SELF-SUPERVISED REPRESENTATION LEARNING ON MANIFOLDS

Anonymous authors
Paper under double-blind review

ABSTRACT

We explore the use of a topological manifold, represented as a collection of charts, as the target space of neural network based representation learning tasks. This is achieved by a simple adjustment to the output of an encoder's network architecture plus the addition of a maximal mean discrepancy based loss function for regularization. Most algorithms in representation learning are easily adaptable to our framework and we demonstrate its effectiveness by adjusting SimCLR to have a manifold encoding space. Our experiments show that we obtain a substantial performance boost over the baseline for low dimensional encodings.

1 INTRODUCTION

Representation learning algorithms typically produce encodings into a Euclidean space. However, if the *manifold hypothesis* (the assumption that the data is well-approximated by a manifold) is taken seriously then a Euclidean target space is unnecessarily big for encoding this manifold: by the Whitney embedding theorem, to embed an n-dimensional manifold into a Euclidean space one may need up to $2n$ dimensions for the ambient space. Additionally, such algorithms seldom regularize the encoding space (i.e. encourage it to look like a prior distribution) to ensure that it is well-behaved.

In this work we develop a framework for using a manifold as a target space of deep representation learning algorithms. Following the work of Korman (2018), which uses manifolds as the latent space of an autoencoder, this is done by having the encoding network output chart embeddings and membership probabilities for an atlas of a manifold. For matching the distribution of embeddings to a manifold prior, we introduce a maximal mean discrepancy (MMD) (Gretton et al., 2012) loss for manifolds.

Our framework is flexible enough to apply to a variety of representation learning algorithms and in this paper we do experiments with a manifold generalization of SimCLR (Chen et al., 2020a). Our results show that, in low dimensions, replacing the Euclidean target space of these algorithms with a manifold gives a significant improvement in the learned embeddings. Besides being of theoretical interest, low dimensional embeddings have the practical benefits of having faster distance computations, requiring less storage, providing visualizations (in the case of dimensions 2 or 3), and avoiding issues with curse of dimensionality for downstream tasks from the embedding (such as clustering).

We note that our manuscript authors (2021) is an extended form of this work, which additionally discusses using manifold encodings for deep metric learning.

1.1 RELATED WORK

Learning an atlas The idea of learning a manifold as an atlas was discussed in Pitelis et al. (2013) where they learn a collection of linear charts via a generalization of principal component analysis. The work of Korman (2018), which we build upon, learns an atlas as the latent space of an autoencoder. In that work the regularization of the latent space is via adversarial training instead of the MMD loss we use in this paper.

Encoding space regularization Regularization of a latent space is typically only done in algorithms that have a decoder/generator, such as variational autoencoders (Kingma & Welling, 2013) and adversarial autoencoders (Makhzani et al., 2015). The effectiveness of using an MMD loss in particular on such an encoding space is demonstrated in (Tolstikhin et al., 2017) and inspired our

MMD loss function. The work of Grattarola et al. (2019) is of similar spirit to ours, as they extend the latent spaces of adversarial autoencoders to more geometrically interesting ones, namely constant curvature Riemannian manifolds.

2 MANIFOLDS AS ENCODING SPACES

2.1 REPRESENTING A DATASET AS A MANIFOLD

For modeling a distribution of data, $\mathcal{X}$, as a manifold, we use the same approach as in Korman (2018). Namely we posit the existence of a latent space $\mathcal{Z} = [0,1]^d \times \{1,\dots,n\}$ of n, d-dimensional charts with coordinate maps $\psi_i : [0,1]^d \to \mathcal{X}$ that forms an atlas of a manifold. We put the uniform distribution on $\mathcal{Z}$ and we let X, Z, J denote the random variables on $\mathcal{X}, [0,1]^d$, and $\{1,\dots,n\}$, respectively. We will denote by $\mathbb{1}_y$ the distribution supported at a single point y. In our decoder-free setup, we wish to learn:

1. The inverse mappings of the ψ_i, which we denote by $\phi_i : \mathcal{X} \to [0,1]^d$ and which satisfy
$$p(z|J = i, X = x) = \mathbb{1}_{\phi_i(x)}.$$
2. The chart membership function $q = (q_1,\dots,q_n) : \mathcal{X} \to [0,1]^n$ defined by
$$q(x) = (p(J = 1 \mid X = x),\dots,p(J = n \mid X = x)). \tag{1}$$

Together these determine the distribution
$$p(z,j) = \mathbb{E}_x p(z,j \mid x) = \mathbb{E}_x p(z \mid j,x)p(j \mid x) = \mathbb{E}_x q_j(x)\mathbb{1}_{\phi_j(x)}, \tag{2}$$
on $\mathcal{Z}$, which we wish to be uniform.

