Deep Metric Learning via Weighted Contrastive Loss

Kihyuk Hong
Independent
hominot@alumni.stanford.edu

Keunchan Park
NAVER Corp.
keunchan.park@navercorp.com

Junmo Kim
KAIST
junmo@ee.kaist.ac.kr

Abstract

Distance metric learning aims to learn an embedding representation of examples consistent with their similarity semantic. Traditionally, the contrastive loss which sums over the loss on pairs of examples was used for this task before the deep network was introduced. Recently, the poor performance of the loss on deep networks and larger datasets directed the research community to develop loss functions and data sampling methods that scale well. In this paper, propose a loss that modifies the contrastive loss with a novel reweighting scheme motivated by a probabilistic analysis of the contrastive loss. The proposed loss coupled with importance sampling outperforms state-of-the-art methods.

1. Introduction

Distance metric learning is a widely used technique for a variety of tasks in computer vision. It learns an embedding representation of examples that preserves the semantic of similarity among examples. Specifically, examples of the same class are clustered in the embedding space while examples of different classes are separated. The technique has been successfully employed for tasks such as face recognition [2, 11, 16, 17], image retrieval [19], and clustering [22, 15].

Distance metric learning uses loss functions that encode the similarity semantic among embeddings. Many loss functions have been studied in the literature. For example, contrastive loss [2, 3] penalizes distances between pairs of similar examples and rewards distances between pairs of dissimilar examples. Another commonly used loss is triplet loss [12, 20, 11, 1]. Triplet loss uses triplets of examples to construct the loss function: an anchor example, a similar and a dissimilar examples to the anchor example. Triplet loss encourages the anchor example to be closer to the similar example than to the dissimilar one. N-pair loss [14] takes this idea further and uses one similar example and multiple dissimilar examples to construct the loss function. It uses a softmax function to encourage the anchor example to be closer to the similar example than to dissimilar examples.

Distance metric learning utilizes various sampling strategies for minimizing the loss function. Since it is prohibitive to sample all pairs, triplets, or tuplets for calculating loss functions, many heuristics are used to sample examples that have better chances of providing useful information. The most basic sampling strategy is to balance the number of similar pairs and dissimilar pairs in a mini-batch. More complicated strategies have been proposed including hard negative sampling [13], semi-hard negative sampling [11], distance weighted sampling [21], and hard class sampling [14]. Empirically, sampling strategies affect the performance of the model as much as the loss functions [21].

In this paper, we provide an importance sampling method for correcting the bias introduced by various batch designs during optimization steps. Strikingly, the choice of batch design no longer affects the performance when importance sampling is used. By analyzing the traditional contrastive loss with a probabilistic argument, we show that the loss can fail when there is a large number of classes in the training data. To find a remedy to the failure case while preserving the structure of the contrastive loss, we first propose the weighted contrastive loss which generalizes the contrastive loss. With a probabilistic approach, we construct an instance of the weighted contrastive loss that solves the aforementioned problem. We show experimentally that the proposed loss with importance sampling outperforms state-of-the-art methods.

2. Related work

2.1. Contrastive loss with random sampling

The contrastive loss [2] minimizes the pairwise distances between examples with the same class and separates examples with different classes by a margin. Hadsell *et al*. [3] use all example pairs to calculate the loss. For experiments with larger datasets, a mini-batch stochastic gradient descent is used with various sampling strategies.

2.2. Triplet loss with semi-hard negative mining

The triplet loss [11] improves upon the contrastive loss by constructing the loss function based on triplets instead of pairs of examples. Each triplet has an anchor, a positive, and a negative example. The positive example has the same class as the anchor and the negative example has a different class. The loss penalizes triplets where the anchor example is closer to the negative example than the positive example. The loss has the form

$$\frac{1}{|\mathcal{T}|}\sum_{(i,j,k)\in\mathcal{T}}\max(0, D_{ij}^2+\alpha-D_{ik}^2) \tag{1}$$

where $\mathcal{T}$ is the set of all triplets (i,j,k) with i an anchor, j a corresponding positive example, and k a corresponding negative example. Since it is prohibitive to calculate the loss using all possible triplets, a sampling strategy is used. FaceNet [11] suggest using a semi-hard negative mining strategy. They first form a mini-batch of examples with a fixed number of examples per class so that enough positive example pairs can be sampled from the mini-batch. Then, for each anchor-positive pair in the mini-batch, they select a semi-hard negative, which is defined to be the closest negative example that is further away from the anchor than the positive example.

