from that class, $\boldsymbol{z}_y$, satisfy a Euclidean proximity constraint: $||\boldsymbol{z}_y - \boldsymbol{\rho}_y|| < ||\boldsymbol{z}_y - \boldsymbol{\rho}_j|| \; \forall \; j \neq y$. For further details, see Equation 2.9. As with PN, SPE assumes each class can be characterized by a prototype in the embedding space and a query embedding is classified based on its proximity to a prototype. In the case of SPE, the embeddings and prototypes are Gaussian random variables, each embedding belonging to a specific class is assumed to be a Gaussian perturbation of the class's prototype, and a query embedding is classified by marginalizing over the embedding uncertainty. Our main contribution is to show that SPE outperforms the only other fully-formulated method

[*]Scott, T. R., Ridgeway, K., and Mozer, M. C. (2019). Stochastic Prototype Embeddings. In ICML Workshop on Uncertainty and Robustness in Deep Learning.

for discovering stochastic, supervised, visual representations at the time of publication, the *Hedged Instance Embedding* (HIB) [88], on a superset of the experiments used to justify HIB. SPE is also more computation-efficient to train compared to HIB, with complexity comparable to that of PN. We also demonstrate that embedding distributions are related to class uncertainty and input ambiguity. Finally, we explore an intriguing emergent property of SPE: that it attains more interpretable representations by disentangling class-discriminative features.

3.1 Related Work

Methods for discovering stochastic, Euclidean embeddings—not necessarily for deep visual representation learning—have begun appearing in the literature. Allen et al. [2] extend PNs via Bayesian nonparametrics to treat each prototype as a mixture distribution, though they do not allow uncertainty in the embedding space, the critical element of our research. Vilnis and McCallum [125] propose a method for learning density-based word embeddings, but it is unsupervised. Deep Variational Transfer [8] is a generative form of the discriminative model we propose, which requires modeling input distributions; this work tackled the somewhat different problem of covariate shift. Likewise, the Oracle-Prioritized Belief Network (OPBN) [54] is a generative model that learns a joint distribution over inputs and oracle-provided triplet constraints, and is evaluated on linear-readout of the factors of variation present in the data. The method closest to ours is an extension of PNs that uses a Mahalanobis distance instead of a Euclidean distance to assess similarity [33]. Although this method lacks probabilistic semantics, it has similarity with SPE. We discuss the comparison further in Section 3.4.

The only prior method proposed for discovering stochastic, supervised embeddings for visual representation learning is the *Hedged Instance Embedding* (HIB) [88]. HIB utilizes a soft, probabilistic alternative to the pairwise contrastive loss (i.e., Equation 2.19) and is trained using a variational approximation to the information bottleneck principle. HIB is critically dependent on a constant, β, that determines characteristics of an information bottleneck (i.e., how much of the input entropy is retained in the embedding).

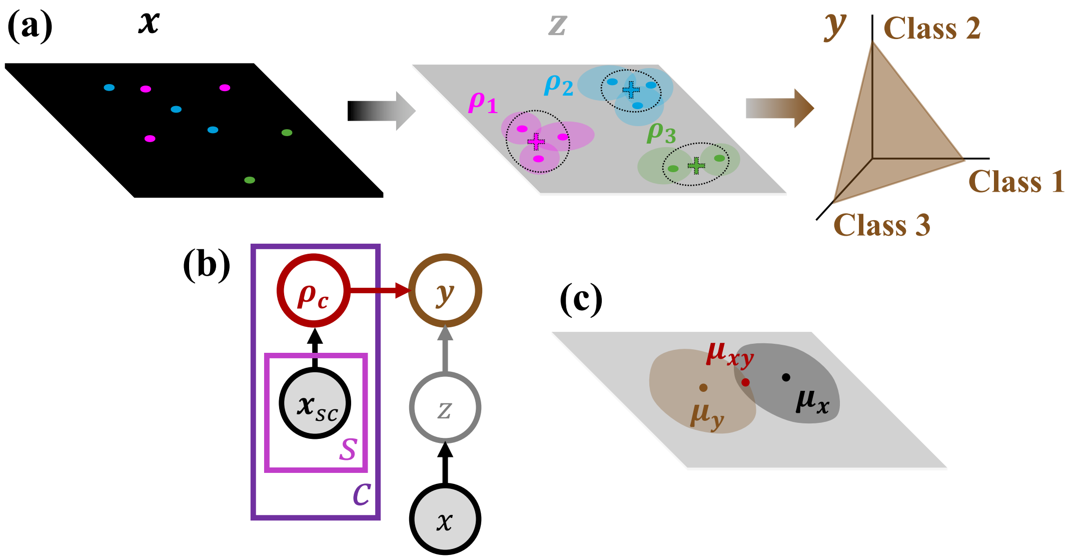


Figure 3.1: (a) Illustration of the SPE. SPE learns a mapping from input space, $\boldsymbol{x}$, to embedding space, $\boldsymbol{z}$, such that same-class instances are near and different-class instances are far. Embeddings are Gaussian random variables. A class prototype posterior, denoted by the + symbol, is obtained via a confidence-weighted average of the embeddings of instances known to belong to a class. Prototype uncertainty is depicted with the dotted ovals. Given the prototypes, a prediction of class y is made for a query instance by marginalizing a softmax prediction over the embedding space. (b) Graphical model underlying SPE. (c) Depiction of the intersection sampler.

3.2 Stochastic Prototype Embeddings

SPE assumes that the latent representation, $\boldsymbol{z}$, is a Gaussian random variable conditioned on the input, $\boldsymbol{x}$:

$$p(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{x}}, \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{x}})), \tag{3.1}$$

with mean, $\boldsymbol{\mu}_{\boldsymbol{x}}$, and variance, $\boldsymbol{\sigma}^2_{\boldsymbol{x}}$, computed by a deep neural network, similar to a Variational Autoencoder [58]. The classification, y, in turn is conditioned on $\boldsymbol{z}$, with $p(y|\boldsymbol{z})$ taking the same form as in the original PN [112], to be described shortly. Given an input, a class prediction is made by marginalizing over the embedding uncertainty:

$$p(y|\boldsymbol{x}) = \int_{\boldsymbol{z}} p(y|\boldsymbol{z})p(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z}. \tag{3.2}$$

Informally, Figure 3.1(a) depicts the relationship between the input, latent, and class representations. We train SPE using the standard paradigm for few-shot classification, via a sequence of *episodes*, each with $k + q$ inputs from each of n classes. We split the $(k + q) \times n$ inputs into $k \times n$

supervised *support* examples (i.e., the support examples are paired with ground-truth class annotations), defining a set S, and $q \times n$ *query* examples. The support instances for each class c, $S_c \subset S$, are used to determine the class prototype, $\boldsymbol{\rho}_c$, and the query instances are used to predict a class via Equation 3.2. Figure 3.1(b) shows the graphical model underlying SPE, which captures the relationship between the support instances, the class prototypes, and query classification. Note that SPE is framed as a discriminative model in contrast to Belhaj et al. [8], which can be interpreted as the generative version of our model.

3.2.1 Forming Class Prototypes

In SPE, each class, c, has an associated prototype, $\boldsymbol{\rho}_c$, in the embedding space, and the ith input of class c, denoted $\boldsymbol{x}_i$, projects to an embedding, $\boldsymbol{z}_i$, in the neighborhood of $\boldsymbol{\rho}_c$ such that:

$$\boldsymbol{\rho}_c = \boldsymbol{z}_i + \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma_\epsilon^2 \boldsymbol{I}). \tag{3.3}$$

We assume that the prototype is consistent with all support embeddings, allowing us to posit the likelihood of $\boldsymbol{\rho}_c$ as a product distribution:

$$p(\boldsymbol{\rho}_c | S_c) = \frac{\prod_{i \in S_c} p(\boldsymbol{\rho}_c | \boldsymbol{x}_i)}{\int_{\boldsymbol{\rho}} \prod_{i \in S_c} p(\boldsymbol{\rho} | \boldsymbol{x}_i) d\boldsymbol{\rho}}. \tag{3.4}$$

Because $p(\boldsymbol{\rho}_c|\boldsymbol{x}_i)$ is Gaussian, the resulting normalized product is too:

$$\boldsymbol{\rho}_c | S_c \sim \mathcal{N}(\boldsymbol{\mu}_c, \operatorname{diag}(\boldsymbol{\sigma}_c^2)) \text{ with } \boldsymbol{\sigma}_c^2 = \left(\sum_{i \in S_c} \hat{\boldsymbol{\sigma}}_{\boldsymbol{x}_i}^{-2} \right)^{-1} \text{ and } \boldsymbol{\mu}_c = \boldsymbol{\sigma}_c^2 \circ \left(\sum_{i \in S_c} \hat{\boldsymbol{\sigma}}_{\boldsymbol{x}_i}^{-2} \circ \boldsymbol{\mu}_{\boldsymbol{x}_i} \right), \tag{3.5}$$

where $\hat{\boldsymbol{\sigma}}_{\boldsymbol{x}_i}^2 = \boldsymbol{\sigma}_{\boldsymbol{x}_i}^2 + \sigma_\epsilon^2$ and $\circ$ denotes the Hadamard product. Essentially, the prototype is a confidence-weighted average of the support embeddings. This formulation has a clear advantage over the deterministic PN—which is premised on an unweighted average—because it de-emphasizes uncertain support embeddings.

3.2.2 Prediction and Approximate Inference

We posit a softmax prediction, like PN, for a query embedding, $\boldsymbol{z}$:

$$p(y | \boldsymbol{z}, \boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_n) \propto \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_y, \operatorname{diag}(\hat{\boldsymbol{\sigma}}_y^2)) \tag{3.6}$$

with $\hat{\boldsymbol{\sigma}}_y^2 = \boldsymbol{\sigma}_y^2 + \sigma_\epsilon^2$ as before, yielding the class posterior for query $\boldsymbol{x}$:

$$p(y|\boldsymbol{x}, \boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_n) = \int_{\boldsymbol{z}} \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{x}}, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}}^2)) \frac{\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_y, \mathrm{diag}(\hat{\boldsymbol{\sigma}}_y^2))}{\sum_c \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_c, \mathrm{diag}(\hat{\boldsymbol{\sigma}}_c^2))} d\boldsymbol{z}. \tag{3.7}$$

The class distribution is equivalent to that produced by the deterministic PN as $\boldsymbol{\sigma}_x^2 \to \mathbf{0}$ when $\boldsymbol{\sigma}_y^2 = \boldsymbol{\sigma}_{y'}^2 = \frac{1}{2}$ for all class pairs (y, y'). However, in the general case, the integral has no closed form solution; thus, we must sample to approximate $p(y|\boldsymbol{x}, \boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_n)$, both for training and testing. We employ two samplers, which we refer to as *naïve* and *intersection*.

3.2.2.1 Naïve Sampling

A direct approach to approximating the class posterior is to express Equation 3.2 as an expectation, $\mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z}|\boldsymbol{x})}\left[p(y|\boldsymbol{z}, \boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_n)\right]$, and to replace the expectation with the average over a set of samples. We utilize the reparameterization trick of Kingma and Ba [58] to train the model. Although this Monte Carlo method is the simplest approach, it is sample-inefficient during training, and when the number of samples is reduced, model performance is impacted, as we will show below.

3.2.2.2 Intersection Sampling

In Equation 3.7, the product of Gaussian densities in the numerator can be rewritten:

$$\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{x}}, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}}^2)) \mathcal{N}\left(\boldsymbol{z}; \boldsymbol{\mu}_y, \mathrm{diag}(\hat{\boldsymbol{\sigma}}_y^2)\right) = \mathcal{N}\left(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{x}y}, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}y}^2)\right) \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}}; \boldsymbol{\mu}_y, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}}^2 + \hat{\boldsymbol{\sigma}}_y^2)\right), \tag{3.8}$$

where $\boldsymbol{\sigma}_{\boldsymbol{x}y}^2 = (\boldsymbol{\sigma}_{\boldsymbol{x}}^{-2} + \hat{\boldsymbol{\sigma}}_y^{-2})^{-1}$ and $\boldsymbol{\mu}_{\boldsymbol{x}y} = \boldsymbol{\sigma}_{\boldsymbol{x}y}^2 \circ (\boldsymbol{\sigma}_{\boldsymbol{x}}^{-2} \circ \boldsymbol{\mu}_{\boldsymbol{x}} + \hat{\boldsymbol{\sigma}}_y^{-2} \circ \boldsymbol{\mu}_y)$. Substituting Equation 3.8 into Equation 3.7:

$$p(y|\boldsymbol{x}, \boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_n) = \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}}; \boldsymbol{\mu}_y, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}}^2 + \hat{\boldsymbol{\sigma}}_y^2)\right) \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}y}, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}y}^2)\right)} \left[\sum_c \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_c, \mathrm{diag}(\hat{\boldsymbol{\sigma}}_c^2))\right]^{-1}. \tag{3.9}$$

By approximating the expectation with samples from $\mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}y}, \mathrm{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}y}^2)\right)$, we obtain an elegant importance sampler that focuses on the intersection of the input distribution and a given class distribution, as illustrated in Figure 3.1(c). During training with cross-entropy loss, we need only

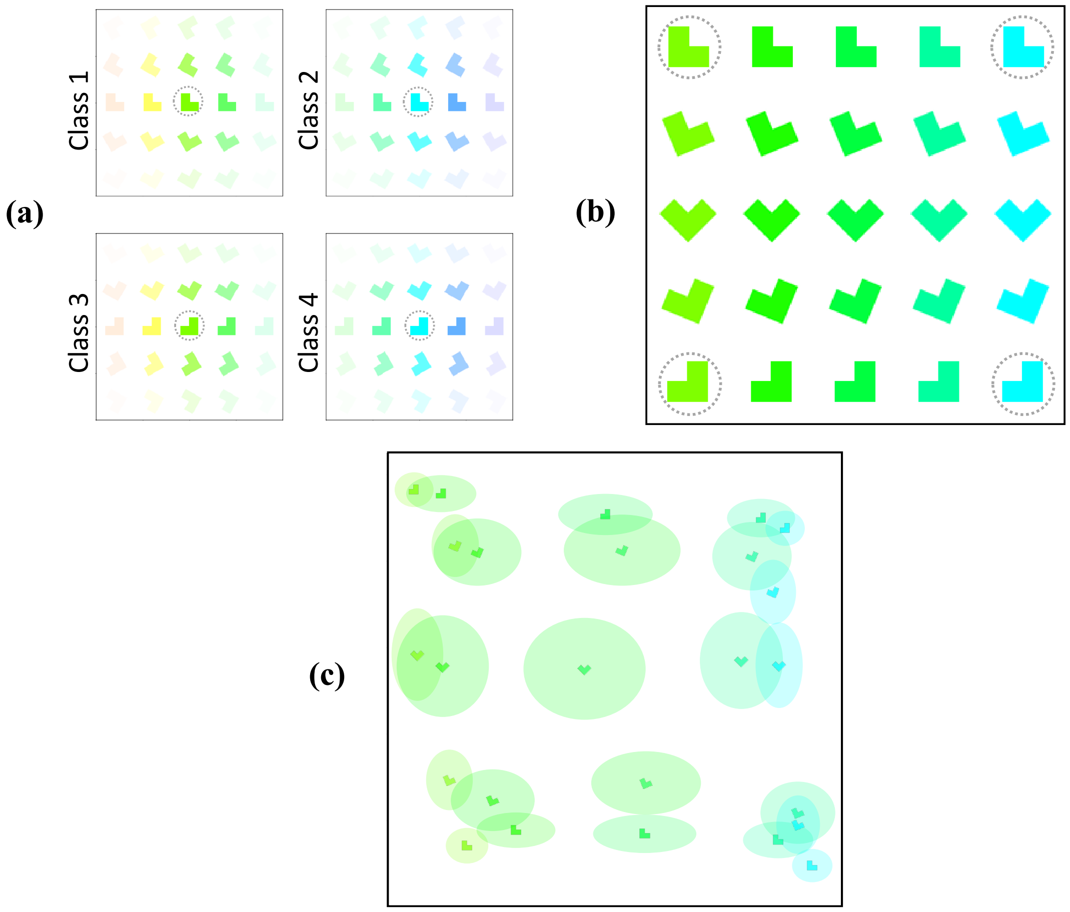
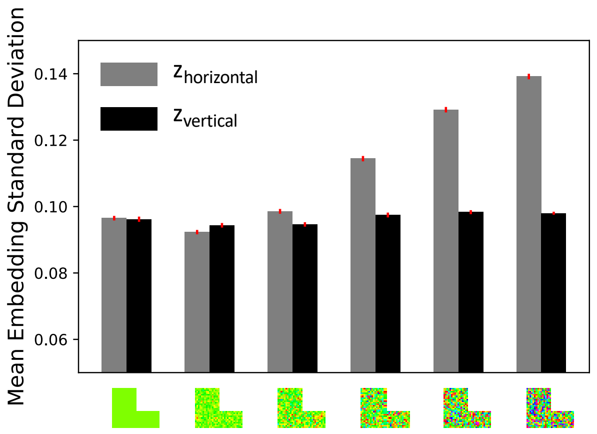


Figure 3.2: (a) Samples from the four classes in our synthetic dataset. In each plot, class centroids are circled, along with samples spanning ± 2 standard deviations in both orientation and color. A sample's transparency is set according to its class-conditional likelihood. Both dimensions can be coded as directional variables. The class centroids on each dimension are $90°$ apart with standard deviation of $30°$. (b) A set of examples, with the four class centroids located in the corners and other examples obtained by linear interpolation in the generative space. (c) The 2D stochastic prototype embedding for the examples in (b). The shape is plotted at the mean of $p(\boldsymbol{z}|\boldsymbol{x})$, and the outlines of the ovals represent equiprobability contours at 0.4 standard deviations.

sample for the known, target class, y. As we will demonstrate, this method is more robust and significantly more sample-efficient than the naïve sampler, requiring only a *single* sample to train effectively.



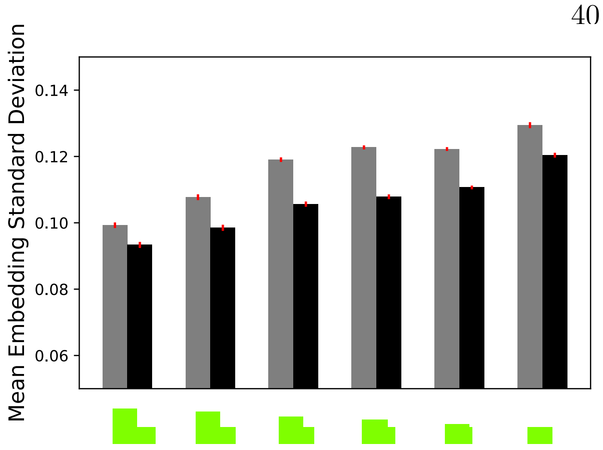


Figure 3.3: Synthetic data set: uncertainty on the two embedding dimensions as it becomes more difficult to discern the hue (left) and orientation (right).

3.3 Experimental Results

We report on three sets of experiments. In Section 3.3.2, we demonstrate, using a synthetic dataset, that SPE infers the generative structure of a domain, disentangles class-discriminating features, and provides meaningful estimates of class uncertainty and input noise. In Section 3.3.3, we show that SPE obtains impressive results on few-shot classification via a comparison to its deterministic sibling, PN, a go-to method due to its strong performance and simplicity. We evaluate on a standard dataset used to compare methods in the few-shot classification literature, Omniglot [68]. In Section 3.3.4, we show that SPE obtains state-of-the-art results on large-set classification via a comparison to the only other fully-developed stochastic method, at the time of publication, for supervised, visual representation learning: HIB [88]. We evaluate on the only dataset that Oh et al. [88] used to explore HIB, a multi-digit variant of MNIST. Before presenting experimental results, we describe methodological details for training SPE.

3.3.1 Methodological Details

For all SPE models,

$$\sigma_\epsilon^2 = \text{softplus}\left(\gamma\right),$$

where γ is a trainable parameter. We initialize γ using the following prescription:

$$\gamma = |S_c|\gamma_0^{2/d},$$

where $|S_c|$ is the number of support examples per class per episode during training and d is the dimensionality of the embedding. We chose this prescription for two reasons: (1) as the number of support examples increases, the variance of the prototype distribution approaches zero, so scaling linearly by $|S|$ tends to provide a stronger training signal early on, and (2) the amount of noise in the projection of an embedding should scale with the dimensionality of the embedding space as to maintain unit-volume. All models use $\gamma_0 = 0.01$. Additionally, the variance of each dimension i, $\sigma^2_{\boldsymbol{x}_i}$, is guaranteed to be non-negative by using a softplus transfer function.

Whether trained with the naïve or intersection sampler, we evaluate model performance using the naïve sampler with 200 samples. This approach ensures that we are comparing the quality of models based only on the method by which they were trained.

3.3.2 Synthetic Color-Orientation Dataset

The synthetic color-orientation dataset consists of 64×64 pixel images of 'L' shapes, with four classes that are distinguished by orientation, color, or both, as seen in Figure 3.2(a). Inputs are sampled from a class-conditional isotropic Gaussian distribution in the generative space. (The isotropy of these qualitatively different dimensions comes from the fact that both can be mapped as directional quantities.) Because classes overlap on both color and orientation dimensions, elicited embeddings should indicate increased uncertainty near class boundaries.

For orientation, we chose class centers at 90° and 180°, with a standard deviation of 30°. For color, we manipulated the hue and kept the value and saturation constant. Like orientation, hue is a circular quantity. If hue ranges from 0 to 360 degrees, we chose color class centers and standard deviation in the same way as orientation. Additionally, we add noise to a minority, specifically 15%, of the images used to train the model. For these, we add Gaussian noise to the hue of each pixel inside the shape. The standard deviation of the hue noise was chosen uniformly between 18° and

54°. We also added noise to the leg lengths of the 'L' shapes. The leg length was chosen uniformly between 10% and 98% of its original length. See Figure 3.3 for some examples.

The network consists of six convolutional blocks. The first five blocks have a convolutional layer with 64 filters, a 3×3 kernel, zero-padding of length 1, and a stride of 1, followed by a batch normalization layer, ReLU activation, and 2×2 max-pooling. The sixth and final block has a convolutional layer with $2 \times d$ filters, a 3×3 kernel, zero-padding of length 1, and a stride of 1, followed by 2×2 max-pooling, where d represents the dimensionality of the embedding space. The flattened output of the network is a vector of length $2d$, where the first d elements were considered the mean of the Gaussian distribution and the remaining d elements were the diagonal covariance entries. The weights are initialized using He initialization and the biases with the following uniform distribution: $\mathcal{U}(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}})$. The network is trained with a learning rate of 0.0001 and the models are stopped early using a patience parameter when performance on the validation set no longer increases.

We trained a two-dimensional, intersection-sampling SPE on samples from this domain, using two instances per class to form prototypes. Classification accuracy of held-out examples is approximately 86%. Accounting for class overlap, a Bayes optimal classifier has an accuracy of approximately 87%. For visualization, Figure 3.2(b) presents a 5×5 array of examples with the class centroids in the corners and the other examples obtained by linear interpolation in the generative space. The resulting embeddings are presented in Figure 3.2(c). Although the correspondence between Figures 3.2(b) and 3.2(c) seems trivial (i.e., mirror one set along the horizontal axis to obtain the other set), remember that the input space is 64×64-dimensional and the latent space is 2-dimensional. The network has captured the structure of the domain by disentangling the two factors of variation. Further, the embedding variance encodes class ambiguity; embeddings halfway between two classes on one dimension have maximal variance along that dimension. Class ambiguity is one type of uncertainty. An equally important source of uncertainty comes from noisy inputs. We examined noisy inputs generated in two different ways. In the left panel of Figure 3.3, we show the consequence of adding pixel hue noise to the four class centroids. Only one of these centroids is shown along the abscissa, but all four are used to make the graph, with many samples per noise level. The grey and black bars in the graph indicate variance on the horizontal and vertical dimensions of the embedding space, respectively. As pixel-hue noise increases, uncertainty in color grows but uncertainty in orientation does not. In the right panel of Figure 3.3, we show the consequence of shortening the leg-length of the shape. Shortening the legs removes cues that can be used both for determining color and orientation. As a result, the uncertainty grows on both dimensions.

3.3.3 Omniglot

The Omniglot dataset contains images of labeled, handwritten characters from diverse alphabets. Omniglot is one of the standard datasets for comparing methods in the few-shot classification literature. The dataset contains 1623 unique characters, each with 20 inputs. Following Snell et al. [112], each grayscale image is resized from 32×32 to 28×28, and we augment the original classes with all 90° rotations, resulting in 6492 total classes. We train PNs and SPEs episodically, where a training episode contains 60 randomly sampled classes and 5 query instances per class with the number of support instances varying per experiment and identical during training and testing. All test episodes contain 15 queries per class.

For all Omniglot experiments, the network consists of four convolutional blocks. The first three blocks have a convolutional layer with 64 filters, a 3×3 kernel, zero-padding of length 1, and a stride of 1, followed by a batch normalization layer, ReLU activation, and 2×2 max-pooling. The fourth and final block has a convolutional layer with $2d$ filters, a 3×3 kernel, zero-padding of length 1, and a stride of 1, followed by 2×2 max-pooling, where d represents the dimensionality of the embedding space. The flattened output of the network is a vector of length $2d$, where the first d elements were considered the mean of the Gaussian distribution and the remaining d elements were the diagonal covariance entries. The weights are initialized using He initialization and the biases with the following uniform distribution: $\mathcal{U}(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}})$. All Omniglot models are trained with an initial learning rate of 0.001 which is cut in half every 50 epochs. The models are stopped early using a patience parameter when performance on the validation set no longer increases.

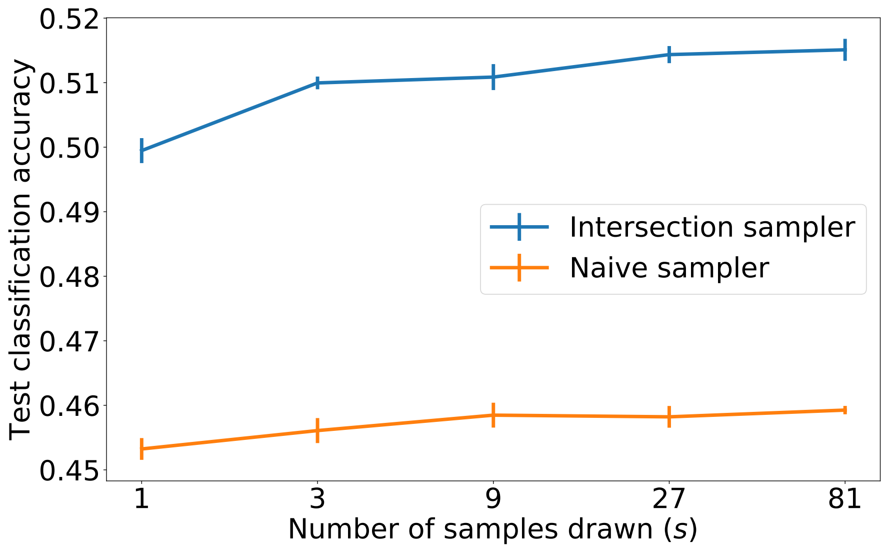


Figure 3.4: Test classification accuracy as a function of number of training samples per query instance for a naïve-sampling and intersection-sampling 2D SPE on a 1-shot, 20-class Omniglot task. Performance is a mean over 5 replications of running the model, showing ±1 standard error of the mean.

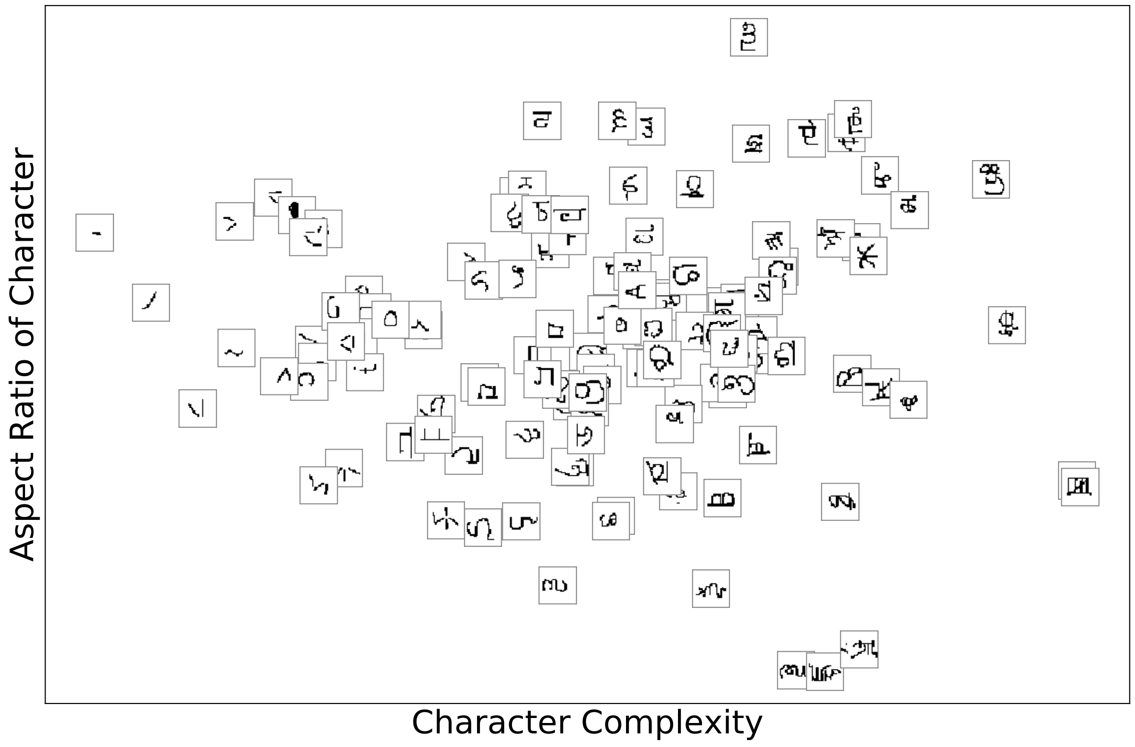


Figure 3.5: 2D embedding learned by the SPE on the Omniglot test set. Each square thumbnail image in the figure is a randomly-sampled instance from one of 125 randomly-sampled test classes and the location of the image represents the location of the class prototype. The images have a gray bounding box for visualization purposes only.

Table 3.1: Test classification accuracy (%) on Omniglot with both a 2D and 64D embedding for clean-support & clean-query, corrupt-support & clean-query, and clean-support & corrupt-query. PN is our implementation of Prototypical Networks [112]. SPE is our model. SPE is trained with intersection sampling and 1 sample per trial. Reported accuracy for each experimental configuration is the mean over 1000 randomly-sampled test episodes. Boldface indicates the best-performing method.

2D Clean Support, Clean Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	75.7	**82.6**	45.0	**55.9**	64.8
SPE	**76.9**	82.3	**49.7**	55.3	**66.1**

2D Corrupt Support, Clean Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	50.0	65.9	23.6	31.7	42.8
SPE	**50.7**	**73.9**	**25.6**	**41.6**	**48.0**

2D Clean Support, Corrupt Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	**48.9**	**52.3**	21.7	25.6	37.1
SPE	47.8	**52.3**	**22.8**	**26.8**	**37.4**

64D Clean Support, Clean Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	**98.5**	**99.6**	**94.9**	**98.6**	**97.9**
SPE	**98.5**	99.5	**94.9**	**98.6**	**97.9**

64D Corrupt Support, Clean Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	85.0	98.7	68.5	95.6	87.0
SPE	**85.7**	**98.8**	**69.3**	**95.7**	**87.4**

64D Clean Support, Corrupt Query

	1-shot, 5-class	5-shot, 5-class	1-shot, 20-class	5-shot, 20-class	Mean
PN	**80.9**	**84.9**	**66.7**	**74.7**	**76.8**
SPE	80.3	84.8	66.3	**74.7**	76.5

To compare the relative effectiveness of the naïve and intersection samplers, we train SPE on Omniglot varying both the sampler and the number of samples drawn per training query, denoted by s. We evaluate in a 1-*shot*, 20-*class* setting, where shot refers to the number of support examples used to compute each prototype and class refers to the number of candidate test classes per episode. Figure 3.4 shows test classification accuracy as the number of samples drawn per training trial, s, increases. As we previously stated, the intersection-sampling SPE is far more sample-efficient, to the point that the intersection sampler with $s = 1$ outperforms the naïve sampler with $s = 81$. We have verified that the pattern in Figure 3.4 is consistent across simulations; consequently, we present only intersection-sampling SPE results in the remainder of the chapter, and all SPEs are trained with a single sample (i.e., $s = 1$) per query. This choice causes the SPE to be on par with the PN in time and space requirements, even though using more samples may boost classification accuracy, as suggested by the trend in Figure 3.4.