An additional desire is that we have an efficient atlas in the sense that any point x should be in as few charts as possible. Thus while $p(J)$ should be uniform, we want the conditional distributions $p(J \mid X = x)$ to have low entropy: if the distributions $p(J \mid X = x)$ are mostly deterministic then encoding x requires us to only keep the coordinates and chart number for the chart with highest probability for x. In other words, for the representation of x at inference time we take
$$x \mapsto (\phi_i(x), i) \in \mathbb{R}^d \times \{1,\dots,n\}, \quad \text{where } i = \operatorname*{argmax}_j q_j(x), \tag{3}$$
which has just $\log n$-more bits of information than the Euclidean case.

2.2 A MAXIMAL MEAN DISCREPANCY LOSS FOR MANIFOLDS

For a given embedding task, we propose to parameterize the functions $\{\phi_1,\dots,\phi_n,q\}$ using neural networks and optimize the parameters via gradient descent for a loss function consisting of a task-specific term (e.g. a contrastive loss function) plus a regularization term that encourages the distribution $p(z,j)$ given by (2) to be close to uniform and the discrete distribution $q(x)$ in (1) to be close to a deterministic distribution for each $x \in \mathcal{X}$.

In the Euclidean case (i.e. when $n = 1$), there are two popular ways for regularizing the latent space to match a prior distribution: using adversarial training (Makhzani et al., 2015; Tolstikhin et al., 2017) or via an MMD (Gretton et al., 2012) loss (Tolstikhin et al., 2017). In Korman (2018) an adversarial loss is used for a manifold latent space but in this work we use an MMD loss due to better training stability and less hyperparameters to tune. If P and Q are two distributions on a common space and k is a reproducing kernel, then the MMD gives a measure of the difference between the distributions, and is defined by
$$\mathrm{MMD}_k(P,Q)^2 = \mathbb{E}_{y_1,y_2\sim P\times P}k(y_1,y_2) - 2\mathbb{E}_{y_1,y_2\sim P\times Q}k(y_1,y_2) + \mathbb{E}_{y_1,y_2\sim Q\times Q}k(y_1,y_2). \tag{4}$$

Let p and q be given by (2) and (1), respectively, $\mathcal{U}_\mathcal{Z}$ denote the uniform distribution on $\mathcal{Z}$, and $\mathcal{U}_\mathcal{J}$ denote the uniform distribution on $\mathcal{J} = \{1,\dots,n\}$. For the loss function encouraging p to be close to $\mathcal{U}_\mathcal{Z}$ we will take an approximation of $\mathrm{MMD}_{k_\mathcal{Z}}(p,\mathcal{U}_\mathcal{Z})^2$ and for the loss function encouraging $q(x)$ to be close to $\mathcal{U}_\mathcal{J}$, we take $-\mathbb{E}_x \mathrm{MMD}_{k_\mathcal{J}}(q(x),\mathcal{U}_\mathcal{J})$. This will achieve the goal of encouraging p to be uniform and for $q(x)$ to be deterministic.

To get a kernel $k_{\mathcal{Z}}$ on $\mathcal{Z}$ we can start with a kernel k_0 on $[0,1]^d$ and then define

$$k_{\mathcal{Z}} : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R},\ ((z_1, i), (z_2, j)) \mapsto \delta_{ij} k_0(z_1, z_2)$$

where $\delta_{ii} = 1, \delta_{ij} = 0$ if $i \neq j$. For k_0 we take, as in Tolstikhin et al. (2017), the inverse multi-quadratics kernel but pulled back via the sigmoid function[1]: $k_0(x, y) = \frac{d/6}{d/6 + |\sigma^{-1}(x) - \sigma^{-1}(y)|^2}$. We approximate $\mathrm{MMD}_{k_{\mathcal{Z}}}(p, \mathcal{U}_{\mathcal{Z}})^2$ using the U-statistic estimator in Gretton et al. (2012), adjusted to our manifold setting. Explicitly:

Proposition 1. *Let p be the distribution on $\mathcal{Z}$ defined by (2) and let $\mathcal{U}_{\mathcal{Z}}$ denote the uniform distribution on $\mathcal{Z}$. Given a random sample $\{x_1, \ldots, x_N\}$ of $\mathcal{X}$ and a random sample $\{w_1, \ldots, w_N\}$ drawn uniformly from $[0,1]^d$, an estimator for $\mathrm{MMD}_{k_{\mathcal{Z}}}(p, \mathcal{U}_{\mathcal{Z}})^2$ is:*

$$\ell_{\mathcal{Z}}(q, \phi_1, \ldots, \phi_n) = \frac{1}{N(N-1)} \sum_{\substack{j,k=1 \\ j \neq k}}^{N} \sum_{i=1}^{n} q_i(x_j) q_i(x_k) k_0(\phi_i(x_j), \phi_i(x_k)) \tag{5}$$
$$- \frac{2}{nN^2} \sum_{j,k=1}^{N} \sum_{i=1}^{n} q_i(x_j) k_0(\phi_i(x_j), w_k) + \frac{1}{nN(N-1)} \sum_{\substack{j,k=1 \\ j \neq k}}^{N} k_0(w_j, w_k).$$

See section A.3 for the proof.