2.3. N-pair loss with negative class mining

Sohn *et al*. [14] propose the n-pair loss which uses cross-entropy loss among pairs of examples in each mini-batch. The loss has the form

$$-\frac{1}{|\mathcal{T}_{n+1}|}\sum_{t\in\mathcal{T}_{n+1}}\log\frac{\exp S_{ij}}{\exp S_{ij}+\sum_{l=1}^{n-1}\exp S_{ik_l}} \tag{2}$$

where S_{ij} is the dot product of embeddings of examples i and j and $t=(i,j,k_1,\ldots,k_{n-1})\in\mathcal{T}_{n+1}$ is a tuplet. They use an l_2 regularizer on the embeddings for a more stable optimization. Since it is prohibitive to use all tuplets in $\mathcal{T}_{n+1}$, they propose a unique n-pair batch design with negative class mining, which selects n classes that violate the triplet constraint the most when constructing a mini-batch.

2.4. Margin loss with distance weighted sampling

Wu *et al*. [21] propose margin loss with distance weighted sampling. Margin loss is similar to contrastive loss but uses euclidean distance instead of euclidean distance squared to form the loss function. The loss function has the form

$$\frac{1}{|\mathcal{P}|}\sum_{(i,j)\in\mathcal{P}}Y_{ij}(D_{ij}-\beta_i+\alpha)_+ + (1-Y_{ij})(\alpha+\beta_i-D_{ij})_+ \tag{3}$$

where α and β_i are margin parameters. For sampling, they propose distance weighted sampling. They first form a set of a fixed number of examples per class. From the set of examples, all positive example pairs are included in the mini-batch. For each positive example pair, they sample a negative pair based on the distance weighted sampling, which makes the distances between the negative pairs uniformly distributed.

2.5. Other related work

There are other related work on deep metric learning that do not focus on sampling strategies. Song *et al*. [6] uses the most information out of mini-batch by using all positive pairs and all negative pairs to form cross-entropy loss. Song *et al*. [15] uses a clustering quality metric in the loss to optimize for the quality of clustering directly. Movshovitz-Attias *et al*. [9] uses auxiliary proxy embeddings so that they no longer need to compute the distances between examples and use distances of anchor embeddings and their proxy embeddings to form the loss function.

3. Preliminaries

In this section, we review the problem setup of deep metric learning and key concepts required to follow this paper.

3.1. Problem setup

Let $\{(x_i,y_i)\}_{i=1}^N$ be a set of data where $x_i\in\mathcal{X}$ is an input example and $y_i\in\{1,\cdots,L\}$ its output class. Let $f(\cdot;\theta):\mathcal{X}\to\mathbb{R}^d$ be a deep network parametrized by θ that maps an input example to a d-dimensional embedding vector. As a shorthand, denote $f_i=f(x_i;\theta)$. Our goal is to learn θ that keeps embeddings of examples of the same class close to each other and separates embeddings of examples of different classes far apart. We refer to a pair of examples of the same class as a positive pair and pair of examples of different classes a negative pair.

To achieve this goal, Chopra *et al*. [2] proposes pairwise loss functions of the form

$$\sum_{(i,j)\in\mathcal{P}}Y_{ij}g(D_{ij})+(1-Y_{ij})h(D_{ij}) \tag{4}$$

where $D_{ij}=\|f_i-f_j\|_2$ is the distance between embeddings, $Y_{ij}=1\{y_i=y_j\}$ indicates whether examples i and j are in the same class, $g:\mathbb{R}^+\to\mathbb{R}$ is an increasing function, $h:\mathbb{R}^+\to\mathbb{R}$ is a decreasing function, and $\mathcal{P}$ is the set of all pairs of data indices $\{1,\cdots,N\}$. Intuitively, the loss penalizes distances between positive pairs and rewards distances between negative pairs. Many variants of the functions g,h have been proposed, the most popular baseline being the pairwise loss with

$$g(d)=d^2,\ \ h(d)=\max(0,\alpha-d)^2 \tag{5}$$

where $\alpha>0$ is a margin hyperparameter. This loss is referred to as the *contrastive loss* [3].

