It Takes Two to Tango: Mixup for Deep Metric Learning

Shashanka Venkataramanan[1] Bill Psomas[2] Yannis Avrithis[1] Ewa Kijak[1]
Laurent Amsaleg[1] Konstantinos Karantzalos[2]
[1]Inria, Univ Rennes, CNRS, IRISA
[2]National Technical University of Athens

Abstract

Metric learning involves learning a discriminative representation such that embeddings of similar classes are encouraged to be close, while embeddings of dissimilar classes are pushed far apart. State-of-the-art methods focus mostly on sophisticated loss functions or mining strategies. On the one hand, *metric learning losses* consider two or more examples at a time. On the other hand, modern *data augmentation* methods for *classification* consider two or more examples at a time. The combination of the two ideas is under-studied.

In this work, we aim to bridge this gap and improve representations using *mixup*, which is a powerful data augmentation approach interpolating two or more examples and corresponding target labels at a time. This task is challenging because unlike classification, the loss functions used in metric learning are not additive over examples, so the idea of interpolating target labels is not straightforward. To the best of our knowledge, we are the first to investigate mixing examples and target labels for deep metric learning. We develop a generalized formulation that encompasses existing metric learning loss functions and modify it to accommodate for mixup, introducing *Metric Mix*, or *Metrix*. We show that mixing inputs, intermediate representations or embeddings along with target labels significantly improves representations and outperforms state-of-the-art metric learning methods on four benchmark datasets.

1 Introduction

Classification is one of the most studied tasks in machine learning and deep learning. It is a common source of pre-trained models for *transfer learning* to other tasks [7, 22]. It has been studied under different *supervision settings* [3, 39], *knowledge transfer* [16] and *data augmentation* [5], including the recent research line on *mixup* [43, 55], where embeddings and labels are interpolated.

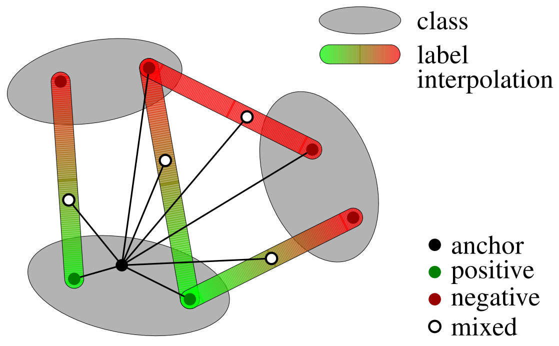

Figure 1: *Metrix* (= *Metric Mix*) allows an anchor to interact with positive (same class), negative (different class) and interpolated examples, which also have interpolated labels.

Deep metric learning is about learning from pairwise interactions such that inference relies on instance embeddings, *e.g.* for *nearest neighbor classification* [30], *instance-level retrieval* [9], *few-shot learning* [44], *face recognition* [37] and *semantic textual similarity* [33]. Following [50], it is most often fully supervised by one class label per example, similar to classification. The two mostly studied problems are *loss functions* [29] and *hard example mining* [34, 49]. In fact, tuple-based losses with example weighting [47] can play the role of both.

Unlike classification, classes (and distributions) at training and inference are different in metric learning. Thus, one might expect interpolation-based data augmentation like mixup to be even more important in metric learning than in classification. Yet, recent attempts are mostly limited to special cases of embedding interpolation and have trouble with label interpolation [21]. This raises the question: *what is a proper way to define and interpolate labels for metric learning?*

In this work, we observe that metric learning is no different from classification, where examples are replaced by pairs of examples and class labels by "positive" or "negative", according to whether class labels of individual examples are the same or not. We say positive/negative pairs or positive/negative examples for another example that we call an *anchor*. Then, as shown in Figure 1, a straightforward way is to use a *binary* (two class) label per pair and interpolate it linearly as in standard mixup. We call our method *Metric Mix*, or *Metrix* for short.

In summary, we make the following contributions:

1. We define a generic way of representing and interpolating labels, which allows straightforward extension of any kind of mixup to deep metric learning for a large class of loss functions. We develop our method on a generic formulation that encapsulates these functions.
2. We define the "positivity" of a mixed example and we study precisely how it increases as a function of the interpolation factor, both in theory and empirically.
3. We systematically evaluate mixup for deep metric learning under different settings, including mixup at different representation levels (input/manifold), mixup of different pairs of examples (anchors/positives/negatives), loss functions and hard example mining.
4. We improve the state of the art on four common metric learning benchmarks.

2 Related Work

Metric learning Metric learning aims to learn a metric such that *positive* pairs of examples are nearby and *negative* ones are far away. In *deep metric learning*, we learn an explicit non-linear mapping from raw input to a low-dimensional *embedding space* [30], where the Euclidean distance has the desired properties. Although learning can be unsupervised [12], deep metric learning has mostly followed the supervised approach, where positive and negative pairs are defined as having the same or different class label, respectively [50].

Loss functions can be distinguished into pair-based and proxy-based [29]. *Pair-based* losses use pairs of examples [12, 12, 49], which can be defined over triplets [15, 37, 46, 48], quadruples [4] or tuples [30, 38, 47]. *Proxy-based* losses use one or more proxies per class, which are learnable parameters in the embedding space [20, 28, 31, 40, 57]. Pair-based losses capture data-to-data relations, but they are sensitive to noisy labels and outliers. They often involve terms where given constraints are satisfied, which produce zero gradients and do not contribute to training. This necessitates *mining* of hard examples that violate the constraints, like semi-hard [37] and distance weighted [49]. By contrast, proxy-based losses use data-to-proxy relations, assuming proxies can capture the global structure of the embedding space. They involve less computations that are more likely to produce nonzero gradient, hence have less or no dependence on mining and converge faster.

Mixup *Input mixup* [55] linearly interpolates between two or more examples in the input space for data augmentation. Numerous variants take advantage of the structure of the input space to interpolate non-linearly, *e.g.* for images [6, 14, 18, 19, 32, 41, 53]. *Manifold mixup* [43] interpolates intermediate representations instead, where the structure is learned. This can be applied to or assisted by decoding back to the input space [1, 2, 25, 42, 56]. In both cases, corresponding labels are linearly interpolated too. Most studies are limited to cross-entropy loss for classification. Pairwise loss functions have been under-studied, as discussed below.

Interpolation for pairwise loss functions As discussed in subsection 3.3, interpolating target labels is not straightforward in pairwise loss functions. In *deep metric learning*, *embedding expansion* [21] and *symmetrical synthesis* [10] interpolate pairs of embeddings in a deterministic way within the same class, applying to pair-based losses, while *proxy synthesis* [11] interpolates between classes, applying to proxy-based losses. None performs label interpolation, which means that [11] risks synthesizing false negatives when the interpolation factor λ is close to 0 or 1.

In *contrastive representation learning*, MoCHi [17] interpolates anchor with negative embeddings but not labels and chooses $\lambda \in [0, 0.5]$ to avoid false negatives. This resembles thresholding of λ at 0.5 in OptTransMix [56]. Finally, *i-mix* [24] interpolates pairs of anchor embeddings as well as their (virtual) class labels linearly. There is only one positive, while all negatives are clean, so it cannot take advantage of interpolation for relative weighting of positives/negatives per anchor [47].

By contrast, our method is developed for deep metric learning and applies to a large class of both pair-based and proxy-based losses. It can interpolate inputs, intermediate features or embeddings of anchors, (multiple) positives or negatives *and* the corresponding two-class (positive/negative) labels per anchor, such that relative weighting of positives/negatives depends on interpolation.