For $\mathbb{E}_x \mathrm{MMD}_{k_{\mathcal{J}}}(q(x), \mathcal{U}_{\mathcal{J}})$ we take the kernel $k_{\mathcal{J}} : \{1, \ldots, n\} \times \{1, \ldots, n\} \to \mathbb{R}, (i, j) \mapsto \delta_{ij}$ and define

$$\ell_{\mathcal{J}}(q) := -\mathbb{E}_x \mathrm{MMD}_{k_{\mathcal{J}}}(q(x), \mathcal{U}_{\mathcal{J}}) = -\sum_{i=1}^{n} \left(q_i(x) - \frac{1}{n} \right)^2.$$

The total regularization loss is then

$$\ell_{reg}(q, \phi_1, \ldots, \phi_n) = \lambda_1 \ell_{\mathcal{Z}}(q, \phi_1, \ldots, \phi_n) + \lambda_2 \ell_{\mathcal{J}}(q) \tag{6}$$

for some hyperparameters $\lambda_1, \lambda_2 \in [0, \infty)$.

2.3 Summary of our framework

We now describe our general technique of turning a deep representation learning algorithm to one that has a manifold encoding space. The typical setup for such an algorithm, $\mathtt{A}$, is to learn an encoder f mapping the dataset $\mathcal{X}$ to the space $\mathbb{R}^d$ via optimizing some loss function $\ell_{\mathtt{A}}$ defined on embeddings of a mini-batch $\{x_1, \ldots, x_N\} \subset \mathcal{X}$.

To adjust the algorithm to one that has a manifold as the encoding space, we see from the discussion in 2.1 that f should be replaced by a collection of maps $\phi_1, \ldots, \phi_n, q$ where $\phi_i : \mathcal{X} \to [0,1]^d$ is the ith coordinate map and $q : \mathcal{X} \to [0,1]^n$ is the chart membership function. In practice, for the functions $\phi_1, \ldots, \phi_n, q$, we take a backbone network and attach $n+1$ linear heads followed by a sigmoid activation on the first n (which have output in $\mathbb{R}^d$) and a softmax activation on the head defining q (which has output in $\mathbb{R}^n$).

In many contrastive representation learning algorithms (such as SimCLR (Chen et al., 2020a), MoCo v2 (Chen et al., 2020b), and BYOL (Grill et al., 2020)), the loss $\ell_{\mathtt{A}}$ is a function of an auxiliary projection head $h : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ applied to the embedding vectors. In these cases, for the manifold version we use n-many projection heads (one for each coordinate chart), and then produce a single projection vector for every data point by taking a sum of these individual projection vectors weighted by q. We supplement the resulting loss $\ell_{\mathtt{MA}}$ with the regularization loss (6).

At inference and evaluation time, we use the compressed representation (3). This ensures that the representation is indeed d-dimensional and gives a fair evaluation comparison to $\mathtt{A}$, which embeds into a d-dimensional Euclidean space. For example, if $q_i(x)$ were the uniform distribution then using the full-representation $(\phi_1(x), \ldots, \phi_n(x))$ instead of the compressed one would essentially be "cheating" into an nd-dimensional Euclidean representation.

[1] this avoids having to compute the final sigmoid activation.

We summarize this procedure in Table 1. We also note that the case of $d = 2$ yields a powerful data visualization by plotting the input data at their embedding coordinates for their most probable chart. We show such visualizations in the appendix A.2.

Table 1: Summary of our framework for elevating an algorithm $\mathtt{A}$ to have a manifold encoding space.