Figure 3.5 is a visualization of a 2D embedding learned by the intersection-sampling SPE on Omniglot. All classes shown in the figure were held-out during training. Omniglot characters clearly vary along more than two dimensions, so a 2D SPE cannot learn a fully-disentangled representation as it did with the synthetic dataset. However, we can still interpret the axes of the embedding. The horizontal axis appears to represent character complexity, with single-stroke characters on the left and many-stroke characters on the right. The vertical axis appears to encode the aspect ratio of the characters, with horizontally-extended characters on the bottom and vertically-extended characters on the top.

Table 3.1 compares PN and SPE with both 2D and 64D embeddings on Omniglot test classes. Each entry in the table represents classification accuracy over 1000 randomly-sampled test episodes for a particular few-shot condition and method. The rightmost column shows average test classification accuracy over all reported few-shot conditions (i.e., average accuracy across 1-shot & 5-class, 5-shot & 5-class, 1-shot & 20-class, and 5-shot & 20-class). The tables denoted with the suffix "Clean Support, Clean Query" consider the standard procedure for comparing methods where the unaltered support set is used to obtain an embedding for each class, prototypes are formed, and unaltered query instances are classified. In this setting, SPE outperforms PN in the constrained 2D representation, but is equivalent in 64D.

Because the Omniglot data are carefully curated, the inputs have little noise and therefore offer little opportunity to leverage SPE's assessment of uncertainty. Consequently, we corrupted instances by masking out rectangular regions of the input, as proposed by Oh et al. [88]. (See Appendix A.2 for details.) The tables denoted with the suffix "Corrupt Support, Clean Query" and "Clean Support, Corrupt Query" consider when the support instances and query instances are corrupted, respectively. SPE's advantage over PN increases significantly when the support instances are corrupted due to the fact that SPE's confidence-weighted prototypes (i.e., Equation 3.5) discount noisier support embeddings. SPE and PN perform similarly when query instances are corrupted, although SPE has a slight edge in 2D where uncertainty likely plays a larger role in the classification decision.

To emphasize, SPE outperforms PN, arguably the leading few-shot classification method at the time of publication, particularly in 2D—where class-overlap in the latent space is more prevalent, and when support inputs are corrupted, at essentially the same computational cost for training. And by providing an estimate of uncertainty associated with embedded instances, SPE results in more interpretable representations and can inform downstream systems that operate on the representation.

3.3.4 N-Digit MNIST

The N-digit MNIST dataset was proposed to evaluate HIB [88]; it is formed by horizontal concatenation of N MNIST digit images. The resulting images are $28 \times 28N$. To compare with HIB, we study 2- and 3-digit MNIST. Oh et al. [88] split the data into a training set with 70% of the total classes, a *seen* test set, and an *unseen* test set. For 2-digit MNIST, the seen test set has the same 70 of 100 classes as the training set and the unseen test set has the remaining 30 classes. For 3-digit MNIST, the training set has 700 classes, the seen and unseen test sets each have a sample of 100 of the 700 seen or 300 unseen classes, respectively. We use the same train and test data splits as Oh et al. [88], but we further divide the training split to include a validation set for early stopping.

For all N-digit MNIST experiments, we constructed an architecture which we believe to

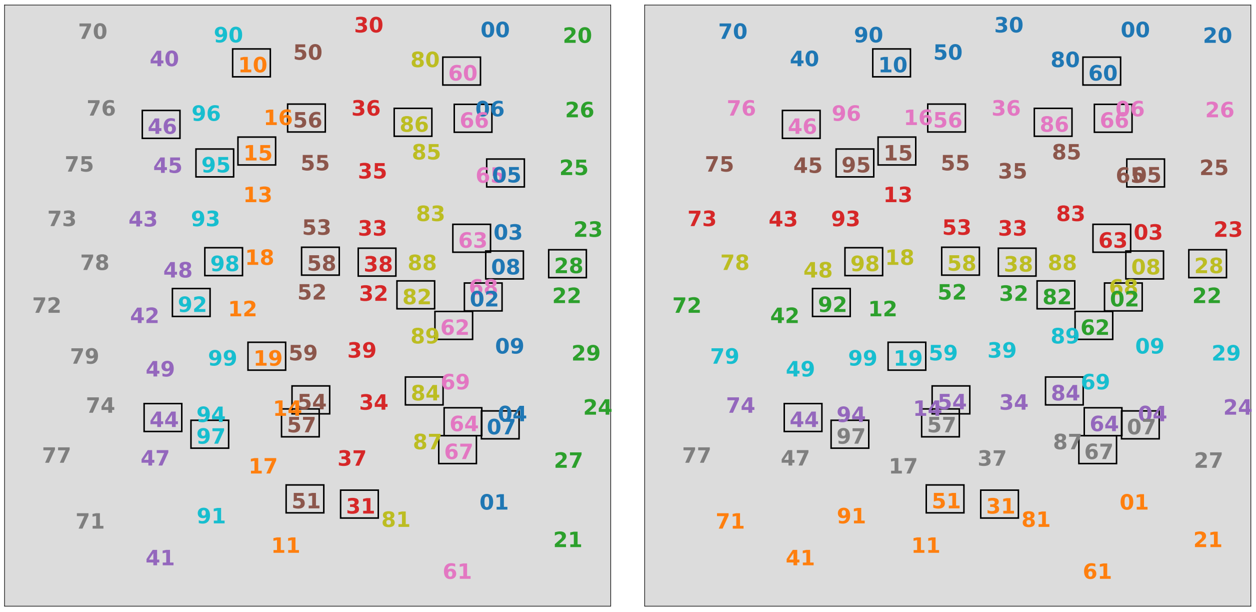


Figure 3.6: Two views of the 2D embedding learned by SPE on the 2-digit MNIST test set. Each number is a class label; for example, 71, located in the lower left of the embedding, is the class in which the first of the two MNIST digits is a 7 and the second is a 1. The same embedding is shown in the left and right plots, but the left plot is colored according to the first digit, the right plot according to the second digit. The location of a class label indicates the mean of its prototype, using 140 support instances to form prototypes. The digits surrounded by a black border are classes whose instances were unseen during training.

be identical to that used for HIB MNIST experiments. The network consists of two convolutional blocks followed by two fully-connected layers. The convolutional blocks each contain a convolutional layer, followed by an ReLU activation, and 2×2 max-pooling. The first convolutional layer has 6 filters, a 5×5 kernel, zero-padding of length 2, and a stride of 1. The second convolutional layer is identical to the first, but has 16 filters instead of 6. The output of the second convolutional block is flattened, passed through a fully-connected layer with 120 units, an ReLU activation, and a final fully-connected layer with $2d$ units, where d represents the dimensionality of the embedding space. Like the Omniglot architectures, the first d entries in the output vector are treated as the mean and the remaining d elements as the diagonal covariance entries. The weights are initialized using a Xavier-uniform initialization and biases are initialized to zero.

PN and SPE are trained episodically with all performance results measured as the mean over 1000 randomly-sampled test episodes. All N-digit MNIST models are trained with an initial

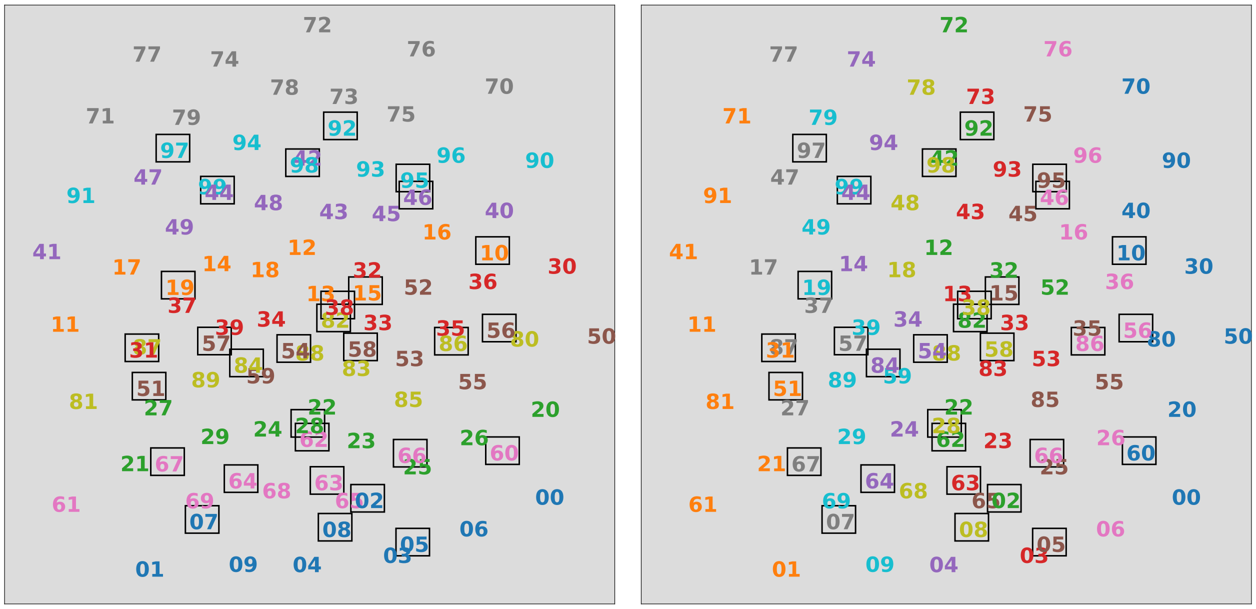


Figure 3.7: 2D embedding learned by the PN on the 2-digit MNIST test set. A class is specified by a two-digit number. In both figures, the location of the class corresponds to the mean of the prototype in the test set using 140 support instances. The digits surrounded by a black border are classes that were not seen during training. In the left and right figures, the prototypes are colored according to the first and second digit of the class, respectively.

learning rate of 0.001 which is cut in half every 50 epochs. The models are stopped early using a patience parameter when performance on the validation set no longer increases. For 2-digit MNIST, each episode in training, validation, and seen-class testing contains all 70 classes and 50 support instances per class. For testing of unseen classes, each episode contains all 30 classes and 50 support instances per class. For 3-digit MNIST, each episode contains 100 classes and either 20 support instances per class for training and validation or 50 support instances per class for seen- and unseen-class testing.

Figure 3.6 shows two views of the 2D embedding learned by the SPE on the 2-digit MNIST test set. Each number is a class label; for example, 71, located in the lower left of the embedding, is the class in which the first of the two MNIST digits is a 7 and the second is a 1. The location of a label in the space corresponds to the mean of its prototype. In the left plot, each class is colored according to the first digit. The right plot is the same embedding, but each prototype is colored according to the second digit. SPE learns an incredibly-robust factorial representation

Table 3.2: Test classification accuracy (%) on 2- and 3-digit MNIST for clean-support & clean-query, corrupt-support & clean-query, and clean-support & corrupt-query. N: number of digits in each image; D: dimensionality of the embedding. Contrastive and HIB results are from Oh et al. [88]. PN is our implementation of Prototypical Networks [112]. SPE is our model. SPE is trained with intersection sampling with 1 sample per trial. Reported accuracy for PN and SPE for each experimental configuration is the mean over 1000 random test episodes.

Clean Support, Clean Query

	Seen test classes					Unseen test classes				
	N=2		N=3			N=2		N=3		
	D=2	D=3	D=2	D=3	Mean	D=2	D=3	D=2	D=3	Mean
Contrastive	88.2	95.0	65.8	87.3	84.1	85.5	84.8	59.0	85.5	78.7
HIB	87.9	**95.2**	65.0	87.3	83.9	87.3	**91.0**	64.4	88.2	82.7
PN	91.1	95.0	65.8	**90.6**	85.6	82.0	89.5	64.3	**89.1**	81.2
SPE	**93.0**	94.2	**80.2**	89.0	**89.1**	**90.0**	89.3	**80.2**	88.2	**86.9**

Corrupt Support, Clean Query

	Seen test classes					Unseen test classes				
	N=2		N=3			N=2		N=3		
	D=2	D=3	D=2	D=3	Mean	D=2	D=3	D=2	D=3	Mean
Contrastive	76.2	92.2	49.5	77.6	73.9	76.5	73.3	42.6	73.2	66.4
HIB	81.6	**94.3**	54.0	81.2	77.8	80.8	**86.7**	53.9	81.2	75.7
PN	72.7	93.3	44.6	82.7	73.3	70.9	86.3	42.9	79.6	69.9
SPE	**92.4**	93.8	**76.7**	**87.8**	**87.7**	**88.8**	86.3	**75.4**	**86.3**	**84.2**

Clean Support, Corrupt Query

	Seen test classes					Unseen test classes				
	N=2		N=3			N=2		N=3		
	D=2	D=3	D=2	D=3	Mean	D=2	D=3	D=2	D=3	Mean
Contrastive	43.5	51.6	29.3	44.7	42.3	46.3	44.8	26.2	42.0	39.8
HIB	49.9	57.8	31.8	49.9	47.4	53.5	57.0	32.1	50.2	48.2
PN	53.1	**61.1**	33.8	**56.4**	**51.1**	51.1	**57.9**	33.0	**54.8**	49.2
SPE	**53.7**	58.2	**40.2**	48.1	50.1	**56.3**	56.5	**39.3**	46.6	**49.7**

in which the horizontal dimension represents the first digit of a class and the vertical dimension represents the second digit. Impressively, the unseen test classes—indicated by a black bounding box—are embedded in exactly the positions where they belong, indicating that the SPE can discover relationships among classes that allow it to generalize to classes it has never seen during training. Furthermore, the embedding has captured inter-class similarity structure by placing visually similar digits close to one another. For example, on both the vertical and horizontal bands, nines (teal) and fours (purple) are adjacent, and fives (brown) and threes (red) are adjacent. HIB discovers a clean decomposition along one dimension [88], but the second dimension is somewhat more entangled, suggesting that the SPE learns a more robust representation. Additionally, embeddings for the unseen class are not presented for HIB. The ability to sensibly embed novel classes is essential for any model that will be used for open-set classification. As we show in Figure 3.7, PN does not obtain a clean compositional structure.

Table 3.2 compares N-digit MNIST test accuracy on seen and unseen classes[†]. Each entry in the table is identified by if the test classes are seen or unseen, the number of horizontally-concatenated digits (N), the dimensionality of the embedding space (D), and the method. The contrastive method corresponds to a deterministic, soft contrastive loss which was used as a baseline to compare with HIB [88]. As in the Omniglot simulation, we varied whether the support and query inputs were clean or corrupted, where the corruption procedure matches Omniglot and is described in Appendix A.2. Looking at the rightmost, grayed column in each sub-table of Table 3.2, which represents the mean test classification accuracy across N and D for a particular setting of seen versus unseen classes and data corruption, SPE outperforms HIB in all six comparisons and PN in five of six. Out of the 24 individual conditions, SPE is worse than HIB on only 7. As in the Omniglot simulation, SPE shines best when support instances may be corrupted.

Whereas SPE is a discriminative model with a specified classification procedure, Oh et al. [88] had the freedom to design one. They use all available data—roughly 140 examples per class—and perform leave-one-out 5-nearest-neighbor classification. To be consistent with our episodic test procedure, SPE uses only 50 support instances per class to form prototypes. It is particularly impressive that SPE, based on a single stored prototype and approximately 1/3 the labeled data, outperforms a nonparametric method that is able to model arbitrary distributions in the embedding space.

[†]Contrastive and HIB results are from Oh et al. [88]. We thank the authors for providing us results on unseen classes, which were not included in their publication.

3.4 Discussion and Conclusions

We proposed the Stochastic Prototype Embedding (SPE) as a method for obtaining supervised, Euclidean, visual embeddings which encode uncertainty in their distributions. Such methods are useful for fixed-set classification, inductive transfer, and few-shot classification, particularly for domains with ambiguous inputs or class noise. We compared SPE to the only fully-developed alternative method at the time of publication, the Hedged Instance Embedding (HIB), on *the complete battery of tasks* used to evaluate HIB. On these large-set classification tasks, SPE consistently outperforms HIB. Beyond its performance gains, SPE has no hand-tuned parameters, whereas HIB has constant, β, that determines characteristics of an information bottleneck (i.e., how much of the input entropy is retained in the embedding).

SPE, which is a stochastic extension of the Prototypical Network (PN), matches or outperforms PN on few-shot classification—the application in which PN was designed, as well as fixed-set classification and inductive transfer learning. Because SPE extends PN, it seems unlikely to fare worse; but because it can handle uncertainty in both the query and support set, it can fare better, particularly when the embedding space is low dimensional and the support instances may be corrupted. Extensions have been proposed to PN that are compatible with ours (e.g., Allen et al. [2]); combining methods may potentially attain even stronger classification performance under uncertainty. The model proposed by Fort [33] shares with SPE the notion that prototypes are a confidence-weighted average of support embeddings (i.e., Equation 3.5), and the embedding procedure produces a scaling matrix for computing a Mahalanobis distance, similar to the Gaussian covariance matrix of SPE. However, Fort [33] does not treat the embedding as stochastic, for to do so, it would need to marginalize over the uncertainty in the embedding to predict a class. This marginalization is the core of a probabilistic model and is the critical component of SPE. Fort [33] also obtains best results with a spherical scaling matrix for the Mahalanobis distance, whereas a critical property of SPE is that the uncertainty varies on each dimension of the latent space; our disentangling and uncertainty results all hinge on using a more flexible diagonal covariance matrix, although we discuss the possibility of using a full covariance in Appendix A.1. And, Fort [33] omits the σ_ϵ noise term which we found to be critical for the model to work in practice.

We proposed and evaluated an intersection sampler to train SPE, which makes SPE as time and space efficient for training as the deterministic PN, and more efficient for training than HIB, which relies on about 8 samples per item. (Our evaluation method for SPE presently involves drawing 200 samples from the naive sampler, though this conservative decision was arbitrary and not tuned.)

An unanticipated virtue of SPE is its ability to obtain interpretable, disentangled representations (e.g., Figures 3.2, 3.5, 3.6). Because uncertainty is encoded in a diagonal covariance matrix, any classification ambiguity maps to uncertainty in the value of individual features of the embedding. Thus, class-discriminating feature dimensions must align with the principle axes of the embedding space. In contrast to traditional unsupervised disentangling methods, which aim to discover the underlying generative factors of a domain, SPE obtains a supervised analog in which the underlying class-discriminative factors are represented explicitly. This representation facilitates generalization to novel unseen classes and is therefore valuable for few-shot and lifelong-learning paradigms.

Chapter 4

von Mises–Fisher Loss: An Exploration of Embedding Geometries for Supervised Learning[*]

Frequently, novel loss functions are proposed that claim superiority over standard losses for supervised representation learning in vision. At a coarse level, these loss functions can be divided into classification based and similarity based. Classification-based losses (e.g., [2, 11, 12, 17, 22, 25, 28, 33, 36, 37, 57, 60, 69, 71, 75, 80, 83, 87, 93, 94, 95, 98, 112, 114, 118, 120, 121, 126, 128, 129, 130, 149]) have generally been applied to fixed-set classification tasks (i.e., tasks in which the the set of classes in training and testing is identical), verification tasks (e.g., face verification), and inductive transfer learning. The prototypical classification-based loss uses a softmax function to map an embedding to a probability distribution over a set of classes, which is then evaluated with cross-entropy [11, 12]. Similarity-based losses (e.g., [13, 21, 35, 42, 50, 53, 73, 88, 97, 103, 115, 116, 119, 122, 123, 132, 133, 134, 136, 140, 147, 151]) have been designed, generally, for open-set classification and retrieval tasks. Open-set tasks refer to situations in which the classes at testing are disjoint from, or sometimes a superset of, those available at training. The prototypical similarity-based method is the triplet loss which discovers embeddings such that an instance is closer to instances of the same class than to instances of different classes [35, 103, 136].

Recent efforts to systematically compare losses support a provocative hypothesis: on open-set tasks, traditional classification-based losses (e.g., softmax cross-entropy over a fixed set of

[*]Scott, T. R., Gallagher, A. C., and Mozer, M. C. (2021). von Mises–Fisher Loss: An Exploration of Embedding Geometries for Supervised Learning. In IEEE/CVF International Conference on Computer Vision. Research conducted as an intern at Google.

classes with learned weight vectors, not embedding-to-embedding classification losses) outperform similarity-based losses by leveraging embeddings in the layer immediately preceding the logits [10, 67, 84, 121, 149]. The apparent advantage of classifiers stems from the fact that similarity losses require sampling informative pairs, triplets, quadruplets, or batches of instances in order to train effectively [10, 28, 83, 128, 129, 130, 149]. However, all classification losses are not equal, and we find systematic differences among them with regard to a fundamental choice: the embedding geometry, which determines the similarity structure of the embedding space.

Classification losses span three embedding geometries: Euclidean, hyperbolic, and spherical. Although some comparisons have been made between geometries, the comparisons have not been entirely systematic and have not covered the variety of supervised applications discussed in Section 2.2. We find this fact somewhat surprising given the many large-scale comparisons of loss functions. Furthermore, the comparisons that have been made appear to be contradictory. Those focused on face verification have led the push for spherical losses, claiming superiority of the spherical geometry over Euclidean. However, this work is limited to face-related tasks [28, 75, 94, 95, 128, 129, 130]. Research on deep-metric-learning has recently refocused attention to classification losses, but it is unclear from empirical comparisons whether the best-performing geometry is Euclidean or spherical [10, 84, 93, 149]. Independently, Khrulkov et al. [57] show that a hyperbolic prototypical network is a strong performer on common few-shot classification benchmarks, and additionally, a hyperbolic softmax classifier outperforms the Euclidean variant on person re-identification. Unfortunately, these results are in contention with Tian et al. [121], where the authors claim a simple Euclidean softmax classifier discovers embeddings that are superior for few-shot classification.

One explanation for the discrepant claims are confounds that make it impossible to determine whether the causal factor for the superiority of one loss over another is embedding geometry or some other ancillary aspect of the loss. Another explanation is that each bit of research examines only a subset of losses or a subset of datasets. Also, as pointed out in Musgrave et al. [84], experimental setups (e.g., using the test set as a validation signal, insufficient hyperparameter tuning, varying forms of data augmentation) make it difficult to trust and reproduce published results. The goal of our research is to take a step toward rigor by reconciling differences among classification losses on both fixed-set classification and open-set retrieval benchmarks.

As discussed in more detail in Section 4.1.3, our investigations led us to uncover an interesting property of spherical losses, which in turn suggested a probabilistic spherical classifier based on the von Mises–Fisher distribution. While our loss is competitive with state-of-the-art alternatives and produces improved out-of-the-box calibration, we avoid unequivocal claims about its superiority. We do, however, believe that it improves on previously proposed stochastic classifiers (e.g., SPE and HIB discussed in Chapter 3 [88, 106]), in, for example, its ability to scale to higher-dimensional embedding spaces. Additionally, SPE and HIB rely on Gaussian distributions which suffer from the curse of dimensionality, the inability to represent a uniform prior over the embedding space, and the "soap-bubble effect" in high-dimensional spaces [24].

Contributions. In this chapter: (1) we characterize classification losses in terms of embedding geometry, (2) we systematically compare classification losses in a well-controlled setting on a range of fixed- and open-set tasks, examining both accuracy and calibration, (3) we reach the surprising conclusion that spherical losses generally outperform the standard softmax cross-entropy loss that is used almost exclusively in practice, (4) we propose a stochastic spherical loss based on von Mises–Fisher distributions, scale it to larger tasks and representational spaces than previous stochastic losses, and show that it can obtain state-of-the-art performance with significantly lower calibration error, and (5) we discuss trade-offs between losses and factors to consider when choosing among them.

4.1 Classification Losses

We consider classification losses that compute the cross-entropy between a predicted class distribution and a one-hot target distribution (or equivalently, as the negative log-likelihood under the model of the target class). The geometry determines the specific mapping from a deep embedding to a class posterior, and in the classification losses we consider, this mapping is determined by a set of parameters learned via gradient descent—not embedding-to-embedding classification losses (e.g., [112, 114]). The following sections summarize the three embedding geometries that serve to differentiate classification losses. The classification losses we focus on are those that have been shown in recent large-scale empirical studies to be the strongest performers [10, 57, 84, 121, 149].

4.1.1 Euclidean

Euclidean embeddings lie in a d-dimensional real-valued space (i.e., $\mathbb{R}^d$ or sometimes $\mathbb{R}^d_+$). The commonly-used dot-product softmax [12], which we refer to as STANDARD, has the form:

$$p(y|\boldsymbol{z}) = \frac{\exp\left(\boldsymbol{w}_y^{\mathrm{T}}\boldsymbol{z}\right)}{\sum_j \exp\left(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{z}\right)}, \tag{4.1}$$

where $\boldsymbol{z}$ is an embedding and $\boldsymbol{w}_j$ are weights for class j. The dot product is a measure of similarity in Euclidean space, and is related to the Euclidean distance by $||\boldsymbol{w}_j - \boldsymbol{z}||^2 = ||\boldsymbol{w}_j||^2 + ||\boldsymbol{z}||^2 - 2\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{z}$. (Classifiers using Euclidean distance have been explored, but gradient-based training methods suffer from the curse of dimensionality because gradients go to zero when all points are far from one another. Prototypical networks [112] and SPE do succeed using a Euclidean distance posterior, but the weights are determined by averaging instance embeddings, not gradient descent.)

4.1.2 Hyperbolic

We follow Ganea et al. [36] and Khrulkov et al. [57] and consider the Poincaré ball model of hyperbolic geometry defined as $\mathbb{D}^d_c = \{\boldsymbol{z} \in \mathbb{R}^d : c\,\|\boldsymbol{z}\|^2 < 1, c \geq 0\}$ where c is a hyperparameter controlling the curvature of the ball. Embeddings thus lie inside a hypersphere of radius $1/\sqrt{c}$. To perform multi-class classification, we employ the hyperbolic softmax generalization derived in [36], hereafter HYPERBOLIC:

$$p(y|\boldsymbol{z}) \propto \exp\left(\frac{\lambda^c_{\boldsymbol{p}_y}\,\|\boldsymbol{a}_y\|}{\sqrt{c}} \sinh^{-1}\left(\frac{2\sqrt{c}\langle -\boldsymbol{p}_y \oplus_c \boldsymbol{z}, \boldsymbol{a}_y\rangle}{\left(1 - c\,\|-\boldsymbol{p}_y \oplus_c \boldsymbol{z}\|^2\right)\|\boldsymbol{a}_y\|}\right)\right), \tag{4.2}$$

where $\boldsymbol{p}_j \in \mathbb{D}^d_c$ and $\boldsymbol{a}_j \in T_{\boldsymbol{p}_j}\mathbb{D}^d_c \setminus \{\boldsymbol{0}\}$[†] are learnable parameters for class j, $\lambda^c_{\boldsymbol{p}_j}$ is the conformal factor of $\boldsymbol{p}_j$, $\langle . \rangle$ is the dot product, and $\oplus_c$ is the Möbius addition operator. Further details can be

[†]$T_{\boldsymbol{z}}\mathbb{D}^d_c$ denotes the tangent space of $\mathbb{D}^d_c$ at $\boldsymbol{z}$.

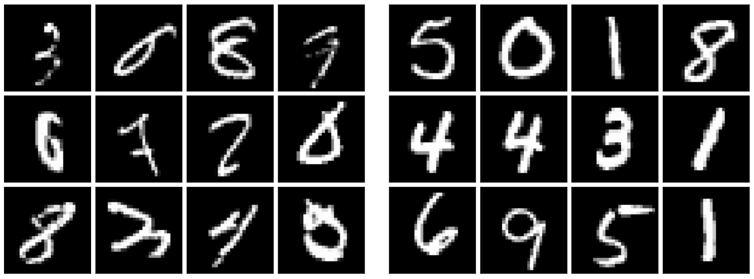


Figure 4.1: MNIST test images corresponding to embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$; trained with COSINE. The left grid clearly contains "noisier" or unorthodox digits.

	MNIST	Fashion MNIST	CIFAR10	CIFAR100
STANDARD	0.92	0.84	0.84	0.66
HYPERBOLIC	0.91	0.81	0.87	0.70
COSINE	0.93	0.84	**0.90**	**0.84**
ARCFACE	0.95	**0.89**	**0.90**	0.80
vMF	**0.97**	0.88	0.82	0.80

Table 4.1: The mean AUROC indicating how well the norm of an embedding, $\|\boldsymbol{z}\|$, discriminates correct and incorrect classifier outputs for five losses (rows) and four datasets (columns). Chance is 0.5; perfect is 1.0. Boldface indicates the highest value. Error bars are negligible across five replications. Although the embedding norm correlates with classifier accuracy for all losses, spherical losses yield the strongest correlation.

found in [36, 57].

4.1.3 Spherical

Spherical embeddings lie on the surface of a d-dimensional unit-hypersphere (i.e., $\mathbb{S}^{d-1}$). The traditional loss, hereafter COSINE, uses cosine similarity [61, 129, 149]:

$$p(y|\boldsymbol{z}) = \frac{\exp(\beta \cos\theta_y)}{\sum_j \exp(\beta \cos\theta_j)}, \tag{4.3}$$

where $\beta > 0$ is an inverse-temperature parameter, $\|\boldsymbol{z}\| = 1$, $\|\boldsymbol{w}_j\| = 1\ \forall\ j$, and θ_j is the angle between $\boldsymbol{z}$ and $\boldsymbol{w}_j$. Note that, in contrast to STANDARD, the ℓ_2-norms are factored out of the weight vectors and embeddings, thus only the *direction* determines class association.

Many variants of COSINE have been proposed, particularly for face verification [28, 75, 94, 95, 128, 129, 130], some of which are claimed to be superior. For completeness, we also experiment with ArcFace [28], one of the top-performing variants, hereafter ARCFACE:

$$p(y|\boldsymbol{z}) = \frac{\exp(\beta\cos(\theta_y + m))}{\exp(\beta\cos(\theta_y + m)) + \sum_{j\neq y}\exp(\beta\cos\theta_j)}, \tag{4.4}$$

where $m > 0$ is an additive-angular-margin hyperparameter penalizing the true class. (Note that we are coloring the loss name by geometry; COSINE and ARCFACE are both spherical losses.)

Early in our investigations, we noticed an interesting property of spherical losses: $\|\boldsymbol{z}\|$ encodes information about uncertainty or ambiguity. For example, the left and right frames of Figure 4.1 show MNIST [70] test images that, when trained with COSINE, produce embeddings that have small and large ℓ_2-norms, respectively. This result is perfectly intuitive for STANDARD since the norm affects the confidence or peakedness of the class posterior distribution—verified in [91, 95], but for COSINE, the norm has *absolutely no effect* on the posterior. Because the norm is factored out by the cosine similarity, there is no force on the model during training to reflect the ambiguity of an instance in the norm. Despite ignoring it, the COSINE model better discriminates correct versus incorrect predictions with the norm than does the STANDARD model (see COSINE and STANDARD rows of Table 4.1; note that the row for vMF corresponds to a loss we introduce in the next section).

Why does the COSINE embedding convey a confidence signal in the norm? One intuition is that when an instance is ambiguous, it could be assigned many different labels in the training set, each pulling the instance's embedding in different directions. If these directions roughly cancel, the embedding will be pulled to the origin.

Due to COSINE having claimed advantages over STANDARD, and also discarding important information conveyed by $\|\boldsymbol{z}\|$, we sought to develop a variant of COSINE that uses the ℓ_2-norm to explicitly represent uncertainty in the embedding space, and thus to inform the classification decision. We refer to this variant as the *von Mises–Fisher loss* or vMF.

