On the Significance of Distance in Machine Learning

Hyeongji Kim

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2023

UNIVERSITY OF BERGEN

On the Significance of Distance in Machine Learning

Hyeongji Kim

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 23.10.2023

Year: 2023

Title: On the Significance of Distance in Machine Learning

Name: Hyeongji Kim

Print: Skipnes Kommunikasjon / University of Bergen

Scientific environment

This study was conducted at the Institute of Marine Research (IMR) and Department of Informatics, University of Bergen (UiB). It was funded by the Institute of Marine Research.
The research was supervised by Assoc. Prof. Ketil Malde (IMR and UiB) and Assoc. Prof. Pekka Parviainen (UiB). Dr. Nils Olav Handegard (IMR) also participated in the early years of research as a co-superviser.

Acknowledgements

First, I would like to express my gratitude to my supervisors, Assoc. Prof. Ketil Malde and Assoc. Prof. Pekka Parviainen for their generous advice. During my PhD studies, they supported me with various aspects, such as helping me to improve my research and providing valuable feedback. I am grateful for their guidance and support in enabling me to develop as a researcher.

I want to thank my colleagues at the Institute of Marine Research (IMR). Dr. Nils Olav Handegard, who was also one of my co-supervisors, gave helpful advice. Thank you, Dr. Vaneeda Shalini Devi Allken, Dr. Yi Liu, Dr. Rokas Kubilius, and Dr. Babak Khodabandeloo for your time and help, and for giving me useful advice on living in Norway and adjusting to the institute. Group leader Dr. Rolf Korneliussen helped me with various administrative tasks. IT and HR teams also deserve a mention for assisting me with administrative works and technical issues.

I also want to thank my fellow PhD students and administrative staff at the Department of Informatics, University of Bergen (UiB) for the time and help that I received during my studies.

Thank you to my friends for your time and friendship. Lastly, I want to thank my family for their support.

Preface

At the beginning of my PhD, I wanted to pursue my research on interpretations of machine learning models, a field also referred to as explainable AI (XAI). Such interpretations can be applied to find important regions in the classification of image data or to validate models [Montavon et al., 2018]. I planned to apply interpretations of machine learning models to marine datasets. Among numerous approaches, I was mostly interested in gradient-based interpretation approaches [Simonyan et al., 2013] as these are similar to the idea that I was considering, which was to use minimum distance perturbations that change the predicted class as interpretations of models. However, gradient-based interpretations usually generate results that are too noisy for human eyes. Hence, it is often thought that such results are due to *the problem of gradient-based interpretation methods*. To avoid noisy interpretation results, several remedies [Bykov et al., 2022; Smilkov et al., 2017] have been proposed. Additionally, Ghorbani et al. [2019] showed that interpretations are vulnerable to slight modification of data points. Thus, when interpretation algorithms are given almost indistinguishable inputs with the same predicted class, interpretations of models can differ greatly. Again, it seems the common viewpoint is that vulnerability of interpretation is due to *the problem of common interpretation methods*. Therefore, few works [Lakkaraju et al., 2020] have proposed "robust interpretation methods" that are less vulnerable to the adversarial perturbation of data points.

Until recently, visual evaluation was commonly used for evaluating interpretation methods. Because this evaluation can be susceptible to human bias, Adebayo et al. [2018] suggested using non-visual evaluation of explanations. For instance, they revealed that a large number of interpretation results do not correlate with model parameters. Specifically, only some methods are dependent on the classification models, which include the gradient approach [Simonyan et al., 2013], even though it may not necessarily give visually appealing results. In addition to these findings, the proposed "improvements" [Bykov et al., 2022; Smilkov et al., 2017] of interpretation methods use average interpretation results to obtain less noisy results. Thus, one can regard their interpretations as the results of an averaging ensemble model rather than a target classification model.

Taking all these findings into consideration, it seems common viewpoints overlook the important possibility that *the problems mostly lie in common machine learning models* rather than in the interpretation methods. Specifically, *the issues may lie in machine learning models* that make highly accurate predictions but *use different reasonings* for their predictions, thus they give unsatisfying interpretation results. If this is correct, then using interpretation methods that provide visually appealing interpretations and "robust interpretation methods" [Lakkaraju et al., 2020] can conceal actual issues of models. Consequently, we may lose the opportunity to improve machine learning models. To focus on such improvement, I started to explore the gap between machine learning predictions and human predictions.

One of the widely known differences is the vulnerability of machine learning models against *adversarial examples* [Szegedy et al., 2013], which refers to carefully crafted inputs to fool models by applying almost indistinguishable modifications. While I was searching for the properties of adversarially robust classifiers, I identified an issue in the common definition of adversarial robustness. I speculated that this might be the reason for an important challenge in robust machine learning: *the trade-off between natural accuracy and adversarial robustness*. This trade-off means that adversarially robust classifiers are less accurate in classifying natural (not perturbed) inputs than standard classifiers. Thus, in paper III, I proposed a new definition of adversarial robustness that resolves the trade-off, at least on training data.

Soon after creating this new definition of adversarial robustness, I found that the nearest neighbor (1-NN) classifier becomes the optimally robust classifier for training data points. However, it is well known that the 1-NN classifier often generalizes poorly on test data points. Additionally, its generalization power will depend on the dataset and distance metric. Through further analysis, I found that normally there is greater similarity between the behavior of robust classifiers and 1-NN classifiers than there is between standard classifiers and 1-NN classifiers. From this, I hypothesized that the trade-off between natural accuracy and adversarial robustness might be due to *the use of not discriminative distance measures*. Specifically, not discriminative distances include distances where their corresponding 1-NN classifiers perform poorly (see section 2.4 for a detailed explanation of discriminality). Because robust models behave similarly to 1-NN classifiers, they will be less accurate in classifying natural inputs. This might also explain why finding accurate and robust classification was relatively easy [Schott et al., 2018b] for the MNIST dataset [LeCun et al., 2010], given that the l_2 norm-based 1-NN classifier performs well on MNIST (accuracy higher than 0.95). In Chapter 5, I explore the possible connection between the discriminality of the distance metric and the trade-off. After noticing that the (poor) choice of distance might be the cause of the trade-off, I explored the field of learning a dissimilarity measure: *metric learning*.

Metric learning learns dissimilarity functions such that similar data points are located close and dissimilar data points far apart in a learned embedding space. Because magnifying distances with a constant value will not change the orders of distances, it seems natural to assume that the scale of the distance should not matter in metric learning models. If metric learning methods are dependent on the scale of the embedding space, training can be unstable or slower due to the unnecessary scale changes. To prevent the dependence on scale, metric learning models need to consider the distance-ratio. Hence, in paper II, I proposed using a distance-ratio-based (DR) formulation in metric learning.

Following this, I experimented with several ideas in related fields. One involved improving episode training in few-shot learning where randomly sampled support points and query data points are used in each episode. My idea was to use each data point as both support and query data points in a modified episode by switching their roles. I found that training with such a "rolling process" was much faster with respect to the number of episodes in training. Unfortunately, when their computation time was considered, the speed gain seemed to be small and was dependent on the model architecture. Hence, I decided to abandon this research and try different ideas. Another idea I experimented with was post hoc modification of metric learning models by changing the discriminative power of embedding space, but this method did not consider the dimensionality of feature space. However, I could not identify any noticeable benefits of the method employed, perhaps because I had not considered the dimensionality of embedding. Hence, I looked for different ideas. While I was thinking about a possible new application of metric learning methods, I notice that learned embedding may be used for estimating hierarchical structures of classes.

Given that hierarchical structure information is important in studying biological datasets, I started to experiment with plankton datasets. While experimenting with classification based metric learning models, I found that learned models can approximately estimate class hierarchies. In situations with a known class hierarchy, such an analysis can also be used for verifying whether a learned class distance matches our knowledge. Moreover, I also found an unexpected benefit of DR formulation in that inference performance was higher with DR formulation than when using a standard formulation. In addition to the plankton datasets, these results were also confirmed by two benchmark datasets. In paper IV, I suggested using classification-based metric learning models for estimating class hierarchy when this is not available.

There are also ideas that were not investigated during my PhD. Hence, I describe three of these in section 6.2 as possible future works.

Abstract in English

The notion of distance is fundamental in machine learning. The choice of distance matters, but it is often challenging to find an appropriate distance. *Metric learning* can be used for learning distance(-like) functions. Common deep learning models are vulnerable to the adversarial modification of inputs. Devising adversarially robust models is of immense importance for the wide deployment of machine learning models, and distance can be used for the study of *adversarial robustness*. Often, hierarchical relationships exist among classes, and these relationships can be represented by the hierarchical distance of classes. For classification problems that must take these class relationships into account, *hierarchy-informed classification* can be used.

I propose distance-ratio-based (DR) formulation for metric learning. In contrast to the commonly used formulation, DR formulation has two favorable properties. First, it is invariant of the scale of an embedding. Secondly, it has optimal class confidence values on class representatives.

For a large perturbation budget, standard adversarial accuracy (SAA) allows natural data points to be considered as adversarial examples. This could be a reason for the tradeoff between accuracy and SAA. To resolve the issue, I proposed a new definition of adversarial accuracy named Voronoi-epsilon adversarial accuracy (VAA). VAA extends the study of local robustness to global robustness.

Class hierarchical information is not available for all datasets. To handle this challenge, I investigated whether classification-based metric learning models can be used to infer class hierarchy.

Furthermore, I explored the possible effects of adversarial robustness on feature space. I found that the distance structure of robustly trained feature space resembles that of input space to a greater extent than does standard trained feature space.

Sammendrag

Avstandsbegrepet er grunnleggende i maskinlæring. Hvordan vi velger å måle avstand har betydning, men det er ofte utfordrende å finne et passende avstandsmål. *Metrisk læring* kan brukes til å lære funksjoner som implementerer avstand eller avstandslignende mål. Vanlige dyplæringsmodeller er sårbare for modifikasjoner av input som har til hensikt å lure modellen (*adversarial examples*, motstridende eksempler). Konstruksjon av modeller som er robuste mot denne typen angrep er av stor betydning for å kunne utnytte maskinlæringsmodeller i større skala, og et passende avstandsmål kan brukes til å studere slik motstandsdyktighet. Ofte eksisterer det hierarkiske relasjoner blant klasser, og disse relasjonene kan da representeres av den hierarkiske avstanden til klasser. I klassifiseringsproblemer som må ta i betraktning disse klasserelasjonene, kan *hierarki-informert klassifisering* brukes.

Jeg har utviklet en metode kalt /distance-ratio/-basert (DR) metrisk læring. I motsetning til den formuleringen som normalt anvendes har DR-formuleringen to gunstige egenskaper. For det første er det skala-invariant med hensyn til rommet det projiseres til. For det andre har optimale klassekonfidensverdier på klasserepresentantene.

Dersom rommet for å konstruere modifikasjoner er tilstrekklig stort, vil man med standard adversarial accuracy (SAA, standard motstridende nøyaktighet) risikere at naturlige datapunkter blir betraktet som motstridende eksempler. Dette kan være en årsak til SAA ofte går på bekostning av nøyaktighet. For å løse dette problemet har jeg utviklet en ny definisjon på motstridende nøyaktighet kalt Voronoi-epsilon adversarial accuracy (VAA, Voronoi-epsilon motstridende nøyaktighet). VAA utvider studiet av lokal robusthet til global robusthet.

Klassehierarkisk informasjon er ikke tilgjengelig for alle datasett. For å håndtere denne utfordringen har jeg undersøkt om klassifikasjonsbaserte metriske læringsmodeller kan brukes til å utlede klassehierarkiet.

Videre har jeg undersøkt de mulige effektene av robusthet på feature space (egenskapsrom). Jeg fant da at avstandsstrukturen til et egenskapsrom trent for robusthet har

større likhet med avstandsstrukturen i rådata enn et egenskapsrom trent uten robusthet.

List of notations

(Some symbols have different meaning in different chapters. Hence, they appear in multiple chapters of this list. Notations in Chapter 7 are not listed here.)

Chapter 1 - Introduction

l_2 norm distance	Euclidean distance.
l_p norm distance	Generalization of l_2 norm distance to $p \geq 1$.
$\|v\|_p$	l_p norm of a vector v.
$\|v\|$	l_2 norm of a vector v.
$d(\cdot, \cdot)$	Distance (metric) function for measuring dissimilarity of data points.
$\mathcal{X}$	Nonempty input space. $\mathcal{X} \subset \mathbb{R}^{d_I}$.
$\mathbb{R}$	The set of real numbers.
d_I	Dimension of an input space.
$\mathcal{Y}$	Set of possible classes.
$\mathcal{D}$	Joint data distribution. $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$.
x	(Input) data point.
c	Corresponding class of an (original) data point $x \in \mathcal{X}$. $c \in \mathcal{Y}$.
$f(\cdot)$	Feature mapping (embedding) function. $f : \mathcal{X} \longrightarrow \mathcal{Z}$.
$\mathcal{Z}$	Feature (embedding) space. $\mathcal{Z} \subset \mathbb{R}^{d_F}$.
d_F	Dimension of a feature space.
$l_y(x)$	Logit value for data point x and class y using a feature mapping function. Logit values are used in softmax classifier.
y	Arbitrary class $y \in \mathcal{Y}$.
W	Weight matrix used for logit calculation.
W_y	Weight vector from the matrix W for class y.
b_y	Bias term for class y used for logit calculation.

$p(y\|x)$	Exact probability that a data point x belongs to class y. It is also known as class probability.
$\hat{p}$	Estimated probability of an exact probability p. For example, $\hat{p}(y\|x)$ is an estimated probability of $p(y\|x)$.
$\hat{c}$	Predicted class of a data point.
$\ell_{\mathrm{CE}}(\cdot, \cdot)$	The cross-entropy loss function for a data point.
$\log(\cdot)$	The element-wise logarithm function.
$\overrightarrow{\mathbb{1}}_y$	One-hot vector whose element has the value of 1 for corresponding element of y.
$\hat{P}(\mathcal{Y}\|x)$	Estimated class probability vector (softmax vector) of a data point x. Its corresponding element of index for class y has value $\hat{p}(y\|x)$.
L_{CE}	The cross-entropy loss function for a mini-batch.
B	Mini-batch. $B \subset \mathcal{D}$.

Chapter 2 - Metric learning

$d_{\mathrm{learned}}(\cdot, \cdot)$	Learned dissimilarity function using metric learning. In general, it is a pseudometric. $d_{\mathrm{learned}}(x_1, x_2) := d(f(x_1), f(x_2))$ for two data points x_1 and x_2.
$\mathcal{Y}_{\mathrm{train}}$	Set of classes used in training phase. $\mathcal{Y}_{\mathrm{train}} \subset \mathcal{Y}$.
$\ell_{\mathrm{Contrastive}}$	Contrastive loss [Hadsell et al., 2006] for a data point pair.
$\mathbb{1}(\cdot)$	The indicator function. $\mathbb{1}$ (True) $= 1$ and $\mathbb{1}$ (False) $= 0$.
m	Margin parameter for metric learning models.
$\mathrm{ReLU}(\cdot)$	Rectified linear unit function. $\mathrm{ReLU}(v) = max(0, v)$ for a value v.
ℓ_{Triplet}	Triplet loss [Wang et al., 2014] for a triplet.
N	Number of training data points.
$\mathcal{O}(\cdot)$	Big O notation.
$\tilde{v}$	Normalized vector of a non-zero vector v. $\tilde{v} = \frac{v}{\|\|v\|\|}$.
θ_y	Angle between $\tilde{W}_y$ and $\tilde{f}(x)$.
s	Scaling parameter for metric learning models. $s > 0$.
L_{NormFace}	Training loss for NormFace model [Wang et al., 2017] for a mini-batch.
μ_y	Average embedding of a class y. For Euclidean embedding space, $\mu_y = \frac{1}{\|X_y\|} \sum_{x \in X_y} f(x)$.
X_y	Set of data points belong to a class y. $X_y \subset \mathcal{X}$.

L_{LMC}	Training loss (large margin cosine loss) for CosFace model [Wang et al., 2018] for a mini-batch.
L_{ArcFace}	Training loss for ArcFace model [Deng et al., 2019] for a mini-batch.
α	Parameter that controls the stability and speed of updates for exponential moving average (EMA) [Zhe et al., 2019]. $0 < \alpha < 1$.
θ'_c	Clipped value of the angle θ_c. $\theta'_c := \text{clip}(\theta_c, \left[0, \frac{\pi}{2}\right])$.
$\text{clip}(\cdot, \cdot)$	The clipping function that limits a value within a specified range.
$\mathbb{S}^{d_F-1}$	Unit spherical (normalized) space. $\mathbb{S}^{d_F-1} = \left\{z \in \mathbb{R}^{d_F} \middle\| \|z\| = 1\right\}$.
ϵ_{stab}	Positive value close to zero used for the numerical stability when normalizing vectors.
$\mathbb{D}_\tau^{d_F}$	Poincaré ball model with a curvature parameter τ. It is one of the hyperbolic space models. $\mathbb{D}_\tau^{d_F} = \left\{z \in \mathbb{R}^{d_F} \middle\| \tau \|z\|^2 < 1\right\}$.
τ	Curvature parameter of a Poincaré ball model. $\tau > 0$.
$\oplus$	The gyrovector addition symbol.
$\exp_{\mathbf{x}}^{\tau}(v)$	Exponential mapping to map vectors from Euclidean space $\mathbb{R}^{d_F}$ into a Poincaré ball model $\mathbb{D}_\tau^{d_F}$.
$\mathbf{x}$	Base point used in a Poincaré ball model $\mathbb{D}_\tau^{d_F}$.
$\lambda_{\mathbf{x}}^{\tau}$	Conformal factor used in a Poincaré ball model $\mathbb{D}_\tau^{d_F}$. $\lambda_{\mathbf{x}}^{\tau} = \frac{2}{1-\tau\|\mathbf{x}\|^2}$
$\phi(m)$	Discriminative function for a margin parameter m [Liu et al., 2020]. $\phi(m) = \frac{D_{\text{inter}}(m)}{D_{\text{intra}}(m)}$.
$D_{\text{inter}}(m)$	Inter-class variance for a margin m.
$D_{\text{intra}}(m)$	Intra-class variance for a margin m.
$\mathcal{Y}_I$	Class set of interest. $\mathcal{Y}_I \subset \mathcal{Y}$.
X_j	Set of data points for the j th class in $\mathcal{Y}_I$.
$\tilde{f}_m(x)$	Normalized embedding of a data point x for a margin m.
$\mu_j(m)$	Average embedding for the j th class in $\mathcal{Y}_I$ for a margin m.
π_{ratio}	Embedding space density [Roth et al., 2020]. $\pi_{\text{ratio}} = \frac{\pi_{\text{intra}}}{\pi_{\text{inter}}}$.
π_{inter}	Average inter-class distance.
π_{intra}	Average intra-class distance.

Z_{inter}	Normalization constant for the calculation of π_{inter}. $Z_{\text{inter}} = \lvert\mathcal{Y}_I\rvert(\lvert\mathcal{Y}_I\rvert - 1)$.
Z_{intra}	Normalization constant for the calculation of π_{intra}. $Z_{\text{intra}} = \sum_{k=1}^{\lvert\mathcal{Y}_I\rvert} \lvert X_k\rvert(\lvert X_k\rvert - 1)$.
ρ	Measure to assess compression in feature representation by using spectral decay of an embedding. $\rho = \text{KL}(\mathcal{U}_{d_F}\lvert\lvert\mathcal{S})$.
$\text{KL}(\cdot\lvert\lvert\cdot)$	The Kullback–Leibler divergence.
$\mathcal{U}_{d_F}$	d_F-dimensional discrete uniform distribution.
$\mathcal{S}$	Normalized spectrum of singular values (SV) sorted in descending order.
R^2	Class separation [Kornblith et al., 2021] to measure discriminative power of a feature representation. $R^2 = 1 - \frac{\bar{d}_{within}}{\bar{d}_{total}}$.
$\bar{d}_{\text{within}}$	Average within-class cosine distance.
$\bar{d}_{\text{total}}$	Overall average cosine distance.
$\text{sim}(\cdot,\cdot)$	Cosine similarity function. $\text{sim}(v_1, v_2) = \tilde{v}_1 \cdot \tilde{v}_2$ for two vectors v_1 and v_2.
$d_{y,x}$	Distance from a data point x to class y. For example, distance from feature vector $f(x)$ to the proxy representative of class y can be used for $d_{y,x}$.

Chapter 3 - Adversarial robustness

$d(\cdot,\cdot)$	Distance (metric) function for measuring adversarial robustness in Chapters 3 and 5.
$\mathcal{C}$	Target classifier. $\mathcal{C} : \mathcal{X} \to \mathcal{Y}$.
x	Original (unmodified) data point in Chapters 3 and 5.
x'	Perturbed (modified) data point from an original data point x. $x' \in \mathcal{X}$. It is an adversarial example when $\mathcal{C}(x') \neq \mathcal{C}(x)$ and $d(x, x') \leq \epsilon$ [Biggio et al., 2013].
ϵ	Allowed perturbation budget.
c_t	(Specific) target class for a targeted adversarial attack. $c_t \neq c$.
$\ell(\cdot,\cdot)$	Classification loss function. It is based on a target classifier $\mathcal{C}$ (or its softmax outputs).
x'_{FGSM}	Adversarially perturbed data point generated by the FGSM attack [Goodfellow et al., 2014].
∇	Gradient operator.
$\text{sign}(\cdot)$	The element-wise sign function.

α_{step}	Step size for iterative adversarial attacks.
k	Number of iterations for iterative adversarial attacks.
$x'_{\text{BIM};i}$	Adversarially perturbed data point using the BIM attack [Kurakin et al., 2016] after i-iteration.
$\text{Clip}_{x,\epsilon}(v)$	The clip function. It clips (limits) the vector v element-wise such that the resulting output vector is within ϵ distance (l_∞ norm) from a data point x.
$\mathbb{B}(x,\epsilon)$	ϵ-ball around a data point x. Mathematically, $\mathbb{B}(x,\epsilon) = \{x' \in \mathcal{X} \mid d(x,x') \leq \epsilon\}$.
$x'_{\text{PGD};i}$	Adversarially perturbed data point using the PGD attack [Madry et al., 2017] after i-iteration.
r^*	Minimal distance perturbation such that the perturbed position $x + r^*$ has a predicted class different from the correct class c. Mathematically, $r^* = \underset{r:\, x+r\in\mathcal{X}}{\text{argmin}}\, d(x, x+r)$ such that $\mathcal{C}(x+r) \neq c$.
r	Minimal distance perturbation. It satisfies $x + r \in \mathcal{X}$.
$f_{\text{obj}}(\cdot)$	Object function [Carlini and Wagner, 2017]. It satisfies $f_{\text{obj}}(x + r) \leq 0$ if and only if $\mathcal{C}(x+r) = c_t$.
λ	Hyperparameter for CW attack [Carlini and Wagner, 2017]. Additionally, a hyperparameter for TRADES training [Zhang et al., 2019a]. $\lambda > 0$.
$R(x)$	Adversary region for a data point x. It means an allowed region of the perturbations for a data point x.
a	Adversarial accuracy. Mathematically, $a = \mathbb{E}_{(x,c)\sim D}[\mathbb{1}(\mathcal{C}(x^*) = c)]$ for $x^* = \underset{x'\in R(x)}{\text{argmax}}\, \ell(x', c)$.
x^*	Worst (strongest) adversarially perturbed data point from the original data points x. Mathematically, $x^* = \underset{x'\in R(x)}{\text{argmax}}\, \ell(x', c)$.
$a_{\text{std}}(\epsilon)$	Standard adversarial accuracy by setting $R(x) = \mathbb{B}(x,\epsilon)$.
$\ell_{\text{AT}}(x,c)$	Adversarial training loss for a data point x and its corresponding class c.
$\hat{x^*}$	Adversarially perturbed data point using an attack algorithm.
α	Parameter for adversarial training [Goodfellow et al., 2014; Madry et al., 2017]. $0 \leq \alpha \leq 1$.
$\ell_{\text{TRADES}}(x,c)$	TRADES training loss [Zhang et al., 2019a] for a data point x and its corresponding class c.
$\mathcal{C}_{\text{smooth}}(\cdot)$	Smoothed classifier based on a base classifier $\mathcal{C}(\cdot)$ for randomized smoothing [Cohen et al., 2019; Lecuyer et al., 2019].
δ	Perturbation noise for randomized smoothing.

σ	Standard deviation of a Gaussian distribution.
$\mathcal{N}$	Multivariate Gaussian distribution.
$d_{\text{combined}}(\cdot, \cdot)$	Combined distance from two distance functions.

Chapter 4 - Hierarchy-informed classification

w	Arbitrary node in a tree (or a DAG).
y	Arbitrary class (class node).
c	Specific class (class node).
$d_H(\cdot, \cdot)$	Hierarchical distance function between classes.
$\mathcal{V}$	Set of nodes in a graph.
$d_{H;\text{LCS}}(\cdot, \cdot)$	Hierarchical distance using lowest common subsumer (LCS). Mathematically, $d_{H;\text{LCS}}(y_1, y_2) = \frac{\text{height}(lcs(y_1, y_2))}{\max_{w \in \mathcal{V}} \text{height}(w)}$ for classes y_1 and y_2. $0 \leq d_{H;\text{LCS}} \leq 1$.
$lcs(\cdot, \cdot)$	LCS of class nodes.
$\text{height}(w)$	Length of the shortest path from node w to a leaf node.
$d_{H;\text{path}}$	Hierarchical distance using the shortest path.
$p(w_{\text{child}} \vert w_{\text{parent}}; x)$	Exact conditional probability that a data point x belongs to a child node w_{child} given that it belongs to parent node w_{parent}.
w_{child}	Child node.
w_{parent}	Corresponding parent node of the node w_{child}.
$y^{(i)}$	Ancestor of a class node y with height i. $y^{(0)} = y$. $y^{(\text{height}(y))} = R_y$.
R_y	The highest ancestor of a class node y. When hierarchical structure is a rooted tree, thus there is only one node without parent, R_y is a root node.
$\text{Leaves}(w)$	The set of class nodes of the subtree rooted by the node w.
L_{HXE}	Hierarchical cross-entropy (HXE) loss [Bertinetto et al., 2020] for hierarchy-informed classification.
$\lambda(c^{(l)})$	Weight for edge between node $c^{(l)}$ and node $c^{(l+1)}$ for HXE loss L_{HXE}. Bertinetto et al. suggested $\lambda(w) = e^{-\alpha \text{height}(w)}$ for weights.

α	Hyperparameter for HXE loss.
$\overrightarrow{\text{soft}}_y$	Soft-label vector of a class y for soft-label based [Bertinetto et al., 2020] hierarchy-informed classification.
$\overrightarrow{\text{soft}}_{y;y_1}$	Element for a class y_1 of soft-label vector of class y. It is defined as: $\overrightarrow{\text{soft}}_{y;y_1} = \frac{\exp(-\beta d_H(y,y_1))}{\sum_{y_2 \in \mathcal{Y}} \exp(-\beta d_H(y_2,y_1))}$.
β	Hyperparameter for soft-label [Bertinetto et al., 2020]. $\beta \geq 0$.
L_{Soft}	Soft-label loss.
$s_H(\cdot,\cdot)$	Hierarchical class similarity defined by Barz and Denzler [2019]. Mathematically, $s_H(y_1,y_2) := 1 - d_{H;\text{LCS}}(y_1,y_2)$.
L_{CORR}	CORR loss [Barz and Denzler, 2019] for hierarchy informed classification. It is defined as: $L_{\text{CORR}} = \frac{1}{\lvert B \rvert} \sum_{(x,c)\in B} (1 - \cos\theta_c)$.
$L_{\text{H-spherical}}$	Training loss suggested by Mettes et al. [2019] for hierarchy informed classification. It is defined as: $L_{\text{H-spherical}} = \sum_{(x,c)\in B} (1 - \cos\theta_c)^2$.
$L_{\text{disto}}(W)$	Distortion-based penalty term proposed by Garnot and Landrieu [2021].
s	Scaling factor for distortion-based penalty term.
$\mathbb{R}^+$	The set of positive real numbers.
D_H	Hierarchical distance matrix whose (l,k)-th element is the hierarchical distance $d_H(y_l,y_k)$.

Chapter 5 - Adversarial robustness and feature space characteristics

D_I	Input space distance matrix. l_p distance between (input) training data points was used for the calculation.
D_F	Feature space distance matrix. Euclidean distance between training data points on the feature space was used for the calculation.
d_{DoV}	Number of dimensions in the normalized singular value spectrum of a feature space whose variance is higher than average $\frac{1}{d_F - 1}$.

Chapter 6 - Discussion and future directions

$\measuredangle(v_1, v_2)$	Angle between two (unit) vectors v_1 and v_2.
$\measuredangle_{\min}(c)$	Smallest angle from the proxy $\tilde{W}_c$ of class c to a different proxy $\tilde{W}_{y'}$. Mathematically, $\measuredangle_{\min}(c) = \min_{y' \in \mathcal{Y}, y' \neq c} \measuredangle(\tilde{W}_c, \tilde{W}_{y'})$.
ϵ_{tiny}	Tiny number for the proposed training in subsection 6.2.1. $\epsilon_{\text{tiny}} > 0$.
u_i	The i-th (unit) principal direction from PCA.
$f_{\text{new}}(x)$	Modified feature from the re-weighting method in subsection 6.2.3.
w_i	Weight for the i-th principal direction for the re-weighting method in subsection 6.2.3. $w_i \geq 0$.

Appendix

$\mathbb{B}_1$	l_1 ball with radius ϵ_1. Mathematically, $\mathbb{B}_1 = \left\{x \in \mathbb{R}^{d_I} : \|x\|_1 \leq \epsilon_1\right\}$.
$\mathbb{B}_\infty$	l_∞ ball with radius ϵ_∞. Mathematically, $\mathbb{B}_\infty = \left\{x \in \mathbb{R}^{d_I} : \|x\|_\infty \leq \epsilon_\infty\right\}$.
ϵ_1	The radius of the l_1 ball $\mathbb{B}_1$. It satisfies the setting $\epsilon_\infty < \epsilon_1 < d_I \epsilon_\infty$.
ϵ_∞	The radius of the l_∞ ball $\mathbb{B}_\infty$. $\epsilon_\infty > 0$.
$\mathcal{H}$	The convex hull of the union of $\mathbb{B}_1$ and $\mathbb{B}_\infty$.
d_{combined}	Combined distance. It is defined as: $d_{\text{combined}}(x_1, x_2) = \beta \|x_2 - x_1\|_1 + (1 - \beta) \|x_2 - x_1\|_\infty$.
β	Constant for the combined distance. It is set to $\beta := \frac{\epsilon_1 - \epsilon_\infty}{\epsilon_\infty (d_I - 1)}$.
$\mathbb{B}_{\text{combined}}$	Ball with radius $\epsilon_{\text{combined}}$ based on the distance d_{combined}. Mathematically, $\mathbb{B}_{\text{combined}} = \left\{x \in \mathbb{R}^{d_I} : d_{\text{combined}}(0, x) = \beta \|x\|_1 + (1 - \beta) \|x\|_\infty \leq \epsilon_1\right\}$.
$\epsilon_{\text{combined}}$	Radius of ball based on the combined distance d_{combined}. It is set to $\epsilon_{\text{combined}} := \epsilon_1$.
t	Value between zero and one used for convex combination of two vectors.
x_1	Vector that belongs to the l_1 ball $\mathbb{B}_1$. Mathematically, $\|x_1\|_1 \leq \epsilon_1$.
x_2	Vector that belongs to the l_∞ ball $\mathbb{B}_\infty$. Mathematically, $\|x_2\|_\infty \leq \epsilon_\infty$.
i^*	Index of x that maximizes its absolute value, thus $\|v_{i^*}\| = \max_{1 \leq i \leq d_I} \|v_i\|$ where v_i is value of the point x of index i.

i^{**}	Index of x that has the second largest absolute value. When $d_I = 2$, $i^{**} = 3 - i^*$.
x_1^*	Intentionally chosen vector to be belong to the l_1 ball $\mathbb{B}_1$.
x_2^*	Intentionally chosen vector to be belong to the l_∞ ball $\mathbb{B}_\infty$.
t^*	Intentionally chosen value between zero and one for convex combination of two vectors.
x^*	Intentionally chosen vector for showing $\mathbb{B}_{\text{combined}} \not\subset \mathcal{H}$ when $d_I > 2$.
ω	Constant used for showing $\mathbb{B}_{\text{combined}} \not\subset \mathcal{H}$ when $d_I > 2$. It is set to $\omega := \frac{(d_I-1)\epsilon_1\epsilon_\infty}{\epsilon_1+(d_I-2)\epsilon_\infty}$. It satisfies the two inequalities $\epsilon_\infty < \omega$ and $\epsilon_1 < 2\omega$.
α_i	Value of the vector x_1^* of index i.
β_i	Value of the vector x_2^* of index i.
$\mathcal{H}_{A\cup B}$	Convex hull of the convex sets A and B.
RHS	The set on the right hand side of equation (7.1). Mathematically, $\text{RHS} = \{ta + (1-t)b : 0 \le t \le 1, a \in A, b \in B\}$.
a_3	Convex combination of two vectors a_1 and a_2 in the convex set A. Mathematically, $a_3 := \frac{\lambda t_1 a_1 + (1-\lambda) t_2 a_2}{\lambda t_1 + (1-\lambda) t_2}$.
b_3	Convex combination of two vectors b_1 and b_2 in the convex set B. Mathematically, $b_3 := \frac{\lambda(1-t_1)b_1 + (1-\lambda)(1-t_2)b_2}{\lambda(1-t_1)+(1-\lambda)(1-t_2)}$.
γ	Convex combination of two values t_1 and t_2. Mathematically, $\gamma := \lambda t_1 + (1-\lambda)t_2$.
λ	A value between zero and one used for convex combination.

Abbreviations

(Abbreviations in Chapter 7 are not listed here.)

Chapter 1 - Introduction

k-NN classifier	k-nearest neighbor classifier.
SVM	Support-vector machine.
RGB	Red, green, and blue.
CIFAR-10	The name of the dataset introduced by Krizhevsky et al. [2009].
DNN	Deep neural network.
CE loss	Cross-entropy loss.

Chapter 2 - Metric learning

ReLU	A rectified linear unit. A type of activation function.
NormFace	Normalized softmax model [Wang et al., 2017; Zhai and Wu, 2019].
CosFace	A modification of NormFace model by introducing cosine margin [Wang et al., 2018].
LMCL	Large margin cosine loss for CosFace model.
ArcFace	A modification of NormFace model by introducing angular margin [Deng et al., 2019].
EMA	Exponential moving average.
AdaCos	NormFace model trained by adaptive scaling factor s [Zhang et al., 2019b].
PCA	Principal component analysis.
CUB	The abbreviated name of the dataset introduced by Wah et al. [2011]. The original name is "Caltech-UCSD Birds-200-2011".

CARS196	The name of the dataset introduced by Krause et al. [2013].
SOP	The abbreviated name of the dataset introduced by Oh Song et al. [2016]. The original name is "Stanford online products".
SV	Singular values.
DoV	Directions of significant variance [Roth et al., 2020].
ImageNet	The short name of the dataset introduced by Deng et al. [2009]; Russakovsky et al. [2015]. The original name is "ImageNet large scale visual recognition challenge (ILSVRC) 2012".
SD-softmax formulation	A formulation for metric learning using negative of squared distance (SD) on softmax formulation. ("Softmax-based formulation" in the paper II)
DR formulation	Distance-ratio-based formulation for metric learning introduced in paper II.

Chapter 3 - Adversarial robustness

CNN	Convolutional neural network.
LPIPS	Learned perceptual image patch similarity [Zhang et al., 2018].
FGSM	Fast gradient sign method for adversarial attack [Goodfellow et al., 2014].
BIM	Basic iterative method for adversarial attack [Kurakin et al., 2016].
PGD	Projected gradient descent method for adversarial attack [Madry et al., 2017].
CW	Carlini-Wagner attack [Carlini and Wagner, 2017] for adversarial attack.
Adam	Adaptive moment optimization [Kingma and Ba, 2014].
DeepFool	The name of an attack suggested by Moosavi-Dezfooli et al. [2015].
FAB	Fast adaptive boundary attack suggested by Croce and Hein [2020b].
SAA	Standard adversarial accuracy. An adversarial accuracy when using ϵ-ball for an adversary region $R(x)$.
AT	Adversarial training [Goodfellow et al., 2014; Madry et al., 2017].
ST	Standard (non-adversarial) training.

TRADES	An adversarial defense method [Zhang et al., 2019a]. The name stands for "TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization."
MEPs	Mutually exclusive perturbations [Tramer and Boneh, 2019]. Two perturbations are MEPs when robustness to one perturbation type necessary implies susceptibility to the other.
RST	Robust self-training suggested by Carmon et al. [2019].
VAA	Voronoi-epsilon adversarial accuracy proposed in paper III.
1-NN classifier	The (single) nearest neighbor classifier.

Chapter 4 - Hierarchy-informed classification

DAG	Directed acyclic graph structure.
LCS	Lowest common subsumer [Barz and Denzler, 2019; Bertinetto et al., 2020].
WordNet	An English word database that includes their semantic relationships [Miller, 1998].
HXE	Hierarchical cross-entropy for hierarchy-informed classification [Bertinetto et al., 2020].
CORR loss	Training loss suggested by Barz and Denzler [2019] for hierarchy-informed classification.

Chapter 5 - Adversarial robustness and feature space characteristics

SVHN	The name of the dataset introduced by Netzer et al. [2011].
MC	MC (Mean correlation) value was introduced in paper IV. It is a measure devised to compare two distance structures based on rank correlation values.
NPC	Nearest prototype classifier.

List of publications

This thesis is based on the following papers.

I Ketil Malde and **Hyeongji Kim**. *Beyond image classification: zooplankton identification with deep vector space embeddings.* arXiv preprint arXiv:1909.11380, 2019.

II **Hyeongji Kim**, Pekka Parviainen, and Ketil Malde. *Distance-Ratio-Based Formulation for Metric Learning.* arXiv preprint arXiv:2201.08676, 2022.

III **Hyeongji Kim**, Pekka Parviainen, and Ketil Malde. *Measuring Adversarial Robustness using a Voronoi-Epsilon Adversary.* In Proceedings of the Northern Lights Deep Learning Workshop, Volume 4, 2023.

IV **Hyeongji Kim**, Pekka Parviainen, Terje Berge, and Ketil Malde. *Inspecting class hierarchies in classification-based metric learning models.* arXiv preprint arXiv:2301.11065, 2023.

Contents

Chapter 1

Introduction

Classification is the process of sorting data into categories. Data are assigned to each category (or class) based on certain characteristics. Thus, the data in each class can be said to be *similar* in some way. Similarity can be used for different purposes, such as comparing data points without considering classes. Similarity can be abstract and is often difficult to define precisely. So how can we define similarity or dissimilarity?

The notion of dissimilarity can be expressed with a *distance*, also referred to as a *metric* or a *metric distance*. Euclidean distance, which denotes the concept of physical distance, is integral to our daily lives. Distance is also used in a wide range of scientific disciplines. For instance, it is used for the study of protein structures [Jumper et al., 2021]. A distance function need not be limited to Euclidean distance. Any function that satisfies certain properties can be considered as a distance.

Distance is a fundamental part of machine learning. It is used in many clustering methods, including k-means clustering and density-based clustering methods [Ester et al., 1996]. Additionally, it is used for modeling and the validation process of regression models. Numerous classification methods use distances for classification. For example, based on a distance measure, k-nearest neighbor (k-NN) classifiers find neighboring data points to classify new data points. Support-vector machine (SVM) uses margin in its optimization, where margin is defined as twice the distance between the decision boundary and the shortest training examples of each class.

Euclidean distance, also known as l_2 *norm distance*, is perhaps the most commonly used and familiar distance. It is used for explaining a multitude of physical phenomena, including gravity and electric forces. It is defined between two vectors (or two points). Euclidean distance can be used for comparing images by considering each RGB color channel value (or each pixel value for grayscale images) as an element of a vector.

The following example illustrates the use of Euclidean distance for comparing images. Figure 1.1 displays three images. As shown by the distance values (in the caption), semantically similar image pairs can have a larger Euclidean distance than semantically dissimilar image pairs. In other words, *Euclidean distance may not necessarily match semantic dissimilarity.*

(a) (b) (c)

Figure 1.1: (a) and (b): Two images (a) and (b) from training data of the CIFAR-10 dataset [Krizhevsky et al., 2009]. (c): shifted image from the first image (a) by one pixel value in the positive x-axis direction. l_2 distance between figures (a) and (b) is 5.5699. l_2 distance between figures 1.1 (a) and (c) is 6.8126, which is larger than the previous image pair. l_∞ distance between figures (a) and (b) is 0.4431. l_∞ distance between figures (a) and (c) is 0.7804, which is larger than the previous image pair.

To understand why the mismatch of semantic distance and Euclidean distance occurs, we need to understand the formal definition of Euclidean distance, namely, l_2 norm distance. Thus, let us consider the mathematical definition and its generalization, known as l_p *norm distance* ($p \geq 1$), which is also commonly used. Let $v = (v_1, v_2, \cdots, v_n) \in \mathbb{R}^n$ be a n-dimensional vector. Its l_p norm, which is denoted as $\|v\|_p$, is defined as:

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}, \qquad \text{(For } 1 \leq p < \infty\text{)}$$

$$\|v\|_\infty = \max_{1 \leq i \leq n} |v_i|. \qquad \text{(For } p = \infty\text{)}$$

l_2 norm is often denoted as $\|v\|$; that is, without explicitly mentioning $p = 2$ due to its popular usage. Because of its definition, l_∞ norm is also called *max norm*. For two n-dimensional vectors u and v, l_p norm (distance) between the two vectors is defined as $\|u - v\|_p$. From the definition, it is clear that l_p norm distance is based on the element-wise difference values of two vectors. From this, it is possible to understand how the mismatch occurs. The first column of Figure 1.1c is black and compared with Figure 1.1a, increased Euclidean distance. Additionally, the colors in the corresponding

positions seem to be similar for Figures 1.1a and 1.1b. This explains why their distance is not large. The same trend is shown when we consider l_∞ norm distance. (Further analysis confirms that for Figures 1.1a and 1.1c, most (86.20 %) of the squared value of l_2 norm distance is due to the first column, while the maximum difference of l_∞ norm distance is in the first column.)

The mismatch between l_p norm distance and perceptual dissimilarity raises the following question. *Can we find distances (or distance-like functions) that match semantic dissimilarity?* To answer this, we first need to understand the formal definition of general distance.

Mathematically, a distance $d(\cdot, \cdot)$ is a function that satisfies the following properties for any three data points x, y, z of a space:

$$d(x, y) = 0 \iff x = y, \qquad (\Rightarrow\text{: The identity of indiscernibles,} \tag{1.1a}$$

$$\Leftarrow\text{: The indiscernibility of identicals)} \tag{1.1b}$$

$$d(x, y) \geq 0, \qquad \text{(Non-negativity)} \tag{1.1c}$$

$$d(x, y) = d(y, x), \qquad \text{(Symmetry)} \tag{1.1d}$$

$$d(x, z) \leq d(x, y) + d(y, z). \qquad \text{(The triangle inequality)} \tag{1.1e}$$

Among these properties, the triangle inequality (1.1e) might be the most useful. It can be intuitively understood as *"the direct path from the data point x to z is the shortest path* between the two data points." The triangle inequality enables distances to be bounded without directly computing them, a property that is essential in many neighbor search algorithms as it reduces search space [Schubert, 2021]. Combined with the other properties, distance function can represent how similar data points are.

Can we find distances that match semantic dissimilarity? Distance-like functions, which satisfy most of the properties (1.1) of distance, can be found by learning feature extracting functions from data. In machine learning, this process is known as *metric learning* and is explained in Chapter 2. I also explore how the performance of metric learning models can be affected by (extracted) feature space. In addition to metric learning, to further examine the importance of distance in machine learning, I also explore other fields of machine learning.

It has been shown that common deep neural network (DNN)-based classifiers are vulnerable to adversarially perturbed data points (input images), known as *adversarial examples* [Szegedy et al., 2013]. In human eyes, image adversarial examples are often indistinguishable from the original data points. The susceptibility of common machine learning models is immensely concerning in areas where safety is crucial, such as autonomous

driving and face verification systems. Hence, the existence of adversarial examples poses a significant obstacle to the widespread adoption of machine learning models, as they can be exploited by malicious attackers. Distance serves as a key component in the formal definition of adversarial examples and adversarially robust classification. In Chapter 3, I delve into this field by examining adversarial attacks and robust models.

Often, classes share common characteristics with others. In other words, some classes are more similar than other classes. Hierarchical structures can represent these relationships, which is especially important in classification of biological and library data. However, standard classification models overlook these relationships. By contrast, *hierarchy-informed classification* models handle existing hierarchical relationships of classes by incorporating class hierarchy into the training process. Distance is also utilized through the concept of *hierarchical class distance*, which denotes the hierarchical distance between classes. The field of hierarchy-informed classification is explored in detail in Chapter 4.

In Chapter 5, I investigate the influence of adversarial training (popular adversarial defense method) in feature space and suggest possible effects of distances in the feature spaces of robust classifiers. In Chapter 6, I explain why distances are important in the explained fields and suggest possible future works. Research works are presented in Chapter 7.

1.1 General setting and terminology

Let $\mathcal{X}$ be an input space with $\mathcal{X} \subset \mathbb{R}^{d_I}$ where d_I is the dimension of the input space. Let $\mathcal{Y}$ be a set of possible classes. From data distribution $\mathcal{D}$, data point and class pairs (x, c) are sampled for $x \in \mathcal{X}$ and $c \in \mathcal{Y}$. There is a feature mapping function $f : \mathcal{X} \longrightarrow \mathcal{Z}$ where $\mathcal{Z} \subset \mathbb{R}^{d_F}$ is a feature space (d_F is dimension of the feature space). Unless otherwise specified, I assume this function is modeled by a DNN. Using this feature mapping $f(\cdot)$, a classification model outputs logit value $l_y(x)$ for $y \in \mathcal{Y}$ (I denote an arbitrary class as y). The standard calculation for logit values can be expressed as:

$$l_y(x) = W_y^T f(x) + b_y,$$

where W is a weight matrix, W_y is a weight vector from the matrix W, and b_y is a bias term for class y. These logit values can be used for estimating class probability $p(y|x)$

by adopting the following softmax formulation:

$$\hat{p}(y|x) = \frac{e^{l_y(x)}}{\sum\limits_{y' \in \mathcal{Y}} e^{l_{y'}(x)}}, \tag{1.2}$$

where $\hat{p}(y|x)$ is an estimated class probability (I denote estimated probability as $\hat{p}$ to contrast it with exact probability p). For each data point x, it is possible to obtain a predicted class $\hat{c}$ using estimated probabilities by finding a class that maximizes estimated class probability $\hat{p}(y|x)$; in other words, by setting $\hat{c} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}(\hat{p}(y|x))$. To train a classifier, cross-entropy (CE) loss is commonly used as an objective function. When $\log(\cdot)$ is the element-wise logarithm function, the point-wise cross-entropy loss between two probability vectors P and Q is defined as:

$$\begin{aligned} \ell_{\mathrm{CE}}(P, Q) &= -P \cdot \log(Q) && \text{(Using dot product)} \\ &= -\sum_{k=1}^{|\mathcal{Y}|} P_k \log(Q_k), && \text{(Using summation)} \end{aligned} \tag{1.3}$$

where P_k and Q_k represent the k-th element of probability vectors P and Q, respectively. Let $\overrightarrow{\mathbb{1}}_y$ be the one-hot vector whose element has the value of 1 for the corresponding element of y. Let $\hat{P}(\mathcal{Y}|x)$ be the estimated class probability vector of data point x; hence, its corresponding element of index for class y has value $\hat{p}(y|x)$. The vector $\hat{P}(\mathcal{Y}|x)$ represents the softmax outputs of data point x. Accordingly, the cross-entropy loss of a mini-batch $B \subset \mathcal{D}$ is defined as:

$$L_{\mathrm{CE}} = \frac{1}{|B|} \sum_{(x,c) \in B} \ell_{\mathrm{CE}}(\overrightarrow{\mathbb{1}}_c, \hat{P}(\mathcal{Y}|x)). \tag{1.4}$$

Taking into consideration the definition of vectors and CE loss (1.3), the loss can be simplified as:

$$L_{\mathrm{CE}} = -\frac{1}{|B|} \sum_{(x,c) \in B} \log(\hat{p}(c|x)). \tag{1.5}$$

Chapter 2

Metric learning

In the introduction (Chapter 1), I posed the question: *Can we find distances that match semantic dissimilarity?* In this chapter, I explore *metric learning* to answer this question. Metric learning is a field of machine learning that learns a feature mapping (embedding function) $f(\cdot)$ that maps similar data points to be close and dissimilar data points to be far apart on the embedding space, also known as the feature space (or representation space). Once embedding function is learned, distance on the embedding space can be calculated using a distance function $d(\cdot, \cdot)$.

Using the calculated distance from the embedding space, a function $d_{\text{learned}}(\cdot, \cdot)$ can be defined as:

$$d_{\text{learned}}(x_1, x_2) := d(f(x_1), f(x_2)),$$

for any two data points x_1 and x_2. One can verify that the function $d_{\text{learned}}(\cdot, \cdot)$ satisfies most properties of distance in definition (1.1) except for the "identity of indiscernibles" property (1.1a) (such a function is known as *pseudometric* in mathematics). When the feature mapping $f(\cdot)$ (and further modifications) is invertible, the function $d_{\text{learned}}(\cdot, \cdot)$ satisfies all properties for distance. Hence, the function $d_{\text{learned}}(\cdot, \cdot)$ can be considered a learned dissimilarity between data points. In other words, *metric learning is a field that involves learning a distance(-like) function.* Although it is a slight abuse of terminology, for the sake of simplicity, I refer to this function $d_{\text{learned}}(\cdot, \cdot)$ as *distance*.

What are the applications of learned distances? In other words, how can metric learning be utilized? It can be used for few-shot classification, where a model needs to classify a query data point from a new class based on a limited number of support points (examples) only for each new class [Snell et al., 2017]. It can also be used for information retrieval, where a model needs to find the most similar data points for a query data point [Musgrave

et al., 2020]. Based on the assumption that augmented data points will be similar to the original data point, some concepts in metric learning methods have applied in self-supervised learning [Chen et al., 2020b], which refers to representation learning in unsupervised settings.

