Supplementary Material

A. Derivation of the von Mises–Fisher Loss

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

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

B. Bounds for $I_{n/2}(\kappa)/I_{n/2-1}(\kappa)$ and $\log C_n(\kappa)$

To approximate $I_{n/2}(\kappa)/I_{n/2-1}(\kappa)$ where n is the dimensionality of the embedding space, we borrow a lower bound from Theorem 4 and an upper bound from Theorem 2 of [50]:

$$
\frac{\kappa}{\frac{n-1}{2} + \sqrt{\left(\frac{n+1}{2}\right)^2 + \kappa^2}} \leq \frac{I_{n/2}(\kappa)}{I_{n/2-1}(\kappa)} \leq \frac{\kappa}{\frac{n-1}{2} + \sqrt{\left(\frac{n-1}{2}\right)^2 + \kappa^2}}. \tag{9}
$$

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

$$
\frac{I_{n/2}(\kappa)}{I_{n/2-1}(\kappa)} \approx \frac{1}{2}(g_n(\kappa) + h_n(\kappa)). \tag{10}
$$

Next, we borrow a clever trick from [31]. Note that:

$$\frac{\mathrm{d}}{\mathrm{d}\kappa}\log C_n(\kappa) = -\frac{I_{n/2}(\kappa)}{I_{n/2-1}(\kappa)}. \tag{11}$$

If we plug in our approximation for the ratio of modified Bessel functions from Equation 10 and integrate with respect to κ, we arrive at the following approximation:

$$\begin{aligned}\log C_n(\kappa) \approx \frac{d-1}{4}\log\left(\frac{d-1}{2}+\sqrt{\left(\frac{d-1}{2}\right)^2+\kappa^2}\right) - \frac{1}{2}\sqrt{\left(\frac{d-1}{2}\right)^2+\kappa^2} \\ + \frac{d-1}{4}\log\left(\frac{d-1}{2}+\sqrt{\left(\frac{d+1}{2}\right)^2+\kappa^2}\right) - \frac{1}{2}\sqrt{\left(\frac{d+1}{2}\right)^2+\kappa^2} + \eta,\end{aligned} \tag{12}$$

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

$$\begin{aligned}\mathcal{L}(y, \boldsymbol{z};\ \boldsymbol{w}_{1:Y}) &\leq \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \frac{C_n(\|\tilde{\boldsymbol{w}}_j\|)}{C_n\left(\|\tilde{\boldsymbol{w}}_j+\beta\boldsymbol{z}\|\right)}\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}] \\ &= \mathbb{E}_{\boldsymbol{z}}\left[\log\left(\sum_j \exp\left(\log C_n(\|\tilde{\boldsymbol{w}}_j\|) - \log C_n\left(\|\tilde{\boldsymbol{w}}_j+\beta\boldsymbol{z}\|\right)\right)\right)\right] - \beta\mathbb{E}[\boldsymbol{w}_y]\mathbb{E}[\boldsymbol{z}].\end{aligned} \tag{13}$$

Equation 13 is the final form of the vMF objective.

C. Initialization of the Concentration for vMF

We seek an initialization of κ for all vMF distributions such that the ratio of modified Bessel functions is a constant greater than zero. Such an initialization would ensure gradients are strong enough for training (see Figure 2) and all vMF distributions are approximately equally concentrated. If we take the upper bound, $h_n(\kappa)$, from Equation 9 and set it equal to a constant, λ, and solve for κ, we get:

$$\kappa = \frac{\lambda}{1-\lambda^2}(n-1). \tag{14}$$

For initializing the concentration of the embedding distribution, $\kappa_{\boldsymbol{z}}$, we introduce a constant scalar, denoted α, that is multiplied with the embedding output of the network, $\tilde{\boldsymbol{z}}$, prior to ℓ_2-normalization. The multiplier does not add any flexibility to the network and also does not affect the direction of the embedding. Since $\kappa_{\boldsymbol{z}} = \|\tilde{\boldsymbol{z}}\|$, we estimate the value of α such that the expected embedding norm over the training dataset is the value we desire from Equation 14:

$$\begin{aligned}\mathbb{E}_{\tilde{\boldsymbol{z}}}\left[\|\alpha\tilde{\boldsymbol{z}}\|\right] &= \frac{\lambda}{1-\lambda^2}(n-1) \\ \Rightarrow \mathbb{E}_{\tilde{\boldsymbol{z}}}\left[\sqrt{\sum_i(\alpha\tilde{\boldsymbol{z}}_i)^2}\right] &= \frac{\lambda}{1-\lambda^2}(n-1),\end{aligned} \tag{15}$$

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

$$\begin{aligned}\mathbb{E}_{\tilde{\boldsymbol{z}}_i}\left[\sqrt{n(\alpha\tilde{\boldsymbol{z}}_i)^2}\right] &= \frac{\lambda}{1-\lambda^2}(n-1) \\ \Rightarrow \alpha &= \frac{\lambda(n-1)}{(1-\lambda^2)\sqrt{n}\,\mathbb{E}[|\tilde{\boldsymbol{z}}_i|]}.\end{aligned} \tag{16}$$

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

We also need to initialize the class weight vectors, $\{\tilde{\boldsymbol{w}}_j\}$, such that their expected ℓ_2-norm matches that of $\alpha\tilde{\boldsymbol{z}}$. Let's denote $\tilde{\boldsymbol{w}}_{j,i}$ as the ith element of the weight vector for class j. Assume each element, $\tilde{\boldsymbol{w}}_{j,i}\ \forall\ i = 1 \dots n$ is independently and identically distributed according to a Gaussian with zero mean and unknown standard deviation. Borrowing Equation 16:

$$\alpha\mathbb{E}[|\tilde{\boldsymbol{w}}_{j,i}|] = \frac{\lambda(n-1)}{(1-\lambda^2)\sqrt{n}}. \tag{17}$$

Note that $\mathbb{E}[|\tilde{\boldsymbol{w}}_{j,i}|] = \sqrt{\frac{2}{\pi}}\sigma$ for a zero-mean Gaussian. Thus:

$$\sigma = \frac{\lambda(n-1)}{(1-\lambda^2)\sqrt{n}}\sqrt{\frac{\pi}{2}}\frac{1}{\alpha}. \tag{18}$$

We empirically determined that $\alpha = \sqrt{\frac{\pi}{2}}$ produced a near-perfect fit for 8D embedding spaces and larger (see Figure 2) resulting in:

$$\tilde{\boldsymbol{w}}_{j,i} \sim \mathcal{N}\left(\mu = 0, \sigma = \frac{\lambda(n-1)}{(1-\lambda^2)\sqrt{n}}\right)\ \forall\ j = 1 \dots Y \text{ and } i = 1 \dots n. \tag{19}$$

For training vMF, λ is a hyperparameter. See Section D.3 for the values we used.

D. Experimental Details

D.1. Architectures

MNIST & FashionMNIST. Experiments on MNIST and FashionMNIST use an architecture adapted from [41]. The architecture has two convolutional blocks each with a convolutional layer followed by batch normalization, ReLU, and 2×2 max-pooling. The first convolutional layer has a 5×5 kernel, 6 filters, zero-padding of length 2, and a stride of 1. The second convolutional layer is identical to the first, but with 16 filters. After the second convolutional block, the representation is flattened and passed through a fully-connected layer with 120 units, followed by batch normalization and a ReLU. The representation is finally passed through two fully-connected layers, the first from 120 units to n units, where n is the dimensionality of the embedding space, and then from n units to Y units, where Y is the total number of classes in the training set. The last fully-connected layer has no bias parameters. All biases in the network are initialized to zero and all weights are initialized with Xavier uniform. For vMF, instead of using Xavier uniform, the weights in the final fully-connected classification layer parameterize vMF distributiuons for each class and thus are initialized using the scheme detailed in Appendix C. For both MNIST and FashionMNIST, $n = 3$ and $Y = 10$.