3.2. Importance sampling

Importance sampling [4] is a technique for estimating the expectation of a random variable X that follows a distribution P using samples from a different distribution Q. Suppose X is a discrete random variable with support $\mathcal{X}$. Then,

$$\mathbb{E}_P[X] = \sum_{x\in\mathcal{X}} xP(x) = \sum_{x\in\mathcal{X}} x\frac{P(x)}{Q(x)}Q(x) = \mathbb{E}_Q\left[X\frac{P(X)}{Q(X)}\right] \tag{6}$$

Using this identity, we can estimate $\mathbb{E}_P[X]$ using samples $\{X_i\}_{i=1}^n$ drawn from the distribution Q by

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n}\frac{P(X_i)}{Q(X_i)}X_i = \frac{1}{n}\sum_{i=1}^{n}W(X_i)X_i \tag{7}$$

where the quantity $W(x) = P(x)/Q(x)$ is referred to as the importance weight of the sample x. Importance sampling is useful when sampling from P is undesired.

4. Analysis on batch designs

In this section, we analyze batch designs widely used in the literature. By a batch design, we mean the sampling strategy for constructing mini-bacthes of positive and negative pairs. We will compute the probability distribution on the set of all pairs $\mathcal{P}$ induced by each batch design. The probability distribution will be used in section 5 to compute the importance weight of each pair.

We use N_c to denote the number of images of class c in the data $\{(x_i, y_i)\}_{i=1}^N$, *i.e.*, $N_c = \sum_{i=1}^N 1\{y_i = c\}$.

4.1. Balanced random batch design

Balanced random batch design samples positive and negative ordered pairs randomly from the set of all pairs $\mathcal{P}$ while keeping the ratio of the number of positive pairs and negative pairs to p. This is done in the following steps. First, decide whether to sample a positive pair or a negative pair with probability p. Then, if decided to sample a positive pair, choose a class uniformly at random from $\{1, \cdots, L\}$ and draw a pair of examples of the chosen class uniformly at random. If decided to sample a negative pair, choose two distinct classes uniformly at random and for each chosen class, choose an example of the class uniformly at random to get a negative pair. The pair sampling is repeated to form a mini-batch of a desired number of pairs. We will refer to this strategy as p-random strategy.

Let $Q_p^{\text{random}}(i, j)$ be the probability that the ordered pair (i, j) is chosen when choosing a pair from the mini-batch sampled by p-random batch design. The probability of choosing a positive pair (i, j) is equal to the probability of deciding to choose a positive pair p times the probability of choosing the right class $\frac{1}{L}$ times the probability of choosing the ordered pair (i, j) among N_{y_i} examples, which is $\frac{1}{N_{y_i}(N_{y_i}-1)}$. The probability of choosing a negative pair (i, j) is equal to $1-p$, the probability of deciding to choose a negative pair, times $\frac{1}{L(L-1)}$, the probability of choosing the classes y_i and y_j, times $\frac{1}{N_{y_i}N_{y_j}}$, the probability of choosing the examples i and j. In summary,

$$Q_p^{\text{random}}(i,j) = \begin{cases} \dfrac{p}{LN_{y_i}(N_{y_i}-1)} & \text{if } y_i = y_j \\ \dfrac{1-p}{L(L-1)N_{y_i}N_{y_j}} & \text{otherwise} \end{cases} \tag{8}$$

4.2. Group batch design

Group batch design samples all ordered pairs from a set of examples consisting of randomly chosen classes with a fixed number of examples per class. The (m, n)-group batch design refers to the strategy of uniformly sampling n classes from $\{1, \cdots, L\}$ without replacement and uniformly sampling m examples per class without replacement.

Let $Q_{m,n}^{\text{group}}(i, j)$ be the probability that the ordered pair (i, j) is chosen when choosing a pair from a mini-batch constructed by m, n-group batch design. The probability of choosing a positive pair (i, j) is equal to the probability of i and j being chosen when sampling mn examples times the probability of the pair (i, j) being chosen when choosing from $mn(mn-1)$ possible pairs. The former probability is equal to $\binom{L-1}{n-1}/\binom{L}{n}$, the probability of the class of the positive pairs being chosen when choosing n classes, times $\binom{N_{y_i}-2}{m-2}/\binom{N_{y_i}}{m}$, the probability of i and j being chosen when choosing m examples among N_{y_i} examples. The latter probability is $\frac{1}{mn(mn-1)}$.