3 Mixup for metric learning

3.1 Preliminaries

Problem formulation We are given a training set $X \subset \mathcal{X}$, where $\mathcal{X}$ is the input space. For each *anchor* $a \in X$, we are also given a set $P(a) \subset X$ of *positives* and a set $N(a) \subset X$ of *negatives*. The positives are typically examples that belong to the same class as the anchor, while negatives belong to a different class. The objective is to train the parameters θ of a model $f : X \to \mathbb{R}^d$ that maps input examples to a d-dimensional *embedding*, such that positives are close to the anchor and negatives are far away in the embedding space. Given two examples $x, x' \in \mathcal{X}$, we denote by $s(x, x')$ the *similarity* between x, x' in the embedding space, typically a decreasing function of Euclidean distance. It is common to ℓ_2-normalize embeddings and define $s(x, x') := \langle f(x), f(x')\rangle$, which is the *cosine similarity*. To simplify notation, we drop the dependence of f, s on θ.

Pair-based losses [12, 30, 46, 47] use both anchors and positives/negatives in X, as discussed above. *Proxy-based* losses define one or more learnable *proxies* in $\mathbb{R}^d$ per class and only use proxies as anchors [20] or positives/negatives [28, 31, 40]. To accommodate for uniform exposition, we extend the definition of similarity as $s(v, x) := \langle v, f(x)\rangle$ for $v \in \mathbb{R}^d, x \in \mathcal{X}$ (proxy anchors) and $s(x, v) := \langle f(x), v\rangle$ for $x \in \mathcal{X}, v \in \mathbb{R}^d$ (proxy positives/negatives). Finally, to accommodate for mixed embeddings in subsection 3.5, we define $s(v, v') := \langle v, v'\rangle$ for $v, v' \in \mathbb{R}^d$. Thus, we define $s : (\mathcal{X} \cup \mathbb{R}^d)^2 \to \mathbb{R}$ over pairs of either inputs in $\mathcal{X}$ or embeddings in $\mathbb{R}^d$. We discuss a few representative loss functions below, before deriving a generic form.

Contrastive The contrastive loss [12] encourages positive examples to be pulled towards the anchor and negative examples to be pushed away by a margin $m \in \mathbb{R}$. This loss is *additive* over positives and negatives, defined as

$$\ell_{\text{cont}}(a; \theta) := \sum_{p \in P(a)} -s(a, p) + \sum_{n \in N(a)} [s(a, n) - m]_+. \tag{1}$$

Multi-Similarity The multi-similarity loss [47] introduces *relative weighting* to encourage positives (negatives) that are farthest from (closest to) the anchor to be pulled towards (pushed away from) the anchor by a higher weight. This loss is *not* additive over positives and negatives:

$$\ell_{\text{MS}}(a; \theta) := \frac{1}{\beta} \log \left(1 + \sum_{p \in P(a)} e^{-\beta(s(a,p)-m)}\right) + \frac{1}{\gamma} \log \left(1 + \sum_{n \in N(a)} e^{\gamma(s(a,n)-m)}\right). \tag{2}$$

Here, $\beta, \gamma \in \mathbb{R}$ are scaling factors for positives, negatives respectively.

Proxy Anchor The proxy anchor loss [20] defines a learnable *proxy* in $\mathbb{R}^d$ for each class and only uses proxies as anchors. For a given anchor (proxy) $a \in \mathbb{R}^d$, the loss has the same form as (2), although similarity s is evaluated on $\mathbb{R}^d \times \mathcal{X}$.

3.2 Generic loss formulation

We observe that both additive (1) and non-additive (2) loss functions involve a sum over positives $P(a)$ and a sum over negatives $N(a)$. They also involve a decreasing function of similarity $s(a, p)$

Loss	Anchor	Pos/Neg	$\tau(x)$	$\sigma^+(x)$	$\sigma^-(x)$	$\rho^+(x)$	$\rho^-(x)$
Contrastive [12]	X	X	x	x	x	$-x$	$[x-m]_+$
Lifted structure [15]	X	X	$[x]_+$	$\log(x)$	$\log(x)$	e^{-x}	e^{x-m}
Binomial deviance [52]	X	X	x	$\log(1+x)$	$\log(1+x)$	$e^{-\beta(x-m)}$	$e^{\gamma(x-m)}$
Multi-similarity [47]	X	X	x	$\frac{1}{\beta}\log(1+x)$	$\frac{1}{\gamma}\log(1+x)$	$e^{-\beta(x-m)}$	$e^{\gamma(x-m)}$
Proxy anchor [20]	proxy	X	x	$\frac{1}{\beta}\log(1+x)$	$\frac{1}{\gamma}\log(1+x)$	$e^{-\beta(x-m)}$	$e^{\gamma(x-m)}$
NCA [8]	X	X	x	$-\log(x)$	$\log(x)$	e^x	e^x
ProxyNCA [28]	X	proxy	x	$-\log(x)$	$\log(x)$	e^x	e^x
ProxyNCA++ [40]	X	proxy	x	$-\log(x)$	$\log(x)$	$e^{x/T}$	$e^{x/T}$

Table 1: Loss functions. Anchor/positive/negative: X: embedding of input example from training set X by f; proxy: learnable parameter in $\mathbb{R}^d$; T: temperature. All loss functions are encompassed by (3) using the appropriate definition of functions $\tau, \sigma^+, \sigma^-, \rho^+, \rho^-$ as given here.

for each positive $p \in P(a)$ and an increasing function of similarity $s(a, n)$ for each negative $n \in N(a)$. Let us denote by ρ^+, ρ^- this function for positives, negatives respectively. Then, non-additive functions differ from additive by the use of a nonlinear function σ^+, σ^- on positive and negative terms respectively, as well as possibly another nonlinear function τ on their sum:

$$\ell(a;\theta) := \tau\left(\sigma^+\left(\sum_{p\in P(a)} \rho^+(s(a,p))\right) + \sigma^-\left(\sum_{n\in N(a)} \rho^-(s(a,n))\right)\right). \tag{3}$$

With the appropriate choice for $\tau, \sigma^+, \sigma^-, \rho^+, \rho^-$, this definition encompasses contrastive (1), multi-similarity (2) or proxy-anchor as well as many pair-based or proxy-based loss functions, as shown in Table 1. It does not encompass the *triplet loss* [46], which operates on pairs of positives and negatives, forming triplets with the anchor. The triplet loss is the most challenging in terms of mining because there is a very large number of pairs and only few contribute to the loss. We only use function τ to accommodate for *lifted structure* [15, 30], where $\tau(x) := [x]_+$ is reminiscent of the triplet loss. We observe that multi-similarity [47] differs from *binomial deviance* [52] only in the weights of the positive and negative terms. Proxy anchor [20] is a proxy version of multi-similarity [47] on anchors and ProxyNCA [28] is a proxy version of NCA [8] on positives/negatives.

This generic formulation highlights the components of the loss functions that are additive over positives/negatives and paves the way towards incorporating mixup.

3.3 Improving representations using mixup

To improve the learned representations, we follow [43, 55] in mixing inputs and features from intermediate network layers, respectively. Both are developed for classification.

Input mixup [55] augments data by linear interpolation between a pair of input examples. Given two examples $x, x' \in \mathcal{X}$, we draw $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$ as *interpolation factor* and mix x with x' using the standard mixup operation $\mathrm{mix}_\lambda(x, x') := \lambda x + (1-\lambda)x'$.

Manifold mixup [43] linearly interpolates between intermediate representations (features) of the network instead. Referring to 2D images, we define $g_m : X \to \mathbb{R}^{c\times w\times h}$ as the mapping from the input to intermediate layer m of the network and $f_m : \mathbb{R}^{c\times w\times h} \to \mathbb{R}^d$ as the mapping from intermediate layer m to the embedding, where c is the number of channels (feature dimensions) and $w \times h$ is the spatial resolution. Thus, our model f can be expressed as the composition $f = g_m \circ f_m$.

For manifold mixup, we follow [42] and mix either features of intermediate layer m or the final embeddings. Thus, we define three *mixup types* in total:

$$f_\lambda(x, x') := \begin{cases} f(\mathrm{mix}_\lambda(x, x')), & \text{input mixup} \\ f_m(\mathrm{mix}_\lambda(g_m(x), g_m(x'))), & \text{feature mixup} \\ \mathrm{mix}_\lambda(f(x), f(x')), & \text{embedding mixup.} \end{cases} \tag{4}$$

Function $f_\lambda : \mathcal{X}^2 \to \mathbb{R}^d$ performs both mixup and embedding. We explore different mixup types in subsection 4.4.