4.1.3.1 von Mises–Fisher Loss

The von Mises–Fisher (vMF) distribution is the maximum-entropy distribution on the surface of a hypersphere, parameterized by a mean unit-vector, $\boldsymbol{\mu}$, and isotropic concentration, κ. The pdf for a d-dimensional unit vector $\boldsymbol{x}$ is:

$$p(\boldsymbol{x};\ \boldsymbol{\mu}, \kappa) = C_d(\kappa) \exp(\kappa \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{x}) \ \text{ with } \ C_d(\kappa) = \frac{\kappa^{d/2-1}}{(2\pi)^{d/2} I_{d/2-1}(\kappa)}, \tag{4.5}$$

where $\boldsymbol{x}, \boldsymbol{\mu} \in \mathbb{S}^{d-1}$, $\kappa \geq 0$, and I_v denotes the modified Bessel function of the first kind at order v.

The von Mises–Fisher loss, hereafter vMF, uses the same form of the posterior as COSINE (Equation 4.3), although $\boldsymbol{z}$ and $\{\boldsymbol{w}_j\}$ are now vMF random variables, defined in terms of the deterministic output of the network, $\tilde{\boldsymbol{z}}$, and the learnable weight vector for each class j, $\tilde{\boldsymbol{w}}_j$:

$$\boldsymbol{z} \sim \text{vMF}\left(\boldsymbol{\mu} = \frac{\tilde{\boldsymbol{z}}}{\|\tilde{\boldsymbol{z}}\|}, \kappa = \|\tilde{\boldsymbol{z}}\|\right),\ \boldsymbol{w}_j \sim \text{vMF}\left(\boldsymbol{\mu} = \frac{\tilde{\boldsymbol{w}}_j}{\|\tilde{\boldsymbol{w}}_j\|}, \kappa = \|\tilde{\boldsymbol{w}}_j\|\right). \tag{4.6}$$

The norm $\|\cdot\|$ directly controls the spread of the distribution with a zero norm yielding a uniform distribution over the hypersphere's surface. The loss remains the negative log-likelihood under the target class, but in contrast to COSINE, it is necessary to marginalize over the the embedding and weight-vector uncertainty:

$$\mathcal{L}(y, \boldsymbol{z};\ \boldsymbol{w}_{1:Y}) = \mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}[-\log p(y|\boldsymbol{z}, \boldsymbol{w}_{1:Y})] = \mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}\left[-\log \frac{\exp(\beta \cos\theta_y))}{\sum_j \exp(\beta \cos\theta_j)}\right], \tag{4.7}$$

where Y is the total number of classes in the training set. Applying Jensen's inequality, we obtain an upper-bound on $\mathcal{L}$ which allows us to marginalize over the $\{\boldsymbol{w}_j\}$ and obtain a form expressed in terms of an expectation over $\boldsymbol{z}$:

$$\mathcal{L}(y, \boldsymbol{z};\ \boldsymbol{w}_{1:Y}) \leq \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \frac{C_d(\|\tilde{\boldsymbol{w}}_j\|)}{C_d\left(\|\tilde{\boldsymbol{w}}_j + \beta \boldsymbol{z}\|\right)}\right)\right] - \beta \mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \tag{4.8}$$

where $\mathbb{E}[\boldsymbol{z}] = (I_{d/2}(\kappa)/I_{d/2-1}(\kappa))\boldsymbol{\mu}_{\boldsymbol{z}}$. This objective can be approximated by sampling only from $\boldsymbol{z}$ and we find that during both training and testing, 10 samples is sufficient. At test time, vMF approximates $\mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}[p(y|\boldsymbol{z}, \boldsymbol{w}_{1:Y})]$ using Monte Carlo samples from each of $\boldsymbol{z}$ and $\{\boldsymbol{w}_j\}$. To sample, we make use of a rejection-sampling reparameterization trick [24].

Bounds for $I_{d/2}(\kappa)/I_{d/2-1}(\kappa)$ and $\log C_d(\kappa)$. Davidson et al. [24] computes modified Bessel functions on the CPU with manually-defined gradients for backpropagation, substantially slowing both the forward and backwards passes through the network. Instead, we borrow tight bounds for $I_{d/2}(\kappa)/I_{d/2-1}(\kappa)$ from Ruiz-Antolín and Segura [100] and $\log C_d(\kappa)$ from Kumar and Tsvetkov [66], which together make Equation 4.8 efficient and tractable to compute.

To approximate $I_{d/2}(\kappa)/I_{d/2-1}(\kappa)$ where d is the dimensionality of the embedding space, we borrow a lower bound from Theorem 4 and an upper bound from Theorem 2 of [100]:

$$\frac{\kappa}{\frac{d-1}{2}+\sqrt{\left(\frac{d+1}{2}\right)^2+\kappa^2}} \leq \frac{I_{d/2}(\kappa)}{I_{d/2-1}(\kappa)} \leq \frac{\kappa}{\frac{d-1}{2}+\sqrt{\left(\frac{d-1}{2}\right)^2+\kappa^2}}. \tag{4.9}$$

Let's denote the lower bound as $g_d(\kappa)$ and the upper bound as $h_d(\kappa)$. Our approximation for the ratio of modified Bessel functions is thus:

$$\frac{I_{d/2}(\kappa)}{I_{d/2-1}(\kappa)} \approx \frac{1}{2}(g_d(\kappa)+h_d(\kappa)). \tag{4.10}$$

Next, we borrow a clever trick from Kumar and Tsvetkov [66]. Note that:

$$\frac{\mathrm{d}}{\mathrm{d}\kappa}\log C_d(\kappa) = -\frac{I_{d/2}(\kappa)}{I_{d/2-1}(\kappa)}. \tag{4.11}$$

If we plug in our approximation for the ratio of modified Bessel functions from Equation 4.10 and integrate with respect to κ, we arrive at the following approximation:

$$\begin{aligned}\log C_d(\kappa) \approx{}& \frac{d-1}{4}\log\left(\frac{d-1}{2}+\sqrt{\left(\frac{d-1}{2}\right)^2+\kappa^2}\right)-\frac{1}{2}\sqrt{\left(\frac{d-1}{2}\right)^2+\kappa^2}\\ &+\frac{d-1}{4}\log\left(\frac{d-1}{2}+\sqrt{\left(\frac{d+1}{2}\right)^2+\kappa^2}\right)-\frac{1}{2}\sqrt{\left(\frac{d+1}{2}\right)^2+\kappa^2}+\eta,\end{aligned} \tag{4.12}$$

where η is an unknown constant resulting from indefinite integration. While the approximation is complicated, it is easy to compute and backpropagate through on accelerated hardware. We can rewrite the first term of the final objective using an exponential-log trick that allows us to apply the approximation of $\log C_d(\kappa)$ and cancel η:

$$\begin{aligned}\mathcal{L}(y, \boldsymbol{z};\ \boldsymbol{w}_{1:Y}) &\leq \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \frac{C_d(\|\tilde{\boldsymbol{w}}_j\|)}{C_d\left(\|\tilde{\boldsymbol{w}}_j + \beta\boldsymbol{z}\|\right)}\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \\ &= \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \exp\left(\log C_d(\|\tilde{\boldsymbol{w}}_j\|) - \log C_d\left(\|\tilde{\boldsymbol{w}}_j + \beta\boldsymbol{z}\|\right)\right)\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}].\end{aligned} \tag{4.13}$$

Equation 4.13 is the final form of the vMF objective. A full derivation of the loss is provided in Appendix B.1.

Initialization of the Concentration for vMF. The network fails to train when the initial $\{\tilde{\boldsymbol{w}}_j\}$ are chosen using standard initializers, particularly in higher dimensional embedding spaces. We discovered the failure to be due to near-zero gradients for the ratio of modified Bessel functions when the vector norms are small (e.g., see the flat slope in Figure 4.2 for small κ). We also include a fixed scale-factor on the embedding, $\tilde{\boldsymbol{z}}$, for the same reason.

We seek an initialization of κ for all vMF distributions such that the ratio of modified Bessel functions is a constant greater than zero. Such an initialization would ensure gradients are strong enough for training and all vMF distributions are approximately equally-concentrated. If we take the upper bound, $h_d(\kappa)$, from Equation 4.9 and set it equal to a constant, λ, and solve for κ, we get:

$$\kappa = \frac{\lambda}{1-\lambda^2}(d-1). \tag{4.14}$$

For initializing the concentration of the embedding distribution, $\kappa_{\boldsymbol{z}}$, we introduce a constant scalar, denoted α, that is multiplied with the embedding output of the network, $\tilde{\boldsymbol{z}}$, prior to ℓ_2-normalization. The multiplier does not add any flexibility to the network and also does not affect the direction of the embedding. Since $\kappa_{\boldsymbol{z}} = \|\tilde{\boldsymbol{z}}\|$, we estimate the value of α such that the expected embedding norm over the training dataset is the value we desire from Equation 4.14:

$$\mathbb{E}_{\tilde{\boldsymbol{z}}}\left[\|\alpha\tilde{\boldsymbol{z}}\|\right] = \frac{\lambda}{1-\lambda^2}(d-1) \Rightarrow \mathbb{E}_{\tilde{\boldsymbol{z}}}\left[\sqrt{\sum_i(\alpha\tilde{\boldsymbol{z}}_i)^2}\right] = \frac{\lambda}{1-\lambda^2}(d-1), \tag{4.15}$$

where $\tilde{\boldsymbol{z}}_i$ is the ith element of $\tilde{\boldsymbol{z}}$. Under the strong assumption that $\tilde{\boldsymbol{z}}_1^2 = \tilde{\boldsymbol{z}}_2^2 = \cdots = \tilde{\boldsymbol{z}}_n^2$, we have:

$$\mathbb{E}_{\tilde{\boldsymbol{z}}_i}\left[\sqrt{d(\alpha\tilde{\boldsymbol{z}}_i)^2}\right] = \frac{\lambda}{1-\lambda^2}(d-1) \Rightarrow \alpha = \frac{\lambda(d-1)}{(1-\lambda^2)\sqrt{d}\,\mathbb{E}[|\tilde{\boldsymbol{z}}_i|]}. \tag{4.16}$$

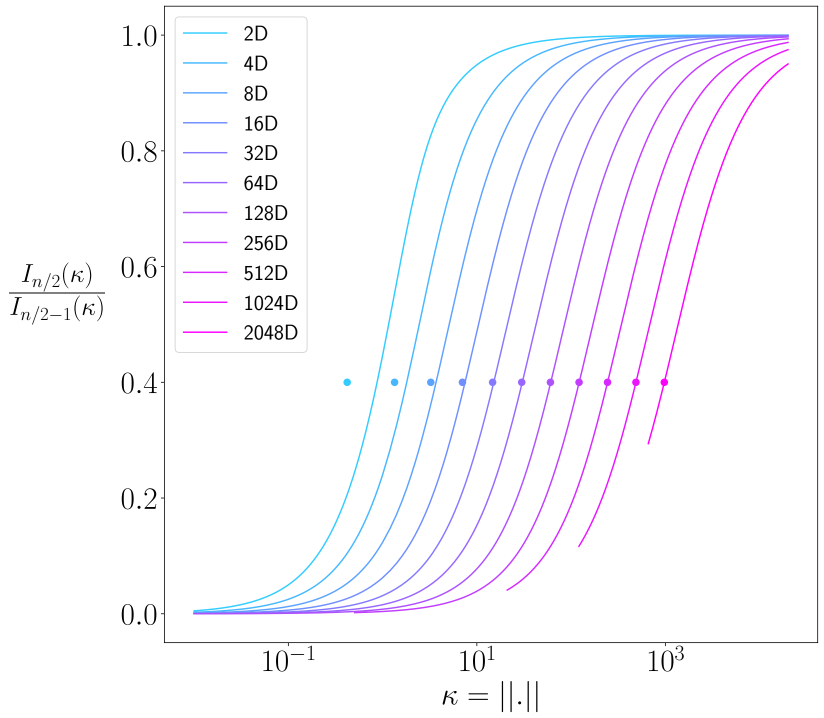


Figure 4.2: The ratio of modified Bessel functions versus κ for various embedding dimensionalities (colored curves), denoted by n. Our initializer for κ ensures the ratio of modified Bessel functions is constant regardless of the dimensionality. The value of κ provided by the initializer for each dimensionality is plotted as a single point. A perfect initializer would ensure the point sits exactly on the matching-colored curve. For this simulation, we initialized such that the y-axis had a constant value of 0.4.

Prior to any training, we compute $\mathbb{E}_{\tilde{\boldsymbol{z}}_i}[|\tilde{\boldsymbol{z}}_i|]$ by passing all of the training data through the network and taking the mean of the resulting tensor across both the embedding-dimension axis and batch axis, prior to ℓ_2-normalization. Then, we are able to determine α, which is fixed for the duration of training.

We also need to initialize the class weight vectors, $\{\tilde{\boldsymbol{w}}_j\}$, such that their expected ℓ_2-norm matches that of $\alpha\tilde{\boldsymbol{z}}$. Let's denote $\tilde{\boldsymbol{w}}_{j,i}$ as the ith element of the weight vector for class j. Assume each element, $\tilde{\boldsymbol{w}}_{j,i}\ \forall\ i = 1 \ldots d$ is independently and identically distributed according to a Gaussian with zero mean and unknown standard deviation. Borrowing Equation 4.16:

$$\xi\mathbb{E}[|\tilde{\boldsymbol{w}}_{j,i}|] = \frac{\lambda(d-1)}{(1-\lambda^2)\sqrt{d}}, \tag{4.17}$$

where ξ replaces the role of α for $\tilde{\boldsymbol{z}}$ above. Note that $\mathbb{E}[|\tilde{\boldsymbol{w}}_{j,i}|] = \sqrt{\frac{2}{\pi}}\sigma$ for a zero-mean Gaussian.

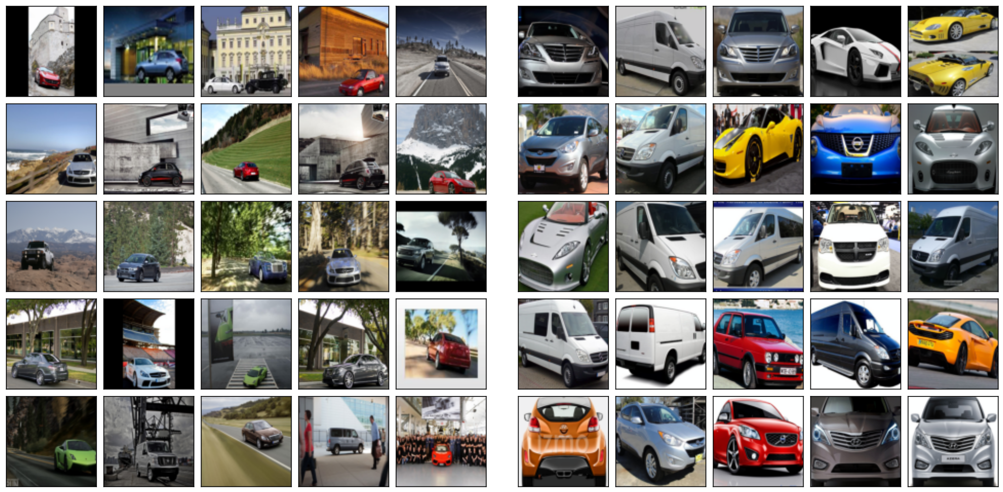


Figure 4.3: Cars196 test images corresponding to vMF embeddings with the (left) smallest $\kappa_{\boldsymbol{z}}$ and (right) largest $\kappa_{\boldsymbol{z}}$. Instances that are more difficult to classify or ambiguous correspond to small $\kappa_{\boldsymbol{z}}$.

Thus:

$$\sigma = \frac{\lambda(d-1)}{(1-\lambda^2)\sqrt{d}}\sqrt{\frac{\pi}{2}}\frac{1}{\xi}. \tag{4.18}$$

We empirically determined that $\xi = \sqrt{\frac{\pi}{2}}$ produced a near-perfect fit for 8D embedding spaces and larger—to be described in the next paragraph—resulting in:

$$\tilde{\boldsymbol{w}}_{j,i} \sim \mathcal{N}\left(\mu = 0, \sigma = \frac{\lambda(d-1)}{(1-\lambda^2)\sqrt{d}}\right)\ \forall\ j = 1 \ldots Y \text{ and } i = 1 \ldots d. \tag{4.19}$$

For training vMF, λ is a hyperparameter. See Appendix B.2 for the values we used.

The points in Figure 4.2 show the scaling of initial $\{\tilde{\boldsymbol{w}}_j\}$ produced by our scheme for various embedding dimensionalities, designed to ensure the ratio of modified Bessel functions has a constant value of 0.4. The expected norms are plotted as individual points with matching color and we find they produce a near-perfect fit for greater than 8D, lying on top of their corresponding curves.

Qualitative Uncertainty. To demonstrate that vMF learns explicit uncertainty structure, we train it on Cars196 [62], a dataset where the class is determined by the make, model, and year of a photographed car. In Figure 4.3, we present images corresponding to embeddings whose distributions have the most uncertainty (i.e., smallest κ_z) and least uncertainty (i.e., largest κ_z) in the test set. vMF behaves quite sensibly: the most uncertain embeddings correspond to images of cars that are far from the camera or at poses where it is difficult to extract the make, model, and year; and the most certain embeddings correspond to images of cars close to the camera in a neutral pose. We include similar figures for all of the losses in Appendix B.3 where uncertainty is determined by the embedding's ℓ_2-norm rather than $\kappa_{\boldsymbol{z}}$.

4.2 Experimental Results

We experiment with four fixed-set classification datasets—MNIST [70], FashionMNIST [141], CIFAR10 [63], and CIFAR100 [63]—as well as three common datasets for open-set image retrieval—Cars196 [62], CUB200-2011 [127], and Stanford Online Products (SOP) [116]. MNIST and FashionMNIST are trained with 3D embeddings, CIFAR10 and CIFAR100 with 128D embeddings, and all open-set datasets with 512D embeddings. We perform a hyperparameter search for all losses on each dataset. The hyperparameters associated with the best performance on the validation set are then used to train five replications of the method. Reported test performance represents the average over the five replications. Details on the network architectures and datasets are presented in the next section, and definitions of the hyperparameters and their chosen values across experiments are provided in Appendix B.2.

4.2.1 Methodological Details

The following sections enumerate details on the network architectures, as well as the preprocessing and data augmentation for each dataset.

4.2.1.1 Architectures

MNIST & FashionMNIST. Experiments on MNIST and FashionMNIST use an architecture adapted from Oh et al. [88]. The architecture has two convolutional blocks each with a convolutional layer followed by batch normalization, ReLU, and 2×2 max-pooling. The first convolutional layer has a 5×5 kernel, 6 filters, zero-padding of length 2, and a stride of 1. The second convolutional layer is identical to the first, but with 16 filters. After the second convolutional block, the representation is flattened and passed through a fully-connected layer with 120 units, followed by batch normalization and a ReLU. The representation is finally passed through two fully-connected layers, the first from 120 units to d units, where d is the dimensionality of the embedding space, and then from d units to Y units, where Y is the total number of classes in the training set. The last fully-connected layer has no bias parameters. All biases in the network are initialized to zero and all weights are initialized with Xavier uniform. For vMF, instead of using Xavier uniform, the weights in the final fully-connected classification layer parameterize vMF distributions for each class and thus are initialized using the scheme detailed in Section 4.1.3.1. For both MNIST and FashionMNIST, $d = 3$ and $Y = 10$.

CIFARs & Open-Set. Experiments on CIFAR10, CIFAR100, Cars196, CUB200-2011, and SOP use a ResNet50 [46]. For CIFAR10 and CIFAR100, the first convolutional layer uses a 3×3 kernel instead of 7×7. For Cars196, CUB200-2011, and SOP, the network is initialized using weights pretrained on ImageNet and all batch-normalization parameters are frozen. We remove the head of the architecture and add two fully-connected layers directly following the global-average-pooling operation. The first fully-connected layer maps from 2048 units to d units, and the second fully-connected layer maps from d units to Y units. The last fully-connected layer has no bias parameters and for vMF, is initialized using the scheme detailed in Section 4.1.3.1. For CIFAR10 and CIFAR100, $d = 128$, and $Y = 10$ and $Y = 100$, respectively. For Cars196, CUB200-2011, and SOP, $d = 512$, and $Y = 83$, $Y = 85$, and $Y = 9620$, respectively.

4.2.1.2 Datasets

Below we detail data preprocessing, data augmentation, and batch sampling for each of the datasets. We follow the preprocessing and augmentation procedures of Boudiaf et al. [10] for Cars196, CUB200-2011, and SOP. For batch sampling during training, we use an episodic scheme where we first sample n classes at random and then sample k instances from each of the n classes.

MNIST & FashionMNIST. For MNIST and FashionMNIST, pixels are linearly scaled to be in $[0, 1]$. No data augmentation is used. The validation set is created by splitting off 15% of the train data, stratified by class label. For batch sampling, $n = 10$ and $k = 13$.

CIFAR10 & CIFAR100. During training, images are padded with 4 pixels on all sides via reflection padding, after which a 32×32 random crop is taken. Then, the image is randomly flipped horizontally and z-score normalized. Finally, each image is occluded with a randomly-located 8×8 patch where occluded pixels are set to zero. During validation and testing, images are only z-score normalized. The validation set is created by splitting off 15% of the train data, stratified by class label. For batch sampling, CIFAR10 has $n = 10$ and $k = 26$ and CIFAR100 has $n = 32$ and $k = 8$.

Cars196. During training, images are resized to 256×256 and brightness, contrast, saturation, and hue are jittered randomly with factors in $[0.7, 1.3]$, $[0.7, 1.3]$, $[0.7, 1.3]$, and $[-0.1, 0.1]$, respectively. A crop of random size in $[0.16, 1.0]$ of the original size and random location is taken and resized to 224×224. Finally, the image is randomly flipped horizontally and z-score normalized. During validation and testing, images are resized to 256×256, center-cropped to size 224×224, and z-score normalized. The train set contains half of the classes and the test contains the other half. The validation set is created by splitting off 15% of the train classes. For batch sampling, $n = 32$ and $k = 4$.

CUB200-2011. During training, images are resized such that the smaller edge has size 256 while maintaining the aspect ratio. Brightness, contrast, and saturation are jittered randomly, all with factors in $[0.75, 1.25]$. A crop of random size in $[0.16, 1.0]$ of the original size and random location is taken with the aspect ratio selected randomly in $[0.75, 1.33]$ and resized to 224×224. Finally, the image is randomly flipped horizontally and z-score normalized. During validation and testing, images are resized such that the smaller edge has size 256, center-cropped to size 224×224, and z-score normalized. The train set contains half of the classes and the test contains the other half. The validation set is created by splitting off 15% of the train classes. For batch sampling, $n = 32$ and $k = 4$.

SOP. During training, images are resized to 256×256, a crop of random size in $[0.16, 1.0]$ of the original size and random location is taken with aspect ratio selected randomly in $[0.75, 1.33]$, and resized to 224×224. Finally, the image is randomly flipped horizontally and z-score normalized.

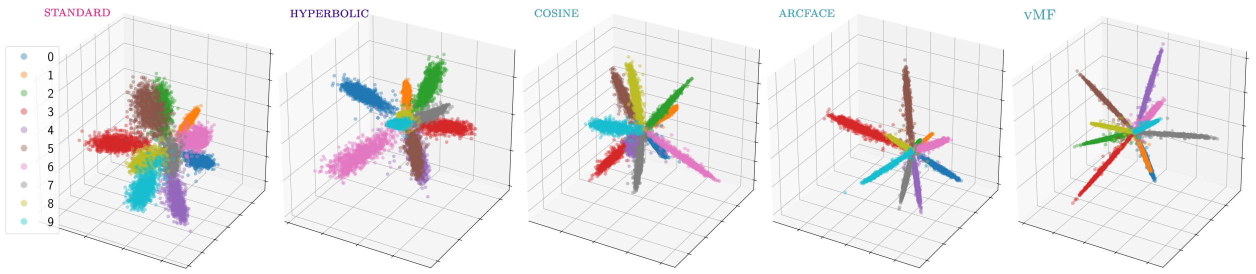


Figure 4.4: 3D embeddings of the MNIST test set for each of the five classification variants. Embedded instances are colored by their ground-truth class. The plotted embeddings for vMF correspond to $\boldsymbol{\mu_z}$. Note that HYPERBOLIC, COSINE, ARCFACE, and vMF are showing embeddings prior to the normalization/projection step. Best viewed in color.

	MNIST	Fashion MNIST	CIFAR10	CIFAR100
STANDARD	98.92 ± 0.03	90.31 ± 0.12	$\mathbf{94.13 \pm 0.05}$	69.21 ± 0.18
HYPERBOLIC	98.92 ± 0.03	90.31 ± 0.13	$\mathbf{94.11 \pm 0.05}$	69.85 ± 0.07
COSINE	98.99 ± 0.03	90.39 ± 0.09	93.99 ± 0.10	$\mathbf{70.57 \pm 0.54}$
ARCFACE	$\mathbf{99.13 \pm 0.02}$	$\mathbf{90.73 \pm 0.12}$	$\mathbf{94.15 \pm 0.05}$	69.08 ± 0.57
vMF	99.02 ± 0.04	$\mathbf{90.82 \pm 0.14}$	$\mathbf{94.00 \pm 0.12}$	$\mathbf{69.94 \pm 0.18}$

Table 4.2: Mean classification accuracy (%) of each loss across four fixed-set classification tasks. Error bars represent ± 1 standard-error of the mean. Boldface indicates the best-performing loss(es). Note that on average, the three spherical losses outperform HYPERBOLIC and STANDARD.

During validation and testing, images are resized to 256×256, center-cropped to size 224×224, and z-score normalized. The train set contains half of the classes and the test contains the other half. The validation set is created by splitting off 15% of the train classes. For batch sampling, $n = 32$ and $k = 2$.

4.2.2 Fixed-Set Classification

We begin by comparing representations learned by the five losses we described: STANDARD, HYPERBOLIC, COSINE, ARCFACE, and vMF. The latter three have spherical geometries. ARCFACE is a minor variant of COSINE claimed to be a top performer for face verification [28]. vMF is our probabilistic extension of COSINE. Using a 3D embedding on MNIST, we observe decreasing intra-class angular variance for the losses appearing from left to right in Figure 4.4. The intra-class

	ECE			
	MNIST	Fashion MNIST	CIFAR10	CIFAR100
STANDARD	2.4 ± 0.2	12.4 ± 0.8	8.8 ± 0.1	20.6 ± 0.2
HYPERBOLIC	2.8 ± 0.1	13.2 ± 0.4	8.9 ± 0.1	22.0 ± 0.2
COSINE	$\mathbf{1.8 \pm 0.2}$	7.9 ± 0.5	8.9 ± 0.2	21.8 ± 1.2
ARCFACE	2.3 ± 0.1	11.0 ± 0.5	10.0 ± 0.1	26.4 ± 0.4
vMF	$\mathbf{1.6 \pm 0.1}$	$\mathbf{4.2 \pm 0.5}$	$\mathbf{5.9 \pm 0.2}$	$\mathbf{7.9 \pm 0.3}$

	ECE after Temperature Scaling			
	MNIST	Fashion MNIST	CIFAR10	CIFAR100
STANDARD	$\mathbf{0.4 \pm 0.1}$	$\mathbf{5.5 \pm 1.3}$	$\mathbf{2.7 \pm 0.2}$	$\mathbf{2.2 \pm 0.1}$
HYPERBOLIC	1.4 ± 0.1	7.2 ± 0.7	$\mathbf{2.8 \pm 0.1}$	2.6 ± 0.1
COSINE	1.6 ± 0.1	$\mathbf{4.2 \pm 0.1}$	6.4 ± 0.2	10.8 ± 0.8
ARCFACE	1.7 ± 0.1	7.2 ± 0.4	8.7 ± 0.1	15.7 ± 0.2
vMF	1.5 ± 0.1	5.0 ± 0.2	5.3 ± 0.2	8.0 ± 0.2

Table 4.3: Mean expected calibration error (%), computed with 15 equal-mass bins, before post-hoc calibration (top table) and after temperature scaling (bottom table) across the four fixed-set classification tasks. Error bars represent ± 1 standard-error of the mean. Boldface indicates the loss(es) with the lowest error.

variance is related to inter-class discriminability, as the 10 classes are similarly dispersed for all losses. The three losses with spherical geometry obtain the lowest variance, with ARCFACE lower than COSINE due to a margin hyperparameter designed to penalize intra-class variance; and vMF achieves the same, if not lower variance still, as a natural consequence of uncertainty reduction.

The test accuracy for each of the fixed-set classification datasets and the five losses is presented in Table 4.2. Across all datasets, spherical losses outperform STANDARD and HYPERBOLIC. Among the spherical losses, ARCFACE and COSINE are deficient on at least one dataset, whereas vMF is a consistently-strong performer.

Table 4.3 presents the top-label expected calibration error (ECE) for each dataset and loss. The top-label expected calibration error approximates the disparity between a model's confidence output, $\max_y p(y|\boldsymbol{z})$, and the ground-truth likelihood of being correct [65, 99]. The top table

	Cars196	CUB200-2011	SOP
STANDARD	21.3 ± 0.2	20.0 ± 0.2	39.7 ± 0.1
+ COSINE AT TEST	23.2 ± 0.1	21.4 ± 0.2	42.1 ± 0.1
HYPERBOLIC	22.9 ± 0.3	20.1 ± 0.3	41.0 ± 0.2
+ COSINE AT TEST	25.0 ± 0.4	21.8 ± 0.2	44.0 ± 0.1
COSINE	24.6 ± 0.4	$\mathbf{22.8 \pm 0.1}$	$\mathbf{44.3 \pm 0.1}$
ARCFACE	$\mathbf{27.4 \pm 0.2}$	$\mathbf{23.1 \pm 0.3}$	40.8 ± 0.3
vMF	$\mathbf{27.2 \pm 0.1}$	22.1 ± 0.1	38.3 ± 0.2

Table 4.4: Mean mAP@R (%) across the three open-set image retrieval tasks. Error bars represent ±1 standard-error of the mean. Boldface indicates the best-performing loss(es). "+ Cosine at Test" replaces the default metric for the geometry (i.e., Euclidean for STANDARD and Poincaré for HYPERBOLIC) with cosine distance to compare embeddings.

shows out-of-the-box ECE on the test set (i.e., prior to any post-hoc calibration). vMF has significantly reduced ECE compared to other losses, with relative error reductions of 40-70% for FashionMNIST, CIFAR10, and CIFAR100. The bottom table shows ECE after applying post-hoc temperature scaling [39]. STANDARD and HYPERBOLIC greatly benefit from temperature scaling, with STANDARD exhibiting the lowest calibration error.

Post-hoc calibration requires a validation set, but many settings cannot afford the data budget to reserve a sizeable validation set, which makes out-of-the-box calibration a desirable property. For example, in open-set classification and inductive transfer learning, one may not have enough data in the target domain to both fine-tune the classifier and validate.

Temperature scaling is not as effective when applied to spherical losses as when applied to STANDARD and HYPERBOLIC. The explanation, we hypothesize, is that the spherical losses incorporate a learned temperature parameter β—discussed below, which is unraveled by the calibration temperature. We leave it as an open question for how to properly post-hoc calibrate spherical losses.

4.2.3 Open-Set Retrieval

For open-set retrieval, we follow the data preprocessing pipeline of [10], where each dataset is first split into a train set and test set with disjoint classes. We additionally split off 15% of the training classes for a validation set, a decision that has been left out of many training procedures of similarity-based losses [84]. We evaluate methods using mean-average-precision at R (mAP@R), a metric shown to be more informative than Recall@1 [84]. For the stochastic loss, vMF, we compute $\mathbb{E}_{\boldsymbol{z}_{1:N}}[\text{mAP@R}(\boldsymbol{z}_{1:N}, y_{1:N})]$, where N is the number of test instances.

Table 4.4 presents the retrieval performance for each loss. As with fixed-set classification, there is no consistent winner across all datasets, but spherical losses tend to outperform STANDARD and HYPERBOLIC. Boudiaf et al. [10] find that retrieval performance can be improved for STANDARD by employing cosine distance at test time. Although no principled explanation is provided, we note from Figure 4.4 that $\|\boldsymbol{z}\|$ introduces a large source of intra-class variance in STANDARD and HYPERBOLIC. This variance is factored out automatically by the spherical losses via cosine similarity. As shown in Table 4.4, cosine distance at test improves both STANDARD and HYPERBOLIC, though not to the level of the best-performing spherical loss.