TYPICAL SETUP	**MANIFOLD VERSION**
learn a neural network encoder $f : \mathcal{X} \to \mathbb{R}^d$	learn a neural network encoder with $n+1$ heads $f = (\phi_1, \dots, \phi_n, q) : \mathcal{X} \to [0,1]^d \times \cdots \times [0,1]^d \times \mathbb{R}^n$
auxiliary projection head $h : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ giving projection map $h \circ f : \mathcal{X} \to \mathbb{R}^{\tilde{d}}$.	n auxiliary projection heads $h_1, \dots, h_n : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$ giving projection map $\mathcal{X} \to \mathbb{R}^{\tilde{d}},\ x \mapsto \sum_i q_i(x) h_i(\phi_i(x))$.
on a minibatch $\{x_1, \dots, x_N\} \subset \mathcal{X}$ optimize a loss $\ell_{\mathtt{A}}(f(x_1), \dots, f(x_N))$.	on a minibatch $\{x_1, \dots, x_N\} \subset \mathcal{X}$ optimize a loss $\ell_{\mathtt{MA}}(f(x_1), \dots, f(x_N)) + \ell_{reg}(f(x_1), \dots, f(x_N))$.
Representation of $x \in X$ at inference/evaluation: $f(x) \in \mathbb{R}^d$.	Representation of $x \in X$ at inference/evaluation: $(\phi_i(x), i) \in \mathbb{R}^d \times \{1, \dots, n\}$, where $i = \operatorname{argmax}_j q_j(x)$.

3 MSIMCLR

We recall that SimCLR trains a neural network encoder $f : \mathcal{X} \to \mathbb{R}^d$ using a projection head $h : \mathbb{R}^d \to \mathbb{R}^{\tilde{d}}$. A mini-batch is formed by choosing N-images and augmenting in two different ways, producing examples $\{x_1, \dots, x_{2N}\}$. The loss function over this batch is a function of the projection head output of the embeddings of these images:

$$\ell_{\text{SimCLR}}(x_1, \dots, x_n) = c(h(f(x_1)), \dots, h(f(x_{2n}))),$$

where c is a contrastive loss. We follow our meta-procedure from the previous section to adjust SimCLR (Chen et al., 2020a) to have a manifold as encoding space (with resulting algorithm denoted by *MSimCLR*) and run experiments on MNIST (LeCun et al., 2010), FashionMNIST (Xiao et al., 2017), and CIFAR10 (Krizhevsky et al., 2009) for $n \in \{1, 4, 16, 32\}$ and $d \in \{2, 4, 8\}$.

In Chen et al. (2020a) representations of SimCLR are evaluated based on the accuracy of a linear classifier trained on top of the representation. In our case, since we have a collection of charts, we evaluate our manifold representation by putting a linear classifier on each chart. We report the mean and standard deviation of the accuracy of our models on the hold out test sets in Table 2, from which we see that our method provides a significant performance boost over vanilla SimCLR, especially in dimensions two and four. Section A.1 outlines the details of our experiments, including hyperparameter selection.

Table 2: Piecewise linear evaluation accuracy (* denotes no convergence)

METHOD	**# CHARTS**	**DATASET**								
		MNIST			**Fashion MNIST**			**CIFAR10**		
		ENCODING DIMENSION			**ENCODING DIMENSION**			**ENCODING DIMENSION**		
		2	4	8	2	4	8	2	4	8
SimCLR	-	15.3 ± 6.8	75.4 ± 2.1	94.5 ± 1.5	39.8 ± 9.3	62.7 ± 1.8	79.3 ± 0.2	*	66.2 ± 0.2	79.6 ± 0.2
MSimCLR	1	35.4 ± 12.4	66.2 ± 0.5	90.5 ± 2.0	36.1 ± 10.8	59.1 ± 4.9	73.5 ± 3.9	30.2 ± 10.8	61.1 ± 2.4	78.0 ± 1.7
MSimCLR	4	75.1 ± 3.8	89.0 ± 1.7	94.1 ± 2.6	62.5 ± 1.2	71.9 ± 1.5	79.0 ± 0.3	54.5 ± 5.8	68.2 ± 4.5	81.0 ± 1.9
MSimCLR	16	90.4 ± 3.0	94.3 ± 3.5	97.1 ± 0.2	68.5 ± 2.2	73.9 ± 1.0	80.8 ± 0.1	74.2 ± 0.8	73.0 ± 2.0	77.3 ± 0.5
MSimCLR	32	91.3 ± 2.8	96.4 ± 0.2	96.6 ± 0.0	72.1 ± 1.0	72.7 ± 0.5	74.6 ± 2.2	71.4 ± 3.5	64.2 ± 1.1	73.3 ± 2.7

REFERENCES

Anonymous authors. Representation and metric learning on manifolds. *Submitted*, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020a.

Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020b.

Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Adversarial autoencoders with constant-curvature latent manifolds. *Applied Soft Computing*, 81:105511, 2019.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33, 2020.