In sections 2.1 and 2.2, I explore two different approaches for metric learning. Section 2.3 describes three popular spaces used in metric learning, while in section 2.4, I describe how discriminative power and dimensionality of feature space can affect metric learning performance. Finally, in section 2.5, I describe common formulations used for the estimation of class probability in some metric learning models.

2.1 Embedding-based metric learning

Embedding-based metric learning methods [Hadsell et al., 2006; Wang et al., 2014] use direct comparison between embeddings of pairs or triplets of data points.

The contrastive training [Hadsell et al., 2006] uses pairs of data points for metric learning. To distinguish classes used in training and testing phases, I denote the set of classes used in training phase as $\mathcal{Y}_{\text{train}} \subset \mathcal{Y}$. For a data point pair (x_i, x_j) and the corresponding class pair (c_i, c_j) (where $c_i, c_j \in \mathcal{Y}_{\text{train}}$), contrastive loss $\ell_{\text{Contrastive}}$ is defined as:

$$\ell_{\text{Contrastive}} = \mathbb{1}(c_i = c_j) \cdot \frac{1}{2} d(f(x_i), f(x_j))^2 + \mathbb{1}(c_i \neq c_j) \cdot \frac{1}{2} \text{ReLU}(m - d(f(x_i), f(x_j)))^2,$$

where $\mathbb{1}(\cdot)$ is the indicator function, $m > 0$ is a margin parameter, and $\text{ReLU}(v) = max(0, v)$ for a value v. When data points x_i and x_j have the same class (positive pair), the loss tries to reduce their distance in the embedding space. In the converse scenario (negative pair), the loss tries to increase their distance in the embedding space up to the margin m.

The triplet training [Wang et al., 2014] selects triplets such that each triplet (x_a, x_p, x_n) has an anchor point x_a, a positive point x_p with the same class as the anchor x_a, and a negative point x_n with a different class from the anchor x_a. Given a triplet (x_a, x_p, x_n), triplet loss ℓ_{Triplet} is defined as:

$$\ell_{\text{Triplet}} = \text{ReLU}(d(f(x_a), f(x_p)) - d(f(x_a), f(x_n)) + m).$$

This loss tries to make the anchor-positive distance $d(f(x_a), f(x_p))$ smaller than the anchor-negative distance $d(f(x_a), f(x_n))$.

One drawback of embedding-based metric learning methods is that the complexity of training is high. Specifically, if we let N be the number of training data points, then the training complexity of contrastive training is $\mathcal{O}(N^2)$ and that of triplet training is $\mathcal{O}(N^3)$. The high training complexity requires special sample mining algorithms [Schroff et al., 2015; Wang et al., 2017]; otherwise their training speed will be slow [Kim et al., 2020]. Unlike embedding-based metric learning with high training complexity, classification-based metric learning methods have $\mathcal{O}(N)$ training complexity. This reduced complexity is made possible by using one class representative [Deng et al., 2019; Liu et al., 2017; Wang et al., 2017, 2018; Zhai and Wu, 2019] or multiple class representatives [Deng et al., 2020; Qian et al., 2019] for each class.

2.2 Classification-based metric learning

Before considering classification-based metric learning models, it is useful to recall the usual setting of a standard softmax classifier. Once a feature mapping $f(\cdot)$ is modeled by a DNN, the softmax classifier models logit value $l_y(x)$ for a data point x and a class $y \in \mathcal{Y}_{\text{train}}$ as:

$$l_y(x) = W_y^T f(x) + b_y, \tag{2.1}$$

where W is a learnable weight matrix, W_y is a weight vector from the matrix W, and b_y is a bias term for class y. From logit values, one can estimate class probability $p(y|x)$ using the softmax formulation (1.2) and then train the model using the cross-entropy loss in the equation (1.5).

NormFace [Wang et al., 2017], also known as normalized softmax [Zhai and Wu, 2019], is one of the simplest classification-based metric learning models. Unlike the softmax classifier that estimates logit value $l_y(x)$ as the expression (2.1), it uses a normalized space (spherical space) and zero bias terms. Specifically, let us denote a normalized vector of a non-zero vector v as $\tilde{v}$, formalized as $\tilde{v} = \frac{v}{||v||}$. Denoting the angle between $\tilde{W}_y$ and $\tilde{f}(x)$ as θ_y then yields an equation with cosine similarity:

$$\tilde{W}_y^T \tilde{f}(x) = \left\|\tilde{W}_y\right\| \left\|\tilde{f}(x)\right\| \cos(\theta_y) = \cos(\theta_y).$$

NormFace models logit value $l_y(x)$ as:

$$l_y(x) = s\tilde{W}_y^T \tilde{f}(x) = s\cos(\theta_y), \tag{2.2}$$

where $s > 0$ is a scaling parameter. From this, estimated class probability $p(y|x)$ using

NormFace is calculated as:

$$\hat{p}(y|x) = \frac{e^{s\cos(\theta_y)}}{\sum_{y' \in \mathcal{Y}_{\text{train}}} e^{s\cos(\theta_{y'})}}. \tag{2.3}$$

Following this, cross-entropy loss for NormFace is calculated as:

$$L_{\text{NormFace}} = -\frac{1}{|B|} \sum_{(x,c)\in B} \log\left(\frac{e^{s\cos(\theta_c)}}{\sum_{y' \in \mathcal{Y}_{\text{train}}} e^{s\cos(\theta_{y'})}}\right),$$

where $B \subset \mathcal{D}$ is a mini-batch.

From the NormFace model [Wang et al., 2017], it is common practice to use a class $\hat{c}$ that maximizes estimated class probability $\hat{p}(y|x)$ as a predicted class. Because softmax function (1.2) is an increasing function with respect to logits, it is equivalent to using a class $\hat{c}$ that maximizes the logit value. From equation (2.2), we can express this as $l_{\hat{c}}(x) = s\cos(\theta_{\hat{c}}) \geq s\cos(\theta_y) = l_y(x)$ for all $y \in \mathcal{Y}_{\text{train}}$. Because cosine is a decreasing function within the interval $[0, \pi]$, we obtain an equivalent expression for the decision boundary: $\theta_{\hat{c}} \leq \theta_y$ for all $y \in \mathcal{Y}_{\text{train}}$. This means that NormFace uses angles between normalized feature $\tilde{f}(x)$ and normalized weight vectors $\tilde{W}_y$ for classification of a data point x, and takes the class $\hat{c}$ with the closest (normalized) weight vector $\tilde{W}_{\hat{c}}$ as a predicted class. Thus, we can consider a normalized weight vector $\tilde{W}_y$ as a class representative of class y. A visualized example is presented in Figure 2.1.

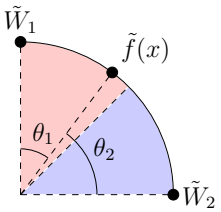


Figure 2.1: Visualization of NormFace classification results for two classes with $\tilde{W}_1$=(0,1) and $\tilde{W}_2$=(1,0). Because $\tilde{f}(x)$ is closer to $\tilde{W}_1$ than $\tilde{W}_2$, data point x is classified as belonging to class 1.

In classification-based metric learning, a class representative can be modeled by a learnable weight vector or an average position (or a normalized average position) of each class in an embedding space. To distinguish them, I call the former class representative (a learnable weight vector) a *proxy* and the latter (an average position of each class) a *prototype*. In the NormFace model [Wang et al., 2017], normalized weight vectors $\tilde{W}_y$ are proxies. These are typically used in the training process of classification-based metric

learning. A prototype is an average embedding of each class (or normalized average embedding for spherical embedding space). Let us denote the average embedding of class y as μ_y. Mathematically, μ_y is defined as:

$$\mu_y = \frac{1}{|X_y|} \sum_{x \in X_y} f(x), \tag{2.4}$$

where $X_y \subset \mathcal{X}$ is the set of data points belonging to class y and $\tilde{f}(x)$ is used instead of $f(x)$ for normalized embedding space. Then, μ_y is a prototype of class y (or $\tilde{\mu}_y = \frac{\mu_y}{\|\mu_y\|}$ is a prototype for normalized embedding).

There are variants of NormFace [Wang et al., 2017] that increase the separation of classes by introducing margins. CosFace [Wang et al., 2018] introduces a cosine margin. CosFace uses a new loss called large margin cosine loss (LMCL). It is defined as:

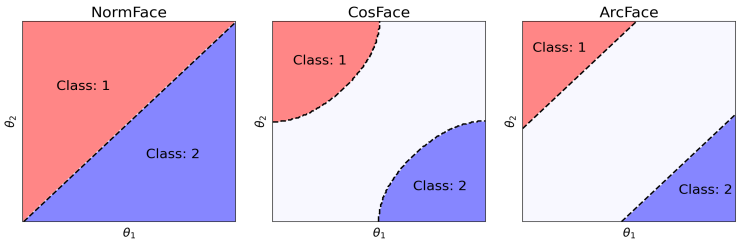
$$L_{\text{LMC}} = -\frac{1}{|B|} \sum_{(x,c) \in B} \log\left(\frac{e^{s(\cos(\theta_c) - m)}}{e^{s(\cos(\theta_c) - m)} + \sum\limits_{y' \in \mathcal{Y}_{\text{train}} - \{c\}} e^{s \cos(\theta_{y'})}}\right).$$

The decision boundary of CosFace becomes $\cos(\theta_c) \geq \cos(\theta_{y'}) + m$ for all $y' \in \mathcal{Y}_{\text{train}} - \{c\}$. Unlike NormFace [Wang et al., 2017], which is sufficient to be classified as the correct class c when cosine similarity is larger (or equal) than other classes, CosFace requires much larger (by the margin m) cosine similarity for class c.

ArcFace [Deng et al., 2019] introduces an angular margin. Its training loss is defined as:

$$L_{\text{ArcFace}} = -\frac{1}{|B|} \sum_{(x,c) \in B} \log\left(\frac{e^{s \cos(\theta_c + m)}}{e^{s \cos(\theta_c + m)} + \sum\limits_{y' \in \mathcal{Y}_{\text{train}} - \{c\}} e^{s \cos(\theta_{y'})}}\right).$$

The decision boundary of ArcFace becomes $\cos(\theta_c + m) \geq \cos(\theta_{y'})$ for all $y' \in \mathcal{Y}_{\text{train}} - \{c\}$ or, more concisely, $\theta_c \leq \theta_{y'} - m$ for all $y' \in \mathcal{Y}_{\text{train}} - \{c\}$. To be classified as the correct class c, ArcFace requires much smaller (by the margin m) angles for class c than NormFace [Wang et al., 2017]. Figure 2.2 compares the decision boundaries of the three models for a binary classification scenario.



(a) (b) (c)

Figure 2.2: Decision boundaries for different models with margin $m = 1$ when there are only two classes (class 1 and class 2). (a): NormFace model. (b) CosFace model. (c): ArcFace model.

Other attempts have been made to improve NormFace [Wang et al., 2017] without using margins. For instance, Zhe et al. [2019] found that the standard gradient update of the NormFace model [Wang et al., 2017] can cause an unstable update of proxy positions as their gradients are affected by data points from different classes. To handle this, they suggested using exponential moving average (EMA) with normalization for proxy updates. For a data point x that belongs to class c, the proxy $\tilde{W}_c$ is updated as:

$$\tilde{W}_c = \frac{\alpha \tilde{f}(x) + (1-\alpha)\tilde{W}_c}{\|\alpha \tilde{f}(x) + (1-\alpha)\tilde{W}_c\|},$$

where $0 < \alpha < 1$ is a parameter that controls the stability and speed of updates.

Through an analysis of scaling factor s and margin parameter m in the ArcFace model [Deng et al., 2019], Zhang et al. [2019b] found that it is possible to control the effects of both hyperparameters with parameter s only. To address the difficulty of hyperparameter tuning in achieving optimal performance, they proposed AdaCos, which is a NormFace model [Wang et al., 2017] trained by the adaptive scaling factor s. Specifically, they suggested choosing parameter s, which significantly changes the estimation of probability $p(c|x)$ where c is the class of the data point x. Mathematically, they attempted to use a scaling factor s that maximizes $\left\| \frac{\partial p(c|x)(\theta_c)}{\partial \theta_c} \right\|$, which can be found by approximating the equation:

$$\frac{\partial^2 p(c|x)(\theta'_c)}{\partial \theta'^2_c} = 0, \tag{2.5}$$

where $\theta'_c := \text{clip}(\theta_c, \left[0, \frac{\pi}{2}\right])$ and $\text{clip}(\cdot, \cdot)$ is a function that limits a value within a specified range.

2.3 Space for embedding

Euclidean space $\mathbb{R}^{d_F}$ (d_F is the dimension of the feature space) is likely to be the simplest space for metric learning. Its simplicity enables straightforward arithmetic operations and average calculations. The standard PCA (principal component analysis) can be used for analyzing Euclidean embedding spaces, unlike non-Euclidean spaces that require more complex methods for proper analysis.

While Euclidean space $\mathbb{R}^{d_F}$ is the most familiar space, non-Euclidean spaces can be used for metric learning. As explained in section 2.2, spherical space $\mathbb{S}^{d_F-1} = \left\{z \in \mathbb{R}^{d_F} | \, \|z\| = 1\right\}$ is a popular non-Euclidean space for embedding space [Barz and Denzler, 2019; Deng et al., 2019; Wang et al., 2017, 2018] as it exhibits improved performance over Euclidean space. In practice, unnormalized feature $f(x) \in \mathbb{R}^{d_F}$ is normalized for numerical stability as follows:

$$\tilde{f}(x) = \frac{f(x)}{||f(x)|| + \epsilon_{\text{stab}}},$$

where ϵ_{stab} is a positive value close to zero. When ϵ_{stab} is ignored, the normalized feature $\tilde{f}(x)$ belongs to $\mathbb{S}^{d_F-1}$. Hence, spherical space $\mathbb{S}^{d_F-1}$ is also called normalized space. Euclidean distance and angular distance between features are metric distances in the spherical space. The Euclidean distance is commonly used in metric learning with hyperspheres [Deng et al., 2019; Wang et al., 2017, 2018; Zhang et al., 2019b].

Recently, hyperbolic spaces have also been used in metric learning. While there are several models for hyperbolic spaces, Poincaré ball model $\mathbb{D}_\tau^{d_F} = \left\{z \in \mathbb{R}^{d_F} | \tau \|z\|^2 < 1\right\}$, where $\tau \geq 0$ is a curvature parameter, is a popular space employed in metric learning [Ermolov et al., 2022; Khrulkov et al., 2020; Yan et al., 2021]. Because it is not a vector space, it requires so-called *gyrovector addition* to replace the vector addition. For two *gyrovectors* $v_1, v_2 \in \mathbb{D}_\tau^{d_F}$, gyrovector addition is defined as:

$$v_1 \oplus_\tau v_2 = \frac{\left(1 + 2\tau \left\langle v_1, v_2 \right\rangle + \tau \left\|v_2\right\|^2\right) v_1 + \left(1 - \tau \left\|v_1\right\|^2\right) v_2}{1 + 2\tau \left\langle v_1, v_2 \right\rangle + \tau^2 \left\|v_1\right\|^2 \left\|v_2\right\|^2}.$$

Using this addition, (geodesic) distance between $v_1, v_2 \in \mathbb{D}_\tau^{d_F}$ is defined as:

$$d(v_1, v_2) = \frac{2}{\sqrt{\tau}} \operatorname{arctanh}\left(\sqrt{\tau} \left\|-v_1 \oplus_\tau v_2\right\|\right).$$

To use this space for metric learning, we need to map vectors from Euclidean space $\mathbb{R}^{d_F}$

into the Poincaré model $\mathbb{D}_\tau^{d_F}$. This so-called *exponential mapping* is defined as:

$$\exp_{\mathbf{x}}^{\tau}(v) = \mathbf{x} \oplus_\tau \left(\tanh\left(\sqrt{\tau} \frac{\lambda_{\mathbf{x}}^{\tau} \|v\|}{2} \right) \frac{v}{\sqrt{\tau}\|v\|} \right)$$

where $\mathbf{x} \in \mathbb{D}_\tau^{d_F}$ is a base point and $\lambda_{\mathbf{x}}^{\tau}$ is a conformal factor defined as $\lambda_{\mathbf{x}}^{\tau} = \frac{2}{1-\tau\|\mathbf{x}\|^2}$. The base point is commonly set to the zero vector $\overrightarrow{0}$.

Using hyperbolic space can be beneficial in two ways. First, Khrulkov et al. [2020] found that unclear data points and data points from new datasets are mapped close to the center, while clearer samples are mapped close to the boundary of the ball. Hence, when the clarity and quality of points in the dataset vary, this can be handled using hyperbolic embedding. This is not possible in spherical embedding that nullifies magnitudes by normalization. Second, tree structures can be mapped into a two-dimensional Poincaré model $\mathbb{D}_\tau^2$ with arbitrarily low distortion [Sarkar, 2012]. Additionally, as explained by Sala et al. [2018], let us consider three points $\overrightarrow{0}, v_1, v_2 \in \mathbb{D}_\tau^2$ with $\|v_1\| = \|v_2\| = t$. As we approach t to the maximal norm $\frac{1}{\sqrt{\tau}}$, the distance $d(v_1, v_2)$ approaches $d(v_1, \overrightarrow{0}) + d(\overrightarrow{0}, v_2)$. This is similar to the property of the trees. That is, "the shortest path between two nodes is the path through their parent". Taking account of both the tree-approximating capability and the tree-like property, the Poincaré ball model can be advantageous in handling the underlying hierarchical tree structures of datasets.

2.4 Discriminative power of embeddings

Several approaches, including the use of margins [Deng et al., 2019; Wang et al., 2018] and an adaptive scaling factor [Zhang et al., 2019b], strive to increase separation between the data points belonging to different classes. Popular usage of these methods has given rise to the following questions. *How can we measure the discriminative power (discriminality) of a learned representation? What are the effects of discriminality on representations? Moreover, is it always beneficial to have higher discriminality?*

Liu et al. [2020] studied the CosFace model [Wang et al., 2018] on few-shot classification tasks by varying the margin parameter m. They defined the discriminative function $\phi(m)$ to measure discriminative power for spherical embeddings. This is based on the inter-class variance $D_{\text{inter}}(m)$ and the intra-class variance $D_{\text{intra}}(m)$. Let us denote the class set of interest as $\mathcal{Y}_I \subset \mathcal{Y}$ and the set of data points for the j th class in $\mathcal{Y}_I$ as X_j. When feature space is learned with margin m, we denote the embedding of a data point x as $\tilde{f}_m(x)$ and the average embedding for the j th class as $\mu_j(m)$. Following this, the

variances $D_{\text{inter}}(m)$ and $D_{\text{intra}}(m)$ are defined as:

$$D_{\text{inter}}(m) = \frac{1}{|\mathcal{Y}_I|(|\mathcal{Y}_I|-1)} \sum_{j=1}^{|\mathcal{Y}_I|-1} \sum_{k=1,k\neq j}^{|\mathcal{Y}_I|} \|\mu_j(m) - \mu_k(m)\|^2,$$

$$D_{\text{intra}}(m) = \frac{1}{|\mathcal{Y}_I|} \sum_{j=1}^{|\mathcal{Y}_I|} \frac{1}{|X_j|} \sum_{x_i \in X_j} \|\tilde{f}_m(x_i) - \mu_j(m)\|^2. \tag{2.6}$$

Discriminative function $\phi(m)$ is the ratio between the two variances and is formulated as:

$$\phi(m) = \frac{D_{\text{inter}}(m)}{D_{\text{intra}}(m)}. \tag{2.7}$$

With varying margin values, Liu et al. [2020] assessed the changes of variances and the discriminative function on training classes, also known as *base classes*, and unseen classes during the training phase, which are often referred to as *novel classes*. They found that as the margin increased, discriminative (function) value $\phi(m)$ increased for base classes. By contrast, the discriminative value decreased for novel classes. This was due to the increase of intra-class variance $D_{\text{intra}}(m)$ while inter-class variance $D_{\text{inter}}(m)$ changed little for novel classes. Similarly, as they increased margin parameter m, few-shot accuracy increased for base classes while accuracy decreased for novel classes. From these observations, they suggested using a CosFace model [Wang et al., 2018] with negative margin m for few-shot classification.

Liu et al. [2020] explained their observations by visualizing data distributions. With a larger margin, data points will be more closely located to the prototypes of the base classes in the embedding space. When data points from the same novel class are mapped closely to multiple base prototypes (peaks), their intra-class variance will increase.

Roth et al. [2020] examined various metric learning models using metric learning performance measures. They defined an assessment measure π_{ratio} named "embedding space density", which is similar to the reciprocal of the discriminative function [Liu et al., 2020]. To avoid confusion, I express the measure with the mathematical symbol π_{ratio}. This measure is also based on two values: average inter-class distance π_{inter} and average intra-class distance π_{intra}. These are defined as:

$$\pi_{\text{inter}} = \frac{1}{Z_{\text{inter}}} \sum_{k,l,k\neq l} d(\mu_k, \mu_l),$$

$$\pi_{\text{intra}} = \frac{1}{Z_{\text{intra}}} \sum_{k=1}^{|\mathcal{Y}_I|} \sum_{x_i, x_j \in X_k} d(\tilde{f}(x_i), \tilde{f}(x_j)),$$

where Z_{inter} and Z_{intra} are the normalization constants. These constants can be expressed as the equations $Z_{\text{inter}} = |\mathcal{Y}_I|(|\mathcal{Y}_I|-1)$ and $Z_{\text{intra}} = \sum_{k=1}^{|\mathcal{Y}_I|} |X_k|(|X_k|-1)$. (Note that although Roth et al. used a different formulation of π_{intra} for the equation presented in their paper and for their code, their implementation of average intra-class distance π_{intra} used class-wise normalization instead of only applying normalization at the end.) The measure π_{ratio} is the ratio between two distances and is formulated as:

$$\pi_{\text{ratio}} = \frac{\pi_{\text{intra}}}{\pi_{\text{inter}}}. \tag{2.8}$$

They used cosine distance on embedding for the distance function $d(\cdot, \cdot)$. (It is not actually a distance as it violates the triangle inequality [1.1e]. The square root of the cosine distance is distance.)

By splitting data classes into training and test classes, Roth et al. [2020] assessed the value of the π_{ratio} on training data and evaluated test class performances for three datasets: CUB200-2011 [Wah et al., 2011], CARS196 [Krause et al., 2013], and Stanford Online Products (SOP) [Oh Song et al., 2016]. They found that π_{ratio} is positively correlated (CUB200-2011: 0.79, CARS196: 0.65, SOP: 0.22) with generalization on novel classes. Because π_{ratio} is approximately the reciprocal of the discriminative function and evaluated performances are affected by the discriminative power on test classes, this finding is similar to the observations of Liu et al. [2020].

In addition to π_{ratio}, Roth et al. [2020] defined a measure ρ to assess the spectral decay of embedding on training data. Its value is defined as:

$$\rho = \text{KL}(\mathcal{U}_{d_F} || \mathcal{S}), \tag{2.9}$$

where $\text{KL}(\cdot||\cdot)$ is the Kullback–Leibler divergence, $\mathcal{U}_{d_F}$ is d_F-dimensional discrete uniform distribution, and $\mathcal{S}$ is the normalized spectrum of singular values (SV) sorted in descending order. A small ρ value means that learned embedding contains many *"directions of significant variance* (DoV)" as less flattening of representation occurs. In other words, value ρ can measure compression in feature representation.

Using the same three datasets, Roth et al. [2020] studied the relation between ρ values and metric learning performances. They found that they are negatively correlated (CUB200-2011: -0.80, CARS196: -0.85, SOP: -0.63). This indicates that having more DoV is beneficial in terms of generalization power as there can be considerable distribu-

tion shifts in test data, which will be negatively impacted by strong compression.

Based on the results, the authors proposed a ρ-regularization for embedding-based metric learning. Specifically, they suggested randomly switching negative data points with the positive data points in their loss. The goal was to avoid excessively strong compression during training. They experimentally demonstrated that their regularization can reduce the ρ value and improve recall performance.

Kornblith et al. [2021] studied the effects of different training losses on the transferability of representations. They trained 8 models with different losses on the ImageNet ILSVRC 2012 dataset [Deng et al., 2009; Russakovsky et al., 2015]. Next, they evaluated model accuracy on the ImageNet test set and different datasets to study the transferability of representations. They proposed a value R^2 to measure class separation in embedding spaces. This is based on the average within-class cosine distance $\bar{d}_{\text{within}}$ and the overall average cosine distance $\bar{d}_{\text{total}}$. These values are defined as:

$$\bar{d}_{\text{within}} = \sum_{k=1}^{|\mathcal{Y}_I|} \sum_{x_i, x_j \in X_k} \frac{1 - \text{sim}(f(x_i), f(x_j))}{|\mathcal{Y}_I||X_k|^2}, \tag{2.10}$$

$$\bar{d}_{\text{total}} = \sum_{k=1}^{|\mathcal{Y}_I|} \sum_{l=1}^{|\mathcal{Y}_I|} \sum_{x_i \in X_k,\, x_j \in X_l} \frac{1 - \text{sim}(f(x_i), f(x_j))}{|\mathcal{Y}_I|^2 |X_k||X_l|}, \tag{2.11}$$

where $\text{sim}(\cdot, \cdot)$ is cosine similarity, formalized as $\text{sim}(v_1, v_2) = \tilde{v}_1 \cdot \tilde{v}_2$. Class separation $R^2 \in [0, 1]$ is then defined as:

$$R^2 = 1 - \frac{\bar{d}_{\text{within}}}{\bar{d}_{\text{total}}}.$$

Kornblith et al. [2021] observed that models that achieved higher accuracy on ImageNet test data also exhibit less transfer accuracy. When they used a modified model of the NormFace [Wang et al., 2017] with varying scaling factor s, they found that class separation R^2 has a generally positive relationship with (test) accuracy on ImageNet data. Conversely, they reported a negative correlation (rank correlation: -0.93, p value: 0.002) between class separation and transfer accuracy. In subsequent experiments, they speculated that representations with higher class separation would overfit the training classes but not necessarily training data points. Hence, the authors explained that these representations perform poorly in classifying new classes even though they can achieve high test accuracy on trained classes.

Overall, the three papers [Kornblith et al., 2021; Liu et al., 2020; Roth et al., 2020]

indicate that more discriminative models increase performance in terms of classifying new data points from the trained classes. However, these models can reduce DoV due to excessive feature compression. Alternatively, they can map data points from the same novel class into multiple prototypes. In all cases, more discriminative models can achieve lower performances on novel classes and datasets. Therefore, recklessly increasing the discriminative powers of models can be detrimental when it comes to new tasks with significantly shifted distributions. Unless these issues are managed properly, caution is needed when increasing discriminative powers.

2.5 Estimating class probability in metric learning

A large number of metric learning models [Deng et al., 2019; Goldberger et al., 2004; Snell et al., 2017; Wang et al., 2017, 2018] use estimation of class probabilities. These include models that utilize a classification loss such as cross-entropy during training. These approaches are beneficial when the certainty of classification needs to be estimated.

The most commonly used estimation formulation is to apply *negative of squared distance* (SD) on softmax formulation (1.2), which I term "SD-softmax formulation". Mathematically, the estimated class probability of $p(y|x)$ is defined as:

$$\hat{p}(y|x) = \frac{e^{-sd_{y,x}^2}}{\sum_{y' \in \mathcal{Y}} e^{-sd_{y',x}^2}}, \tag{2.12}$$

where s is a scaling factor and $d_{y,x}$ is distance from the data point x to class y. For instance, NormFace [Wang et al., 2017] is equivalent to using the formulation (2.12) on normalized embedding by defining the distance $d_{y,x}$ with proxy representatives. Prototypical networks [Snell et al., 2017] also use the formulation (2.12) by defining the distance $d_{y,x}$ with local prototypes.

An alternative estimation is to apply the negative of (non-squared) distance on softmax formulation (1.2), as suggested by Garnot and Landrieu [2021]. This formulation is defined as:

$$\hat{p}(y|x) = \frac{e^{-sd_{y,x}}}{\sum_{y' \in \mathcal{Y}} e^{-sd_{y',x}}}. \tag{2.13}$$

The authors observed that using formulation (2.13) achieved significantly better results than SD-softmax formulation.

In paper I, the first author proposed using metric learning rather than the standard softmax classifier for a challenging classification problem: plankton classification. The number of classes in plankton classification is often large. Moreover, the choice of plankton categories might differ between researchers. The fact that softmax outputs are only available for training classes can be problematic when classifying unseen classes for different plankton datasets. On the other hand, because metric learning learns distance function, it can be directly employed for classifying novel classes by using support points for each novel class to estimate class prototypes. Our experiment revealed that using triplet training [Wang et al., 2014] achieved comparable test accuracy on trained classes and moderate accuracy in classifying novel classes.

Paper II was motivated by the analyzed limitations of the SD-softmax formulation (2.12). That is, 1) *training loss based on the formulation is dependent on the scale of embedding space* and 2) *estimated class probabilities* $\hat{p}(y|x)$ *are not maximized at the class representatives*. To manage these limitations, I proposed an alternative formulation named *distance-ratio-based* (DR) *formulation* for metric learning.

Theoretically, using DR formulation will avoid scale change during training process. More precisely, we can avoid unnecessary model updates that do not change relative locations in the embedding space. Additionally, using DR formulation will be advantageous when proxy representatives are used for metric learning because data points can easily converge to corresponding proxies that maximize probabilities $\hat{p}(y|x)$. In experiments with Euclidean space embedding, using DR formulation yielded generally faster training and improved performance on few-shot classification tasks.

Chapter 3

Adversarial robustness

Over the last decade, deep neural networks (DNNs) have generated promising results and been successfully applied in various tasks [Gatys et al., 2015; Jumper et al., 2021; OpenAI, 2022; Ramesh et al., 2022]. For instance, a convolutional neural network (CNN) model [He et al., 2015] surpassed human level top-5 classification accuracy (94.9%) [Russakovsky et al., 2015] on the ImageNet 2012 [Deng et al., 2009] classification dataset. However, DNNs have also been shown to be vulnerable [Szegedy et al., 2013] to adversarial examples, which are defined as examples carefully modified from the original examples with (commonly) only imperceptible changes. The existence of adversarial examples enables malicious attacks to exploit the vulnerability of models. For instance, the susceptibilities of automatic driving [Kong et al., 2020] and face-recognition systems [Sharif et al., 2016] raise concerns regarding the adoption of machine learning models. Mathematically, when we have a distance metric $d(\cdot, \cdot)$, a target classifier $\mathcal{C} : \mathcal{X} \longrightarrow \mathcal{Y}$ and an original data point x, an adversarial example is defined as a data point x' that satisfies:

$$\mathcal{C}(x') \neq \mathcal{C}(x) \tag{3.1}$$

where $d(x, x') \leq \epsilon$ for perturbation budget ϵ [Biggio et al., 2013]. To only allow imperceptible changes, the budget ϵ is often set to be small. l_2 norm and l_∞ norm are commonly used for the distance metric $d(\cdot, \cdot)$. However, to handle the limitation of l_p norms explained in the introduction (Chapter 1), one can also use different distances like Wasserstein distance [Wong et al., 2019] and Learned Perceptual Image Patch Similarity (LPIPS) [Laidlaw et al., 2021; Zhang et al., 2018].

In section 3.1, I explore various adversarial attack algorithms. The adversarial accuracy of classification models and defense methods against adversarial attacks are described in

section 3.2. In section 3.3, I explore the crucial issue of adversarially robust classifiers that often have reduced natural accuracy compared to standard classifiers. Hence, there is *a tradeoff between natural accuracy and adversarial accuracy.* I then describe several suggested hypotheses to explain the tradeoff.

3.1 Adversarial attacks

Depending on the accessibility of attackers on a target model, adversarial attacks can be divided into white-box attacks and black-box attacks [Chen et al., 2020a]. A white-box attack occurs when an attacker has access to all of the target model, including model architecture and weights. Because the gradient with respect to the input is accessible for a white-box attack case, an attacker can use gradient-based attacks to generate adversarial examples. More common scenarios are when an attacker has only limited access to the target model. The attack method used in these cases is called a black-box attack. As classification models output either a predicted class $\hat{c}$ alone or a predicted class probability (score) vector $\hat{P}(\mathcal{Y}|\cdot)$, black-box attacks can be further divided into decision-based and score-based attacks. The common approach for a decision-based attack is to train a surrogate model and use white-box attack on the surrogate model as an attack against the target model [Papernot et al., 2017]. Chen et al. [2020a] proposed a different approach for a decision-based attack using gradient-direction estimation and a bin-search algorithm. In this thesis, I focus on white-box attacks.

Depending on the goal of the attackers, attacks can be divided into targeted and untargeted attacks. A targeted attack tries to fool a target classifier so that a perturbed data point is classified as a specific (target) class that differs from the original class. For a data point x, its corresponding class c, and a target class $c_t \neq c$, the attacker's goal is to generate a perturbed data point $x' \in \mathcal{X}$ such that $\mathcal{C}(x') = c_t$. Conversely, an untargeted attack tries to fool a target classifier so that the perturbed image is misclassified, which means there is no specific target class. For a data point x, with corresponding class c, the attacker's goal is to generate a perturbed data point $x' \in \mathcal{X}$ such that $\mathcal{C}(x') \neq c$. Usually, the condition $d(x, x') \leq \epsilon$, for perturbation budget ϵ, is added for both targeted and untargeted attacks. Unless specified otherwise, I focus on untargeted attacks.

Let us consider attack algorithms that use a fixed perturbation budget ϵ to generate adversarial examples. When l_∞ norm is used for measuring the change (distance) of perturbation, the Fast Gradient Sign Method (FGSM) [Goodfellow et al., 2014] can be utilized to generate adversarial examples. It is motivated by the approximate linear behaviors of DNNs. Specifically, given an perturbation budget ϵ and a classification

loss function $\ell(\cdot,\cdot)$ used for training, an adversarially perturbed data point x'_{FGSM} is generated as:

$$x'_{\text{FGSM}} = x + \epsilon\,\text{sign}(\nabla_x \ell(x, c)).$$

where $\text{sign}(\cdot)$ is the element-wise sign function. Using $\text{sign}(\cdot)$ function exploits the near optimal changes (assuming the near linear behavior) for a fixed l_∞ norm budget ϵ.

The Basic Iterative Method (BIM) [Kurakin et al., 2016] extends the FGSM algorithm by iteratively applying an adversarial moving and clipping (projection) process. Typically, the BIM method will generate stronger perturbed data points than FGSM as DNN models are not exactly linear. Given a step size α_{step} and a number of iterations k, an adversarially perturbed data point $x'_{\text{BIM};k}$ is generated as:

$$x'_{\text{BIM};0} = x,\; x'_{\text{BIM};i+1} = \text{Clip}_{x,\epsilon}\left(x'_{\text{BIM};i} + \alpha_{\text{step}}\,\text{sign}(\nabla_x \ell(x'_{\text{BIM};i}, c))\right). \tag{3.2}$$

Here, the function $\text{Clip}_{x,\epsilon}(v)$ clips (limits) the vector v element-wise such that the resulting output vector is within ϵ distance (l_∞ norm) from the data point x.

The Projected Gradient Descent (PGD) [Madry et al., 2017] uses the same iterative steps as in equation (3.2). Instead of initializing from the original data position, formalized as $x'_{\text{BIM};0} = x$, PGD starts from a random initial position $x'_{\text{PGD};0} \in \mathbb{B}(x, \epsilon)$ where $\mathbb{B}(x, \epsilon)$ is an ϵ-ball around x. Using a random initial position enables multiple local maxima to be explored, which will be beneficial for robust optimization of classifiers.

While fixing the perturbation budget ϵ is a popular approach, there are attack algorithms that do not specify a perturbation budget. For an original data point x, these methods find the minimal distance perturbation r^* such that the perturbed position $x + r^*$ has a predicted class different from the correct class c. Mathematically, the minimal distance perturbation r^* is defined as:

$$r^* = \underset{r:\, x+r\in\mathcal{X}}{\text{argmin}}\, d(x, x + r) \text{ such that } \mathcal{C}(x + r) \neq c. \tag{3.3}$$

A CW (Carlini-Wagner) attack [Carlini and Wagner, 2017] considers a targeted attack with target class $c_t \neq c$. In so doing, the corresponding definition of r^* can be obtained by substituting the constraint $\mathcal{C}(x + r) \neq c$ into $\mathcal{C}(x + r) = c_t$ into the expression (3.3). To reformulate the constraint, Carlini and Wagner introduced an object function $f_{\text{obj}}(\cdot)$ such that $\mathcal{C}(x + r) = c_t$ if and only if $f_{\text{obj}}(x + r) \leq 0$. There are multiple possible object functions using logit values or softmax outputs of classification models. Using an object function $f_{\text{obj}}(\cdot)$, the authors rewrote the problem of finding the minimal distance

perturbation as:

$$\text{minimize } d(x, x+r) \text{ such that } f_{\text{obj}}(x+r) \leq 0. \tag{3.4}$$

Instead of directly solving the problem (3.4), they suggested solving an alternative formulation defined as:

$$\text{minimize } d(x, x+r) + \lambda f_{\text{obj}}(x+r), \tag{3.5}$$

where $\lambda > 0$ is a hyperparameter. That is, they relaxed the constraint into restraint. They then used an Adam optimizer [Kingma and Ba, 2014] to minimize the alternative formulation (3.5).

The DeepFool attack [Moosavi-Dezfooli et al., 2015] estimates the minimal distance perturbation r^* by iteratively updating the orthogonal projection in which it considers the target model as a locally linear classifier. Each projection is followed by the clipping operation to ensure elements (pixel values) of input images are within $[0, 1]$. Thus, their combined process is not an exact projection. Croce and Hein [2020b] proposed the FAB-attack, which improves the DeepFool attack by using exact projection on the decision boundary close to the original point.

3.2 Adversarially robust classifiers

The vulnerability of DNN models [Szegedy et al., 2013] motivates us to find models that are robust against adversarial perturbations. To achieve this, we first need to evaluate the robustness of classification models. Adversarial accuracy [Madry et al., 2017; Tsipras et al., 2018] is one of the most commonly used measures of adversarial robustness. Unlike standard (natural) accuracy that uses an original data point x and its corresponding class c, adversarial accuracy uses adversary region $R(x)$ and class c to calculate accuracy, where adversary region $R(x)$ refers to the allowed region of the perturbations for the data point x. Specifically, adversarial accuracy a is defined as:

$$a = \mathbb{E}_{(x,c)\sim D}[\mathbb{1}(\mathcal{C}(x^*) = c)] \text{ for } x^* = \underset{x' \in R(x)}{\operatorname{argmax}}\, \ell(x', c), \tag{3.6}$$

where $\ell(\cdot, \cdot)$ is classification loss based on the target classifier $\mathcal{C}$ (or its softmax outputs). While cross-entropy loss is commonly used for DNN classifiers, the zero-one loss can be used for non-differentiable classifiers.

Typically, ϵ-ball $\mathbb{B}(x, \epsilon)$ is used for modeling an adversary region $R(x)$ [Madry et al., 2017;

Tsipras et al., 2018]. In this case, we refer to adversarial accuracy as standard adversarial accuracy (SAA). Based on the definition (3.6), for a max perturbation budget ϵ, SAA $a_{\text{std}}(\epsilon)$ is defined as:

$$a_{\text{std}}(\epsilon) = \mathbb{E}_{(x,c)\sim D}[\mathbb{1}(\mathcal{C}(x^*) = c)] \text{ for } x^* = \underset{x' \in \mathbb{B}(x,\epsilon)}{\text{argmax}}\, \ell(x', c).$$

A classification model with relatively high adversarial accuracy is called *an (adversarially) robust classifier (model).*

Adversarial training (AT) [Goodfellow et al., 2014; Madry et al., 2017] is one of the most successful methods for defending against adversarial attacks. Unlike standard training (ST) that only uses the original data points to train DNNs (ST may also use non-adversarially augmented data points), AT also uses adversarially perturbed data points and the class of the original data point x to train neural networks. It is useful here to refer back to the definitions in section 1.1 regarding point-wise cross-entropy loss $\ell_{\text{CE}}(\cdot,\cdot)$, one-hot vector $\overrightarrow{\mathbb{1}}_c$, and the estimated class probability vector (the softmax outputs) $\hat{P}(\mathcal{Y}|\cdot)$. The AT loss $\ell_{\text{AT}}(x, c)$ for a data point x is defined as:

$$\begin{aligned}\ell_{\text{AT}}(x, c) &= \alpha \ell_{\text{CE}}(\overrightarrow{\mathbb{1}}_c, \hat{P}(\mathcal{Y}|x)) + (1-\alpha)\ell_{\text{CE}}(\overrightarrow{\mathbb{1}}_c, \hat{P}(\mathcal{Y}|\hat{x^*})) \\ &= -\alpha \hat{p}(c|x) - (1-\alpha)\hat{p}(c|\hat{x^*}),\end{aligned} \tag{3.7}$$

where $\hat{x^*}$ is an adversarially perturbed data point using an attack algorithm and $0 \leq \alpha < 1$ is an AT parameter. Parameter value $\alpha = 0$ [Madry et al., 2017] and $\alpha = 0.5$ [Goodfellow et al., 2014] are commonly used in AT. By guiding adversarially perturbed data point $\hat{x^*}$ to have the same estimated class c as the original x, AT increases robustness against adversarial attacks.

The TRADES [Zhang et al., 2019a] is a different form of adversarial training that guides adversarially perturbed data point $\hat{x^*}$ to have the same estimated class probability vector as the original x. Specifically, TRADES loss $\ell_{\text{TRADES}}(x, c)$ for a data point x is defined as:

$$\begin{aligned}\ell_{\text{TRADES}}(x, c) &= \ell_{\text{CE}}(\overrightarrow{\mathbb{1}}_c, \hat{P}(\mathcal{Y}|x)) + \frac{1}{\lambda}\ell_{\text{CE}}(\hat{P}(\mathcal{Y}|x), \hat{P}(\mathcal{Y}|\hat{x^*})) \\ &= -\hat{p}(c|x) - \frac{1}{\lambda}\hat{P}(\mathcal{Y}|x) \cdot \log(\hat{P}(\mathcal{Y}|\hat{x^*})),\end{aligned}$$

where $\lambda > 0$ is a hyperparameter and adversarially perturbed data point $\hat{x^*}$ maximizes $\ell_{\text{CE}}(\hat{P}(\mathcal{Y}|x), \hat{P}(\mathcal{Y}|\hat{x^*}))$, in contrast to original AT. Compared to the standard AT loss in equation (3.7) that uses the one-hot vector $\overrightarrow{\mathbb{1}}_c$ in the second term, the TRADE loss uses the original estimated class probability vector $\hat{P}(\mathcal{Y}|x)$ for adversarially perturbed data

point $\hat{x^*}$.

While adversarial training methods [Goodfellow et al., 2014; Madry et al., 2017; Zhang et al., 2019a] have exhibited empirical success in increasing adversarial accuracy, adversarially trained models can be defeated by stronger attack algorithms. One would want classifiers with no adversarial examples under certain conditions. In this regard, certified defenses [Cohen et al., 2019; Hein and Andriushchenko, 2017; Lecuyer et al., 2019; Wong and Kolter, 2018] guarantee that the predicted class is invariant within a radius. One approach to achieving certifiable robustness is to make a classifier invariant within an adversarial polytope $\mathcal{C}(\mathbb{B}(x, \epsilon))$, which is the set of outputs for inputs within a ball $\mathbb{B}(x, \epsilon)$. Wong and Kolter [2018] used a convex outer bound of adversarial polytope for certified defense. Another approach for certified defense is randomized smoothing [Cohen et al., 2019; Lecuyer et al., 2019]. Randomized smoothing uses a base classifier to construct a smoothed classifier for adversarial defense. Specifically, for a base classifier $\mathcal{C}(\cdot)$, randomized smoothing defines the smoothed classifier $\mathcal{C}_{\text{smooth}}(\cdot)$ as:

$$\mathcal{C}_{\text{smooth}}(x') = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, p(\mathcal{C}(x' + \delta) = y) \text{ for } \delta \sim \mathcal{N}(0, \sigma^2 I),$$

where δ is perturbation noise and σ is standard deviation.

Although most adversarial defenses are focused on defending only one type of attack, these models can still be vulnerable to other types of attacks [Schott et al., 2018a]. Tramer and Boneh [2019] investigated whether it is possible to achieve robustness against multiple types of attacks. They defined pairs of perturbation types as mutually exclusive perturbations (MEPs) when robustness to one perturbation type necessarily implies susceptibility to the other. They demonstrated that for the synthetic distribution suggested by Tsipras et al. [2018], l_1 and l_∞ perturbations are MEPs, which indicate it is not possible to achieve robustness against multiple attack types.

Croce and Hein [2020a] suggested a provably robust classifier against all l_p type attacks for all $p \geq 1$. (While this could be confusing for readers as it appears to contradict the findings of Tramer and Boneh [2019], it does not need to be. The proof about MEPs is for the particular distribution.) They showed that a guarantee for both l_1 and l_∞ norm based balls with the same center ensures a guarantee for all l_p. From this, they used the convex hull of the union of l_1 and l_∞ balls for provable robustness. Our analysis in Appendix A suggests that their defense can be regarded as a robustness guarantee on a subset of a ball, where this ball is based on a combined distance $d_{\text{combined}}(\cdot, \cdot)$ and this distance is defined as $d_{\text{combined}}(x_1, x_2) = \beta \left\|x_2 - x_1\right\|_1 + (1 - \beta) \left\|x_2 - x_1\right\|_\infty$ for a constant $\beta \in (0, 1)$.

Engstrom et al. [2019] assessed the feature mapping $f(x)$ learned by adversarial training (AT) and compared it with that of standard training (ST). Through a process named *representation inversion*, for the representation (feature vector) of a target image, they reconstructed a visually similar image with adversarially trained feature mapping which was not possible with ST. Hence, they found that adversarially trained feature mapping $f(x)$ is approximately invertible. Additionally, when they visualized a feature by maximizing a specific feature component, they found that the visualized features of robust models capture high-level features.

Inspired by the human-aligned properties of adversarially trained feature mapping [Engstrom et al., 2019], Salman et al. [2020] assessed the transfer classification of adversarially trained models. Transfer classification is a classification task that involves extracting features from a trained model and classifying new classification tasks using the features obtained. The authors experimentally demonstrated that adversarially trained models perform better than models undergoing standard training in transfer classification.

3.3 Tradeoff between natural accuracy and adversarial accuracy

Although finding adversarially robust classifiers is important, there is a tradeoff between natural accuracy and adversarial accuracy [Tsipras et al., 2018; Zhang et al., 2019a]. This phenomenon obstructs the adoption of robust models in multiple applications as robust models have lower standard (natural) accuracy than standard models. Therefore, it is crucial to identify *why this tradeoff occurs* and *how to avoid or reduce it* if possible. There are several hypotheses that explain the tradeoff.

One hypothesis is that the tradeoff between accuracy and adversarial accuracy is inevitable [Tsipras et al., 2018; Zhang et al., 2019a]. Tsipras et al. [2018] explained the tradeoff by providing a toy example that both high standard accuracy and high robust accuracy cannot be achieved simultaneously. However, Nakkiran [2019] argued that humans are both an accurate and robust classifier, and thus argued that the tradeoff is not inherent in general. He suggested that the tradeoff may be due to the required high model complexity of robust classifications. Another hypothesis suggested by Kim and Wang [2019]; Suggala et al. [2019] (and also paper III) is that the tradeoff originates from the standard definition of adversarial accuracy that allows to genuinely change classes of samples for large perturbation budget ϵ. They suggested an alternative definition of adversarial robustness using a reference classifier. Yang et al. [2020b] argued that the tradeoff is not inherent and common perturbation budget ϵ values are not large enough

to change classes. To obtain a model that is both accurate and robust, they suggested that it may be necessary to use augmentation techniques to reduce generalization errors, in addition to imposing local Lipschitzness for robustness. Schmidt et al. [2018] and Raghunathan et al. [2020] suggested that the tradeoff originates, at least partially, from the reduced generalization power of adversarially trained classifiers. Schmidt et al. explained that the sample complexity of robust learning can be vastly greater than that of standard training. Raghunathan et al. [2020] provided cases where some augmentations like adversarial perturbations can harm the generalization of models. To limit this, they suggested using robust self-training (RST) [Carmon et al., 2019] by exploiting unlabeled data points.

Recently, Wu et al. [2021] have found that adversarially robust training methods increase inter-class similarity on the penultimate layer of a neural network. Such change can reduce the discriminative power of classification models and thus lessen the generalization power of classifiers [Kornblith et al., 2021]. To explain the increased inter-class similarity, they suggested that forcing both adversarially perturbed points and original points to the same predicted class may make their distributions closer, and also increase inter-class similarity.

Paper III is motivated by the tradeoff between natural accuracy and standard adversarial accuracy (SAA). As in Kim and Wang [2019] and Suggala et al. [2019], I speculated that the tradeoff is due to the definition of adversarial robustness which allows perceptual classes of images to be changed for a large perturbation norm ϵ. As identified by Yang et al. [2020b], commonly used ϵ values might not be large enough to change classes of samples. However, being robust only against small perturbation ϵ value enables attackers to fool the models with larger perturbation. Ghiasi et al. [2019] demonstrated that even certifiably robust models can be spoofed by their attack, which exploits large perturbation budget ϵ while maintaining the semantic class of images.

To overcome such issues of SAA, alternative definitions of adversarial robustness can be used based on a reference classifier [Kim and Wang, 2019; Suggala et al., 2019]. However, the meaning of robustness will be dependent on the choice of the reference model. By using Voronoi cells [Khoury and Hadfield-Menell, 2019] based on data points, in paper III I suggested a new definition of adversarial accuracy named Voronoi-epsilon adversarial accuracy (VAA) that avoids the tradeoff in definition, even for a large ϵ value without an external classifier. VAA enables the study of global robustness of classifiers and shows the connection with the SAA. I demonstrated that the nearest neighbor (1-NN) classifier maximizes VAA.

Chapter 4

Hierarchy-informed classification

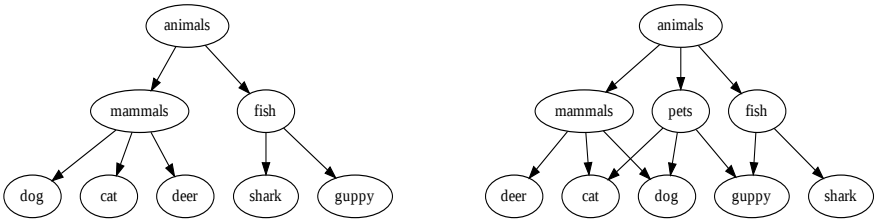
While common classification methods treat all misclassifications equally, the severity of misclassifications may be vastly different. For example, when a self-driving system observes an object, mistaking a person as a tree will be more dangerous than mistaking a streetlight as a tree [Bertinetto et al., 2020]. Additionally, some classification problems require the classification of a large number of classes and there can be underlying hierarchical relationships among classes. In such cases, it will be hard to obtain highly accurate classifiers. If there are multiple classifiers with comparable accuracy, it is desirable to choose a classifier whose predicted class is hierarchically similar to the true class. These explanations demonstrate the importance of classification models that take into account the hierarchical relationships of classes. By incorporating class hierarchy, these models can improve "hierarchy-informed performance," which is model performance by considering predefined hierarchical structures.