In contrast to other stochastic losses [88, 106], including SPE from Chapter 3, vMF scales to high-dimensional embeddings—512D in this case—and can be competitive with state-of-the-art. However, it has the worst performance on SOP which has 9620 training classes; many more than all other datasets. In Section 4.1.3.1, we mentioned that to marginalize out the weight distributions, we successively apply Jensen's inequality to their expectations. The decision resulted in a tractable loss, but it is an upper-bound on the true loss and the bound likely becomes loose as the number of training classes increases. We hypothesize the inferior performance is due to this design choice, and despite experimentation with curriculum-learning techniques to condition on a subset of classes during training, results did not improve.

4.2.4 Role of Temperature

Due to spherical losses using cosine similarity, the logits are bounded in $[-1, 1]$. Consequently, it is necessary to scale the logits to a more suitable range. In past research, spherical losses have incorporated an inverse-temperature constant, $\beta > 0$ [28, 128, 130, 149]. Past efforts to turn β into a trainable parameter find that it either does not work as well as fixing it, or no comparison

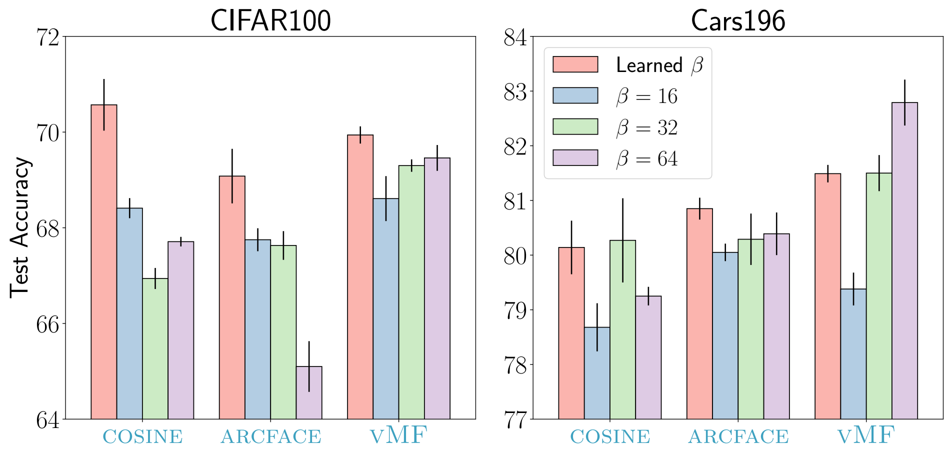


Figure 4.5: Comparison between a learned temperature and various values of a fixed temperature on (left) CIFAR100 and (right) Cars196. Learning the temperature performs at least as well as fixing it, with exception to vMF on Cars196.

is made to a fixed value [94, 95, 129].

In contrast to past research, we parameterize the inverse-temperature, β, as $\beta = \exp(\tau)$ where $\tau \in \mathbb{R}$ is an unconstrained network parameter learned via gradient descent. For fixed-set classification tasks, the network is initialized randomly. We find that fixing β to a large value adversely affects the training dynamics, as small improvements in angular separation can lead to drastic reductions in the loss caused by the peakedness of the posterior (e.g., see the reduction in performance for a fixed value of β in the left frame of Figure 4.5). A challenge in learning τ directly is the network can cheat by maximizing it within the first several epochs. Our parameterization prevents "early cheating" since smaller values of τ result in smaller gradients. For all fixed-set tasks, we initialize $\tau = 0$. Additionally, we use a smaller learning rate for τ, so the network is further incentivized to focus on the angular discrimination between classes. The backbone for the open-set tasks is a ResNet50 pretrained on ImageNet [27]. Because the network reuses previously learned features, we found initializing τ to a larger value sometimes results in improved performance. See the hyperparameter tables in Appendix B.2 for Cars196, CUB200-2011, and SOP for the initial values of τ.

To study our modifications to the parameterization of β, Figure 4.5 contains an ablation study comparing a fixed temperature—using constant values common in past research—to a learned temperature for both CIFAR100 (fixed-set classification) and Cars196 (open-set retrieval). We find that our parameterization for a trained β performs at least as well as a fixed value and avoids the manual search.

4.2.5 Runtime of the von Mises–Fisher Loss

One concern for the vMF loss is that it requires rejection sampling to both train and test, however, in practice, the sampling has marginal impact on runtime. We measured the runtime of vMF on both Cars196 and CIFAR100 and compared it to COSINE. For Cars196, training is 75 seconds per epoch for both vMF and COSINE, and 32 seconds versus 28 seconds for computing test-set embeddings, respectively. For CIFAR100, vMF is slightly slower to train than COSINE, 35 seconds per epoch versus 28 seconds per epoch, respectively, but computing test-set predictions is 3 seconds for both. vMF is no more sensitive to hyperparameters, but it does require slightly more epochs to converge: for Cars196 and CIFAR100, vMF trained for 1.8x and 1.1x as many epochs compared to COSINE, respectively.

4.3 Related Work

At the beginning of the chapter, we dichotomized the literature in a coarse manner based on whether losses are similarity based or classification based. In this section, we focus on recent work relevant to the vMF, including losses functions using vMF distributions and stochastic classifiers.

A popular use of vMF distributions in machine learning has been clustering [7, 38]. Banerjee et al. [7] consider a vMF mixture model trained with expectation-maximization, and Gopal and Yang [38] propose a fully-Bayesian extension along with hierarchical and temporal versions trained with variational inference. In addition, vMF distributions have begun being applied in supervised settings. Hasnat et al. [44] suggest a supervised classifier for face verification where each term in the softmax is interpreted as the probability density of a vMF distribution. Zhe et al. [152] propose a classification-based loss that is functionally identical to Hasnat et al. [44], except the mean parameters of the vMF distributions for each class are estimated as the maximum-likelihood estimate of the training data. Park et al. [92] describe the spherical analog to the prototypical network [112], but use a generator network to output the prototypes. The downside of these supervised losses compared to vMF is they assume the concentration parameter across all classes is identical and fixed, canceling $C_d(\kappa)$ in the softmax. Such a decision is mathematically convenient, but removes a significant amount of flexibility in the model. Davidson et al. [24] propose a variational autoencoder (VAE) with hyperspherical latent structure by using a vMF distribution. They find it can outperform a Gaussian VAE, but only in lower-dimensional latent spaces. We leverage the rejection-sampling reparameterization scheme they compose to train vMF.

Other work has sought losses that consider either stochastic embeddings or stochastic logits, but they suggest Gaussians instead of vMFs. Chang et al. [17] suppose stochastic embeddings, but deterministic classification weights, and train a classifier using Monte Carlo samples. They also add a KL-divergence regularizer between the embedding distribution and a zero-mean, unit-variance Gaussian. Collier et al. [22] propose a classification loss with stochastic logits that uses a temperature-parameterized softmax. They show it can train under the influence of heteroscedastic label noise with improved accuracy and calibration. Shi and Jain [109] convert deterministic face embeddings into Gaussians by training a post-hoc network to estimate the covariances. Their objective maximizes the mutual likelihood of same-class embeddings. Scott et al. [106] and Oh et al. [88] propose SPE and HIB, which are the Gaussian-based analogs of the prototypical network [112] and pairwise contrastive loss [42], respectively. These losses were the focus in Chapter 3. Neither loss shows promise with high-dimensional embeddings. SPE struggles to compete with a standard prototypical network in 64D, and HIB [88] omits any results with embeddings larger than 3D. Gaussian distributions suffer from the curse of dimensionality, the soap-bubble effect, and aren't capable of expressing uniformity over the embedding space [24]. In contrast, von Mises–Fisher distributions revert to a uniform distribution over the hypersphere when the concentration parameter is zero. These reasons likely explain the inferior performance of Gaussian-based methods compared to deterministic alternatives, particularly in higher dimensions.

The work most closely related to ours is Kornblith et al. [61], which compares STANDARD, COSINE, and an assortment of alternatives including mean-squared error, sigmoid cross-entropy, and various regularizers applied to STANDARD. In contrast, we focus on multiple variants of spherical losses and comparing geometries. Their findings are compatible with ours.

We note that we are not the first to consider the ℓ_2-norm of an embedding vector as a signal for uncertainty. For example, Sabour et al. [101] use the ℓ_2-norm of the output vector of a capsule to represent the probability of an entity or object existing in the input.

4.4 Conclusions

In this chapter, we perform a systematic comparison of classification losses that span three embedding geometries—Euclidean, hyperbolic, and spherical—and attempt to reconcile the discrepancies in past research regarding their performance. Our investigations have led to a stochastic spherical classifier where embeddings and class weight vectors are von Mises–Fisher random variables. Our proposed loss is on par with other classification variants and also produces consistently-reduced out-of-the-box calibration error. Our loss encodes instance ambiguity using the concentration parameter of the vMF distribution.

Consistent with the no-free-lunch theorem [139], we find there is no one loss to rule them all. The performance jump claimed for novel losses often vanishes with rigorous, systematic comparisons in controlled settings—in settings where the network architecture is identical, hyperparameters are optimized with equal vigor, regularization and data augmentation is matched, and a held-out validation set is used to choose hyperparameters. Musgrave et al. [84] reach a similar conclusion: the gap between various similarity-based losses, as well as the spherical classification losses—COSINE and ARCFACE—was much smaller in magnitude than was claimed in the original papers proposing them. We are hopeful that future work on supervised loss functions will also prioritize rigorous experiments and step away from the compulsion to show unqualified improvements over state of the art.

Pedantics aside, we are able to glean some positive messages by systematically comparing performance across geometries and across different classification paradigms. We focus on specific recommendations in the remainder of the paper that address trade-offs among the embedding geometries.

Accuracy. Many losses are designed primarily with accuracy in mind. Across both fixed- and open-set tasks, we find that losses operating in a spherical geometry discriminate, and thus perform, best. Our results support the following ranking of losses: STANDARD $\leq$ HYPERBOLIC $\leq$ $\{$COSINE, ARCFACE, vMF$\}$. Additionally, our results corroborate two conclusions from Chen et al. [19]. First, smaller intra-class angular variance yields better generalization, and second, STANDARD focuses too much on separating classes by increasing embedding norms rather than reducing angular variance (e.g., Figure 4.4). Although the best of the spherical losses appears to be dataset dependent, the guidance to focus on spherical losses and perform empirical comparisons is not business as usual for practitioners, who treat STANDARD as the go-to loss. For practitioners using models with non-spherical geometries (i.e., STANDARD and HYPERBOLIC), we can still provide the guidance to use cosine distance at test time—discarding the embedding magnitude—which seems to reliably lead to improved retrieval performance.

An aspect of accuracy we do not consider, however, is the performance of downstream target tasks where a classification loss is used to pretrain weights (i.e., inductive transfer learning). Kornblith et al. [61] discover that better class separation on the pretraining task can lead to worse transfer performance, as improved discriminability implies task specialization (i.e., throwing away inter-class variance necessary to relate classes). Spherical losses perform well on fixed-set tasks as well as on open-set tasks where the novel test classes are drawn from a distribution very similar to that of the training distribution, but transfer learning is typically a setting where STANDARD has been shown to be superior. We believe that it would be useful in future research to distinguish between near- and far-transfer, as doing so may yield distinct conclusions.

Calibration. In sensitive domains, deployed machine learning systems must produce trustworthy predictions, which requires that model confidence scores match model accuracy (i.e., they must be well calibrated). To our knowledge, we are the first to rigorously examine the effect of embedding geometry on calibration performance. Our findings indicate that when a validation set is available for temperature scaling, STANDARD consistently produces predictions with the lowest calibration error, but STANDARD generally underperforms in accuracy. Additionally, there does not exist a significant gap in out-of-the-box calibration performance across previously proposed losses from the three geometries. However, our novel vMF loss achieves superior calibration while maintaining state-of-the-art classification performance.

Future Work. There are several interesting directions for future work. First, De Cao and Aziz [26] introduce a novel probability distribution—the Power Spherical distribution—with support on the surface of the hypersphere. They claim it has several key improvements over vMF distributions including a reparameterization trick that does not require rejection sampling, improved stability in high dimensions and for large values of the concentration parameter, and no dependence on Bessel functions. A stochastic classifier based on the Power Spherical distribution would likely improve the computational efficiency as well as the optimization procedure, particularly for high-dimensional embedding spaces. Second, we note that our formulation of vMF is a special case of a class of objectives based on the deep variational information bottleneck [1]: $I(Z, Y) - \gamma I(Z, X)$, where $\gamma = 0$ and the classification weights are also stochastic. Our objective thus lacks a regularizer attempting to compress the amount of input information contained in the embedding. Adding this regularization term may lead to improved performance and robustness. Third, all of the experimental datasets are approximately balanced and without label noise. Due to the stochasticity of the classification weights, vMF seems likely to benefit in supervised settings with long-tailed distributions or heteroscedastic label noise [22].

It is unfortunate that the field of supervised visual representation learning has become so vast that research tends to be specialized for a particular application (e.g., fixed-set classification, open-set classification, inductive transfer learning, retrieval) or domain (e.g., face verification, person re-identification, image classification). As a result, losses can be pigeonholed to one application or domain. The objective of this work is to lay out the space of losses in terms of embedding geometry and systematically survey losses that are not typically compared to one another. One surprising and important result in this survey is the strength of spherical losses, and the resulting dissociation of embedding norm and output confidence.

Chapter 5

Unifying Few- and Zero-Shot Egocentric Action Recognition[*]

The egocentric action recognition task consists of observing short first-person video segments of an action being performed, and predicting the label—typically a verb–noun pair—that a human would assign (e.g., "pick-up plate" or "mix pasta"). Many supervised methods (e.g., [16, 153]) treat the problem as a fixed-set classification task, where the set of action classes is identical during training and testing. Fixed-set classification is useful for benchmarking methods, but is often unrealistic in practical settings due to the compositionality of actions, resulting in a functionally infinite-cardinality label set. We, instead, treat egocentric action recognition as an open-set classification task. Previous chapters focused on classification-based losses for discovering visual representations of static imagery. In this chapter, we employ state-of-the-art spherical similarity-based losses—commonly employed by practitioners—for discovering representations of egocentric video[†].

We consider two popular applications of open-set classification: few-shot classification (FSC) and zero-shot classification (ZSC). In the former, we use the model to classify query inputs from classes unseen during training using a small support set of labeled samples. In the latter, we use the model to map video clips to a latent representation that captures the semantic structure of the

[*]Scott, T. R., Shvartsman, M., and Ridgeway, K. (2020). Unifying Few- and Zero-Shot Egocentric Action Recognition. In CVPR Workshop on Egocentric Perception, Interaction, and Computing. Research conducted as an intern at Meta.

[†]The revitalization of standard classification-based losses discussed in Chapter 4 happened after this research was conducted.

label space, and recognize inputs from new classes by matching them to prototypes that are known a priori.

While the two have been proposed as separate tasks, we recognize that ZSC can be framed as another instance of FSC, in which the support set contains a semantic representation of the class labels. We use this insight to generalize ZSC to a task we term *cross-modal few-shot classification* (CM-FSC). CM-FSC includes ZSC, as well as other task variants such as ones where the cross-modal information is not derived from natural language, or multiple instances of the semantic representation are available for a class.

In this research, we identify four main contributions: first, we formally unify FSC and CM-FSC into a framework that promotes inter-method comparison and provides the ability to compare open-set tasks. Second, we present three new data splits from the original EPIC-KITCHENS [23] training set—each with its own train, validation, and test subset—specifically designed to evaluate open-set generalization. Third, we detail several candidate spherical loss functions to train neural networks to jointly perform the two tasks. Fourth, we conduct a head-to-head comparison of FSC and CM-FSC on identical data splits and show that among the spherical losses explored, the ones that do best in one task also do best in the other (i.e., there is no performance trade-off). In addition, our results emphasize the importance of similarity-based losses not only for FSC, but CM-FSC, where we observe improvements upwards of 10% over the conventional baseline. We hope our research bridges advancements in open-set classification with egocentric action recognition, and that our results serve as a first benchmark.

5.1 Open-Set Classification Tasks

Below we formalize the two open-set classification tasks with respect to action recognition. Let $\boldsymbol{x}^v \in \mathbb{R}^{F\times C\times H\times W}$ denote an input video clip consisting of F C-channel frames with height, H, and width, W, and let x^ℓ denote an action label (e.g., "take fork"). Both FSC and ZSC are evaluated *episodically*, where each episode contains a random sample of n action classes, denoted $\mathcal{Y}_{\text{test}}$, which are disjoint from the set of training action classes, $\mathcal{Y}_{\text{train}}$ (i.e., $\mathcal{Y}_{\text{test}} \cap \mathcal{Y}_{\text{train}} = \emptyset$). We make the distinction between the action class (e.g., “class 345”) and the semantic action label (e.g., “take fork”) explicit here, as this is what allows us to unify FSC and ZSC in a common framing below.

5.1.1 Few-Shot Classification

In FSC, the goal is to generalize to classes in $\mathcal{Y}_{\text{test}}$ using only a few video inputs from each. In each episode, $k+q$ instances are sampled from each class in $\mathcal{Y}_{\text{test}}$. The first k instances (or “shots” from “few-shot”) make up the support set, $\mathcal{S}$, and the remaining q make up the query set, $\mathcal{Q}$:

$$\begin{aligned} \mathcal{S} &= \{(\boldsymbol{x}^v_{ij}, y_{ij}) | y_{ij} \in \mathcal{Y}_{\text{test}}\}_{i=1:n,\, j=1:k}, \\ \mathcal{Q} &= \{\boldsymbol{x}^v_{ij}\}_{i=1:n,\, j=k+1:k+q}, \end{aligned} \tag{5.1}$$

where $\boldsymbol{x}^v_{ij}$ is the jth video instance of the ith class in the episode. Evaluation proceeds by classifying each element in the query set using the support set. In EPIC-KITCHENS, there are a number of classes with very few instances. Therefore, during evaluation, we sample up to q query instances per class. Since every episode will have a different number of queries, we report accuracy over all episodes.

5.1.2 Cross-Modal Few-Shot Classification

To see how FSC is related to ZSC, recall above the distinction we made between the set of action labels and the set of action classes. The action labels are ignored in standard FSC, since each support tuple consists of a video and class. If one were to replace the video, $\boldsymbol{x}^v$, with the natural language description of the action class (e.g., “take cup,” denoted by $\boldsymbol{x}^\ell$), one would be in a *cross-modal few-shot classification* (CM-FSC) setting, where the support set contains the action labels associated with each of the n classes in $\mathcal{Y}_{\text{test}}$, and the query set remains unchanged [‡]:

$$\begin{aligned} \mathcal{S} &= \{(\boldsymbol{x}^\ell_{ij}, y_{ij}) | y_{ij} \in \mathcal{Y}_{\text{test}}\}_{i=1:n, j=1:k}, \\ \mathcal{Q} &= \{\boldsymbol{x}^v_{ij}\}_{i=1:n,\, j=k+1:k+q}. \end{aligned} \tag{5.2}$$

[‡]In some cases, the class may be defined by a class-attribute vector, as opposed to a semantic class label [113].

When $k = 1$, CM-FSC reduces to ZSC. When $k > 1$, we obtain a novel task. This novel task is not possible in the conventional ZSC setting as the classes can be fully described with a single instance (i.e., the action label or class-attribute vector), but can be possible with noisy labels or potentially richer narrations from which the action label is generated.

5.2 Related Work

We now highlight several common approaches for FSC and ZSC. Many FSC methods learn an embedding of the visual inputs—typically images or video clips—where embedded inputs that are farther apart are less likely to belong to the same class. These methods can be dichotomized as classification-based (e.g., [112]) or similarity-based, where the similarity-based methods make use of pairwise (e.g., [42]), triplet (e.g., [133]), quadruplet (e.g., [123]), or higher-order (e.g., [122]) constraints directly on the embeddings. All of the above methods are designed to promote intra-class similarity and inter-class dissimilarity. FSC methods also make use of *meta-learning* (e.g., [32, 82]): a learning algorithm focused on quick adaptation of the model to unseen classes. Memory-augmented neural networks [102] have also been explored in FSC because they can use external memory mechanisms to store and recall data from unseen classes. Recently, the above few-shot approaches have begun being applied in the domain of action recognition [9, 14, 15, 82, 155].

For ZSC, there are three common approaches: (1) learn a function that maps visual inputs directly to a class-attribute vector, where new classes constitute novel compositions of attributes [90], (2) map inputs into a pretrained semantic space learned with Word2Vec [81] or BERT [29], for example, where new classes can be directly interpreted [43, 113], and (3) learn two functions that map visual inputs and attribute/semantic vectors, respectively, to a joint latent space [9, 82, 112]. In approaches (1) and (2), the desired representation is typically fixed—either predefined class-attribute vectors or predefined word embeddings. This discourages the model from representing features that are unique to the visual domain, in our case, the visual and temporal features from video that may not correspond to semantic features of the labels (e.g., that bananas tend to be yellow). These approaches are similarly applicable to CM-FSC. We explore state-of-the-art

Split	Train	Validation			Test		
		HoV	HoN	All	HoV	HoN	All
1	1715	158	102	262	248	249	536
2	1732	135	97	239	257	247	542
3	1731	130	104	239	280	238	543

Table 5.1: Counts of classes in each split, broken down by set (i.e., train, validation, test) and by type (i.e., held-out verb (HoV) and held-out noun (HoN)).

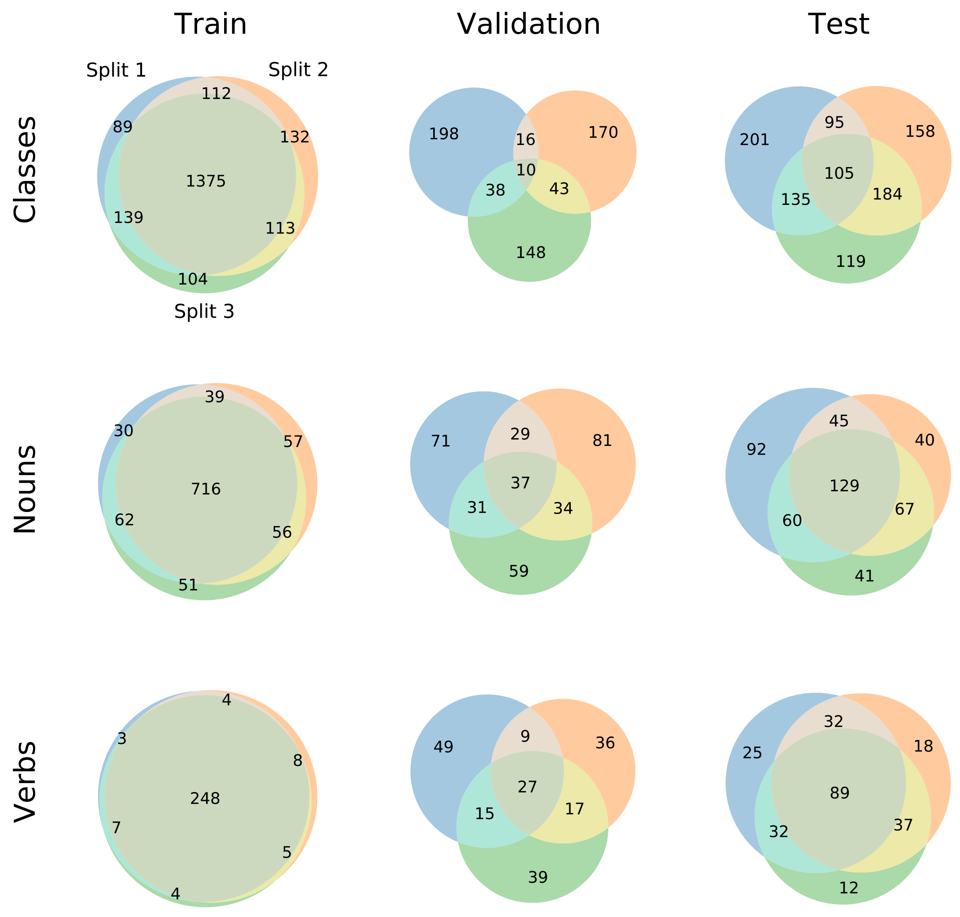


Figure 5.1: Venn diagrams showing overlap in splits for the train, validation, and test sets, broken down by class, noun, and verb. Each colored circle corresponds to one of the three splits. The overlapping regions between circles are annotated with the number of classes, nouns, or verbs corresponding to that region. The training sets are largely similar with a majority of classes, nouns and verbs identical between splits 1, 2, and 3. The validation and test splits show greater heterogeneity, providing support for the success of our splitting procedure.

similarity-based losses from FSC in conjunction with approaches (2) and (3) from ZSC further in Section 5.4.

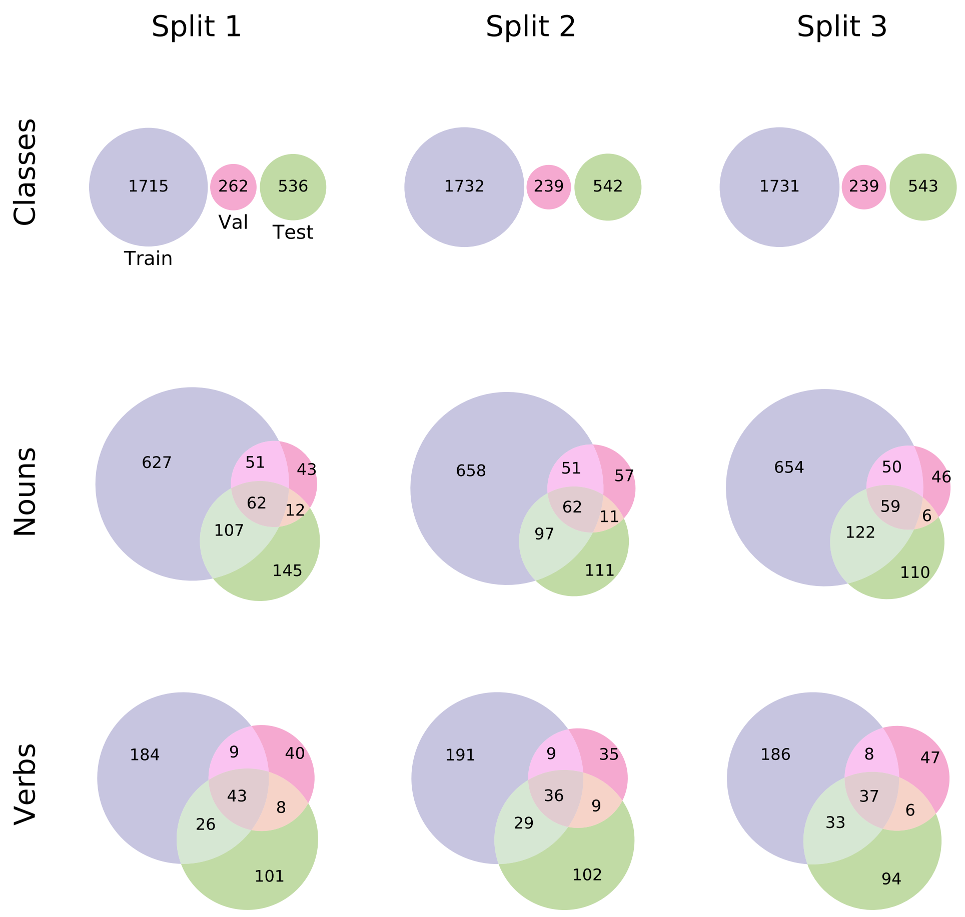


Figure 5.2: Venn diagrams showing overlap in train, validation, and test sets for each split, broken down by class, noun, and verb. As expected in the open-set setting, the classes are all distinct between training, validation and test. At the same time, the validation and test splits include some nouns and verbs not seen at all during training (i.e. the HoN and HoV subsets), and some others that were seen as part of a different class context.

5.3 Dataset Construction

Using the EPIC-KITCHENS [23] training set, we constructed three new open-set splits, each with its own train, validation, and test set, where the classes are defined by the verb- and primary-noun-class as in the EPIC-KITCHENS challenge. Within a split, classes are disjoint across train, validation, and test, as standard for the open-set classification setting. We further sub-divided the test classes into (1) those with a held-out verb (HoV), but trained noun, (2) those with a held-out noun (HoN), but trained verb, and (3) those with a held-out verb and noun pair, where both the verb was unseen during training and the noun was unseen during training. Class counts for each split are given in Table 5.1. Since few classes fall into (3), we report performance on HoV, HoN, and the entire test set, denoted “All Test.”

To generate the novel splits, we first cross-tabulated the verb and noun classes in the original EPIC-KITCHENS training set, so that we could consider the dataset at the class level rather than the instance level. Next, we constructed a set of verbs and nouns eligible to be included in the validation and test sets by excluding verbs that appeared in fewer than v_l contexts (i.e., with fewer than that many nouns) or those that appeared in more than v_u contexts. We did the same for nouns with cutoffs n_l and n_u. We did this to ensure there were sufficiently varied noun and verb contexts in the training set, and to ensure there were no singleton and near-singleton classes in the validation or test sets. Next, among the remaining classes, we uniformly sampled p_v verbs and p_n nouns to be excluded from the training set, and further subsampled those into the test set with proportions p_v^t and p_n^t. The remaining verbs and nouns not subsampled into the test set were included in the validation set. We selected all of the parameters $(v_l, v_y, n_l, n_u, p_v, p_n, p_v^t, p_n^t)$ by trial-and-error so that the number of classes in each held-out subset (i.e., HoN validation, HoN test, HoV validation, HoV test) were roughly comparable. We next performed the same procedure for different random seeds, and retained splits where the counts were mostly balanced. Figure 5.1 shows the number of overlapping classes, nouns, and verbs in each of our three novel splits. Note that while the training sets are fairly similar—owing to our class-eligibility cutoff above, there is substantial variability in the classes, nouns, and verbs included in the validation and test sets between our splits, providing support for the success of our splitting procedure. The aforementioned variability is likely contributing to the variability we observe in results across splits—to be described in Section 5.5. Figure 5.2 shows the counts of overlapping classes, nouns, and verbs between the training, validation and test sets for each split. Consistent with the open-set setting, there are no overlapping classes between the sets, but there are some overlapping nouns and verbs, allowing us to evaluate performance for held-out nouns and held-out verbs separately from overlapping classes.

5.4 Methods

Generalizing FSC and CM-FSC into a common framework lets us seek a method that is capable of performing successfully in both tasks. We begin by introducing a video embedding (i.e., a unimodal FSC-only method) and then extend it to methods that perform both tasks. All methods are assumed to be spherical and thus operate on ℓ_2-normalized embeddings.

Video Embedding (VE). VE learns a deep embedding using video inputs, similar to Careaga et al. [15]. This is an FSC-only baseline because it does not align videos to a second modality (e.g., class-attribute vectors or semantic word embeddings). Training VE proceeds by first sampling a set $\mathcal{Y}_{\text{batch}} \subset \mathcal{Y}_{\text{train}}$ of n training classes. Then, a batch is formed by embedding k inputs of each class with a neural network, $f_{\boldsymbol{\phi}}$:

$$\mathcal{B}_v = \{(f_{\boldsymbol{\phi}}(\boldsymbol{x}^v_{ij}), y_{ij}) | y_{ij} \in \mathcal{Y}_{\text{batch}}\}_{i=1:n,\, j=1:k}. \tag{5.3}$$

We estimate $\boldsymbol{\phi}$ via backpropagation to minimize a similarity-based loss denoted by $\mathcal{L}_{\text{SIM}}(\mathcal{B}_v)$.