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.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Eric O Korman. Autoencoding topology. *arXiv preprint arXiv:1803.00156*, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Nikolaos Pitelis, Chris Russell, and Lourdes Agapito. Learning a manifold as an atlas. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1642–1649, 2013.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

A Appendix

A.1 Experiments setup

We ran our experiments using PyTorch Paszke et al. (2019). In all cases we use a batch size of 128 and the Adam optimizer with learning rate 10^{-4} and $\beta_1 = 0.9, \beta_2 = 0.999$ (the PyTorch defaults). For MNIST and FashionMNIST we use a ResNet18 He et al. (2016) backbone and train for 100 epochs while for CIFAR10 we use a ResNet50 backbone and train for 1,000 epochs. We use the same data augmentation used in the CIFAR10 experiments in the original work Chen et al. (2020a): color jitter with strength 0.5 (and a probability of 0.8 of applying), random grayscale with probability 0.2, and a random resized crop.

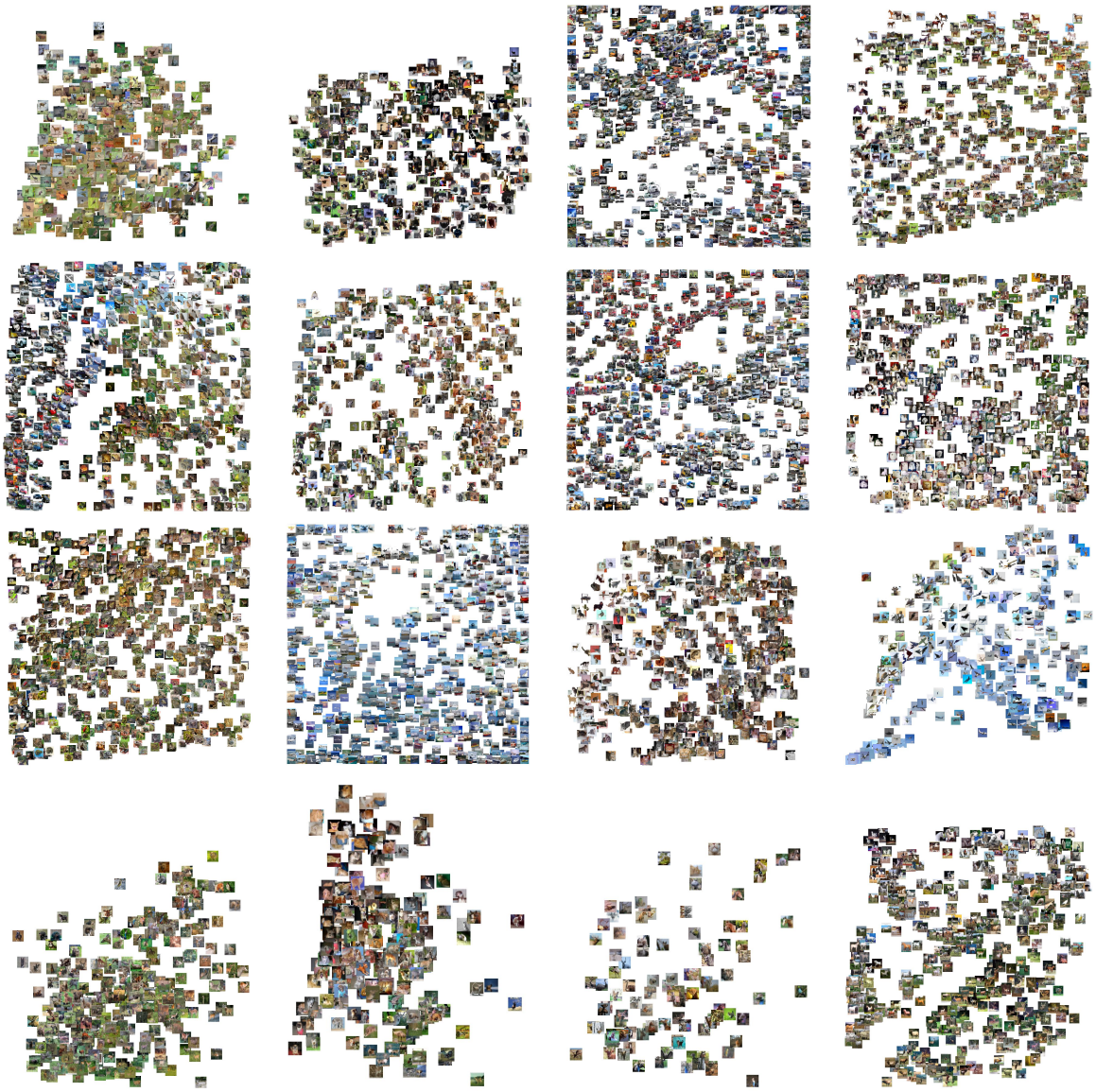
Each of the datasets comes with a standard train/test split. For hyperparameter selection of λ_1, λ_2, and τ (the temperature used in the contrastive loss function) we do a grid search (with $d = 2, n = 16$) over $\lambda_1 \in \{0.1, 1, 5, 10, 20\}, \lambda_2 \in \{0.05, 0.1, 0.2\}$, and $\tau \in \{0.1, 0.5, 1\}$ by training on a random selection of 80% of the training data and then computing the piece-wise linear evaluation