3.4 Label representation

Classification In supervised classification, each example $x \in X$ is assigned an one-hot encoded label $y \in \{0,1\}^C$, where C is the number of classes. Label vectors are also linearly interpolated: Given two labeled examples $(x,y),(x',y')$, the interpolated label is $\operatorname{mix}_\lambda(y,y')$. The loss (cross-entropy) is a continuous function of the label vector. We extend this idea to metric learning.

Metric learning Positives $P(a)$ and negatives $N(a)$ of anchor a are defined as having the same or different class label as the anchor, respectively. To every example in $P(a) \cup N(a)$, we assign a binary (two-class) label $y \in \{0,1\}$, such that $y = 1$ for positives and $y = 0$ for negatives:

$$U^+(a) := \{(p,1) : p \in P(a)\} \tag{5}$$

$$U^-(a) := \{(n,0) : n \in N(a)\} \tag{6}$$

Thus, we represent both positives and negatives by $U(a) := U^+(a) \cup U^-(a)$. We now rewrite the generic loss function (3) as

$$\ell(a;\theta) := \tau\left(\sigma^+\left(\sum_{(x,y)\in U(a)} y\rho^+(s(a,x))\right) + \sigma^-\left(\sum_{(x,y)\in U(a)} (1-y)\rho^-(s(a,x))\right)\right). \tag{7}$$

Here, every labeled example (x,y) in $U(a)$ appears in both positive and negative terms. However, because label y is binary, only one of the two contributions is nonzero. Now, in the presence of mixup, we can linearly interpolate labels exactly as in classification.

3.5 Mixed loss function

Mixup For every anchor a, we are given a set $M(a)$ of pairs of examples to mix. This is a subset of $(S(a) \cup U(a)) \times U(a)$ where $S(a) := (a,1)$. That is, we allow mixing between positive-negative, positive-positive and negative-negative pairs, where the anchor itself is also seen as positive. We define the possible choices of *mixing pairs* $M(a)$ in subsection 4.1 and we assess them in subsection 4.4. Let $V(a)$ be the set of corresponding *labeled mixed embeddings*

$$V(a) := \{(f_\lambda(x,x'), \operatorname{mix}_\lambda(y,y')) : ((x,y),(x',y')) \in M(a), \lambda \sim \operatorname{Beta}(\alpha,\alpha)\}, \tag{8}$$

where f_λ is defined by (4). With these definitions in place, the generic loss function $\widetilde{\ell}$ over mixed examples takes exactly the same form as (7), with only $U(a)$ replaced by $V(a)$:

$$\widetilde{\ell}(a;\theta) := \tau\left(\sigma^+\left(\sum_{(v,y)\in V(a)} y\rho^+(s(a,v))\right) + \sigma^-\left(\sum_{(v,y)\in V(a)} (1-y)\rho^-(s(a,v))\right)\right), \tag{9}$$

where similarity s is evaluated on $\mathcal{X} \times \mathbb{R}^d$ for pair-based losses and $\mathbb{R}^d \times \mathbb{R}^d$ for proxy anchor. Now, every labeled embedding (v,y) in $V(a)$ appears in both positive and negative terms and *both* contributions are nonzero for positive-negative pairs, because after interpolation, $y \in [0,1]$.

Error function Parameters θ are learned by minimizing the error function, which is a linear combination of the *clean loss* (3) and the *mixed loss* (9), averaged over all anchors

$$E(X;\theta) := \frac{1}{|X|}\sum_{a\in X} \ell(a;\theta) + w\widetilde{\ell}(a;\theta), \tag{10}$$

where $w \geq 0$ is the *mixing strength*. At least for manifold mixup, this combination comes at little additional cost, since clean embeddings are readily available.

3.6 Analysis: Mixed embeddings and positivity

Let $\operatorname{Pos}(a,v)$ be the event that a mixed embedding v behaves as "positive" for anchor a, *i.e.*, minimizing the loss $\widetilde{\ell}(a;\theta)$ will increase the similarity $s(a,v)$. In subsection A.2, we explain that this "positivity" is equivalent to $\partial\widetilde{\ell}(a;\theta)/\partial s(a,v) \leq 0$. Under positive-negative mixing, *i.e.*,

$M(a) \subset U^+(a) \times U^-(a)$, we then estimate the probability of $\text{Pos}(a, v)$ as a function of λ in the case of multi-similarity (2) with a single mixed embedding v:

$$\text{P}(\text{Pos}(a, v)) = F_\lambda \left(\frac{1}{\beta + \gamma} \ln \left(\frac{\lambda}{1 - \lambda} \right) + m \right), \tag{11}$$

where F_λ is the CDF of similarities $s(a, v)$ between anchors a and mixed embeddings v with interpolation factor λ. In Figure 2, we measure the probability of $\text{Pos}(a, v)$ as a function of λ in two ways, both purely empirically and theoretically by (11). Both measurements are increasing functions of λ of sigmoidal shape and they confirm that a mixed embedding is mostly positive for λ close to 1 and mostly negative for λ close to 0.

4 Experiments

4.1 Setup

Datasets We experiment on Caltech-UCSD Birds (CUB200) [45], Stanford Cars (Cars196) [23], Stanford Online Products (SOP) [30] and In-Shop Clothing retrieval (In-Shop) [26] image datasets. More details are in subsection B.1.

Network, features and embeddings We use Resnet-50 [13] (R-50) pre-trained on ImageNet [35] as a backbone network. We obtain the intermediate representation (*feature*), a $7 \times 7 \times 2048$ tensor, from the last convolutional layer. Following [20], we combine adaptive average pooling with max pooling, followed by a fully-connected layer to obtain the *embedding* of $d = 512$ dimensions.

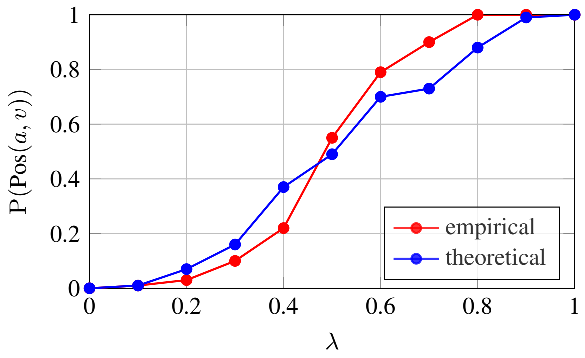

Figure 2: *"Positivity" of mixed embeddings* vs. λ. We measure $\text{P}(\text{Pos}(a, v))$ empirically as $\text{P}(\partial \widetilde{\ell}_{\text{MS}}(a; \theta) / \partial s(a, v) \leq 0)$ and theoretically by (11), where F_λ is again measured from data. We use embedding mixup on MS (2) on CUB200 at epoch 0, according to the setup of subsection 4.1.

Loss functions We reproduce *contrastive* (Cont) [12], *multi-similarity* (MS) [47] and *proxy anchor* (PA) [20] and we evaluate them under different mixup types. For MS (2), we use $\beta = 18$, $\gamma = 75$ and $m = 0.77$. For PA, we use $\beta = \gamma = 32$ and $m = 0.1$. As baselines, we reproduce and compare with *triplet* [48], *lifted structure* [30], ProxyNCA [28], *margin* [49] and SoftTriple [31] losses, without mixup. By reporting published results, we also compare with D&C [36], EPSHN [51] and ProxyNCA++ [40] on the four datasets and PA [20] on CUB200 and Cars196. Details on training are in subsection B.1.

Methods We compare our method, *Metrix*, with *proxy synthesis* (PS) [11] and MoChi [17]. For PS, we adapt the official code[1] to PA on all datasets, and use it with PA only, because it is designed for proxy-based losses. PS has been shown superior to [10, 21], although in different networks. MoChi and *i-mix* [24] are meant for contrastive representation learning, but we do compare with our reproduction of MoChi. We evaluate using Recall@K [30].