Section 4.1 explains the representations of hierarchical structures and hierarchical distance of classes. Here, I describe methods for determining class hierarchy structures. In section 4.2, I explore three approaches to hierarchy-informed classification.

4.1 Determination of hierarchy

Hierarchical relationships of classes are usually represented by a tree structure. In a class hierarchical structure, classes are represented as leaf nodes (i.e., nodes with no children), and directions of edges represent inclusion relationships. Therefore, if there is an edge from node w_1 to node w_2, it denotes that node w_1 includes node w_2. When there is a unique node without a parent node, that node is the root node. For instance, the "animals" node in Figure 4.1a is the root node. Because each class has a corresponding

leaf node and vice versa, I regard class and class node as interchangeable. In this chapter, I denote an arbitrary node as w, an arbitrary class (class node) as y, and a special class (class node) as c. As visualized in Figure 4.1a, the tree structure allows only one parent node for a node. Unlike a tree, a directed acyclic graph (DAG) allows multiple parent nodes for a node. This enables us to represent more complex relationships among classes. Because a tree structure is easier to handle than a DAG structure, it is more commonly used to model the hierarchical structure of classes [Barz and Denzler, 2019; Bertinetto et al., 2020; Garnot and Landrieu, 2021]. Hence, in this thesis, I focus solely on a tree structured class hierarchy.



(a) (b)

Figure 4.1: (a) Example of a tree structure. Each node has, at most, one parent node in a tree hierarchy. (b) Example of a DAG structure where "cat", "dog", and "guppy" classes have more than one parent node.

From a predefined hierarchical tree, one can derive hierarchical distances d_H for any two classes. These can be used for hierarchy-informed classification and measuring the hierarchical performances of classification models. There are two ways to define hierarchical distance between two classes: lowest common subsumer (LCS) [Barz and Denzler, 2019; Bertinetto et al., 2020] and the shortest path [Garnot and Landrieu, 2021]. Barz and Denzler [2019] used LCS to define hierarchical distance. When $\mathcal{V}$ is the set of nodes, they defined the hierarchical distance between classes y_1 and y_2 as:

$$d_{H;\mathrm{LCS}}(y_1, y_2) = \frac{\mathrm{height}(lcs(y_1, y_2))}{\max\limits_{w \in \mathcal{V}} \mathrm{height}(w)},$$

where $lcs(\cdot, \cdot)$ is the LCS of class nodes and $\mathrm{height}(w)$ is the length of the shortest path from node w to a leaf node, also known as a class node. Note that LCS-based hierarchical distance $d_{H;\mathrm{LCS}}$ satisfies $0 \leq d_{H;\mathrm{LCS}} \leq 1$.

Let us consider the tree structure in Figure 4.1a. When hierarchical distance is defined by LCS, the hierarchical distance between "dog" and "cat" classes is $\frac{1}{2}$, formalized as

$d_{H;\mathrm{LCS}}(\mathrm{dog}, \mathrm{cat}) = \frac{1}{2}$. Similarly, the distance between "dog" and "shark" classes is $\frac{2}{2} = 1$, formalized as $d_{H;\mathrm{LCS}}(\mathrm{dog}, \mathrm{shark}) = 1$. When hierarchical distance is defined by the shortest path distance, the hierarchical distance between "dog" and "cat" classes is 2, formalized as $d_{H;\mathrm{path}}(\mathrm{dog}, \mathrm{cat}) = 2$, as the shortest path is through the "mammals" node. Similarly, the distance between "dog" and "shark" classes is 4, formalized as $d_{H;\mathrm{path}}(\mathrm{dog}, \mathrm{shark}) = 4$.

How can a class hierarchical structure be determined? There are several approaches to defining hierarchical structures. Class hierarchy can be defined by domain experts [Garnot and Landrieu, 2021] when there is rich knowledge of the classes. It can be extracted from a WordNet database [Miller, 1998], which is a English word database that contains their semantic relationships. When such approaches are not available, class hierarchical structure can be inferred from a trained classifier by estimating hierarchical distances. One such approach introduced by Godbole [2002] is to use the confusion matrix of a classifier to estimate class relations. When there are two similar classes y_1 and y_2, then a classifier might predict data points in class y_1 as class y_2, and vice versa. Hence, he suggested using normalized rows (row vectors) of the confusion matrix to represent classes and distance between two row vectors as an estimation of hierarchical distance. When hierarchical distances are estimated, hierarchical structure can be inferred using a hierarchical clustering algorithm such as the neighbor-joining method [Saitou and Nei, 1987].

4.2 Hierarchy-informed classification methods

Hierarchical information can be incorporated into DNN classifiers in three distinct ways: changing architecture, modifying cross-entropy loss calculation, and metric learning.

For a tree structure, there is a unique path from highest node to each class node (i.e., each leaf node). Thus, there is a unique factorization of a class probability along the path in the tree, a fact that can be used for hierarchy-informed classification.

Redmon and Farhadi [2017] estimated class probability by considering the conditional probabilities between the parent node and child node. For instance, when we apply their estimation method in Figure 4.1a, the class probability $p(\mathrm{dog}|x)$ is estimated as:

$$\hat{p}(\mathrm{dog}|x) = \hat{p}(\mathrm{dog}|\mathrm{mammals}; x)\hat{p}(\mathrm{mammals}|\mathrm{animals}; x)\hat{p}(\mathrm{animals}|x),$$

where $\hat{p}$ represents an estimated probability and $\hat{p}(\mathrm{animals}|x) = 1$. They employed a modified architecture using softmax for each parent node to its children nodes in order

to directly estimate conditional probability $p(w_{\text{child}}|w_{\text{parent}}; x)$ where w_{child} is a child node and w_{parent} is the corresponding parent node of w_{child}.

Bertinetto et al. [2020] proposed two approaches that modify the calculation of cross-entropy loss: hierarchical cross-entropy and soft-labels. As in Redmon and Farhadi [2017], hierarchical cross-entropy (HXE) exploits the fact that class probability $p(y|x)$ can be calculated by the product of conditional probabilities due to the unique factorization of tree paths. When we denote nodes from a class node y to its highest ancestor node R_y as $y^{(0)} = y, y^{(1)}, \cdots, y^{(\text{height}(y))} = R_y$, the factorization of the estimated class probability can be expressed as:

$$\hat{p}(y|x) = \prod_{l=0}^{(\text{height}(y))-1} \hat{p}(y^{(l)}|y^{(l+1)}; x). \tag{4.1}$$

The estimated conditional probabilities can be calculated using the class probabilities as follows:

$$\hat{p}(y^{(l)}|y^{(l+1)}; x) = \frac{\sum_{y_1 \in \text{Leaves}(y^{(l)})} \hat{p}(y_1|x)}{\sum_{y_2 \in \text{Leaves}(y^{(l+1)})} \hat{p}(y_2|x)},$$

where $\text{Leaves}(w)$ denotes the set of class nodes of the subtree rooted by the node w. HXE loss is then defined as:

$$L_{\text{HXE}} = -\frac{1}{|B|} \sum_{(x,c) \in B} \sum_{l=0}^{(\text{height}(c))-1} \lambda(c^{(l)}) \log(\hat{p}(c^{(l)}|c^{(l+1)}; x)),$$

where $\lambda(c^{(l)})$ is the corresponding weight for edge between node $c^{(l)}$ and node $c^{(l+1)}$. When weights $\lambda(c^{(l)}) = 1$ for all index l, HXE is equivalent to the standard cross entropy (1.5) due to equation (4.1). The authors suggested choosing $\lambda(w) = e^{-\alpha \text{height}(w)}$ for weights where $\alpha > 0$ is a hyperparameter. The term $\log(\hat{p}(c^{(l)}|c^{(l+1)}; x))$ corresponds to the information associated with the edge between node $c^{(l)}$ and $c^{(l+1)}$. Thus, HXE can be regarded as weighing the information associated with these edges differently.

Bertinetto et al.'s [2020] soft-label method modifies one-hot vectors in cross-entropy loss (1.4) into soft-labels using predefined hierarchical distances. Specifically, soft-label vector $\overrightarrow{\text{soft}}_y$ is defined for each class $y \in \mathcal{Y}$. Its element for class y_1 is defined as:

$$\overrightarrow{\text{soft}}_{y;y_1} = \frac{\exp\left(-\beta d_H(y, y_1)\right)}{\sum_{y_2 \in \mathcal{Y}} \exp\left(-\beta d_H(y_2, y_1)\right)},$$

where $\beta \geq 0$ is a hyperparameter. Accordingly, soft-label loss L_{Soft} is:

$$\begin{aligned} L_{\text{Soft}} &= -\frac{1}{|B|} \sum_{(x,c)\in B} \overrightarrow{\text{soft}}_c \cdot \log(\hat{P}(\mathcal{Y}|x)) && \text{(Using dot product)} \\ &= -\frac{1}{|B|} \sum_{(x,c)\in B} \sum_{y_1 \in \mathcal{Y}} \overrightarrow{\text{soft}}_{c;y_1} \log(\hat{p}(y_1|x)) && \text{(Using summation),} \end{aligned}$$

where $\hat{P}(\mathcal{Y}|x)$ is the estimated class probability vector of data point x. For an appropriate hyperparameter β, the loss L_{Soft} forces the model to estimate higher class probability on classes hierarchically similar to the true class. (Even though the authors described this method as a "label-embedding" approach, it did not consider distance nor similarity in feature space. Hence, I do not categorize their approach as metric learning approach.)

Metric learning can be used for hierarchy-informed classification. A common approach is to prefix the positions of proxy representatives - weight vectors $\tilde{W}_y$ - of classification-based metric learning using hierarchical information [Barz and Denzler, 2019; Jayathilaka et al., 2021; Mettes et al., 2019]. This is then followed by moving data points into the corresponding position for each class. For instance, Barz and Denzler [2019] prefixed proxies for spherical space by defining hierarchical class similarity $s_H(\cdot,\cdot)$ as $s_H(y_1, y_2) := 1 - d_{H;\text{LCS}}(y_1, y_2)$ for $y_1, y_2 \in \mathcal{Y}$. Next, they made the inner product of two proxies $\tilde{W}_{y_1}$ and $\tilde{W}_{y_2}$ match the corresponding class similarity $s_H(y_1, y_2)$. They proposed the CORR loss for training embedding using prefixed proxies. This loss is defined as:

$$L_{\text{CORR}} = \frac{1}{|B|} \sum_{(x,c)\in B} \left(1 - \tilde{W}_c^T \tilde{f}(x)\right) = \frac{1}{|B|} \sum_{(x,c)\in B} (1 - \cos\theta_c),$$

where $B \subset \mathcal{D}$ is a mini-batch and θ_c is the angle between $\tilde{W}_c$ and $\tilde{f}(x)$.

Instead of matching the inner product of proxies, Mettes et al. [2019] suggested optimizing proxies to have the same rank as class similarity by using ranking loss based on class triplets. To train embedding, they proposed a similar formulation to CORR loss, which is defined as:

$$L_{\text{H-spherical}} = \sum_{(x,c)\in B} \left(1 - \tilde{W}_c^T \tilde{f}(x)\right)^2 = \sum_{(x,c)\in B} (1 - \cos\theta_c)^2.$$

In contrast to approaches that use fixed proxies for incorporating hierarchical information in metric learning, Garnot and Landrieu [2021] suggested only penalizing distortion between proxy distance matrix and hierarchical distance matrix using Euclidean space

as an embedding. Their distortion-based penalty is defined as:

$$L_{\text{disto}}(W) = \frac{1}{|\mathcal{Y}|(|\mathcal{Y}|-1)} \min_{s\in\mathbb{R}^+} \sum_{k,l,k\neq l}^{|\mathcal{Y}|} \left(\frac{s\, d(W_k, W_l) - D_H(k,l)}{D_H(k,l)} \right)^2,$$

where s is a scaling factor and D_H is a hierarchical distance matrix whose (l,k)-th element is the hierarchical distance $d_H(y_l, y_k)$. This penalty L_{disto} guides any distance between proxy pair to its corresponding hierarchical distance using an appropriate scale factor.

Depending on the datasets, hierarchical information of classes may be unavailable. To address this challenge, in paper IV, I investigated whether learned proxy representatives of classification-based metric learning models can estimate class hierarchical relationships and achieve adequate hierarchy-informed performance, which is metric learning performance and classification performance by considering the hierarchical relationships of classes. Such analysis can also be beneficial when the hierarchical relationships of classes are known because we can verify whether learned class relationships match existing ones. For my experiments, I trained a softmax classifier and metric learning models on a spherical embedding space with several training options. When I trained models without a predefined hierarchy, *using DR formulation in paper III achieved better hierarchical inference performance and better hierarchy-informed performance in two measures than using the NormFace model [Wang et al., 2017].* Estimating class relationships using proxies is more advantageous than a confusion matrix based approach [Godbole, 2002], which can be problematic when there are only small number of data points in classes. Additionally, the confusion matrix based approach requires separate training and evaluation (validation) processes for hierarchical inference. Using proxies of classification-based metric learning is more convenient as it automatically learns such proxies during training.

Chapter 5

Adversarial robustness and feature space characteristics

DNN classifiers learn a complex feature mapping function $f(\cdot)$ to solve classification problems. The transformed (feature) space usually has higher discriminality compared to the input space, which is beneficial for achieving high generalization performances. One of the results presented in paper III is that adversarially robust classifiers have locally (within ϵ-balls containing original data points) equivalent classifications with the nearest neighbor (1-NN) classifier based on the same distance metric for robustness (equivalent findings are described in Khoury and Hadfield-Menell [2019]; Yang et al. [2020a,b]). 1-NN classifiers are distance-based models. Because an adversarially robust classifier will be locally similar to an (input) distance-based classifier, adversarial training (AT) will likely ensure that the feature spaces of adversarially robust models more closely resemble the input space. One way to investigate this is by analyzing the *distance structure*, which denotes the collection of pairwise-distance values between data points for a given space. Specifically, we can expect the similarity between the distance structure of a feature space of an adversarially trained model and the input space distance structure to be greater than the similarity between the distance structure of a feature space of a standard (non-robust) model and the input space distance structure. As increased resemblance to the input space can also alter characteristics of feature space such as discriminative power and dimensionality (see section 2.4 for details), this will result in changes of classification and transfer performance.

In section 5.1, I investigate whether the distance structure of an adversarially trained feature space more closely resembles the input space distance structure than does the distance structure of a standard model. Section 5.2 revisits the characteristic measurements of spaces and suggests possible effects of increased resemblance to input space on mod-

els. In section 5.3, I explore whether resembling input distance structure is correlated with the characteristics of the feature space. In section 5.4, I discuss the implications of the relationships identified in this chapter.

5.1 Comparing distance structures of feature space and input space

In this part of the thesis, I investigate whether *distance structures of the feature spaces of adversarially robust models more closely resemble the distance structure of input space than does the distance structure of a standard model.* To achieve this, using CIFAR-10 [Krizhevsky et al., 2009] and SVHN [Netzer et al., 2011] datasets, I assessed the mean correlation (MC) values introduced in paper IV. MC is a measure devised to compare two distance structures based on rank correlation values. Because each element of a distance matrix represents the distance between two data points (or between two class representatives in paper IV), MC value is calculated from two distance matrices (with the same size). MC will take a value between -1 and 1 as it is a similarity measure. High MC indicates that the distance structures of the matrices are similar (correlated). MC close to 0 indicates the distance structures of the matrices are not correlated. For the current assessment, I calculated MC from input space distance matrix D_I and feature space distance matrix D_F. If it is confirmed that the distance structures of the feature space of robust models more closely resemble the distance structure of input space than do the distance structures of the feature space of standard models, we would expect MC values for adversarially trained models to be higher than for a standard model.

Table 5.1 presents the MC values for standard training (ST) and adversarial training (AT) for the CIFAR-10 dataset with the same network architecture. The corresponding results for the SVHN dataset are provided in Table 5.2. The results revealed that AT increased MC values. Most notably, increasing perturbation budget ϵ for AT increased the MC values. These results suggest that AT would make the distance structures of feature outputs of models more similar to that of the input space. *What are the possible effects of this on feature space?*

Table 5.1: MC values between two distance matrices D_I and D_F for the CIFAR-10 dataset. For the input space distance matrix D_I, distance between training data points was calculated using both l_2 and l_∞ norm distances. For the feature space distance matrix D_F, Euclidean distance (l_2 norm) between training data points on the feature space was calculated for various training options.

Method / Distance	ST	AT (l_2 norm, $\epsilon = 0.25$)	AT (l_2 norm, $\epsilon = 0.5$)	AT (l_2 norm, $\epsilon = 1.0$)	AT (l_∞ norm, $\epsilon = \frac{8}{255}$)
l_2 norm	0.0818	0.1586	0.2109	0.3003	0.2719
l_∞ norm	0.0517	0.1239	0.1689	0.2229	0.1914

Table 5.2: MC values between two distance matrices D_I and D_F for the SVHN dataset. Both distance matrices D_I and D_F are calculated as in Table 5.1. Note that AT models for the SVHN dataset were trained also with natural images as full AT could not easily converge.

Method / Distance	ST	AT (l_2 norm, $\epsilon = 0.25$)	AT (l_2 norm, $\epsilon = 0.5$)	AT (l_2 norm, $\epsilon = 1.0$)	AT (l_∞ norm, $\epsilon = \frac{8}{255}$)	AT (l_∞ norm, $\epsilon = \frac{12}{255}$)
l_2 norm	-0.0118	0.0301	0.1003	0.1737	0.0134	0.0371
l_∞ norm	-0.0061	0.0576	0.1490	0.2162	0.0307	0.0619

5.2 Effects of increased MC values on discriminality and dimensionality

Before explaining the possible effects of increased MC values, I revisit the characteristics of spaces described in section 2.4. Discriminality (discriminative power) measures represent the degree of closeness to the corresponding class representatives (or other data points with the same class) compared to the overall distances of a space. Class separation R^2, which is a discriminality measure proposed by Kornblith et al. [2021], is positively related to standard (non-transfer) test accuracy. Another discriminality measures is discriminative value ϕ [Liu et al., 2020]. As in equation (2.7), discriminative value ϕ is defined as the ratio of inter class variances D_{inter} to intra class variances D_{intra}. While the discriminality of feature space was considered by using learned features in equations (2.4), (2.6), (2.10), and (2.11), discriminality can also be assessed for input space by directly using data points x for calculation.

Roth et al. [2020] defined a measure ρ to assess "the amount of directions of significant variance (DoV)". This can represent the amount of compression (flattening) that occurs in feature mapping. Small ρ value means more DoV and less compression. Roth et al. identified a negative correlation between ρ value and performance on novel classes (unseen classes during the training phase). They explained that retaining high dimensionality will be beneficial for generalization when there is a considerable amount of distribution shift, such as in the classification of novel classes.

The first possible effect of increased MC values is discriminality of learned features. If data points are not discriminative enough according to the distance metric for adversarial robustness, in our case the input distance metric, AT with the same distance metric can diminish the discriminality of learned features. This may reduce the generalization power of the models in standard (non-transfer) classification [Kornblith et al., 2021]. This is one possible reason for the tradeoff between standard accuracy and adversarial robustness.

The second possible effect is the amount of directions of significant variance (DoV) [Roth et al., 2020]. If input dimension d_I is high dimensional and l_p norm is used for robustness, high dimensional distance relationships on the input space may be reflected in the learned feature space. Hence, AT may increase the amount of DoV. This may explain why adversarially trained models usually transfer more effectively [Salman et al., 2020].

To investigate the discriminative power of input space distances, I assessed two discriminative measures: discriminative value ϕ [Liu et al., 2020] and class separation R^2 [Kornblith et al., 2021] using (input) data points. Because there is no standard reference value for comparison, these values alone may not be sufficient to determine whether the input space is discriminative. Therefore, I also assessed the test accuracy of two distance-based models using input space information: 1-NN (single nearest neighbor) classifier and NPC (nearest prototype classifier). If input space is highly discriminative, then distance information will be helpful in classification, and thus we can expect high test accuracy of distance-based classifiers (higher than 0.8 for instance). Note that 1-NN test accuracy is based on local (neighbor) distance, which is in contrast to discriminative measures ϕ, R^2, and NPC test accuracy that are based on global statistics. The results are summarized in Table 5.3. With respect to discriminative measures based on global statistics, SVHN dataset has less discriminative input features than the CIFAR-10 dataset. The reverse was the case for 1-NN test accuracy. Thus, the SVHN dataset has more discriminative input features when it comes to a local distance measure. Given that the test accuracy of both distance-based classifiers is smaller than 0.5, the discriminative powers of input spaces are not high in either dataset.

Table 5.3: Discriminative value ϕ, class separation R^2, and test accuracy of 1-NN (single nearest neighbor) classifier and NPC (nearest prototype classifier) of input space using two distance metrics on two datasets.

Dataset	CIFAR-10	SVHN
ϕ (l_2 norm)	0.1635	0.0021
ϕ (l_∞ norm)	0.1804	0.0073
R^2	0.0550	0.0016
1-NN (l_2 norm)	0.3546	0.4236
1-NN (l_∞ norm)	0.1816	0.3113
NPC (l_2 norm)	0.2771	0.1003
NPC (l_∞ norm)	0.2136	0.1171

Previously, I suggested that AT may reduce discriminality. To test this, I assessed inter class variances D_{inter}, intra class variances D_{intra}, and discriminative values ϕ on feature space (after normalization) [Liu et al., 2020]. I also assessed class separation R^2 [Kornblith et al., 2021]. To measure changes in the amount of DoV from AT, I counted the number of dimensions in the normalized singular value spectrum of feature space whose variance is higher than $\frac{1}{d_F - 1}$, and denoted this value as d_{DoV}. Higher d_{DoV} value indicates there are more principal directions that have higher variances than average, which indicates less compression of the data. Additionally, ρ values [Roth et al., 2020] are calculated as defined in section 2.4.

Table 5.4 presents the investigated values for the CIFAR-10 dataset. The corresponding results for the SVHN dataset are shown in Table 5.5. Similar to the observations of Wu et al. [2021] who found that AT increased inter-class similarity, I observed decreased inter-class variance D_{inter} from AT. However, in contrast to their observations, which showed decreased intra-class variance from AT, there was increased intra-class variance D_{intra} from AT. Furthermore, AT decreased discriminative value ϕ and class separation R^2. This indicates the possible existence of the suggested effect AT on discriminative power. AT increased the number of dimensions in singular value spectrum d_{DoV}, but inconsistent changes were observed in ρ values. This also supports the suggested effect of AT on the dimensionality of feature space. In most cases, increasing the budget ϵ of AT magnified the changes.

Table 5.4: Inter class variances D_{inter}, intra class variances D_{intra}, discriminative values ϕ, class separation R^2, number of dimensions in singular value spectrum d_{DoV}, and ρ with different training options on the CIFAR-10 dataset. These values were calculated for normalized representations.

Method	ST	AT (l_2 norm, $\epsilon = 0.25$)	AT (l_2 norm, $\epsilon = 0.5$)	AT (l_2 norm, $\epsilon = 1.0$)	AT (l_∞ norm, $\epsilon = \frac{8}{255}$)
D_{inter}	1.0245	0.6783	0.6500	0.5376	0.5312
D_{intra}	0.0683	0.1312	0.1513	0.2606	0.1991
ϕ	15.0003	5.1692	4.2946	2.0631	2.6684
R^2	0.8709	0.6993	0.6590	0.4814	0.5456
d_{DoV}	32	51	76	75	71
ρ	2.4053	1.6701	0.9996	1.4570	1.3003

Table 5.5: Inter class variances D_{inter}, intra class variances D_{intra}, and discriminative values, class separation R^2, number of dimensions in singular value spectrum d_{DoV}, and ρ with different training options on the SVHN dataset. These values were calculated for normalized representations.

Method	ST	AT (l_2 norm, $\epsilon = 0.25$)	AT (l_2 norm, $\epsilon = 0.5$)	AT (l_2 norm, $\epsilon = 1.0$)	AT (l_∞ norm, $\epsilon = \frac{8}{255}$)	AT (l_∞ norm, $\epsilon = \frac{12}{255}$)
D_{inter}	1.2494	0.7214	0.5972	0.4889	0.8629	1.0091
D_{intra}	0.0468	0.0578	0.1215	0.2735	0.0564	0.0671
ϕ	26.6981	12.4786	4.9135	1.7879	15.2928	15.0282
R^2	0.9231	0.8488	0.6885	0.4458	0.8731	0.8711
d_{DoV}	21	35	48	52	34	31
ρ	2.3753	2.8203	2.5358	2.2234	2.6031	2.5739

5.3 Correlation between MC values and representation measures

To investigate whether resembling input distance relationships is correlated with characteristics of the feature spaces, I performed a correlation analysis on the results by calculating both the Pearson correlation coefficient and Spearman's rank correlation coefficient between MC values (mean correlations in Tables 5.1 and 5.2) and representation measures (discriminative and dimensionality measures in Tables 5.4 and 5.5). Using these measures, my analysis will reveal whether representation measures are correlated with the MC value, which measures the degree to which the feature distance structure is similar to input distance structure. If correlations are found, this will support the suggested effects of AT on representation, albeit not conclusively. Because MC values based on both l_2 and l_∞ norms are significantly correlated (p-value < 0.01) with coefficients higher than 0.98 in both datasets, I only used l_2 norm-based MC values for analyses.

The results are presented in Table 5.6. These reveal that discriminative power ϕ is negatively correlated with the MC value (only marginally significant in terms of Pearson's correlation). Class separation R^2 has a significant negative correlation with MC value. d_{DoV} has a positive correlation with mean correlation values but with marginal or non-significant results. ρ value has a negative correlation with mean correlation values, but this was not significant in any of the cases.

Table 5.6: Correlations between mean correlation (MC) values (in Tables 5.1 and 5.2) and representation measures (in Tables 5.4 and 5.5). Representation measures include discriminative values ϕ, class separation R^2, number of dimensions in singular value spectrum d_{DoV}, and ρ value. The letter "M" indicates marginal p-value (between 0.01 and 0.05). "*" indicates significant p-value (between 0.001 and 0.01). "**" indicates highly significant p-value (< 0.001).

	CIFAR-10		SVHN	
	Pearson	Spearman	Pearson	Spearman
ϕ	-0.9058^{M}	-1.0000^{**}	-0.9082^{M}	-0.9429^{*}
R^2	-0.9917^{**}	-1.0000^{**}	-0.9860^{**}	-0.9429^{*}
d_{DoV}	0.9115^{M}	0.7000	0.9337^{*}	0.8286^{M}
ρ	-0.7513	-0.6000	-0.5271	-0.3714

The table reveals a (1) negative correlation between MC value and discriminative measures (ϕ and R^2) and (2) positive correlation between similarity to input distance structure and dimensionality of features d_{DoV}. These suggest that if AT forces feature space distance structure to become more similar to the input distance structure then, correspondingly, feature space discriminality may be decreased and feature dimensionality may be increased.

5.4 Discussion

The identified relationships in this chapter can be summarized as follows. First, AT may force feature space to more closely resemble the distance structure of the input space. Consequently, AT may reduce discriminative powers of representations (ϕ and R^2). Moreover, by mimicking the distance relationships in the input space, AT may increase the number of directions of significant variance (d_{DoV}). If the decrease of discriminative power in the AT model is due to increased resemblance to the distance structure of the input space, this has important implications. Because the discriminative power of input features is not high enough in the datasets considered, as shown in

Table 5.3, adhering to l_p norm of input space as a distance measure implies that the corresponding robust models will have less discriminative features space than ST models. Thus, with respect to generalization performances, the *accuracy-robustness tradeoff may be inevitable* in both datasets. However, the tradeoff can be avoided when a more advanced distance metric is used such that discriminative power based on the distance is high. This highlights *the importance of devising and using advance metrics*. However, one limitation with this work is that I only studied two data sets. In particular, both have low input space discriminality. To study the relationship between the resemblance to the distance structure of the input space and discriminality of feature space more generally, it would be useful to experiment with synthetic datasets that have varying degrees of (input) discriminality and dimensions.

Chapter 6

Discussion and future directions

In this thesis, I investigated different fields of machine learning: metric learning (Chapter 2), adversarial robustness (Chapter 3), and hierarchy-informed classification (Chapter 4). Moreover, in Chapter 5, I explored the possible effects of adversarial robustness on a feature space. In this chapter, I highlight the significance of distance in the explored fields, summarize the contributions of my research in these areas, and suggest possible future works.

6.1 Discussion

Metric learning is concerned with learning a feature mapping $f(\cdot)$ such that distance on the feature (embedding) space represents dissimilarity between data points. Therefore, metric learning can be regarded as learning a distance. In section 2.4, I explored how the discriminative power of learned feature space can affect the performance of models. The fact that the feature space represents the similarity of data points highlights the importance of the characteristics of distance regarding model performances. In paper I, the first author proposed using metric learning for the classification of plankton images. Using metric learning models can be advantageous in classifying data points from novel plankton classes as it does not require retraining of the embedding function. In paper II, I proposed distance-ratio-based (DR) formulation for metric learning, which has two useful properties. First, the corresponding loss is not affected by the scale of an embedding. Second, it outputs the optimal (maximum or minimum) classification confidence scores for each class representative.

Distance plays an important role in adversarial examples and robustness. For instance, it has been used in formally defining adversarial examples [Biggio et al., 2013] and the

commonly employed ϵ-ball-based definition [Madry et al., 2017; Tsipras et al., 2018] of adversarial accuracy. A number of analytical results highlight the importance of the choice of distance for adversarial robustness. While Croce and Hein [2020a] suggested a model provably robust against all l_p attacks ($p \geq 1$), comprising multiple perturbation types, my analysis in Appendix A shows that their defense used a subset of a ball based on a combined distance $d_{\text{combined}}(\cdot, \cdot)$. This suggests that it is possible to achieve robustness against multiple attacks by being robust to a single attack type (d_{combined} atttack). My proposed measure of adversarial robustness in paper III, Voronoi-epsilon adversarial accuracy (VAA), aims to overcome the tradeoff between accuracy and adversarial accuracy for large ϵ values. Furthermore, I define global Voronoi-epsilon robustness as a limit of the Voronoi-epsilon adversarial accuracy, and demonstrate that the nearest neighbor (1-NN) classifier maximizes this. One of the findings from the paper is the local equivalence of adversarially robust classifiers with the nearest neighbor (1-NN) classifier. Given that 1-NN classifiers are distance-based models, this indicates that the local robustness will be affected by the choice of distance measure for adversarial robustness. Moreover, the experiments in Chapter 5 suggest that for adversarially robust models, the distance structure of feature space will resemble the distance structure of input space. Such resemblance may also change the generalization power of the robust models and transfer performances, further highlighting the importance of the choice of distance in adversarial robustness.

Distance has crucial roles to play hierarchy informed classifications. From a tree structure, we can obtain hierarchical distances between classes. Conversely, when we do not know the hierarchical structure of classes, we can use class distances to infer the hierarchy tree through hierarchical clustering algorithms. Hierarchical class distances are also used in measuring the performance of hierarchy informed classifications. Moreover, hierarchical distances are used for metric learning-based hierarchy-informed classifications by prefixing or guiding the positions of proxy representatives. Paper IV suggests estimating hierarchical class distances from classification-based metric learning models. When we know class hierarchy, we can verify whether learned semantic distances of the model match our prior knowledge. In a situation without a known hierarchy of classes, estimated hierarchical class distances can be used for inferring a hierarchical tree. The paper also demonstrates the advantages of using the DR formulation, which was introduced in paper II. Specifically, DR formulation leads to improved hierarchical inference performance and better hierarchy-informed performance measures (i.e., measures considering predefined hierarchical structures) compared to the SD-softmax formulation.

6.2 Future works

I have devised possible applications and methods for adapting metric learning models that require further investigation. These are now summarized as possible future works.

6.2.1 Improving discriminality of an adversarially robust model

Although further investigation is required, Wu et al. [2021] suggested that the reduction of discriminality could be an important factor in the accuracy-robustness tradeoff so I employed this in Chapter 5. If true, it will be worth exploring metric learning methods that increase discriminality such as the NormFace model [Wang et al., 2017] and adaptive scaling factor [Zhang et al., 2019b] (both are explained in section 2.2). Specifically, NormFace loss can be used for training natural data points x with class c, with an additional loss (such as the second term in expression [3.7]) for adversarial robustness.

Let $\measuredangle(v_1, v_2)$ be the angle between two unit vectors v_1 and v_2, and $\measuredangle_{\min}(c)$ be the smallest angle from the proxy $\tilde{W}_c$ of class c to a different proxy $\tilde{W}_{y'}$, formalized as $\measuredangle_{\min}(c) = \min_{y' \in \mathcal{Y}, y' \neq c} \measuredangle(\tilde{W}_c, \tilde{W}_{y'})$. The additional loss for adversarial robustness should then enforce the following condition:

$$\measuredangle(\tilde{f}(\hat{x^*}), \tilde{W}_c) < \frac{\measuredangle_{\min}(c)}{2},$$

where $\hat{x^*}$ is an adversarially perturbed data point using an attack algorithm. This condition will classify $\hat{x^*}$ as class c without moving $\tilde{f}(\hat{x^*})$ too closely to the proxy W_c. Given the decision boundary of ArcFace [Deng et al., 2019], which can be written as $\measuredangle(\tilde{f}(\hat{x^*}), \tilde{W}_c) \leq \measuredangle(\tilde{f}(\hat{x^*}), \tilde{W}_{y'}) - m$, the condition can be met by using ArcFace loss for $\hat{x^*}$ with negative margin $m(c) = -\frac{\measuredangle_{\min}(c)}{2} + \epsilon_{\text{tiny}}$ for a tiny number $\epsilon_{\text{tiny}} > 0$ (a more complex formulation might be needed depending on the range of angles).

6.2.2 Estimation and enhancement of intra-class variance with metric learning

Metric learning methods learn feature representations such that classification can be geometrically understood and learned distances resemble semantic dissimilarity. Lin et al. [2018] analyzed a learned metric learning model trained on an image dataset by subtracting the corresponding prototypes from their feature vectors to obtain residual feature vectors. They observed that regardless of their classes, the residual feature

vectors of data points with similar poses are located closely on the residual (feature) space. This observation indicates that metric learning models can automatically learn intra-class variances such as image poses. When no augmentation is applied for training, analyzing a metric learning model would enable researchers to estimate existing intra-class variances.

How can the estimated intra-class variance of data be utilized? Let us consider two image data points x_1 and x_2 with the same class. One of these (x_2 for example) can be obtained by modifying characteristics, such as the pose, angle, and color, of the other data point. Thus, one may consider that data points are obtained by augmenting different data points with certain combinations of transformations. In this view, existing intra-class variance in the data reflects the data generation process. Hence, with intra-class variance estimated, we can apply augmentation that mimics the data generation process. Doing so will enable the distribution of augmented data points to mimic the distribution of actual data points, but with increased variance. This can be helpful in improving model performance as a trained model with such augmentation may also be adept at classifying test data points. Further investigation will be required to verify the effectiveness of such augmentation.

6.2.3 Re-weighting for direct modification of a feature space

As explained in section 2.4, previous analyses [Kornblith et al., 2021; Liu et al., 2020; Roth et al., 2020] of learned representations have revealed that characteristics such as discriminative power ϕ and DoV affect the generalization power of metric learning models on downstream tasks. Several approaches have been suggested to change these characteristics in the training phase. These include using negative margin [Liu et al., 2020] and regularization to avoid excessive compression of representation [Roth et al., 2020].

However, instead of modification during the training phase, we could *directly modify the learned representations to change the characteristics of a feature space.* For instance, for Euclidean embedding, we can apply principal component analysis (PCA) to identify the principal directions of features. After centering the features, a feature vector $f(x)$ can be represented by the obtained directions as follows:

$$f(x) = \sum_{i=1}^{d_F} (f(x) \cdot u_i) u_i,$$

where u_i is i-th (unit) principal direction. We then simply apply weights for each principal direction to obtain a modified feature $f_{\text{new}}(x)$. Mathematically, modified feature

$f_{\text{new}}(x)$ can be defined as:

$$f_{\text{new}}(x) = \sum_{i=1}^{d_F} w_i (f(x) \cdot u_i) u_i,$$

where $w_i \geq 0$ is the weight for the i-th principal direction. When the feature vectors are concentrated on the corresponding class representatives, class representatives and features will be roughly spanned by a small number of principal directions. In this case, setting small weights for the first few principal directions and larger weights for the rest would reduce the discriminative power of the representation by spreading feature positions. Such post hoc change has an advantage in that it does not require retraining of feature mapping when we encounter multiple tasks with varying distribution shifts. Future work should investigate whether this proposed re-weighting can effectively change discriminative power and DoV. One of the important investigations will be to verify whether the re-weighting method can improve performances even with the distortion of feature space. If it is confirmed that this is negative, it will suggest that there is an unknown characteristic of learned features that affect performance which require further investigation.

Bibliography

J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. *Advances in neural information processing systems*, 31, 2018.

B. Barz and J. Denzler. Hierarchy-based image embeddings for semantic image retrieval. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 638–647. IEEE, 2019.

L. Bertinetto, R. Mueller, K. Tertikas, S. Samangooei, and N. A. Lord. Making better mistakes: Leveraging class hierarchies with deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12506–12515, 2020.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

K. Bykov, A. Hedström, S. Nakajima, and M. M.-C. Höhne. Noisegrad—enhancing explanations by introducing stochasticity to model weights. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6132–6140, 2022.

N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

Y. Carmon, A. Raghunathan, L. Schmidt, J. C. Duchi, and P. S. Liang. Unlabeled data improves adversarial robustness. *Advances in Neural Information Processing Systems*, 32, 2019.

J. Chen, M. I. Jordan, and M. J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 ieee symposium on security and privacy (sp)*, pages 1277–1294. IEEE, 2020a.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020b.

J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

F. Croce and M. Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. In *International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=rklk_ySYPB`.

F. Croce and M. Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pages 2196–2205. PMLR, 2020b.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

J. Deng, J. Guo, N. Xue, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

J. Deng, J. Guo, T. Liu, M. Gong, and S. Zafeiriou. Sub-center arcface: Boosting face recognition by large-scale noisy web faces. In *European Conference on Computer Vision*, pages 741–757. Springer, 2020.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, B. Tran, and A. Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

A. Ermolov, L. Mirvakhabova, V. Khrulkov, N. Sebe, and I. Oseledets. Hyperbolic vision transformers: Combining improvements in metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7409–7419, 2022.

M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

V. S. F. Garnot and L. Landrieu. Leveraging class hierarchies with metric-guided prototype learning. In *British Machine Vision Conference (BMVC)*, 2021.

L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

A. Ghiasi, A. Shafahi, and T. Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. In *International Conference on Learning Representations*, 2019.

A. Ghorbani, A. Abid, and J. Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3681–3688, 2019.

S. Godbole. Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multi-way classifiers. 2002.

J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17:513–520, 2004.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *Advances in neural information processing systems*, 30, 2017.

M. Jayathilaka, T. Mu, and U. Sattler. Towards knowledge-aware few-shot learning with ontology-based n-ball concept embeddings. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 292–297. IEEE, 2021.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

M. Khoury and D. Hadfield-Menell. Adversarial training with voronoi constraints. *arXiv preprint arXiv:1905.01019*, 2019.

V. Khrulkov, L. Mirvakhabova, E. Ustinova, I. Oseledets, and V. Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6428, 2020.

J. Kim and X. Wang. Sensible adversarial learning. 2019.

S. Kim, D. Kim, M. Cho, and S. Kwak. Proxy anchor loss for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3238–3247, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Z. Kong, J. Guo, A. Li, and C. Liu. Physgan: Generating physical-world-resilient adversarial examples for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

S. Kornblith, T. Chen, H. Lee, and M. Norouzi. Why do better loss functions lead to less transferable features? *Advances in Neural Information Processing Systems*, 34: 28648–28662, 2021.

J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, July 2016. URL `http://arxiv.org/abs/1607.02533`. arXiv: 1607.02533.

C. Laidlaw, S. Singla, and S. Feizi. Perceptual adversarial robustness: Defense against unseen threat models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=dFwBosAcJkN`.

H. Lakkaraju, N. Arsov, and O. Bastani. Robust and stable black box explanations. In *International Conference on Machine Learning*, pages 5628–5638. PMLR, 2020.

Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

X. Lin, Y. Duan, Q. Dong, J. Lu, and J. Zhou. Deep variational metric learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 689–704, 2018.

B. Liu, Y. Cao, Y. Lin, Q. Li, Z. Zhang, M. Long, and H. Hu. Negative margin matters: Understanding margin in few-shot classification. *arXiv preprint arXiv:2003.12060*, 2020.

W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

P. Mettes, E. van der Pol, and C. Snoek. Hyperspherical prototype networks. *Advances in neural information processing systems*, 32, 2019.

G. A. Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital signal processing*, 73:1–15, 2018.

S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *arXiv preprint arXiv:1511.04599*, 2015.

K. Musgrave, S. Belongie, and S.-N. Lim. A metric learning reality check. In *European Conference on Computer Vision*, pages 681–699. Springer, 2020.

P. Nakkiran. Adversarial robustness may be at odds with simplicity. *arXiv preprint arXiv:1901.00532*, 2019.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. URL `http://ufldl.stanford.edu/housenumbers`.

H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

OpenAI. Chatgpt, 2022. URL `https://chat.openai.com/chat`. Retrieved on 2023-02-06.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

Q. Qian, L. Shang, B. Sun, J. Hu, H. Li, and R. Jin. Softtriple loss: Deep metric learning without triplet sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6450–6458, 2019.

A. Raghunathan, S. M. Xie, F. Yang, J. Duchi, and P. Liang. Understanding and mitigating the tradeoff between robustness and accuracy. *arXiv preprint arXiv:2002.10716*, 2020.

A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

K. Roth, T. Milbich, S. Sinha, P. Gupta, B. Ommer, and J. P. Cohen. Revisiting training strategies and generalization performance in deep metric learning. In *International Conference on Machine Learning*, pages 8242–8252. PMLR, 2020.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

F. Sala, C. De Sa, A. Gu, and C. Ré. Representation tradeoffs for hyperbolic embeddings. In *International conference on machine learning*, pages 4460–4469. PMLR, 2018.

H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry. Do adversarially robust imagenet models transfer better? *Advances in Neural Information Processing Systems*, 33:3533–3545, 2020.

R. Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane. In *Graph Drawing: 19th International Symposium, GD 2011, Eindhoven, The Netherlands, September 21-23, 2011, Revised Selected Papers 19*, pages 355–366. Springer, 2012.

L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry. Adversarially robust generalization requires more data. *Advances in neural information processing systems*, 31, 2018.

L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on mnist. *arXiv preprint arXiv:1805.09190*, 2018a.

L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on MNIST. *arXiv:1805.09190 [cs]*, May 2018b. URL `http://arxiv.org/abs/1805.09190`. arXiv: 1805.09190.

F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. *arXiv:1503.03832 [cs]*, pages 815–823, June 2015. doi: 10.1109/CVPR.2015.7298682. URL `http://arxiv.org/abs/1503.03832`. arXiv: 1503.03832.

E. Schubert. A triangle inequality for cosine similarity. In *Similarity Search and Applications: 14th International Conference, SISAP 2021, Dortmund, Germany, September 29–October 1, 2021, Proceedings*, pages 32–44. Springer, 2021.

M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 1528–1540, 2016.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 30:4077–4087, 2017.

A. S. Suggala, A. Prasad, V. Nagarajan, and P. Ravikumar. Revisiting adversarial risk. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2331–2339. PMLR, 2019.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

F. Tramer and D. Boneh. Adversarial training and robustness for multiple perturbations. *Advances in Neural Information Processing Systems*, 32, 2019.

D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. *California Institute of Technology*, 2011.

F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1041–1049, 2017.

H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018.

J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

E. Wong, F. Schmidt, and Z. Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning*, pages 6808–6817. PMLR, 2019.

Z. Wu, H. Gao, S. Zhang, and Y. Gao. Understanding the robustness-accuracy tradeoff by rethinking robust fairness. 2021.

J. Yan, L. Luo, C. Deng, and H. Huang. Unsupervised hyperbolic metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12465–12474, 2021.

Y.-Y. Yang, C. Rashtchian, Y. Wang, and K. Chaudhuri. Robustness for non-parametric classification: A generic attack and defense. In *International Conference on Artificial Intelligence and Statistics*, pages 941–951. PMLR, 2020a.

Y.-Y. Yang, C. Rashtchian, H. Zhang, R. R. Salakhutdinov, and K. Chaudhuri. A closer look at accuracy vs. robustness. *Advances in Neural Information Processing Systems*, 33, 2020b.

A. Zhai and H.-Y. Wu. Classification is a strong baseline for deep metric learning. *British Machine Vision Conference (BMVC)*, 2019.

H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019a.

R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

X. Zhang, R. Zhao, Y. Qiao, X. Wang, and H. Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10823–10832, 2019b.

X. Zhe, L. Ou-Yang, and H. Yan. Improve l2-normalized softmax with exponential moving average. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2019.

Chapter 7

Scientific results

Article I

I Beyond image classification: zooplankton identification with deep vector space embeddings

Ketil Malde and **Hyeongji Kim**. *arXiv preprint: 1909.11380*, 2019.

Beyond image classification: zooplankton identification with deep vector space embeddings

Ketil Malde and Hyeongji Kim

Institute of Marine Research, PB-1507 Nordnes, Bergen, Norway

Department of Informatics, University of Bergen, Norway

September 26, 2019

Abstract

Zooplankton images, like many other real world data types, have intrinsic properties that make the design of effective classification systems difficult. For instance, the number of classes encountered in practical settings is potentially very large, and classes can be ambiguous or overlap. In addition, the choice of taxonomy often differs between researchers and between institutions. Although high accuracy has been achieved in benchmarks using standard classifier architectures, biases caused by an inflexible classification scheme can have profound effects when the output is used in ecosystem assessments and monitoring.

Here, we propose using a deep convolutional network to construct a vector embedding of zooplankton images. The system maps (embeds) each image into a high-dimensional Euclidean space so that distances between vectors reflect semantic relationships between images. We show that the embedding can be used to derive classifications with comparable accuracy to a specific classifier, but that it simultaneously reveals important structures in the data. Furthermore, we apply the embedding to new classes previously unseen by the system, and evaluate its classification performance in such cases.

Traditional neural network classifiers perform well when the classes are clearly defined *a priori* and have sufficiently large labeled data sets available. For practical cases in ecology as well as in many other fields this is not the case, and we argue that the vector embedding method presented here is a more appropriate approach.

1 Introduction

In classification problems, the goal is to map each input to one of a discrete set of classes. A typical example is labeling images according to objects pictured, e.g., distinguishing pictures of cats from pictures of dogs. The output of a classifier

can be a single value, but is often a vector where each element represents the classifier's confidence that the input belongs to the corresponding class.

Recently, deep neural networks have been used with great success for many classification tasks. Often, these classifiers apply a softmax function (a generalization of the logistic function to multiple outputs) to generate the final output. This scales the output vector so that the scores for the classes sum to one, resembling a set of probabilities for the class assignment.

This approach is commonly used for image classification, where it has been overwhelmingly successful for many benchmark data sets. Yet, it relies on a set of assumptions that can be naive in many practical situations. Here, we will use the classification of zooplankton images to illustrate why a vector space embedding can be a more appropriate approach.

1.1 Zooplankton classification

Plankton constitute a fundamental component of aquatic ecosystems, and since they form the basis for many food chains and also rapidly adapt to changes in the environment, monitoring plankton diversity and abundances is a central input to marine science and management [ICES, 2018].

Imaging systems are being deployed to scale up sampling efforts [Stemmann and Boss, 2012, Benfield et al., 2007], but the manual curation process remains expensive and time consuming [ICES, 2018]. Recently, automated classifiers based on deep neural networks have been developed and applied successfully to benchmark problem sets, but deployment in a practical marine management situation poses some challenges.

For standard classifiers, the set of target classes is an integral part of the structure of the classifier. In other words, the set of target classes must be finite and known in advance. In contrast, plankton communities often consists of surprisingly large numbers of species (e.g., [Huisman and Weissing, 1999, Schippers et al., 2001]), with highly varying abundance. Even if all species were known, many would not be represented in the training data, and the long-tailed abundance distribution poses a challenge to standard methods [Van Horn and Perona, 2017]. A further complication is the various forms of artifacts, including detritus, clusters of multiple specimens, and pieces of fragile plankton that break apart during processing [Benfield et al., 2007].

In addition, different researchers may operate with different taxonomies, or otherwise suffer from inconsistent annotation [Malde et al., 2019]. It is symptomatic that comparing the ZooScan data set used here with another, similar data set [Orenstein et al., 2015] with around 100 classes, we find that only three of the classes are shared. Two of those represent artifacts (*bubble* and *detritus*), and only one plankton taxon (*coscinodiscus*) was present as a class in both data sets. While it is possible to train classifiers separately for each taxonomy, this diminishes the total value of the data and inhibits comparisons and reproducibility.

Several automated systems for plankton classification have been developed and applied to benchmark data sets (e.g, [Luo et al., 2018, Dai et al., 2016,

Lee et al., 2016]), but report problems stemming from the severe class imbalance in the data. In addition, image quality is often poor, and image sizes can vary enormously. In practice, automation is still mainly used to aid or supplement a manual curation process [Uusitalo et al., 2016]. For interactive processes, methods that reveal more of the structure of the data are more useful than categorical class assignments [ICES, 2018].

1.2 Vector embeddings as an alternative

Here we explore *vector embedding* of the input space as an alternative to the standard approach. Each input is mapped to a vector in a high-dimensional space with no *a priori* relationship between classes and dimensions. Instead, the mapping (or embedding) is constructed to reflect some concept of similarity between inputs. In our case, class membership represents similarity, and the goal of the embedding is to map inputs from the same class to vectors that are close to each other, and inputs belonging to different classes to vectors that are farther apart.

Compared to traditional classification, the embedding models the structure of the input space with high resolution. This is important when the system deals with new classes of inputs. Whether two inputs belong to the same or different classes can be determined solely from the distance between their corresponding vector space embeddings. Similarly, new classes can be constructed based on clusters or other structure in the embedding vector space, without retraining or other modifications to the system.