Similarly, the probability of choosing a negative pair (i, j) is equal to $\binom{L-2}{n-2}/\binom{L}{n}$, the probability of y_i and y_j being chosen when choosing n classes among L classes, times $\frac{1}{N_{y_i}N_{y_j}}$, the probability of choosing i and j among N_{y_i} and N_{y_j} examples respectively, times $\frac{1}{mn(mn-1)}$, the probability of the pair (i, j) being chosen when choosing from $mn(mn-1)$ ordered pairs. In summary,

$$Q_{m,n}^{\text{group}}(i,j) = \begin{cases} \dfrac{m-1}{L(mn-1)N_{y_i}(N_{y_i}-1)} & \text{if } y_i = y_j \\ \dfrac{m(n-1)}{L(L-1)(mn-1)N_{y_i}N_{y_j}} & \text{otherwise} \end{cases} \tag{9}$$

5. Proposed method

In this section, we first generalize the contrastive loss to *weighted contrastive loss* and propose an improvement to the contrastive loss. Then we provide importance sampling methods for both traditional contrastive loss and the proposed loss.

We define the weighted contrastive loss as

$$\sum_{(i,j)\in\mathcal{P}} [Y_{ij} g(D_{ij}) + (1 - Y_{ij})\eta_{ij} h(D_{ij})] \tag{10}$$

where $\eta_{ij} > 0$ is the weight for the negative example pair (i, j). Note that the contrastive loss defined in (4) is a special case of the weighted contrastive loss with $\eta_{ij} = 1$. In the first subsection, we analyze the contrastive loss to give an insight into how negative examples are weighted for each positive pair. Based on the insight, we propose an improvement by choosing η_{ij} in the weighted contrastive loss framework to have a desired property.

5.1. Balanced contrastive loss

We first provide an analysis on the contrastive loss function (4) and then propose an improvement. The contrastive loss function can be rewritten as

$$\sum_{(i,j)\in\mathcal{P}^+} \left[g(D_{ij}) + \frac{1}{N_{y_i} - 1} \sum_{l\in\mathcal{Y}\setminus\{y_i\}} \sum_{k\in\mathcal{X}_l} h(D_{ik}) \right] \tag{11}$$

where $\mathcal{P}^+$ is the set of all positive pairs and $\mathcal{X}_l$ is the set of examples of class l. Note that the rewritten form can be interpreted as summing over losses of all tuplets of the form $(i, j, \mathcal{K})$ where i is an anchor example, j a positive example, and $\mathcal{K} = \mathcal{X} \setminus \mathcal{X}_{y_i}$ a set of negative examples. The tuplet loss in this case is $g(D_{ij}) + \frac{1}{N_{y_i}-1} \sum_{k\in\mathcal{K}} h(D_{ik})$. Introducing a random example of class l denoted K_l, which is a random variable uniformly distributed on $\mathcal{X}_l$, the expression can be rewritten as

$$\sum_{(i,j)\in\mathcal{P}^+} \mathbb{E}\left[g(D_{ij}) + \sum_{l\in\mathcal{Y}\setminus\{y_i\}} \frac{N_l}{N_{y_i} - 1} h(D_{iK_l}) \right] \tag{12}$$

where we used $\mathbb{E}h(D_{iK_l}) = \frac{1}{N_l} \sum_{k\in\mathcal{X}_l} h(D_{ik})$. This expression is the sum of expected tuplet losses for tuplets sampled as follows. For each pair of an anchor example i and a positive example j, sample one negative example K_l from each of $L - 1$ negative classes and weight the loss $h(D_{iK_l})$ by $\frac{N_l}{N_{y_i}-1}$. The tuplet used in this case is $(i, j, K_1, \ldots, K_{y_i-1}, K_{y_i+1}, \ldots, K_L)$. One potential problem with this particular tuplet loss is that when the number of classes L is large, each positive pair (i, j) is compared with too many negative examples. In this case, the overall optimization process can fail to learn the closeness semantic of positive pairs and focus only on separating negative pairs apart. We show experimentally that comparing with too many negative examples can hurt performance in section 6.5.

The problem mentioned above may be solved by defining tuplet loss on $(i, j, K_1, \ldots, K_{y_i-1}, K_{y_i+1}, \ldots, K_L)$ carefully. As an example, we expect the n-pair loss proposed by Sohn *et al*. [14] to mitigate this problem. But in this paper, we stick to the contrastive loss and provide a simple solution in the weighted contrastive loss framework. In particular, we introduce a hyperparameter λ that controls the number of negative examples each positive pair is compared with.