4.2 Mixup settings

In mixup for classification, given a batch of n examples, it is standard to form n pairs of examples by pairing the batch with a *random permutation* of itself, resulting in n mixed examples, either for input or manifold mixup. In metric learning, it is common to obtain n embeddings and then use all $\frac{1}{2}n(n-1)$ pairs of embeddings in computing the loss. We thus treat mixup types differently.

Input mixup Mixing all pairs would be computationally expensive in this case, because we would compute $\frac{1}{2}n(n-1)$ embeddings. A random permutation would not produce as many hard examples

[1]https://github.com/navervision/proxy-synthesis

as can be found in all pairs. Thus, for each anchor (each example in the batch), we use the k *hardest negative* examples and mix them with positives or with the anchor. We use $k = 3$ by default.

Manifold mixup Originally, manifold mixup [43] focuses on the *first* few layers of the network. Mixing all pairs would then be even more expensive than input mixup, because intermediate features (tensors) are even larger than input examples. Hence, we focus on the *last* few layers instead, where features and embeddings are compact, and we mix all pairs. We use feature mixup by default and call it *Metrix/feature* or just *Metrix*, while input and embedding mixup are called *Metrix/input* and *Metrix/embed*, respectively. All options are studied in subsection 4.4.

Mixing pairs Whatever the mixup type, we use clean examples as anchors and we define a set $M(a)$ of pairs of examples to mix for each anchor a, with their labels (positive or negative). By default, we mix positive-negative or anchor-negative pairs, according to $M(a) := U^+(a) \times U^-(a)$ and $M(a) := S(a) \times U^-(a)$, respectively, where $U^-(a)$ is replaced by hard negatives only for input mixup. The two options are combined by choosing uniformly at random in each iteration. More options are studied in subsection 4.4.

Hyper-parameters For any given mixup type or set of mixup pairs, the interpolation factor λ is drawn from $\text{Beta}(\alpha, \alpha)$ with $\alpha = 2$. We empirically set the mixup strength (10) to $w = 0.4$ for positive-negative pairs and $w = 0.3$ for anchor-negative pairs.

4.3 Results

Improving the state of the art As shown in Table 2, Metrix consistently improves the performance of all baseline losses (Cont, MS, PA) across all datasets. More results in subsection B.2 reveal that the same is true for Metrix/input and Metrix/embed too. Surprisingly, while baseline PA outperforms MS, MS outperforms PA under mixup on all datasets but SOP, where the two losses are on par. Both contrastive and MS are significantly improved by mixup. By contrast, improvements on PA are marginal, which may be due to the already strong performance of PA, or further improvement is possible by employing different mixup methods that take advantage of the image structure.

In terms of Recall@1, our MS+Metrix is best overall, improving by 3.6% ($67.8 \to 71.4$) on CUB200, 1.8% ($87.8 \to 89.6$) on Cars196, 4.1% ($76.9 \to 81.0$) on SOP and 2.1% ($90.1 \to 92.2$) on In-Shop. The same solution sets new state of the art, outperforming the previously best PA by 1.7% ($69.7 \to 71.4$) on CUB200, MS by 1.8% ($87.8 \to 89.6$) on Cars196, ProxyNCA++ by 0.3% ($80.7 \to 81.0$) on SOP and SoftTriple by 1.2% ($91.0 \to 92.2$) on In-Shop. Importantly, while the previous state of the art comes from a different loss per dataset, MS+Metrix is almost consistently best across all datasets.

Alternative mixing methods In Table 3, we compare our Metrix/embed with MoCHi [17] using contrastive loss and with PS [11] using PA on Cars196. Both mix embeddings only, while labels are always negative. In MoCHi, the anchor is clean and we mix negative-negative ($U^-(a)^2$) and anchor-negative ($S(a) \times U^-(a)$) pairs, where $U^-(a)$ is replaced by $k = 100$ hardest negatives and $\lambda \in (0, 0.5)$ for anchor-negative. PS mixes embeddings of different classes and treat them as new classes. For clean anchors, this corresponds to positive-negative ($U^+(a) \times U^-(a)$) and negative-negative ($U^-(a)^2$) mixing pairs, but PS also supports mixed anchors.

In terms of Recall@1, our Metrix/embed outperforms MoCHI with anchor-negative pairs by 1.2% ($65.2 \to 66.4$) on CUB200, 1.4% ($82.5 \to 83.9$) on Cars196, 0.9% ($75.8 \to 76.7$) and 1.2% ($87.2 \to 88.4$) on In-Shop. The gain over MoCHi with negative-negative pairs is significantly higher. Metrix/embed also outperforms PS by 0.4% ($70.0 \to 70.4$) on CUB200, 1% ($87.9 \to 88.9$) on Cars196, 1% ($79.6 \to 80.6$) on SOP and 1.3% ($90.3 \to 91.6$) on In-Shop.

4.4 Ablations

We perform ablations on Cars196 using R-50 with $d = 512$, applying mixup on contrastive loss. More ablations are in subsection B.3.

Hard negatives We study the effect of the number k of hard negatives using different mixup types. The set of mixing pairs is chosen from (positive-negative, anchor-negative) uniformly at random per

	CUB200			CARS196			SOP			IN-SHOP		
METHOD	R@1	R@2	R@4	R@1	R@2	R@4	R@1	R@10	R@100	R@1	R@10	R@20
Triplet [48]	63.5	75.6	84.4	77.3	85.4	90.8	70.5	85.6	94.3	85.3	96.6	97.8
LiftedStructure [30]	65.9	75.8	84.5	81.4	88.3	92.4	76.1	88.6	95.2	88.6	97.6	98.4
ProxyNCA [28]	65.2	75.6	83.8	81.2	87.9	92.6	73.2	87.0	94.4	86.2	95.9	97.0
Margin [49]	65.0	76.2	84.6	82.1	88.7	92.7	74.8	87.8	94.8	88.6	97.0	97.8
SoftTriple [31]	67.3	77.7	86.2	86.5	91.9	95.3	79.8	91.2	96.3	**91.0**	97.6	98.3
D&C [36]*	65.9	76.6	84.4	84.6	90.7	94.1	75.9	88.4	94.9	85.7	95.5	96.9
EPSHN [51]*	64.9	75.3	83.5	82.7	89.3	93.0	78.3	90.7	96.3	87.8	95.7	96.8
ProxyNCA++ [40]*	69.0	79.8	**87.3**	86.5	92.5	95.7	**80.7**	**92.0**	**96.7**	90.4	**98.1**	**98.8**
Cont [12]	64.7	75.9	84.6	81.6	88.2	92.7	74.9	87.0	93.9	86.4	94.7	96.2
+Metrix	67.4	77.9	85.7	85.1	91.1	94.6	77.5	89.1	95.5	89.1	95.7	97.1
	+2.7	+2.0	+1.1	+3.5	+2.9	+1.9	+2.6	+2.1	+1.5	+2.7	+1.0	+0.9
MS [47]	67.8	77.8	85.6	**87.8**	92.7	95.3	76.9	89.8	95.9	90.1	97.6	98.4
+Metrix	**71.4**	80.6	86.8	**89.6**	**94.2**	96.0	81.0	**92.0**	**97.2**	**92.2**	**98.5**	98.6
	+3.6	+2.8	+1.2	+1.8	+1.5	+0.7	+4.1	+2.2	+1.3	+2.1	+0.9	+0.2
PA [20]*	**69.7**	**80.0**	87.0	87.7	**92.9**	**95.8**	–	–	–	–	–	–
PA [20]	69.5	79.3	87.0	87.6	92.3	95.5	79.1	90.8	96.2	90.0	97.4	98.2
+Metrix	71.0	**81.8**	**88.2**	89.1	93.6	**96.7**	**81.3**	91.7	96.9	91.9	98.2	**98.8**
	+1.3	+1.8	+1.2	+1.4	+0.7	+0.9	+2.2	+0.9	+0.7	+1.9	+0.8	+0.6
Gain over SOTA	+1.7	+1.8	+0.9	+1.8	+1.3	+0.9	+0.6	+0.0	+0.5	+1.2	+0.4	+0.0