One application where neural networks that output embeddings have been applied with particular success, is face recognition [Taigman et al., 2014, Schroff et al., 2015]. Not unlike plankton classification, the goal is to identify a large number of classes (for face recognition, each individual person represent one class). Thus we have a classification problem with an unknown, large, and possibly open-ended number of classes, often with very sparse data and poor annotation. As for face recognition, it is important to be able to identify classes from few samples, so called *low-shot*, *one-shot* [Fei-Fei et al., 2006], and *zero-shot* [Larochelle et al., 2008, Yu and Aloimonos, 2010] classification.

Inspired by this, we here apply a vector embedding approach to the task of classifying zooplankton images, and compare the results to using a straightforward classifier based on the Inception v3 [Szegedy et al., 2016] neural network architecture. We show how classes form clusters in the embedding space, discuss confoundings, and explore how the vector embedding performs on previously unseen classes.

2 Methods

2.1 Data set

Recently, a large set of ZooScan [Gorsky et al., 2010, Grosjean et al., 2004] images of plankton was made available to the public [Elineau et al., 2018]. The data set consists of monochromatic images organized into 93 categories, most of them representing zooplankton taxa. In addition, several error categories exist, with names like *artefact*, *detritus*, and *bubble*. Abundances range from the 39 images labeled *Ctenophora*, up to the 511,700 labeled *detritus*. Three of the four most abundant categories represent various types of artifact.

The images vary widely in size. We converted the images to a standard size of 299x299 pixels. Smaller images were padded up to this size, while larger images were scaled down. The resized images were then used to construct data sets for training, validation, and testing. For training, we used 65 non-artifact classes with abundances above 500, in addition to *bubble*. From each class, 100 random images were selected to serve as a validation set, and then another 100 images for the test set. The remaining images constituted the training set.

A second test set consisted of 100 images sampled randomly from each of the 38 classes not represented in the other sets. For the classes with less than 100 images, all images were used.

2.2 Standard neural network classifier

To provide a baseline for achievable classification accuracy, we used the convolutional neural network Inception v3, initialized with weights pre-trained on the ImageNet data set [Deng et al., 2009]. The default 1000-class output layer was replaced with a 65-class softmax output to match the number of classes.

The network was trained using the SGD optimizer with a learning rate of 0.0001 and momentum of 0.9, using a categorical cross-entropy cost function. During training, mean square error and accuracy were reported.

All neural networks were implemented using Keras [Chollet et al., 2015] with a Tensorflow [Abadi et al., 2016] backend, and run on a computer with RTX2080 Ti GPU accelerators (Nvidia Corporation, Santa Clara, California, USA).

2.3 Siamese networks

The particular embedding technique we will investigate here is called *siamese networks* [Bromley et al., 1994, Hoffer and Ailon, 2015, Wang et al., 2014], in a variant using what is called a triplet loss function. The network is given three inputs, one from a randomly selected class (the *anchor*), one randomly sampled from the same class (the *positive*) and a random sample from another class (the *negative*). The cost function J is designed to reward a small distances from the anchor to the positive and a large distance from the anchor to the negative.

$J(a, p, n) = \max(0, ||a - p||^2 - ||a - n||^2 + \alpha)$

The parameter α serves as a margin to avoid the network learning a trivial, zero-cost solution of embedding all inputs in the same point.

For vector space embedding, we again used Inception v3, but replaced the softmax with a global average pooling layer and a 128-dimensional vector output layer. The output vector was further constrained to unit length, so that the vector embedding results in a point on a hypersphere with a radius of one.

Training was performed using the SGD optimizer and a batch size of 20. The learning rate was set to decay of 0.9 and an initial value of 0.01. The margin parameter α was initially set to 1.0, but raised to 1.3 after 20 iterations, and to 1.5 after 30 iterations.

2.4 Classification from a vector space embedding

A vector space embedding does not directly present a classification, but we can use any of a number of methods suitable for euclidean spaces. An advantage of vector space embeddings is to allow the use of unsupervised methods, and when no known data is available, classes can be determined using standard approaches like k-means clustering.

Here, we will compare classifications in the embedding space using two simple supervised methods. First, using data with known classes we calculate the centroids for each class and assign new data to the class represented by the closest centroid. Alternatively, we use nearest neighbor classification (kNN, using the approximative algorithm BallTree from Scikit-learn [Pedregosa et al., 2011]) against data with known classifications.

3 Results

3.1 Baseline classification

Inception v3 was trained for 220 epochs on the 65-class training data set, the metrics are shown in Fig. 1. The classifier reaches 80% accuracy on validation data after 67 epochs, and appears to converge to approximately 86% accuracy after around 150 epochs.

We select the classifier trained for 200 epochs, and use it to classify the test set. Total accuracy was 87.7%, a table with more detailed results for the different classes can be found as supplementary information.

3.2 Training the vector embedding

For validation, we calculated the centroid of the embeddings for each category of plankton. We define the cluster radius to be the average distance from the centroid for each image in the validation set. During training, we calculate the cluster radius (Suppl. Fig 1) and the change in centroid (Suppl. Fig 2) for every class in the training set. As training progresses, cluster radii shrink, while the magnitude of the changes to the embedding decreases. In some cases, large

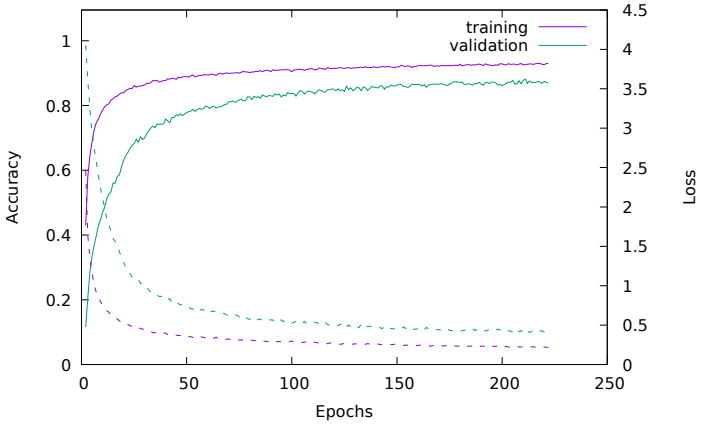


Figure 1: Training Inception v3 for plankton classification. The loss (solid) and accuracy (dashed) for training (purple) and validation (green) data are shown as training progresses.

magnitude changes affect many or all clusters simultaneously, indicating larger scale rearrangements in the embedding.

We can also check if we are able to correctly predict the correct class by assigning each image to the closest centroid. The results are shown in Suppl Fig. 3. Both analyses show rapid improvement for 10 iterations, slower gains the next 20, and only small improvements after 30 iterations. In the following, we use the network trained for 30 iterations to construct the vector embeddings.

3.3 Clusters in the embedding space

As training progresses, clusters start to emerge in the embedding space. A t-SNE [Maaten and Hinton, 2008] rendering is shown in Fig. 2, where the structure of the input data is evident.

3.4 Classification in the embedding space

For classification using kNN, we investigate possible choices for the parameter k. We split the validation data set in two (50 instances for each class in each partition), and used one partition as a reference to classify the other. Experimenting with different values of k indicates that $k = 10$ might be a good value to use (see supplementary figure).

Fig. 3 shows the F1 scores using the default classifier on the whole data set. In addition, we show the centroid-based classification in the embedded space

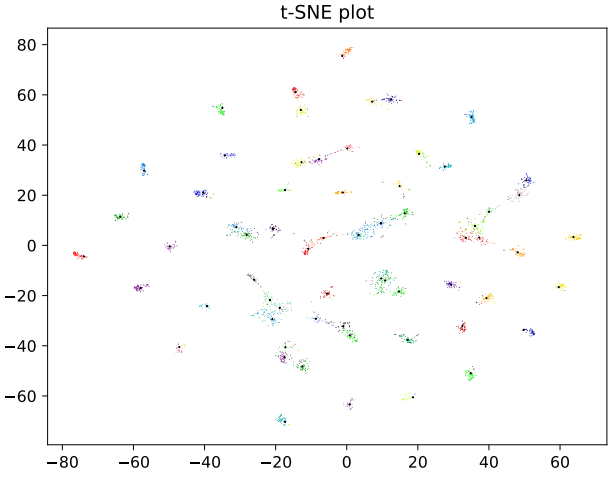


Figure 2: The data projected into the embedding space and displayed using a t-SNE rendering. Class centroids are marked by black crosses.

and kNN classification using various values of k, splitting the test set into equal partitions for reference and an evaluation.

We see that performance is comparable across most classes, but there are some classes where the standard classifier gives different performance from the embedding. The standard classifier outperforms the embedding for *nauplii__Cursacea* (class 65, F1 scores of 0.93 and 0.69) and *nauplii__Cirripedia* (class 12, F1 0.97 and 0.51). A substantial difference is also observed for *egg__Cavolinia_inflexa* (class 64, F1 0.93 and 0.33) and *egg__Actinopterygii* (class 17, F1 0.94 and 0.70). In contrast, the embedding has better performance for *Calanoida* (class 36, F1 0.50 and 0.96) and *larvae__Crustacea* (class 36, F1 0.62 and 0.84).

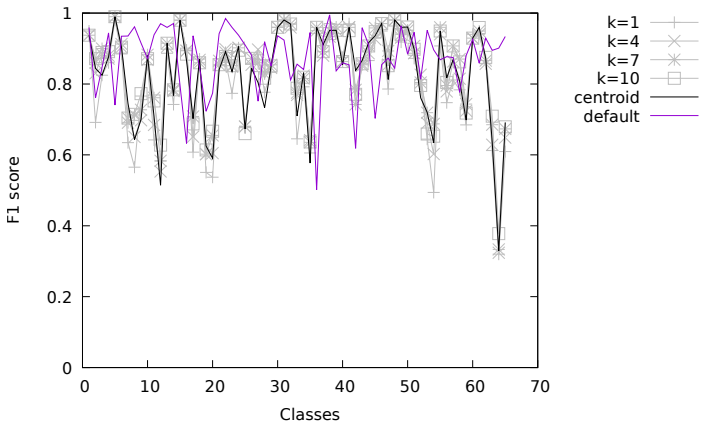


Figure 3: F1-scores across classes in the test set for centroid-based classification in black and kNN using various values for k in gray. The standard classifier perfomance is shown in purple.

To elucidate the misclassifications, the ten most commonly occurring confoundings with kNN ($k = 10$) are shown in Table 1.

Not unexpectedly, confoundings occur between classes of organism fragments or parts. The most commonly occurring confounding consists of the two classes of tails, and confounding *Chaetognatha* with the class of its tails is the third most common occurrence (see also Fig. 4, middle row). In addition, species are confounded with their different stages, e.g., we see confounding between different forms of the *Diphyidae* species (Fig. 4, top row).

We also see pairs of similar species being confounded with each other (e.g., *Oncaeidae* with *Harpacticoida*, and *Eucalanidae* with *Rhincalanidae*).

Table 1: Commonly seen confoundings in the test data set.

True class	**Predicted class**	**rate**
tail__Appendicularia	*tail__Chaetognatha*	0.380
Oncaeidae	*Harpacticoida*	0.260
Chaetognatha	*tail__Chaetognatha*	0.260
Euchaetidae	*Candaciidae*	0.220
Eucalanidae	*Rhincalanidae*	0.200
Harpacticoida	*Oncaeidae*	0.180
nectophore__Diphyidae	*gonophore__Diphyidae*	0.180
Rhincalanidae	*Eucalanidae*	0.180
Centropagidae	*Euchaetidae*	0.160
Limacidae	*Limacinidae*	0.160

Figure 4: Example of plankton images that are difficult to resolve. Upper row, from the left: *Decapoda*, *zooa__Decapoda*, and *larvae__Crustacea*. Middle row: *tail__Appendicularia*, *tail__Chaetognatha*, and *Chaetognatha*. Second row shows variants of *Abylopsis*, from the left: *eudoxie*, *gonophore*, and *nectophore*.

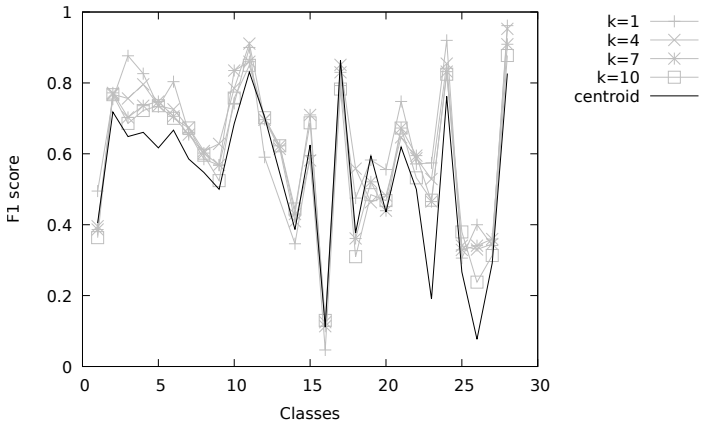


Figure 5: F1-scores across classes for centroid-based classification (black) and kNN using various values for k (gray).

3.5 Previously unseen classes

For the previously unseen classes, we use the same approach of dividing the test set in two and using one part for reference and the other for evaluation. The results are shown in Fig. 3. Here we see that performance is highly variable. Using centroid classification, the highest performing classes were Rhopalonema (number 17, F1 0.86), badfocus_artifact (number 28, F1 0.82), and egg_other (number 11, 0.83). The lowest scoring classes were Euchirella (number 26, F1 0.07), Aglaura (number 23, 0.19), and multiple__other (number 27, F1 0.29). Several low performing classes are caused by confusing the *Abylopsis_tetragona* variants (number 14, gonophore, F1 0.38, number 16 eudoxie, F1 0.11, and number 25, nectophore, F1 0.26). kNN classifications outperforms centroids slightly for several classes, but the overall picture remains the same.

Again we see that a large fraction of the confoundings occur between variants of species, in particular *Abylopsis tetragona* (Fig. 4, bottom row). In addition, there is several cases of confounding between artifact classes.

4 Discussion

Using the average F1 score over the classes, the standard deep learning classifier achieves a score of 0.87 on the data set. Using our vector space embedding and classifying using kNN (k=10), we achieve a score of 0.84. The standard classifier thus outperforms the embedding, but not by a large margin.

Table 2: Commonly seen confoundings in previously unseen classes.

True class	Predicted class	rate
eudoxie__Abylopsis_tetrag	*nectophore__Abylopsis_tet*	0.320
Scyphozoa	*ephyra*	0.280
Rhopalonema	*Aglaura*	0.260
gonophore__Abylopsis_tetr	*nectophore__Abylopsis_tet*	0.240
Calocalanus pavo	*Euchirella*	0.240
nectophore__Abylopsis_tet	*eudoxie__Abylopsis_tetrag*	0.220
badfocus__artefact	*detritus*	0.200
Calocalanus pavo	*part__Copepoda*	0.180
Echinoidea	*larvae__Annelida*	0.180
artefact	*badfocus__artefact*	0.180

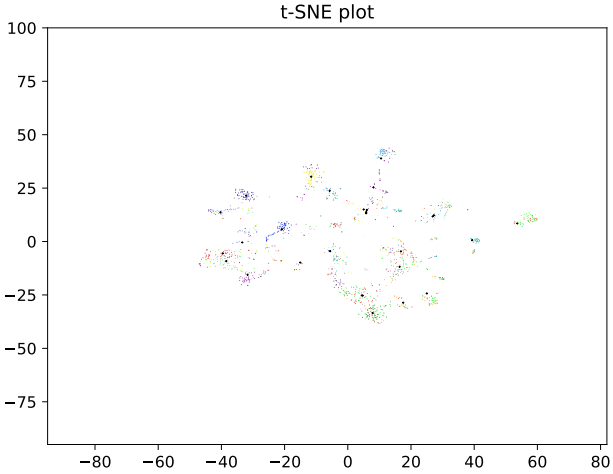


Figure 6: A t-SNE rendering of the data in the unseen classes after 30 iterations.

Interestingly, the vector space embedding performs better on several classes. The standard classifier often mislabels many species as *Calanoida*, resulting in a low F1 score of 0.50, while the embedding classifier achieves an F1 score of 0.96 for this class. In contrast, the standard classifier appears to be better at precisely separating classes with very similar morphology, for instance classes of eggs or nauplii. As similar classes are embedded close to each other, they are more difficult to differentiate. Although the proximity is semantically meaningful, this reduces accuracy somewhat. For maximizing absolute classification performance, an ensemble using both methods is likely to be optimal.

In contrast to classification, the embedding is able to better capture the underlying structure of the data. This has many potential uses, for instance to identify misclassified data, or to allow switching to a different taxonomy. In this way, the embedding can be used actively to evaluate and even refine the choice of classes used.

As a more challenging test case, we applied the embedding approach to data in classes not present in the training data. Here we achieve a more modest performance, with an average F1 score of 0.61. Some of the classes gave particularly poor results, while other classes were accurately identified. Even for classes where performance is too low to be used directly, the information provided by the embedding can guide and accelerate manual or semi-interactive processing. We believe training with more diverse data is likely to improve generality of the embedding.

The use of very simple schemes used to compare classification performance in the embedding space (*i.e.*, centroid clustering and kNN) is a deliberate choice. More complex schemes may be able to give better classification performance, but our goal here is to emphasize the ability of the embedding to capture the structure of the input. Using a complex non-linear classifier on the embedding vectors would defeat this purpose, since it would be more difficult to separate complexity captured by the embedding from complexity captured by the final classification stage.

5 Conclusions

Classification of zooplankton is an important task, but the inherent complexity and other limitations of the data requires more flexibility than that provided by standard classifiers. Earlier attempts have successfully been able to classify benchmark data sets [Py et al., 2016, Lee et al., 2016], but achieve high accuracy at the expense of removing low abundance or otherwise difficult classes [Luo et al., 2018].

Here we have shown that using a deep learning vector space embedding, we can model important structure in the data, while retaining the flexibility to perform classification with accuracy comparable to state of the art classifiers.

6 Author's contributions

KM conceived of the ideas and methodology and led the writing of the manuscript, HK implemented benchmarks and visualizations of the results. Both authors contributed critically to the draft and approved of its publication.

7 Availability

The data set and software used here is publicly available as described above. Source code for network construction, training, and analysis can be found as GitHub repositories at

https://github.com/ketil-malde/plankton-siamese and https://github.com/ketil-malde/plankton-learn

An interactive rendering of the data sets and classifications using https://projector.tensorflow.org/ can be found here:

https://projector.tensorflow.org/?config=https://home.malde.org/vector_embeddings/

References

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[Benfield et al., 2007] Benfield, M. C., Grosjean, P., Culverhouse, P. F., Irigoien, X., Sieracki, M. E., Lopez-Urrutia, A., Dam, H. G., Hu, Q., Davis, C. S., Hansen, A., et al. (2007). Rapid: research on automated plankton identification. *Oceanography*, 20(2):172–187.

[Bromley et al., 1994] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. https://keras.io.

[Dai et al., 2016] Dai, J., Wang, R., Zheng, H., Ji, G., and Qiao, X. (2016). Zooplanktonet: Deep convolutional network for zooplankton classification. In *OCEANS 2016-Shanghai*, pages 1–6. IEEE.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Elineau et al., 2018] Elineau, A., Desnos, C., Jalabert, L., Olivier, M., Romagnan, J.-B., Brandao, M., Lombard, F., Llopis, N., Courboulès, J., Caray-Counil, L., Serranito, B., Irisson, J.-O., Picheral, M., Gorsky, G., and Stemmann, L. (2018). Zooscannet: plankton images captured with the zooscan.

[Fei-Fei et al., 2006] Fei-Fei, L., Fergus, R., and Perona, P. (2006). One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611.

[Gorsky et al., 2010] Gorsky, G., Ohman, M. D., Picheral, M., Gasparini, S., Stemmann, L., Romagnan, J.-B., Cawood, A., Pesant, S., García-Comas, C., and Prejger, F. (2010). Digital zooplankton image analysis using the zooscan integrated system. *Journal of plankton research*, 32(3):285–303.

[Grosjean et al., 2004] Grosjean, P., Picheral, M., Warembourg, C., and Gorsky, G. (2004). Enumeration, measurement, and identification of net zooplankton samples using the zooscan digital imaging system. *ICES Journal of Marine Science*, 61(4):518–525.

[Hoffer and Ailon, 2015] Hoffer, E. and Ailon, N. (2015). Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer.

[Huisman and Weissing, 1999] Huisman, J. and Weissing, F. J. (1999). Biodiversity of plankton by species oscillations and chaos. *Nature*, 402(6760):407.

[ICES, 2018] ICES (2018). WKMLEARN: Report of the workshop on machine learning in marine science. Technical Report ICES CM 2018/EOSG:20, International Council for Exploration of the Seas.

[Larochelle et al., 2008] Larochelle, H., Erhan, D., and Bengio, Y. (2008). Zero-data learning of new tasks. In *AAAI*, volume 1, page 3.

[Lee et al., 2016] Lee, H., Park, M., and Kim, J. (2016). Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning. In *2016 IEEE international conference on image processing (ICIP)*, pages 3713–3717. IEEE.

[Luo et al., 2018] Luo, J. Y., Irisson, J.-O., Graham, B., Guigand, C., Sarafraz, A., Mader, C., and Cowen, R. K. (2018). Automated plankton image analysis using convolutional neural networks. *Limnology and Oceanography: Methods*, 16(12):814–827.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Malde et al., 2019] Malde, K., Handegard, N. O., Eikvil, L., and Salberg, A.-B. (2019). Machine intelligence and the data-driven future of marine science. *ICES Journal of Marine Science*.

[Orenstein et al., 2015] Orenstein, E. C., Beijbom, O., Peacock, E. E., and Sosik, H. M. (2015). Whoi-plankton- A large scale fine grained visual recognition benchmark dataset for plankton classification. *CoRR*, abs/1510.00745.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Py et al., 2016] Py, O., Hong, H., and Zhongzhi, S. (2016). Plankton classification with deep convolutional neural networks. In *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pages 132–136. IEEE.

[Schippers et al., 2001] Schippers, P., Verschoor, A. M., Vos, M., and Mooij, W. M. (2001). Does "supersaturated coexistence" resolve the "paradox of the plankton"? *Ecology letters*, 4(5):404–407.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Stemmann and Boss, 2012] Stemmann, L. and Boss, E. (2012). Plankton and particle size and packaging: from determining optical properties to driving the biological pump. *Annual Review of Marine Science*, 4:263–290.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Taigman et al., 2014] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.

[Uusitalo et al., 2016] Uusitalo, L., Fernandes, J. A., Bachiller, E., Tasala, S., and Lehtiniemi, M. (2016). Semi-automated classification method addressing marine strategy framework directive (msfd) zooplankton indicators. *Ecological indicators*, 71:398–405.

[Van Horn and Perona, 2017] Van Horn, G. and Perona, P. (2017). The devil is in the tails: Fine-grained classification in the wild. *arXiv preprint arXiv:1709.01450*.

[Wang et al., 2014] Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393.

[Yu and Aloimonos, 2010] Yu, X. and Aloimonos, Y. (2010). Attribute-based transfer learning for object categorization with zero/one training example. In *European conference on computer vision*, pages 127–140. Springer.

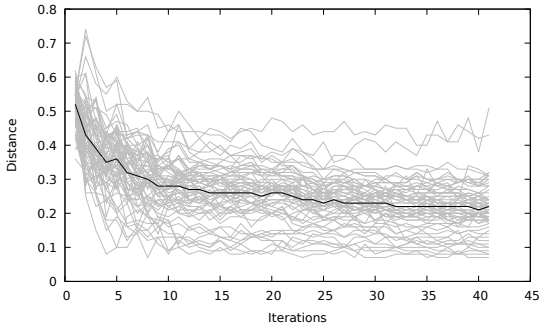


Figure 1: Cluster radius for the 65 individual categories (gray) and the average (black) as training progresses.

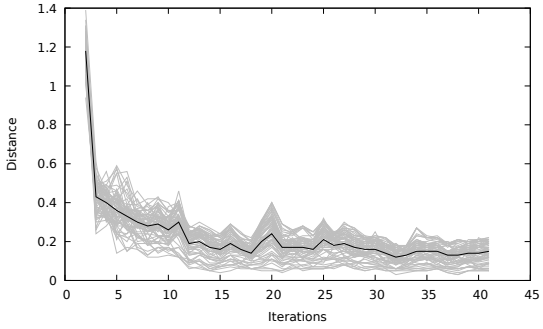


Figure 2: Output change measured as the distance cluster centroids move between iterations. Individual centroids are shown in gray and the average in black.

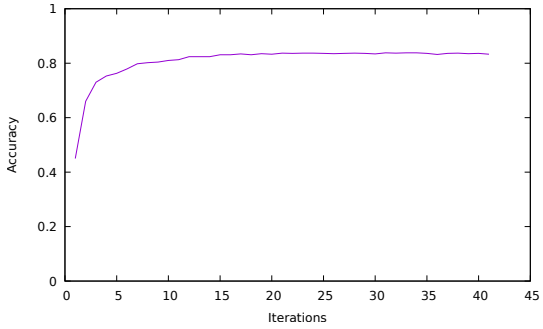


Figure 3: Prediction accuracy from assigning each image to the nearest centroid. Only the validation set is shown. Accuracy plateaus at 0.838 after 30 iterations, and decreases slightly after 35.

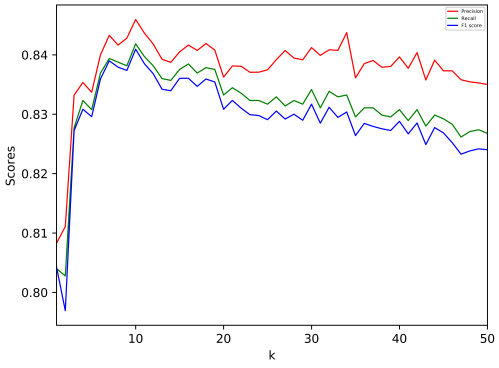


Figure 4: Classification performance using the kNN algorithm for different values of k.

Table 1: Accuracy per class on the test set using Inception v3 for classification. Confounders are the most frequent incorrect labels reported when the error occurs in more than 5% of the cases.

Class	Recall	Confounders	Class	Recall	Confounders
Acantharea	82	Phaeodaria (12)	Harpacticoida	86	Oncaeidae (8)
Acartiidae	93	Calanoida (7)	Hyperiidea	76	Calanoida (7)
Actinopterygii	95		larvae__Crustacea	47	calyptopsis (22)
Annelida	84		Limacidae	58	Limacinidae (36)
Bivalvia__Mollusca	93		Limacinidae	87	
Brachyura	98		Luciferidae	72	Decapoda (15)
bubble	97		megalopa	94	
Calanidae	74	Calanoida (24)	multiple__Copepoda	79	Calanoida (7)
Calanoida	97		nauplii__Cirripedia	97	
calyptopsis	91		nauplii__Crustacea	99	
Candaciidae	74	Calanoida (16)	nectophore__Diphyidae	88	gonophore__Diphyidae (9)
Cavoliniidae	91		nectophore__Physonectae	94	
Centropagidae	60	Calanoida (39)	Neoceratium	90	seaweed (6)
Chaetognatha	96		Noctiluca	97	
tail__Chaetognatha	39		Obelia	95	
Copilia	92		Oikopleuridae	99	
Corycaeidae	94		Oithonidae	95	Calanoida (5)
Coscinodiscus	99		Oncaeidae	89	Corycaeidae (5)
Creseidae	95		Ophiuroidea	93	
cyphonaute	100		Ostracoda	94	
cypris	81	Ostracoda (13)	Penilia	99	
Decapoda	92		Phaeodaria	95	Foraminifera (5)
zoea__Decapoda	6		Podon	76	Evadne (17)
Doliolida	96		Pontellidae	94	
egg__Actinopterygii	95		Rhincalanidae	77	Eucalanidae (21)
egg__Cavolinia_inflexa	82	egg__Actinopterygii (5)	Salpida	92	
Eucalanidae	78	Calanoida (9)	Sapphirinidae	85	
Euchaetidae	65	Calanoida (32)	scale	82	Noctiluca (5)
eudoxie__Diphyidae	80	nectophore__Diphyidae (13)	seaweed	86	
Evadne	95		tail__Appendicularia	85	Chaetognatha (5)
Foraminifera	87		tail__Chaetognatha	49	Chaetognatha (39)
Fritillariidae	87	Oikopleuridae (10)	Temoridae	97	
gonophore__Diphyidae	84	nectophore__Diphyidae (12)	zoea__Decapoda	92	Decapoda (6)
Haloptilus	91	Calanoida (5)			

Article II

II Distance-Ratio-Based Formulation for Metric Learning

Hyeongji Kim, Pekka Parviainen, and Ketil Malde. *arXiv preprint: 2201.08676*, 2022.

DISTANCE-RATIO-BASED FORMULATION FOR METRIC LEARNING

A PREPRINT

Hyeongji Kim*[,2], Pekka Parviainen[2], and Ketil Malde[1,2]

[1]Institute of Marine Research, Bergen, Norway
[2]Department of Informatics, University of Bergen, Norway

January 24, 2022

ABSTRACT

In metric learning, the goal is to learn an embedding so that data points with the same class are close to each other and data points with different classes are far apart. We propose a distance-ratio-based (DR) formulation for metric learning. Like softmax-based formulation for metric learning, it models $p(y = c|x')$, which is a probability that a query point x' belongs to a class c. The DR formulation has two useful properties. First, the corresponding loss is not affected by scale changes of an embedding. Second, it outputs the optimal (maximum or minimum) classification confidence scores on representing points for classes. To demonstrate the effectiveness of our formulation, we conduct few-shot classification experiments using softmax-based and DR formulations on CUB and *mini*-ImageNet datasets. The results show that DR formulation generally enables faster and more stable metric learning than the softmax-based formulation. As a result, using DR formulation achieves improved or comparable generalization performances.

1 Introduction

Modeling probability $p(y = c|x')$, which is a probability that a query point x' belongs to a class c, plays an important role in discriminative models. Standard neural network based classifiers use the softmax activation function to estimate this probability. When $l_c(x')$ is the logit (pre-softmax) value from the network for class c and the point x' and $\mathcal{Y}$ is a set of classes, the softmax function models $p(y = c|x')$ as:

$$\hat{p}(y = c|x') = \frac{\exp(l_c(x'))}{\sum_{y \in \mathcal{Y}} \exp(l_y(x'))}, \tag{1}$$

where $\hat{p}(y = c|x')$ is an estimation of the probability $p(y = c|x')$.

Standard classifiers work well on classifying classes with enough training examples. However, we often encounter few-shot classification tasks that we need to classify points from unseen classes with only a few available examples per class. In such cases, standard classifiers may not perform well (Vinyals et al., 2016). Moreover, standard classifiers do not model similarity between different data points on the logit layer. Metric learning methods learn pseudo metrics such that points with the same classes are close, and points with different classes are far apart on the learned embedding spaces. As a result, metric learning models can work well on classifying classes with a few examples (Chen et al., 2019), and they can be used to find similar data points for each query point (Musgrave et al., 2020).

Several metric learning models (Goldberger et al., 2004; Snell et al., 2017; Allen et al., 2019) use softmax-based formulation to model $p(y = c|x')$ by replacing logits $l_c(x')$ in Equation (1) with negative squared distances between data points on embedding spaces. We found that *1) softmax-based models can be affected by scaling embedding space* and thus possibly weaken the training process. Moreover, *2) they do not have the maximum (or minimum)*

*kim.hyeongji@hi.no

confidence scores[2] *on representing points of classes.* It implies that when the softmax-based formulation is used for metric learning, query points do not directly converge (approach) to points representing the same class on embedding space, and query points do not directly diverge (be far apart) from points representing different classes. As a result, metric learning with softmax-based formulation can be unstable.

To overcome these limitations, we propose an alternative formulation named *distance-ratio-based (DR) formulation* to estimate $p(y = c|x')$ in metric learning models. Unlike softmax-based formulation, *1) DR formulation is not affected by scaling embedding space.* Moreover, *2) it has the maximum confidence score* 1 *on the points representing the same class with query points and the minimum confidence score* 0 *on the points representing the different classes.* Hence, when we use DR formulation for metric learning, query points can directly approach to corresponding points and directly diverge from points that represent different classes. We analyzed the metric learning process with both formulations on few-shot learning tasks. Our experimental results show that using our formulation is less likely to be affected by scale changes and more stable. As a result, our formulation enables faster training (when Conv4 backbone was used) or comparable training speed (when ResNet18 backbone was used).

1.1 Problem Settings

Problem Setting. *Let $\mathcal{X} \subset \mathbb{R}^{d_I}$ be an input space, $\mathcal{Z} = \mathbb{R}^{d_F}$ be an unnormalized embedding space, and $\mathcal{Y}$ be a set of possible classes. The set $\mathcal{Y}$ also includes classes that are unseen during training. From a joint distribution $\mathcal{D}$, data points $x \in \mathcal{X}$ and corresponding classes $c \in \mathcal{Y}$ are sampled. An embedding function $f_\theta : \mathcal{X} \to \mathcal{Z}$ extracts features (embedding vectors) from inputs where θ represents learnable parameters. We consider the Euclidean distance $d(\cdot, \cdot)$ on the embedding space $\mathcal{Z}$.*

In this paper, we only cover unnormalized embedding space $\mathbb{R}^{d_F}$. One might be interested in using normalized embedding space $\mathbb{S}^{(d_F-1)} = \left\{z \in \mathbb{R}^{d_F} \middle| \|z\| = 1\right\}$. For normalized embedding space, one can still use Euclidean distance or angular distance (arc length) as both are proper distances.

To compare softmax-based and distance-ratio-based formulation that estimate $p(y = c|x')$ for metric learning, in this work, we use prototypical network (Snell et al., 2017) for explanation and experiments. We do this because the prototypical network is one of the simplest metric learning models.

1.2 Prototypical Network

Prototypical network (Snell et al., 2017) was devised to solve few-shot classification problems, which require to recognize unseen classes during training based on only a few labeled points (support points). It is learned by episode training (Vinyals et al., 2016) whose training batch (called an episode) consists of a set of support points and a set of query points. Support points act as guidelines that represent classes. Query points act as evaluations of a model to update the model (embedding function f_θ) in a training phase and to measure few-shot classification performances in a testing phase.

Using embedding vectors from support points, prototypical network calculates a *prototype* $\mathbf{p}_c$ to represent a class c. A prototype $\mathbf{p}_c$ is defined as:

$$\mathbf{p}_c = \frac{1}{|S_c|} \sum_{(x_i, y_i) \in S_c} f_\theta(x_i),$$

where S_c is a set of support points with class c. (When $|S_c|$ is fixed with $K = |S_c|$, a few-shot learning task is called a K-shot learning task.)

We can use the Euclidean distance with a prototype $\mathbf{p}_c$ on the embedding space to estimate how close a query point x' is to a class c. We denote the distance as $d_{x',c}$. Mathematically, $d_{x',c}$ is:

$$d_{x',c} = d(f_\theta(x'), \mathbf{p}_c) \tag{2}$$

Using this distance $d_{x',c}$, prototypical network estimates the probability $p(y = c|x')$. We will explain later about the softmax-based formulation and our formulation for this estimation. Based on the estimated probability, training loss L is defined as the average classification loss (cross-entropy) of query points. The loss L can be written as:

$$L = -\frac{1}{|Q|} \sum_{(x', y') \in Q} \log(\hat{p}(y = c|x')), \tag{3}$$

[2]Confidence score (value) is an estimated probability of $p(y = c|x')$ using a model.

where Q is a set of query points in an episode and $\hat{p}(y = c|x')$ is an estimation of the probability $p(y = c|x')$.

Based on the training loss L, we can update the embedding function f_θ.

1.3 Metric Learning with Softmax-Based Formulation

In the original prototypical network (Snell et al., 2017), the softmax-based formulation was used to model the probability $p(y = c|x')$. The softmax-based formulation is defined by the softmax (in Equation (1)) over negative squared distance $-d^2_{x',c}$. Thus, the formulation can be written as:

$$\hat{p}(y = c|x') = \frac{\exp(-d^2_{x',c})}{\sum_{y \in \mathcal{Y}_E} \exp(-d^2_{x',y})}, \tag{4}$$

where $\mathcal{Y}_E$ is a subset of $\mathcal{Y}$ that represents a set of possible classes within an episode. (When $|\mathcal{Y}_E|$ is fixed with $N = |\mathcal{Y}_E|$, a few-shot learning task is called a N-way learning task.)
We denote the value in Equation (4) as $\sigma_c(x')$. When we use $\sigma_c(x')$ to estimate the probability $p(y = c|x')$, we denote the corresponding loss (defined in Equation (3)) as L_S.

The softmax-based formulation can be obtained by estimating a class-conditional distribution $p(x'|y = c)$ with a Gaussian distribution. Based on this, in Appendix A, we explain why an average point is an appropriate point to represent a class when we use the softmax-based formulation.

1.3.1 Analysis of Softmax-Based Formulation

To analyze the formulation in Equation (4), let us consider a toy example. In this example, there are only two classes c_1 and c_2 and corresponding prototypes $\mathbf{p}_{c_1}$ and $\mathbf{p}_{c_2}$. Let us consider a query point x' that has distance d_{x',c_1} and d_{x',c_2} as in two cases in Table 1. When we compare case (a) and 2 times scaled case (case (b)), we can check that the loss L_s is much smaller for the scaled case (6.1442×10^{-6}). In other words, *simply scaling an embedding can change the confidence score and thus corresponding training loss*. It implies that embedding can be scaled to reduce training loss. Thus, using softmax-based models may weaken a training process by allowing unnecessary model updates that do not change relative locations of data points.

To inspect the locations that maximize or minimize confidence scores, in Figure 1, we visualized the estimated probability $\hat{p}(y = red|x')$ using three prototypes $\mathbf{p}_{red}$, $\mathbf{p}_{green}$, and $\mathbf{p}_{blue}$. For the softmax-based model in Figure 1a (Goldberger et al., 2004; Snell et al., 2017; Allen et al., 2019), *the maximum confidence value is not even at the prototype* $\mathbf{p}_{red}$. It implies that when we train an embedding with training loss L_S, query points with the red class do not converge directly to the prototype $\mathbf{p}_{red}$. In Figure 1a, the prototypes with different classes $\mathbf{p}_{green}$ and $\mathbf{p}_{blue}$ are not the points that minimize confidence values. It means that query points with red class do not directly diverge (get far apart) from prototypes $\mathbf{p}_{green}$ and $\mathbf{p}_{blue}$. As prototypes do not provide direct guidelines for query points, metric learning with softmax-based formulation can be unstable.

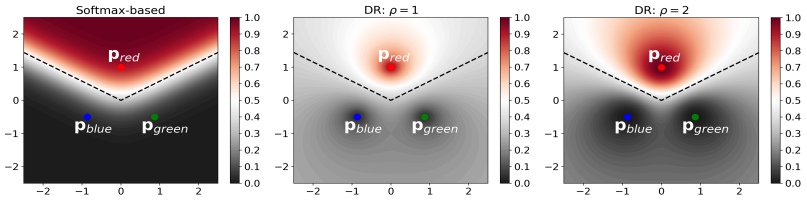
Table 1: A toy example with different d_{x',c_1} and d_{x',c_2}, and the corresponding values. We assumed the query point x' has class c_1 to calculate the losses. In this table, we set $\rho = 2$ for DR formulation. δ is defined in Section 2.

Cases	d_{x',c_1}	d_{x',c_2}	$\sigma_{c_1}(x')$	$\delta_{c_1}(x')$	L_s	L_{DR}
Case (a)	1	2	0.95257	0.80000	0.048587	0.22314
Case (b)	2	4	0.99999	0.80000	6.1442×10^{-6}	0.22314

2 Metric Learning with Distance-Ratio-based Formulation

To handle the limitations of softmax-based formulation in metric learning, we propose an alternative form called *distance-ratio-based (DR) formulation* for estimating probability $p(y = c|x')$. When we use distance $d_{x',c}$ as in Equation (2), DR formulation is defined as:

$$\hat{p}(y = c|x') = \frac{\frac{1}{d^\rho_{x',c}}}{\sum_{y \in \mathcal{Y}_E} \frac{1}{d^\rho_{x',y}}} = \frac{{d_{x',c}}^{-\rho}}{\sum_{y \in \mathcal{Y}_E} {d_{x',y}}^{-\rho}}, \tag{5}$$



(a) $\sigma_{red}(x')$ (b) $\delta_{red}(x')$ with $\rho = 1$ (c) $\delta_{red}(x')$ with $\rho = 2$

Figure 1: Visualization of estimations of $p(y = red|x')$ based on softmax-based formulation and distance-ratio-based formulation.

where $\rho > 0$ is a learnable parameter. When $d_{x',c} = 0$ and $d_{x',c'} > 0$ for all $c' \neq c$, we define $\hat{p}(y = c|x')$ as 1 and $\hat{p}(y = c'|x')$ as 0. As this formulation uses ratios of distances for classification, we call it as *distance-ratio-based formulation*. (One can check that Equation (5) can be obtained by replacing the negative squared distance $-d^2_{x',c}$ in Equation (4) with $-\rho \ln(d_{x',c})$.)

Let us denote the value in Equation (5) as $\delta_c(x')$. Then, when we use DR formulation to estimate the probability $p(y = c|x')$, we denote the corresponding training loss (defined in Equation (3)) as L_{DR}. Based on the training loss L_{DR}, we can update the embedding function f_θ and also the learnable parameter ρ.

2.1 Analysis of Distance-Ratio-Based Formulation

To analyze our formulation, let us consider when we change the scale of an embedding space. When we scale embedding with a scale parameter $\alpha > 0$, then the corresponding estimation of probability $p(y = c|x')$ with DR formulation is:

$$\begin{aligned}\hat{p}(y = c|x') &= \frac{\frac{1}{\left(\alpha d_{x',c}\right)^\rho}}{\sum_{y \in \mathcal{Y}_E} \frac{1}{\left(\alpha d_{x',y}\right)^\rho}} = \frac{\frac{1}{\alpha^\rho d^\rho_{x',c}}}{\sum_{y \in \mathcal{Y}_E} \frac{1}{\alpha^\rho d^\rho_{x',y}}} \\ &= \frac{\frac{1}{\alpha^\rho}\frac{1}{d^\rho_{x',c}}}{\frac{1}{\alpha^\rho}\sum_{y \in \mathcal{Y}_E} \frac{1}{d^\rho_{x',y}}} = \frac{\frac{1}{d^\rho_{x',c}}}{\sum_{y \in \mathcal{Y}_E} \frac{1}{d^\rho_{x',y}}}\end{aligned} \tag{6}$$

Equation (6) shows that when we use our formulation, *scaling an embedding has no effect on the confidence scores and thus the training loss*. (This property can also be checked from the cases (a) and (b) in Table 1.)

In addition to the scale invariance property, $\delta_c(x')$ has an additional property that has *optimal confidence scores on prototypes*. In detail, if we assume $d(\mathbf{p}_c, \mathbf{p}_{c'}) > 0$ for $\forall c' \in \mathcal{Y}_E$ with $c' \neq c$, then the following two equations hold:

$$\lim_{x' \to \mathbf{p}_c} \delta_c(x') = 1 \tag{7}$$

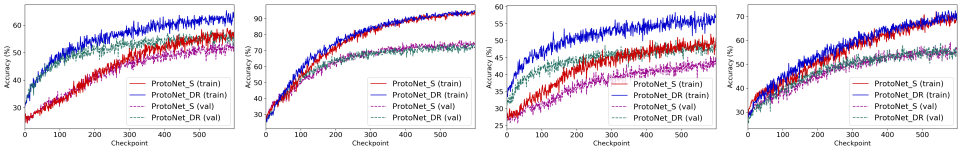
$$\lim_{x' \to \mathbf{p}_{c'}} \delta_c(x') = 0 \tag{8}$$

This property can be checked from Figures 1b and 1c that visualize the estimated probability $\hat{p}(y = red|x')$ using DR formulation. Proof of the property is in Appendix B. As prototypes provide optimal guidelines for query points, when we used DR formulation, query points can easily get close to the prototypes with their corresponding classes ($\mathbf{p}_c$) and get far away from the prototypes with different classes ($\mathbf{p}_{c'}$). Hence, metric learning with DR formulation can be stable.

3 Experiments

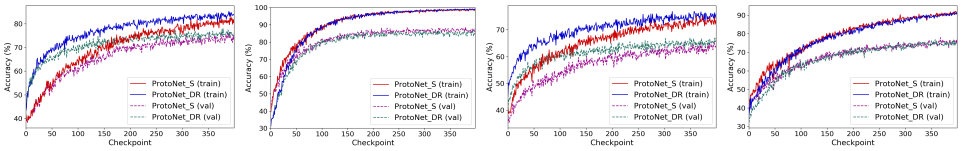
3.1 Experiment Settings

In our experiments, we wanted to investigate the effectiveness of the distance-ratio-based (DR) formulation compared to the softmax-based formulation. For that, we trained prototypical networks (Snell et al., 2017) based on two formulations for each experiment: softmax-based (ProtoNet_S) and DR formulation (ProtoNet_DR).



(a) Data: CUB, Backbone: Conv4 (b) Data: CUB, Backbone: ResNet18 (c) Data: *mini*-ImageNet, Backbone: Conv4 (d) Data: *mini*-ImageNet, Backbone: ResNet18

Figure 2: Training and validation accuracy curves for two different backbones on 1-shot learning tasks.



(a) Data: CUB, Backbone: Conv4 (b) Data: CUB, Backbone: ResNet18 (c) Data: *mini*-ImageNet, Backbone: Conv4 (d) Data: *mini*-ImageNet, Backbone: ResNet18

Figure 3: Training and validation accuracy curves for two different backbones on 5-shot learning tasks.

We implement 1-shot and 5-shot learning tasks with five classes for each episode (5-way). Details of the settings are described in the following paragraphs. Codes for our experiments are available in `https://github.com/hjk92g/DR_Formulation_ML`.

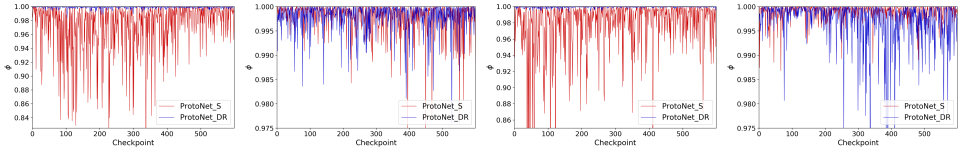
Dataset We conduct experiments using two common benchmarks for few-shot classification tasks: CUB (200-2011) (Wah et al., 2011) and *mini*-ImageNet dataset (Vinyals et al., 2016). The CUB dataset has 200 classes and 11,788 images. We use the same 100 training, 50 validation, and 50 test classes split as Chen et al. (2019). The *mini*-ImageNet dataset is a subset of the ImageNet dataset (Deng et al., 2009) suggested by Vinyals et al. (2016). It has 100 classes and 600 images per class. We use the same 64 training, 16 validation, and 20 test classes split as Ravi & Larochelle (2017); Chen et al. (2019). For both datasets, we apply data augmentation for training data. Applied data augmentation includes random crop, left-right flip, and color jitter.

Backbone (Architecture) We use two different backbones as embedding functions f_θ for each experiment: Conv4 (Snell et al., 2017) and ResNet18 (He et al., 2016). The Conv4 consists of four convolutional blocks. Each block is composed of a 64-filter 3×3 convolution, batch normalization, a ReLU activation function, and a 2×2 max-pooling layer. It takes 84×84 sized color images and outputs 1600 dimensional embedding vectors. The ResNet18 backbone is the same as in He et al. (2016). It contains convolutions, batch normalizations, ReLU activation functions like the Conv4, but it also has skip connections. It takes 224×224 sized color images and outputs 512 dimensional embedding vectors.

Optimization Backbones are trained from random weights. We use Adam optimizer (Kingma & Ba, 2014) with a learning rate 10^{-3}. To investigate training steps, we save training information and validation accuracy for each 100 training episodes, and we call each of these steps a checkpoint. For 1-shot classification tasks, we train embedding for 60,000 episodes (600 checkpoints). For 5-shot classification tasks, we train embedding for 50,000 episodes (500 checkpoints). Based on validation accuracies on each checkpoint, we select the best model among the checkpoints.

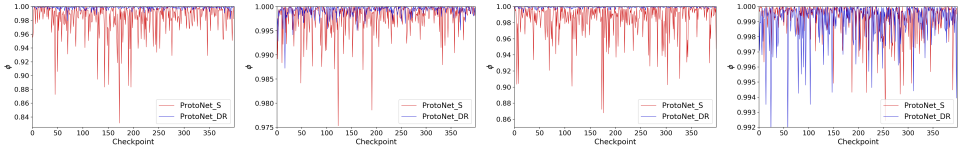
To implement our DR formulation, we modify the implementation of the standard softmax-based prototypical network (Snell et al., 2017) by replacing negative squared distance $-d^2_{x',c}$ in Equation (4) by $-\rho \ln(d_{x',c})$. For numerical stability, we add a small positive value 10^{-10} before taking square root in the calculation of Euclidean distance $d_{x';c}$. For the DR formulation, we use $\ln(\rho) \in \mathbb{R}$ to model $\rho = \exp(\ln(\rho))$. We set the initial parameter for $\ln(\rho)$ as 2.0. Based on this initial value, $\log(\rho)$ value is trained for all experiments.

To analyze the local training steps, for each checkpoint (every 100 episodes of training), we checked the positions of episode points (both support and query points) on embedding space just before the weight updates and right after the weight updates. When we consider positions on embedding space, we denote a matrix that represents the original



(a) Data: CUB, Backbone: Conv4 (b) Data: CUB, Backbone: ResNet18 (c) Data: *mini*-ImageNet, Backbone: Conv4 (d) Data: *mini*-ImageNet, Backbone: ResNet18

Figure 4: Norm ratio ϕ curves for two different datasets and backbones on 1-shot learning tasks. Note that the ranges of the y-axis are smaller in (b) and (d) than (a) and (c).