on the remaining 20%. For MNIST and FashionMNIST we chose the hyperparameters that give the best average rank across the two datasets. This yielded $\lambda_1 = 20, \lambda_2 = 0.1$, and $\tau = 1$ for MNIST and FashionMNIST and $\lambda_1 = 20, \lambda_2 = 0.1$, and $\tau = 0.5$ for CIFAR10. Using these parameters we train on the entire train dataset across $d \in \{2, 4, 8\}$ (we noticed for higher d there is not much improvement over the baseline) and $n \in \{1, 4, 16, 32\}$. We also train baseline SimCLR for comparison (but were unable to get convergence in dimension two for CIFAR10). We repeat each training configuration three times and report the mean $\pm$ std of accuracy on the holdout test set in Table 2.

A.2 VISUALIZATIONS

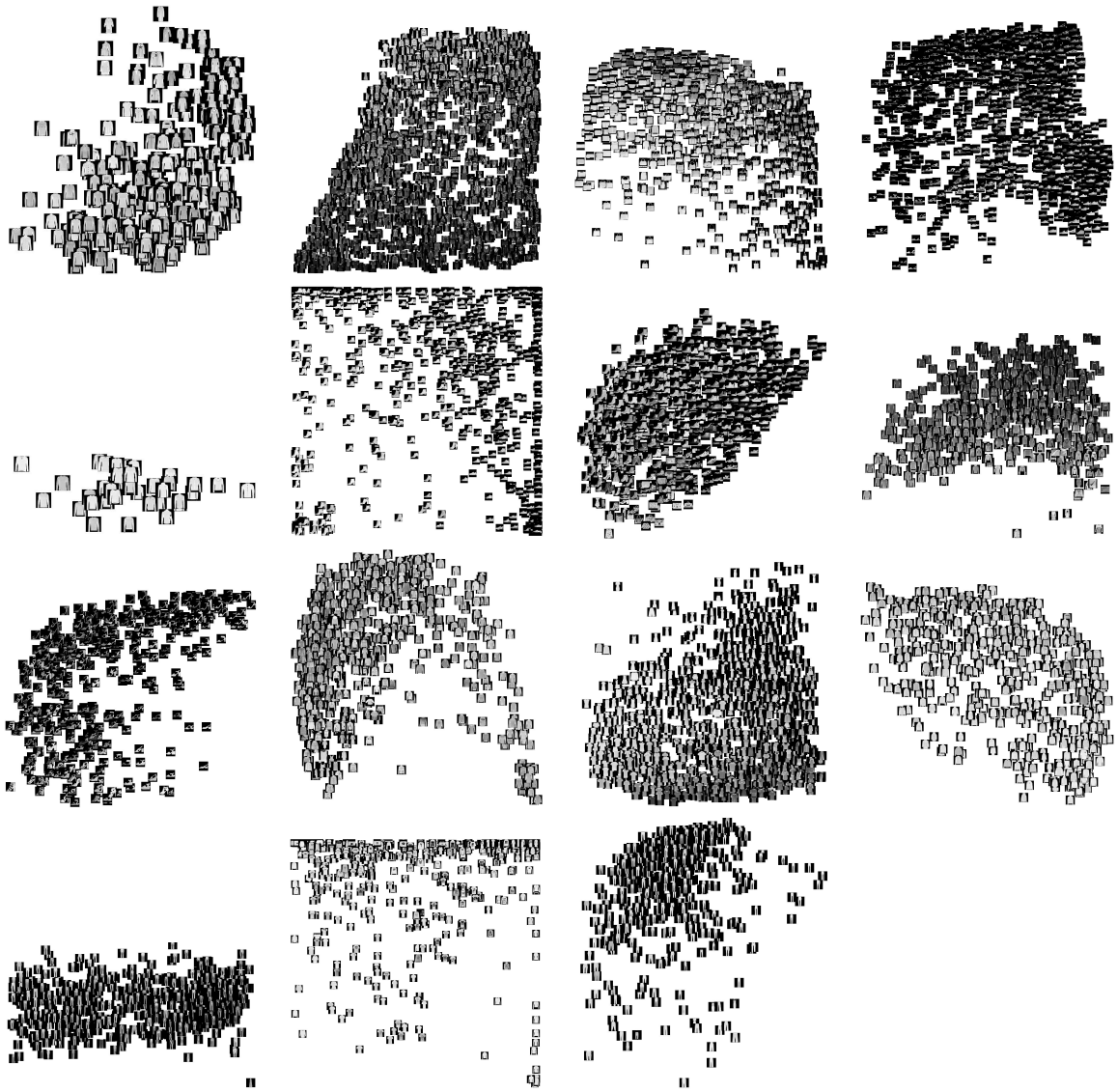
In this section we display visualizations for each dataset using a model with $d = 2, n = 16$. Each image corresponds to a chart and every image in the test set gets plotted at its (x, y) coordinate in the chart with highest probability. The images are best viewed by zooming in.