Table 2: *Improving the SOTA with our Metrix* (Metrix/feature) using Resnet-50 with embedding size $d = 512$. R@K (%): Recall@K; higher is better. *: reported by authors. **Bold black**: best baseline (previous SOTA, one per column). **Red**: Our new SOTA. Gain over SOTA is over best baseline. MS: Multi-Similarity, PA: Proxy Anchor. Additional results are in subsection B.2

		CUB200			CARS196			SOP			IN-SHOP		
METHOD	MIXING PAIRS	R@1	R@2	R@4	R@1	R@2	R@4	R@1	R@10	R@100	R@1	R@10	R@20
Cont [12]	–	64.7	75.9	84.6	81.6	88.2	92.7	74.9	87.0	93.9	86.4	94.7	96.3
+MoCHi [17]	neg-neg	63.1	74.3	83.8	76.3	84.0	89.3	68.9	83.1	91.8	81.8	91.9	93.9
+MoCHi [17]	anc-neg	65.2	75.8	84.2	82.5	88.0	92.9	75.8	87.1	94.8	87.2	92.8	94.9
+Metrix/embed	pos-neg / anc-neg	**66.4**	**77.6**	**85.4**	**83.9**	**90.3**	**94.1**	**76.7**	**88.6**	**95.2**	**88.4**	**95.4**	**96.9**
PA [20]	–	69.7	80.0	87.0	87.6	92.3	95.5	79.1	90.8	96.2	90.0	97.4	98.2
+PS [11]	pos-neg / neg-neg	70.0	79.8	87.2	87.9	92.8	95.6	79.6	90.9	96.4	90.3	97.4	98.0
+Metrix/embed	pos-neg / anc-neg	**70.4**	**81.1**	**87.9**	**88.9**	**93.3**	**96.4**	**80.6**	**91.7**	**96.6**	**91.6**	**98.3**	**98.3**

Table 3: *Comparison of our Metrix/embed with other mixing methods* using R-50 with embedding size $d = 512$ on Cars196. R@K (%): Recall@K; higher is better. PA: Proxy Anchor, PS: Proxy Synthesis.

iteration. We choose $k = 3$ for input mixup. For feature/embedding mixup, we mix all pairs in a batch by default, but also study $k \in \{20, 40\}$. As shown in Table 4, $k = 3$ for input and all pairs for feature/embedding mixup works best. Still, using few hard negatives for feature/embedding mixup is on par or outperforms input mixup. All choices significantly outperform the baseline.

Mixing pairs We study the effect of mixing pairs $M(a)$, in particular, $U^+(a)^2$ (positive-positive), $U^+(a) \times U^-(a)$ (positive-negative) and $S(a) \times U^-(a)$ (anchor-negative), again using different mixup types. As shown in Table 4, when using a single set of mixing pairs during training, positive-negative and anchor-negative consistently outperform the baseline, while positive-positive is actually outperformed by the baseline. This may be due to the lack of negatives in the mixed loss (9), despite the presence of negatives in the clean loss (3). Hence, we only use positive-negative and anchor-negative by default, combined by choosing uniformly at random in each iteration.

Mixup types We study the effect of mixup type (input, feature, embedding), when used alone. The set of mixing pairs is chosen from (positive-negative, anchor-negative) uniformly at random per iteration. As shown in both “hard negatives” and “mixing pairs” parts of Table 4, our default feature mixup works best, followed by embedding and input mixup. More results in subsection B.3 reveal that the same is true across all three loss functions and all four datasets considered.

Study	Hard Negatives k	Mixing Pairs	Mixup Type	R@1	R@2	R@4	R@8
baseline				81.6	88.2	92.7	95.8
hard negatives	1	pos-neg / anc-neg	input	82.0	89.1	93.1	96.1
	2	pos-neg / anc-neg	input	82.5	89.2	93.4	96.2
	3	pos-neg / anc-neg	input	82.9	89.3	93.7	95.5
	20	pos-neg / anc-neg	feature	83.5	90.1	94.0	96.5
	40	pos-neg / anc-neg	feature	84.0	90.4	94.2	96.8
	all	pos-neg / anc-neg	feature	**85.1**	**91.1**	**94.6**	**97.0**
	20	pos-neg / anc-neg	embed	82.7	89.2	93.4	96.1
	40	pos-neg / anc-neg	embed	83.0	90.0	93.8	96.4
	all	pos-neg / anc-neg	embed	83.4	89.9	94.1	96.4
mixing pairs	–	pos-pos	input	81.0	88.2	92.6	95.6
	3	pos-neg	input	82.4	89.1	93.3	95.6
	3	anc-neg	input	81.8	89.0	93.6	95.4
	–	pos-pos	feature	81.1	88.3	92.9	95.8
	all	pos-neg	feature	84.0	90.2	94.2	96.6
	all	anc-neg	feature	83.7	90.1	94.4	96.7
	–	pos-pos	embed	78.3	85.7	90.8	94.4
	all	pos-embed	embed	83.1	90.0	93.9	96.6
	all	anc-neg	embed	82.7	89.5	93.5	96.3
mixup type combinations	{1, all}	pos-neg / anc-neg	{input, feature}	83.7	94.2	95.9	96.7
	{3, all}	pos-neg / anc-neg	{input, embed}	83.0	90.9	94.1	96.4
	{all, all}	pos-neg / anc-neg	{feature, embed}	84.7	90.6	94.4	96.9
	{1, all, all}	pos-neg / anc-neg	{input, feature, embed}	**85.3**	**94.9**	**96.2**	**97.1**

Table 4: *Ablation study of our Metrix* using contrastive loss and R-50 with embedding size $d = 512$ on Cars196. R@K (%): Recall@K; higher is better.

Mixup type combinations We study the effect of using more than one mixup type (input, feature, embedding), chosen uniformly at random per iteration. The set of mixing pairs is also chosen from (positive-negative, anchor-negative) uniformly at random per iteration. As shown in Table 4, mixing inputs, features and embeddings works best. Although this solution outperforms feature mixup alone by 0.2% Recall@1 ($85.1 \rightarrow 85.3$), it is computationally expensive because of using input mixup. The next best efficient choice is mixing features and embeddings, which however is worse than mixing features alone (84.7 *vs*. 85.1). This is why we chose feature mixup by default.

5 Conclusion

Based on the argument that metric learning is binary classification of pairs of examples into "positive" and "negative", we have introduced a direct extension of mixup from classification to metric learning. Our formulation is generic, applying to a large class of loss functions that separate positives from negatives per anchor and involve component functions that are additive over examples. Those are exactly loss functions that require less mining. We contribute a principled way of interpolating labels, such that the interpolation factor affects the relative weighting of positives and negatives. Other than that, our approach is completely agnostic with respect to the mixup method, opening the way to using more advanced mixup methods for metric learning.

We consistently outperform baselines using a number of loss functions on a number of benchmarks and we improve the state of the art using a single loss function on all benchmarks, while previous state of the art was not consistent in this respect. Surprisingly, this loss function, multi-similarity [47], is not the state of the art without mixup. Because metric learning is about generalizing to unseen classes and distributions, our work may have applications to other such problems, including transfer learning, few-shot learning and continual learning.