CIFARs & Open-Set. Experiments on CIFAR10, CIFAR100, Cars196, CUB200-2011, and SOP use a ResNet50 [22]. For CIFAR10 and CIFAR100, the first convolutional layer uses a 3×3 kernel instead of 7×7. For Cars196, CUB200-2011, and SOP, the network is initialized using weights pre-trained on ImageNet and all batch-normalization parameters are frozen. We remove the head of the architecture and add two fully-connected layers directly following the global-average-pooling operation. The first fully-connected layer maps from 2048 units to n units, and the second fully-connected layer maps from n units to Y units. The last fully-connected layer has no bias parameters and for vMF, is initialized using the scheme detailed in Appendix C. For CIFAR10 and CIFAR100, $n = 128$, and $Y = 10$ and $Y = 100$, respectively. For Cars196, CUB200-2011, and SOP, $n = 512$, and $Y = 83$, $Y = 85$, and $Y = 9620$, respectively.

D.2. Datasets

Below we detail data preprocessing, data augmentation, and batch sampling for each of the datasets. We follow the preprocessing and augmentation procedures of [4] for Cars196, CUB200-2011, and SOP. For batch sampling during training, we use an episodic scheme where we first sample N classes at random and then sample K instances from each of the N classes.

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

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

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

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

SOP During training, images are resized to 256×256, a crop of random size in $[0.16, 1.0]$ of the original size and random location is taken with aspect ratio selected randomly in $[0.75, 1.33]$, and resized to 224×224. Finally, the image is randomly flipped horizontally and z-score normalized. During validation and testing, images are resized to 256×256, center-cropped to size 224×224, and z-score normalized. The train set contains half of the classes and the test contains the other half. The validation set is created by splitting off 15% of the train classes. For batch sampling, $N = 32$ and $K = 2$.

D.3. Hyperparameters

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

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

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

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

Table 5. Hyperparameter values for MNIST.

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

Table 6. Hyperparameter values for FashionMNIST.

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

Table 7. Hyperparameter values for CIFAR10.

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

Table 8. Hyperparameter values for CIFAR100.

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

Table 9. Hyperparameter values for Cars196.

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

Table 10. Hyperparameter values for CUB200-2011.

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

Table 11. Hyperparameter values for SOP.

E. Temperature Details

We parameterize the inverse-temperature, β, as $\beta = \exp(\tau)$ where $\tau \in \mathbb{R}$ is an unconstrained network parameter learned automatically via gradient descent.

For fixed-set classification tasks, the network is initialized randomly. We find that fixing β to a large value adversely affects the training dynamics, as small improvements in angular separation can lead to drastic reductions in the loss caused by the peakedness of the posterior (see the reduction in performance for a fixed value of β in the left frame of Figure 5). A challenge in learning τ directly is the network can cheat by maximizing it within the first several epochs. Our parameterization prevents "early cheating" since smaller values of τ result in smaller gradients. For all fixed-set tasks, we initialize $\tau = 0$. By using a smaller learning rate for τ, the network is initially incentivized to focus on the angular discrimination between classes.

The backbone for the open-set tasks is a ResNet50 pretrained on ImageNet. Because the network reuses previously learned features, we found initializing τ to a larger value sometimes results in improved performance. See the hyperparameter tables above for Cars196, CUB200-2011, and SOP for the initial values of τ.

F. Computing Infrastructure

Each model was trained using a single NVIDIA Tesla v100 on Google Cloud Platform. Code was implemented using PyTorch v1.6.0 and Python 3.8.2 on Ubuntu 18.04.

G. Runtime of the von Mises–Fisher Loss

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

H. Cars196 Test Images

Below we show paired grids of Cars196 test images for each of the losses where the left grid contains images corresponding to the smallest $\|\boldsymbol{z}\|$ and the right grid corresponding to the largest $\|\boldsymbol{z}\|$. The provided figures are analogous to Figure 3. Note that for VMF, $\|\tilde{\boldsymbol{z}}\| = \kappa_{\boldsymbol{z}}$.