Consider a tuplet sampling method as follows. For each pair of an anchor example i and a positive example j, sample a negative class λ times uniformly from the set of negative classes $\mathcal{Y} \setminus \{y_i\}$. For each negative class C, sample one negative example of the chosen negative class. The resulting tuplet is of the form $(i, j, K_{c_1}, \ldots, K_{c_\lambda})$ where $c_1, \ldots, c_\lambda$ are negative classes sampled uniformly from $\mathcal{Y} \setminus \{y_i\}$ and K_{c_l} is the l-th negative example chosen from the set of examples of class c_l. The expected loss is

$$\sum_{(i,j)\in\mathcal{P}^+} \mathbb{E}\left[g(D_{ij}) + \lambda h(D_{iK_C}) \right] \tag{13}$$

$$= \sum_{(i,j)\in\mathcal{P}^+} \mathbb{E}\left[g(D_{ij}) + \sum_{l\in\mathcal{Y}\setminus\{y_i\}} \frac{\lambda}{L-1} h(D_{iK_l}) \right] \tag{14}$$

where $\mathbb{E}\left[h(D_{iK_C}) \mid \boldsymbol{K}\right] = \frac{1}{L-1} \sum_{l\in\mathcal{Y}\setminus\{y_i\}} h(D_{iK_l})$ is used. Rearranging, we get

$$\sum_{(i,j)\in\mathcal{P}} \left[Y_{ij} g(D_{ij}) + (1 - Y_{ij}) \frac{\lambda}{L-1} \frac{N_{y_i} - 1}{N_{y_j}} h(D_{ij}) \right] \tag{15}$$

which we call the *balanced contrastive loss*. The hyperparameter λ has the interpretation of the number of negative examples compared for each positive example pair. The weights for the balanced contrastive loss in the weighted contrastive loss framework is

$$\eta_{ij} = \frac{\lambda}{L-1} \frac{N_{y_i} - 1}{N_{y_j}} \tag{16}$$

In section 6, we show that the balanced contrastive loss coupled with importance sampling method introduced in the next subsection outperforms the state-of-the-art methods.

Recall that the contrastive loss has $\eta_{ij} = 1$. Solving for λ that gives $\eta_{ij} = 1$, we get

$$\lambda_{ij}^{\text{traditional}} = \frac{N_{y_j}(L-1)}{N_{y_i} - 1} \tag{17}$$

We attach the subscript ij to λ since the value depends on the examples i, j. We can interpret this value as the number of negative examples the contrastive loss compares for each positive pair (i, j). Note that λ_{ij} depends on the number of classes L in the training dataset, which is not a desired property since the model should handle datasets with different number of classes. For large L, λ_{ij} gets large value

leading to a problem as per the previous discussion on the number of negative examples compared affecting the performance. We show experimentally in section 6 that the traditional contrastive loss can be improved if $\lambda_{ij}^{\text{traditional}}$ is artificially controlled by using a subset of the dataset.

5.2. Importance sampling for weighted contrastive loss

The standard gradient descent method computes the gradient of the full loss at each optimization step. The gradient of the weighted contrastive loss (10) is

$$\nabla_\theta \mathcal{L} = \sum_{(i,j)\in\mathcal{P}} Y_{ij}\nabla_\theta g(D_{ij}) + (1-Y_{ij})\eta_{ij}\nabla_\theta h(D_{ij}) \quad (18)$$

Since it is computationally prohibitive to compute the full gradient at each optimization step, the stochastic gradient descent method is employed to approximate the true gradient of the loss using a mini-batch.

Suppose P_U is a uniform distribution on $\mathcal{P}$. Then the gradient can be rewritten using expectation as

$$\frac{1}{N(N-1)}\mathbb{E}_{P_U}\left[Y_{IJ}\nabla_\theta g(D_{IJ}) + (1-Y_{IJ})\eta_{ij}\nabla_\theta h(D_{IJ})\right] \quad (19)$$

where (I, J) is a random vector with distribution P_U. The stochastic gradient descent method approximates the gradient of the loss using a mini-batch of samples from P_U to approximate the expectation as

$$\frac{1}{n}\sum_{k=1}^{n} Y_{i_kj_k}\nabla_\theta g(D_{i_kj_k}) + (1-Y_{i_kj_k})\eta_{ij}\nabla_\theta h(D_{i_kj_k}) \quad (20)$$

where $\{(i_k, j_k)\}_{k=1}^n$ is a mini-batch of n sampled pairs from P_U. However, a bias is introduced if the pairs are not sampled uniformly, as is the case for batch designs introduced in section 4.