References

[1] C. Beckham, S. Honari, V. Verma, A. Lamb, F. Ghadiri, R. D. Hjelm, Y. Bengio, and C. Pal. On adversarial mixup resynthesis. *arXiv preprint arXiv:1903.02709*, 2019. 2

[2] D. Berthelot, C. Raffel, A. Roy, and I. Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*, 2018. 2

[3] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018. 1

[4] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In *CVPR*, 2017. 2

[5] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 1

[6] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 2

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 1

[8] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2005. 4

[9] A. Gordo, J. Almazán, J. Revaud, and D. Larlus. Deep image retrieval: Learning global representations for image search. In *ECCV*, 2016. 1

[10] G. Gu and B. Ko. Symmetrical synthesis for deep metric learning. In *AAAI*, 2020. 2, 6

[11] G. Gu, B. Ko, and H.-G. Kim. Proxy synthesis: Learning with synthetic classes for deep metric learning. In *AAAI*, 2021. 3, 6, 7, 8

[12] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006. 2, 3, 4, 6, 8, 15

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[14] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *ICLR*, 2020. 2

[15] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017. 2, 4

[16] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1

[17] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus. Hard negative mixing for contrastive learning. *NeurIPS*, 2020. 3, 6, 7, 8

[18] J.-H. Kim, W. Choo, H. Jeong, and H. O. Song. Co-mixup: Saliency guided joint mixup with supermodular diversity. In *ICLR*, 2021. 2

[19] J.-H. Kim, W. Choo, and H. O. Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *ICML*, 2020. 2

[20] S. Kim, D. Kim, M. Cho, and S. Kwak. Proxy anchor loss for deep metric learning. In *CVPR*, 2020. 2, 3, 4, 6, 8, 15

[21] B. Ko and G. Gu. Embedding expansion: Augmentation in embedding space for deep metric learning. In *CVPR*, 2020. 2, 6

[22] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. Big transfer (bit): General visual representation learning. In *ECCV*, 2020. 1

[23] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. *ICCVW*, 2013. 6, 14

[24] K. Lee, Y. Zhu, K. Sohn, C.-L. Li, J. Shin, and H. Lee. I-mix: A domain-agnostic strategy for contrastive representation learning. In *ICLR*, 2021. 3, 6

[25] X. Liu, Y. Zou, L. Kong, Z. Diao, J. Yan, J. Wang, S. Li, P. Jia, and J. You. Data augmentation via latent space interpolation for image classification. In *ICPR*, 2018. 2

[26] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *CVPR*, 2016. 6, 14

[27] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 14

[28] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. In *ICCV*, 2017. 2, 3, 4, 6, 8, 15

[29] K. Musgrave, S. Belongie, and S.-N. Lim. A metric learning reality check. In *ECCV*, 2020. 2

[30] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 1, 2, 3, 4, 6, 8, 14, 15

[31] Q. Qian, L. Shang, B. Sun, J. Hu, H. Li, and R. Jin. Softtriple loss: Deep metric learning without triplet sampling. In *ICCV*, 2019. 2, 3, 6, 8, 15

[32] J. Qin, J. Fang, Q. Zhang, W. Liu, X. Wang, and X. Wang. Resizemix: Mixing data with preserved object information and true labels. *arXiv preprint arXiv:2012.11101*, 2020. 2

[33] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*, 2019. 1

[34] J. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka. Contrastive learning with hard negative samples. *ICLR*, 2021. 2

[35] 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. *IJCV*, 2015. 6

[36] A. Sanakoyeu, V. Tschernezki, U. Buchler, and B. Ommer. Divide and conquer the embedding space for metric learning. In *CVPR*, 2019. 6, 8, 15

[37] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 1, 2

[38] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016. 2

[39] K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020. 1

[40] E. W. Teh, T. DeVries, and G. W. Taylor. Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. In *ECCV*, 2020. 2, 3, 4, 6, 8, 15

[41] A. F. M. Uddin, M. Monira, W. Shin, T. Chung, and S.-H. Bae. Saliencymix: A saliency guided data augmentation strategy for better regularization. In *ICLR*, 2021. 2

[42] S. Venkataramanan, Y. Avrithis, E. Kijak, and L. Amsaleg. Alignmix: Improving representation by interpolating aligned features. *arXiv preprint arXiv:2103.15375*, 2021. 2, 4

[43] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, 2019. 1, 2, 4, 7

[44] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*, 2016. 1

[45] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. *California Institute of Technology*, 2011. 6, 14

[46] 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 *CVPR*, 2014. 2, 3, 4

[47] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *CVPR*, 2019. 2, 3, 4, 6, 8, 9, 15

[48] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 2009. 2, 6, 8, 15

[49] C. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl. Sampling matters in deep embedding learning. In *ICCV*, 2017. 2, 6, 8, 15

[50] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2003. 1, 2

[51] H. Xuan, A. Stylianou, and R. Pless. Improved embeddings with easy positive triplet mining. In *WACV*, 2020. 6, 8, 15

[52] D. Yi, Z. Lei, and S. Z. Li. Deep metric learning for practical person re-identification. *arXiv preprint arXiv:1703.07737*, 2014. 4

[53] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 2

[54] A. Zhai and H.-Y. Wu. Classification is a strong baseline for deep metric learning. *arXiv preprint arXiv:1811.12649*, 2018. 14

[55] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 1, 2, 4

[56] J. Zhu, L. Shi, J. Yan, and H. Zha. Automix: Mixup networks for sample interpolation via cooperative barycenter learning. In *ECCV*, 2020. 2, 3

[57] Y. Zhu, M. Yang, C. Deng, and W. Liu. Fewer is more: A deep graph metric learning perspective using fewer proxies. *NeurIPS*, 2020. 2

A More on the method

A.1 Mixed loss function

Interpretation To better understand the two contributions of a labeled embedding (v, y) in $V(a)$ to the positive and negative terms of (9), consider the case of positive-negative mixing pairs, $M(a) \subset U^+(a) \times U^-(a)$. Then, for $((x, y), (x', y')) \in M(a)$, the mixed label is $\operatorname{mix}_\lambda(y, y') = \operatorname{mix}_\lambda(1, 0) = \lambda$ and (9) becomes

$$\widetilde{\ell}(a;\theta) = \tau\left(\sigma^+\left(\sum_{(v,\lambda)\in V(a)} \lambda\rho^+(s(a,v))\right) + \sigma^-\left(\sum_{(v,\lambda)\in V(a)} (1-\lambda)\rho^-(s(a,v))\right)\right). \tag{12}$$

Thus, the mixed embedding v is both positive (with weight λ) and negative (with weight $1 - \lambda$). Whereas for positive-positive mixing, that is, for $M(a) \subset U^+(a)^2$, the mixed label is 1 and the negative term vanishes. Similarly, for negative-negative mixing, that is, for $M(a) \subset U^-(a)^2$, the mixed label is 0 and the positive term vanishes.

In the particular case of contrastive (1) loss, positive-negative mixing (12) becomes

$$\widetilde{\ell}_{\text{cont}}(a;\theta) := \sum_{(v,\lambda)\in V(a)} -\lambda s(a,v) + \sum_{(v,\lambda)\in V(a)} (1-\lambda)[s(a,v)-m]_+. \tag{13}$$

Similarly, for multi-similarity (2),

$$\begin{aligned}\widetilde{\ell}_{\text{MS}}(a;\theta) :=& \frac{1}{\beta}\log\left(1 + \sum_{(v,\lambda)\in V(a)} \lambda e^{-\beta(s(a,v)-m)}\right) + \\ & \frac{1}{\gamma}\log\left(1 + \sum_{(v,\lambda)\in V(a)} (1-\lambda) e^{\gamma(s(a,v)-m)}\right).\end{aligned} \tag{14}$$

A.2 Analysis: Mixed embeddings and positivity

Positivity Under positive-negative mixing, (12) shows that a mixed embedding v with interpolation factor λ behaves as both positive and negative to different extents, depending on λ: mostly positive for λ close to 1, mostly negative for λ close to 0. The net effect depends on the derivative of the loss with respect to the similarity $\partial\widetilde{\ell}(a;\theta)/\partial s(a,v)$: if the derivative is negative, then v behaves as positive and vice versa. This is clear from the chain rule

$$\frac{\partial\widetilde{\ell}(a;\theta)}{\partial v} = \frac{\partial\widetilde{\ell}(a;\theta)}{\partial s(a,v)} \cdot \frac{\partial s(a,v)}{\partial v}, \tag{15}$$

because $\partial s(a,v)/\partial v$ is a vector pointing in a direction that makes a, v more similar and the loss is being minimized. Let $\operatorname{Pos}(a,v)$ be the event that v behaves as "positive", *i.e.*, $\partial\widetilde{\ell}(a;\theta)/\partial s(a,v) \leq 0$ and minimizing the loss will increase the similarity $s(a,v)$.