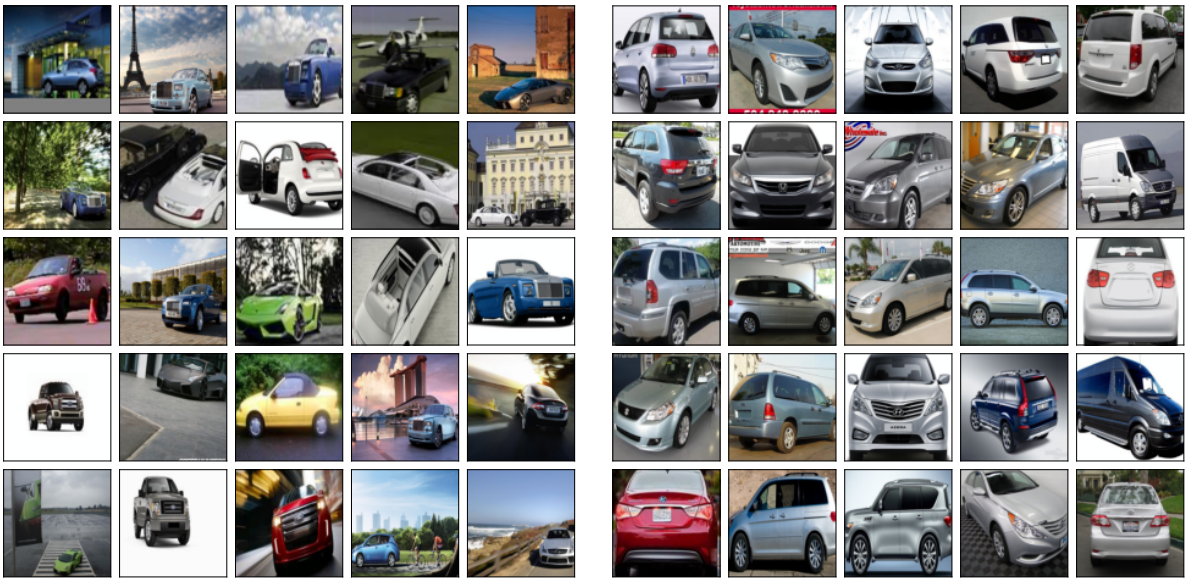

Figure 6. STANDARD embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

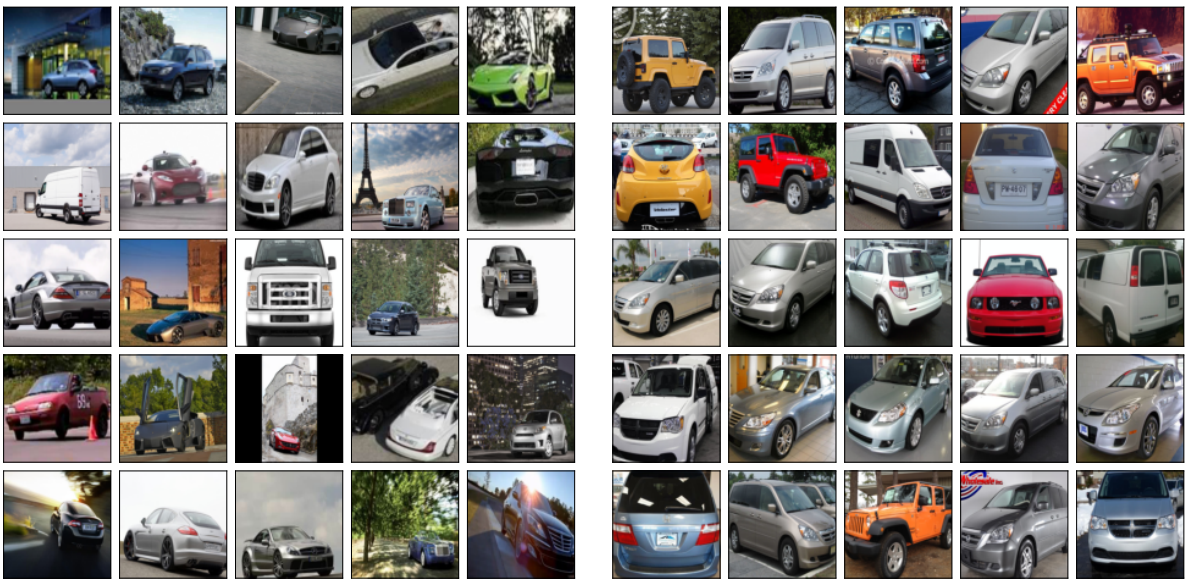

Figure 7. HYPERBOLIC embeddings