A.2.1 CIFAR10

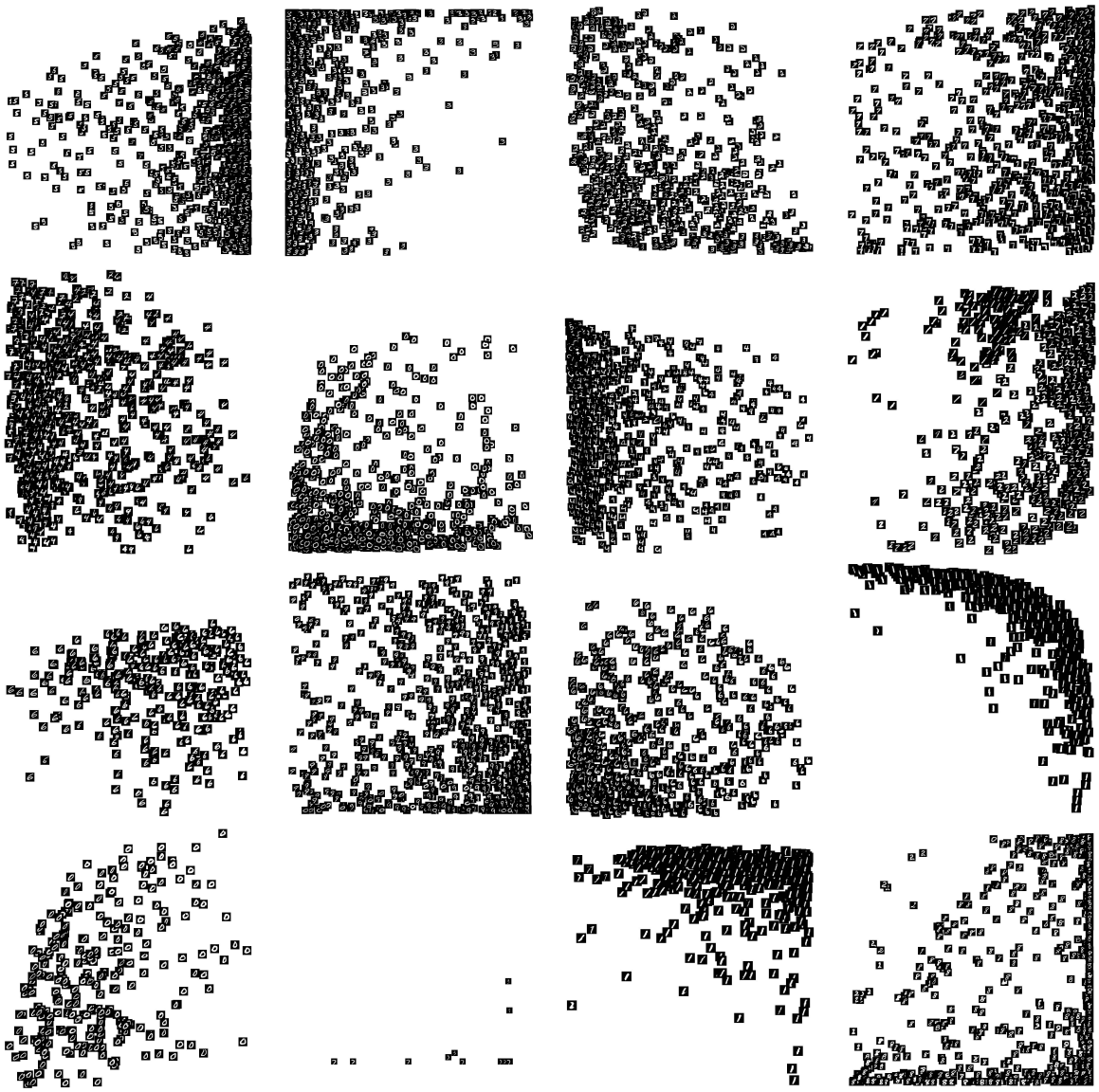


A.2.2 FASHION MNIIST

We note that in this case only 15 charts are displayed since the network did not end up assigning any points to one of them.