Word Embedding (WE). To extend VE for CM-FSC, WE maps $\boldsymbol{x}^v$ directly to a word-embedding space of the class labels, denoted $b(\boldsymbol{x}^\ell)$. $\mathcal{L}_{\text{WE}}$ combines $\mathcal{L}_{\text{SIM}}$ with an alignment term between video and word embeddings:

$$\begin{aligned} \mathcal{B}_{v,\ell} &= \{(f_{\boldsymbol{\phi}}(\boldsymbol{x}^v_{ij}), b(\boldsymbol{x}^\ell_{ij})\}_{i=1:n,\, j=1:k}, \\ \mathcal{L}_{\text{WE}} &= \lambda \mathcal{L}_{\text{SIM}}(\mathcal{B}_v) + \mathbb{E}_{\mathcal{B}_{v,l}} \left[|| f_{\boldsymbol{\phi}}(\boldsymbol{x}^v) - b(\boldsymbol{x}^\ell) ||_2^2 \right], \end{aligned} \tag{5.4}$$

where $\mathcal{B}_v$ is defined as for VE (i.e., Equation 5.3). When $\lambda = 0$, Equation 5.4 is equivalent to the loss from Socher et al. [113]. When $\lambda > 0$, Equation 5.4 is similar to Hahn et al. [43], except (1) we use $\mathcal{L}_{\text{SIM}}(\mathcal{B}_v)$ instead of a linear layer trained with softmax cross-entropy and (2) we use mean-squared-error (MSE) between $f_{\boldsymbol{\phi}}(\boldsymbol{x}^v)$ and $b(\boldsymbol{x}^\ell)$ instead of a contrastive loss.

Joint Embedding (JE). The downside of WE is that MSE imposes direct alignment between $f_{\boldsymbol{\phi}}(\boldsymbol{x}^v)$ and $b(\boldsymbol{x}^\ell)$ (i.e., the network is encouraged to throw away visual features that are not represented in $b(\boldsymbol{x}^\ell)$). Instead, JE maps both videos and word embeddings to a shared, joint embedding space. To do this, we train an independent neural network, $g_{\boldsymbol{\nu}}$, that maps the word

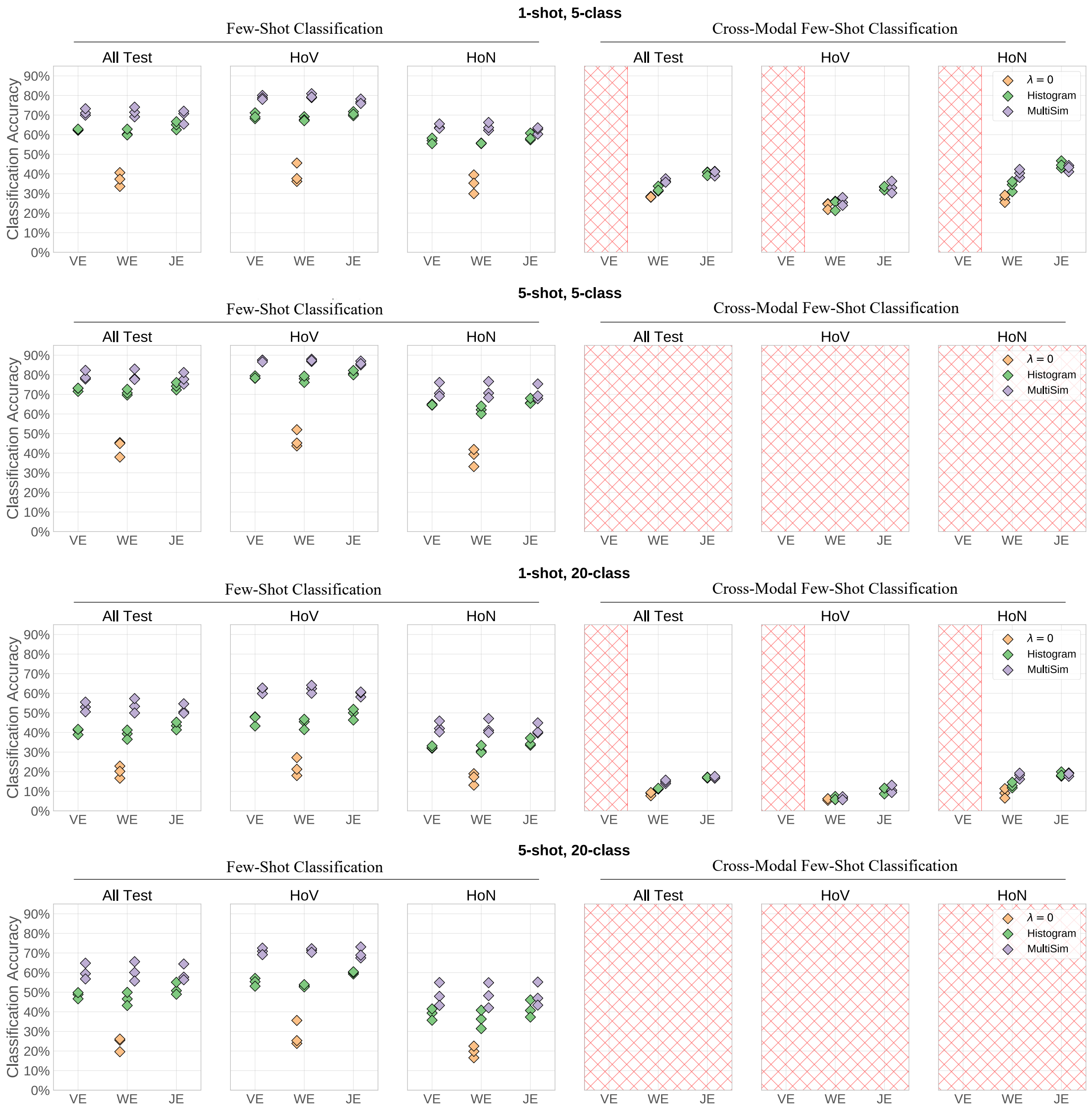


Figure 5.3: Classification accuracy, computed over 500 test episodes, for the video embedding (VE), word embedding (WE), and joint embedding (JE). Each row in the plot corresponds to a setting of k ("shot") and n ("class"). $q = 20$ in all cases. Each pane is characterized according to a generalization task (FSC or CM-FSC) and a subset of the test set (All Test, HoV, or HoN). For a given generalization task, test subset, and method, the same-colored points represent performance on each of the three data splits. The red hatching indicates that the given method(s) could not be used to compute accuracy. For all settings, VE doesn't support CM-FSC. Furthermore, CM-FSC is only valid when $k = 1$, since we use class-labels as the support modality.

embeddings of the labels into a latent space of the same dimensionality as $f_{\boldsymbol{\phi}}(\boldsymbol{x}^v)$. The embeddings are thus modality agnostic, which lets us apply a cross-modal, similarity-based loss to a shared batch defined as the union of the video and twice-embedded label batches:

$$\begin{aligned} \mathcal{B}_g &= \{(g_{\boldsymbol{\nu}}\big(b(\boldsymbol{x}^{\ell}_{ij})\big), y_{ij}) | y_{ij} \in \mathcal{Y}_{\text{batch}}\}_{i=1:n,\, j=1}, \\ \mathcal{L}_{\text{JE}} &= \mathcal{L}_{\text{SIM}}(\mathcal{B}_v \cup \mathcal{B}_g), \end{aligned} \tag{5.5}$$

where $\mathcal{B}_v$ is defined as for VE (i.e., Equation 5.3).

5.4.1 Training Details

For all methods described above, $f_{\boldsymbol{\phi}}$ is an I3D network initialized using inflated ImageNet features [16], followed by an LSTM which collapses remaining timesteps into a single latent visual-embedding. The LSTM network has a single layer with d hidden units, where d is the dimensionality of the embedding space, and is initialized using samples from a standard Gaussian distribution. The latent word-embedding network, b, is a frozen, pretrained BERT model [138][§], and $g_{\boldsymbol{\nu}}$ is a single fully-connected layer. For $\mathcal{L}_{\text{SIM}}$ we experiment with both histogram loss [123] and multi-similarity (multi-sim) loss [133], and all embeddings are ℓ_2-normalized. The backbones are shared between the FSC and CM-FSC models in all of our experiments.

Both VE and JE are trained using a 256-dimensional embedding, while WE operates in the same space as the BERT embeddings, which has 768 dimensions. We consider two variations of WE, one with $\lambda = 0$ (i.e., no similarity-based loss), which we denote $\text{WE}_{\lambda=0}$ and a second $\text{WE}_{\lambda=10}$, where $\lambda = 10$ was chosen based on validation performance. To compute accuracy, we use a κ-nearest neighbor classifier over the embeddings, where $\kappa = k$.

For training and validation, we randomly sample two-second video clips within the labeled beginning and end frames, at 24 FPS, where each frame is resized to 256×256. During training, we augment the data by randomly applying a horizontal flip to all of the frames in each video clip. For testing, we use the same procedure, but the clips are sampled to be the central 48 frames without

[§]The BERT model from Wolf et al. [138] produces un-normalized embeddings, so we post-hoc ℓ_2-normalized them for the WE method. We confirmed that the relative similarity structure among classes remains majorly intact.

mirroring. For video clips less than two seconds, we add zero padding. The LSTM only processes non-padded frames.

To construct the batches used for training and validation, we sample $n = 12$ classes and up to $k = 8$ inputs per class. We ensure the batch contains at least 36 total instances or it is resampled. For $\mathcal{B}_{v,\ell}$, we need a parallel set of word- and video-embeddings. Since there is only one word-embedding per class, we create k copies of it when constructing the batch.

In all experiments, models are fit with the Adam optimizer [58]. The initial learning rate is set to 1×10^{-5}, and is multiplied by 0.8 every 15000 training batches. Every 500 training steps, the validation loss is averaged over 250 batches. The model with the lowest validation loss is used for final evaluation. Models are trained for a maximum of 75000 batches, but could stop early based on a patience hyperparameter that checks if the validation loss has decreased in the previous 15000 batches.

5.5 Experiments and Conclusions

Figure 5.3 shows classification accuracy across the three methods—VE, WE, and JE—for each split, test subset, and classification task where $k \in \{1, 5\}$ and $n \in \{5, 20\}$. Tables containing all of the results are provided in Appendix C.1. Our unified framing of FSC and ZSC, as CM-FSC, allows us to compare performance of methods on both tasks for the first time. First, we observe that among CM-FSC-capable methods, the ones incorporate a similarity-based loss (i.e., $\text{WE}_{\lambda=10}$ and JE) reliably outperform $\text{WE}_{\lambda=0}$, a method designed for cross-modal prediction, on CM-FSC. Second, JE leads to strictly superior CM-FSC and equivalent FSC when compared to VE and WE, indicating that among methods explored, there appears to be minimal trade-off between FSC and CM-FSC performance. Third, we note that while there is some variability in performance across splits, it is smaller than variability across methods. This provides some evidence that our results are reliable, and that the splitting procedure generates useful, novel evaluation splits of the EPIC-KITCHENS dataset. Incidentally, we find that the multi-sim loss systematically outperforms histogram loss, matching results from Ustinova and Lempitsky [123] and Wang et al. [133].

Our results, although preliminary, provide a strong baseline for comparison on a novel set of open-set classification splits derived from the fixed-set EPIC-KITCHENS dataset. Future work could exploit the broader framework defined here to further explore the space of evaluation paradigms for open-set classification. For example, the textual descriptions provided for action segments in EPIC-KITCHENS contain information beyond the verb and primary noun. These longer descriptions could serve as a more informative input for cross-modal inference, and would enable evaluation of CM-FSC with $k > 1$. Also, future work could explore mixed-modal FSC, where the support sets contain a mixture of video and language samples.

Chapter 6

An Empirical Study on Clustering Pretrained Embeddings: Is Deep Strictly Better?[*]

Clustering—the process by which a set of items are semantically-partitioned into finite groups—has been employed in many domains of computing including machine learning, computer graphics, information retrieval, and bioinformatics. It has numerous applications including interpretability, compression, visualization, outlier detector, and zero-shot classification.

While clustering methods have traditionally ingested raw features (e.g., pixels) as input, an alternative explored particularly in the face-verification literature is to use deep embeddings [40, 72, 86, 89, 108, 110, 135, 142, 143, 144, 150]. They form a natural space for clustering because they are learned with loss functions that encourage grouping of semantically-similar inputs (e.g., [10, 21, 28, 42, 112, 114, 136]) and explicit representation of task-relevant features (e.g., [96, 103]).

Face verification systems rely on large backbone networks that produce highly-discriminative embeddings of face images. Improvements to these systems come largely through increasing the available supervised data, but with an expensive annotation cost. To overcome the annotation challenge, recent work has successfully used clustering to pseudo-supervise [47, 86, 108, 135, 142, 143, 150]. The approach is cyclical: train a face-embedding model, embed and cluster unsupervised face images, pseudo-supervised the images via the clustering assignments, merge the pseudo-supervised data into the training set, and re-train the model. Since the crux of the approach is the clus-

[*]Scott, T. R., Liu, T., Mozer, M. C. and Gallagher, A. C. (2022). An Empirical Study on Clustering Pretrained Embeddings: Is Deep Strictly Better? Submitted for publication. Research conducted as an intern at Google.

tering step, as improved clustering leads to better pseudo-supervision and thus improved embedding quality, research has shifted from unsupervised, shallow, heuristic-based clustering methods [7, 20, 30, 34, 38, 49, 51, 64, 72, 74, 78, 85, 89, 107, 110, 111, 117] to supervised, deep, inductive methods [40, 47, 86, 108, 135, 142, 143, 144, 150], with claimed improvements upward of 20% over k-means [78] and hierarchical agglomerative clustering [111], for example.

Even though improvements from the deep methods are impressive, the prior work has limitations. First, by focusing primarily on face datasets, the clustering methods operate on embedding spaces that are highly discriminative and not representative of the many embedding spaces that could benefit from clustering. Deng et al. [28] show that both verification and Recall@1 [84] performance of the embeddings are well-above 90% and can even approach the ceiling across a range of face datasets, including MegaFace [55]. We also confirm that the embeddings for DeepFashion [77]—one of the only non-face datasets explored for clustering pretrained embeddings—exhibit Recall@1 above 90%. Second, the commonly-used face datasets (e.g., MS-Celeb1M [41] and MegaFace [55]) are no longer publicly available, with progress reliant on using unofficial, shared embeddings extracted from pretrained backbones. Third, similar to Musgrave et al. [84], we find methodological choices that may implicitly favor the recent, deep methods; these choices include the lack of a validation split, monitoring test performance for hyperparameter tuning and early stopping, and copying baseline results from previous work.

For the reasons above, we conducted a large-scale empirical study of methods for clustering pretrained embeddings. We focused on outlining an end-to-end, reproducible pipeline, including dataset curation, backbone training, and clustering. We then benchmarked 17 clustering methods—including two state-of-the-art, supervised, deep methods, GCN-VE [142] and STAR-FC [108]—on three datasets. The first two datasets are Cars196 [62] and Stanford Online Products (SOP) [116]. We specifically chose to benchmark against Cars196 and SOP because they are: (1) popular in the embedding-learning literature [10, 84, 104], (2) extend beyond the face-verification domain, and (3) produce embeddings that are significantly less discriminative, as discussed in Section 6.3.1. We investigate a third dataset that has similar statistics, similar Recall@1 performance, and thus similar clustering results compared to common face datasets (e.g., MS-Celeb1M) used solely in past research. The third dataset's purpose is simply to corroborate past results.

Our results indicate that the deep methods are indeed superior when operating on highly-discriminative embeddings, but their performance drops otherwise, matching or even underperforming the shallow, heuristic-based methods. We conclude that the Recall@1 performance of the embeddings is generally an accurate proxy for predicting the benefits of deep clustering methods. Additionally, where the deep methods do improve performance, the improvements are of much smaller magnitude than previously reported, likely as a result of our systematic training pipeline and matched hyperparameter tuning. Lastly, we find that ℓ_2-normalization of embeddings prior to clustering leads to a stark improvement across all heuristic-based methods, exploiting the geometrical properties of embeddings learned with softmax cross-entropy [104].

6.1 Related Work

6.1.1 Unsupervised Clustering

Clustering is normally characterized as an unsupervised learning task where methods leverage heuristics based on assumptions about the data-generation process or the structure of the data. Adopting the terminology of Jain [51], we discuss four common classes of heuristics: partition-based, density-based, hierarchical, and graph-theoretic.

Methods using partition-based heuristics [34, 49, 64, 78, 107, 117] have a representation of clusters and decide which data points belong to each cluster, in many cases, using a similarity or distance function. The most popular method is k-means clustering [78] which estimates empirical cluster centers minimizing within-cluster variance. Other variants make assumptions about the geometry of the data, e.g., spherical k-means [49], make assumptions about the scale of the data, e.g., mini-batch k-means [107], or explore alternative formulations that remove the need for a predefined number of clusters, e.g., Dirichlet-process (DP) k-means [64, 117].

Density-based methods assume that clusters represent high-density regions in the feature space and are separated from other clusters by low-density regions. Popular density-based methods attempt to estimate cluster densities via expectation-maximization, e.g., the Gaussian mixture model (GMM) & the von Mises–Fisher mixture model (vMF-MM) [7], variational inference [38], Parzen windows, e.g., DBSCAN [30] & deep density clustering [72], and kernel density estimation, e.g., MeanShift [20].

Hierarchical methods form clusters by starting either with a singleton cluster containing all data points and iteratively splitting it (i.e., top-down or divisive) or with a cluster for each data point and iteratively merging them (i.e., bottom-up or agglomerative). The split or merge is decided greedily using a linkage rule until a criteria is satisfied, such as the total number of clusters. We experiment with hierarchical agglomerative clustering (HAC) [111] and several popular linkage rules—single, complete, average, and Ward linkage—as well as approximate rank-order (ARO) [89] based on the rank-order linkage metric [154].

The final common class of heuristics use graph theoretics. These methods represent data points as a graph with edges weighted by pairwise similarities. One of the most popular methods, and the one we employ in our experiments, is spectral clustering [85]. Spectral clustering performs k-means clustering on the eigenvectors of the normalized Laplacian matrix constructed from the graph's affinity matrix.

6.1.2 Supervised Clustering

Recent research, instead of relying on heuristics, has proposed deep, supervised methods that inductively cluster. These methods rely on supervision indicating which data points belong to the same cluster. By *inductive*, we refer to methods that learn a function, for example, a function that predicts if two data points should be clustered together, which can then be applied to unseen data. This is in contrast to the unsupervised methods which lack generalizability from one dataset to another. Generally, the idea is to leverage local and/or global structure in the feature space to learn what points belong to the same cluster or what pairs of points should be linked. Clusters can then be formed using simple algorithms such as connected-component labeling.

Consensus-driven propagation (CDP) [150] uses an ensemble of backbone networks to produce statistics across many k-nearest-neighbor affinity graphs and then trains a *mediator* network to predict links between data points from which clusters are formed. Wang et al. [135] improves link prediction by capturing structure in local affinity graphs directly with graph convolutional networks (GCNs) [59]. A series of GCN-methods followed that incorporate both local and global structure via density information [40] and multi-scale views [143], for example. Alternatives beyond GCNs have been proposed such as using self-attention via transformers [86, 144] and learning an agglomerative clustering policy with inverse reinforcement learning [47].

Our experiments include results from CDP, as well as two state-of-the-art GCN approaches: GCN-VE [142] and STAR-FC [108]. GCN-VE is a supervised approach that trains two GCNs. The first, GCN-V, estimates a confidence for each data point based on a supervised density measure. The second, GCN-E, constructs subgraphs based on the estimated confidences and predicts pairwise links. The links are used in a *tree-deduction* algorithm to construct final clusters. STAR-FC trains a single GCN to predict pairwise links, but uses a structure-preserving sampling strategy to train the GCN with both local and global graph structure. During inference, links are predicted followed by *graph parsing and refinement* steps to construct clusters.

Consistent with past work, we do not train an ensemble and mediator for CDP, but rather use just the unsupervised *label-propagation* step to form clusters. Note that this removes all supervised components of CDP, which is why we classify it as an unsupervised method in Section 6.3.

6.2 Methodology

Our pipeline for clustering pretrained embeddings involves three steps: dataset curation, backbone training, and clustering. We discuss the methodology for each of the steps below.

6.2.1 Dataset Curation

We consider three datasets for our experiments: Cars196 [62], Stanford Online Products (SOP) [116], and a private, third dataset, *Dataset 3*. Each dataset is partitioned into a *backbone*

	Cars196			SOP		
	m	n	$\frac{m}{n}$	m	n	$\frac{m}{n}$
Backbone Train	$3,963$	49	81	$32,277$	$5,658$	6
Clustering Train	$4,091$	49	83	$27,265$	$5,658$	5
Validation	$4,058$	49	83	$32,734$	$5,658$	6
Test	$4,073$	49	83	$27,777$	$5,660$	5

Table 6.1: Dataset splits with number of instances, m, number of classes, n, and approximate number of instances per class, $\frac{m}{n}$, for Cars196 and SOP. The classes in each split are disjoint.

train split, used to train the backbone, a *clustering train* split, used only to train the deep clustering methods, a *validation* split, used for hyperparameter tuning and early stopping for the backbone and deep clustering methods, and one or more *test* splits, used to fit the unsupervised clustering methods and evaluate all clustering methods.

Cars196 and SOP have a single test split while Dataset 3 has five test splits that cumulatively grow in size, enabling investigation into the scalability of clustering methods. All splits, except for the test splits of Dataset 3, have disjoint sets of classes. Table 6.1 has per-split statistics on the number of instances, number of classes, and approximate number of instances per class for both Cars196 and SOP. Dataset 3 has $\mathcal{O}(100,000)$ instances, $\mathcal{O}(1,000)$ classes, and roughly $\mathcal{O}(100)$ instances per class for all splits.

We acknowledge that the deep methods are provided an advantage through access to the train-clustering and validation splits, which are unavailable to the shallow methods. We are unaware of systematic approaches to include these data splits across all shallow methods, and instead opted to exclude them, matching the protocol of past research on clustering face embeddings.

6.2.2 Backbone Training

We use a ResNet50 [46] for the backbone with an additional fully-connected layer added after global average pooling and prior to the classification head, which produces a 256D embedding. We chose a ResNet50 backbone to match prior research on face clustering [86, 108, 142, 143, 144], but note that consistent clustering results were found with an Inception v3 backbone [47]. The model is trained with cosine softmax cross-entropy—based on findings from Chapter 4—using the backbone-train split of each dataset, with Recall@1 monitored on the validation split for early stopping and hyperparameter tuning. A hyperparameter search was performed for each of the datasets. Additional details on training the backbone, including specifics on the architecture, data augmentation, and hyperparameters are included in Appendix D.5.1.

6.2.3 Clustering

The clustering methodology varies between the supervised, deep methods (e.g., GCN-VE and STAR-FC) and the unsupervised, shallow methods (e.g., k-means). We discuss each below. Appendix D.5.2 contains additional methodological details, architecture details for GCN-VE and STAR-FC, and an enumeration of all hyperparameters and their tuned values across all clustering methods.

For supervised clustering, we compute the 256D embeddings for the clustering-train, validation, and test splits. Methods are trained using the clustering-train embeddings from each dataset with the loss monitored on the validation split for early stopping and hyperparameter tuning. A hyperparameter search was performed for each of the datasets and we report results using the set of hyperparameter values associated with maximal clustering performance on the validation split. The model is then applied, inductively, on the test embeddings and final performance is reported.

In contrast, the unsupervised methods operate directly on the embeddings from the test split(s). A hyperparameter search was also conducted for the unsupervised methods, however, we report results using the set of values associated with maximal clustering performance on the test split(s). We admit that optimizing for test performance appears to provide an unfair advantage to the unsupervised methods, however, the supervised methods: (1) have access to additional splits of data unavailable to the unsupervised methods, (2) have thousands of learnable parameters, and (3) have 12 and 16 hyperparameters for STAR-FC and GCN-VE, respectively, compared to at-most 4 hyperparameters for the unsupervised methods.

Unnormalized Embeddings			
	Clustering Train	Validation	Test
Cars196	0.67	0.76	0.76
SOP	0.72	0.70	0.73
Dataset 3	0.91	0.91	0.91, 0.89, 0.88, 0.87, 0.86

ℓ_2-Normalized Embeddings			
	Clustering Train	Validation	Test
Cars196	0.69	0.80	0.79
SOP	0.75	0.74	0.77
Dataset 3	0.94	0.95	0.94, 0.93, 0.92, 0.91, 0.91

Table 6.2: Recall@1 on the clustering-train, validation, and test splits of each dataset for both unnormalized and ℓ_2-normalized embeddings. The comma-separated values for the test column of Dataset 3 represent the five test splits.

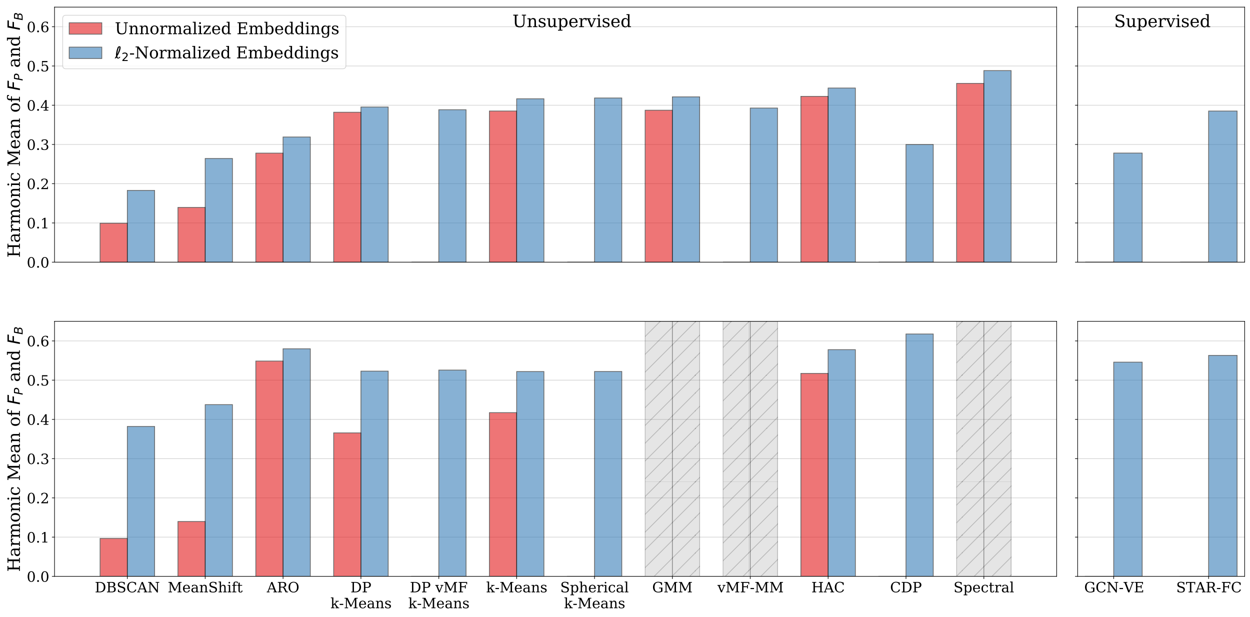


Figure 6.1: Harmonic mean of Pairwise (F_P) and BCubed (F_B) F-scores across clustering methods for Cars196 (top) and SOP (bottom). The left and right pane contain results for unsupervised and supervised clustering methods, respectively, and the red and blue bars indicate clustering of unnormalized and ℓ_2-normalized embeddings, respectively. GMM, vMF-MM, and Spectral could not be run on SOP due to runtime inefficiencies, indicated by the gray, hatched bars. The methods that omit a result for unnormalized embeddings assume ℓ_2-normalized embeddings, by default.

	Test #1		Test #2		Test #3		Test #4		Test #5	
	F_P	F_B	F_P	F_B	F_P	F_B	F_P	F_B	F_P	F_B
Minibatch k-Means	0.64	0.66	0.60	0.61	0.57	0.58	0.55	0.56	0.54	0.55
DP k-Means	0.67	0.68	0.62	0.60	0.61	0.59	0.45	0.53	0.43	0.49
DP vMF k-Means	0.72	0.70	0.67	0.65	0.60	0.58	0.61	0.59	0.61	0.59
HAC	0.74	0.75	0.69	0.72	0.66	0.69	DNC		DNC	
ARO	0.55	0.59	0.53	0.59	0.50	0.56	0.50	0.51	0.48	0.49
DBSCAN	0.32	0.41	0.21	0.40	0.20	0.20	0.20	0.20	0.19	0.20
CDP	0.62	0.62	0.59	0.59	0.56	0.56	0.54	0.54	0.52	0.53
GCN-VE	**0.78**	**0.79**	**0.74**	**0.75**	**0.71**	**0.72**	**0.68**	**0.70**	0.62	**0.65**
STAR-FC	0.65	0.75	0.64	0.72	0.65	0.69	0.65	0.66	**0.63**	0.64

	Test #1	Test #2	Test #3	Test #4	Test #5
Minibatch k-Means	0.2h	0.7h	1.3h	2.5h	4h
DP k-Means	13h	70h	127h	253h	324h
DP vMF k-Means	29h	98h	27h	9h	12h
HAC	6h	35h	97h	DNC	DNC
ARO	0.5h	1.8h	4h	3h	9h
DBSCAN	0.1h	0.4h	0.8h	1.4h	1.9h
CDP	0.4h	1.3h	3h	5h	8h
GCN-VE	132h, 0.3h	132h, 0.7h	132h, 1.6h	132h, 2.8h	37h, 2.5h
STAR-FC	45h, 0.2h	45h, 0.6h	45h, 0.9h	45h, 1.3h	45h, 1.8h

Table 6.3: Pairwise (F_P) and BCubed (F_B) F-scores (top) and time to cluster in hours (bottom) across clustering methods for all test splits of Dataset 3. *DNC* indicates that the method *did not converge* in a reasonable amount of time. The time to cluster for GCN-VE and STAR-FC contains the train time and test time separated by a comma. Boldface indicates the highest value for a metric. Blue indicates that the particular run used unnormalized embeddings instead of ℓ_2-normalized embeddings.

There are no alterations to the embeddings other than an ℓ_2-normalization step that we found to unequivocally improve clustering performance. Section 6.3.2 further discusses the benefits of ℓ_2-normalization.

6.3 Experimental Results

Using the methodology outlined in Section 6.2, we conduct extensive experiments on Cars196, Stanford Online Products (SOP), and Dataset 3 across 17 clustering methods. We begin by measuring the discriminability of the embeddings learned by the backbones and then move on to the clustering results. Details on compute resources are included in Appendix D.1.

6.3.1 Backbone Results

One connection we attempt to make in our experimentation is between embedding discriminability and clustering performance: the likelihood of deep clustering methods outperforming shallow methods increases as classes are better discriminated. Thus, we quantify embedding discriminability across all three datasets for reference in coming sections. Table 6.2 contains Recall@1 for the clustering-train, validation, and test splits of each dataset. The top table uses unnormalized embeddings (i.e., embeddings that lie in Euclidean space) and the bottom table uses ℓ_2-normalized embeddings (i.e., embeddings projected onto the surface of a unit hypersphere). Consistent with Chapter 4, we find that ℓ_2-normalization leads to strictly improved discrimination, as it removes intra-class variance.

We chose Cars196, SOP, and Dataset 3 as benchmarks because the discriminability of the embeddings varies significantly, with Cars196 and SOP performance between 70% and 80%, and Dataset 3 above 90%, for ℓ_2-normalized embeddings. We emphasize that this is a confound not considered in past work on face clustering. The datasets previously explored all have Recall@1 performance above 90% and some even near ceiling [28].

6.3.2 Clustering Results

We measure clustering performance on the test splits of all datasets. The clustering results focus on two metrics: Pairwise (F_P) [110] and BCubed (F_B) [3] F-scores. Because the metrics have different emphases—specifically, F_P emphasizes fidelity of larger clusters and F_B emphasizes fidelity of clusters proportional to their size—we report their harmonic mean when not reporting both, individually.

Figure 6.1 shows clustering performance for Cars196 (top) and SOP (bottom) across 12 unsupervised methods and the 2 supervised methods. Appendix D.2 contains tabulated results used to construct the figure, as well as additional metrics such as normalized mutual information. The red and blue bars represent performance on unnormalized and ℓ_2-normalized embeddings, respectively. There are two key takeaways: (1) using ℓ_2-normalized embeddings consistently improves clustering performance, sometimes upward of 30%, and (2) the supervised, deep methods are outperformed by unsupervised, shallow methods and are not even among the top-3 performers, contrary to past work reporting consistent state-of-the-art performance. Among the unsupervised methods, there is no clear best-performing method, however, HAC performs strongly on both datasets. As a point of comparison, HAC outperforms GCN-VE and STAR-FC by 16% and 6% for Cars196, respectively, and 3% and 1% for SOP, respectively. The results on Cars196 and SOP highlight a downside of the deep methods, particularly that they become fragile in the presence of uncertainty (i.e., less discriminability) in the embedding space.