To correct the bias, we use importance sampling. Suppose that each pair in a mini-batch constructed by the sampling strategy has distribution Q. Then the expectation in equation (19) is equal to

$$\mathbb{E}_Q\left[W_{IJ}\left(Y_{IJ}\nabla_\theta g(D_{IJ}) + (1-Y_{IJ})\eta_{ij}\nabla_\theta h(D_{IJ})\right)\right] \quad (21)$$

where $W_{IJ} = \frac{P_U(I,J)}{Q(I,J)}$ is the importance weight. Using this identity, the expectation can be approximated by

$$\frac{1}{B}\sum_{k=1}^{B} W_{i_kj_k}(Y_{i_kj_k}\nabla_\theta g(D_{i_kj_k}) + (1-Y_{i_kj_k})\eta_{ij}\nabla_\theta h(D_{i_kj_k})) \quad (22)$$

where $\{(i_k, j_k)\}_{k=1}^B$ is a mini-batch of samples from Q. For implementation using deep learning packages, it is often more convenient to know the equation of the loss function instead of the gradient of it. The mini-batch loss function is just

$$\frac{1}{B}\sum_{k=1}^{B} W_{i_kj_k}(Y_{i_kj_k}g(D_{i_kj_k}) + (1-Y_{i_kj_k})\eta_{ij}h(D_{i_kj_k})) \quad (23)$$

The key is to calculate the importance weight W_{ij}. For the p-random and (m, n)-group mini-batch sampling strategies, we can directly use Q provided in section 4 and $P_U(i, j) = \frac{1}{N(N-1)}$ to compute the importance weight W_{ij}. For p-random batch design,

$$W_{ij} = \begin{cases} \dfrac{LN_{y_i}(N_{y_i}-1)}{pN(N-1)} & \text{if } y_i = y_j \\ \dfrac{L(L-1)N_{y_i}N_{y_j}}{(1-p)N(N-1)} & \text{otherwise} \end{cases} \quad (24)$$

For (m, n)-group batch design,

$$W_{ij} = \begin{cases} \dfrac{L(mn-1)N_{y_i}(N_{y_i}-1)}{(m-1)N(N-1)} & \text{if } y_i = y_j \\ \dfrac{L(L-1)(mn-1)N_{y_i}N_{y_j}}{m(n-1)N(N-1)} & \text{otherwise} \end{cases} \quad (25)$$

For p-random and (m, n)-group batch design, we can use the corresponding weights to correct the bias of the estimation for the full gradient. In theory, we can calculate weights for any batch designs, but in this paper we only use these simple batch designs to show that even simple batch designs without careful negative mining strategies can perform well if coupled with balanced contrastive loss and importance sampling.

In summary, the proposed method is to use the balanced contrastive loss with importance weights corresponding to the batch design. The mini-batch loss is of the form (23) where W_{ij} is the importance weight of the batch design and $\eta_{ij} = \frac{\lambda}{L-1}\frac{N_{y_i}-1}{N_{y_j}}$. In the next section, we show experimentally that the proposed method outperforms state-of-the-art methods.

6. Experiments

We closely follow the experimental design of Song *et al*. [6] with minor changes. We use the ResNet-50 [5] network architecture pre-trained with ILSVRC 2012-CLS [10] dataset. After pre-training, the last softmax layer is replaced by a linear dense layer with output embedding size of 128. Input images are first resized and zero-padded to 256×256. Then, we use random horizontal mirroring and random crops from 256×256 to 224×224 for data augmentation during training. During testing we use a single center crop. We use 64 images in a mini-batch unless otherwise stated.

We use the Stanford Online Products [6], CARS196 [8], and CUB200-2011 [18] datasets. The Stanford Online Product dataset contains 120,053 images of 22,634 categories. The first 11,318 categories are used for training and the remaining for testing. The CARS196 dataset contains 16,185 car images of 196 models. We use the first 98 models for training and the remaining for testing. The CUB200-2011 dataset contains 11,788 bird images of 200 species. The first 100 species are used for training the remainder for testing. We do not use the bounding box information in any of the experiments.