(a) Data: CUB, Backbone: Conv4 (b) Data: CUB, Backbone: ResNet18 (c) Data: *mini*-ImageNet, Backbone: Conv4 (d) Data: *mini*-ImageNet, Backbone: ResNet18

Figure 5: Norm ratio ϕ curves for two different datasets and backbones on 5-shot learning tasks. Note that the ranges of the y-axis are smaller in (b) and (d) than (a) and (c).

positions of episode points as X_{origin} and the corresponding matrix with updated weights as X_{new}. We assume these matrices are mean-centered. When the matrices are not mean-centered, we center the matrices so that an average point is located at zero. Then, we model the matrix X_{new} as a modification of $\alpha^* X_{origin}$ for an unknown scale parameter α^*. Based on this model on X_{new}, we calculated a score called *norm ratio* ϕ that measures the relative effect of scaling ($0 \leq \phi \leq 1$). It is defined as:

$$\phi = \frac{\left\| X_{new} - \hat{\alpha^*} X_{origin} \right\|_F}{\left\| X_{new} - X_{origin} \right\|_F}, \tag{9}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\hat{\alpha^*}$ is an estimated scaling parameter by minimizing the numerator of Equation (9). Norm ratio ϕ is close to 0 when the major changes are due to scaling. Norm ratio ϕ is close to 1 when a magnitude of an embedding is not changed. A detailed explanation for the norm ratio is in Appendix C.1.

In addition to norm ratio ϕ, to analyze the location changes of query points relative to the positions of prototypes, for each checkpoint, we also calculated other proposed measures called *con-alpha ratio* $\frac{\psi_{con}}{\hat{\alpha^*}}$, *div-alpha ratio* $\frac{\psi_{div}}{\hat{\alpha^*}}$, and *con-div ratio* $\frac{\psi_{con}}{\psi_{div}}$. We use the same estimated scale parameter $\hat{\alpha^*}$ as defined in the previous paragraph and Appendix C.1. Value ψ_{con} measures a degree of convergence of query points to the prototypes with the same class. Con-alpha ratio $\frac{\psi_{con}}{\hat{\alpha^*}}$ is the corresponding value after adjusting scale changes. It is smaller than 1 when query points get close to the prototypes with the same classes after adjusting the scale changes. Value ψ_{div} measures a degree of divergence (separation) of query points to the prototypes with different classes. Div-alpha ratio $\frac{\psi_{div}}{\hat{\alpha^*}}$ is the corresponding value after adjusting the scale changes. It is larger than 1 when query points get far apart from the prototypes with different classes after adjusting the scaling. Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$ is defined as con-alpha ratio $\frac{\psi_{con}}{\hat{\alpha^*}}$ divided by div-alpha ratio $\frac{\psi_{div}}{\hat{\alpha^*}}$. It measures a relative degree of intended convergence of query points compared to divergence to the prototypes with different classes. Detailed explanations for these measures are in Appendix C.2.

3.2 Experiment Results

Using the Conv4 backbone (Figures 2a, 2c, 3a, and 3c), we can observe that utilization of the DR formulation helps to train faster than using the softmax-based prototypical networks (Snell et al., 2017). When using the ResNet18 backbone, the differences are smaller in 1-shot learning tasks (Figures 2b and 2d) or reversed in 5-shot tasks (Figures 3b and 3d).

Table 2: Few-shot classification accuracies (%) for CUB and *mini*-ImageNet datasets. Each cell reports mean accuracy based on 600 random test episodes and the corresponding 95% confidence interval from a single trained model.

Backbone	Method	CUB		*mini*-ImageNet	
		1-shot	5-shot	1-shot	5-shot
Conv4	ProtoNet_S	50.46 ± 0.88	76.39 ± 0.64	44.42 ± 0.84	64.24 ± 0.72
	ProtoNet_DR	57.13 ± 0.95	76.50 ± 0.66	48.71 ± 0.78	65.90 ± 0.69
ResNet18	ProtoNet_S	72.99 ± 0.88	86.64 ± 0.51	54.16 ± 0.82	73.68 ± 0.65
	ProtoNet_DR	73.33 ± 0.90	86.63 ± 0.49	54.86 ± 0.86	72.93 ± 0.64

Figures 4 and 5 visualize the changes of norm ratio ϕ. Tables 3 to 10 (in Appendix C.3) report geometric means and statistical test results on norm ratio values. In all the experiments with the Conv4 backbone, norm ratios ϕ were significantly smaller when we used the softmax-based formulation than the DR formulation. In other words, there were more scale changes in embedding when we used the softmax-based formulation. It indicates that when we use the Conv4 backbone, using softmax-based formulation can be more prone to weaken the training process due to the unnecessary scale changes. When using the ResNet18 backbone, norm ratio values were very close to 1 (geometric mean at least 0.99705) on both formulations.

Tables 3 to 10 (in Appendix C.3) also report geometric means, proportions of properly learned cases ($\frac{\psi_{con}}{\alpha^*} < 1$, $\frac{\psi_{div}}{\alpha^*} > 1$, $\frac{\psi_{con}}{\psi_{div}} < 1$), and statistical test results on con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$, div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$, and con-div ratio $\frac{\psi_{con}}{\psi_{div}}$. When we consider con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$ for 1-shot learning tasks, the properly converged cases ($\frac{\psi_{con}}{\alpha^*} < 1$) were significantly more frequent when we used DR formulation. It means that in 1-shot training, our DR formulation model is more stable in decreasing distances between query points and prototypes with the corresponding classes. When we consider div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$, for all the experiments, the properly diverged cases ($\frac{\psi_{div}}{\alpha^*} > 1$) were significantly more frequent when we used the DR formulation. It indicates that the DR formulation-based model is more stable in increasing the distance between query points and prototypes with different classes.

Table 2 reports few-shot classification accuracies on test episodes. First, we consider the results when the Conv4 backbone was used for training. Except for the 5-shot classification task on the CUB dataset, which resulted in comparable accuracies (difference was 0.11%), the test accuracies were higher with the DR formulation. The accuracy differences ranged from 1.66% to 6.67%.

Now, we consider the results with the ResNet18 backbone in Table 2. First, the accuracy gaps were reduced. Chen et al. (2019) also observed this phenomenon when they compared accuracy gaps with different backbones using several few-shot learning models. While the differences were small in the 1-shot classification task (differences were 0.34% or 0.70%), using DR formulation achieved higher accuracies. For the 5-shot classification task on the *mini*-ImageNet dataset, using DR formulation achieved 0.75% lower accuracy.

4 Discussion

In this work, we address the limitations of softmax-based formulation for metric learning by proposing a distance-ratio-based (DR) formulation. DR formulation focuses on updating relative positions on embedding by ignoring the scale of an embedding space. It also enables stable training by using each representing point as an optimal position. Our experiments show that using DR formulation resulted in faster training speed in general and improved or comparable generalization performances.

When distance $d_{x',c}$ is a distance between a query point x' and the nearest support point with class c, distance ratio $\frac{d_{x',c_1}}{d_{x',c_2}}$ for two different classes c_1 and c_2 has been utilized in previous literature. By setting c_1 as the nearest class and c_2 as the second nearest class from a query point x', Júnior et al. (2017) have used the distance ratio named *nearest neighbor distance ratio (NNDR)* for handling open-set classification problems (Geng et al., 2020). Independently, Jiang et al. (2018) have utilized the inverse value $\frac{d_{x',c_2}}{d_{x',c_1}}$ to define *trust score*, which is an alternative value for confidence value of the standard softmax classifiers. Unlike these works that directly use distance-ratios without modeling confidence values, distance-ratio-based (DR) formulation models probability $p(y = c|x')$ using distance ratios.

To output confidence scores that are either 0 or 1 on some areas, sparsemax (Martins & Astudillo, 2016), sparsegenlin, and sparsehourglass (Laha et al., 2018) were proposed as alternatives for softmax formulation in non-metric learning cases. Unlike DR formulation, which has a confidence score 0 or 1 only on (countable) *representing points*, these

formulations have *areas* (sets of uncountable points) that output confidence score 0 or 1. Such property is inappropriate for metric learning as confidence scores can be 1 even for non-representing points, and thus query points do not need to converge very close to the corresponding representing points.

Recent works (Wang et al., 2017; Liu et al., 2017; Wang et al., 2018a,b; Deng et al., 2019) proposed to use modifications of the standard softmax classifier for metric learning. These modified softmax classifiers showed competitive performances on metric learning (Musgrave et al., 2020). Unlike traditional metric learning models that use data points or prototypes, which are obtained from data points, to represent classes, they use learnable parameter vectors to represent classes. They use cosine similarity on normalized embedding space $\mathbb{S}^{(d_F-1)}$. That is equivalent to using the softmax-based formulation with Euclidean distance. While scale dependency of the softmax-based formulation can be addressed due to normalization, the softmax-based formulation still lacks the second property of DR formulation. Thus, a representing parameter vector may not be a vector that maximizes the confidence value. To handle this issue, DR formulation can also be used as an alternative by using Euclidean or angular distance on normalized embedding space (see example in Appendix E).

In addition to the supervised metric learning, cosine similarity on a normalized representation space is also used in contrastive self-supervised learning (Chen et al., 2020) and in recent data augmentation strategy (Khosla et al., 2020) which uses supervised contrastive loss. DR formulation can also be applied to these models for possible performance improvements.

While using DR formulation resulted in faster or comparable training speed in most experiments, in Figure 3b and 3d, we observe slightly slower training speed in 5-shot learning with the ResNet18 backbone. We speculate the reason is that an average point is not an optimal point to represent a class in DR formulation (explained in Appendix A.2), unlike softmax-based formulation (explained in Appendix A.1). To investigate this, in Appendix D, we conduct 5-shot learning experiments with nearest neighbor-based models instead of using an average point to represent a class.

Our experiment results showed that the scale changes of softmax-based prototypical networks are decreased dramatically when the ResNet18 was used as a backbone. One possible reason for this phenomenon can be the skip connections in residual modules (He et al., 2016) and the fact that the scale of the input layer is fixed. It requires further investigation to draw a conclusion.

References

Kelsey Allen, Evan Shelhamer, Hanul Shin, and Joshua Tenenbaum. Infinite mixture prototypes for few-shot learning. In *International Conference on Machine Learning*, pp. 232–241. PMLR, 2019.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4690–4699, 2019.

Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17:513–520, 2004.

John C Gower, Garmt B Dijksterhuis, et al. *Procrustes problems*, volume 30. Oxford University Press on Demand, 2004.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Heinrich Jiang, Been Kim, Melody Y Guan, and Maya R Gupta. To trust or not to trust a classifier. In *NeurIPS*, pp. 5546–5557, 2018.

Pedro R Mendes Júnior, Roberto M De Souza, Rafael de O Werneck, Bernardo V Stein, Daniel V Pazinato, Waldir R de Almeida, Otávio AB Penatti, Ricardo da S Torres, and Anderson Rocha. Nearest neighbors distance ratio open-set classifier. *Machine Learning*, 106(3):359–386, 2017.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Anirban Laha, Saneem A Chemmengath, Priyanka Agrawal, Mitesh M Khapra, Karthik Sankaranarayanan, and Harish G Ramaswamy. On controllable sparse alternatives to softmax. *arXiv preprint arXiv:1810.11975*, 2018.

Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 212–220, 2017.

Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pp. 50–60, 1947.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pp. 1614–1623. PMLR, 2016.

Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision*, pp. 681–699. Springer, 2020.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017.

Nicu Sebe, Michael S Lew, Ira Cohen, Ashutosh Garg, and Thomas S Huang. Emotion recognition using a cauchy naive bayes classifier. In *Object recognition supported by user interaction for service robots*, volume 1, pp. 17–20. IEEE, 2002.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems*, 30:4077–4087, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. *California Institute of Technology*, 2011.

Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*, pp. 1041–1049, 2017.

Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018a.

Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5265–5274, 2018b.

APPENDIX

A Using an Average Point to Represent a Class

Here, we consider what is an appropriate point to represent a class c based on given support points. We denote a set of support points with class c as S_c.

A.1 Softmax-Based Formulation

The softmax-based formulation can be considered as an estimation of probability $p(y = c|x')$ when Gaussian distribution is used to estimate class-conditional distribution $p(x'|y = c)$. Mathematically, we consider an estimation $\hat{p}(x'|y = c)$ as:

$$\hat{p}(x'|y=c) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{d(f_\theta(x'), \mathbf{r}_c))}{\sigma}\right)^2\right),$$

where σ is a standard deviation of the distribution and $\mathbf{r}_c$ is a representing point of class c. When we fix $\sigma = \frac{1}{\sqrt{2}}$, we get a simpler equation:

$$\hat{p}(x'|y=c) = \frac{1}{\sqrt{\pi}} \exp\left(-d(f_\theta(x'), \mathbf{r}_c))^2\right)$$

Based on this equation and support points, we use MLE (maximum likelihood estimation) approach to find a good estimation for representing point $\mathbf{r}_c$. To maximize the likelihood, let us denote the corresponding likelihood function as $L(\mathbf{r}_c; S_c)$. It satisfies equations:

$$\begin{aligned} L(\mathbf{r}_c; S_c) &= \prod_{(x_i,y_i)\in S_c} \hat{p}(x_i|y=y_i) \\ &= \frac{1}{\sqrt{\pi}^{|S_c|}} \exp\left(-\sum_{(x_i,y_i)\in S_c} d(f_\theta(x_i), \mathbf{r}_c))^2\right) \end{aligned}$$

As the sum $\sum_{(x_i,y_i)\in S_c} d(f_\theta(x_i), \mathbf{r}_c))^2$ is minimized by the average points of support points on embedding space, $\mathbf{r}_c = \mathbf{p}_c = \frac{1}{|S_c|} \sum_{(x_i,y_i)\in S_c} f_\theta(x_i)$ becomes MLE solution.

A.2 Distance-Ratio-Based (DR) Formulation

For special case $\rho = 2$, DR formulation can be considered as a limit distribution of $p(y = c|x')$ when Cauchy distribution is used to estimate class-conditional distribution $p(x'|y = c)$. Mathematically, we consider equation (Sebe et al., 2002):

$$\hat{p}(x'|y=c) = \frac{\gamma}{\pi\left(\gamma^2 + d(f_\theta(x'), \mathbf{r}_c)^2\right)},$$

where γ is a parameter of the distribution and $\mathbf{r}_c$ is a representing point of class c. Then, when γ approaches 0 with uniform probability for $\hat{p}(y = c)$, the limit distribution of $\hat{p}(y = c|x')$ (defined using Bayes theorem) becomes the DR formulation.

When we denote the corresponding likelihood function as $L(\mathbf{r}_c; S_c)$, it satisfies the equation:

$$L(\mathbf{r}_c; S_c) = \left(\frac{\gamma}{\pi}\right)^{|S_c|} \frac{1}{\prod_{(x_i,y_i)\in S_c} (\gamma^2 + d(f_\theta(x_i), \mathbf{r}_c)^2)}$$

For MLE, when γ is very close to 0, we need to minimize product defined as:

$$\prod_{(x_i,y_i)\in S_c} d(f_\theta(x_i), \mathbf{r}_c)^2 = \left(\prod_{(x_i,y_i)\in S_c} d(f_\theta(x_i), \mathbf{r}_c)\right)^2$$

The points that minimize the product are $\mathbf{r}_c = f_\theta(x_i)$ for $(x_i, y_i) \in S_c$. It means that when we consider K-shot classification with $K > 1$, using an average point would not be an optimal point to represent a class.

B Proof of Equations (7) and (8)

Property When $d(\mathbf{p}_c, \mathbf{p}_{c'}) > 0$ for $\forall c' \in \mathcal{Y}_E$ with $c' \neq c$, then the following two equations hold:

$$\lim_{x'\to\mathbf{p}_c} \delta_c(x') = 1 \tag{7}$$

$$\lim_{x'\to\mathbf{p}_{c'}} \delta_c(x') = 0 \tag{8}$$

Proof. First, let us assume $d(\mathbf{p}_c, \mathbf{p}_{c'}) > 0$ for $c' \neq c$.

$d_{x',c'} = d(f_\theta(x'), \mathbf{p}_{c'}) \geq |d(f_\theta(x'), \mathbf{p}_c) - d(\mathbf{p}_c, \mathbf{p}_{c'})| = |d_{x',c} - d(\mathbf{p}_c, \mathbf{p}_{c'})|$ ($\because$ The reverse triangle inequality from the triangle inequality).

$\lim_{x'\to\mathbf{p}_c} d_{x',c'} \geq \lim_{x'\to\mathbf{p}_c} |d_{x',c} - d(\mathbf{p}_c, \mathbf{p}_{c'})| = |\left(\lim_{x'\to\mathbf{p}_c} d_{x',c}\right) - d(\mathbf{p}_c, \mathbf{p}_{c'})| = |0 - d(\mathbf{p}_c, \mathbf{p}_{c'})| = d(\mathbf{p}_c, \mathbf{p}_{c'}) > 0.$

As $\lim_{x'\to\mathbf{p}_c} d_{x',c'} > 0$, we get $\lim_{x'\to\mathbf{p}_c} \frac{1}{d^\rho_{x',c'}} < \infty$.

Thus, we also get $\lim_{x'\to\mathbf{p}_c} \left(\sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}}\right) < \infty$. Let us denote the limit value as C_1.

$$\begin{aligned}
\lim_{x'\to\mathbf{p}_c} \delta_c(x') &= \lim_{x'\to\mathbf{p}_c} \frac{\frac{1}{d^\rho_{x',c}}}{\sum_{y\in\mathcal{Y}_E} \frac{1}{d^\rho_{x',y}}} = \lim_{x'\to\mathbf{p}_c} \frac{1}{1 + d^\rho_{x',c} \sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}}} = \frac{1}{1 + \lim_{x'\to\mathbf{p}_c} \left(d^\rho_{x',c} \sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}}\right)} \\
&= \frac{1}{1 + \lim_{x'\to\mathbf{p}_c} d^\rho_{x',c} \lim_{x'\to\mathbf{p}_c} \left(\sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}}\right)} \\
&= \frac{1}{1 + 0 \times C_1} = 1
\end{aligned} \tag{10}$$

We proved Equation (7).

With a similar process as above, we get $\lim_{x'\to\mathbf{p}_{c'}} d_{x',c} > 0$ and $\lim_{x'\to\mathbf{p}_{c'}} d^\rho_{x',c} > 0$. Let us denote the later limit value as C_2.

$\lim_{x'\to\mathbf{p}_{c'}} \frac{1}{d^\rho_{x',c'}} = \lim_{x'\to\mathbf{p}_{c'}} \left(\frac{1}{d_{x',c'}}\right)^\rho = \infty$, and thus $\lim_{x'\to\mathbf{p}_{c'}} \sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}} = \infty$.

With a similar process as in Equation (10), we get an equation:

$$\lim_{x'\to\mathbf{p}_{c'}} \delta_c(x') = \frac{1}{1 + \lim_{x'\to\mathbf{p}_{c'}} d^\rho_{x',c} \lim_{x'\to\mathbf{p}_{c'}} \left(\sum_{y\in\mathcal{Y}_E, y\neq c} \frac{1}{d^\rho_{x',y}}\right)} = \frac{1}{1 + C_2 \times \infty} = 0$$

We proved Equation (8). □

C Proposed Ratios to Analyze Training Process and Their Results

C.1 Norm Ratio ϕ

To check the effect of scale changes in metric learning, we introduce norm ratio ϕ to measure irrelevance to scale change. It is based on the positions of episode points (both support and query points) on embedding space. Let us denote the original positions of embedding outputs as matrix X_{origin} and updated positions as matrix X_{new}. When these matrices are not mean- centered, we center the matrices so that an average point is located at zero. Then, we consider X_{new} as a modification of scaled original data matrix $\alpha^* X_{origin}$ for an unknown scaling parameter α^*. Mathematically, we consider the decomposition:

$$X_{new} = \alpha^* X_{origin} + (X_{new} - \alpha^* X_{origin})$$

As the scaling parameter α^* is unknown, we estimate α^* using Procrustes analysis (Gower et al., 2004). Procrustes analysis is used for superimposing two sets of points with optimal changes. We denote an estimated value for α^* as $\hat{\alpha^*}$ and define it as:

$$\hat{\alpha^*} = \arg\min_{\alpha \in \mathbb{R}} \|X_{new} - \alpha X_{origin}\|_F , \tag{11}$$

where $\|\cdot\|_F$ is the Frobenius norm.

Let us denote the Frobenius inner product as $\langle \cdot, \cdot \rangle_F$. Then, for $\|X_{origin}\|_F \neq 0$, we get the equations:

$$\begin{aligned}
\|X_{new} - \alpha X_{origin}\|_F^2 &= \langle X_{new} - \alpha X_{origin}, X_{new} - \alpha X_{origin} \rangle_F \\
&= \|X_{new}\|_F^2 + \alpha^2 \|X_{origin}\|_F^2 - 2\alpha \langle X_{origin}, X_{new} \rangle_F \\
&= \|X_{origin}\|_F^2 \left(\alpha - \frac{\langle X_{origin}, X_{new} \rangle_F}{\|X_{origin}\|_F^2} \right)^2 + \|X_{new}\|_F^2 - \frac{\langle X_{origin}, X_{new} \rangle_F^2}{\|X_{origin}\|_F^2}
\end{aligned}$$

Thus, we get $\hat{\alpha^*} = \frac{\langle X_{origin}, X_{new} \rangle_F}{\|X_{origin}\|_F^2} = \frac{\mathrm{Tr}\,(X_{origin}^T X_{new})}{\|X_{origin}\|_F^2}$ for $\|X_{origin}\|_F \neq 0$.

Once we get an estimated α^*, norm ratio ϕ is defined as:

$$\phi = \frac{\left\|X_{new} - \hat{\alpha^*} X_{origin}\right\|_F}{\|X_{new} - X_{origin}\|_F}$$

Norm ratio ϕ is at most one because of our definition of $\hat{\alpha^*}$ in Equation (11). Thus, $0 \leq \phi \leq 1$.

When major changes are due to scaling of an embedding space, $X_{new} \approx \hat{\alpha^*} X_{origin}$, and thus $\phi \approx 0$. On the other hand, when changes are irrelevant to scaling, $X_{new} - \hat{\alpha^*} X_{origin} \approx X_{new} - X_{origin}$, and thus $\phi \approx 1$.

C.2 Con-Alpha Ratio $\frac{\psi_{con}}{\hat{\alpha^*}}$, Div-Alpha Ratio $\frac{\psi_{div}}{\hat{\alpha^*}}$, and Con-Div Ratio $\frac{\psi_{con}}{\psi_{div}}$

Inspired by the rate of convergence in numerical analysis, we also introduce more measures to analyze the training processes. As we are using episode training with the prototypical network (Snell et al., 2017), we expect a query point x' with class c to get closer to the prototype $\mathbf{p}_c$ on the embedding space. Similarly, we expect a query point x' to get far apart from prototypes with different classes ($\mathbf{p}_{c'}$). To measure the degree (speed) of convergence or divergence with prototypes, we denote the original parameters of the embedding function as θ_{origin} and updated parameters as θ_{new}. Then, $f_{\theta_{origin}}$ represents the original embedding function, and $f_{\theta_{new}}$ represents the updated embedding function. We denote a query point with an index j as x'_j. We use the following value $\psi_{y,j}$ to check if a query point gets closer to or far away from a prototype $\mathbf{p}_y$. It is defined as:

$$\psi_{y,j} = \frac{d(f_{\theta_{new}}(x'_j), \mathbf{p}_y)}{d(f_{\theta_{origin}}(x'_j), \mathbf{p}_y)}$$

Case $\psi_{y,j} < 1$ means that the j th query point gets closer to the prototype $\mathbf{p}_y$ for $y \in \mathcal{Y}_E$. Case $\psi_{y,j} > 1$ means that the j th query point gets far apart from the prototype $\mathbf{p}_y$.

Based on $\psi_{y,j}$ values, we can estimate an average degree of convergence to the corresponding class (denoted as ψ_{con}) and an average speed of divergence from different classes (denoted as ψ_{div}). These values are calculated by taking geometric means of $\psi_{y,j}$ values. Mathematically, we define ψ_{con} and ψ_{div} as:

$$\psi_{con} = \left(\prod_{y=y'_j, y \in \mathcal{Y}_E} \psi_{y,j} \right)^{\frac{1}{|I_{\text{same}}|}},$$

$$\psi_{div} = \left(\prod_{y \neq y'_j, y \in \mathcal{Y}_E} \psi_{y,j} \right)^{\frac{1}{|I_{\text{diff}}|}},$$

where y'_j is the class of j th query point, I_{same} is the set of prototype-query index pairs with same classes, and I_{diff} is the set of prototype-query index pairs with different classes.

As these values can also be affected by scaling, we use normalized values using the estimated α^*. We call the corresponding values $\frac{\psi_{con}}{\alpha^*}$ and $\frac{\psi_{div}}{\alpha^*}$ as con-alpha ratio and div-alpha ratio, respectively. Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$ represents a degree of convergence of query points to the prototypes with the same class after adjusting scale changes. When query points get close to prototypes with the same classes (compared to the scale change), on average, con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$ will be smaller than 1. Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$ represents a degree of divergence of query points to the prototypes with different classes after adjusting scale changes. When query points get far apart from prototypes with the other classes (compared to the scale change), on average, the div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$ will be larger than 1. When we divide con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$ with div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$, we get another value $\frac{\psi_{con}}{\psi_{div}}$ called con-div ratio. It measures a relative degree of convergence of query points to prototypes with the same class compared to divergence to the prototypes with different classes. We expect the con-div ratio $\frac{\psi_{con}}{\psi_{div}}$ to be smaller than 1 for appropriate training.

C.3 Results of Proposed Measures

Table 3 to 10 show the analysis results of norm ratio ψ, con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$, div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$, and con-div ratio $\frac{\psi_{con}}{\psi_{div}}$ on few-shot classification tasks. Results are based on only one experiment for each formulation. For each experiment, statistical tests are applied using 600 training checkpoints for 1-shot learning or 400 training checkpoints for 5-shot learning. Mann–Whitney U test (Mann & Whitney, 1947) is applied to check if the obtained values using two formulations are significantly different. We report the proportions of properly learned counts (properly learned cases: con-alpha ratio $\frac{\psi_{con}}{\alpha^*} < 1$, div-alpha ratio $\frac{\psi_{div}}{\alpha^*} > 1$, and con-div ratio $\frac{\psi_{con}}{\psi_{div}} < 1$). Fisher's exact test is applied to check if the obtained counts using two formulations are significantly different. In tables, significant p-values (< 0.01) are written in bold text.

Table 3: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 1-shot task for CUB data with Conv4 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.97218		0.99898		$\mathbf{< 10^{-120}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99710	0.87000	0.99762	0.95667	**0.00244**	$\mathbf{< 10^{-7}}$
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00060	0.63667	1.00076	0.80833	0.61722	$\mathbf{< 10^{-10}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99650	0.96833	0.99687	0.99500	0.05251	**0.00077**

Table 4: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 1-shot task for CUB data with ResNet18 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.99778		0.99789		0.88640	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.98729	0.94833	0.98835	0.99333	0.34990	$< \mathbf{10^{-5}}$
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00257	0.72500	1.00247	0.86000	0.11755	$< \mathbf{10^{-8}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.98476	0.99500	0.98592	0.99667	0.02294	1.00000

Table 5: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 1-shot task for *mini*-ImageNet data with Conv4 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.97620		0.99940		$< \mathbf{10^{-126}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99776	0.85500	0.99805	0.95000	0.07212	$< \mathbf{10^{-7}}$
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00075	0.67500	1.00087	0.83333	0.07522	$< \mathbf{10^{-9}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99701	0.95333	0.99718	0.99000	0.41342	**0.00016**

Table 6: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 1-shot task for *mini*-ImageNet data with ResNet18 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.99889		0.99705		$< \mathbf{10^{-15}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99194	0.95000	0.99207	0.98333	0.11545	**0.00187**
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00177	0.67167	1.00179	0.88167	0.35608	$< \mathbf{10^{-17}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99019	0.99500	0.99030	0.99500	0.68248	1.00000

Table 7: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 5-shot task for CUB data with Conv4 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.98439		0.99863		$< \mathbf{10^{-61}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99885	0.81500	0.99906	0.87250	$< \mathbf{10^{-4}}$	0.03188
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00095	0.70500	1.00114	0.91500	0.02577	$< \mathbf{10^{-13}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99791	0.92000	0.99793	0.97000	0.29667	**0.00281**

Table 8: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 5-shot task for CUB data with ResNet18 backbone.

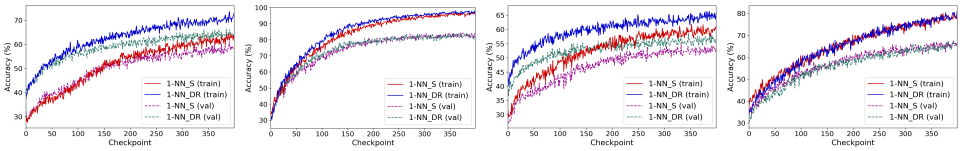
Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.99790		0.99928		$< \mathbf{10^{-14}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99435	0.96999	0.99482	0.98250	0.22264	0.088458
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00210	0.78000	1.00202	0.96750	0.05866	$< \mathbf{10^{-15}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99227	0.99250	0.99281	0.99500	0.11123	1.00000

Table 9: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 5-shot task for *mini*-ImageNet data with Conv4 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.98565		0.99930		$< \mathbf{10^{-69}}$	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99917	0.79500	0.99934	0.79500	0.04840	1.0
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00110	0.78000	1.00144	0.96000	**0.00054**	$< \mathbf{10^{-14}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99807	0.94750	0.99790	0.97250	0.40437	0.10306

Table 10: Analysis results of norm ratio, con-alpha ratio, div-alpha ratio, and con-div ratio on 5-shot task for *mini*-ImageNet data with ResNet18 backbone.

Measures	ProtoNet_S		ProtoNet_DR		Mann–Whitney U test	Fisher's exact test
	Geometric mean	Proportion	Geometric mean	Proportion		
Norm ratio ϕ	0.99925		0.99914		0.22914	
Con-alpha ratio $\frac{\psi_{con}}{\alpha^*}$	0.99465	0.96250	0.99554	0.97500	**0.00810**	0.41691
Div-alpha ratio $\frac{\psi_{div}}{\alpha^*}$	1.00163	0.72750	1.00195	0.95000	0.20071	$< \mathbf{10^{-17}}$
Con-div ratio $\frac{\psi_{con}}{\psi_{div}}$	0.99303	0.99000	0.99360	1.0000	0.01124	0.12406



(a) Data: CUB, Backbone: Conv4 (b) Data: CUB, Backbone: ResNet18 (c) Data: *mini*-ImageNet, Backbone: Conv4 (d) Data: *mini*-ImageNet, Backbone: ResNet18

Figure 6: Training and validation accuracy curves for two different backbones on 5-shot classification task (experiment with nearest neighbors based model).

D 5-shot Learning Experiment with Nearest-Neighbor based Model

5-shot learning with the ResNet18 backbone (in Figures 3b and 3d) showed a slightly slower training speed when using DR formulation than using softmax-based formulation. To investigate if the reason is that a mean point is not an optimal point to represent a class in formulation (explained in Appendix A.2), we try an additional experiment using nearest neighbors (1-NN) instead of the prototypical network (Snell et al., 2017). In detail, in stead of using a prototype to define a distance to a class as in Equation (2), we use a distance defined as:

$$d_{x';c} = \min_{(x_i,y_i)\in S_c} d(f_\theta(x'), f_\theta(x_i)),$$

where S_c is a set of support points with class c. This distance represents a distance to the nearest support point on embedding space. Thus, the corresponding model becomes a differentiable nearest neighbor (1-NN) classifier. Here, we train differentiable 1-NN classifiers for 5-shot classification based on two formulations for each experiment: softmax-based (1-NN_S) and distance-ratio-based (DR) formulation (1-NN_DR).

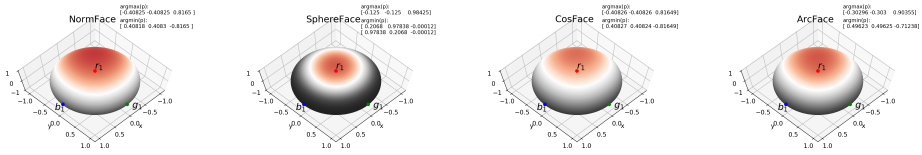
Figure 6 shows the training and validation accuracy curves when we use differentiable 1-NN classifiers. It shows that using DR formulation enables faster (Figure 6a, 6b, 6c) or almost comparable (slightly slower) (Figure 6d) training also for 5-shot tasks.

E An Example of a Normalized Embedding Space

To show the limitation of using cosine similarity in normalized embedding, we consider an example on a normalized embedding space $\mathbb{S}^2$. $r_1 = (0, 0, 1)$, $g_1 = (0, 1, 0)$, and $b_1 = (1, 0, 0)$ are parameter vectors that represent red, green, and blue classes, respectively. In this example, we consider estimated probabilities $\hat{p}(y = red|x')$ and the maximizing and minimizing positions using different models. (We used numerical optimization to get maximizing and minimizing positions of $\hat{p}(y = red|x')$.)

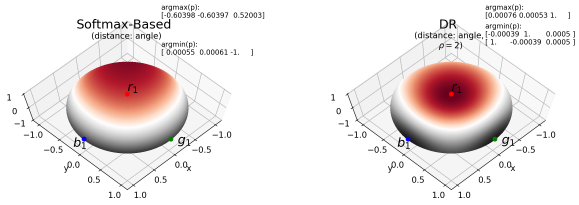
Figure 7 shows $\hat{p}(y = red|x')$ and the maximizing and minimizing positions for models that use cosine similarity (NormFace (Wang et al., 2017), SphereFace (Liu et al., 2017), CosFace (Wang et al., 2018b)=AM-softmax (Wang et al., 2018a), ArcFace (Deng et al., 2019)). All cosine similarity-based models take a different position than r_1 as the position that maximizes $\hat{p}(y = red|x')$. Also, these models take different positions than g_1 and b_1 as the positions that minimize $\hat{p}(y = red|x')$.

Figure 8 shows $\hat{p}(y = red|x')$ and the maximizing and minimizing positions for models that use angular distance. In both models, we use angular distances with parameter vectors. By using angles (angular distances), we use the equations (4) and (5) to calculate $\hat{p}(y = red|x')$. The angle-based softmax-based model takes a different position than r_1 as the position that maximizes $\hat{p}(y = red|x')$. It takes $-r_1$ as the position that minimizes $\hat{p}(y = red|x')$. Angle-based DR formulation-based model takes r_1 as the position that maximizes $\hat{p}(y = red|x')$. It takes g_1 and b_1 as the positions that minimize $\hat{p}(y = red|x')$. The example shows that using DR formulation helps data points to converge directly to the corresponding parameter vectors and data points can also diverge from parameter vectors for different classes.



(a) (b) (c) (d)

Figure 7: Visualization of $\hat{p}(y = red|x')$ for normalized embedding. Color is close to black when the value is small (close to zero). Color is close to dark red when the value is large (close to one). Models use cosine similarity. We used scaling parameter 2.0 for all models and margin 2 (with $k = 0$) in SphereFace (Liu et al., 2017). We used 0.25 as margins in CosFace (Wang et al., 2018b) and ArcFace (Deng et al., 2019).



(a) (b)

Figure 8: Visualization of $\hat{p}(y = red|x')$ for normalized embedding. The same visualization was used as Figure 7. Models use angular distance. We used $\rho = 2$ for DR formulation.

Article III

III Measuring Adversarial Robustness using a Voronoi-Epsilon Adversary

Hyeongji Kim, Pekka Parviainen, and Ketil Malde. *In Proceedings of the Northern Lights Deep Learning Workshop*, Volume 4, 2023.

Previous version was presented at ICLR 2021 Workshop on Security and Safety in Machine Learning Systems. The published version of the paper is available with DOI 10.7557/18.6827.

Measuring Adversarial Robustness using a Voronoi-Epsilon Adversary

Hyeongji Kim*[1,2], Pekka Parviainen[1], and Ketil Malde[1,2]

[1]Department of Informatics, University of Bergen, Norway
[2]Institute of Marine Research, Bergen, Norway

Abstract

Previous studies on robustness have argued that there is a tradeoff between accuracy and adversarial accuracy. The tradeoff can be inevitable even when we neglect generalization. We argue that the tradeoff is inherent to the commonly used definition of adversarial accuracy, which uses an adversary that can construct adversarial points constrained by ϵ-balls around data points. As ϵ gets large, the adversary may use real data points from other classes as adversarial examples. We propose a Voronoi-epsilon adversary which is constrained both by Voronoi cells and by ϵ-balls. This adversary balances two notions of perturbation. As a result, adversarial accuracy based on this adversary avoids a tradeoff between accuracy and adversarial accuracy on training data even when ϵ is large. Finally, we show that a nearest neighbor classifier is the maximally robust classifier against the proposed adversary on the training data.

1 Introduction

By applying a carefully crafted, but imperceptible perturbation to input images, so-called adversarial examples can be constructed that cause classifiers to misclassify the perturbed inputs [Szegedy et al., 2014]. Defense methods like adversarial training [Madry et al., 2018] and certified defenses [Wong and Kolter, 2018] against adversarial examples have often resulted in decreased accuracies on clean samples [Tsipras et al., 2019]. Previous studies have argued that the tradeoff between accuracy and adversarial accuracy may be inevitable in classifiers [Tsipras et al., 2019, Dohmatob, 2019, Zhang et al., 2019].

*Corresponding Author: hjk92g@gmail.com

1.1 Problem Settings

Problem setting. *Let $\mathcal{X} \subset \mathbb{R}^{\dim}$ be a nonempty input space and $\mathcal{Y}$ be a set of possible classes. Data points $x \in \mathcal{X}$ and corresponding classes $c_x \in \mathcal{Y}$ are sampled from a joint distribution $\mathcal{D}$. The distribution $\mathcal{D}$ should satisfy the condition that c_x is unique for all x. The set of the data points is a finite, nonempty set X. A classifier f assigns a class label from $\mathcal{Y}$ for each point $x \in \mathcal{X}$. $l(y_1, y_2)$ is a classification loss function for $y_1, y_2 \in \mathcal{Y}$ and it satisfies the necessary condition:*

$$\forall y_1, y_2, y_3, y_4 \in \mathcal{Y},$$
$$l(y_1, y_2) \leq l(y_3, y_4) \Longrightarrow \mathbb{1}(y_1 = y_2) \geq \mathbb{1}(y_3 = y_4).$$

$L(x, y)$ is a classification loss based on the classifier f provided an input $x \in \mathcal{X}$ and a label $y \in \mathcal{Y}$. Mathematically, $L(x, y) := l(f(x), y)$.

To simplify the analysis, we do not consider generalization.

1.2 Adversarial Accuracy (AA)

For a classifier, (natural) accuracy a is the expectation of a correct classification of data sampled from the data distribution. Mathematically, it is defined as:

$$a = \mathbb{E}_{(x,c_x)\sim\mathcal{D}} \left[\mathbb{1} \left(f(x) = c_x \right) \right].$$

Adversarial accuracy is a commonly used measure of adversarial robustness of classifiers [Madry et al., 2018, Tsipras et al., 2019]. It is defined by an adversary region $R(x) \subset \mathcal{X}$, which is an allowed region of the perturbations for a data point x.

https://doi.org/10.7557/18.6827

Definition 1 (**Adversarial accuracy**). *Given an adversary that is constrained to an adversary region $R(x)$, adversarial accuracy a is defined as:*

$$a = \mathbb{E}_{(x,c_x)\sim\mathcal{D}}\left[\mathbb{1}\left(f(x^*) = c_x\right)\right],$$

$$\text{where } x^* = \underset{x'\in R(x)}{\arg\max}\, L(x', c_x).$$

The choice of $R(x)$ will determine the adversarial accuracy that we are measuring. Commonly considered adversary region is $\mathbb{B}(x, \epsilon)$, which is a ϵ-ball around a data point x based on a distance metric d [Biggio et al., 2013, Madry et al., 2018, Tsipras et al., 2019, Zhang et al., 2019].

Definition 2 (**Standard adversarial accuracy**). *When the adversary region is $\mathbb{B}(x, \epsilon)$, we refer to the adversarial accuracy a as standard adversarial accuracy (SAA) $a_{std}(\epsilon)$. For SAA, we denote $R(x)$ as $R_{std}(\epsilon; x)$.*

$$a_{std}(\epsilon) = \mathbb{E}_{(x,c_x)\sim\mathcal{D}}\left[\mathbb{1}\left(f(x^*) = c_x\right)\right],$$

$$\text{where } x^* = \underset{x'\in R_{std}(\epsilon;x)}{\arg\max}\, L(x', c_x).$$

This adversary region $\mathbb{B}(x, \epsilon)$ is based on an implicit assumption that there might be an adequate single epsilon ϵ that perturbed samples do not change their classes. However, this assumption has some limitations. We explain that in the next section.

1.3 The Tradeoff Between Accuracy and Standard Adversarial Accuracy

The usage of ϵ-ball-based adversary can cause the tradeoff between accuracy and adversarial accuracy. When the two clean samples x_1 and x_2 with $d(x_1, x_2) \leq \epsilon$ have different classes, achieving local SAA higher than 0 on these two points implies misclassification. We illustrate this with a toy example.

1.3.1 Toy Example

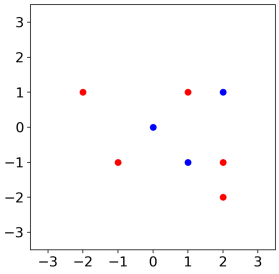
Let us consider an example visualized in Figure 1a. The input space is $\mathbb{R}^2$. There are only two classes A and B, i.e., $\mathcal{Y} = \{A, B\}$. We use the l_2 norm as a distance metric in this example.

Let us consider a situation when $\epsilon = 1.0$ (see Figure 1c). In this case, clean samples can also be considered as adversarial examples. For example, the point $(2, 1)$ can be considered as an adversarial example originating from the point $(1, 1)$. If one choose a robust model based on SAA, one might choose a model with excessive invariance. For example, one might choose a model that predicts points belong to $\mathbb{B}((1, 1), 1)$ (including the point $(2, 1)$) have class A. Or, one can choose a model that predicts points belong to $\mathbb{B}((2, 1), 1)$ (including the point $(1, 1)$) have class B. In either case, the accuracy of the chosen model is smaller than 1. This situation explains the tradeoff between accuracy and standard adversarial accuracy when large ϵ is used. It originates from the overlapping adversary regions from the samples with different classes.

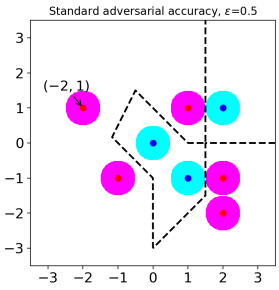
To avoid the tradeoff between accuracy and adversarial accuracy, one can use small ϵ values. Actually, a previous study has argued that commonly used ϵ values are small enough to avoid the tradeoff [Yang et al., 2020b]. However, when small ϵ values are used, we can only analyze local robustness, and we need to ignore robustness beyond the chosen ϵ. For instance, let us consider our example when $\epsilon = 0.5$ (see Figure 1b). In this case, we ignore robustness on $\mathbb{B}((-2, 1), 1.0) - \mathbb{B}((-2, 1), 0.5)$. Models with local but without global robustness enable attackers to use large ϵ values to fool the models. Ghiasi et al. [2019] have experimentally shown that even models with certified local robustness can be attacked by attacks with large ϵ values. Note that their attack applies little semantic perturbations even though the perturbation norms measured by l_p norms are large.

These limitations motivate us to find an alternative way to measure robustness. **The contributions of this paper are as follows.**

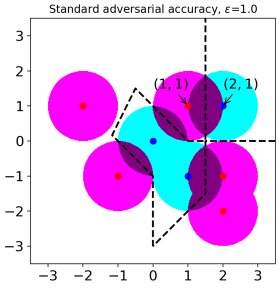
- We propose Voronoi-epsilon adversarial accuracy (VAA) that avoids the tradeoff between accuracy and adversarial accuracy. This allows the adversary regions to scale to cover most of the input space without incurring a tradeoff. To our best knowledge, this is the first work to achieve this without an external classifier.
- We explain the connection between SAA and VAA. We define global Voronoi-epsilon robustness as a limit of the Voronoi-epsilon adversarial accuracy. We show that a nearest neighbor (1-NN) classifier maximizes global Voronoi-epsilon robustness.



(a)



(b)



(c)

Figure 1: (a): Plot of the two-dimensional toy example. Data points are colored based on their classes (class A: red and class B: blue). (b): Visualization of the adversary regions for SAA when $\epsilon = 0.5$. The regions are colored differently depending on their classes (class A: magenta and class B: cyan). The decision boundary of a single nearest neighbor classifier is shown as a dashed black curve. (c): Visualization of the adversary regions for SAA when $\epsilon = 1.0$. The overlapping adversary regions from the samples with different classes are colored in purple.

2 Voronoi-Epsilon Adversarial Accuracy (VAA)

Our approach restricts the allowed region of the perturbations to avoid the tradeoff originating from the definition of standard adversarial accuracy. This is achieved without limiting the magnitude of ϵ and without using an external model. We want to have the following property to avoid the tradeoff.

$$\forall x_i, x_j \in X,\ x_i \neq x_j \implies R(x_i) \cap R(x_j) = \varnothing \quad (1)$$

When Property (1) holds for the adversary region, we no longer have the tradeoff as $x_i \notin R(x_j)$ for $x_i \neq x_j$. In other words, a clean sample cannot be an adversarial example originating from another clean sample. We propose a new adversary called a Voronoi-epsilon adversary that combines the Voronoi-adversary introduced by Khoury and Hadfield-Menell [2019] with an ϵ-ball-based adversary. This adversary is constrained to an adversary region $Vor(x) \cap \mathbb{B}(x, \epsilon)$ where $Vor(x)$ is the (open) Voronoi cell around a data point $x \in X$. $Vor(x)$ consists of every point in $\mathcal{X}$ that is closer than any $x_{clean} \in X - \{x\}$. Mathematically, $Vor(x) = \{x' \in \mathcal{X} | d(x, x') < d(x_{clean}, x'), \forall x_{clean} \in X - \{x\}\}$. Then, Property (1) holds as $Vor(x_i) \cap Vor(x_j) = \varnothing$ for $x_i \neq x_j$.

Based on a Voronoi-epsilon adversary, we define Voronoi-epsilon adversarial accuracy (VAA).

Definition 3 (Voronoi-epsilon adversarial accuracy). *When a Voronoi-epsilon adversary is used for the adversary, we refer to the adversarial accuracy as Voronoi-epsilon adversarial accuracy (VAA) $a_{Vor}(\epsilon)$. For VAA, we denote $R(x)$ as $R_{Vor}(\epsilon; x)$, i.e., $R_{Vor}(\epsilon; x) = Vor(x) \cap \mathbb{B}(x, \epsilon)$.*

$$a_{Vor}(\epsilon) = \mathbb{E}_{(x,c_x)\sim\mathcal{D}} \left[\mathbb{1}\left(f(x^*) = c_x\right)\right]^1$$

where $x^* = \underset{x' \in R_{Vor}(\epsilon;x)}{\arg\max} L(x', c_x)$.

Figure 2 shows the adversary regions for VAA with varying ϵ values. When $\epsilon = 0.5$, the regions are same with SAA except for the points $(1.5, 1), (1.5, -1)$ and $(2, -1.5)$. Even when ϵ is large ($\epsilon > 0.5$), there is no overlapping adversary

[1] Using the expectation here is a slight abuse of notation, since $a_{Vor}(\cdot)$ is defined on a finite set. We retain it for consistency with previous definitions, and understand it to mean the empirical average.

region, which was a source of the tradeoff in SAA. Therefore, when we choose a robust model based on VAA, we can get a model that is both accurate and robust. Figure 2c shows the single nearest neighbor (1-NN) classifier would maximize VAA. The adversary regions cover most of the points in $\mathbb{R}^2$ for large ϵ.

Observation 1. *Let d_{min} be the nearest distance of the data point pairs, i.e., $d_{min} = \min_{x_i, x_j \in X, x_i \neq x_j} d(x_i, x_j)$. Then, the following equivalence holds.*

$$a_{Vor}(\epsilon) = a_{std}(\epsilon), \tag{2}$$

when $\epsilon < \frac{1}{2} d_{min}$.

Observation 1 shows that VAA is equivalent to SAA for sufficiently small ϵ values. This indicates that VAA is an extension of SAA that avoids the tradeoff when ϵ is large. The proof of the observation is in Appendix A.1. We point out that equivalent findings were also mentioned in Yang et al. [2020a,b], Khoury and Hadfield-Menell [2019].

As explained in Section 1.3.1, studying the local robustness of classifiers has a limitation. Attackers can attack models with only local robustness by using large ϵ values. The absence of a tradeoff between accuracy and VAA enables us to increase ϵ values and to study global robustness. We define a measure for global robustness using VAA.

Definition 4 (Global Voronoi-epsilon robustness). *Global Voronoi-epsilon robustness a_{global} is defined as:*

$$a_{global} = \lim_{\epsilon \to \infty} a_{Vor}(\epsilon).$$

Global Voronoi-epsilon robustness considers the robustness of classifiers for most points in $\mathcal{X}$ (all points except for Voronoi boundary $VB(X)$, which is the complement set of the unions of Voronoi cells.). We derive the following theorem from global Voronoi-epsilon robustness.

Theorem 1. *A single nearest neighbor (1-NN) classifier maximizes global Voronoi-epsilon robustness a_{global} on training data. 1-NN classifier is a unique classifier that satisfies this except for Voronoi boundary $VB(X)$.*

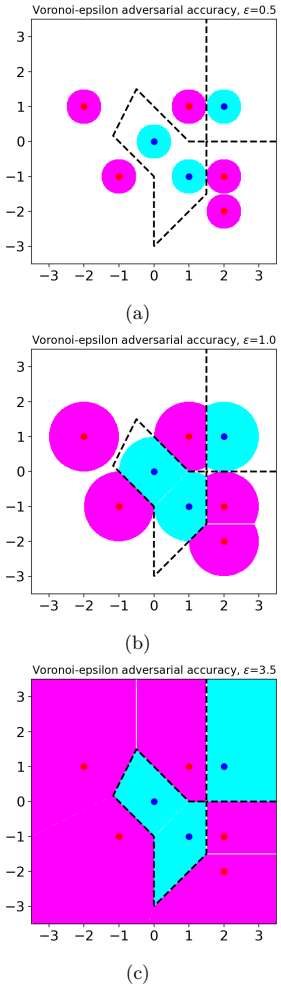


Figure 2: Visualization of the adversary regions for VAA with varying ϵ values. The data points and regions are colored as in Figure 1. (a): When $\epsilon = 0.5$. (b): When $\epsilon = 1.0$. (c): When $\epsilon = 3.5$.

When l_p norm with $1 < p < \infty$ is used as a distance metric, a data point will almost never lie on the Voronoi boundary $VB(X)$ in practical situations with only finite number of available training data points. Note that Theorem 1 only holds for exactly the same data under the exclusive class condition as mentioned in the problem settings 1.1. It does not take into account generalization. The proof of the theorem is in A.2.