To verify we can reproduce the benefits of GCN-VE and STAR-FC, we performed a similar clustering analysis on Dataset 3, where Recall@1 is above 90% for all splits. Table 6.3 contains Pairwise and BCubed F-scores (top) and the time to cluster (bottom) across the five test splits. Notice that the results are inverted: GCN-VE and STAR-FC consistently outperform the shallow methods. However, the margin between HAC, for example, and GCN-VE is less than 5%—for the test splits where HAC converged. Comparing to the results in Yang et al. [142] and Shen et al. [108], GCN-VE is shown to consistently outperform HAC by upward of 20%. In agreement with results reported above for Cars196 and SOP, ℓ_2-normalized embeddings, again, outperform unnormalized embeddings, compatible with prior work on embedding-learning and face verification [28, 75, 94, 95, 104, 128, 129, 131]. The only exception is for test splits #4 and #5 for DP k-means, where unnormalized embeddings were superior.

In addition to state-of-the-art performance, another benefit underlined in past work for GCN-VE and STAR-FC is their efficient inference time. We computed the time to cluster the Dataset 3 test splits in the bottom of Table 6.3. We also find that GCN-VE and STAR-FC are among the most efficient methods at inference time, outpaced only by DBSCAN and minibatch k-means. However, one factor not presented in past work is the *total* time to employ deep clustering methods, that is, including the additional time to train the methods. We find that training can take tens

	Cars196		SOP	
	F_P	F_B	F_P	F_B
GCN-VE [142]	0.26	0.37	0.44	0.63
GCN-VE (ours)	0.22	0.37	0.47	0.65

Table 6.4: Comparison of Pairwise (F_P) and BCubed (F_B) F-scores for the open-source GCN-VE code and our reimplementation on Cars196 and SOP.

	Cars196		SOP		Dataset 3 Test #1	
	F_P	F_B	F_P	F_B	F_P	F_B
Tree Deduction	0.13	0.36	0.44	0.63	0.38	0.41
+ ℓ_2-Norm	**0.25**	**0.37**	0.44	0.63	0.62	0.65
+ GCN-E	0.19	**0.37**	0.44	0.62	0.62	0.65
GCN-VE	0.22	**0.37**	**0.47**	**0.65**	**0.78**	**0.79**

Table 6.5: Ablation study comparing GCN-VE to simpler variants. The reported results are Pairwise (F_P) and BCubed (F_B) F-scores on Cars196, SOP, and test split #1 of Dataset 3. Boldface indicates the highest value for a metric.

of hours and when factored in with the inference time, presents a trade-off for the deep methods. Depending on the application, GCN-VE and STAR-FC may only be used for inference one time, whereas if they are used in a production system may be used for inference hundreds of times. When considering the total time, in addition to their marginal improvements over much simpler methods, we encourage inspection into the amortized time (i.e., the dispersion of training time amongst the expected number of inference runs). In the case of fewer inference runs, methods such as minibatch k-means may provide adequate performance while producing a more-efficient amortized runtime.

6.3.2.1 Investigation into GCN-VE

Due to the unexpected results of GCN-VE on Cars196 and SOP, we conducted further investigation of the method to verify our findings. First, we compared our reimplementation of GCN-VE to the open-source version. Table 6.4 compares the two implementations on Cars196 and SOP. We confirm that our results, within reasonable variance, match the implementation from Yang et al.

[142]. Next, we ablated the components of GCN-VE to better understand which contributed to final performance. In general, GCN-VE works by: (1) estimating a confidence for each embedding via the GCN-V graph convolutional network, (2) constructing subgraphs using the estimated confidences, (3) estimating which edges in the subgraphs are valid connections via the GCN-E graph convolutional network, and (4) running a tree-deduction algorithm to form the final clusters.

We consider three ablated variants of GCN-VE, described below. All that is needed to form clusters is to run tree deduction on a k-nearest-neighbors subgraph with edges pruned by a distance threshold. We refer to this first ablation as *Tree Deduction*. Based on empirical analysis from Chapter 4, the embedding ℓ_2-norm is a measure of confidence, thus, we can replace the GCN-V network with the ℓ_2-norm of each embedding. Additionally, we can replace the GCN-E network simply by considering edges valid if they are connected to higher-confidence embeddings within a distance threshold, and run tree deduction as is. We refer to this second ablation as *Tree Deduction + ℓ_2-norm*. The third ablation, *Tree Deduction + ℓ_2-norm + GCN-E*, reintroduces GCN-E, but still uses ℓ_2-norm instead of the GCN-V network.

Table 6.5 measures clustering performance of the three ablations against GCN-VE for Cars196, SOP, and test split #1 of Dataset 3. Interestingly, one can recover the majority of GCN-VE performance with the Tree Deduction + ℓ_2-norm variant, corroborating results from Chapter 4 on ℓ_2-norm conveying embedding confidence. Additionally, the critical component of GCN-VE is not the GCN-E network, but GCN-V. Tree Deduction + ℓ_2-norm performs as well, if not better, than Tree Deduction + ℓ_2-norm + GCN-E.

6.3.2.2 Degradation Study

While the experimental results thus far support our claim that the benefits of deep clustering methods are tied to embedding discriminability, we admit there are confounds. First, Cars196 and SOP are non-face datasets and second, they have very different dataset statistics compared to Dataset 3 and the historically-used face datasets, summarized in Table 6.1. To remove these confounds, we perform a degradation study on Dataset 3 where we randomly select a subset of

	32D	64D	128D	256D (original)
Dataset 3 Clustering Train	0.68	0.86	0.92	0.94

Table 6.6: Recall@1 on the clustering-train split of Dataset 3 as the embedding dimensionality increases.

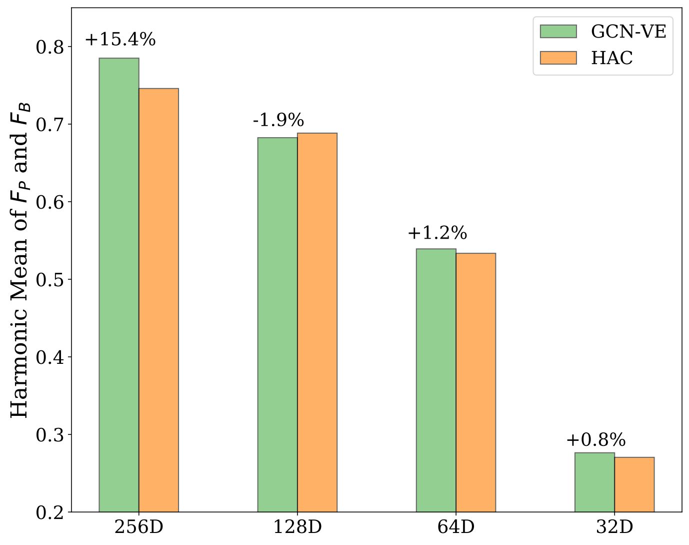


Figure 6.2: Harmonic mean between Pairwise (F_P) and BCubed (F_B) F-scores of GCN-VE and HAC on test split #1 of Dataset 3 for decreasing embedding dimensionalities. The percentages above the GCN-VE (green) bars indicate its relative reduction in error compared to HAC.

embedding dimensions to ignore, thus decreasing the discriminability. We remove the same embedding dimensions for all splits of the data to ensure they are compatible. Table 6.6 measures the Recall@1 performance of the clustering-train split of Dataset 3 for varying embedding dimensionalities. As expected, the lower-dimensional spaces have reduced discriminability, for example, the 32D space has Recall@1 dropping 26% compared to the original 256D embedding. Figure 6.2 measures the harmonic mean of F_P and F_B for GCN-VE and HAC on test split #1 of Dataset 3 as we decrease the embedding dimensionality. The tabulated results composing the figure are included in Appendix D.4. The percentage above the GCN-VE (green) bars indicates the relative error reduction that GCN-VE achieves over HAC. Note that for the original 256D embedding space, GCN-VE has a 15% relative error reduction over HAC, but by projecting the embedding space to

128D—decreasing embedding discriminability by a mere 2%—GCN-VE loses all advantage over HAC. Thus, while we do not see as large of a performance drop for GCN-VE relative to HAC, comparatively, as we do for Cars196, the degradation study is consistent with our SOP results.

6.4 Discussion and Conclusions

Many of the innovations in supervised methods for clustering pretrained embeddings are tied to face verification. The methods were benchmarked primarily on face datasets and motivated by enlarging them via pseudo-supervising the vast amounts of unlabeled face data. Common face datasets (e.g., MS-Celeb1M [41] and MegaFace [55]) are no longer publicly available, with progress reliant on using unofficial, shared embeddings extracted from pretrained backbones. Additionally, the impressive improvements of supervised methods are bound to face embeddings, shown to have verification and Recall@1 performance well above 90% and in some cases near ceiling [28]. Finally, results of recent work may implicitly favor the novel, supervised methods based on methodological choices such as the lack of a validation split and hyperparameter tuning based on test performance.

Our goals in conducting a large-scale empirical study on clustering pretrained embeddings are to: (1) broaden the scope of supervised clustering methods beyond faces, (2) present an end-to-end pipeline including backbone training along with results for benchmarking future methods, and (3) benchmark the robustness of supervised, deep methods on embeddings with less discriminability. We do so by presenting benchmarks for 17 clustering methods on three datasets: Cars196, SOP, and a third dataset, Dataset 3. Cars196 and SOP are not only from diverse visual domains, but have very different dataset statistics (i.e., Table 6.1) and embedding discriminability (i.e., Table 6.2) compared to the historically-used face datasets. We emphasize that Dataset 3 was chosen to corroborate results from past embedding-clustering research. We discuss conclusions from our study below.

Embedding discriminability vs. supervised clustering performance. The main hypothesis we verify is the existence of a relationship between how discriminative the embeddings are (i.e., Recall@1) and the benefits of supervised, deep clustering methods. We see for Cars196 and

SOP—where embedding Recall@1 is between 70% and 80%—that the state-of-the-art supervised methods are not among the top 3 performers. They are outperformed by HAC and spectral clustering, for example. In contrast, for Dataset 3—where embedding Recall@1 is above 90%—GCN-VE is a consistent state-of-the-art method followed by STAR-FC. To remove any confounds caused by the domain or the dataset statistics, we show that we can remove all benefits of the supervised methods by randomly projecting the Dataset 3 embeddings from 256D to 128D (i.e., decreasing Recall@1 by a mere 2%). We leave it to future work to amend the fragility of supervised, deep methods, but we hypothesize that it may be caused by susceptibility of these methods to overfit spurious correlations in the embedding space that aren't generalizable and that simpler heuristics are more effective in cases of high embedding uncertainty.

Consistency with results from past work. We believe certain methodological choices such as not using a validation split and using the test split for hyperparameter tuning and early stopping may have implicitly favored the deep methods, which are more flexible and have significantly more hyperparameters than the shallow methods. Instead, we use a validation set for hyperparameter selection and early stopping for the deep methods, as well as reimplement and evaluate all baselines, and find that the unsupervised methods are actually competitive. For example, HAC only underperforms state-of-the-art on Dataset 3 by 5%, whereas past work indicates it can underperform by 20% or more consistently [108, 142]. While supervised methods are clearly superior on Dataset 3, the margin of improvement is of much smaller magnitude than previously reported.

Benefits of ℓ_2-normalization. It has been shown that softmax cross-entropy and its variants discover the most-discriminative embedding spaces [10, 84, 104, 121]. Due to the structure of the embeddings, the majority of intra-class variance extends outward from the origin. As a result, we verify in Chapter 4 that ℓ_2-normalization of non-spherical embeddings leads to robust Recall@1 improvements and spherical methods result in further improvements still, which we find transfers directly to clustering performance (i.e., Figure 6.1). For clustering pretrained embeddings, particularly trained with some form of softmax cross-entropy, we recommend ℓ_2-normalization prior to clustering.

Performance vs. runtime tradeoff. One benefit of the supervised methods is their efficient inference time. GCN-VE and STAR-FC are outpaced only by DBSCAN and minibatch k-means in Table 6.3 for Dataset 3. However, the time to train supervised methods is non-negligible and one should consider amortized runtime when choosing among methods. If the goal is to use clustering for visualization, for example, and inference is only going to run once, the total runtime may be too costly given the marginal improvements over much simpler and faster unsupervised methods.

On the value of shallow methods. Given recent trends, one might expect the supervised, deep methods to be strictly superior to the unsupervised, shallow methods that have become commonplace for clustering. However, broadening the scope beyond the face domain has underlined their fragility in the presence of embedding uncertainty and emphasized the value of shallow methods. We find that fundamental methods such as spectral clustering and HAC can outperform GCN-VE and STAR-FC despite having three times fewer hyperparameters and no learnable parameters. By proposing new benchmarks on Cars196 and SOP, we hope that our empirical study serves as the foundation for which supervised methods can be further improved.

6.5 Ethical Considerations

A major goal of our research is to broaden the study of clustering pretrained embeddings beyond the facial verification domain, which we do by providing benchmarks on Cars196 and SOP. We appreciate the sensitivity associated with research on human data, including images of faces, and emphasize that clustering should only be employed in contexts that are responsible, socially beneficial, and fair, for example, in personalization of products for better user experiences, and not as part of harmful technologies or systems. Additionally, we acknowledge that non-human datasets, including Cars196 and SOP, can exhibit bias through lack of representation. We chose to report only on the downstream clustering performance, as to not reflect on or reinforce biases present in the data.

Chapter 7

Conclusions

Our research on deep visual representations for classification and retrieval focuses on three aspects: (1) uncertainty, (2) geometry, and (3) applicability. We present and evaluate two of the first supervised, stochastic methods for deep visual representation learning: the Stochastic Prototype Embedding (SPE) in Chapter 3 and von Mises–Fisher Softmax (vMF) in Chapter 4. The former is a Euclidean variant extended from Prototypical Networks [112] and the latter is a spherical variant based on Cosine Softmax [61, 129, 149]. Table 7.1 partitions the space of stochasticity and geometry where our methods are italicized. In addition to research on *learning* deep visual representations, we also present two studies *applying* them in practical domains. Chapter 5 leverages spherical representations for joint few- and zero-shot egocentric action recognition and Chapter 6 explores a suite of methods for clustering pretrained deep visual representations highlighting the robust improvements from using spherical embeddings. Below is a detailed summary of our contributions:

- *The Stochastic Prototype Embedding is a novel stochastic method for learning supervised,*

Table 7.1: Methods for discovering visual representations at the conjunction of geometry (columns) and uncertainty (rows). Stochastic Prototype Embeddings and von Mises–Fisher Softmax are methods we presented, denoted by italic typeface.

	Euclidean	**Spherical**	**Hyperbolic**
Deterministic	Prototypical Networks [112]	Cosine Softmax [61, 129, 149]	Hyperbolic Softmax [36]
Stochastic	*Stochastic Prototype Embeddings* (Chapter 3) [106]	*von Mises–Fisher Softmax* (Chapter 4) [104]	N/A

Euclidean embeddings which encode axis-aligned uncertainty in their distributions.

Extended from Prototypical Networks (PN) [112], SPE is a supervised, Euclidean method for discovering visual representations that treats prototypes and queries as Gaussian random variables, where queries are classified based on proximity to prototypes. SPE outperforms PN, as well as the only other fully-developed alternative stochastic method at the time, Hedged Instance Embedding, on few-shot classification, fixed-set classification, and inductive transfer learning. Due to a confidence-weighted average, SPE is particularly robust to uncertainty in the support set and is as time- and space-efficient as the PN because of a novel intersection sampler. Lastly, SPE obtains interpretable, robust, and disentangled representations by encoding uncertainty with a diagonal covariance matrix.

- *The von Mises–Fisher Loss is a novel stochastic method for learning supervised, spherical embeddings that is on-par with state-of-the-art deterministic methods while reducing calibration error up to 70%, relatively.*

Our investigations into deterministic, supervised, visual representation learning methods led us to discover that for spherical methods, the ℓ_2-norm of the embedding encodes information about uncertainty or ambiguity. Rather than factoring the embedding norm out of the classification decision, as other spherical methods do, we explicitly represent the norm as the concentration of a von Mises–Fisher distribution, where the normalized embedding serves as the direction. Marginalizing embedding and class-weight uncertainty over the cosine-softmax classification posterior results in a tractable loss that performs on-par, or in some instances better, than state-of-the-art deterministic methods. The stochasticity in the representation improves robustness, reducing expected calibration error significantly over the deterministic alternatives. Additionally, vMF is a significant improvement over SPE, as it is able to train effectively in high-dimensional embedding spaces and compete with state-of-the-art methods, whereas SPE is only on-par with PN—a method no longer state-of-the-art—in higher dimensions.

- *Cross-modal few-shot classification is a unification of few- and zero-shot classification. Representations can be trained to jointly perform both tasks, and those that perform best in one task also perform best in the other, implying no performance trade-off.*

Cross-modal few-shot classification is a unification of few- and zero-shot classification that acknowledges a different modality can be used as a support set when its representation is fused to a representation for the query-set modality. We propose and evaluate procedures for spherical methods jointly performing few- and zero-shot classification. The results indicate that a single representation can be state-of-the-art in both few- and zero-shot classification, indicating no performance trade-off.

- *We developed three new dataset splits for EPIC-KITCHENS [23] specifically designed to evaluate open-set generalization and facilitate future research on egocentric action recognition.*

Through work on unifying few- and zero-shot egocentric action recognition, we present three new splits for a popular large-scale dataset, EPIC-KITCHENS [23]. The splits are designed for open-set classification, where the set of classes, or actions, during testing are disjoint from those used during training. The splits form the data used to evaluate spherical methods on few-classification, zero-shot classification, and their unification, cross-modal few-shot classification.

- *We study a suite of 17 methods for clustering pretrained embeddings on datasets outside of face verification and find that the state-of-the-art, deep methods are surprisingly fragile, where they underperform shallow methods such as k-means. Additionally, ℓ_2-normalization of the embeddings prior to clustering leads to robust improvements across all shallow methods capable of operating on spherical embeddings.*

Many of the innovations in supervised, deep methods for clustering pretrained embeddings are tied to the face verification domain, where the embeddings are highly discriminative. We conduct an empirical analysis of 17 methods across three datasets—including Cars196 and SOP—with the goal of broadening the scope beyond faces. We find that (1) unsupervised, shallow methods (e.g., k-means) are surprisingly competitive when carefully tuned and benchmarked, (2) state-of-the-art, supervised, deep methods are fragile when operating on less-discriminative embeddings, and (3) ℓ_2-normalization of embeddings (i.e., spherical embeddings) results in robust improvements across all clustering methods. The last finding, particularly, validates results from Chapter 4, which emphasizes the benefits of spherical embeddings for classification and retrieval.

7.1 Limitations

While the research presented does highlight the effectiveness and broader potential for learning embeddings with a focus on uncertainty and geometry, we acknowledge the limitations of our research below.

Nearly all of the commonly-used datasets for learning deep discriminative visual representations have been preprocessed to produce clean, single-object images ready for classification or to be queried for retrieval. Employing methods such as SPE or vMF would require similarly clean, supervised data for training, and other machine learning models prior to the classification or retrieval stage to produce quality images at inference time. Such models may perform tasks including detection and segmentation, for example. Additionally, we assume data that is independently and identically distributed (IID) from a single snapshot in time. Real-world applications, for example, egocentric action recognition, involve data to be processed in a time series exhibiting temporal structure and other methods may have inductive biases better equipped to handle said data.

Another limitation of our research involves scale. While Chapters 5 and 6 validate the benefits of spherical embeddings and ℓ_2-norm as a confidence signal on large-scale datasets (e.g., EPIC-KITCHENS), the experimental results for SPE and vMF were more constrained in terms of scale. The aspects of scale relevant to the stochastic embedding methods and the datasets used are the resolution of the input images, number of instances per class, number of classes, and the embedding dimensionality. SPE was much more limited in terms of these aspects than vMF, but we reached an asymptote with SPE's performance, where scaling the method further would not overcome the challenges associated with the curse of dimensionality. For vMF, however, the 7 datasets explored varied from low-resolution images (e.g., 28x28) to high-resolution images matching those of ImageNet (e.g., 224x224). Additionally, the number of instances per class and number of classes varied from approximately 5 instances (i.e., few-shot) up to 5000 instances, and 10 classes up to nearly 10000 classes, respectively. The strongest limitation of vMF, noted in Chapter 4, is that the method struggled to generalize when trained on a large number of classes (e.g., 9620 for SOP). The largest embedding dimensionality explored was 512D for vMF and we note that 128D is very common in production settings that balance the trade-off of fast similarity search with high retrieval performance. Table 6.6 also shows that the gains in Recall@1 rapidly decline as the embedding dimensionality increases. For example, moving from 128D to 256D results in only 2% improved Recall@1, despite doubling the required memory.

The last limitation we acknowledge is our notion of an "open set" of classes. For many downstream, open-set applications of embeddings trained for classification and retrieval (e.g., few-shot classification or clustering), it is assumed that the novel classes at inference time are very similar to the classes learned during training. As the inference classes drift from the training classes, the discriminative features explicitly represented in the embedding become less useful. Thus, the transferability of representations to downstream tasks far from the training domain diminishes, especially for spherical methods that produce highly-discriminative embeddings [61]. For downstream tasks such as segmentation, detection, or generation, it is beneficial for the representation to preserve information beyond that necessary for producing a classification or retrieval decision. Self-supervised methods have been shown to be better for a general set of vision-based tasks [45].

7.2 Future Work

Of course, research never truly comes to an end, and many of the ideas we presented bring about new questions and directions for future work. We enumerate several of these ideas below.

First, it is not apparent how well SPE and vMF would generalize in real-world, production settings where discriminative representations are employed—especially those that require segmentation or detection models that produce the input to the embedding method. A domain that would be valuable to evaluate stochastic embedding methods for the above generalization is self-driving cars. One could imagine SPE or vMF trained to recognize objects detected by a self-driving vehicle, especially because the set of classes is functionally open. Furthermore, the uncertainty in the embeddings could be very useful for downstream models that may plan the vehicle's trajectory, for example.

Second, as indicated by Table 7.1, we are not aware of stochastic embedding methods developed for the hyperbolic geometry. Hyperbolic representations are useful for tasks with hierarchical structure, not just within visual domains, and the stochasticity is likely beneficial for interpretation and robustness in the same ways it is beneficial in Euclidean and spherical geometries.

Third, our experimental results are solely limited to vision-based tasks. There is no reason, in principle, that prevents SPE and vMF from being used in, for example, natural language or audio processing. Both domains have classification and retrieval tasks requiring discriminative representations that are interpretable and robust. Additionally, benchmarking SPE and vMF outside of the visual domain would provide insight into the generality and usefulness of the methods, and could lead to wider adoption of stochastic representations.

Lastly, we verify the robustness of stochastic embeddings via calibration and interpretability, but an open question is if the uncertainty is useful for out-of-distribution detection broadly. It is still unclear if the uncertainty can be used for answering questions like: "Does this input belong to a class never seen before?" or "Is this input from a completely different domain?" Models that can effectively answer these questions could opt not to produce a classification decision or be able to provide anomaly detection out-of-the-box.

Bibliography

[1] Alemi, A. A., Fischer, I., Dillon, J. V., and Murphy, K. (2017). Deep Variational Information Bottleneck. In International Conference on Learning Representations.

[2] Allen, K. R., Shelhamer, E., Shin, H., and Tenenbaum, J. B. (2019). Infinite Mixture Prototypes for Few-Shot Learning. In International Conference on Machine Learning.

[3] Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, F. (2009). A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints. Information Retrieval, 12(4).

[4] Arthur, D. and Vassilvitskii, S. (2007). k-Means++: The Advantages of Careful Seeding. In ACM-SIAM Symposium on Discrete Algorithms.

[5] Attarian, M., Roads, B. D., and Mozer, M. C. (2020). Transforming Neural Network Visual Representations to Predict Human Judgments of Similarity. NeurIPS Workshop on Shared Visual Representations in Human and Machine Intelligence.

[6] Ba, J. L., Swersky, K., Fidler, S., and Salakhutdinov, R. (2015). Predicting Deep Zero-Shot Convolutional Neural Networks Using Textual Descriptions. In IEEE/CVF International Conference on Computer Vision.

[7] Banerjee, A., Dhillon, I. S., Ghosh, J., and Sra, S. (2005). Clustering on the Unit Hypersphere Using von Mises–Fisher Distributions. Journal of Machine Learning Research, 6.

[8] Belhaj, M., Protopapas, P., and Pan, W. (2018). Deep Variational Transfer: Transfer Learning through Semi-supervised Deep Generative Models. arXiv e-prints 1812.03123 cs.LG.

[9] Bishay, M., Zoumpourlis, G., and Patras, I. (2019). TARN: Temporal Attentive Relation Network for Few-Shot and Zero-Shot Action Recognition. In British Machine Vision Conference.

[10] Boudiaf, M., Rony, J., Ziko, I. M., Granger, E., Pedersoli, M., Piantanida, P., and Ayed, I. B. (2020). A Unifying Mutual Information View of Metric Learning: Cross-Entropy vs. Pairwise Losses. In European Conference on Computer Vision.

[11] Bridle, J. S. (1989). Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. In Advances in Neural Information Processing Systems 2.

[12] Bridle, J. S. (1990). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In Neurocomputing. Springer.

[13] Cakir, F., He, K., Xia, X., Kulis, B., and Sclaroff, S. (2019). Deep Metric Learning to Rank. In IEEE Conference on Computer Vision and Pattern Recognition.

[14] Cao, K., Ji, J., Cao, Z., Chang, C., and Niebles, J. C. (2020). Few-Shot Video Classification via Temporal Alignment. In IEEE Conference on Computer Vision and Pattern Recognition.

[15] Careaga, C., Hutchinson, B., Hodas, N., and Phillips, L. (2019). Metric-Based Few-Shot Learning for Video Action Recognition. arXiv e-prints 1909.09602 cs.CV.

[16] Carreira, J. and Zisserman, A. (2017). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In IEEE Conference on Computer Vision and Pattern Recognition.

[17] Chang, J., Lan, Z., Cheng, C., and Wei, Y. (2020). Data Uncertainty Learning in Face Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[18] Chen, B., Liu, W., Yu, Z., Kautz, J., Shrivastava, A., Garg, A., and Anandkumar, A. (2020a). Angular Visual Hardness. In International Conference on Machine Learning.

[19] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020b). A Simple Framework for Contrastive Learning of Visual Representations. In International Conference on Machine Learning.

[20] Cheng, Y. (1995). Mean Shift, Mode Seeking, and Clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[21] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a Similarity Metric Discriminatively, with Application to Face Verification. In IEEE Conference on Computer Vision and Pattern Recognition.

[22] Collier, M., Mustafa, B., Kokiopoulou, E., Jenatton, R., and Berent, J. (2020). A Simple Probabilistic Method for Deep Classification under Input-Dependent Label Noise. arXiv e-prints 2003.06778 cs.LG.

[23] Damen, D., Doughty, H., Farinella, G. M., Fidler, S., Furnari, A., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., and Wray, M. (2018). Scaling Egocentric Vision: The EPIC-KITCHENS Dataset. In European Conference on Computer Vision.

[24] Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. (2018). Hyperspherical Variational Auto-Encoders. In Conference on Uncertainty in Artificial Intelligence.

[25] de Brébisson, A. and Vincent, P. (2016). An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family. In International Conference on Learning Representations.

[26] De Cao, N. and Aziz, W. (2020). The Power Spherical Distrbution. In International Conference on Machine Learning.

[27] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A Large-Scale Hierarchical Image Database. In IEEE Conference on Computer Vision and Pattern Recognition.

[28] Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[29] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv e-prints 1810.04805 cs.CL.

[30] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In AAAI Conference on Knowledge Discovery and Data Mining.

[31] Farhadi, A., Endres, I., Hoiem, D., and Forsyth, D. (2009). Describing Objects by Their Attributes. In IEEE Conference on Computer Vision and Pattern Recognition.

[32] Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In International Conference on Machine Learning.

[33] Fort, S. (2017). Gaussian Prototypical Networks for Few-Shot Learning on Omniglot. NeurIPS Workshop on Bayesian Deep Learning.

[34] Frey, B. J. and Dueck, D. (2007). Clustering by Passing Messages Between Data Points. Science.

[35] Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Ranzato, M., and Mikolov, T. (2013). DeViSE: A Deep Visual-Semantic Embedding Model. In Advances in Neural Information Processing Systems 26.

[36] Ganea, O., Becigneul, G., and Hofmann, T. (2018). Hyperbolic Neural Networks. In Advances in Neural Information Processing Systems 31.

[37] Goldberger, J., Hinton, G. E., Roweis, S., and Salakhutdinov, R. R. (2004). Neighbourhood Components Analysis. In Advances in Neural Information Processing Systems 17.

[38] Gopal, S. and Yang, Y. (2014). von Mises–Fisher Clustering Models. In International Conference on Machine Learning.

[39] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. In International Conference on Machine Learning.

[40] Guo, S., Xu, J., Chen, D., Zhang, C., Wang, X., and Zhao, R. (2020). Density-Aware Feature Embedding for Face Clustering. In IEEE Conference on Computer Vision and Pattern Recognition.

[41] Guo, Y., Zhang, L., Hu, Y., He, X., and Gao, J. (2016). MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition. In European Conference on Computer Vision.

[42] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. In IEEE Conference on Computer Vision and Pattern Recognition.

[43] Hahn, M., Silva, A., and Rehg, J. M. (2019). Action2Vec: A Crossmodal Embedding Approach to Action Learning. arXiv e-prints 1901.00484 cs.CV.

[44] Hasnat, M. A., Bohné, J., Milgram, J., Gentric, S., and Chen, L. (2017). von Mises–Fisher Mixture Model-based Deep learning: Application to Face Verification. arXiv e-prints 1706.04264 cs.CV.

[45] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum Contrast for Unsupervised Visual Representation Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[46] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[47] He, Y., Cao, K., Li, C., , and Loy, C. C. (2018). Merge or Not? Learning to Group Faces via Imitation Learning. In AAAI Conference on Artificial Intelligence.

[48] Hermans, A., Beyer, L., and Leibe, B. (2017). In Defense of the Triplet Loss for Person Re-Identification. arXiv e-prints 1703.07737 cs.CV.

[49] Hornik, K., Feinerer, I., Kober, M., and Buchta, C. (2012). Spherical k-Means Clustering. Journal of Statistical Software.

[50] Hu, J., Lu, J., and Tan, Y. P. (2014). Discriminative Deep Metric Learning for Face Verification in the Wild. In IEEE Conference on Computer Vision and Pattern Recognition.

[51] Jain, A. K. (2010). Data Clustering: 50 Years Beyond k-Means. Pattern Recognition Letters.

[52] Jégou, H., Douze, M., and Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[53] Kaiser, L., Nachum, O., Roy, A., and Bengio, S. (2017). Learning to Remember Rare Events. In International Conference on Learning Representations.

[54] Karaletsos, T., Belongie, S., and Rätsch, G. (2016). Bayesian Representation Learning with Oracle Constraints. In International Conference on Learning Representations.

[55] Kemelmacher-Shlizerman, I., Seitz, S. M., Miller, D., and Brossard, E. (2016). The Megaface Benchmark: 1 Million Faces for Recognition at Scale. In IEEE Conference on Computer Vision and Pattern Recognition.

[56] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Krishnan, D., and Liu, C. (2020). Supervised Contrastive Learning. In Advances in Neural Information Processing Systems 33.

[57] Khrulkov, V., Mirvakhabova, L., Ustinova, E., Oseledets, I., and Lempitsky, V. (2020). Hyperbolic Image Embeddings. In IEEE Conference on Computer Vision and Pattern Recognition.

[58] Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv e-prints 1412.6980 cs.LG.

[59] Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In International Conference on Learning Representations.

[60] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese Neural Networks for One-Shot Image Recognition. ICML Workshop on Deep Learning.

[61] Kornblith, S., Lee, H., Chen, T., and Norouzi, M. (2020). What's in a Loss Function for Image Classification? arXiv e-prints 2010.16402 cs.CV.

[62] Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3D Object Representations for Fine-Grained Categorization. In ICCV Workshop on 3D Representation and Recognition.