Multi-similarity We estimate the probability of $\operatorname{Pos}(a,v)$ as a function of λ in the case of multi-similarity with a single embedding v obtained by mixing a positive with a negative:

$$\widetilde{\ell}_{\text{MS}}(a;\theta) = \frac{1}{\beta}\log\left(1+\lambda e^{-\beta(s(a,v)-m)}\right) + \frac{1}{\gamma}\log\left(1+(1-\lambda)e^{\gamma(s(a,v)-m)}\right). \tag{16}$$

In this case, $\operatorname{Pos}(a,v)$ occurs if and only if

$$\frac{\partial\widetilde{\ell}_{\text{MS}}(a;\theta)}{\partial s(a,v)} = \frac{-\lambda e^{-\beta(s(a,v)-m)}}{(1+\lambda e^{-\beta(s(a,v)-m)})} + \frac{(1-\lambda)e^{\gamma(s(a,v)-m)}}{(1+(1-\lambda)e^{\gamma(s(a,v)-m)})} \leq 0. \tag{17}$$

DATASET	CUB200 [45]	CARS196 [23]	SOP [30]	IN-SHOP [26]
Objects	birds	cars	household furniture	clothes
# classes	200	196	$22,634$	$7,982$
# training images	$5,894$	$8,092$	$60,026$	$26,356$
# testing images	$5,894$	$8,093$	$60,027$	$26,356$
sampling	random	random	balanced	balanced
samples per class	–	–	5	5
classes per batch	$65^\dagger$	$70^\dagger$	20	20
learning rate	1×10^{-4}	1×10^{-4}	3×10^{-5}	1×10^{-4}

Table 5: *Statistics and settings* for the four datasets we use in our experiments. †: average.

By letting $t := s(a, v) - m$, this condition is equivalent to

$$\frac{(1-\lambda)e^{\gamma t}}{(1+(1-\lambda)e^{\gamma t})} \leq \frac{\lambda e^{-\beta t}}{(1+\lambda e^{-\beta t})} \tag{18}$$

$$(1-\lambda)e^{\gamma t}(1+\lambda e^{-\beta t}) \leq \lambda e^{-\beta t}(1+(1-\lambda)e^{\gamma t}) \tag{19}$$

$$(1-\lambda)e^{\gamma t} + \lambda(1-\lambda)e^{(\gamma-\beta)t} \leq \lambda e^{-\beta t} + \lambda(1-\lambda)e^{(\gamma-\beta)t} \tag{20}$$

$$e^{(\beta+\gamma)t} \leq \frac{\lambda}{1-\lambda} \tag{21}$$

$$(\beta+\gamma)(s(a,v)-m) \leq \ln\left(\frac{\lambda}{1-\lambda}\right) \tag{22}$$

$$s(a,v) \leq \frac{1}{\beta+\gamma}\ln\left(\frac{\lambda}{1-\lambda}\right) + m. \tag{23}$$

Finally, the probability of $\text{Pos}(a, v)$ as a function of λ is

$$\text{P}(\text{Pos}(a,v)) = F_\lambda\left(\frac{1}{\beta+\gamma}\ln\left(\frac{\lambda}{1-\lambda}\right) + m\right), \tag{24}$$

where F_λ is the CDF of similarities $s(a, v)$ between anchors a and mixed embeddings v with interpolation factor λ.

In Figure 2, we measure the probability of $\text{Pos}(a, v)$ as a function of λ in two ways. First, we measure the derivative $\partial\widetilde{\ell}_{\text{MS}}(a;\theta)/\partial s(a, v)$ for anchors a and mixed embeddings v over the entire dataset and we report the empirical probability of this derivative being non-positive versus λ. Second, we measure $\text{P}(\text{Pos}(a, v))$ theoretically using (24), where the CDF of similarities $s(a, v)$ is again measured empirically for a and v over the dataset, as a function of λ. Despite the simplifying assumption of a single positive and a single negative in deriving (24), we observe that the two measurements agree in general. They are both increasing functions of λ of sigmoidal shape, they roughly yield $\text{P}(\text{Pos}(a, v)) \geq 0.5$ for $\lambda \geq 0.5$ and they confirm that a mixed embedding is mostly positive for λ close to 1 and mostly negative for λ close to 0.

B More on experiments

B.1 Setup

Datasets and sampling Dataset statistics are summarized in Table 5. Since the number of classes is large compared to the batch size in SOP and In-Shop, batches would rarely contain a positive pair when sampled uniformly at random. Hence, we use *balanced sampling* [54], *i.e.*, a fixed number of classes and examples per class, as shown in Table 5. For fair comparison with baseline methods, images are randomly flipped and cropped to 224×224 at training. At inference, we resize to 256×256 and then center-crop to 224×224.

Training We train R-50 using AdamW [27] optimizer for 100 epochs with a batch size 100. The initial learning rate per dataset is shown in Table 5. The learning rate is decayed by 0.1 for Cont and

	CUB200			CARS196			SOP			IN-SHOP		
Method	1	2	4	1	2	4	1	10	100	1	10	20
Triplet [48]	63.5	75.6	84.4	77.3	85.4	90.8	70.5	85.6	94.3	85.3	96.6	97.8
LiftedStructure [30]	65.9	75.8	84.5	81.4	88.3	92.4	76.1	88.6	95.2	88.6	97.6	98.4
ProxyNCA [28]	65.2	75.6	83.8	81.2	87.9	92.6	73.2	87.0	94.4	86.2	95.9	97.0
Margin [49]	65.0	76.2	84.6	82.1	88.7	92.7	74.8	87.8	94.8	88.6	97.0	97.8
SoftTriple [31]	67.3	77.7	86.2	86.5	91.9	95.3	79.8	91.2	96.3	**91.0**	97.6	98.3
D&C [36]*	65.9	76.6	84.4	84.6	90.7	94.1	75.9	88.4	94.9	85.7	95.5	96.9
EPSHN [51]*	64.9	75.3	83.5	82.7	89.3	93.0	78.3	90.7	96.3	87.8	95.7	96.8
ProxyNCA++ [40]*	69.0	79.8	**87.3**	86.5	92.5	95.7	**80.7**	**92.0**	**96.7**	90.4	**98.1**	**98.8**
Cont [12]	64.7	75.9	84.6	81.6	88.2	92.7	74.9	87.0	93.9	86.4	94.7	96.2
+Metrix/input	66.3	77.1	85.2	82.9	89.3	93.7	75.8	87.8	94.6	87.7	95.9	96.5
	+1.6	+1.2	+0.6	+1.3	+1.1	+1.0	+0.9	+0.8	+0.7	+1.3	+1.2	+0.3
+Metrix	67.4	77.9	85.7	85.1	91.1	94.6	77.5	89.1	95.5	89.1	95.7	97.1
	+2.7	+2.0	+1.1	+3.5	+2.9	+1.9	+2.6	+2.1	+1.5	+2.7	+1.0	+0.9
+Metrix/embed	66.4	77.6	85.4	83.9	90.3	94.1	76.7	88.6	95.2	88.4	95.4	96.8
	+1.7	+1.7	+0.8	+2.3	+2.1	+1.4	+1.8	+1.6	+1.3	+2.0	+0.7	+0.6
MS [47]	67.8	77.8	85.6	**87.8**	92.7	95.3	76.9	89.8	95.9	90.1	97.6	98.4
+Metrix/input	69.0	79.1	86.0	89.0	93.4	96.0	77.9	90.6	95.9	91.8	98.0	98.9
	+1.2	+1.3	+0.4	+1.2	+0.7	+0.7	+1.0	+0.8	+0.0	+1.7	+0.4	+0.5
+Metrix	**71.4**	80.6	86.8	**89.6**	**94.2**	96.0	81.0	**92.0**	**97.2**	**92.2**	**98.5**	98.6
	+3.6	+2.8	+1.2	+1.8	+1.5	+0.7	+4.1	+2.2	+1.3	+2.1	+0.9	+0.2
+Metrix/embed	70.2	80.4	86.7	88.8	92.9	95.6	78.5	91.3	96.7	91.9	98.3	98.7
	+2.4	+2.6	+1.1	+1.0	+0.2	+0.3	+1.6	+1.5	+0.8	+1.8	+0.7	+0.3
PA [20]*	**69.7**	**80.0**	87.0	87.7	**92.9**	**95.8**	–	–	–	–	–	–
PA [20]	69.5	79.3	87.0	87.6	92.3	95.5	79.1	90.8	96.2	90.0	97.4	98.2
+Metrix/input	70.5	81.2	87.8	88.2	93.2	96.2	79.8	91.4	96.5	90.9	98.1	98.4
	+0.8	+1.2	+0.8	+0.5	+0.3	+0.4	+0.7	+0.6	+0.3	+0.9	+0.7	+0.2
+Metrix	71.0	**81.8**	**88.2**	89.1	93.6	**96.7**	**81.3**	91.7	96.9	91.9	98.2	**98.8**
	+1.3	+1.8	+1.2	+1.4	+0.7	+0.9	+2.2	+0.9	+0.7	+1.9	+0.8	+0.6
+Metrix/embed	70.4	81.1	87.9	88.9	93.3	96.4	80.6	91.7	96.6	91.6	98.3	98.3
	+0.7	+1.1	+0.9	+1.2	+0.4	+0.6	+1.5	+0.9	+0.4	+1.6	+0.9	+0.1
Gain over SOTA	+1.7	+1.8	+0.9	+1.8	+1.3	+0.9	+0.6	+0.0	+0.5	+1.2	+0.4	+0.0