with the (left) smallest $\|z\|$ and (right) largest $\|z\|$.

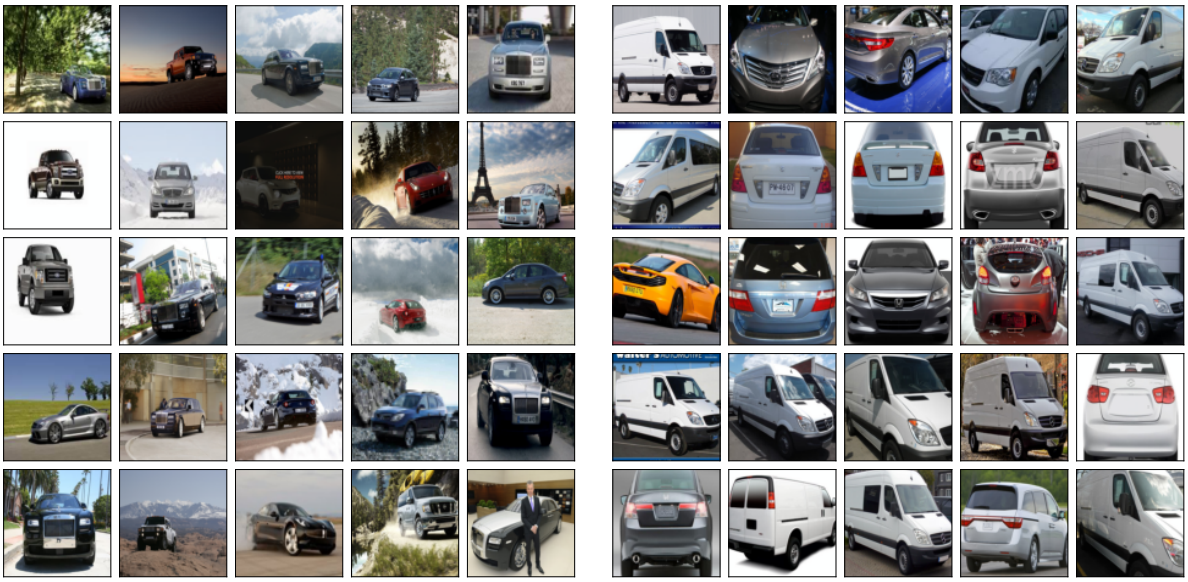

Figure 8. COSINE embeddings with the (left) smallest $\|z\|$ and (right) largest $\|z\|$.