[63] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images.

[64] Kulis, B. and Jordan, M. I. (2012). Revisiting k-Means: New Algorithms via Bayesian Non-parametrics. In International Conference on Machine Learning.

[65] Kumar, A., Liang, P., and Ma, T. (2019). Verified Uncertainty Calibration. In Advances in Neural Information Processing Systems 32.

[66] Kumar, S. and Tsvetkov, Y. (2019). von Mises–Fisher Loss for Training Sequence to Sequence Models with Continuous Outputs. In International Conference on Learning Representations.

[67] Laenen, S. and Bertinetto, L. (2021). On Episodes, Prototypical Networks, and Few-Shot Learning. In Advances in Neural Information Processing Systems 34.

[68] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-Level Concept Learning through Probabilistic Program Induction. Science, 350(6266).

[69] Law, M., Liao, R., Snell, J., and Zemel, R. (2019). Lorentzian Distance Learning for Hyperbolic Representations. In International Conference on Machine Learning.

[70] LeCun, Y. and Cortes, C. (2010). MNIST Handwritten Digit Database.

[71] Li, W., Zhao, R., Xiao, T., and Wang, X. (2014). DeepReID: Deep Filter Pairing Neural Network for Person Re-Identification. In IEEE Conference on Computer Vision and Pattern Recognition.

[72] Lin, W.-A., Chen, J.-C., Castillo, C. D., and Chellappa, R. (2018). Deep Density Clustering of Unconstrained Faces. In IEEE Conference on Computer Vision and Pattern Recognition.

[73] Liu, J., Deng, Y., Bai, T., Wei, Z., and Huang, C. (2015). Targeting Ultimate Accuracy: Face Recognition via Deep Embedding. arXiv e-prints 1506.07310 cs.CV.

[74] Liu, T., Jones, C., Seyedhosseini, M., and Tasdizen, T. (2014). A Modular Hierarchical Approach to 3D Electron Microscopy Image Segmentation. Journal of Neuroscience Methods.

[75] Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. (2017). SphereFace: Deep Hypersphere Embedding for Face Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[76] Liu, W., Wen, Y., Yu, Z., and Yang, M. (2016a). Large-Margin Softmax Loss for Convolutional Neural Networks. In International Conference on Machine Learning.

[77] Liu, Z., Luo, P., Qiu, S., Wang, X., and Tang, X. (2016b). DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. In IEEE Conference on Computer Vision and Pattern Recognition.

[78] Lloyd, S. (1982). Least Squares Quantization in PCM. IEEE Transactions on Information Theory.

[79] Manning, C. D., Raghavan, P., and Schütze, H. (2009). An Introduction to Information Retrieval. Cambridge University Press.

[80] Mettes, P., van der Pol, E., and Snoek, C. G. M. (2019). Hyperspherical Prototype Networks. In Advances in Neural Information Processing Systems 32.

[81] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and Their Compositionality. In Advances in Neural Information Processing Systems 26.

[82] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A Simple Neural Attentive Meta-Learner. In International Conference on Learning Representations.

[83] Movshovitz-Attias, Y., Toshev, A., Leung, T. K., Ioffe, S., and Singh, S. (2017). No Fuss Distance Metric Learning Using Proxies. In IEEE/CVF International Conference on Computer Vision.

[84] Musgrave, K., Belongie, S., and Lim, S. N. (2020). A Metric Learning Reality Check. In European Conference on Computer Vision.

[85] Ng, A., Jordan, M., and Weiss, Y. (2001). On Spectral Clustering: Analysis and an Algorithm. In Advances in Neural Information Processing Systems.

[86] Nguyen, X.-B., Bui, D. T., Duong, C. N., Bui, T. D., and Luu, K. (2021). Clusformer: A Transformer Based Clustering Approach to Unsupervised Large-Scale Face and Visual Landmark Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[87] Nickel, M. and Kiela, D. (2017). Poincaré Embeddings for Learning Hierarchical Representations. In Advances in Neural Information Processing Systems 30.

[88] Oh, S. J., Murphy, K., Pan, J., Roth, J., Schroff, F., and Gallagher, A. (2019). Modeling Uncertainty with Hedged Instance Embedding. In International Conference on Learning Representations.

[89] Otto, C., Wang, D., and Jain, A. K. (2018). Clustering Millions of Faces by Identity. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[90] Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M. (2009). Zero-Shot Learning with Semantic Output Codes. In Advances in Neural Information Processing Systems 22.

[91] Parde, C. J., Castillo, C., Hill, M. Q., Colon, Y. I., Sankaranarayanan, S., Chen, J.-C., and O'Toole, A. J. (2017). Deep Convolutional Neural Network Features and the Original Image. arXiv e-prints 1611.01751 cs.CV.

[92] Park, J., Yi, S., Choi, Y., Cho, D.-Y., and Kim, J. (2019). Discriminative Few-Shot Learning Based on Directional Statistics. arXiv e-prints 1906.01819 cs.LG.

[93] Qian, Q., Shang, L., Sun, B., Hu, J., Tacoma, T., Li, H., and Jin, R. (2019). SoftTriple Loss: Deep Metric Learning Without Triplet Sampling. In IEEE/CVF International Conference on Computer Vision.

[94] Ranjan, R., Bansal, A., Xu, H., Sankaranarayanan, S., Chen, J.-C., Castillo, C. D., and Chellappa, R. (2019). Crystal Loss and Quality Pooling for Unconstrained Face Verification and Recognition. arXiv e-prints 1804.01159 cs.CV.

[95] Ranjan, R., Castillo, C. D., and Chellappa, R. (2017). L2-Constrained Softmax Loss for Discriminative Face Verification. arXiv e-prints 1703.09507 cs.CV.

[96] Ridgeway, K. (2016). A Survey of Inductive Biases for Factorial Representation-Learning. arXiv e-prints 1612.05299 cs.LG.

[97] Ridgeway, K. and Mozer, M. C. (2018). Learning Deep Disentangled Embeddings with the F-Statistic Loss. In Advances in Neural Information Processing Systems 31.

[98] Rippel, O., Paluri, M., Dollar, P., and Bourdev, L. (2016). Metric Learning with Adaptive Density Discrimination. In International Conference on Learning Representations.

[99] Roelofs, R., Cain, N., Shlens, J., and Mozer, M. C. (2021). Mitigating Bias in Calibration Error Estimation. arXiv e-prints 2012.08668 cs.LG.

[100] Ruiz-Antolín, D. and Segura, J. (2016). A New Type of Sharp Bounds for Ratios of Modified Bessel Functions. Journal of Mathematical Analysis and Applications, 443(2).

[101] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic Routing Between Capsules. In Advances in Neural Information Processing Systems 30.

[102] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-Learning with Memory-Augmented Neural Networks. In International Conference on Machine Learning.

[103] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. In IEEE Conference on Computer Vision and Pattern Recognition.

[104] Scott, T. R., Gallagher, A. C., and Mozer, M. C. (2021). von Mises–Fisher Loss: An Exploration of Embedding Geometries for Supervised Learning. In IEEE/CVF International Conference on Computer Vision.

[105] Scott, T. R., Ridgeway, K., and Mozer, M. C. (2018). Adapted Deep Embeddings: A Synthesis of Methods for k-Shot Inductive Transfer Learning. In Advances in Neural Information Processing Systems 31.

[106] Scott, T. R., Ridgeway, K., and Mozer, M. C. (2019). Stochastic Prototype Embeddings. In ICML Workshop on Uncertainty and Robustness in Deep Learning.

[107] Sculley, D. (2010). Web-Scale k-Means Clustering. In International Conference on World Wide Web.

[108] Shen, S., Li, W., Zhu, Z., Huang, G., Du, D., Lu, J., and Zhou, J. (2021). Structure-Aware Face Clustering on a Large-Scale Graph with 10^7 Nodes. In IEEE Conference on Computer Vision and Pattern Recognition.

[109] Shi, Y. and Jain, A. K. (2019). Probabilistic Face Embeddings. In IEEE/CVF International Conference on Computer Vision.

[110] Shi, Y., Otto, C., and Jain, A. K. (2018). Face Clustering: Representation and Pairwise Constraints. IEEE Transactions on Information Forensics and Security.

[111] Sibson, R. (1973). SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method. The Computer Journal.

[112] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical Networks for Few-Shot Learning. In Advances in Neural Information Processing Systems 30.

[113] Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. Y. (2013). Zero-Shot Learning Through Cross-Modal Transfer. In Advances in Neural Information Processing Systems 26.

[114] Sohn, K. (2016). Improved Deep Metric Learning with Multi-Class N-Pair Loss Objective. In Advances in Neural Information Processing Systems 29.

[115] Song, H. O., Jegelka, S., Rathod, V., and Murphy, K. (2017). Deep Metric Learning via Facility Location. In IEEE Conference on Computer Vision and Pattern Recognition.

[116] Song, H. O., Xiang, Y., Jegelka, S., and Savarese, S. (2016). Deep Metric Learning via Lifted Structured Feature Embedding. In IEEE Conference on Computer Vision and Pattern Recognition.

[117] Straub, J., Campbell, T., How, J. P., and III, J. W. F. (2015). Small-Variance Nonparametric Clustering on the Hypersphere. In IEEE Conference on Computer Vision and Pattern Recognition.

[118] Sun, Y., Cheng, C., Zhang, Y., Zhang, C., Zheng, L., Wang, Z., and Wei, Y. (2020). Circle Loss: A Unified Perspective of Pair Similarity Optimization. In IEEE Conference on Computer Vision and Pattern Recognition.

[119] Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to Compare: Relation Network for Few-Shot Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[120] Teh, E. W., DeVries, T., and Taylor, G. W. (2020). ProxyNCA++: Revisiting and Revitalizing Proxy Neighborhood Component Analysis. In European Conference on Computer Vision.

[121] Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J. B., and Isola, P. (2020). Rethinking Few-Shot Image Classification: A Good Embedding is All You Need? In European Conference on Computer Vision.

[122] Triantafillou, E., Zemel, R., and Urtasun, R. (2017). Few-Shot Learning Through an Information Retrieval Lens. In Advances in Neural Information Processing Systems 30.

[123] Ustinova, E. and Lempitsky, V. (2016). Learning Deep Embeddings with Histogram Loss. In Advances in Neural Information Processing Systems 29.

[124] van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation Learning with Contrastive Predictive Coding. arXiv e-prints 1807.03748 cs.LG.

[125] Vilnis, L. and McCallum, A. (2018). Word Representations via Gaussian Embedding. In International Conference on Learning Representations.

[126] Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching Networks for One Shot Learning. In Advances in Neural Information Processing Systems 29.

[127] Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. Technical report, California Institute of Technology.

[128] Wang, F., Cheng, J., Liu, W., and Liu, H. (2018a). Additive Margin Softmax for Face Verification. IEEE Signal Processing Letters.

[129] Wang, F., Xiang, X., Cheng, J., and Yuille, A. L. (2017a). NormFace: L2 Hypersphere Embedding for Face Verification. In ACM Multimedia Conference.

[130] Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., and Liu, W. (2018b). CosFace: Large Margin Cosine Loss for Deep Face Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[131] Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., and Liu, W. (2018c). CosFace: Large Margin Cosine Loss for Deep Face Recognition. In IEEE Conference on Computer Vision and Pattern Recognition.

[132] Wang, J., Zhou, F., Wen, S., Liu, X., and Lin, Y. (2017b). Deep Metric Learning with Angular Loss. In IEEE/CVF International Conference on Computer Vision.

[133] Wang, X., Han, X., Huang, W., Dong, D., and Scott, M. R. (2019a). Multi-Similarity Loss with General Pair Weighting for Deep Metric Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[134] Wang, X., Hua, Y., Kodirov, E., Hu, G., Garnier, R., and Robertson, N. M. (2019b). Ranked List Loss for Deep Metric Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[135] Wang, Z., Zheng, L., Li, Y., and Wang, S. (2019c). Linkage Based Face Clustering via Graph Convolution Network. In IEEE Conference on Computer Vision and Pattern Recognition.

[136] Weinberger, K. Q. and Saul, L. K. (2009). Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of Machine Learning Research, 10.

[137] Wilber, M. J., Kwak, I. S., and Belongie, S. (2014). Cost-Effective HITs for Relative Similarity Comparisons. In AAAI Conference on Human Computation and Crowdsourcing.

[138] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. In Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics.

[139] Wolpert, D. H. and Macready, W. G. (1997). No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation.

[140] Wu, C.-Y., Manmatha, R., Smola, A. J., and Krähenbühl, P. (2017). Sampling Matters in Deep Embedding Learning. In IEEE/CVF International Conference on Computer Vision.

[141] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv e-prints 1708.07747 cs.LG.

[142] Yang, L., Chen, D., Zhan, X., Zhao, R., Loy, C. C., and Lin, D. (2020). Learning to Cluster Faces via Confidence and Connectivity Estimation. In IEEE Conference on Computer Vision and Pattern Recognition.

[143] Yang, L., Zhan, X., Chen, D., Yan, J., Loy, C. C., and Lin, D. (2019). Learning to Cluster Faces on an Affinity Graph. In IEEE Conference on Computer Vision and Pattern Recognition.

[144] Ye, J., Peng, X., Sun, B., Wang, K., Sun, X., Li, H., and Wu, H. (2021). Learning to Cluster Faces via Transformer. arXiv e-prints 2104.11502 cs.CV.

[145] Yi, D., Lei, Z., and Li, S. Z. (2014). Deep Metric Learning for Practical Person Re-Identification. arXiv e-prints 1407.4979 cs.CV.

[146] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How Transferable are Features in Deep Neural Networks? In Advances in Neural Information Processing Systems 27.

[147] Yuan, T., Deng, W., Tang, J., Tang, Y., and Chen, B. (2019). Signal-to-Noise Ratio: A Robust Distance Metric for Deep Metric Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[148] Yuan, Y., Chen, W., Yang, Y., and Wang, Z. (2020). In Defense of the Triplet Loss Again: Learning Robust Person Re-Identification with Fast Approximated Triplet Loss and Label Distillation. arXiv e-prints 1912.07863 cs.CV.

[149] Zhai, A. and Wu, H. Y. (2019). Classification is a Strong Baseline for Deep Metric Learning. In British Machine Vision Conference.

[150] Zhan, X., Liu, Z., Yan, J., Lin, D., and Loy, C. C. (2018). Consensus-Driven Propagation in Massive Unlabeled Data for Face Recognition. In European Conference on Computer Vision.

[151] Zhang, L., Xiang, T., and Gong, S. (2017). Learning a Deep Embedding Model for Zero-Shot Learning. In IEEE Conference on Computer Vision and Pattern Recognition.

[152] Zhe, X., Chen, S., and Yan, H. (2018). Directional Statistics-Based Deep Metric Learning for Image Classification and Retrieval. arXiv e-prints 1802.09662 cs.CV.

[153] Zhou, B., Andonian, A., Oliva, A., and Torralba, A. (2018). Temporal Relational Reasoning in Videos. In European Conference on Computer Vision.

[154] Zhu, C., Wen, F., and Sun, J. (2011). A Rank-Order Distance Based Clustering Algorithm for Face Tagging. In IEEE Conference on Computer Vision and Pattern Recognition.

[155] Zhu, L. and Yang, Y. (2018). Compound Memory Networks for Few-Shot Video Classification. In European Conference on Computer Vision.

Appendix A

Stochastic Prototype Embeddings

A.1 SPE Variants

We assumed only diagonal covariance matrices for SPE. Switching to a full covariance would require matrix inversion, which is ordinarily infeasible, but because one purpose of deep visual representations is visualization and interpretation, there may be interesting cases involving 2D embeddings where the cost of inversion is trivial. However, using a diagonal covariance matrix causes class-discriminating features to be aligned with the axes of the latent space, as we argued in Chapter 3, and this alignment is a virtue for interpretation.

A.2 Corruption Procedure

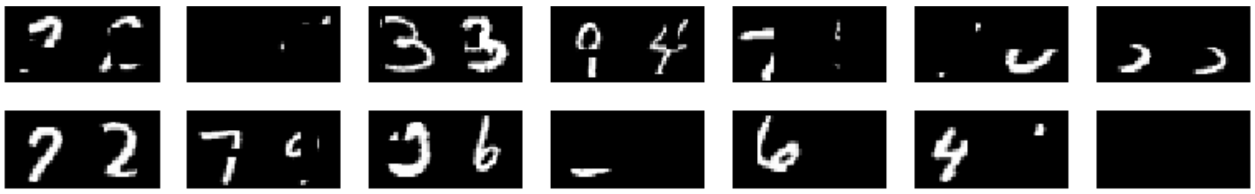


Figure A.2.1: Examples of occluded 2-digit sequences. Occlusion is based on random rectangles that black out portions of each digit.

The algorithm for applying corruption was identical to the scheme used in Oh et al. [88]. A random rectangular-sized occlusion of black pixels was determined by first sampling a patch width, L_x, and patch height, L_y, from a uniform distribution, $L_x, L_y \sim \mathcal{U}(0, 28)$, and then sampling the top-left corner coordinates, $TL_x \sim \mathcal{U}(0, 28-L_x)$, $TL_y \sim \mathcal{U}(0, 28-L_y)$. This resulted in an occlusion of area $L_x \times L_y$. Note that if $L_x = 0$ or $L_y = 0$, the image was left unoccluded. Figure A.2.1 shows examples of occluded 2-digit images, where each digit was occluded independently.

For Omniglot, we only trained and validated on corrupted imagery if the test set contained a corrupted support or corrupted query set. When testing on clean support and clean query, the training and validation sets were left unoccluded. When testing on corrupted imagery, the training and validation sets corrupted each character independently with a probability of 0.2.

The training and validation sets for N-digit MNIST corrupted each digit of each image independently with a probability of 0.2, regardless of test imagery. This matched Oh et al. [88].

During testing on both datasets, we considered both clean and corrupt support sets, as well as clean and corrupt query sets. A clean set was one in which all digits/characters were unoccluded. A corrupt set occluded each digit/character in each image according to the procedure described above.

Appendix B

von Mises–Fisher Loss: An Exploration of Embedding Geometries for Supervised Learning

B.1 Derivation of the von Mises–Fisher Loss

Let $\boldsymbol{z}$ and $\{\boldsymbol{w}\}_{1:Y}$ be von Mises–Fisher random variables for the embedding and class weight vectors, respectively, and Y be the number of training classes. In addition, let $\tilde{\boldsymbol{w}}_j$ be the learnable weight vector for class j that is used to parameterize $\boldsymbol{w}_j$. Below is a derivation of the von Mises–Fisher loss:

$$\begin{aligned}
&\mathcal{L}(y, \boldsymbol{z};\ \boldsymbol{w}_{1:Y}) = \mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}[-\log p(y|\boldsymbol{z}, \boldsymbol{w}_{1:Y})] \\
&\quad = \mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}\left[-\log \frac{\exp\left(\beta \cos\theta_y\right)}{\sum_j \exp(\beta\cos\theta_j)}\right] \\
&\quad = \mathbb{E}_{\boldsymbol{z},\boldsymbol{w}_{1:Y}}\left[\log\left(\sum_j \exp\left(\beta \boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{z}\right)\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] && \text{(Independence of } \boldsymbol{z} \text{ and } \boldsymbol{w}_y\text{)} \\
&\quad \leq \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \mathbb{E}_{\boldsymbol{w}_j}\left[\exp\left(\beta\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{z}\right)\right]\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] && \text{(Jensen's inequality with respect to } \{\boldsymbol{w}_j\}\text{)} \\
&\quad = \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \int_{\boldsymbol{w}_j} C_d(\kappa_{\boldsymbol{w}_j}) \exp(\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j}^{\mathrm{T}}\boldsymbol{w}_j)\exp(\beta\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{z})\,\mathrm{d}\boldsymbol{w}_j\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \\
&\quad = \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j C_d(\kappa_{\boldsymbol{w}_j})\int_{\boldsymbol{w}_j} \exp((\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z})^{\mathrm{T}}\boldsymbol{w}_j)\,\mathrm{d}\boldsymbol{w}_j\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \\
&\quad = \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j C_d(\kappa_{\boldsymbol{w}_j})\int_{\boldsymbol{w}_j} \exp\left(\frac{\left\|\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}\right\|}{\left\|\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}\right\|}(\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z})^{\mathrm{T}}\boldsymbol{w}_j\right)\,\mathrm{d}\boldsymbol{w}_j\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \\
&\quad = \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \frac{C_d(\kappa_{\boldsymbol{w}_j})}{C_d\left(\left\|\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}\right\|\right)}\int_{\boldsymbol{w}_j} \mathrm{vMF}\left(\boldsymbol{w}_j;\ \boldsymbol{\mu} = \frac{\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}}{\left\|\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}\right\|}, \kappa = \left\|\kappa_{\boldsymbol{w}_j}\boldsymbol{\mu}_{\boldsymbol{w}_j} + \beta\boldsymbol{z}\right\|\right)\,\mathrm{d}\boldsymbol{w}_j\right)\right] \\
&\qquad - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}]
\end{aligned}$$

$$= \mathbb{E}_{\boldsymbol{z}} \left[\log \left(\sum_j \frac{C_d(\|\tilde{\boldsymbol{w}}_j\|)}{C_d\left(\|\tilde{\boldsymbol{w}}_j + \beta \boldsymbol{z}\|\right)} \right) \right] - \beta \mathbb{E}[\boldsymbol{w}_y] \mathbb{E}[\boldsymbol{z}].$$

B.2 Hyperparameters

Models were optimized with SGD and either standard momentum or Nesterov momentum. Validation accuracy was monitored throughout training and if it did not improve for 15 epochs, the learning rate was cut in half. If the validation accuracy did not improve for 35 epochs, model training was stopped early. Model parameters were saved for the epoch resulting in the highest validation accuracy. For fixed-set datasets, we found ARCFACE was unstable during training caused by a degeneracy in the objective where embeddings were pushed to the opposite side of the hypersphere from the class weight vectors. To fix the degeneracy, we introduced an additional hyperparameter for the number of epochs at the beginning of training where the margin, m, was set to zero. Once the network had trained for several epochs without the margin, we set the margin to a value greater than zero and trained ARCFACE normally. Here is a list of all hyperparameters with abbreviations:

- Learning rate (LR)
- Temperature learning rate (TLR)
- Momentum (MOM)
- Nesterov momentum (NMOM)
- ℓ_2 weight decay factor (WD)
- Margin for ARCFACE (m)
- Number of initial epochs with $m = 0$ ($m = 0$ Epochs)
- Curvature for HYPERBOLIC (c)
- λ for vMF (λ)
- Initial value of τ (Init τ)

Hyperparameters were selected using a grid search. The tables below specify the best hyperparameter values found for each of the losses. If a hyperparameter is not applicable for a loss, we mark the value with “–”.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.1	–	0.9	False	0.0	–	–	–	–	–
HYPERBOLIC	0.05	–	0.99	True	0.0	–	–	1×10^{-5}	–	–
COSINE	0.5	0.001	0.9	True	0.0	–	–	–	–	0.0
ARCFACE	0.05	0.001	0.9	False	0.0	0.5	20	–	–	0.0
vMF	1.0	0.001	0.99	True	0.0	–	–	–	0.4	0.0

Table B.1: Hyperparameter values for MNIST.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.01	–	0.99	False	0.0	–	–	–	–	–
HYPERBOLIC	0.1	–	0.9	True	0.0	–	–	1×10^{-5}	–	–
COSINE	0.5	0.001	0.9	True	0.0	–	–	–	–	0.0
ARCFACE	0.01	0.001	0.99	True	0.0	0.5	20	–	–	0.0
vMF	0.05	0.001	0.99	False	0.0	–	–	–	0.4	0.0

Table B.2: Hyperparameter values for FashionMNIST.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.05	–	0.99	True	1×10^{-4}	–	–	–	–	–
HYPERBOLIC	0.01	–	0.99	True	5×10^{-4}	–	–	1×10^{-5}	–	–
COSINE	0.5	0.001	0.8	False	5×10^{-4}	–	–	–	–	0.0
ARCFACE	0.1	0.001	0.9	True	5×10^{-4}	0.4	60	–	–	0.0
vMF	0.5	0.001	0.9	False	1×10^{-4}	–	–	–	0.4	0.0

Table B.3: Hyperparameter values for CIFAR10.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.1	–	0.9	True	5×10^{-4}	–	–	–	–	–
HYPERBOLIC	0.01	–	0.99	True	5×10^{-4}	–	–	1×10^{-5}	–	–
COSINE	0.5	0.0001	0.8	False	5×10^{-4}	–	–	–	–	0.0
ARCFACE	0.01	0.0001	0.99	True	5×10^{-4}	0.3	100	–	–	0.0
vMF	0.1	0.01	0.99	True	1×10^{-4}	–	–	–	0.4	0.0

Table B.4: Hyperparameter values for CIFAR100.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.005	–	0.8	True	5×10^{-5}	–	–	–	–	–
HYPERBOLIC	0.01	–	0.9	True	5×10^{-4}	–	–	1×10^{-5}	–	–
COSINE	0.005	0.001	0.9	True	5×10^{-4}	–	–	–	–	3.466
ARCFACE	0.0001	0.0001	0.9	False	1×10^{-4}	0.3	0	–	–	3.466
vMF	0.0001	0.01	0.9	False	1×10^{-4}	–	–	–	0.7	0.0

Table B.5: Hyperparameter values for Cars196.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.001	–	0.8	True	5×10^{-4}	–	–	–	–	–
HYPERBOLIC	0.005	–	0.9	False	5×10^{-4}	–	–	1×10^{-5}	–	–
COSINE	0.001	0.0001	0.9	False	5×10^{-4}	–	–	–	–	2.773
ARCFACE	0.001	0.0001	0.8	True	5×10^{-4}	0.3	0	–	–	2.773
vMF	0.001	0.01	0.9	False	1×10^{-4}	–	–	–	0.7	2.773

Table B.6: Hyperparameter values for CUB200-2011.

	LR	TLR	MOM	NMOM	WD	m	$m = 0$ Epochs	c	λ	Init τ
STANDARD	0.005	–	0.8	True	5×10^{-4}	–	–	–	–	–
HYPERBOLIC	0.0005	–	0.99	False	1×10^{-4}	–	–	1×10^{-5}	–	–
COSINE	0.001	0.0001	0.99	True	1×10^{-4}	–	–	–	–	0.0
ARCFACE	0.005	0.0001	0.8	True	5×10^{-4}	0.3	0	–	–	2.773
vMF	0.0001	0.001	0.8	True	5×10^{-4}	–	–	–	0.7	2.773

Table B.7: Hyperparameter values for SOP.

B.3 Cars196 Test Images

Below we show paired grids of Cars196 test images for each of the losses where the left grid contains images corresponding to the smallest $\|\boldsymbol{z}\|$ and the right grid corresponding to the largest $\|\boldsymbol{z}\|$. The provided figures are analogous to Figure 4.3. Note that for vMF, $\|\tilde{\boldsymbol{z}}\| = \kappa_{\boldsymbol{z}}$.

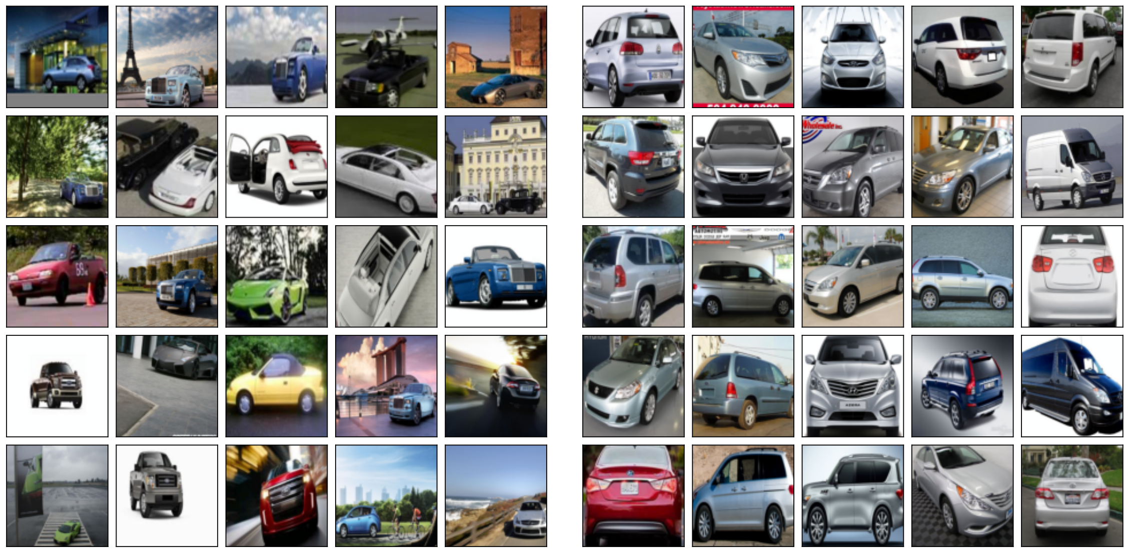


Figure B.3.1: STANDARD embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

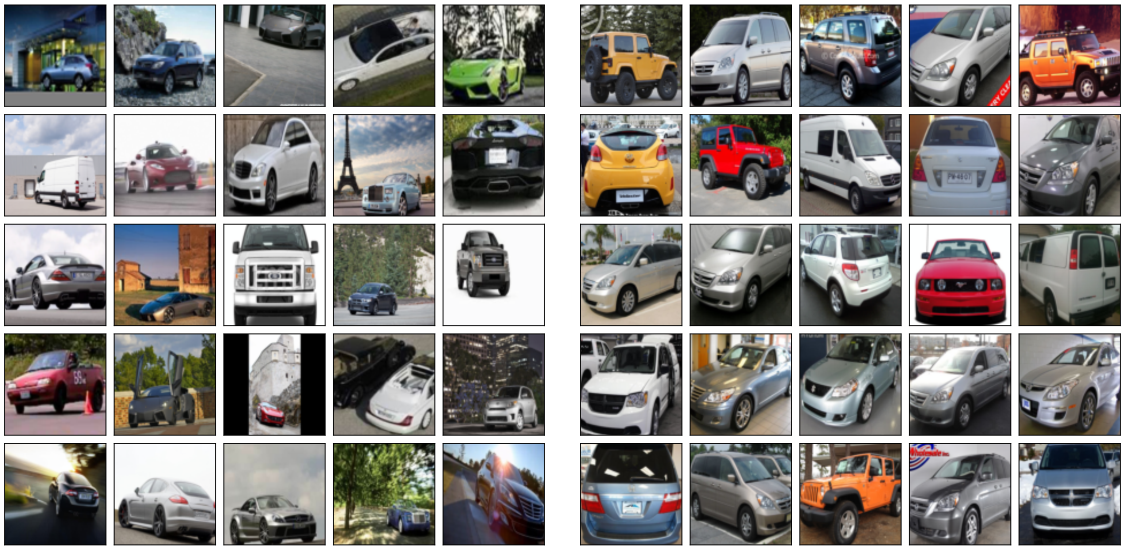


Figure B.3.2: hyperbolic embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