Table 6: *Improving the SOTA with our Metrix* (Metrix/feature) using Resnet-50 with embedding size $d = 512$. R@K (%): Recall@K; higher is better. *: reported by authors. **Bold black**: best baseline (previous SOTA, one per column). **Red**: Our new SOTA. Gain over SOTA is over best baseline. MS: Multi-Similarity, PA: Proxy Anchor

by 0.5 for MS and PA on CUB200 and Cars196. For SOP and In-Shop, we decay the learning rate by 0.25 for all losses. The weight decay is set to 0.0001.

Evaluation protocol We follow the standard evaluation protocol of [30], where half classes are used for training and the other half for testing. For each test example taken as a query, we find its K-nearest neighbors in the test set excluding itself in the embedding space and we assign it a score of 1 if an example of the same class is contained in the neighbors and 0 otherwise. We measure Recall@K, which is the average of this score over the test set.

B.2 More results

Table 6 is an extension of Table 2 that includes all three mixup types (input, feature, embedding). It shows that not just feature mixup but *all* mixup types consistently improve the performance of all baseline losses (Cont, MS, PA) across all datasets. It also shows that across all baseline losses and all datasets, feature mixup works best, followed by embedding and input mixup. This result confirms the findings of Table 4 on Cars196.

B.3 More ablations

Mixup strength w We study the effect of the mixup strength w in the combination of the clean and mixed loss (10) for different mixup types. As shown in Figure 3, mixup consistently improves the baseline and the effect of w is small, especially for input and embedding mixup. Feature mixup works best and is slightly more sensitive.

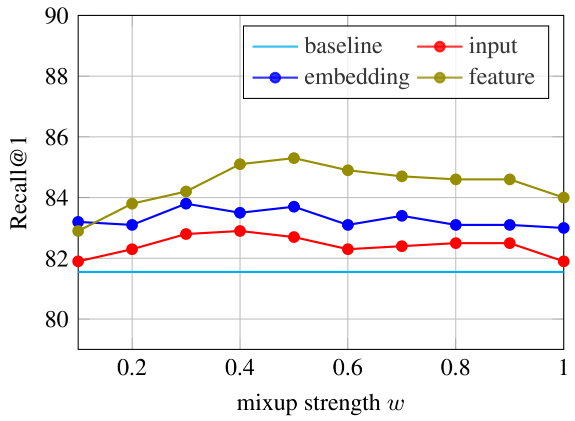

Figure 3: *Effect of mixup strength* for different mixup types using contrastive loss and R-50 with embedding size $d = 512$ on Cars196. Recall@K (%): higher is better.

Ablation on CUB200 We perform additional ablations on CUB200 using R-50 with $d = 128$ by applying contrastive loss. All results are shown in Table 7. One may draw the same conclusions as from Table 4 on Cars196 with $d = 512$, which confirms that our choice of hard negatives and mixup pairs is generalizable across different datasets and embedding sizes.

In particular, following the settings of subsection 4.4, we observe in Table 7 that using $k = 3$ hard negatives for input mixup and all pairs for feature/embedding mixup achieves the best performance in terms of Recall@1. Similarly, using a single set of mixing pairs, positive-negative and anchor-negative consistently outperform the baseline, whereas positive-positive is inferior than the baseline. Furthermore, combining positive-negative and anchor-negative pairs by choosing uniformly at random in each iteration achieves the best overall performance.

We also study the effect of using more than one mixup type (input, feature,embedding), chosen uniformly at random per iteration. The set of mixing pairs is also chosen from (positive-negative, anchor-negative) uniformly at random per iteration in this study. From Table 7, we observe that although mixing input, features and embedding works best with an improvement of 0.8% over feature mixup alone ($64.5 \rightarrow 65.3$), it is computationally expensive due to using input mixup. The next best choice is mixing features and embeddings, which is worse than using feature mixup alone (64.2 *vs.* 64.5). This confirms our choice of using feature mixup as default.

STUDY	HARD NEGATIVES k	MIXING PAIRS	MIXUP TYPE	R@1	R@2	R@4	R@8
baseline				61.6	73.7	83.6	90.1
hard negatives	1	pos-neg / anc-neg	input	62.4	73.9	83.0	89.7
	2	pos-neg / anc-neg	input	62.7	74.2	**83.6**	90.0
	3	pos-neg / anc-neg	input	63.1	74.5	83.5	90.3
	20	pos-neg / anc-neg	feature	63.9	75.0	83.9	89.9
	40	pos-neg / anc-neg	feature	63.5	75.2	83.5	89.8
	all	pos-neg / anc-neg	feature	**64.5**	**75.4**	84.3	**90.6**
	20	pos-neg / anc-neg	embed	63.1	74.3	83.1	90.0
	40	pos-neg / anc-neg	embed	63.5	74.7	83.6	90.1
	all	pos-neg / anc-neg	embed	64.0	75.1	84.8	90.9
mixing pairs	–	pos-pos	input	58.7	70.7	80.1	87.1
	3	pos-neg	input	62.9	75.1	83.4	90.6
	3	anc-neg	input	62.8	74.7	83.6	90.1
	–	pos-pos	feature	61.0	73.1	82.5	89.7
	all	pos-neg	feature	63.9	75.0	83.9	89.9
	all	anc-neg	feature	63.8	74.8	83.6	90.2
	–	pos-pos	embed	59.7	72.2	82.7	89.5
	all	pos-neg	embed	63.8	75.1	83.3	90.5
	all	anc-neg	embed	63.5	75.0	83.9	90.5
mixup type combinations	{1, all}	pos-neg / anc-neg	{input, feature}	63.9	75.1	**84.9**	90.5
	{3, all}	pos-neg / anc-neg	{input, embed}	63.4	74.9	84.5	90.1
	{all, all}	pos-neg / anc-neg	{feature, embed}	64.2	75.2	84.1	90.7
	{1, all, all}	pos-neg / anc-neg	{input, feature, embed}	**65.3**	**76.2**	84.4	**91.2**

Table 7: *Ablation study of our Metrix* using contrastive loss and R-50 with embedding size $d = 128$ on CUB200. R@K (%): Recall@K; higher is better.