Evaluation follows Song *et al*. [6]. The quality of image retrieval is evaluated by Recall@k metric. The Recall@k is the success rate of finding an image of the same class as the query image in the set of k closest images. For clustering, we use K-means algorithm and evaluate the quality using NMI score which is equal to $I(\Omega, \mathbb{C})/\sqrt{H(\Omega)H(\mathbb{C})}$ where $I(\cdot,\cdot)$ and $H(\cdot)$ are mutual information and entropy respectively.

For batch designs, we use $(4, 16)$-group batch design unless otherwise stated. We use l_2 normalization of the output of the last dense layer of the network for triplet and cluster losses. We use Adam [7] for mini-batch stochastic gradient descent. For learning rate, we perform hyperparameter search on 0.0003, 0.0001, and 0.00003. At every 100 iteration, we decay the learning rate by 0.9. We train for 2,000 iterations for all experiments. We use the same learning rate for the base network and the final dense layer. The hyperparameter λ is set to 256 for all experiments with balanced contrastive loss unless otherwise stated.

6.1. Performance of importance sampling

To see the effect of importance sampling, we evaluate the balanced contrastive loss function on CUB200 dataset with different batch designs. For all experiments, the same model hyperparameters are used. The only difference is in the batch design and whether importance sampling is used or not. As for the batch designs, (m, n)-group and p-random are used with various m, n and p. For fair comparison, the number of pairs sampled per mini-batch for all batch designs are fixed to 2016. To achieve this, we use 4032 examples to sample 2016 random unordered pairs for p-random batch designs. For m, n-group batch designs, we choose m, n satisfying $mn = 64$ so that there are 2016 unordered pairs. For each batch design, the corresponding importance weight calculated in section 4 is used.

The results are shown in table 1. Without importance sampling, the performance of the model varies significantly with batch designs. The best performing batch design when importance sampling is not used is $(2, 32)$-group which achieves recall@1 of 51.55. However the metric can drop significantly to 31.65 and 18.32 for (m, n)-group and p-random batch designs respectively depending on m, n and p. Strikingly, the performance is no longer affected by the batch design when importance sampling is used. Even the metric of the worst performing batch design becomes comparable to the best performing batch design when importance sampling is used. With importance sampling, we can worry less about how to design the mini-batch and focus on other aspects of the model.

Batch design		IS	R@1	R@2	R@8	NMI
Group	2,32	N	51.55	63.89	83.54	61.37
		Y	52.48	65.09	84.62	60.73
	4,16	N	49.75	62.41	83.02	58.80
		Y	53.02	65.40	84.77	61.92
	8,8	N	46.00	57.58	80.35	56.71
		Y	53.31	64.91	84.66	61.97
	16,4	N	42.78	55.77	77.77	54.29
		Y	51.27	63.35	83.29	61.41
	32,2	N	31.65	43.70	70.26	44.92
		Y	46.12	58.31	80.55	57.70
Random	0.9	N	18.32	27.11	53.21	37.84
		Y	52.75	65.51	85.18	61.08
	0.5	N	34.77	47.23	71.05	47.36
		Y	52.48	65.28	84.76	60.94
	0.1	N	46.32	58.59	80.87	56.44
		Y	52.99	65.46	84.47	61.45

Table 1. Effect of importance sampling on balanced contrastive loss with various batch designs. The group batch design with varying m, n and the random batch design with varying p are used. The IS column indicates whether importance sampling is used.

6.2. Perturbation on importance sampling weights

We now show experimentally that the importance weights calculated in section 4 are precise in the sense that deviating from the weights only hurts the performance. To see this, we train the balanced contrastive loss model on CUB200 with $(4, 16)$-group and $(32, 2)$-group batch designs and importance sampling, varying the degree of perturbation on importance weights. The perturbation is done by exponentiating the correct importance weights W_{ij} by δ so that $\tilde{W}_{ij}^{\delta} = (W_{ij})^{\delta}$ is used instead of the correct importance weight. If $\delta = 1$, then $\tilde{W}_{ij}^{\delta} = W_{ij}$ and no perturbation is done to the weights. As δ decreases from 1 to 0, the effect of importance sampling fades. When δ reaches 0, all importance weights become 1 and the mini-batch loss matches the original mini-batch loss without importance sampling. If $\delta < 0$, then the effects of importance weights reverse: $\tilde{W}_{ij}^{\delta} > 1$ for $W_{ij} < 1$ and vice versa. The degree of the reversal increases as δ becomes more negative. As δ increases from 1, the importance weights are more emphasized: the sample pairs (i, j) with $W_{ij} > 1$ get even greater weights