A.2.3 MNIST



A.3 PROOF OF PROPOSITION 1

Proof. We breakdown the three terms defining MMD (4). For the first term, we have:

$$\begin{aligned}
&\mathbb{E}_{(z_1,i),(z_2,j)\sim p\times p} k_{\mathcal{Z}}((z_1,i),(z_2,j)) \\
&\qquad = \mathbb{E}_{x,\tilde{x}\sim p_{\text{data}}\times p_{\text{data}}} \sum_{i,j=1}^{n} q_i(x) q_j(\tilde{x}) k_{\mathcal{Z}}((\phi_i(x),i),(\phi_j(\tilde{x}),j)) \\
&\qquad = \mathbb{E}_{x,\tilde{x}\sim p_{\text{data}}\times p_{\text{data}}} \sum_{i=1}^{n} q_i(x) q_i(\tilde{x}) k_0(\phi_i(x),\phi_i(\tilde{x})) \\
&\qquad \approx \frac{1}{N(N-1)} \sum_{\substack{j,k=1\\ j\neq k}}^{N} \sum_{i=1}^{n} q_i(x_j) q_i(x_k) k_0(\phi_i(x_j),\phi_i(x_k)),
\end{aligned}$$

where the first equality comes from (applying twice) the fact that

$$\mathbb{E}_{z,i\sim p} f(z,i) = \mathbb{E}_{x\sim p_{\text{data}}} \mathbb{E}_{z,i\sim q(\cdot|x)} f(z,i) = \mathbb{E}_{x\sim p_{\text{data}}} \sum_i q_i(x) f(\phi_i(x),i), \text{for any } f:\mathcal{Z}\to\mathbb{R} \quad (7)$$

and the approximation in the last line uses the U-statistic estimate

$$\mathbb{E}_{y,\tilde{y}\sim Q\times Q} h(y,\tilde{y}) \approx \frac{1}{N(N-1)} \sum_{\substack{j,k=1\\ j\neq k}}^{N} h(y_j,y_k) \text{ for any } h:\mathcal{X}\times\mathcal{X}\to\mathbb{R}, \quad (8)$$

which holds for any distribution Q and random sample $\{y_1,\ldots,y_N\}$ drawn from Q.

For the second term we have, letting $\mathcal{U}_{[0,1]^d}$ denote the uniform distribution on $[0,1]^d$,

$$\begin{aligned}
&\mathbb{E}_{(z_1,i),(z_2,j)\sim p\times \mathcal{U}_{\mathcal{Z}}} k_{\mathcal{Z}}((z_1,i),(z_2,j)) \\
&\qquad = \mathbb{E}_{x\sim p_{\text{data}}} \sum_{i=1}^{n} q_i(x) \mathbb{E}_{z_2,j\sim \mathcal{U}_{\mathcal{Z}}} k_{\mathcal{Z}}((\phi_i(x),i),(z_2,j)) \\
&\qquad = \mathbb{E}_{x\sim p_{\text{data}}} \sum_{i=1}^{n} q_i(x) \frac{1}{n} \mathbb{E}_{w\sim \mathcal{U}_{[0,1]^d}} k_0(\phi_i(x),w) \\
&\qquad \approx \frac{1}{nN^2} \sum_{j,k=1}^{N} \sum_{i=1}^{n} q_i(x) k_0(\phi_i(x_j),w_k),
\end{aligned}$$

where the first equality again follows from (7) and the approximation in the last line uses the standard expected value estimator applied to each factor. Finally for the third term we again use the estimate (8) to obtain

$$\begin{aligned}
&\mathbb{E}_{(z_1,i),(z_2,j)\sim \mathcal{U}_{\mathcal{Z}}\times \mathcal{U}_{\mathcal{Z}}} k_{\mathcal{Z}}((z_1,i),(z_2,j)) \\
&\qquad = \mathbb{E}_{w\sim \mathcal{U}_{[0,1]^d}} \mathbb{E}_{\tilde{w}\sim \mathcal{U}_{[0,1]^d}} \sum_{i,j=1}^{n} \frac{1}{n^2} k_{\mathcal{Z}}((w,i),(\tilde{w},j)) \\
&\qquad = \mathbb{E}_{w\sim \mathcal{U}_{[0,1]^d}} \mathbb{E}_{\tilde{w}\sim \mathcal{U}_{[0,1]^d}} \frac{1}{n} k_0(w,\tilde{w}) \\
&\qquad \approx \frac{1}{nN(N-1)} \sum_{\substack{j,k=1\\ j\neq k}}^{N} k_0(w_j,w_k).
\end{aligned}$$

Combining these gives the right hand side of (5). □