3 Discussion

In this work, we address the tradeoff between accuracy and adversarial robustness by introducing the Voronoi-epsilon adversary. Another way to address this tradeoff is to use a Bayes optimal classifier [Suggala et al., 2019, Kim and Wang, 2020]. Since this is not available in practice, a reference model must be used as an approximation. In that case, the meaning of adversarial robustness is dependent on the choice of the reference model. VAA removes the need for a reference model by using the data point set X and the distance metric d to construct adversary. This is in contrast to Khoury and Hadfield-Menell [2019] who used Voronoi cell-based constraints (without ϵ-balls) for an adversarial training purpose, but not for measuring adversarial robustness.

By avoiding the tradeoff with VAA, we can extend the study of local robustness to global robustness. Also, Theorem 1 implies that VAA is a measure of agreement with the 1-NN classifier. For sufficiently small ϵ values, SAA is also a measure of agreement with the 1-NN classifier because SAA is equivalent to VAA as in Observation 1. This implies that many adversarial defenses [Goodfellow et al., 2015, Madry et al., 2018, Zhang et al., 2019, Wong and Kolter, 2018, Cohen et al., 2019] with small ϵ values unknowingly try to make locally the same predictions with a 1-NN classifier.

In our analysis, we do not take into account generalization, and robust models are known to often generalize poorly [Raghunathan et al., 2020]. Many defense models use softmax classifiers and the final classifications of softmax classifiers are done on the trained feature representations. The close relationship between adversarially robust models and the 1-NN classifier revealed by Observation 1 and Theorem 1 indicates that feature representations are affected by the distance relationship in the input space. It will be worth exploring if that can explain the reduced discriminative power [Wu et al., 2021] of robust models and their decreased generalization power.

Acknowledgments

We thank Dr. Nils Olav Handegard, Dr. Yi Liu, and Jungeum Kim for the helpful feedback. We also thank Dr. Wieland Brendel for the helpful discussions.

References

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013. doi: https://doi.org/10.1007/978-3-642-40994-3_25.

J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019. URL `https://proceedings.mlr.press/v97/cohen19c.html`.

E. Dohmatob. Generalized no free lunch theorem for adversarial robustness. In *International Conference on Machine Learning*, pages 1646–1654. PMLR, 2019. URL `https://proceedings.mlr.press/v97/dohmatob19a.html`.

A. Ghiasi, A. Shafahi, and T. Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. In *International Conference on Learning Representations*, 2019. URL `https://iclr.cc/virtual_2020/poster_HJxdTxHYvB.html`.

I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. doi: https://doi.org/10.48550/arXiv.1412.6572.

M. Khoury and D. Hadfield-Menell. Adversarial training with Voronoi constraints. *arXiv preprint*

arXiv:1905.01019, 2019. doi: https://doi.org/10.48550/arXiv.1905.01019.

J. Kim and X. Wang. Sensible adversarial learning, 2020. URL https://openreview.net/forum?id=rJlf_RVKwr.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

A. Raghunathan, S. M. Xie, F. Yang, J. Duchi, and P. Liang. Understanding and mitigating the tradeoff between robustness and accuracy. In *International Conference on Machine Learning*, pages 7909–7919. PMLR, 2020. URL https://proceedings.mlr.press/v119/raghunathan20a.html.

A. S. Suggala, A. Prasad, V. Nagarajan, and P. Ravikumar. Revisiting adversarial risk. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2331–2339. PMLR, 2019. URL http://proceedings.mlr.press/v89/suggala19a.html.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. 2014. doi: https://doi.org/10.48550/arXiv.1312.6199. 2nd International Conference on Learning Representations, ICLR 2014; Conference date: 14-04-2014 Through 16-04-2014.

D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyxAb30cY7.

E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018. URL http://proceedings.mlr.press/v80/wong18a.html?ref=https://githubhelp.com.

Z. Wu, H. Gao, S. Zhang, and Y. Gao. Understanding the robustness-accuracy tradeoff by rethinking robust fairness. 2021. URL https://openreview.net/forum?id=bl9zYxOVwa.

Y.-Y. Yang, C. Rashtchian, Y. Wang, and K. Chaudhuri. Robustness for non-parametric classification: A generic attack and defense. In *International Conference on Artificial Intelligence and Statistics*, pages 941–951. PMLR, 2020a. URL http://proceedings.mlr.press/v108/yang20b.html.

Y.-Y. Yang, C. Rashtchian, H. Zhang, R. R. Salakhutdinov, and K. Chaudhuri. A closer look at accuracy vs. robustness. *Advances in Neural Information Processing Systems*, 33, 2020b. URL https://proceedings.neurips.cc/paper/2020/hash/61d77652c97ef636343742fc3dcf3ba9-Abstract.html.

H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pages 7472–7482. PMLR, 2019. URL https://proceedings.mlr.press/v97/zhang19p.html.

A Appendix

A.1 Proof of Observation 1

To prove Observation 1, we introduce the following lemma.

Lemma 1. *When N is the number of data points, let $x_2, \cdots, x_N \in X - \{x\}$ be the sorted neighbors of a data point $x \in X$. Mathematically, $d(x, x_2) \leq d(x, x_3) \leq \cdots \leq d(x, x_N)$. Then, when $\epsilon < \frac{1}{2}d(x, x_2)$, the following equation holds.*

$$R_{Vor}(\epsilon; x) = \mathbb{B}(x, \epsilon). \tag{3}$$

Proof. **Lemma 1**
We only consider when $\epsilon < \frac{1}{2}d(x, x_2)$.
Let $x' \in \mathbb{B}(x, \epsilon)$. Then, $d(x, x') \leq \epsilon$.
$\frac{1}{2}d(x, x_2) \leq \frac{1}{2}d(x, x_{clean}), \forall x_{clean} \in X - \{x\}$.
Due to the triangle inequality, $\frac{1}{2}d(x, x_{clean}) \leq \frac{1}{2}d(x, x') + \frac{1}{2}d(x', x_{clean})$.
When we combine the above inequalities, $d(x, x') \leq \epsilon < \frac{1}{2}d(x, x_2) \leq \frac{1}{2}d(x, x_{clean}) \leq \frac{1}{2}d(x, x') + \frac{1}{2}d(x', x_{clean}), \forall x_{clean} \in X - \{x\}$.
Then, $\frac{1}{2}d(x, x') < \frac{1}{2}d(x', x_{clean}) = \frac{1}{2}d(x_{clean}, x'), \forall x_{clean} \in X - \{x\}$. Thus, $x' \in Vor(x)$.
Hence, $\mathbb{B}(x, \epsilon) \subset Vor(x)$ and $R_{Vor}(\epsilon; x) = \mathbb{B}(x, \epsilon) \cap Vor(x) = \mathbb{B}(x, \epsilon)$.
□

Now, we prove Observation 1.

Proof. **Observation 1**
$d_{min} \leq d(x, x_i), \forall x, x_i \in X, x \neq x_i$.
When $\epsilon < \frac{1}{2}d_{min}$, $\epsilon < \frac{1}{2}d_{min} \leq \frac{1}{2}d(x, x_i), \forall x, x_i \in X, x \neq x_i$. Thus, $R_{Vor}(\epsilon; x) = \mathbb{B}(x, \epsilon), \forall x \in X$ due to the equation (3) in Lemma 1.
Then, $a_{Vor}(\epsilon)$ is same with $a_{std}(\epsilon)$ as $R_{Vor}(\epsilon; x) = \mathbb{B}(x, \epsilon) = R_{std}(\epsilon; x), \forall x \in X$.
□

A.2 Proof of Theorem 1

To prove Theorem 1, we introduce the following lemma.

Lemma 2. *By changing ϵ and $x \in X$, x' that satisfies $x' \in R_{Vor}(\epsilon; x)$ can fill up $\mathcal{X}$ except for Voronoi boundary $VB(X)$. In other words, $VB(X)^{\mathsf{c}} = \mathcal{X} - VB(X) \subset \bigcup_{\epsilon \geq 0} \left(\bigcup_{x \in X} R_{Vor}(\epsilon; x) \right)$.*

Proof. **Lemma 2**
Let $x' \in VB(X)^{\mathsf{c}}$.
Note that mathematically, $VB(X) = \left(\bigcup_{x \in X} Vor(x) \right)^{\mathsf{c}}$.
Hence, $VB(X)^{\mathsf{c}} = \left(\left(\bigcup_{x \in X} Vor(x) \right)^{\mathsf{c}} \right)^{\mathsf{c}} = \bigcup_{x \in X} Vor(x)$.
$\exists x \in X$ such that $x' \in Vor(x)$.
Let $\epsilon^* = d(x, x')$. Then, $d(x, x') \leq \epsilon^*$ and $x' \in Vor(x)$.
$x' \in \mathbb{B}(x, \epsilon^*) \cap Vor(x) = R_{Vor}(\epsilon^*; x) \subset \bigcup_{\epsilon \geq 0} \left(\bigcup_{x \in X} R_{Vor}(\epsilon; x) \right)$.
We proved $VB(X)^{\mathsf{c}} \subset \bigcup_{\epsilon \geq 0} \left(\bigcup_{x \in X} R_{Vor}(\epsilon; x) \right)$. □

Now, we prove Theorem 1.

Proof. **Part 1**
First, we prove that a 1-NN classifier maximizes global Voronoi-epsilon robustness. We denote the 1-NN classifier as f_{1-NN} and calculate its global Voronoi-epsilon robustness.
For a data point $x \in X$, let $x' \in R_{Vor}(\epsilon; x) = \mathbb{B}(x, \epsilon) \cap Vor(x)$.
$x' \in Vor(x) \iff d(x, x') < d(x_{clean}, x'), \forall x_{clean} \in X - \{x\}$.
As $x' \in R_{Vor}(\epsilon; x) \subset Vor(x)$, x is unique nearest data point in X and thus $f_{1-NN}(x') = c_x$.
When $x^* = \underset{x' \in R_{Vor}(\epsilon; x)}{\arg\max} L(x', c_x)$,
$a_{Vor}(\epsilon) = \mathbb{E}_{(x, c_x) \sim \mathcal{D}} \left[\mathbb{1} \left(f_{1-NN}(x^*) = c_x \right) \right] = \mathbb{E}_{(x, c_x) \sim \mathcal{D}} [1] = 1$.
$a_{global} = \lim_{\epsilon \to \infty} a_{Vor}(\epsilon) = \lim_{\epsilon \to \infty} 1 = 1$. Thus, f_{1-NN} takes the maximum global Voronoi-epsilon robustness 1.

Part 2
Now, we prove that if f^* maximizes global Voronoi-epsilon robustness, then f^* becomes the 1-NN classifier except for Voronoi boundary $VB(X)$.
Let f^{*1} be a function that maximizes global Voronoi-epsilon robustness.
From the last part of the part 1, when we calculate global Voronoi-epsilon robustness of f^{*1}, it should satisfy the equation $a_{global} = 1$.
For a data point $x \in X$ and $\epsilon_1 < \epsilon_2$, $R_{Vor}(\epsilon_1; x) = \mathbb{B}(x, \epsilon_1) \cap Vor(x) \subset \mathbb{B}(x, \epsilon_2) \cap Vor(x) = R_{Vor}(\epsilon_2; x)$.
Thus, for a data point $x \in X$ and $\epsilon_1 < \epsilon_2$,

$L(x^{*1}, c_x) \leq L(x^{*2}, c_x)$ where $x^{*1} = \underset{x' \in R_{Vor}(\epsilon_1; x)}{\arg\max} L(x', c_x)$ and $x^{*2} = \underset{x' \in R_{Vor}(\epsilon_2; x)}{\arg\max} L(x', c_x)$. From the definition of L, $l(f^{*1}(x^{*1}), c_x) \leq l(f^{*1}(x^{*2}), c_x)$. From the necessary condition of classification loss l, we obtain the inequality $\mathbb{1}\left(f^{*1}(x^{*1}) = c_x\right) \geq \mathbb{1}\left(f^{*1}(x^{*2}) = c_x\right)$. $a_{Vor}(\epsilon_1) = \mathbb{E}_{(x,c_x)\sim\mathcal{D}}\left[\mathbb{1}\left(f^{*1}(x^{*1}) = c_x\right)\right] \geq \mathbb{E}_{(x,c_x)\sim\mathcal{D}}\left[\mathbb{1}\left(f^{*1}(x^{*2}) = c_x\right)\right] = a_{Vor}(\epsilon_2)$ for $\epsilon_1 < \epsilon_2$. In other words, $a_{Vor}(\epsilon)$ is a monotonically decreasing (non-increasing) function.

$a_{Vor}(\epsilon) = 1, \forall \epsilon \geq 0$ ($\because$ If $a_{Vor}(\epsilon^*) < 1$ for an $\epsilon^* > 0$, then it is a contradictory to $a_{global} = 1$ as $a_{Vor}(\epsilon)$ is a monotonically decreasing function.)

$1 = a_{Vor}(\epsilon) = \mathbb{E}_{(x,c_x)\sim\mathcal{D}}\left[\mathbb{1}\left(f^{*1}(x^*) = c_x\right)\right]$ where $x^* = \underset{x' \in R_{Vor}(\epsilon; x)}{\arg\max} L(x', c_x)$.

As the calculation is based on the finite set X, $f^{*1}(x^*) = c_x$ ($\because \mathbb{1}\left(f^{*1}(x^*) = c_x\right) = 1$) where $x^* = \underset{x' \in R_{Vor}(\epsilon; x)}{\arg\max} L(x', c_x)$.

As x^* are the worst case adversarially perturbed samples, i.e., samples that output mostly different from c_x, $f^{*1}(x') = c_x = f_{1-NN}(x')$ where $x' \in R_{Vor}(\epsilon; x)$.

By changing ϵ and $x \in X$, x' that satisfies $x' \in R_{Vor}(\epsilon; x)$ can fill up $\mathcal{X}$ except for $VB(X)$ ($\because$ Lemma 2). Hence, f^{*1} is equivalent to f_{1-NN} except for Voronoi boundary $VB(X)$.

□

Article IV

IV Inspecting class hierarchies in classification-based metric learning models

Hyeongji Kim, Pekka Parviainen, Terje Berge, and Ketil Malde. *arXiv preprint: 2301.11065*, 2023.

Inspecting class hierarchies in classification-based metric learning models

Hyeongji Kim[1,2*], Pekka Parviainen[2], Terje Berge[1], Ketil Malde[1,2]

1 Institute of Marine Research, Bergen, Norway
2 Department of Informatics, University of Bergen, Norway

* hjk92g@gmail.com

Abstract

Most classification models treat all misclassifications equally. However, different classes may be related, and these hierarchical relationships must be considered in some classification problems. These problems can be addressed by using hierarchical information during training. Unfortunately, this information is not available for all datasets. Many classification-based metric learning methods use class representatives in embedding space to represent different classes. The relationships among the learned class representatives can then be used to estimate class hierarchical structures. If we have a predefined class hierarchy, the learned class representatives can be assessed to determine whether the metric learning model learned semantic distances that match our prior knowledge. In this work, we train a softmax classifier and three metric learning models with several training options on benchmark and real-world datasets. In addition to the standard classification accuracy, we evaluate the hierarchical inference performance by inspecting learned class representatives and the hierarchy-informed performance, i.e., the classification performance, and the metric learning performance by considering predefined hierarchical structures. Furthermore, we investigate how the considered measures are affected by various models and training options. When our proposed ProxyDR model is trained without using predefined hierarchical structures, the hierarchical inference performance is significantly better than that of the popular NormFace model. Additionally, our model enhances some hierarchy-informed performance measures under the same training options. We also found that convolutional neural networks (CNNs) with random weights correspond to the predefined hierarchies better than random chance.

Introduction

Neural network-based classifiers have shown impressive classification accuracy. For instance, a convolutional neural network (CNN) classifier [1] surpassed human-level top-5 classification accuracy (94.9%) [2] on the 1000-class classification challenge on the ImageNet dataset [3]. Most training loss functions in neural network classifiers treat all misclassifications equally. However, in practice, the severity of various misclassifications may differ considerably. For instance, in an autonomous vehicle system, mistaking a person as a tree can result in a more catastrophic consequence than mistaking a streetlight as a tree [4]. In addition, some classification tasks include a large number of classes, such as the 1000-class ImageNet classification challenge, and hierarchical relationships may exist among these classes. When several classification models

achieved similar accuracy, one would prefer to choose models in which the wrongly predicted classes are "hierarchically" close to the ground-truth classes. To address the severity of misclassifications and relationships among classes, hierarchical information can be used. For instance, predefined hierarchical structures can be incorporated into training by replacing standard labels with soft labels based on this hierarchical information [4]. Some metric learning approaches also use hierarchical information [5–7].

In general, metric learning methods learn embedding functions in which similar data points are close and dissimilar data points are far apart according to the distance metric in the learned embedding space. For class-labeled datasets, data points in the same class are regarded as similar, while data points in different classes are regarded as dissimilar. Metric learning can be applied in image retrieval tasks [8] to identify relevant images or few-shot classification tasks [9, 10], which are classification tasks with only a few examples per class. Usually, metric learning methods assume that there are no special relations among classes, i.e., a flat hierarchy is assumed. However, a predefined class hierarchy can be incorporated into the training process of metric learning models to improve the hierarchy-informed performance [5–7].

Class hierarchical structures can be defined in several ways. Hierarchical structures can be defined by domain experts [6] or extracted from WordNet [11], which is a database that contains semantic relations among English words. For instance, ImageNet classes [3] are organized according to WordNet. However, hierarchy determination from WordNet requires that a class name or its higher class is in the WordNet database. When these approaches are not applicable, hierarchy can be inferred by estimating the class distance matrix based on learned classifiers. For instance, the confusion matrix of a classifier can be used to estimate relations among class pairs [12]. Each row in the confusion matrix can be treated as a vector, and the distance between these vectors can be calculated to estimate the class hierarchical structure. However, this approach can be cumbersome for hierarchy-informed classification tasks, as they require separate training and evaluation (validation) processes to determine the hierarchical structure. Moreover, this approach becomes challenging when some classes contain a very small number of data points, as some elements in the confusion matrix may be uninformative. One type of metric learning model uses a unique position in embedding space (class representative) to represent each class [13–16]. Class representatives can also be used to infer hierarchical structures by considering their distances [6, 17]. This approach does not require a separate training process, as class representatives can be learned automatically.

On the other hand, when we have a predefined hierarchy, the learned class representatives can improve our understanding of the trained metric learning model. For example, we can determine if the semantic distance learned by a metric learning model matches our prior knowledge (that is, the predefined hierarchy). For instance, when a model has been trained to classify species, we can determine if the model regards a dog as closer to a cat than to a rose. Furthermore, these inspections can be used to evaluate the trustworthiness of a model. However, previous works have paid little attention to the relationships among class representatives. In this work, we assess several metric learning methods and training options by focusing on their learned class representatives and hierarchy-informed performance. Moreover, we attempt to determine conditions that improve the hierarchy inference performance and evaluate whether such models enhance the hierarchy-informed performance. We also investigate different training options with predefined hierarchies for comparison.

Problem settings

The predefined hierarchical distance, i.e., the distance between two classes, often needs to be considered when training models with hierarchical information and evaluating

model performance. Barz and Denzler [5] and Bertinetto et al. [4] used bounded ($[0, 1]$) dissimilarity based on the height of the lowest common ancestor (LCA) between two classes. In this work, similar to Garnot and Landrieu [6], we define the hierarchical distance as the shortest path distance between two classes. For instance, in the predefined hierarchical tree shown in Fig. 1, the hierarchical distance between the classes "tiger" and "woman" is 4 ($= 2 + 2$), as we need to move two steps upward and two steps downward to move from one class to the other. Similarly, the hierarchical distance between the classes "tiger" and "shark" is 6. Hierarchical structures can be expressed as trees or directed acyclic graph (DAG) structures [18]. In this work, we consider tree-structured hierarchies. In other words, each node cannot have more than one parent node.

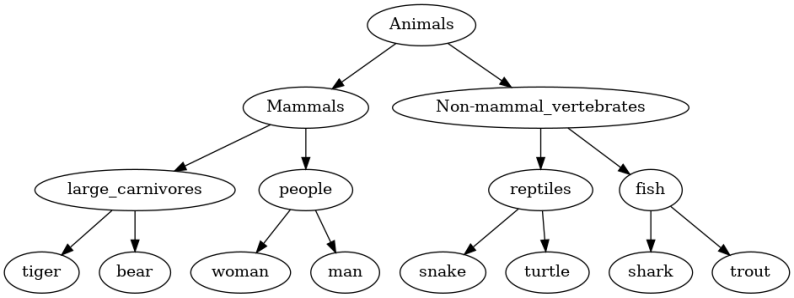


Fig 1. Pruned CIFAR100 tree structure for visualization. The whole hierarchical structure is shown in a table in S2 Appendix.

Let $\mathcal{X} \subseteq \mathbb{R}^{d_I}$ be an input space and $\mathcal{Y}$ be a set of classes. The data points $x \in \mathcal{X}$ and corresponding classes $c \in \mathcal{Y}$ are sampled from the joint distribution $\mathcal{D}$. The feature mapping $f : \mathcal{X} \to \mathcal{Z}$ extracts feature vectors according to the inputs, where $\mathcal{Z} = \mathbb{R}^{d_F}$ is a raw feature space. In this work, we assume that this mapping is modeled by a neural network.

Softmax classification

The softmax classifier is commonly used in neural network-based classification tasks. The softmax classifier estimates the class probability $p(c|x)$, which is the probability that data point x belongs to class c, as follows:

$$p(c|x) = \frac{\exp(l_c(x))}{\sum_{y \in \mathcal{Y}} \exp(l_y(x))}, \tag{1}$$

where logit $l_y(x)$ is the neural network output according to input $x \in \mathcal{X}$ and class $y \in \mathcal{Y}$. Usually, logit $l_y(x)$ is calculated as:

$$l_y(x) = W_y^T f(x) + b_y, \tag{2}$$

where the feature vector $f(x)$ is an output of a penultimate layer, W_y is a weight vector, and b_y is a bias term for class y. The probability $p(c|x)$ estimated by the model is often called the confidence value (score). Based on the estimated class probabilities, we can use the cross-entropy loss to train the model. This loss measures the difference between the predicted and target probability distributions. The cross-entropy (CE) loss of a

mini-batch $B \subseteq \mathcal{D}$ is defined as:

$$L_{CE} = -\frac{1}{|B|} \sum_{(x_i, c_i) \in B} \log(p(c_i|x_i)). \tag{3}$$

Metric learning

While softmax classifiers are commonly used in deep learning classification, metric learning approaches can be beneficial, as they better control data points in embedding space. Hence, metric learning can provide information on the similarity between different data points. Metric learning approaches can be divided into two categories: (direct) embedding-based methods and classification-based methods. Embedding-based methods [19,20] directly compare data points in embedding space to train an embedding function (feature map) $f(\cdot)$. Because embedding-based methods compare data points directly, they must be trained with pairs (using the contrastive loss) or triplets (using the triplet loss) of data points. Thus, embedding-based methods have high training complexity and require special mining algorithms [13] to prevent slow convergence speeds [14]. On the other hand, classification-based methods [13,15,16,21,22] use class representatives to represent classes. According to the classification loss (cross-entropy loss), class representatives guide data points to converge to class-specific positions. Classification-based methods converge faster than embedding-based methods due to their reduced sampling complexity (training with single data points). In this paper, we focus on classification-based metric learning methods.

NormFace

NormFace [13], which is also known as normalized softmax [23], modifies Eq. 2 in the softmax classifier. NormFace was motivated by the normalization of features during feature comparisons to improve face verification during the testing phase. To apply normalization during both the testing and training phases, NormFace normalizes the feature (embedding) vectors and weight vectors and uses a zero bias term. Previous experiments on metric learning models [8] have shown that NormFace and its variants [15,16,24] achieved competitive performance on metric learning tasks.

We denote $\tilde{v} = \frac{v}{\|v\|}$ for any nonzero vector v and the angle between vectors $\tilde{W}_y$ and $\tilde{f}(x)$ as θ_y. Then, as $\left\|\tilde{W}_y\right\| = 1 = \left\|\tilde{f}(x)\right\|$, we obtain the following equation:

$$\tilde{W}_y^T \tilde{f}(x) = \left\|\tilde{W}_y\right\| \left\|\tilde{f}(x)\right\| \cos\theta_y = \cos\theta_y. \tag{4}$$

According to Eq. 4, the class probability $p(c|x)$ estimated by NormFace can be expressed as:

$$p(c|x) = \frac{\exp(s\tilde{W}_c^T \tilde{f}(x))}{\sum_{y \in \mathcal{Y}} \exp(s\tilde{W}_y^T \tilde{f}(x))} = \frac{\exp(s\cos\theta_c)}{\sum_{y \in \mathcal{Y}} \exp(s\cos\theta_y)}, \tag{5}$$

where $s > 0$ is a scaling factor. NormFace learns embeddings according to this estimation and the cross-entropy loss in Eq. 3.

We next investigate the geometrical meaning of the NormFace classification results. If a data point x is classified as belonging to class c by NormFace, according to Eq. 5, we obtain $\exp(s\cos\theta_c) \geq \exp(s\cos\theta_y)$, i.e., $\cos\theta_c \geq \cos\theta_y$. As the cosine function is a monotonically decreasing function in the interval $(0, \pi)$, we obtain $\theta_c \leq \theta_y$. In terms of angles, the normalized embedding vector $\tilde{f}(x)$ is closer (or equally close) to $\tilde{W}_c$ than $\tilde{W}_y$. As we can classify data points using normalized weight vectors according to this

geometrical interpretation, we can consider $\tilde{W}_y$ as the class representative for a class $y \in \mathcal{Y}$. Fig. 2 visualizes the above explanation.

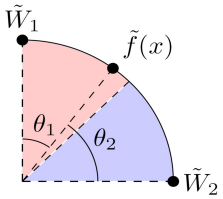


Fig 2. Visualization of NormFace classification results for two classes with $\tilde{W}_1$=(0,1) and $\tilde{W}_2$=(1,0). As $\tilde{f}(x)$ is closer to $\tilde{W}_1$ than $\tilde{W}_2$, data point x is classified as belonging to class 1.

Proxies and prototypes

Although NormFace [13] uses a learnable weight vector as a class representative for each class, the average point for a class can also be used as a class representative. To prevent confusion, we define two kinds of class representatives for each class: a proxy representative and a prototype representative. We refer to the learnable weight vectors as proxy representatives. In NormFace, the weight vectors $\tilde{W}_y$ are proxies. In this work, the proxy representative are used to define the training loss and directly update the network. Note that we can predefine the proxy representatives, and the proxies can be fixed. On the other hand, we refer to the (normalized) average embedding for each class as a prototype representative. We denote the average embedding for class y as vector μ_y. In normalized space, this vector is defined as:

$$\mu_y = \frac{1}{|X_y|} \sum_{x \in X_y} \tilde{f}(x),$$

where $X_y \subseteq \mathcal{X}$ is a set of data points in class y. Then, the prototype representatives are defined as $\tilde{\mu}_y = \frac{\mu_y}{\|\mu_y\|}$. A metric learning method known as the prototypical network [9] was devised for few-shot learning tasks. During the training process, this network uses local prototypes based on special mini-batches known as episodes. In their work, the authors used unnormalized average embedding as prototypes because they considered Euclidean space. In this paper, we do not use prototypes for training, and (global) prototypes using training data are used only for evaluation.

Note that the proxy and prototype representatives are not necessarily the same. For instance, when we train the model with fixed proxies, the positions of the prototypes change during training, while the positions of the proxies remain fixed. Moreover, the confidence values are not necessarily maximized at the proxies [25]. In this case, the data points may not converge to their proxies. This concept is explained further in the next subsubsection.

SD softmax and DR formulation

In our previous work [25], we analyzed a softmax formulation for metric learning in which negative squared distances were used as logit values. We called this formulation the "softmax-based formulation" in our previous paper. In this work, we refer to this formulation as "squared distance (SD) softmax" to prevent confusion with the standard softmax formulation shown in Eq. 1.

We denote the class representative for class y as R_y and the distance to class y as $d_{x,y} := d(f(x), R_y)$ according to the distance function $d(\cdot, \cdot)$. For the normalized embeddings, we define $d_{x,y} := d(\tilde{f}(x), R_y)$ by assuming that point R_y is normalized. Then, the class probability $p(c|x)$ estimated using the SD softmax formulation is:

$$p(c|x) = \frac{\exp(-d_{x,c}^2)}{\sum\limits_{y \in \mathcal{Y}} \exp(-d_{x,y}^2)}. \tag{6}$$

The SD softmax formulation uses the difference in the squared distance for training.

Consider the following equation:

$$\left\| \tilde{W}_y - \tilde{f}(x) \right\|^2 = \left\| \tilde{W}_y \right\|^2 + \left\| \tilde{f}(x) \right\|^2 - 2\tilde{W}_y^T \tilde{f}(x) = 1 + 1 - 2\cos\theta_y.$$

Then, we obtain:

$$s\cos\theta_y - s = -\frac{s}{2} \left\| \tilde{W}_y - \tilde{f}(x) \right\|^2.$$

Thus, we obtain the equation:

$$\frac{\exp(s\cos\theta_c)}{\sum\limits_{y \in \mathcal{Y}} \exp(s\cos\theta_y)} = \frac{\exp(s\cos\theta_c - s)}{\sum\limits_{y \in \mathcal{Y}} \exp(s\cos\theta_y - s)} = \frac{\exp(-\frac{s}{2} \left\| \tilde{W}_c - \tilde{f}(x) \right\|^2)}{\sum\limits_{y \in \mathcal{Y}} \exp(-\frac{s}{2} \left\| \tilde{W}_y - \tilde{f}(x) \right\|^2)}, \tag{7}$$

(This equation is taken from [13]). The above equation shows that the NormFace formulation in Eq. 5 can be considered an SD softmax formulation 6 that uses the Euclidean distance on a hypersphere with radius $\sqrt{\frac{s}{2}}$.

In [25], we found that the SD softmax formulation had two main limitations. First, the estimated probability and corresponding loss may be affected by scaling changes. However, while this limitation must be considered when a Euclidean space is used, this limitation can be addressed by using normalized embeddings, as in the NormFace model [13]. The second limitation is that the estimated class probabilities are not optimized at the class representatives. For instance, the maximum estimated class probability $p(c|x)$ is not found at the class representative of class c. The NormFace model also encounters this issue. In the example shown in Fig. 2 with scaling factor $s = 2$, point $(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ is the point that maximizes the confidence value of class 1.

To address the above limitations, we proposed the distance ratio (DR)-based formulation [25] for metric learning models. Mathematically, the DR formulation estimates the class probability $p(c|x)$ as:

$$p(c|x) = \frac{\frac{1}{d_{x,c}^s}}{\sum\limits_{y \in \mathcal{Y}} \frac{1}{d_{x,y}^s}} = \frac{d_{x,c}^{-s}}{\sum\limits_{y \in \mathcal{Y}} d_{x,y}^{-s}}. \tag{8}$$

The DR formulation uses ratios of distances for training.

Moreover, the DR formulation [25] resolves the above two limitations of the SD softmax formulation. In the example shown in Fig. 2, for any scaling factor s, point $(0, 1) = \tilde{W}_1$ is the point that maximizes the confidence value of class 1. Thus, our experiments showed that the DR formulation has faster or comparable training speed in Euclidean (unnormalized) embedding spaces.

Exponential moving average (EMA) approach

Zhe et al. [21] theoretically showed that, while training NormFace [13], the commonly used gradient descent update method for the proxies based on the training loss cannot guarantee that the updated proxies approach the corresponding prototypes. To address this issue, they proposed using the normalized exponential moving average (EMA) to update the proxies. Mathematically, when updating a proxy for a data point x in class c, the proxy $\tilde{W}_c$ is updated as:

$$\tilde{W}_c := \frac{\alpha\tilde{f}(x) + (1-\alpha)\tilde{W}_c}{\left\|\alpha\tilde{f}(x) + (1-\alpha)\tilde{W}_c\right\|}, \tag{9}$$

where $0 < \alpha < 1$ is a parameter that controls the speed and stability of the updates. Their experimental results showed that the EMA approach achieved better performance than the standard NormFace model on multiple datasets.

Adaptive scaling factor approach

Based on previous observations that the convergence and performance of NormFace models [13] depend on the scale parameter s, Zhang et al. [22] proposed AdaCos, which is a NormFace model trained by using the adaptive scale factor s in Eq. (5). Moreover, they suggested to use parameter s, which significantly changes the probability $p(c|x)$ estimated by Eq. (5), where c is the class of data point x. In other words, they attempted to find a parameter s that maximizes $\left\|\frac{\partial p(c|x)(\theta_c)}{\partial\theta_c}\right\|$ by approximating an s value that satisfies the equation:

$$\left\|\frac{\partial^2 p(c|x)(\theta'_c)}{\partial\theta'^2_c}\right\| = 0, \tag{10}$$

where $\theta'_c := \text{clip}(\theta_c, 0, \frac{\pi}{2})$ and $\text{clip}(\cdot,\cdot,\cdot)$ is a function that limits a value within a specified range. They proposed two AdaCos models: static (fixed) and dynamic versions. The static model determines a good scale parameter s before training the NormFace model based on observations of the angles between the data points and proxies. In the static model, the scale parameter s is not updated. The dynamic model updates the scale parameter s during each iteration based on the current angles between the data points and proxies.

CORR loss

In contrast to previous metric learning approaches that ignored hierarchical relationships among classes, Barz and Denzler [5] used normalized embeddings to achieve hierarchy-informed classification. First, they predefined the positions of the proxies using a given hierarchical structure. These proxies are fixed, i.e., they are not updated during training. Then, they used the predefined proxies to train the models according to the CORR loss. For a data point x in class c, the CORR loss ensures that the embedding vector $\tilde{f}(x)$ is close to the corresponding proxy $\tilde{W}_c$. The CORR loss of a mini-batch $B \subseteq \mathcal{D}$ is defined as:

$$L_{CORR} = \frac{1}{|B|}\sum_{(x_i,c_i)\in B}\left(1 - \tilde{W}_{c_i}^T\tilde{f}(x_i)\right) = \frac{1}{|B|}\sum_{(x_i,c_i)\in B}(1-\cos\theta_{c_i}), \tag{11}$$

where θ_{c_i} is the angle between $\tilde{W}_{c_i}$ and $\tilde{f}(x_i)$.

Methods

We investigated several methods and training options. The details of the settings are described in the following sections. The codes for our experiments will be available in `https://github.com/hjk92g/Inspecting_Hierarchies_ML`.

Dataset We conducted experiments using three plankton datasets (small microplankton, large microplankton, and mesozooplankton) and two benchmark datasets (CIFAR100 [26] and NABirds [27]). Table 1 summarizes the number of classes and images in each dataset. The three plankton datasets were obtained from the Institute of Marine Research (IMR), where flow imaging microscopy is used during routine monitoring. The plankton samples were imaged using three FlowCams, ©2022 Yokogawa Fluid Imaging Technologies, Inc., with different magnification settings. The three plankton datasets contain nonliving classes of artifacts and debris. Moreover, these datasets contain class names that are not in the WordNet database [11]. In addition, the three plankton datasets have severe class imbalances. For example, each class in the small microplankton (MicroS) dataset contains 1 to 456 images. Further details on the plankton datasets can be found in S2 Appendix. We randomly divided each plankton dataset into 70% training data, 10% validation data, and 20% test data. For the plankton datasets, we use images without any augmentation. As the input images have diverse image sizes, we resized the input images to 128×128. The CIFAR100 dataset contains 100 classes and 600 images per class. The CIFAR100 contains 50000 training and 10000 test images, and we randomly divided the original training data into 45000 training and 5000 validation data. Furthermore, we applied random transformations (15 degree range rotations, 10% range translations, 10% range scaling, 10 degree range shearing, and horizontal flips) on the CIFAR100 training data. We did not resize images from their original size of 32×32. The NABirds dataset contains 23929 training and 24633 test images. Similar to the plankton datasets, we randomly divided this dataset into 70% training data, 10% validation data, and 20% test data. We resized the images to 128×128 pixels. We also applied the same augmentations applied to the CIFAR100 dataset on the NABirds dataset. The predefined hierarchical structures are available in S1 Appendix.

Table 1. Summary of the studied datasets.

Dataset	Target particle (plankton datasets)	Images per class	Images	Classes
Small microplankton (MicroS)	5 to 50 μm	1 to 456	6738	109
Large microplankton (MicroL)	35 to 500 μm	2 to 613	8348	102
Mesozooplankton (MesoZ)	180 to 2000 μm	3 to 486	6738	52
CIFAR100 [26]	—	600	60000	100
NABirds [27]	—	13 to 120	48562	555

Models We consider four types of models: the softmax classifier, NormFace, ProxyDR (explained below), and a CORR loss-based model. We focus on metric learning models with normalized embeddings, as normalization is commonly used in metric learning models to improve performance [5, 13, 28]. ProxyDR is a model that uses

proxies for classification and DR formulations (8) to estimate class probabilities $p(c|x)$. Similar to the NormFace model, ProxyDR uses the Euclidean distance on a hypersphere. The difference between the two models is that ProxyDR uses DR formulations, while NormFace uses SD softmax formulations (as shown in Eq. 7). For the plankton datasets, we used a pretrained Inception version 3 architecture [29] as the backbone. For the CIFAR100 and NABirds datasets, we used a pretrained ResNet50 [30] as the backbone. In addition to the backbones, we applied a learnable linear transformation to obtain 128-dimensional embeddings $f(x)$. For the plankton datasets, we incorporate the size information. Specifically, when a data point x has size $v_{size} = [width, height]^T$, we take the elementwise logarithm $v'_{size} = [\log(width), \log(height)]^T$. By applying a linear transformation, we obtain an embedding vector $f_{size}(x)$ according to the size information, i.e., $f_{size}(x) = W_{size}^T v'_{size} + b_{size}$, where W_{size} is a learnable matrix with shape 2×128 and $b_{size} \in \mathbb{R}^2$ is a learnable vector. Then, we add this vector to the original embedding vector, namely, $f(x) := f(x) + f_{size}(x)$.

Training settings We trained the models according to the backbone weights and other weights (linear transformations, proxies). We used the Adam optimizer [31] with a learning rate of 10^{-4}. Except in the case of a dynamic approach (explained below), we use 10.0 as a scaling factor in both NormFace and ProxyDR. We set the training batch size in all experiments to 32. For the plankton datasets, we trained the models for 50 epochs. For the CIFAR100 and NABirds datasets, we trained the models for 100 epochs. During each epoch, we assessed the model accuracy. We chose the model with the highest validation accuracy for testing. For each setting, we trained the models five times with different seeds for the random split.

Training options The different training options are described as follows. The dynamic approach affects the scale factors in Eqs. 5 and 8. The EMA and MDS approaches both affect the proxy calculations.

- Standard: standard training with a fixed scale factor $s = 10$, with proxies updated using the cross-entropy loss (3).
- EMA (exponential moving average): training using the normalized exponential moving average [21], as shown in Eq. 9, to update the proxies. When multiple data points have the same class c in a mini-batch, we apply a modified expression. Specifically, instead of applying the EMA using a single data point, as in Eq. 9, we use the normalized average embedding (local prototype) of the mini-batch. Mathematically, when updating a proxy for m data points $x_{B;1}, \cdots, x_{B;m}$ with class c in mini-batch B, the normalized average embedding is defined as:

$$\tilde{\mu}_{B;c} = \frac{\sum_{i=1}^{m} \tilde{f}(x_{B;i})}{\left\| \sum_{i=1}^{m} \tilde{f}(x_{B;i}) \right\|}.$$

  Then, proxy $\tilde{W}_c$ is updated as:

$$\tilde{W}_c := \frac{\alpha \tilde{\mu}_{B;c} + (1-\alpha)\tilde{W}_c}{\left\| \alpha \tilde{\mu}_{B;c} + (1-\alpha)\tilde{W}_c \right\|}, \tag{12}$$

  where α is the same parameter as in Eq. 9. We set the parameter α to 0.001.

- Dynamic: training with a dynamic scale factor, similar to AdaCos [22]. In contrast to the original paper, which chooses a scale factor using an approximate expression, we use the Adam [31] optimizer to determine a scale factor that satisfies Eq. 10. More details are included in S1 Appendix.

- MDS (multidimensional scaling): training according to predefined hierarchical information. We use the hierarchical information to set (fixed) proxies. First, as in the distance calculation method in the problem setting subsection, we use a predefined hierarchy to generate a distance matrix. We denote the (hierarchical) distance between the ith and jth classes as $d_H(i, j)$. For each hierarchical distance d_H, we apply a transformation $d_T = \frac{\sqrt{2}d_H}{\beta + d_H}$ for a scalar $\beta > 0$ to address the limited Euclidean distance ($\leq \sqrt{2}$) on unit spherical spaces. We set $\beta = 1.0$ in all of our experiments. (Note that if d is a metric, the transformed distance $\frac{\sqrt{2}d}{\beta + d}$ is also a metric.) Then, according to the transformed distance matrix D_T, we use multidimensional scaling (MDS) to set the proxies. Mathematically, we minimize a value known as the normalized stress, which can be expressed as:

$$\text{Stress}(D_{\tilde{W}}) := \frac{\|D_{\tilde{W}} - D_T\|_F}{\|D_T\|_F}, \tag{13}$$

  where $D_{\tilde{W}}$ is a pairwise Euclidean distance matrix according to proxies $\tilde{W}_y$ and $\|\cdot\|_F$ is the Frobenius norm. We use the Adam optimizer with a learning rate of 10^{-3} for 1000 iterations to obtain the proxies. While we used stochastic gradient descent for the MDS option, different methods, such as those applied by Barz and Denzler [5], can also be used for MDS. During training, we fix the obtained proxies and update only the embedding function $f(\cdot)$.

Performance measures In our experiments, we consider three types of performance measures: standard classification measures, hierarchical inference performance measures, and hierarchy-informed performance measures. The hierarchical inference performance measures are used to estimate how well the learned class representatives match the predefined hierarchies. The hierarchy-informed performance measures are used to estimate how well a model performs on classification or similarity measures according to the predefined hierarchies.

We used the top-k accuracy as a standard classification measure, the mean correlation as the hierarchical inference performance measure, and the average hierarchical distance (AHD), hierarchical precision at k (HP@k), hierarchical similarity at k (HS@k), and average hierarchical similarity at k (AHS@k) as hierarchy-informed performance measures. The utilized measures are defined as follows.

- Top-k accuracy: The classification accuracy was calculated, with correct classification defined as whether the labeled class is in the top-k predictions (most likely classes), i. e., the k classes with the highest confidence values. We report results for $k = 1, 5$.

- Mean correlations: We introduce this measure to evaluate how well the learned class representatives match the predefined hierarchical structure. We obtain the class representatives (either proxies or prototypes) from the learned model using training data points. Then, we obtain the pairwise distance matrix D_L based on the class representatives. We compare matrix D_L with the distance matrix D_H based on the predefined hierarchical structure. Specifically, for each class (row), we calculate Spearman's rank correlation coefficient according to the two matrices.

Then, we determine the mean correlation using Fisher transformations. More precisely, we apply a Fisher transformation $\operatorname{arctanh}(\cdot)$ on each correlation coefficient, take the average of the transformed values, and apply $\tanh(\cdot)$ on the average value.
For the plankton datasets, the predefined hierarchical structures of the living classes are based on biological taxonomies. However, these datasets also contain some nonliving classes, such as "large bubbles" and "dark debris". Considering that the predefined hierarchical structures of the living classes are scientifically defined, for the plankton datasets, we report mean correlations among whole classes or only among living classes.

- AHD: The average hierarchical distance of the top-k predictions [4] was calculated as the average hierarchical distance d_H between the labeled classes and each of the top-k most likely classes. In contrast to Bertinetto et al. [4], who considered only misclassified cases, we consider all cases. Hence, in our calculations, the final denominators differ. When $k = 1$, the AHD measure is the same as the average hierarchical cost (AHC) measure defined by Garnot and Landrieu [6]. We report results for $k = 1, 5$.

- HP@k: The hierarchical precision at k [32] was taken as a performance measure. Specifically, let us denote a (hierarchical) neighborhood set of a class c with the distance threshold ϵ as $N(c, \epsilon)$, i.e., $n \in N(c, \epsilon) \iff d_H(c, n) \leq \epsilon$. We define $\text{hCorrectSet}(c, k)$ as the neighborhood set $N(c, \epsilon)$ with the smallest ϵ such that $|\text{hCorrectSet}(c, k)| \geq k$. Then, the hierarchical precision at k is calculated as the fraction of the top-k predictions in $\text{hCorrectSet}(c, k)$. We report the results for $k = 5$.

- HS@k: The hierarchical similarity at k is a measure that was introduced by Barz and Denzler [5] with the name "hierarchical precision at k", although this metric does not evaluate precision and instead assesses similarity. Here, we use a different measure with the same name that was defined by Frome et al. [32]. Hence, we renamed the measure "hierarchical similarity at k". When c is a label of a query data point x, let $R = ((x_1, c_1), \cdots, (x_m, c_m))$ be the ordered list of image-label pairs based on the distance (sorted by ascending distance) to point x in the normalized embedding space. Considering $\cos(\theta) = 1 - \frac{\|u_1 - u_2\|^2}{2}$, where u_1 and u_2 are unit vectors and θ is the angle between u_1 and u_2, we defined the similarity between the ith and jth classes $s_H(i, j)$ as:

$$s_H(i, j) = 1 - \frac{d_T(i, j)^2}{2}, \tag{14}$$

  where i and j are the indices for classes ($1 \leq i, j \leq |\mathcal{Y}|$).
  The hierarchical similarity at k is then defined as:

$$HS@k := \frac{\sum_{i=1}^{k} s_H(I(c), I(c_i))}{\max_\pi \sum_{i=1}^{k} s_H(I(c), I(c_{\pi_i}))}, \tag{15}$$

  where $I(\cdot)$ is an index function that outputs the corresponding index (between 1 and $|\mathcal{Y}|$) for a class and π is an index permutation that ranges from 1 to m. We report results for $k = 50, 250$.

- AHS@K: The average hierarchical similarity at K was introduced by Barz and Denzler [5] as the "average hierarchical precision at K". Due to similar reasons as

for HS@k, we renamed the measure. The average hierarchical similarity at K is defined as the area under the curve of HS@k from $k = 1$ to $k = K$. We report results for $K = 250$.

Results

Main results

The performance measure evaluation results are shown in bar plots with 95% confidence intervals. Dashed lines separate models trained with or without predefined hierarchical knowledge. We show only the results on the CIFAR100 and NABirds datasets in the main text. All results are included in S3 Appendix. To reduce spurious findings, we focus on consistent trends across the five datasets.

Figs. 3 and 4 and the figures in S3 Appendix show the top-k accuracy for various training settings. The NormFace and ProxyDR models achieved comparable top-k accuracy for both k values on most datasets. The softmax loss model obtained low top-k accuracy. While the result was not significant, using the dynamic option achieved higher top-1 accuracy than standard training. When the EMA option was added to the standard and dynamic options, the top-5 accuracy decreased, except for standard NormFace on the NABirds dataset. While the CORR loss achieved top-1 accuracy that was comparable to that achieved by other training options with predefined hierarchical information, this model obtained low top-5 accuracy on all datasets. The top-5 accuracy with the CORR loss was even lower than that with the softmax loss, except on the NABirds dataset. Although the results were better than those of the CORR loss model, the use of predefined hierarchical information during training for NormFace and ProxyDR also reduced the top-5 accuracy. These results show the opposite trend to the changes in the top-1 accuracy, which showed comparable or enhanced performance. Moreover, the dynamic MDS approach obtained lower top-5 accuracy than the MDS approach without the dynamic option.

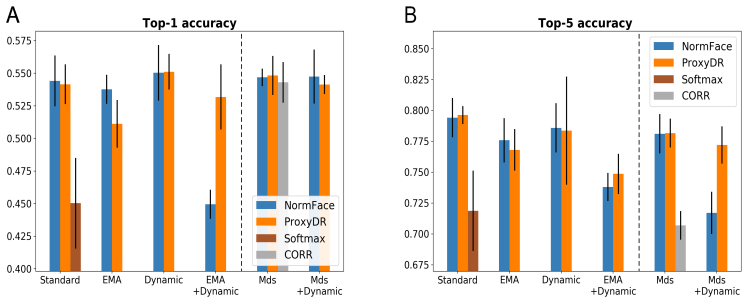


Fig 3. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the CIFAR100 dataset.

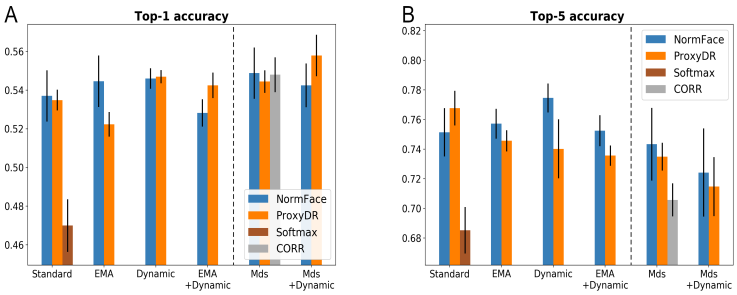


Fig 4. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the NABirds dataset.

Figs. 5 and 6 and the figures in S3 Appendix show the mean correlation values for various training options. When we consider training options that do not use predefined hierarchical information, ProxyDR obtains higher mean correlations than NormFace, except for the EMA approaches. The ProxyDR model with the dynamic option obtained higher mean correlations than the standard ProxyDR model. As expected, the use of predefined hierarchical information greatly increased the mean correlations based on prototypes in both the NormFace and ProxyDR models, except the ProxyDR model on the NABirds dataset. When we consider training options that utilize predefined hierarchical information, the CORR loss model achieved the highest mean correlations based on prototypes in most cases. The ProxyDR model typically achieved the second-best mean correlations.

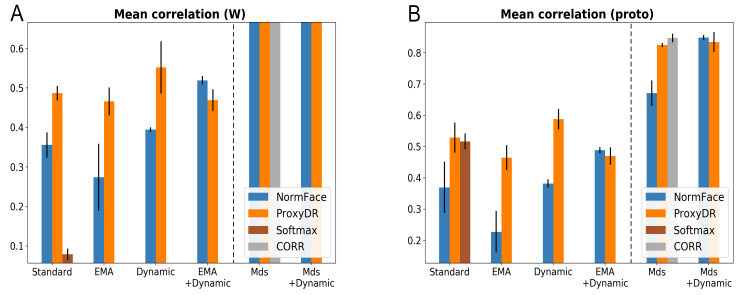


Fig 5. Correlation measures on the CIFAR100 dataset. (A) Values using proxies. (B) Values using prototypes. The mean correlation value based on proxies with the MDS option was 0.8580.