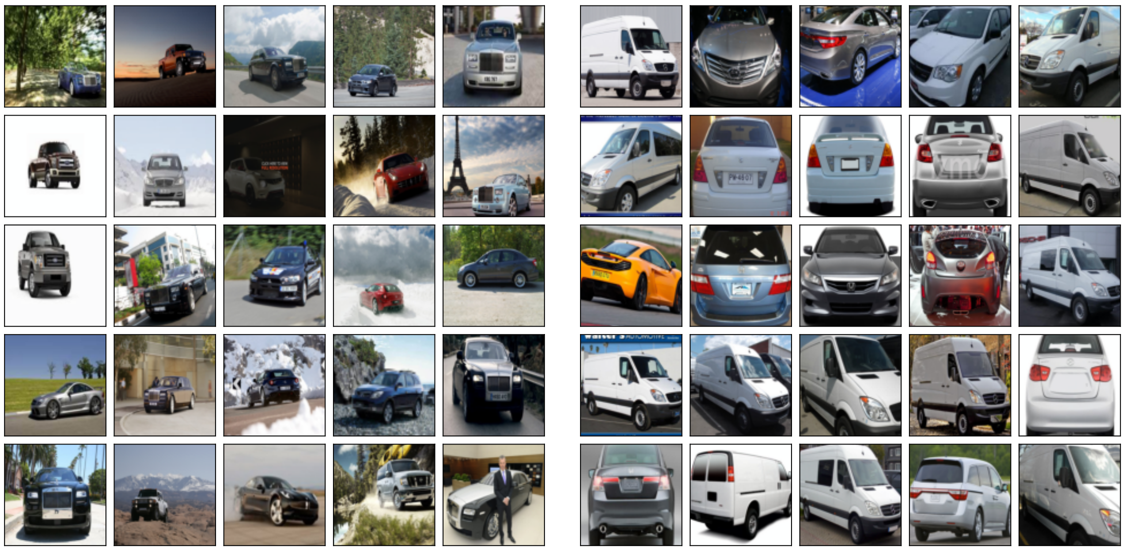


Figure B.3.3: cosine embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

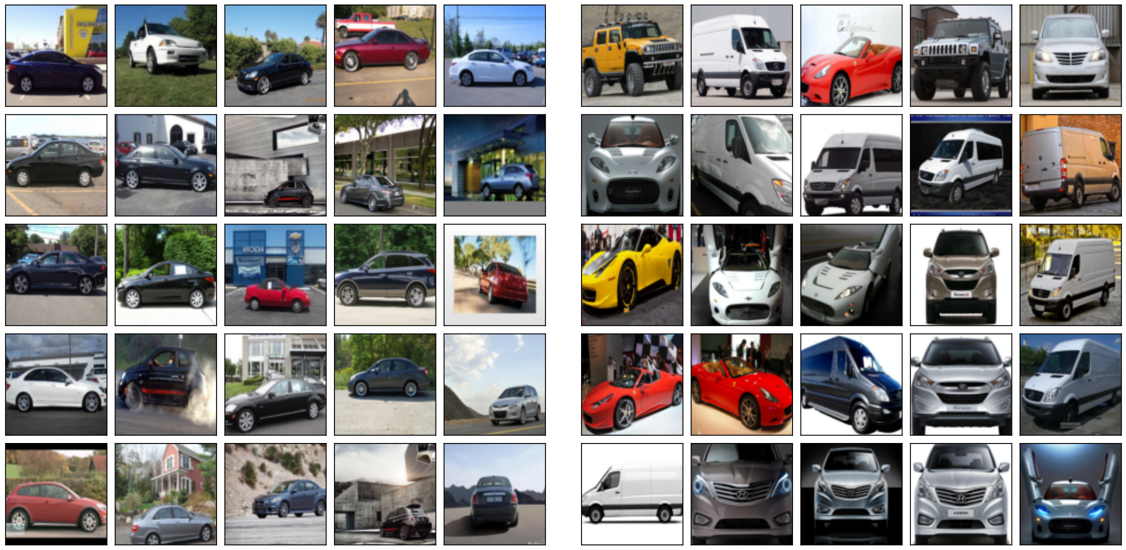


Figure B.3.4: ARCFACE embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

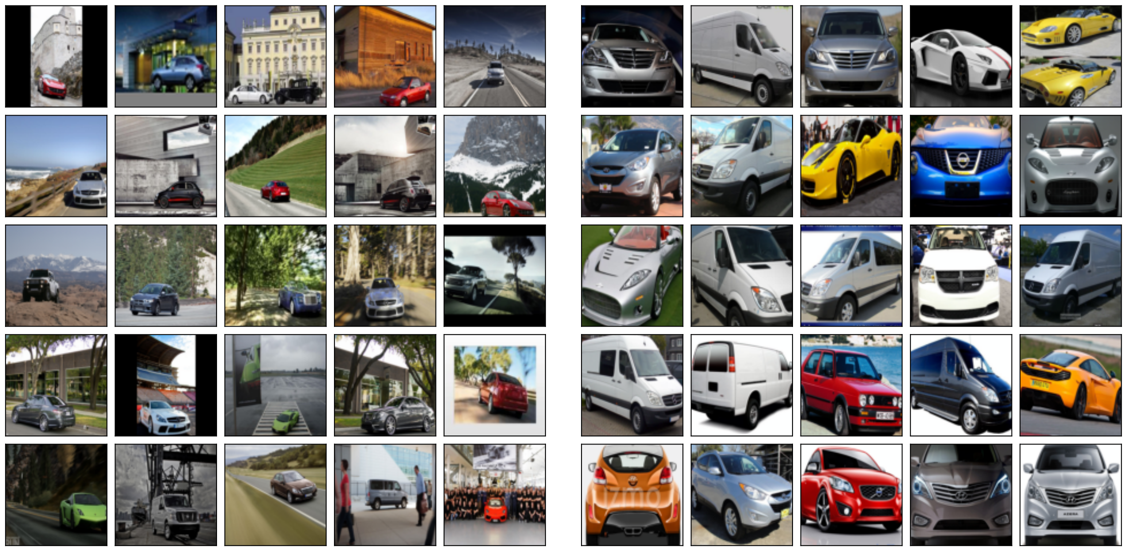


Figure B.3.5: vMF embeddings with the (left) smallest $\kappa_{\boldsymbol{z}}$ and (right) largest $\kappa_{\boldsymbol{z}}$.

Appendix C

Unifying Few- and Zero-Shot Egocentric Action Recognition

C.1 Tabular Results for CM-FSC and FSC

Table A.1: Tabulated CM-FSC classification-accuracy results for the word embedding (WE) and joint embedding (JE). These results match those presented in Figure 5.3. The three values grouped together in each row for a given class-type (All Test, HoV, HoN) correspond to the performance on splits 1, 2, and 3, respectively.

CM-FSC: 1-shot, 5-class

		All Test			HoV			HoN		
WE	$\lambda = 0$	28.1	28.5	28.2	24.8	24.6	21.8	27.2	25.4	29.2
	Histogram	31.2	33.7	31.7	26.1	25.7	21.3	30.9	34.5	36.1
	MultiSim	36.3	37.5	35.7	25.5	28.0	23.9	38.2	40.4	42.3
JE	Histogram	40.9	40.7	39.1	33.2	31.7	33.6	43.0	46.7	44.5
	MultiSim	41.3	38.9	41.1	32.9	30.3	36.3	41.1	44.3	43.3

CM-FSC: 1-shot, 20-class

		All Test			HoV			HoN		
WE	$\lambda = 0$	9.0	7.7	9.4	6.2	5.4	6.2	9.1	6.5	11.4
	Histogram	11.1	11.4	11.5	7.4	5.8	5.8	11.8	12.8	14.6
	MultiSim	14.0	14.9	15.7	6.5	7.2	5.7	16.2	18.2	19.3
JE	Histogram	16.7	17.2	17.1	11.4	8.7	11.6	17.7	19.9	18.1
	MultiSim	16.7	16.8	17.6	10.5	9.4	13.1	17.7	19.4	19.0

Table A.2: Tabulated FSC classification-accuracy results for the video embedding (VE), word embedding (WE), and joint embedding (JE). These results match those presented in Figure 5.3. The three values grouped together in each row for a given class-type (All Test, HoV, HoN) correspond to the performance on splits 1, 2, and 3, respectively.

FSC: 1-shot, 5-class

		All Test			HoV			HoN		
VE	Histogram	62.2	62.6	62.8	71.2	68.2	69.1	57.0	58.2	55.4
	MultiSim	69.9	71.1	73.3	80.0	78.8	78.0	63.6	63.3	65.5
WE	$\lambda = 0$	33.6	40.6	37.3	36.2	45.5	37.6	29.9	39.5	35.3
	Histogram	60.2	59.9	62.8	69.2	67.6	67.1	55.4	55.7	55.6
	MultiSim	69.1	71.4	74.1	80.9	78.9	79.2	62.2	63.6	66.2
JE	Histogram	62.5	65.0	66.6	71.8	69.7	70.5	57.4	60.8	58.0
	MultiSim	65.4	70.8	72.0	76.7	78.2	75.9	60.3	62.8	63.5

FSC: 5-shot, 5-class

		All Test			HoV			HoN		
VE	Histogram	71.5	73.1	73.2	79.4	78.2	78.3	65.0	64.5	64.6
	MultiSim	77.8	78.5	82.3	87.5	87.5	86.6	70.6	69.1	76.1
WE	$\lambda = 0$	38.0	45.4	44.9	43.7	52.0	45.2	33.2	39.4	41.9
	Histogram	69.6	70.7	72.6	78.0	76.1	79.3	62.2	60.1	64.1
	MultiSim	78.0	77.6	82.9	88.0	86.8	87.3	70.7	68.4	76.6
JE	Histogram	72.3	74.2	76.0	80.6	80.0	82.2	65.6	65.4	68.0
	MultiSim	75.2	77.8	81.2	85.0	87.0	85.7	67.8	69.3	75.3

FSC: 1-shot, 20-class

		All Test			HoV			HoN		
VE	Histogram	41.3	38.9	41.6	48.0	43.3	47.8	31.9	32.2	33.2
	MultiSim	53.0	50.5	55.5	62.4	59.8	62.7	42.0	40.3	45.9
WE	$\lambda = 0$	16.7	22.9	20.2	18.0	27.2	21.3	13.1	19.0	17.3
	Histogram	39.4	36.5	41.2	45.5	41.5	46.8	30.5	29.9	33.4
	MultiSim	53.3	50.0	57.3	62.4	60.0	64.1	41.1	40.0	47.1
JE	Histogram	43.4	41.3	45.3	50.0	46.4	51.8	33.5	34.1	37.2
	MultiSim	50.5	49.8	54.7	58.1	60.3	60.7	39.7	40.3	44.9

FSC: 5-shot, 20-class

		All Test			HoV			HoN		
VE	Histogram	48.8	46.6	49.8	57.0	55.4	53.1	39.4	35.7	41.5
	MultiSim	59.4	56.8	64.8	70.9	72.5	69.2	47.9	43.3	54.9
WE	$\lambda = 0$	19.6	25.5	26.1	23.9	35.6	25.3	16.5	19.8	22.5
	Histogram	46.6	43.2	49.9	53.2	52.7	53.9	36.3	31.4	40.8
	MultiSim	60.1	55.7	65.6	71.4	72.2	70.3	48.2	42.0	54.8
JE	Histogram	50.8	49.0	55.1	59.3	60.0	60.4	40.7	37.3	46.1
	MultiSim	57.7	56.3	64.4	67.5	73.1	68.9	46.9	43.4	55.2

Appendix D

An Empirical Study on Clustering Pretrained Embeddings: Is Deep Strictly Better?

D.1 Compute Resources

Each backbone was trained using a single NVIDIA Tesla T4, 16 CPUs, and 64 GB of RAM on Google Cloud Platform. Code was implemented using PyTorch v1.6.0 and Python 3.8.2 on Ubuntu 18.04.

For clustering, all methods had access to 16 CPUs, 96 GB of RAM, and a single NVIDIA Tesla T4, when appropriate. Code was implemented with PyTorch v1.6.0, scikit-learn v0.24.2, FAISS v1.7.1, and Python 3.8.2 on Ubuntu 18.04. Additionally, we note that both HAC and spectral clustering could be made faster by computing the nearest-neighbor subgraph on GPU instead of CPU.

D.2 Cars196 and SOP Tabulated Results

The tables below contains clustering results for Cars196 and SOP. The F_P and F_B values are used to compute the harmonic means for ℓ_2-normalized embeddings presented in Figure 6.1.

Cars196							
	Adj. Rand Index	NMI	AMI	F_P	F_B	Time to Cluster (s)	Number of Clusters
DBSCAN	0.10	0.63	0.28	0.12	0.35	3	$2,425$
MeanShift	0.18	0.65	0.42	0.20	0.38	177	$1,407$
ARO	0.28	0.69	0.49	0.29	0.36	3	$1,415$
DP k-Means	0.37	0.69	0.61	0.38	0.41	10	403
DP vMF k-Means	0.36	0.67	0.61	0.38	0.40	12	155
k-Means	0.40	0.65	0.62	0.41	0.42	12	50
Spherical k-Means	0.40	0.65	0.62	0.41	0.42	12	50
GMM	0.40	0.66	0.62	0.41	0.43	62	50
vMF-MM	0.37	0.65	0.61	0.38	0.40	267	60
HAC	0.42	0.68	0.65	0.44	0.45	4	55
CDP	0.24	0.68	0.46	0.26	0.36	2	$1,686$
Spectral	**0.47**	0.70	**0.68**	**0.48**	**0.50**	99	50
GCN-VE	0.20	0.65	0.49	0.22	0.37	1140, 16	905
STAR-FC	0.36	**0.71**	0.53	0.37	0.40	341, 1	$1,624$

SOP							
	Adj. Rand Index	NMI	AMI	F_P	F_B	Time to Cluster (m)	Number of Clusters
DBSCAN	0.29	0.93	0.49	0.29	0.58	0.8	$17,801$
MeanShift	0.35	0.93	0.48	0.35	0.57	54	$18,096$
ARO	0.52	**0.94**	0.64	0.52	0.66	0.2	$9,915$
DP k-Means	0.45	0.93	0.59	0.45	0.63	13	$12,261$
DP vMF k-Means	0.45	0.93	0.59	0.45	0.63	3	$11,282$
k-Means	0.46	0.92	0.58	0.46	0.61	210	$7,750$
Spherical k-Means	0.45	0.93	0.59	0.45	0.62	242	$8,250$
HAC	0.52	0.93	0.64	0.52	0.65	4	$7,250$
CDP	**0.56**	**0.94**	**0.67**	**0.56**	**0.69**	0.1	$11,480$
GCN-VE	0.47	0.93	0.62	0.47	0.65	20, 13s	$8,808$
STAR-FC	0.50	0.93	0.61	0.50	0.65	8, 2s	$10,393$

Table B.1: Clustering results for Cars196 and SOP. NMI, AMI, F_P, and F_B represent normalized mutual information, adjusted mutual information, Pairwise F-score, and BCubed F-score, respectively. The time to cluster for Cars196 and SOP are measured in seconds and minutes, respectively. The time to cluster for GCN-VE and STAR-FC contains the train time and test time separated by a comma. Boldface indicates the highest value for a metric.

D.3 Comparison of k-Means Initialization Strategies

There are two common strategies for initializing clusters in k-means and spherical k-means. The simplest is to initialize clusters by randomly selecting embeddings from the dataset. The alternative is to use k-means++ [4] which tries to pick clusters that are generally distant from one another, leading to faster convergence and better expected performance. We find that k-means++ indeed is a better initialization strategy, however, for large k it increases the runtime drastically. The increased runtime is caused by successively computing the nearest cluster-center for each embedding, and thus scales poorly as k grows. For our experiments, we use the k-means++ implementation from scikit-learn and admit that performance could be improved by running nearest-neighbor computation on accelerated hardware or by using approximate nearest-neighbor methods.

Based on our results summarized in the tables below, we find that the benefits of k-means++ initialization are not always significant (e.g., Table C.3) and may not be worth the increased runtime, which can be upwards of 900x slower (e.g., SOP results in Table C.2) than a random initialization for large k.

Cars196				
	Initialization	F_P	F_B	Time to Cluster (s)
k-Means	Random	**0.41**	0.42	1
Spherical k-Means	Random	**0.41**	**0.43**	1
k-Means	k-Means++	**0.41**	0.42	12
Spherical k-Means	k-Means++	**0.41**	0.42	12

SOP				
	Initialization	F_P	F_B	Time to Cluster (s)
k-Means	Random	0.36	0.53	14
Spherical k-Means	Random	0.36	0.53	31
k-Means	k-Means++	**0.46**	0.61	$12,591$
Spherical k-Means	k-Means++	0.45	**0.62**	$14,491$

Table C.2: Comparison of k-means initialization strategies on Cars196 and SOP. F_P and F_B represent Pairwise F-score and BCubed F-score, respectively. Boldface indicates the highest value for a metric.

	Initialization	Test #1		Test #2		Test #3		Test #4		Test #5	
		F_P	F_B	F_P	F_B	F_P	F_B	F_P	F_B	F_P	F_B
Minibatch *k*-Means	Random	**0.64**	0.65	**0.60**	**0.61**	**0.57**	**0.58**	**0.55**	**0.56**	0.53	0.54
Minibatch *k*-Means	*k*-Means++	**0.64**	**0.66**	**0.60**	**0.61**	**0.57**	**0.58**	**0.55**	**0.56**	**0.54**	**0.55**

	Initialization	Test #1	Test #2	Test #3	Test #4	Test #5
Minibatch *k*-Means	Random	3m	7m	12m	18m	23m
Minibatch *k*-Means	*k*-Means++	9m	41m	78m	147m	233m

Table C.3: Comparison of *k*-means initialization strategies on Dataset 3. For each of the five Dataset 3 test splits, the top table contains Pairwise (F_P) and BCubed (F_B) F-scores, and the bottom table contains the time to cluster in minutes. Boldface indicates the highest value for a metric.

D.4 Dataset 3 Degradation Study

The table below contains clustering results for GCN-VE and HAC on test split #1 of Dataset 3 where we reduce the dimensionality of the input embedding. We reduce the dimensionality as a method for degrading the embedding discriminability, measured via Recall@1 in Table 6.6. These results are used to compute the harmonic means presented in Figure 6.2.

Dataset 3 Test Split #1								
	32D		64D		128D		256D (original)	
	F_P	F_B	F_P	F_B	F_P	F_B	F_P	F_B
HAC	**0.27**	0.27	**0.53**	0.54	**0.68**	**0.70**	0.74	0.75
GCN-VE	0.26	**0.29**	**0.53**	**0.55**	**0.68**	0.69	**0.78**	**0.79**

Table D.4: Pairwise (F_P) and BCubed (F_B) F-scores for HAC and GCN-VE on test split #1 of Dataset 3 as the embedding dimensionality increases. Boldface indicates the highest value for a metric.

D.5 Experimental Details

D.5.1 Backbone

For Cars196 and SOP, the experimental details mimic Chapter 4 unless explicitly noted. For completeness, we recount the details below.

D.5.1.1 Architecture

The backbone network architecture is a ResNet50 [46]. The network is initialized using weights pretrained on ImageNet and all batch-normalization parameters are frozen. We remove the head of the architecture and add two fully-connected layers, with no activations, directly following the global-average-pooling layer. The first fully-connected layer maps from 2048 units to 256 units, and the second fully-connected layer maps from 256 units to Y units, where Y is the number of classes in the backbone-training split of the dataset. The embedding dimensionality is thus 256D. For Cars196 and SOP, $Y = 49$ and $Y = 5658$, respectively.

D.5.1.2 Dataset Augmentation

Cars196. Data augmentation during training includes: (1) resizing images to 256×256, (2) random jittering of brightness, contrast, saturation, and hue with factors in [0.7, 1.3], [0.7, 1.3], [0.7, 1.3], and [-0.1, 0.1], respectively, (3) cropping at a random location with random size between 16% and 100% of the input size, (4) resizing the final cropped images to 224×224, and (5) random horizontal flipping and z-score normalization. Data augmentation during validation and testing includes: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization. The mean and standard deviation for z-score normalization are the same values used for ImageNet. We match Boudiaf et al. [10] and sample batches randomly.

SOP. Data augmentation during training includes: (1) resizing images to 256×256, (2) cropping at a random location with random size between 16% and 100% of the input size with the aspect ratio randomly selected in [0.75, 1.33], (3) resizing the final cropped images to 224×224, and

(4) random horizontal flipping and z-score normalization. Data augmentation during validation and testing includes: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization. The mean and standard deviation for z-score normalization are the same values used for ImageNet. We match Boudiaf et al. [10] and sample batches randomly.

D.5.1.3 Backbone Network Hyperparameters

Models are trained with SGD and either standard or Nesterov momentum. Based on results from Chapter 4, we use cosine softmax cross-entropy as the loss. For Cars196 and SOP, the validation split's Recall@1 is monitored throughout training and if it does not improve for 15 epochs, the learning rate is cut in half. If the validation split's Recall@1 does not improve for 35 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the highest validation Recall@1.

The inverse-temperature, β, is parameterized as $\beta = \exp(\tau)$ where $\tau \in \mathbb{R}$ is an unconstrained network parameter learned automatically via gradient descent. Unlike in Chapter 4, τ is optimized using the same learning rate as all other network parameters.

The remaining hyperparameters for training the backbone network are:

- Learning rate
- ℓ_2 weight decay
- Momentum
- Nesterov momentum
- Initial value of τ

Hyperparameters were optimized with a grid search. The table below contains the hyperparameter values used for the model achieving the best validation Recall@1.

	Learning rate	ℓ_2 weight decay	Momentum	Nesterov momentum	Initial value of τ
Cars196	0.005	5×10^{-5}	0.0	False	0.0
SOP	0.005	5×10^{-5}	0.9	True	-2.773

Table E.5: Backbone hyperparameter values for Cars196 and SOP.

D.5.2 Clustering

For clustering, the inputs are the 256D embeddings from the penultimate output of the trained backbone network. The embeddings are ℓ_2-normalized unless explicitly noted otherwise. To produce the embeddings for clustering, the images were augmented using the testing augmentation, which is: (1) resizing images to 256×256, (2) center cropping to 224×224, and (3) z-score normalization.

D.5.2.1 GCN-VE Details

We match Yang et al. [142] and use a one-layer graph convolutional network (GCN) [59] with mean aggregation [143] and ReLU activation for GCN-V, and a similar four-layer graph convolutional network for GCN-E. GCN-V maps the 512D output of the GCN through a fully-connected layer to 512 units, then through a PReLU activation, followed by a final fully-connected layer to a single output unit. The model is trained with mean-squared-error to match an empirically-estimated confidence based on the density of an embedding's neighborhood. GCN-E has identical structure to GCN-V except that it contains a four-layer GCN, the GCN output is 256D, and the first fully-connected layer has 256 units. GCN-E is trained with softmax cross-entropy to predict the likelihood that a pair of nearby embeddings belong to the same cluster. The GCN layers are initialized using Xavier-uniform for the weights and zero for the biases. The fully-connected layers are initialized using Kaiming-uniform for both weights and biases.

Both GCN-V and GCN-E are trained with SGD and standard momentum. If the validation split's loss does not decrease for 100 epochs, the learning rate is cut in half, and if it does not decrease for 250 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the lowest validation loss.

D.5.2.2 STAR-FC Details

We match Shen et al. [108] and use a one-layer graph convolutional network (GCN) [59] with mean aggregation [143] and ReLU activation. The GCN output is 1024D which is then passed through a fully-connected layer to 512 units, then through a PReLU activation, followed by a final fully-connected layer to a single output unit. The model is trained with softmax cross-entropy to predict the likelihood that a pair of embeddings belong to the same cluster. The GCN layers are initialized using Xavier-uniform for the weights and zero for the biases. The fully-connected layers are initialized using Kaiming-uniform for both weights and biases.

STAR-FC is trained with SGD and standard momentum. If the validation split's loss does not decrease for 15 epochs, the learning rate is cut in half, and if it does not decrease for 35 epochs, model training terminates early. The model parameters are saved for the epoch resulting in the lowest validation loss.

D.5.2.3 Clustering Hyperparameters

For each clustering method, we list the hyperparameters and their values for Cars196 and SOP. All hyperparameters were optimized with a grid search.

k-Means with Random Initialization [78, 107]. The only hyperparameters are k, the number of clusters, and the minibatch size. A minibatch size of -1 indicates the batch size is equal to the full dataset size.

	k	Minibatch size
Cars196	40	-1
SOP	$8,250$	-1

Table E.6: Hyperparameter values for k-means with random initialization.

k-Means with k-Means++ Initialization [4, 107]. The only hyperparameters are k, the number of clusters, and the minibatch size. A minibatch size of -1 indicates the batch size is equal to the full dataset size.

	k	Minibatch size
Cars196	50	-1
SOP	$7,750$	-1

Table E.7: Hyperparameter values for k-means with k-means++ initialization.

Spherical k-Means with Random Initialization [49]. The only hyperparameter is k, the number of clusters. Due to the performance of spherical k-means closely matching that of k-means with ℓ_2-normalized embeddings, and the lack of a readily-available implementation of minibatch spherical k-means, we only ran spherical k-means for Cars196 and SOP, where the minibatch size could be -1.

	k
Cars196	45
SOP	$8,000$

Table E.8: Hyperparameter values for spherical k-means with random initialization.

Spherical k-Means with k-Means++ Initialization [49]. The only hyperparameter is k, the number of clusters. Due to the performance of spherical k-means closely matching that of k-means with ℓ_2-normalized embeddings, and the lack of a readily-available implementation of minibatch spherical k-means, we only ran spherical k-means for Cars196 and SOP, where the minibatch size could be -1.

	k
Cars196	50
SOP	$8,250$

Table E.9: Hyperparameter values for spherical k-means with k-means++ initialization.

Dirichlet-Process (DP) k-Means [64]. The hyperparameters are:

- λ, the cluster penalty
- Initialization strategy, either 'Global Centroid' or 'Random'

- Whether to do an online EM update

	λ	Initialization strategy	Online EM
Cars196	0.9	Global Centroid	True
SOP	0.85	Random	False

Table E.10: Hyperparameter values for DP k-means.

Dirichlet-Process (DP) von Mises–Fisher (vMF) k-Means [117]. The hyperparameters are:

- λ, the cluster penalty
- Initialization strategy, either 'Global Centroid' or 'Random'
- Whether to do an online EM update

	λ	Initialization strategy	Online EM
Cars196	-0.55	Random	True
SOP	-0.38	Random	False

Table E.11: Hyperparameter values for DP vMF k-means.

Gaussian Mixture Model (GMM). The hyperparameters are:

- k, the number of clusters
- Initialization strategy, either 'k-Means' or 'Random'
- Type of covariance, either 'Full' or 'Diagonal'

GMM was run only on Cars196 due to runtime inefficiencies.

	k	Initialization strategy	Covariance type
Cars196	50	k-Means	Diagonal

Table E.12: Hyperparameter values for GMM.

von Mises–Fisher Mixture Model (vMF-MM) [7, 38]. The hyperparameters are:

- k, the number of clusters
- Initialization strategy, one of 'k-Means++' or 'Random', 'Random-Class', or 'Random-Orthonormal'
- Posterior type, either 'Hard' or 'Soft'

vMF-MM was run only on Cars196 due to runtime inefficiencies.

	k	Initialization strategy	Posterior type
Cars196	50	Random-Class	Soft

Table E.13: Hyperparameter values for vMF-MM.

Hierarchical Agglomerative Clustering (HAC) [111]. The hyperparameters are:

- k, the number of clusters
- Affinity metric, one of 'Euclidean' or 'Cosine'
- Linkage criterion, one of 'Ward', 'Complete', 'Average', or 'Single'
- Number of neighbors for computing the nearest-neighbor affinity subgraph

	k	Affinity metric	Linkage criterion	Number of neighbors
Cars196	55	Euclidean	Ward	40
SOP	$7,250$	Euclidean	Ward	2

Table E.14: Hyperparameter values for HAC.

Approximate Rank Order (ARO) [89]. The hyperparameters are:

- Number of neighbors for computing the nearest-neighbors subgraph
- Threshold on rank-order distances
- Distance metric, one of 'Euclidean' or 'Cosine'

	Number of neighbors	Threshold	Distance metric
Cars196	30	0.65	Euclidean
SOP	5	1.0	Euclidean

Table E.15: Hyperparameter values for ARO.

Note that for ℓ_2-normalized embeddings, both Euclidean and cosine distance metrics produce identical results because the rank-order is unchanged.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [30]. The hyperparameters are:

- ϵ, the maximum distance between two embeddings in the same neighborhood
- Minimum number of samples in neighborhood for core point
- Distance metric, either 'Euclidean' or 'Cosine'
- Whether a sparse matrix is used for nearest-neighbor subgraph

	ϵ	Minimum number of samples	Distance metric	Sparse matrix
Cars196	0.25	5	Cosine	False
SOP	0.66	2	Euclidean	False

Table E.16: Hyperparameter values for DBSCAN.

MeanShift [20]. The hyperparameters are:

- Bandwidth for RBF kernel
- Minimum bin frequency
- Whether to cluster all embeddings including orphans

	Bandwidth	Minimum bin frequency	Cluster all embeddings
Cars196	0.75	5	True
SOP	0.65	1	False

Table E.17: Hyperparameter values for MeanShift.

Spectral Clustering [85]. The hyperparameters are:

- k, the number of clusters
- Method for constructing the affinity matrix
- Number of neighbors for computing the nearest-neighbor affinity subgraph
- Number of components for the spectral embedding

Spectral clustering was run only on Cars196 due to runtime inefficiencies.

	k	Affinity	Number of neighbors	Number of components
Cars196	50	Nearest neighbor subgraph	20	50

Table E.18: Hyperparameter values for spectral clustering.

Consensus-Driven Propagation (CDP) [150]. The hyperparameters are:

- Number of neighbors for computing the nearest-neighbor subgraph
- Threshold on cosine similarity in the nearest-neighbor subgraph for pruning edges
- Threshold step size
- Max cluster size

	Number of neighbors	Threshold	Threshold step size	Max cluster size
Cars196	2	0.6	0.01	320
SOP	5	−0.2	0.05	10

Table E.19: Hyperparameter values for CDP.

Tree Deduction [142]**.** The hyperparameters are the number of neighbors for computing the nearest-neighbor subgraph and a threshold on cosine similarity for pruning edges during tree deduction.

	Number of neighbors	Threshold
Cars196	2	0.75
SOP	1	0.6

Table E.20: Hyperparameter values for tree deduction.

Tree Deduction with Embedding ℓ_2-Norm Confidence. The hyperparameters are the number of neighbors for computing the nearest-neighbor subgraph and a threshold on cosine similarity for pruning edges during tree deduction.

	Number of neighbors	Threshold
Cars196	20	0.6
SOP	3	0.65

Table E.21: Hyperparameter values for tree deduction with embedding ℓ_2-norm confidence.

Embedding ℓ_2-Norm Confidence + GCN-E + Tree Deduction. The hyperparameters are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ_1)
- Ignore ratio (IR)
- Threshold on cosine similarity for pruning edges during tree deduction (τ_2)

- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

	N	τ_1	IR	τ_2	H	LR	MOM	ℓ_2	D
Cars196	60	0.0	0.1	0.6	512	0.01	0.9	1×10^{-5}	0.2
SOP	5	0.0	0.7	0.7	512	0.01	0.9	1×10^{-5}	0.0

Table E.22: Hyperparameter values for embedding ℓ_2-norm confidence + GCN-E + tree deduction.

GCN-VE [142]**.** GCN-VE has two sub-networks, GCN-V and GCN-E. We present the hyperparameters associated with each sub-network independently.

The hyperparameters for GCN-V are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

The hyperparameters for GCN-E are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ_1)

	N	τ	H	LR	MOM	ℓ_2	D
Cars196	10	0.0	512	0.01	0.9	1×10^{-5}	0.0
SOP	2	0.0	512	0.1	0.9	1×10^{-5}	0.0

Table E.23: Hyperparameter values for GCN-V.

- Ignore ratio (IR)
- Threshold on cosine similarity for pruning edges during tree deduction (τ_2)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)
- Dropout probability (D)

	N	τ_1	IR	τ_2	H	LR	MOM	ℓ_2	D
Cars196	200	0.0	0.0	0.7	512	0.01	0.9	1×10^{-5}	0.0
SOP	5	0.0	0.7	0.8	512	0.01	0.9	1×10^{-5}	0.0

Table E.24: Hyperparameter values for GCN-E.

STAR-FC [108]. The hyperparameters are:

- Number of neighbors for computing the adjacency matrix (N)
- Threshold on cosine similarity for pruning edges in the adjacency matrix (τ)
- Number of seed clusters (SC)
- Number of nearest clusters (NC)
- Number of random clusters (RC)
- Random node proportion (RNP)

- Prune threshold (P)
- Intimacy threshold (I)
- Number of units in the hidden layers of the network (H)
- Learning rate (LR)
- Momentum (MOM)
- ℓ_2 weight decay (ℓ_2)

	N	τ	SC	NC	RC	RNP	P	I	H	LR	MOM	ℓ_2
Cars196	10	0.0	4	6	10	0.9	0.15	0.6	512	0.1	0.9	1×10^{-5}
SOP	3	0.0	4	500	250	0.9	0.5	0.7	512	0.1	0.9	1×10^{-5}

Table E.25: Hyperparameter values for STAR-FC.