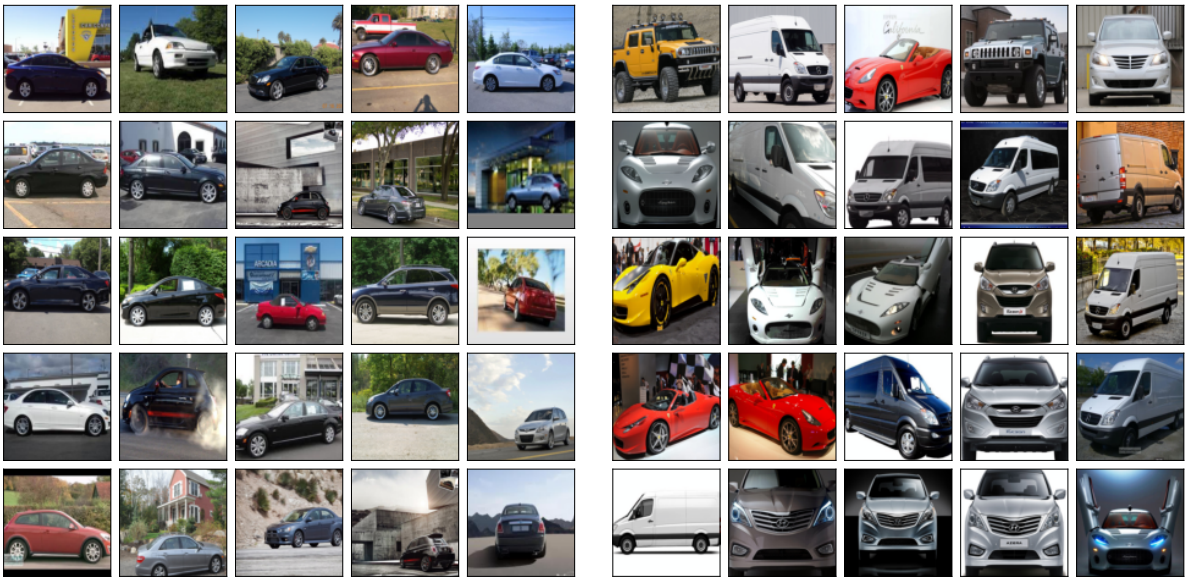

Figure 9. ARCFACE embeddings with the (left) smallest $\|\boldsymbol{z}\|$ and (right) largest $\|\boldsymbol{z}\|$.

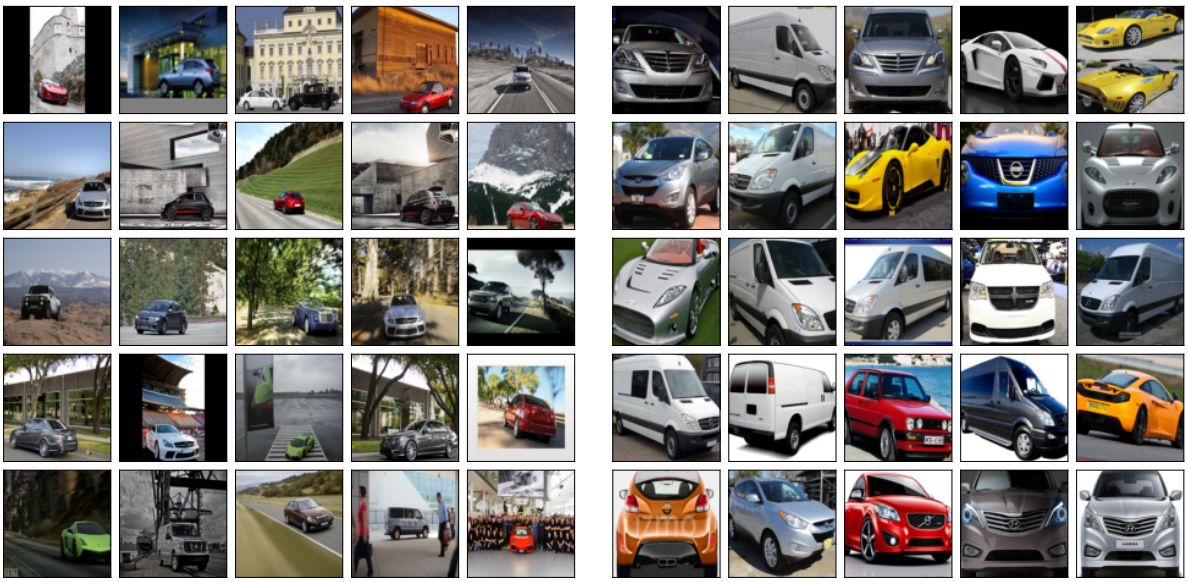

Figure 10. vMF embeddings with the (left) smallest $\kappa_{\boldsymbol{z}}$ and (right) largest $\kappa_{\boldsymbol{z}}$.