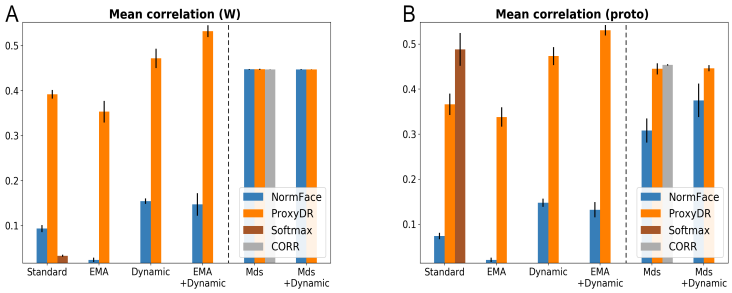


Fig 6. Correlation measures on the NABirds dataset. (A) Values using proxies. (B) Values using prototypes. The mean correlation value based on proxies with the MDS option was 0.4476 (this value is small because the dataset contains 555 classes and the embedding dimension is 128.).

Figs. 7 and 8 and the figures in S3 Appendix show the hierarchical performance measures obtained with various training options. The softmax loss option achieved the worst hierarchical performance, except for AHS@50 on the NABirds dataset. Although these measures are used to assess the hierarchy-informed performance, some of the measures are not substantially affected by the use of predefined hierarchical information during training. For instance, the use of predefined hierarchical information in the CIFAR100 dataset (Fig. 7) did not show noticeable improvements in terms of the AHD (k=1), HS@50, and AHS@250 measures. Moreover, while the use of predefined hierarchical information significantly improved the HS@250 results on most datasets, only marginal improvements were observed for the CIFAR100 dataset. On the other hand, the use of predefined hierarchical information significantly improved the AHD (k=5) and HP@5 results on all datasets. The CORR loss achieved the best results on these two measures, except on the NABirds dataset. Adding the dynamic option to the standard and MDS options improved performance on these two measures, except for the ProxyDR model on the CIFAR100 dataset. When we consider training options that do not use predefined hierarchical information, the ProxyDR model shows better performance than NormFace in terms of these two measures, except for the EMA and dynamic options on the CIFAR100 dataset. Under the same settings, ProxyDR performed better than NormFace in terms of the HS@250 measure, except for the EMA option on the CIFAR100 dataset. Moreover, under the same settings, the ProxyDR model with the dynamic option achieved the highest HS@250 and AHS@250 values among the compared models.

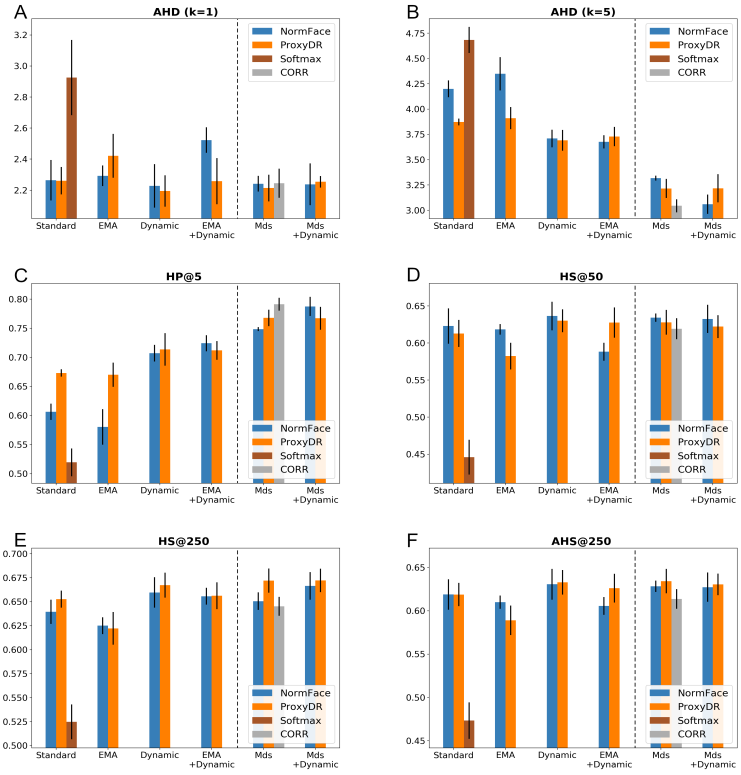


Fig 7. Hierarchical performance measures on the CIFAR100 dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

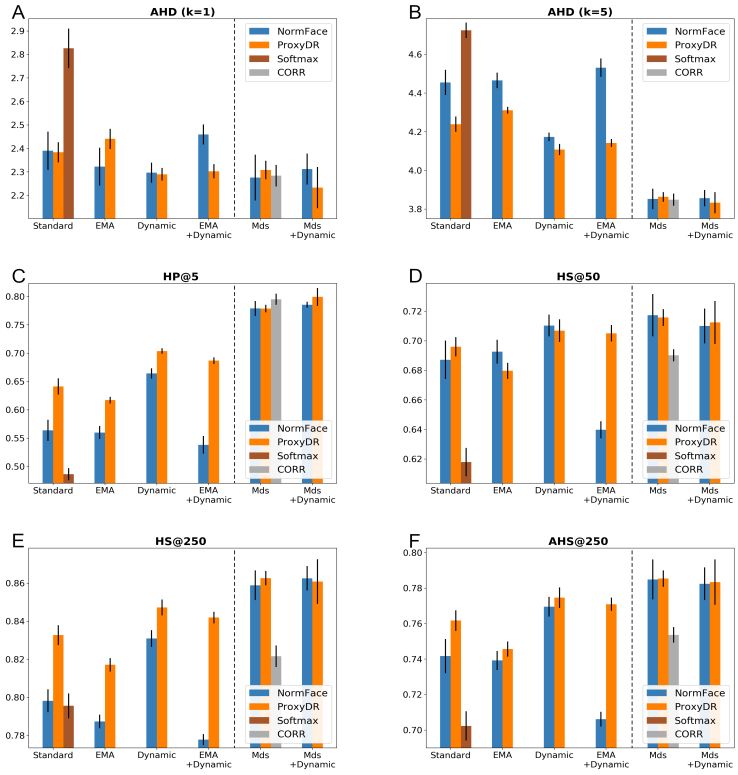


Fig 8. Hierarchical performance measures on the NABirds dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

Additional mean correlation results

To investigate the changes in the class representatives, we evaluated the mean correlations at the end of each training epoch. We report only the results on the CIFAR100 and NABirds datasets. Moreover, we report results for ProxyDR models with the standard and dynamic options. More results are included in S3 Appendix. Furthermore, we report mean correlations based on random networks, i.e., networks with random weights, pretrained networks, and unnormalized and normalized input spaces.

Figs. 9 and 10 visualize the changes in the mean correlation values during ProxyDR model training (averaged values from five different seeds). Surprisingly, we found that the prototypes of the approximately untrained networks (pretrained on ImageNet [3] and trained on the target dataset, e.g., CIFAR100 or NABirds, for only one epoch) already have relatively high correlations (approximately 0.4), with predefined hierarchical structures. While the accuracy curves show no noticeable differences, using the dynamic option modified the transition in the mean correlations. In particular,

prototype-based mean correlations increased after 20 to 40 training epochs, and the maximum values were obtained near the end of training (approximately 100 epochs). The proxy-based mean correlations started at low values, and the difference with the prototype-based mean correlations was reduced. Moreover, the training epoch during which the validation accuracy is maximized often differs from the training epoch during which the correlation measures are maximized.

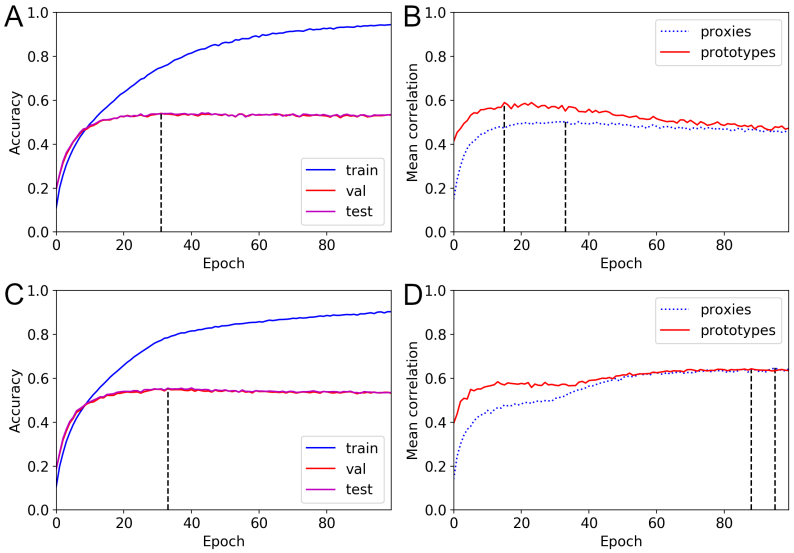


Fig 9. Changes in accuracy and mean correlations for the CIFAR100 dataset (ProxyDR). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option.

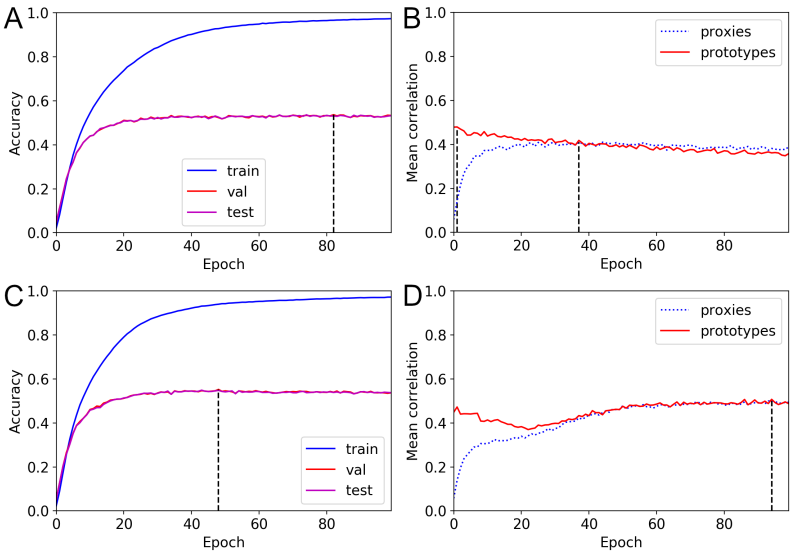


Fig 10. Changes in accuracy and mean correlations for the NABirds dataset (ProxyDR). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option.

Table 2 shows prototype-based mean correlation values and their 95 percent confidence intervals based on five different seeds. The mean correlations with the random networks show that the prototypes and ground-truth hierarchy are correlated. While these values are smaller than the other cases shown in the table, the results show that random networks have some degree of semantic understanding.

Table 2. Mean correlations based on prototypes.

Space	Random weights	Pretrained	Input space	Normalized input space
CIFAR100	0.2841 ± 0.0303	0.3816 ± 0.0116	0.3414 ± 0.0353	0.2609 ± 0.0416
NABirds	0.1088 ± 0.0839	0.2648 ± 0.0349	0.1740 ± 0.0056	0.2369 ± 0.0039

"Random weights" and "pretrained" mean embedding space based on the ResNet50 [30] backbone.

Discussion

In this work, we investigate classification and hierarchical performance under different models and training options. Our experiments reveal several important findings. Under the training options that do not consider predefined hierarchical information, the ProxyDR model achieved better hierarchical inference performance than NormFace in most cases. Furthermore, under the same training options, ProxyDR achieved better hierarchy-informed performance in terms of the AHD (k=5) and HP@5 measures. Moreover, we observed that the use of a dynamic scaling factor improved the hierarchical inference performance. The changes in the mean correlation values (Figs. 9

and 10) verified the effect of the dynamic training option. These results reveal the importance of dynamic training approach. We also found that some hierarchy-informed performance measures are not significantly improved by the use of known hierarchical structures. This finding indicates that multiple hierarchy-informed performance measures should be considered to compare the hierarchy-informed performance of different models. We also observed a trade-off between the hierarchy-informed performance and top-5 accuracy. While the CORR loss model typically achieved the best hierarchical performance, this model obtained the lowest top-5 accuracy among the experimental models. Similarly, the use of predefined hierarchical information in NormFace and ProxyDR significantly improved the AHD (k=5) and HP@5 performance but reduced the top-5 accuracy. In contrast to previous works that observed a trade-off between hierarchy-informed performance and top-1 accuracy [4], we did not observe this trade-off with top-1 accuracy.

Surprisingly, we found that prototypes based on CNNs with random weights showed correspondence with predefined hierarchies that was higher than random chance. Because CNNs combine convolutional and pooling layers, most CNN architectures have translation invariance properties. Because of such priors, random networks may know weak perceptual similarity [33]. Another possible reason for this result is that prototypes of input spaces show higher correspondence with predefined hierarchies than random chance. The Johnson-Lindenstrauss lemma [34] shows that linear projections using random matrices approximately preserve distances. If this property holds for nonlinear projections based on random neural networks, network-based prototypes may also correspond to predefined hierarchies. This phenomenon suggests that when we use metric learning models with proxies, the proxies can be assigned based on prototypes instead of starting at random positions. This may improve training during the initial epochs, as we start from proxies that are more semantically reasonable than random positions.

Although we observed that the DR formulation improves the hierarchical inference performance and hierarchy-informed performance when training models without predefined hierarchies, we did not study the reasons underlying these phenomena. We suggest one possible hypothesis. While NormFace prevents sudden changes in the scaling factor by using normalized embeddings, its loss function is based on the squared difference of the distance (as it uses the SD softmax formulation). As this loss can increase the squared difference of the distance among different proxies, the absolute distance between any pairs of proxies may be increased. This tendency can result in larger distances, even among semantically similar classes, and proxy positions that are less organized in terms of visual similarity. On the other hand, the DR formulation-based loss is based on distance ratios. Thus, there is no tendency to increase absolute distances among proxies, and proxies can be structured according to visual similarity. However, further investigations are needed to verify this hypothesis and reveal the underlying cause.

Conclusion

The hierarchy-informed performance must be improved to more broadly adopt classification models. We explored this concept with classification-based metric learning models in situations in which hierarchical information is and is not available during training. Our results show that when the class hierarchical relations are unknown, the ProxyDR model achieves the best hierarchical inference and hierarchy-informed performance. In contrast, with hierarchy-informed training, the CORR loss model achieves the best hierarchy-informed performance but the lowest top-5 accuracy on most datasets. Since some hierarchy-informed measures may not be improved by the use of

hierarchical information during training, multiple hierarchy-informed performance measures should be used to obtain appropriate comparisons. Additionally, our experiments reveal that during classification-based metric learning, initializing proxies based on prototypes may be beneficial.

Acknowledgments

Hyeongji Kim, Terje Berge, and Ketil Malde acknowledge the Ministry of Trade, Industry and Fisheries for financial support. The authors thank Gayantonia Franze, Hege Lyngvær Mathisen, Magnus Reeve, and Mona Ring Kleiven, from the Plankton Research Group at Institute of Marine Research (IMR), for their contributions to the plankton datasets.

References

1. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision; 2015. p. 1026–1034.
2. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. Imagenet large scale visual recognition challenge. International journal of computer vision. 2015;115(3):211–252.
3. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. Ieee; 2009. p. 248–255.
4. Bertinetto L, Mueller R, Tertikas K, Samangooei S, Lord NA. Making better mistakes: Leveraging class hierarchies with deep networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 12506–12515.
5. Barz B, Denzler J. Hierarchy-based image embeddings for semantic image retrieval. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE; 2019. p. 638–647.
6. Garnot VSF, Landrieu L. Leveraging Class Hierarchies with Metric-Guided Prototype Learning. In: British Machine Vision Conference (BMVC); 2021.
7. Jayathilaka M, Mu T, Sattler U. Ontology-based n-ball concept embeddings informing few-shot image classification. arXiv preprint arXiv:210909063. 2021;.
8. Musgrave K, Belongie S, Lim SN. A metric learning reality check. In: European Conference on Computer Vision. Springer; 2020. p. 681–699.
9. Snell J, Swersky K, Zemel R. Prototypical Networks for Few-shot Learning. Advances in Neural Information Processing Systems. 2017;30:4077–4087.
10. Chen WY, Liu YC, Kira Z, Wang YCF, Huang JB. A closer look at few-shot classification. arXiv preprint arXiv:190404232. 2019;.
11. Miller GA. WordNet: An electronic lexical database. MIT press; 1998.
12. Godbole S. Exploiting confusion matrices for automatic generation of topic hierarchies and scaling up multi-way classifiers. Annual Progress Report, Indian Institute of Technology–Bombay, India. 2002;.

13. Wang F, Xiang X, Cheng J, Yuille AL. Normface: L2 hypersphere embedding for face verification. In: Proceedings of the 25th ACM international conference on Multimedia; 2017. p. 1041–1049.

14. Kim S, Kim D, Cho M, Kwak S. Proxy anchor loss for deep metric learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 3238–3247.

15. Wang H, Wang Y, Zhou Z, Ji X, Gong D, Zhou J, et al. Cosface: Large margin cosine loss for deep face recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2018. p. 5265–5274.

16. Deng J, Guo J, Xue N, Zafeiriou S. Arcface: Additive angular margin loss for deep face recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 4690–4699.

17. Wan A, Dunlap L, Ho D, Yin J, Lee S, Petryk S, et al. {NBDT}: Neural-Backed Decision Tree. In: International Conference on Learning Representations; 2021.Available from: `https://openreview.net/forum?id=mCLVeEpplNE`.

18. Silla CN, Freitas AA. A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery. 2011;22(1):31–72.

19. Weinberger KQ, Blitzer J, Saul L. Distance metric learning for large margin nearest neighbor classification. Advances in neural information processing systems. 2005;18.

20. Hadsell R, Chopra S, LeCun Y. Dimensionality reduction by learning an invariant mapping. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2. IEEE; 2006. p. 1735–1742.

21. Zhe X, Ou-Yang L, Yan H. Improve L2-normalized Softmax with Exponential Moving Average. In: 2019 International Joint Conference on Neural Networks (IJCNN). IEEE; 2019. p. 1–7.

22. Zhang X, Zhao R, Qiao Y, Wang X, Li H. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 10823–10832.

23. Zhai A, Wu HY. Classification is a strong baseline for deep metric learning. British Machine Vision Conference (BMVC). 2019;.

24. Liu W, Wen Y, Yu Z, Li M, Raj B, Song L. Sphereface: Deep hypersphere embedding for face recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2017. p. 212–220.

25. Kim H, Parviainen P, Malde K. Distance-Ratio-Based Formulation for Metric Learning. arXiv preprint arXiv:220108676. 2022;.

26. Krizhevsky A. Learning multiple layers of features from tiny images; 2009.

27. Van Horn G, Branson S, Farrell R, Haber S, Barry J, Ipeirotis P, et al. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015. p. 595–604.

28. Movshovitz-Attias Y, Toshev A, Leung TK, Ioffe S, Singh S. No fuss distance metric learning using proxies. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 360–368.

29. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 2818–2826.

30. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770–778.

31. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014;.

32. Frome A, Corrado GS, Shlens J, Bengio S, Dean J, Ranzato M, et al. Devise: A deep visual-semantic embedding model. Advances in neural information processing systems. 2013;26.

33. Zhang R, Isola P, Efros AA, Shechtman E, Wang O. The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2018. p. 586–595.

34. Johnson W, Lindenstrauss J. Extensions of Lipschitz mappings into a Hilbert space. Contemporary Mathematics. 1984;26:189–206.

Supporting information

S1 Appendix

Detailed explanation of the dynamic (adaptive) scaling factors in the NormFace and ProxyDR models.

Zhang et al. [22] rewrote Eq. 5 as follows:

$$p(c|x) = \frac{\exp(s\cos\theta_c)}{\exp(s\cos\theta_c) + B_x},$$

where c is the corresponding class of point x and $B_x = \sum_{y\neq c, y\in\mathcal{Y}} \exp(s\cos\theta_y)$. They found that θ_y was close to $\frac{\pi}{2}$ during the training process for $y \neq c$, i.e., for different classes. Thus, $B_x \approx \sum_{y\neq c, y\in\mathcal{Y}} \exp(s\cos\frac{\pi}{2}) = \sum_{y\neq c, y\in\mathcal{Y}} \exp(0) = |\mathcal{Y}| - 1$.

In Eq. 10, $\frac{\partial^2 p(c|x)(\theta_c)}{\partial\theta_c^2}$ can be written as:

$$\frac{\partial^2 p(c|x)(\theta_c)}{\partial\theta_c^2} = \frac{-sB_x\exp(s\cos\theta_c)\psi_{NormFace}(s,\theta_c)}{(\exp(s\cos\theta_c) + B_x)^3},$$

where $\psi_{NormFace}(s,\theta_c) = \cos\theta_c\,(\exp(s\cos\theta_c) + B_x) + s\sin^2\theta_c\,(\exp(s\cos\theta_c) - B_x)$. Moreover, they used $s = \frac{\log B_x}{\cos\theta_c}$ to approximate the solution for Eq. 10. For the static version, they used $|\mathcal{Y}| - 1$ to estimate B_x and $\frac{\pi}{4}$ to estimate θ_c. For the dynamic version, they used $B_{x;avg}$ to estimate B_x and $\theta_{c;med}$ to estimate θ_c, where $B_{x;avg}$ is the average of B_x in a mini-batch and $\theta_{c;med}$ is the median of the θ_c values in a mini-batch. They clipped the $\theta_{c;med}$ value to be in the range $\left[0, \frac{\pi}{4}\right]$.

Instead of using $s = \frac{\log B_x}{\cos\theta_c}$, in our implementation, we use the Adam optimizer [31] to update a scale factor s that minimizes $\psi^2_{NormFace}(s,\theta_c)$, i.e., $\psi_{NormFace}(s,\theta_c) \approx 0$. For $\psi_{NormFace}(s,\theta_c)$, we use $|\mathcal{Y}| - 1$ to estimate B_x and $\frac{\pi}{4}$ to estimate θ_c to initialize the value of s. During model training, we use $B_{x;avg}$ to estimate B_x and $\theta_{c;med}$ to estimate θ_c.

We can also apply the dynamic scaling factor to the ProxyDR model. First, we define $d_{x,y} := \left\|\tilde{f}(x) - \tilde{W}_y\right\|$. We rewrite Eq. 8 as:

$$p(c|x) = \frac{d_{x,c}^{-s}}{d_{x,c}^{-s} + B_x},$$

where $B_x = \sum_{y\neq c, y\in\mathcal{Y}} d_{x,y}^{-s}$. Assuming $\theta_y \approx \frac{\pi}{2}$ for $y \neq c$, we obtain $B_x \approx \sum_{y\neq c, y\in\mathcal{Y}} \left(\frac{\pi}{2}\right)^{-s} = (|\mathcal{Y}| - 1)\left(\frac{\pi}{2}\right)^{-s}$.

The expression $\frac{\partial^2 p(c|x)(\theta_c)}{\partial\theta_c^2}$ can be written as:

$$\frac{\partial^2 p(c|x)(\theta_c)}{\partial\theta_c^2} = \frac{B_x s\theta_c^{(s-2)}\psi_{ProxyDR}(s,\theta_c)}{(B_x\theta_c^s + 1)^3}$$

where $\psi_{ProxyDR}(s,\theta_c) = B_x(s+1)\theta_c^s - s + 1$. We then use the Adam optimizer to update a scale factor s that minimizes $\psi^2_{ProxyDR}(s,\theta_c)$.

S2 Appendix

Dataset details.

All plankton images were obtained using FlowCam (Yokogawa Fluid Imaging Technologies). FlowCam is a flow imaging microscope that captures particles flowing through glass flowcells with well-defined volumes. The three plankton datasets were obtained according to different types of samples (live and Lugol fixed whole seawater or 180 μm WP2 plankton net samples) using three different FlowCams (FlowCam 8400, FlowCam VS, and FlowCam Macro) with various magnifications. Thus, the datasets include particles ranging from 5 to 2000 μm in size, thus representing nano-, micro-, and mesozooplankton. The plankton samples were obtained from three coastal monitoring stations (Institute of Marine Research) along the Norwegian coast, including Holmfjord in the north, Austevoll in the west and Torungen in the south. In addition, for the nano- and microplankton, seawater samples were obtained from a tidal zone at a depth of 1 meter at the research station at Flødevigen in southern Norway, which is approximately 2 nautical miles from the southern monitoring station at Torungen. The sampling period for the three datasets covered all seasons over a period of approximately 2.5 years.

Small microplankton (MicroS) This dataset contains images of fixed and live seawater samples acquired at a depth of 5 m at the three monitoring stations and a depth of 1 m in the tidal zone (see above). The seawater samples were carefully filtered through a 80 μm mesh to ensure that 100 μm flowcell was not clogged and imaged using a 10× objective. This FlowCam configuration results in a total magnification of 100× and images particles ranging from 5 to 50 μm. Before resizing, one pixel in an image represented 0.7330 μm.

Large microplankton (MicroL) This dataset contains images of fixed and live seawater samples acquired at a depth of 5 m at the three monitoring stations and a depth of 1 m in the tidal zone (see above). The seawater samples were not filtered and were imaged using a 2× objective, targeting 35 to 500 μm particles. Before resizing, one pixel in an image represented 2.9730 μm. Due to instrument repair and adjustments to improve image quality, the camera settings were modified during the 3 years of imaging to acquire this dataset. Therefore, the image appearance and quality are slightly variable.

Mesozooplankton (MesoZ) This dataset contains images of mesozooplankton samples acquired at the three coastal monitoring stations (see above) and a transect in the Norwegian Sea (Svinøysnittet). The samples were obtained using an IMR (Institute of Marine Research) standard plankton net (WP2) or a multinet mammoth (both 180 μm mesh) and fixed with 4% formaldehyde. The images were acquired by two FlowCam instruments (one in Bergen and one in Flødevigen), and the image appearance differs slightly between the two instruments. The FlowCam macro was equipped with a 0.5× objective, resulting in a total magnification of 12.5 and imaging organisms ranging from 180 to 2000 μm. Before resizing, one pixel in an image represented 9.05 μm. All images in the mesozooplankton dataset are in grayscale.

NABirds Data provided by the Cornell Lab of Ornithology, with thanks to photographers and contributors of crowdsourced data at `AllAboutBirds.org/Labs`. This material is based upon work supported by the National Science Foundation under Grant No. 1010818.

Hierarchical structures of the datasets Figs. S1, S2, and S3 and Table S1 show the hierarchical structures of the datasets. As the NABirds dataset contains too many (555) classes to visualize, we do not show the hierarchical structures of the NABirds dataset [27]. We used the hierarchy provided by the Cornell Lab of Ornithology.

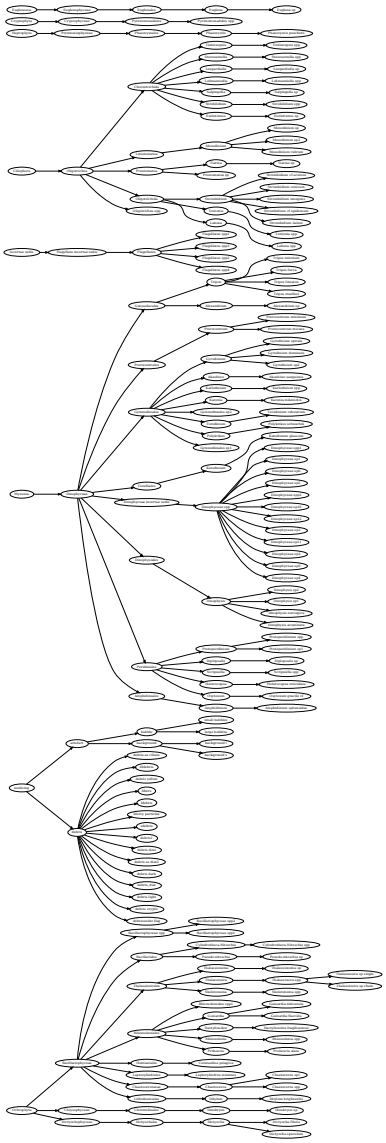


Fig S1. The hierarchical structure of the MicroS dataset used in our experiment. Best viewed by zooming in.

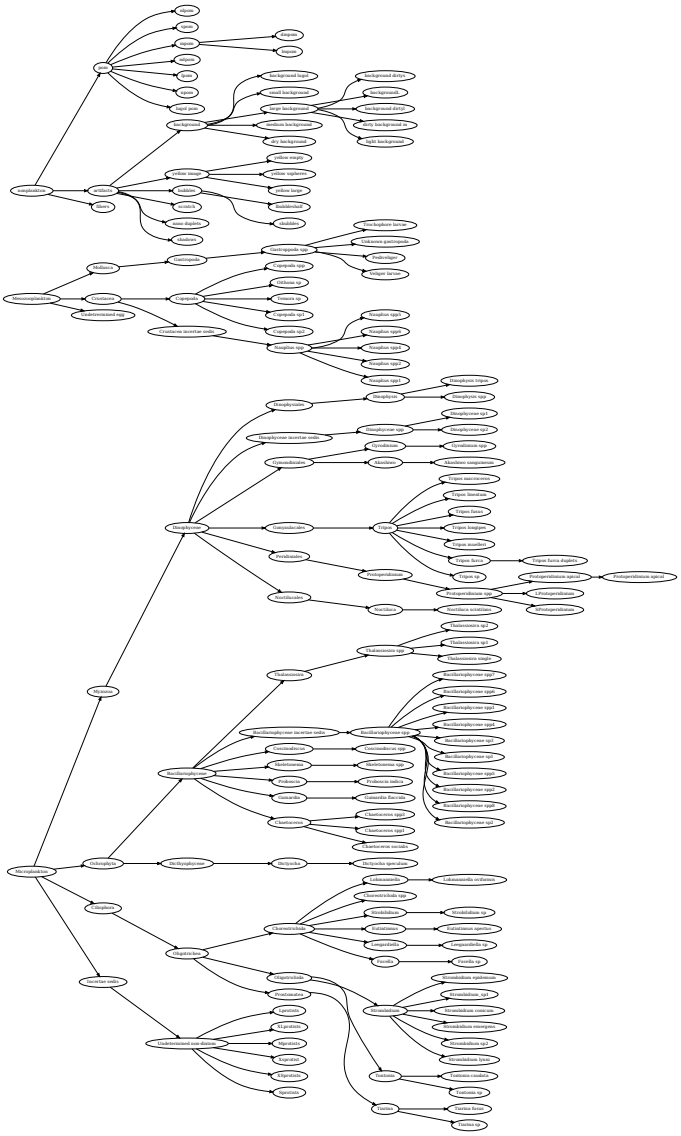


Fig S2. The hierarchical structure of the MicroL dataset used in our experiment. Best viewed by zooming in.

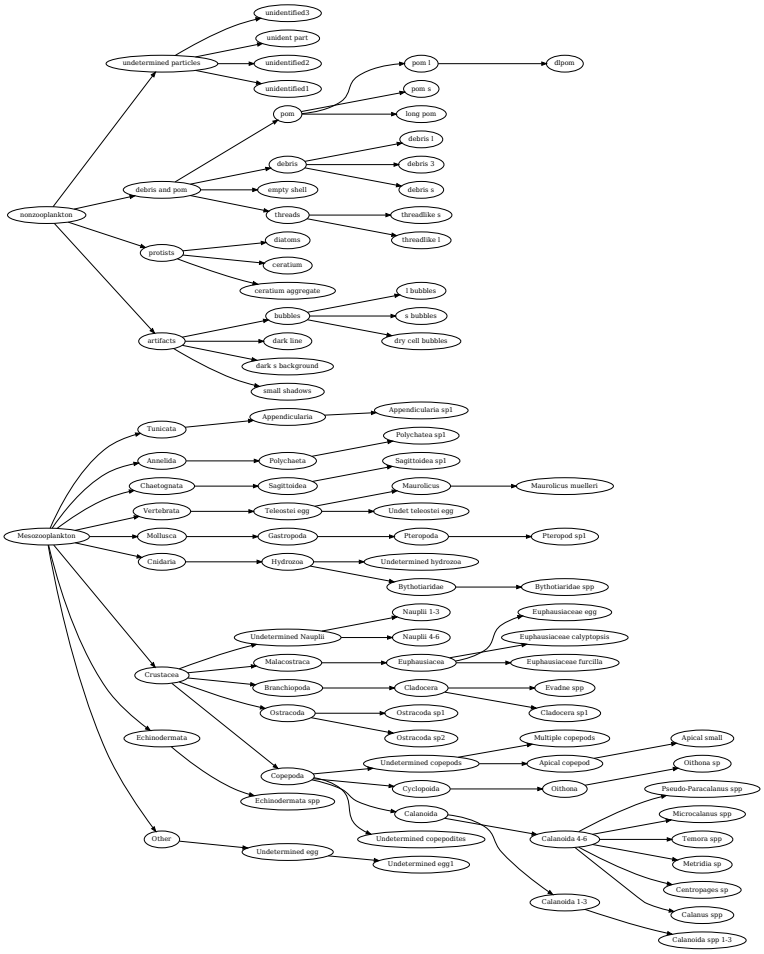


Fig S3. The hierarchical structure of the MesoZ dataset used in our experiment. Best viewed by zooming in.

Table S1. The hierarchical structure of the CIFAR100 dataset used in our experiment.

Level 0	Level 1	Level 2	Level 3 (class level)
Animals	Invertebrates	insects	bee, beetle, butterfly, caterpillar, cockroach
		non-insect invertebrates	crab, lobster, snail, spider, worm
	Mammals	aquatic mammals	beaver, dolphin, otter, seal, whale
		large carnivores	bear, leopard, lion, tiger, wolf
		large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
		medium-sized mammals	fox, porcupine, possum, raccoon, skunk
		people	baby, boy, girl, man, woman
		small mammals	hamster, mouse, rabbit, shrew, squirrel
	Non-mammal vertebrates	fish	aquarium fish, flatfish, ray, shark, trout
		reptiles	crocodile, dinosaur, lizard, snake, turtle
Artificial	Artificial (indoor)	food containers	bottles, bowls, cans, cups, plates
		household electrical devices	clock, computer keyboard, lamp, telephone, television
		household furniture	bed, chair, couch, table, wardrobe
	Artificial (outdoor)	large man-made outdoor things	bridge, castle, house, road, skyscraper
		vehicles 1	bicycle, bus, motorcycle, pickup truck, train
		vehicles 2	lawn-mower, rocket, streetcar, tank, tractor
Nature (non-animal)	Nature (non-specific organism)	large natural outdoor scenes	cloud, forest, mountain, plain, sea
	Plants	flowers	orchids, poppies, roses, sunflowers, tulips
		fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
		trees	maple, oak, palm, pine, willow

S3 Appendix

Results for all five datasets.

Figs. S4, S5, S6, S7, and S8 show the top-k accuracies on the different datasets.

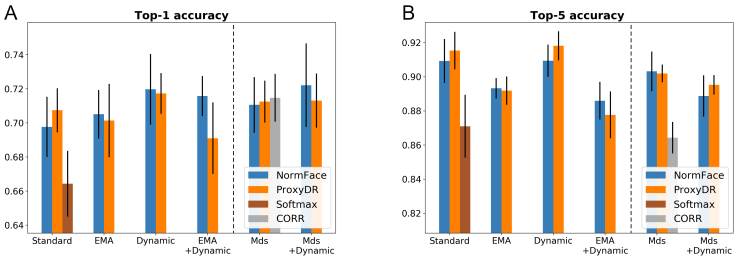


Fig S4. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the MicroS dataset.

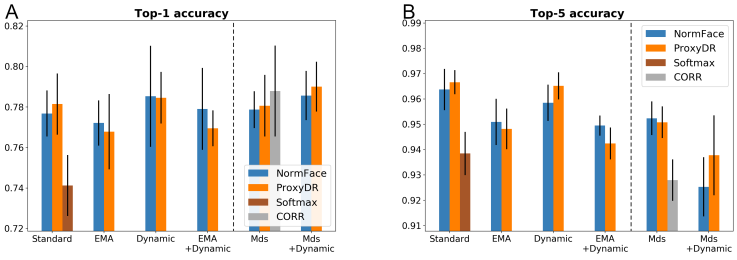


Fig S5. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the MicroL dataset.

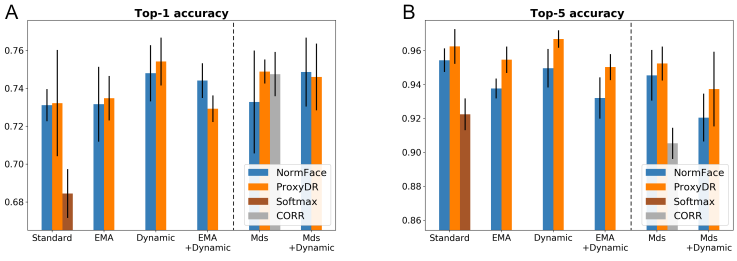


Fig S6. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the MesoZ dataset.

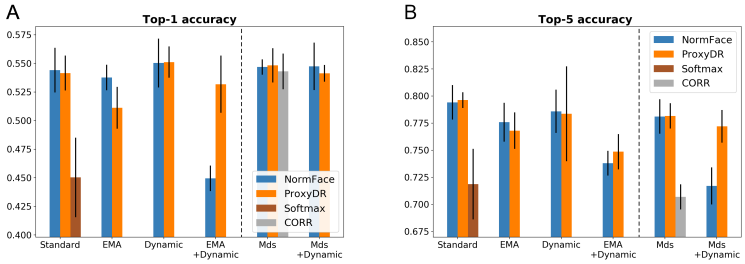


Fig S7. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the CIFAR100 dataset.

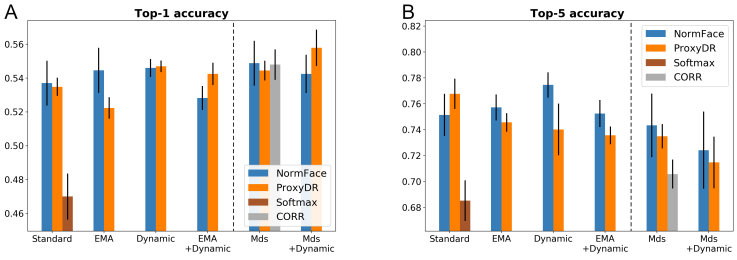


Fig S8. Top-k accuracy results (A: $k = 1$, B: $k = 5$) on the NABirds dataset.

Figs. S9, S10, S11, S12, and S13 show the mean correlation values on the different datasets.

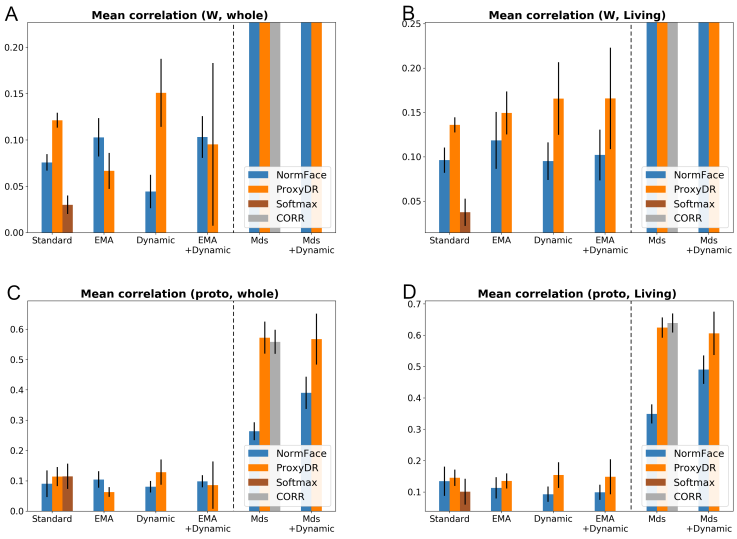


Fig S9. Correlation measures on the MicroS dataset. (Top) Values using proxies (A: whole classes, B: living classes). (Bottom) Values using prototypes (C: whole classes, D: living classes). 'Living' indicates that only biological classes were used (no nonliving classes). The mean correlation values based on proxies with the MDS option were 0.9306 (whole) and 0.9011 (living).

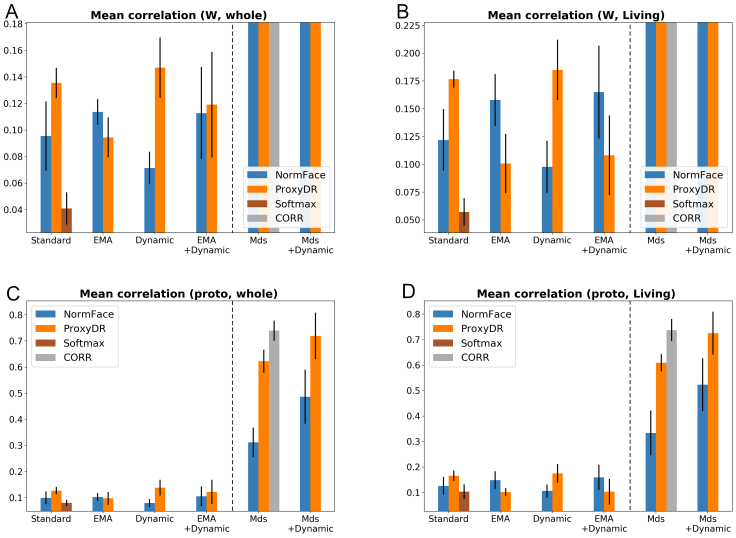


Fig S10. Correlation measures on the MicroL dataset. (Top) Values using proxies (A: whole classes, B: living classes). (Bottom) Values using prototypes (C: whole classes, D: living classes). The mean correlation values based on proxies with the MDS option were 0.9543 (whole) and 0.9426 (living).

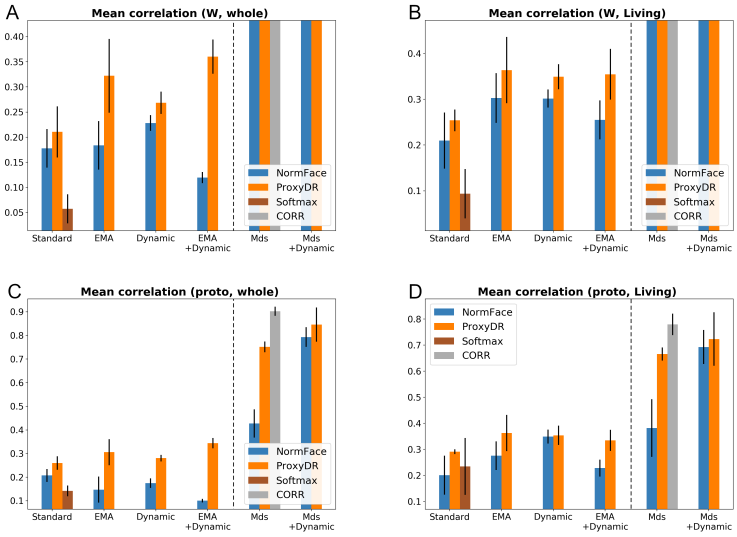


Fig S11. Correlation measures on the MesoZ dataset. (Top) Values using proxies (A: whole classes, B: living classes). (Bottom) Values using prototypes (C: whole classes, D: living classes). The mean correlation values based on proxies with the MDS option were 0.9783 (whole) and 0.9602 (living).

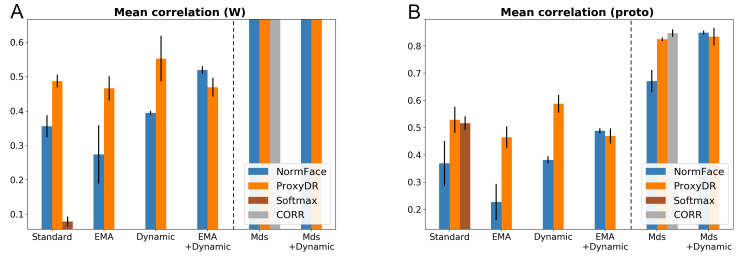


Fig S12. Correlation measures on the CIFAR100 dataset. (A) Values using proxies. (B) Values using prototypes. The mean correlation value based on proxies with the MDS option was 0.8580.

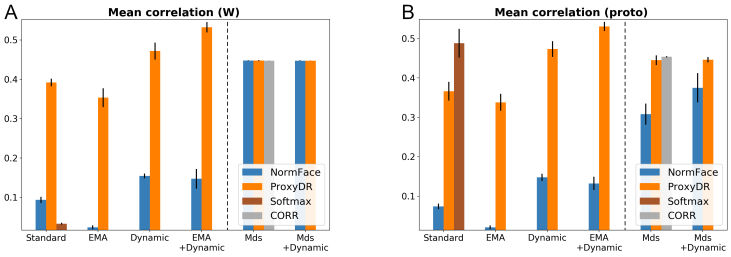


Fig S13. Correlation measures on the NABirds dataset. (A) Values using proxies. (B) Values using prototypes. The mean correlation value based on proxies with the MDS option was 0.4476 (this value is small as the dataset contains 555 classes and the embedding dimension is 128.).

Figs. S14, S15, S16, S17, and S18 show the hierarchy-informed performance measures on the different datasets.

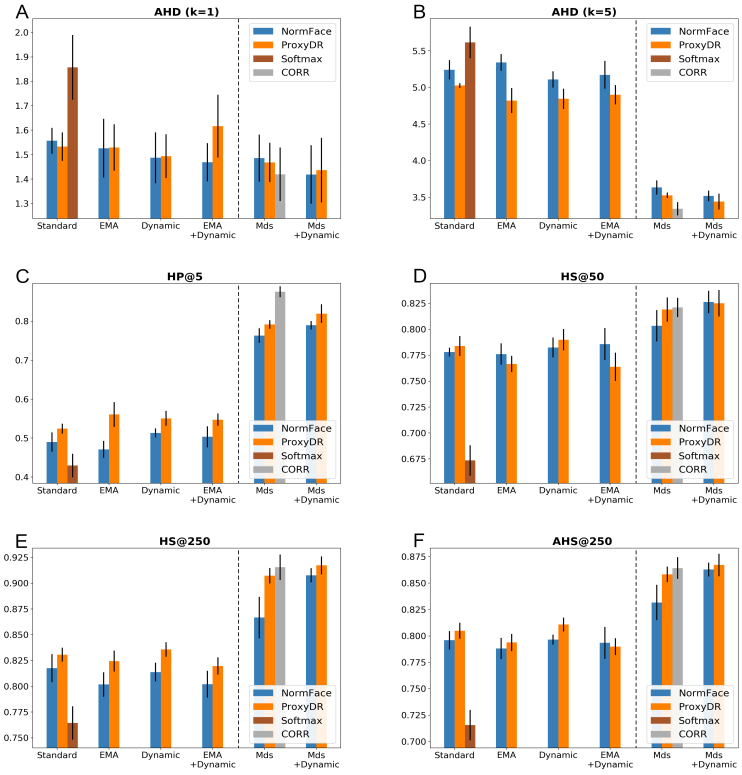


Fig S14. Hierarchical performance measures on the MicroS dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

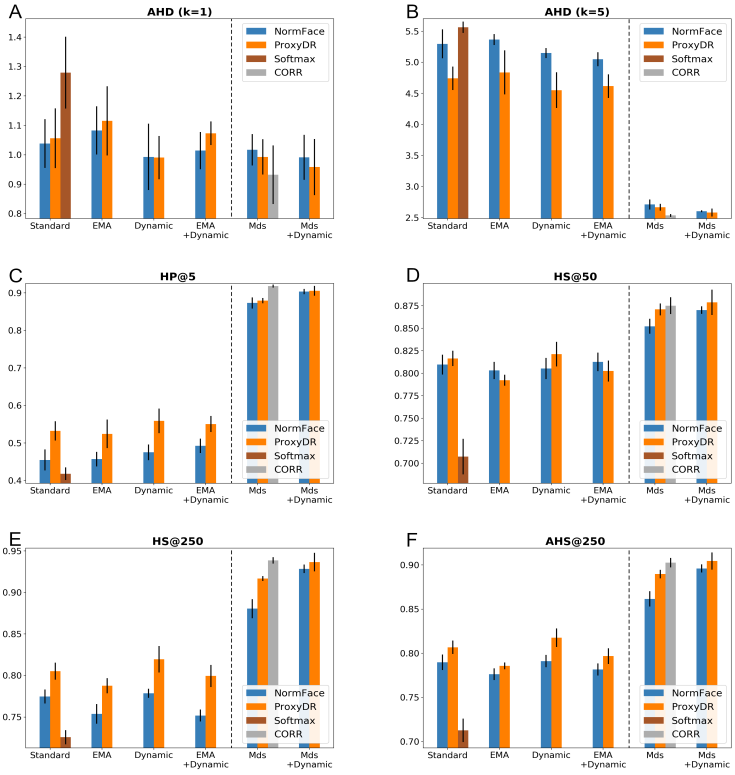


Fig S15. Hierarchical performance measures on the MicroL dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

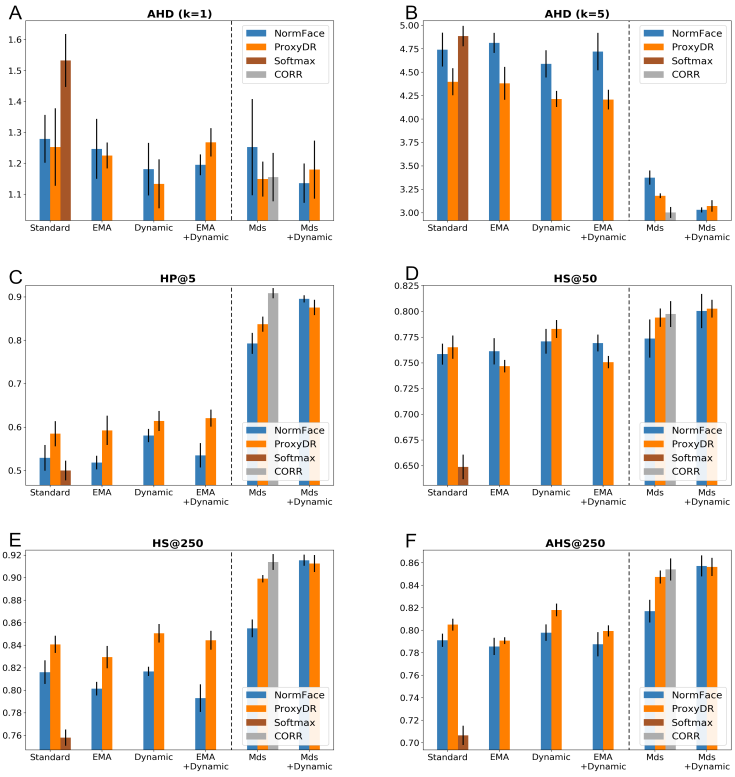


Fig S16. Hierarchical performance measures on the MesoZ dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

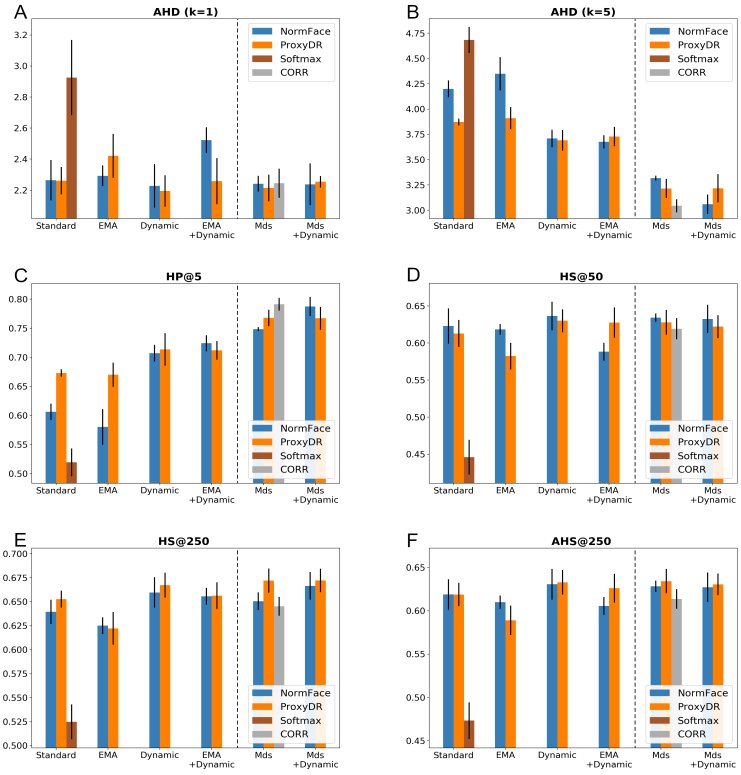


Fig S17. Hierarchical performance measures on the CIFAR100 dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

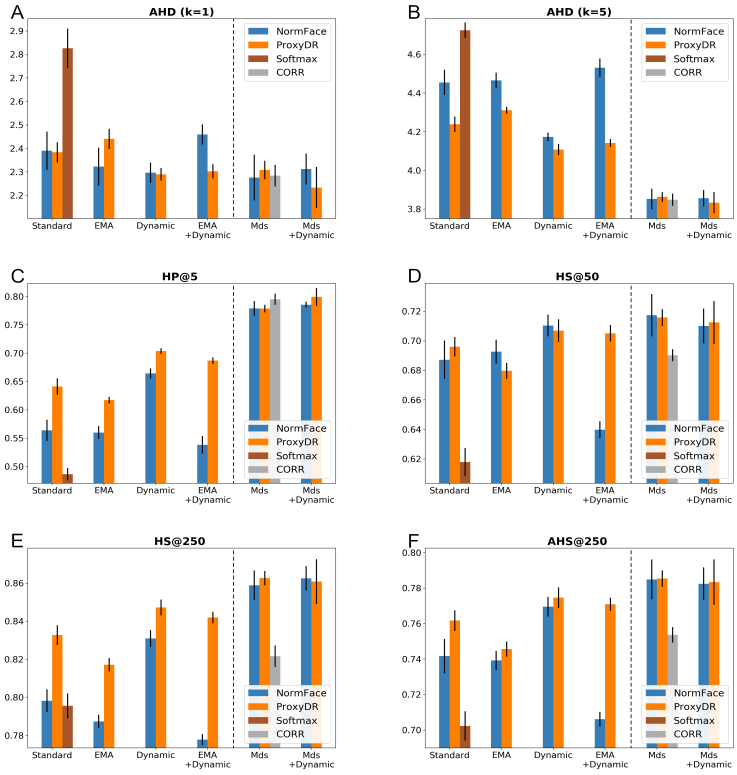


Fig S18. Hierarchical performance measures on the NABirds dataset. The symbol ↓ denotes that lower values indicate better performance. The symbol ↑ denotes that higher values indicate better performance. (A) AHD (k=1): ↓. (B) AHD (k=5): ↓. (C) HP@5: ↑. (D) HS@50: ↑. (E) HS@250: ↑. (F) AHS@250: ↑.

Additional results on mean correlations

Figs. S19, S20, S21, S22, S23 and S24 visualize the changes in the mean correlation values with different models and training options, showing the averaged values for five different seeds. Figs. S19 and S22 show that the accuracy of the NormFace models decreased after a certain number of epochs when the models were trained with the dynamic option. Interestingly, the mean correlation values increased even when the accuracy decreased. When using predefined hierarchical information and dynamic options (if applicable), the prototype-baesd mean correlation values approached the fixed proxy-based mean correlation values, except the NormFace model on the NABirds dataset.

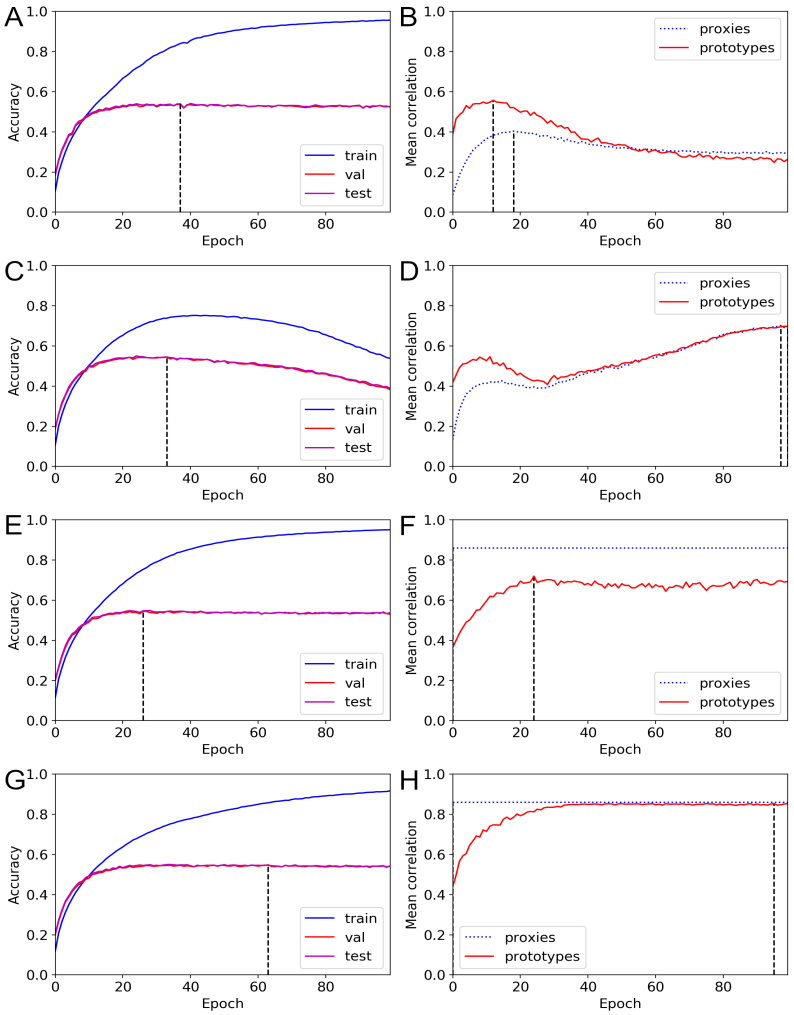


Fig S19. Changes in accuracy and mean correlations for the CIFAR100 dataset (NormFace). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option. (E) Accuracy curve with the MDS option. (F) Mean correlation curve with the MDS option. (G) Accuracy curve with the MDS & dynamic options. (H) Mean correlation curve with the MDS & dynamic options.

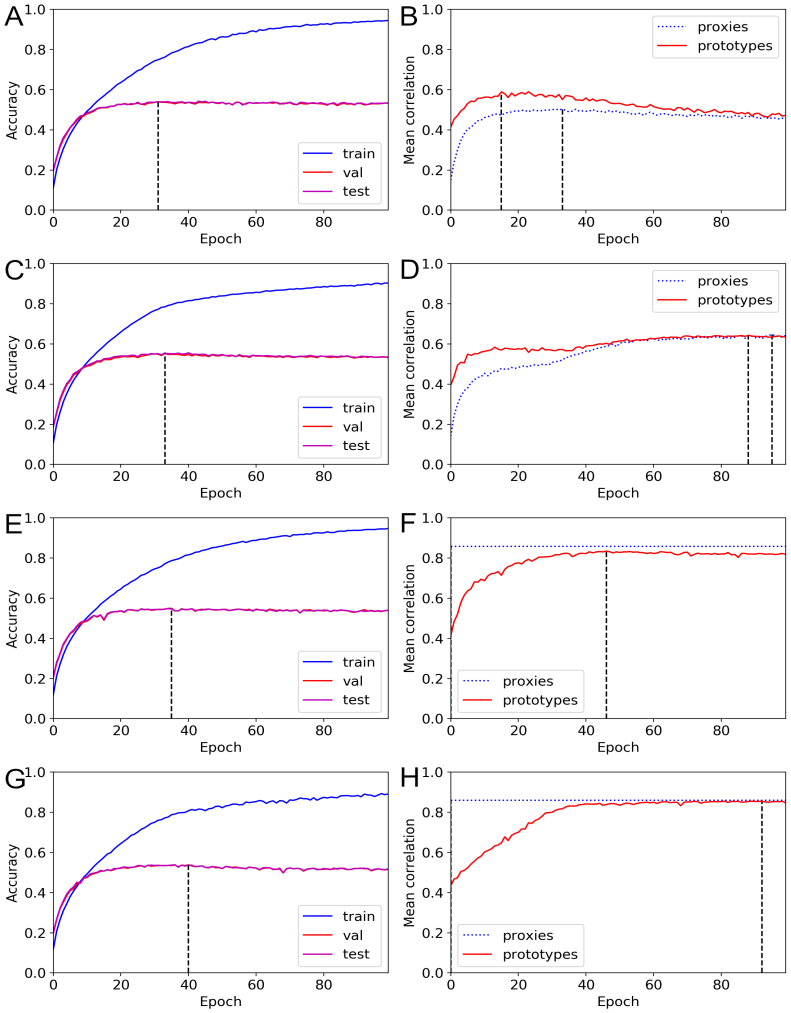


Fig S20. Changes in accuracy and mean correlations for the CIFAR100 dataset (ProxyDR). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option. (E) Accuracy curve with the MDS option. (F) Mean correlation curve with the MDS option. (G) Accuracy curve with the MDS & dynamic options. (H) Mean correlation curve with the MDS & dynamic options.

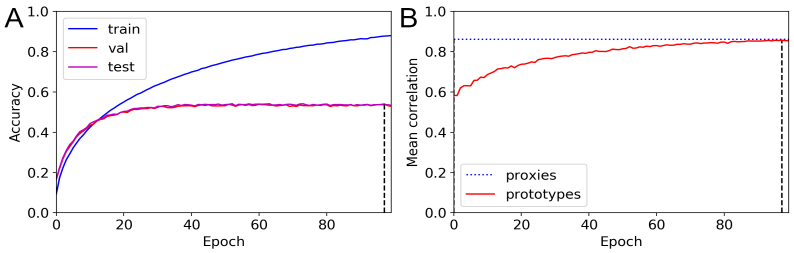


Fig S21. Changes in accuracy and mean correlations for the CIFAR100 dataset (CORR). (A) Accuracy curve with CORR loss training. (B) Mean correlation curve with CORR loss training.

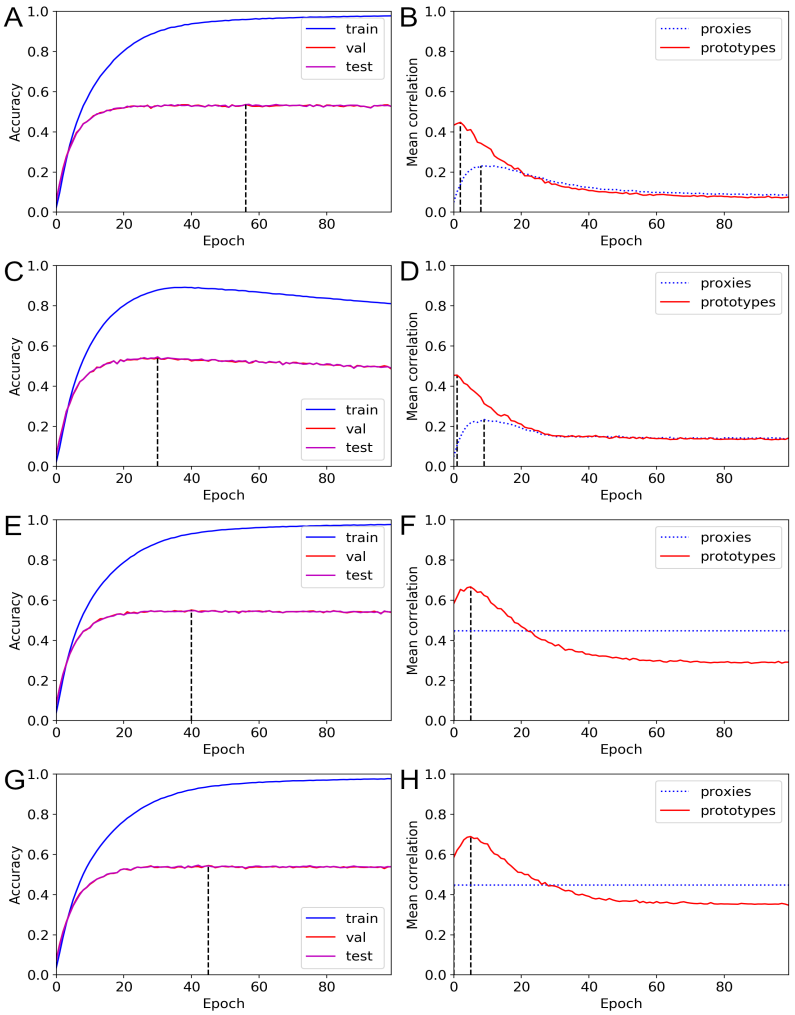


Fig S22. Changes in accuracy and mean correlations for the NABirds dataset (NormFace). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option. (E) Accuracy curve with the MDS option. (F) Mean correlation curve with the MDS option. (G) Accuracy curve with the MDS & dynamic options. (H) Mean correlation curve with the MDS & dynamic options.

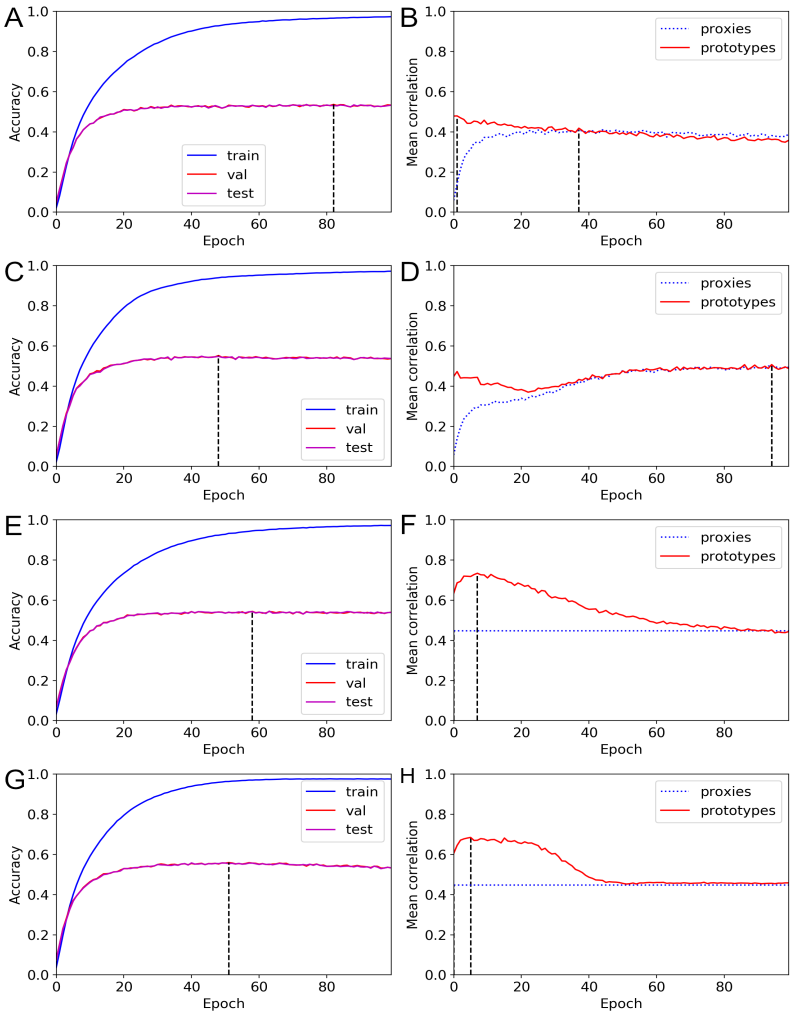


Fig S23. Changes in accuracy and mean correlations for the NABirds dataset (ProxyDR). (A) Accuracy curve with standard training. (B) Mean correlation curve with standard training. (C) Accuracy curve with the dynamic option. (D) Mean correlation curve with the dynamic option. (E) Accuracy curve with the MDS option. (F) Mean correlation curve with the MDS option. (G) Accuracy curve with the MDS & dynamic options. (H) Mean correlation curve with the MDS & dynamic options.

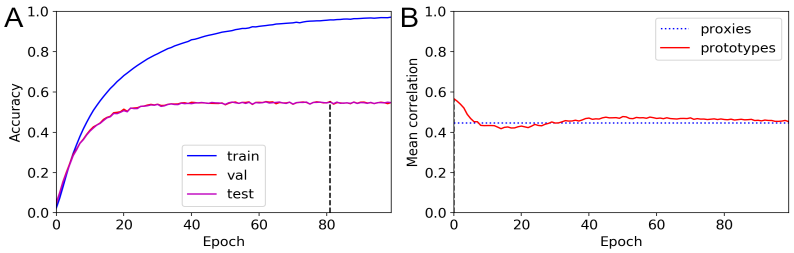


Fig S24. Changes in accuracy and mean correlations for the NABirds dataset (CORR). (A) Accuracy curve with CORR loss training. (B) Mean correlation curve with CORR loss training.

Rank correlations between performance measures We investigate whether different performance measures are correlated. While we cannot derive causal relationships, this investigation is useful for determining if there are trade-offs between various performance measures. We used rank correlation coefficients between the mean correlation values and other measures. We ignored the softmax loss results, as this model is not based on metric learning.

In Table S2, we investigate whether mean correlation values based on proxies (Figs. S9, S10, S11, S12, and S13) are correlated with other performance measures (Figs. S4, S5, S6, S7, S8, S14, S15, S16, S17, and S18). The table shows that the AHD (k=5), HP@5, HS@250, and AHS@250 measures are correlated with the proxy-based mean correlation values. Moreover, there does not appear to be any trade-off between the top-1 accuracies and proxy-based mean correlations. However, a trade-off between the top-5 accuracy and proxy-based mean correlations is observed on some datasets.

Table S2. Rank (Spearman) correlation coefficients of different measures with mean correlations (based on proxies)

Measure	MicroS	MicroL	MesoZ	CIFAR100	NABirds
Top-1	0.2039 (1.0336×10^{-1})	0.2260 (7.0284×10^{-2})	0.2171 (8.2377×10^{-2})	0.1675 (1.8239×10^{-1})	0.3112 (1.1627×10^{-2})
Top-5	-0.1350 (2.8361×10^{-1})	**-0.3716** (**2.3043×10^{-3}**)	-0.2120 (9.0045×10^{-2})	**-0.3364** (**6.1476×10^{-3}**)	**-0.5669** (**8.4964×10^{-7}**)
AHD (k=1)	**-0.3423** (**5.2583×10^{-3}**)	**-0.3855** (**1.5179×10^{-3}**)	-0.2959 (1.6684×10^{-2})	-0.1821 (1.4658×10^{-1})	**-0.4267** (**3.9270×10^{-4}**)
AHD (k=5)	**-0.7407** (**1.7522×10^{-12}**)	**-0.8258** (**2.5540×10^{-17}**)	**-0.9026** (**9.6705×10^{-25}**)	**-0.8856** (**1.1648×10^{-22}**)	**-0.7023** (**7.0541×10^{-11}**)
HP@5	**0.7392** (**2.0489×10^{-12}**)	**0.8350** (**5.4193×10^{-18}**)	**0.9001** (**2.0612×10^{-24}**)	**0.8955** (**7.8000×10^{-24}**)	**0.7070** (**4.6412×10^{-11}**)
HS@50	**0.7671** (**9.2047×10^{-14}**)	**0.7962** (**2.2284×10^{-15}**)	**0.5634** (**1.0275×10^{-6}**)	0.1878 (1.3410×10^{-1})	**0.5276** (**6.3074×10^{-6}**)
HS@250	**0.7911** (**4.4738×10^{-15}**)	**0.8167** (**1.1029×10^{-16}**)	**0.8860** (**1.0514×10^{-22}**)	**0.5514** (**1.9333×10^{-6}**)	**0.7210** (**1.2570×10^{-11}**)
AHS@250	**0.7950** (**2.6376×10^{-15}**)	**0.8186** (**8.1611×10^{-17}**)	**0.8034** (**8.1089×10^{-16}**)	**0.3335** (**6.6355×10^{-3}**)	**0.6862** (**2.8150×10^{-10}**)

p values are written in parentheses. Significant results (p value < 0.01) are written in bold text.

In Table S3, we investigate whether the top-1 accuracy (Figs. S4, S5, S6, S7, and S8) is correlated with other performance measures (Figs. S4, S5, S6, S7, S8, S14, S15, S16, S17, and S18). The table shows that the AHD (k=1), HS@50, and AHS@250 measures are correlated with the top-1 accuracy. Note that these are the measures that did not show noticeable changes on some datasets when predefined hierarchical information was used during training.

Table S3. Rank (Spearman) correlation coefficients of different measures with the top-1 accuracy

Measure	MicroS	MicroL	MesoZ	CIFAR100	NABirds
Top-5	0.2199 (7.8357×10^{-2})	0.0365 (7.7302×10^{-1})	0.1033 (4.1302×10^{-1})	**0.4398** (**2.4659×10^{-4}**)	-0.1043 (4.0815×10^{-1})
AHD (k=1)	**-0.8005** (**1.2168×10^{-15}**)	**-0.8302** (**1.2369×10^{-17}**)	**-0.9309** (**2.9921×10^{-29}**)	**-0.9507** (**9.4257×10^{-34}**)	**-0.9509** (**8.6632×10^{-34}**)
AHD (k=5)	-0.1906 (1.2829×10^{-1})	**-0.3259** (**8.0758×10^{-3}**)	-0.2908 (1.8752×10^{-2})	-0.2877 (2.0115×10^{-2})	**-0.6136** (**5.4992×10^{-8}**)
HP@5	0.2263 (6.9923×10^{-2})	0.2956 (1.6811×10^{-2})	0.2886 (1.9712×10^{-2})	0.2439 (5.0222×10^{-2})	**0.6029** (**1.0692×10^{-7}**)
HS@50	**0.4325** (**3.2041×10^{-4}**)	**0.5085** (**1.5301×10^{-5}**)	**0.6979** (**1.0420×10^{-10}**)	**0.8967** (**5.5302×10^{-24}**)	**0.6832** (**3.6174×10^{-10}**)
HS@250	0.1369 (2.7674×10^{-1})	0.3057 (1.3279×10^{-2})	0.2945 (1.7269×10^{-2})	**0.5535** (**1.7312×10^{-6}**)	**0.4915** (**3.2203×10^{-5}**)
AHS@250	**0.3394** (**5.6730×10^{-3}**)	**0.4233** (**4.4195×10^{-4}**)	**0.5067** (**1.6583×10^{-5}**)	**0.8517** (**2.4413×10^{-19}**)	**0.5733** (**6.0057×10^{-7}**)
Mean correlation (proxy)	0.2039 (1.0336×10^{-1})	0.2260 (7.0284×10^{-2})	0.2171 (8.2377×10^{-2})	0.1675 (1.8239×10^{-1})	0.3112 (1.1627×10^{-2})
Mean correlation (prototype)	0.2150 (8.5503×10^{-2})	0.2055 (1.0051×10^{-1})	0.1738 (1.6616×10^{-1})	0.1978 (1.1423×10^{-1})	0.2664 (3.1936×10^{-2})

p values are written in parentheses. Significant results (p value < 0.01) are written in bold text.

In Table S4, we investigate whether the top-5 accuracy (Figs. S4, S5, S6, S7, and S8) is correlated with other performance measures (Figs. S4, S5, S6, S7, S8, S14, S15, S16, S17, and S18). The table shows that no measures are correlated with the top-5 accuracy on all datasets.

Table S4. Rank (Spearman) correlation coefficients of different measures with the top-5 accuracy

Measure	MicroS	MicroL	MesoZ	CIFAR100	NABirds
Top-1	0.2199 (7.8357×10^{-2})	0.0365 (7.7302×10^{-1})	0.1033 (4.1302×10^{-1})	**0.4398** (**2.4659×10^{-4}**)	-0.1043 (4.0815×10^{-1})
AHD (k=1)	-0.0716 (5.7106×10^{-1})	0.0033 (9.7911×10^{-1})	-0.0406 (7.4809×10^{-1})	**-0.4245** (**4.2428×10^{-4}**)	0.1196 (3.4268×10^{-1})
AHD (k=5)	0.2417 (5.2436×10^{-2})	**0.4842** (**4.3828×10^{-5}**)	0.2732 (2.7654×10^{-2})	**0.4561** (**1.3422×10^{-4}**)	**0.4949** (**2.7791×10^{-5}**)
HP@5	-0.2765 (2.5757×10^{-2})	**-0.5318** (**5.1600×10^{-6}**)	-0.3113 (1.1590×10^{-2})	**-0.4902** (**3.3964×10^{-5}**)	**-0.5733** (**5.9953×10^{-7}**)
HS@50	0.0115 (9.2784×10^{-1})	-0.3164 (1.0231×10^{-2})	-0.2331 (6.1677×10^{-2})	0.2515 (4.3295×10^{-2})	-0.0071 (9.5522×10^{-1})
HS@250	-0.0677 (5.9231×10^{-1})	**-0.4343** (**3.0034×10^{-4}**)	-0.1965 (1.1677×10^{-1})	0.0955 (4.4900×10^{-1})	**-0.3192** (**9.5513×10^{-3}**)
AHS@250	0.0249 (8.4393×10^{-1})	**-0.3603** (**3.1949×10^{-3}**)	-0.1446 (2.5059×10^{-1})	0.3157 (1.0404×10^{-2})	-0.2203 (7.7867×10^{-2})
Mean correlation (proxy)	-0.1350 (2.8361×10^{-1})	**-0.3716** (**2.3043×10^{-3}**)	-0.2120 (9.0045×10^{-2})	**-0.3364** (**6.1476×10^{-3}**)	**-0.5669** (**8.4964×10^{-7}**)
Mean correlation (prototype)	-0.1040 (4.0985×10^{-1})	**-0.4784** (**5.5679×10^{-5}**)	-0.2910 (1.8677×10^{-2})	**-0.3675** (**2.5967×10^{-3}**)	**-0.5457** (**2.5927×10^{-6}**)

p values are written in parentheses. Significant results (p value < 0.01) are written in bold text.

Appendix

A The convex hull of the union of l_1 and l_∞ balls and its relation to a combined distance d_{combined} based ball

Following Croce and Hein [2020a], let us consider l_1 and l_∞ balls centered at the origin. We denote them as $\mathbb{B}_1$ and $\mathbb{B}_\infty$, respectively. The balls have radii $\epsilon_1 > 0$ and $\epsilon_\infty > 0$, respectively. These radii satisfy $\epsilon_1 \in (\epsilon_\infty, d_I \epsilon_\infty)$ where $d_I \geq 2$ is the dimension of (input) space. Namely, the balls satisfy $\mathbb{B}_1 \not\subset \mathbb{B}_\infty$ and $\mathbb{B}_\infty \not\subset \mathbb{B}_1$. Mathematically, $\mathbb{B}_1 = \left\{x \in \mathbb{R}^{d_I} : \|x\|_1 \leq \epsilon_1\right\}$ and $\mathbb{B}_\infty = \left\{x \in \mathbb{R}^{d_I} : \|x\|_\infty \leq \epsilon_\infty\right\}$. Let $\mathcal{H}$ be the convex hull of the union of $\mathbb{B}_1$ and $\mathbb{B}_\infty$. Specifically, $\mathcal{H}$ is the smallest convex set that contains the union $\mathbb{B}_1 \cup \mathbb{B}_\infty$. The authors used the convex hull $\mathcal{H}$ for provable robustness.

Here, I show that the convex hull $\mathcal{H}$ is related to a ball based on a combined distance d_{combined}. I define this distance as:

$$d_{\text{combined}}(x_1, x_2) = \beta \|x_2 - x_1\|_1 + (1 - \beta) \|x_2 - x_1\|_\infty ,$$

where β is set to $\beta := \frac{\epsilon_1 - \epsilon_\infty}{\epsilon_\infty (d_I - 1)}$. I then consider a ball $\mathbb{B}_{\text{combined}}$ with radius $\epsilon_{\text{combined}} := \epsilon_1$ based on the distance d_{combined}. Mathematically, this ball is defined as:

$$\mathbb{B}_{\text{combined}} = \left\{x \in \mathbb{R}^{d_I} : d_{\text{combined}}(0, x) = \beta \|x\|_1 + (1 - \beta) \|x\|_\infty \leq \epsilon_1\right\} .$$

This ball $\mathbb{B}_{\text{combined}}$ has the following subset relationship with the convex hull $\mathcal{H}$:

$$\begin{aligned} &\mathcal{H} = \mathbb{B}_{\text{combined}}, && \text{(When } d_I = 2\text{)} \\ &\mathcal{H} \subsetneq \mathbb{B}_{\text{combined}}. && \text{(When } d_I > 2\text{)} \end{aligned}$$

This relationship reveals that their defense [Croce and Hein, 2020a], which utilizes the

convex hull $\mathcal{H}$, uses a subset of a ball $\mathbb{B}_{\text{combined}}$ that uses a single distance d_{combined}. Therefore, robustness against multiple l_p norm attacks ($p \geq 1$) could be achieved by being robust against one type of attack based on a combined distance d_{combined}. Furthermore, when we use ball $\mathbb{B}_{\text{combined}}$ for adversarial defense instead of the convex hull $\mathcal{H}$, it might be possible to achieve stronger robustness (larger perturbation budget).

Proof. **Relationship: $\mathcal{H} \subset \mathbb{B}_{\text{combined}}$**
Let $x \in \mathcal{H}$.
Based on the fact that both $\mathbb{B}_1$ and $\mathbb{B}_\infty$ are convex, we obtain:
$\mathcal{H} = \{tx_1 + (1-t)x_2 : 0 \leq t \leq 1, x_1 \in \mathbb{B}_1, x_2 \in \mathbb{B}_\infty\}$ (because of the lemma 1 explained below.)
$\exists t, x_1, x_2$ such that $x = tx_1 + (1-t)x_2, 0 \leq t \leq 1, x_1 \in \mathbb{B}_1, x_2 \in \mathbb{B}_\infty$.
From the definitions of balls $\mathbb{B}_1$ and $\mathbb{B}_\infty$, $\|x_1\|_1 \leq \epsilon_1$ and $\|x_2\|_\infty \leq \epsilon_\infty$.
Let us consider the inequality for $\|x\|_1$.

$$\begin{aligned}
\|x\|_1 = \|tx_1 + (1-t)x_2\|_1 &\leq \|tx_1\|_1 + \|(1-t)x_2\|_1 && (\because \text{The triangle inequality}) \\
&\leq t\epsilon_1 + (1-t)\|x_2\|_1 && (\because \|x_1\|_1 \leq \epsilon_1) \\
&\leq t\epsilon_1 + (1-t)d_I\epsilon_\infty && (\because \|x_2\|_1 \leq d_I \|x_2\|_\infty \leq d_I\epsilon_\infty)
\end{aligned}$$

Similarly, let us consider the inequality for $\|x\|_\infty$.

$$\begin{aligned}
\|x\|_\infty = \|tx_1 + (1-t)x_2\|_\infty &\leq \|tx_1\|_\infty + \|(1-t)x_2\|_\infty && (\because \text{The triangle inequality}) \\
&\leq t\|x_1\|_\infty + (1-t)\epsilon_\infty && (\because \|x_2\|_\infty \leq \epsilon_\infty) \\
&\leq t\epsilon_1 + (1-t)\epsilon_\infty && (\because \|x_1\|_\infty \leq \|x_1\|_1 \leq \epsilon_1)
\end{aligned}$$

By combining the above two inequalities, we obtain the following inequality.

$$\begin{aligned}
\beta\|x\|_1 + (1-\beta)\|x\|_\infty &\leq \beta(t\epsilon_1 + (1-t)d_I\epsilon_\infty) + (1-\beta)(t\epsilon_1 + (1-t)\epsilon_\infty) \\
&= t\epsilon_1 + (1-t)(d_I\beta + 1 - \beta)\epsilon_\infty \\
&= t\epsilon_1 + (1-t)\epsilon_1 = \epsilon_1 && (\because \text{The definition of } \beta)
\end{aligned}$$

Hence, $x \in \mathbb{B}_{\text{combined}}$. Because we derived $x \in \mathbb{B}_{\text{combined}}$ from $x \in \mathcal{H}$, we proved $\mathcal{H} \subset \mathbb{B}_{\text{combined}}$.

Relationship: $\mathcal{H} = \mathbb{B}_{\text{combined}}$ when $d_I = 2$
Having already proved $\mathcal{H} \subset \mathbb{B}_{\text{combined}}$, we only need to prove $\mathbb{B}_{\text{combined}} \subset \mathcal{H}$ when $d_I = 2$.
Hence, let $x \in \mathbb{B}_{\text{combined}}$ and $d_I = 2$, which denotes that x is a two-dimensional vector.
Let $x = (v_1, v_2)$ and i^* be the index of x that maximizes its absolute value, hence we can express this as $|v_{i^*}| = \max_{i=1,2} |v_i|$. We then denote the other index of x as i^{**}, hence

$i^{**} = 3 - i^*$.

(i) First, let us consider when $|v_{i^{**}}| \neq \epsilon_\infty$.

We can set $x_2^* = \epsilon_\infty \text{sign}(x)$, $t^* = 1 - \frac{|v_{i^{**}}|}{\epsilon_\infty}$, and $x_1^* = \frac{1}{t^*}\left(x - (1-t^*)\, x_2^*\right)$ where $\text{sign}(\cdot)$ is the element-wise sign function.

Then, $x = t^* x_1^* + (1-t^*)x_2^*$ and $\|x_2^*\|_\infty = \epsilon_\infty \|\text{sign}(x)\|_\infty \leq \epsilon_\infty$. Hence, to show that $x \in \mathcal{H}$, we only need to show that $\|x_1^*\|_1 \leq \epsilon_1$. Let us consider x_1^* value of index i^*, which can be simplified as:

$$\begin{aligned}
\frac{1}{t^*}\left(v_{i^*} - (1-t^*)\epsilon_\infty \text{sign}(v_{i^*})\right) &= \frac{1}{t^*}\left(v_{i^*} + (t^*-1)\epsilon_\infty \text{sign}(v_{i^*})\right) \\
&= \frac{1}{t^*}\left(v_{i^*} - \frac{|v_{i^{**}}|}{\epsilon_\infty}\epsilon_\infty \text{sign}(v_{i^*})\right) \quad (\because \text{The definition of } t^*) \\
&= \frac{1}{t^*}\left(v_{i^*} - |v_{i^{**}}|\text{sign}(v_{i^*})\right).
\end{aligned}$$

Similarly, when we consider x_1^* value of index i^{**}, it can be simplified as:

$$\begin{aligned}
\frac{1}{t^*}\left(v_{i^{**}} - (1-t^*)\epsilon_\infty \text{sign}(v_{i^{**}})\right) &= \frac{1}{t^*}\left(v_{i^{**}} - \frac{|v_{i^{**}}|}{\epsilon_\infty}\epsilon_\infty \text{sign}(v_{i^{**}})\right) \quad (\because \text{The definition of } t^*) \\
&= \frac{1}{t^*}\left(v_{i^{**}} - |v_{i^{**}}|\text{sign}(v_{i^{**}})\right) = \frac{1}{t^*}0 = 0.
\end{aligned}$$

Thus, $\|x_1^*\|_1 = \left|\frac{1}{t^*}\left(v_{i^*} - |v_{i^{**}}|\text{sign}(v_{i^*})\right)\right| + |0| = \frac{1}{t^*}\left|v_{i^*} - |v_{i^{**}}|\text{sign}(v_{i^*})\right| = \frac{1}{t^*}\left(|v_{i^*}| - |v_{i^{**}}|\right)$.

The condition $x \in \mathbb{B}_{\text{combined}}$ can be rewritten as:

$$\begin{aligned}
\epsilon_1 \geq \beta \|x\|_1 + (1-\beta)\|x\|_\infty &= \beta\left(|v_{i^*}| + |v_{i^{**}}|\right) + (1-\beta)|v_{i^*}| \\
&= |v_{i^*}| + \beta|v_{i^{**}}|.
\end{aligned}$$

Thus, we obtain the inequality $|v_{i^*}| \leq \epsilon_1 - \beta|v_{i^{**}}|$.

Using this, we obtain the inequality that $\|x_1^*\|_1 \leq \epsilon_1$.

$$\begin{aligned}
\|x_1^*\|_1 = \frac{1}{t^*}\left(|v_{i^*}| - |v_{i^{**}}|\right) &\leq \frac{1}{t^*}\left(\epsilon_1 - \beta|v_{i^{**}}| - |v_{i^{**}}|\right) \\
&= \frac{1}{t^*}\left(\epsilon_1 - \frac{\epsilon_1 - \epsilon_\infty + \epsilon_\infty(2-1)}{\epsilon_\infty(2-1)}|v_{i^{**}}|\right) \\
&= \frac{1}{t^*}\epsilon_1\left(1 - \frac{|v_{i^{**}}|}{\epsilon_\infty}\right) = \frac{1}{t^*}\epsilon_1 t^* = \epsilon_1
\end{aligned}$$

Hence, we derived $x \in \mathcal{H}$ when $|v_{i^{**}}| \neq \epsilon_\infty$.

(ii) Now, let us consider when $|v_{i^{**}}| = \epsilon_\infty$.

From the condition $x \in \mathbb{B}_{\text{combined}}$, we obtain the inequality $|v_{i^*}| \leq \epsilon_1 - \beta|v_{i^{**}}|$ (just as explained above.). From this inequality and the condition $|v_{i^{**}}| = \epsilon_\infty$, we obtain the

inequality $|v_{i^*}| \leq \epsilon_\infty$ as follows.

$$|v_{i^*}| \leq \epsilon_1 - \beta|v_{i^{**}}| = \epsilon_1 - \beta\epsilon_\infty = \epsilon_1 - \frac{\epsilon_1 - \epsilon_\infty}{\epsilon_\infty (2-1)}\epsilon_\infty = \epsilon_\infty$$

From the inequality $\epsilon_\infty = |v_{i^{**}}| \leq |v_{i^*}| \leq \epsilon_\infty$, we obtain $|v_{i^{**}}| = |v_{i^*}| = \epsilon_\infty$.
Let us set $x_1^* = 0$, $x_2^* = x$, and $t^* = 0$. Then, $x = 0 + 1x = t^* x_1^* + (1 - t^*)x_2^*$, $\|x_1^*\|_1 = 0 \leq \epsilon_1$, and $\|x_2^*\|_\infty = \epsilon_\infty \leq \epsilon_\infty$. Hence, $x \in \mathcal{H}$ when $|v_{i^{**}}| = \epsilon_\infty$.
From cases (i) and (ii), we derived $\mathbb{B}_{\text{combined}} \subset \mathcal{H}$ from $x \in \mathbb{B}_{\text{combined}}$ when $d_I = 2$, thus $\mathbb{B}_{\text{combined}} = \mathcal{H}$ for $d_I = 2$.

Relationship: $\mathcal{H} \subsetneq \mathbb{B}_{\text{combined}}$ when $d_I > 2$
We need to show that $\mathbb{B}_{\text{combined}} \not\subset \mathcal{H}$ when $d_I > 2$.
First, we consider the point $x^* = (\omega, \omega, 0, 0, \cdots, 0)$ where $\omega := \frac{(d_I-1)\epsilon_1\epsilon_\infty}{\epsilon_1 + (d_I-2)\epsilon_\infty}$.
Then, $\|x^*\|_1 = 2\omega$ and $\|x^*\|_\infty = \omega$. From these, we obtain the following equation:

$$\begin{aligned}
\beta \|x^*\|_1 + (1-\beta)\|x^*\|_\infty &= 2\beta\omega + (1-\beta)\omega = (1+\beta)\omega \\
&= \left(1 + \frac{\epsilon_1 - \epsilon_\infty}{\epsilon_\infty (d_I - 1)}\right) \frac{(d_I - 1)\epsilon_1\epsilon_\infty}{\epsilon_1 + (d_I - 2)\epsilon_\infty} \\
&\qquad (\because \text{The definitions of } \beta \text{ and } \omega) \\
&= \frac{\epsilon_\infty (d_I - 1) + \epsilon_1 - \epsilon_\infty}{\epsilon_\infty (d_I - 1)} \frac{(d_I - 1)\epsilon_1\epsilon_\infty}{\epsilon_1 + (d_I - 2)\epsilon_\infty} \\
&= (\epsilon_\infty (d_I - 1) + \epsilon_1 - \epsilon_\infty) \frac{\epsilon_1}{\epsilon_1 + (d_I - 2)\epsilon_\infty} \\
&= (\epsilon_\infty (d_I - 2) + \epsilon_1) \frac{\epsilon_1}{\epsilon_1 + (d_I - 2)\epsilon_\infty} = \epsilon_1
\end{aligned}$$

Thus, $x^* \in \mathbb{B}_{\text{combined}}$.
Before verifying whether x^* also belongs to $\mathcal{H}$, let us consider inequalities for ω.
From the condition $\epsilon_\infty < \epsilon_1$, we can derive the inequality $\epsilon_\infty < \omega$ as follows.

$$\begin{aligned}
\epsilon_\infty &< \epsilon_1 \\
(d_I - 2)\epsilon_\infty &< (d_I - 2)\epsilon_1 \quad (\text{Multiplying } (d_I - 2) \text{ on both sides}) \\
\epsilon_1 + (d_I - 2)\epsilon_\infty &< (d_I - 1)\epsilon_1 \quad (\text{Adding } \epsilon_1 \text{ on both sides}) \\
\epsilon_\infty &< \omega \quad \left(\text{Multiplying } \frac{\epsilon_\infty}{\epsilon_1 + (d_I - 2)\epsilon_\infty} \text{ on both sides}\right)
\end{aligned}$$

Similarly, from the condition $\epsilon_1 < d_I\epsilon_\infty$, we can derive the inequality $\epsilon_1 < 2\omega$ as follows.

$$\begin{aligned}
\epsilon_1 &< d_I\epsilon_\infty \\
\epsilon_1 + (d_I - 2)\epsilon_\infty &< 2(d_I - 1)\epsilon_\infty \quad (\text{Adding } (d_I - 2)\epsilon_\infty \text{ on both sides}) \\
\epsilon_1 &< 2\omega \quad \left(\text{Multiplying } \frac{\epsilon_1}{\epsilon_1 + (d_I - 2)\epsilon_\infty} \text{ on both sides}\right)
\end{aligned}$$

To verify whether $x^* \in \mathcal{H}$ or $x^* \notin \mathcal{H}$, we must try to find x_1^*, x_2^*, and $0 \leq t^* \leq 1$ such that $x^* = t^* x_1^* + (1 - t^*)x_2^*$, $x_1^* \in \mathbb{B}_1$, and $x_2^* \in \mathbb{B}_\infty$.
First, let us denote $x_1^* = (\alpha_1, \alpha_2, \cdots, \alpha_d)$ and $x_2^* = (\beta_1, \beta_2, \cdots, \beta_d)$.
Because x_2^* needs to satisfy $\|x_2^*\|_\infty = \max_{1 \leq i \leq d} |\beta_i| \leq \epsilon_\infty$, we obtain inequalities $|\beta_1| \leq \epsilon_\infty$ and $|\beta_2| \leq \epsilon_\infty$. Given these inequalities, as well as $\omega > \epsilon_\infty$, and the condition that x^* should be a convex combination of x_1^* and x_2^*, the first two elements of x_1^* need to satisfy the conditions $|\alpha_1| \geq w$ and $|\alpha_2| \geq w$. From these conditions, we can derive the following inequality.

$$\|x_1^*\|_1 = \sum_{1 \leq i \leq d} |\alpha_i| \geq |\alpha_1| + |\alpha_2| \geq w + w = 2w > \epsilon_1$$

Because we obtain inequality $\|x_1^*\|_1 > \epsilon_1$, x_1^* cannot be belong to $\mathbb{B}_1$. Thus, there are no points x_1^*, x_2^*, and value t^* that satisfy conditions for x^* to belong to the convex hull $\mathcal{H}$, thus $x^* \notin \mathcal{H}$.
Because there exists a point x^* such that $x^* \in \mathbb{B}_{\text{combined}}$ and $x^* \notin \mathcal{H}$, we know $\mathcal{H} \subsetneq \mathbb{B}_{\text{combined}}$ when $d_I > 2$ (we already proved $\mathcal{H} \subset \mathbb{B}_{\text{combined}}$). □

Lemma 1. *There are convex sets A and B. Let us denote the convex hull of their union as $\mathcal{H}_{A \cup B}$. The following equation then holds:*

$$\mathcal{H}_{A \cup B} = \{ta + (1 - t)b : 0 \leq t \leq 1, a \in A, b \in B\} \tag{7.1}$$

(Note that this lemma is not my contribution.)

Proof. For convenience, let us denote the set on the right hand side of the equation (7.1) as RHS.

Relationship: $\mathcal{H}_{A \cup B} \subset$ RHS
The definition of the convex hull of a set S is the smallest convex set that contains the set S. Therefore, it is sufficient to prove that the set RHS is a convex set containing $A \cup B$.
First, let us show the set RHS is convex. To do this, we need to show that any convex combination of two points in the set belongs to the set RHS. Let us consider two points of the set RHS: $t_1 a_1 + (1 - t_1)b_1$ and $t_2 a_2 + (1 - t_2)b_2$ where $0 \leq t_1, t_2 \leq 1, a_1, a_2 \in A$, and $b_1, b_2 \in B$. Then, let us consider their convex combination for a value $0 \leq \lambda \leq 1$:

$$\lambda \{t_1 a_1 + (1 - t_1)b_1\} + (1 - \lambda) \{t_2 a_2 + (1 - t_2)b_2\}.$$

Let us define $a_3 := \frac{\lambda t_1 a_1 + (1-\lambda) t_2 a_2}{\lambda t_1 + (1-\lambda) t_2}$, $b_3 := \frac{\lambda(1-t_1) b_1 + (1-\lambda)(1-t_2) b_2}{\lambda(1-t_1) + (1-\lambda)(1-t_2)}$, and $\gamma := \lambda t_1 + (1 - \lambda) t_2$.

Because point a_3 is a convex combination of two points a_1 and a_2 in the convex set A, we know that $a_3 \in A$. Similarly, we know that $b_3 \in B$ and $0 \leq \gamma \leq 1$. Additionally, we obtain the following equation which shows that the convex combination of two points of RHS can be rewritten as $\gamma a_3 + (1-\gamma)b_3$.

$$\begin{aligned}
\gamma a_3 + (1-\gamma)b_3 &= \{\lambda t_1 + (1-\lambda)t_2\} \frac{\lambda t_1 a_1 + (1-\lambda)t_2 a_2}{\lambda t_1 + (1-\lambda)t_2} + (1-\gamma)b_3 \\
&= \lambda t_1 a_1 + (1-\lambda)t_2 a_2 + \{1 - \lambda t_1 - (1-\lambda)t_2\} \frac{\lambda(1-t_1)b_1 + (1-\lambda)(1-t_2)b_2}{\lambda(1-t_1) + (1-\lambda)(1-t_2)} \\
&= \lambda t_1 a_1 + (1-\lambda)t_2 a_2 + \lambda(1-t_1)b_1 + (1-\lambda)(1-t_2)b_2 \\
(\because \lambda(1-t_1) &+ (1-\lambda)(1-t_2) = \lambda - \lambda t_1 + 1 - \lambda - t_2 + \lambda t_2 = 1 - \lambda t_1 - (1-\lambda)t_2) \\
&= \lambda \{t_1 a_1 + (1-t_1)b_1\} + (1-\lambda)\{t_2 a_2 + (1-t_2)b_2\}
\end{aligned}$$

From the definition of the set, we know that the convex combination also belongs to the set RHS. Thus, the set RHS is a convex set.
Now, let us try to prove $A \cup B \subset$ RHS. When we fix $t^* = 1$, we obtain point: $t^* a + (1-t^*)b = 1a + 0b = a$. Similarly, when we fix $t^* = 0$, we obtain point: $t^* a + (1-t^*)b = b$. Thus, any point in set A belongs to the set RHS, and any point in set B belongs to the set RHS. Hence, we know $A \cup B \subset$ RHS.
Because we have shown that the set RHS is convex and contains $A \cup B$, we have proved that $\mathcal{H}_{A \cup B} \subset$ RHS.

Relationship: $\mathcal{H}_{A \cup B} \supset$ RHS
We know that $A, B \subset A \cup B \subset \mathcal{H}_{A \cup B}$. Given that the convex hull $\mathcal{H}_{A \cup B}$ is convex, convex combination $ta + (1-t)b$ also belongs to the convex hull $\mathcal{H}_{A \cup B}$ where $a \in A \subset \mathcal{H}_{A \cup B}$, $b \in B \subset \mathcal{H}_{A \cup B}$, and $0 \leq t \leq 1$. Hence, $\mathcal{H}_{A \cup B} \supset$ RHS. □

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230852750 (print)
9788230841914 (PDF)