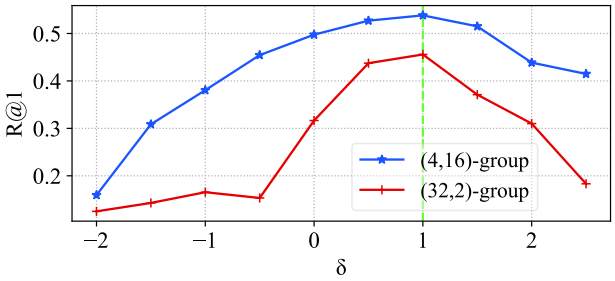

Figure 1. Importance weight perturbation experiment. No perturbation is done when $\delta = 1$. As δ deviates from 1, the degree of perturbation increases. The graph shows that the model performs best when the correct importance weights are used for both (4,16)-group and (32,2)-group batch designs.

($\tilde{W}_{ij}^{\delta} > W_{ij}$) and the pairs with weights less than 1 get even smaller weights.

We expect the performance to degrade as δ deviates from 1. This is consistent with the experimental result shown in figure 1. We plot the recall@1 metric for both batch designs with varying δ. We see bell-shaped curves for both batch designs with peaks at $\delta = 1$ which corresponds to using the correct importance weights. This suggests that the importance weights are precise.

6.3. Efficiency in train loss minimization

The stochastic gradient descent method used for minimizing the train loss (10) is expected to perform better if an unbiased estimate of the gradient (18) is used at every gradient descent step. Hence, we expect the train loss to be minimized more efficiently when importance sampling is used since it corrects the bias of the mini-batch gradient. We show experimentally that the train loss (10) converges to a lower value when importance sampling is used. The contrastive loss model is trained on CUB200 dataset with $(4, 16)$-group batch design with and without importance sampling. At every 100 iterations, the loss (10) on the full train dataset is recorded.

The result is shown in figure 2. We graph the train loss at every 100 iterations on the top and the recall@1 metric on the bottom. As expected, the train loss converges to a lower value when importance sampling is used. Also, the loss minimization is faster and more stable.

6.4. Number of examples per class in training data

When the number of examples per class in the training data is balanced among classes, the importance weights for (m, n)-group and (p)-random batch designs are homogeneous among positive pairs and negative pairs. To show that importance sampling handles training data with varying numbers of examples among classes so that the importance weights differ among example pairs, we train the balanced contrastive loss model on CUB200 dataset with artificial variability on the number of examples among classes. The variability is introduced by randomly removing some percentage of examples in a class where the percentage is randomly drawn from the interval from 0 to γ uniformly for each class. Note that $N_l(1-\gamma)$ number of examples are guaranteed to be retained for class l. If $\gamma = 0$, then no examples are removed. The variability on the numbers of examples increases as γ increases. It reaches the maximum when $\gamma = 1$. We train the model with increments of 0.1 on γ.

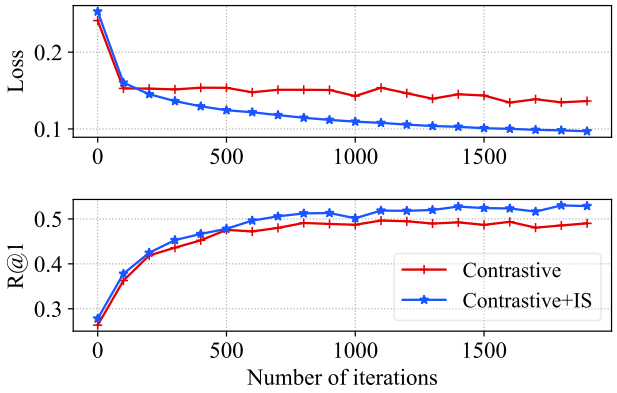

Figure 2. Experiment showing that importance sampling makes the training loss minimization more effective. Top graph shows the train loss at every 100 iterations for traditional contrastive loss with and without importance sampling. Bottom graph shows recall@1 at every 100 iterations. The train loss minimization is faster and more stable when importance sampling is used.

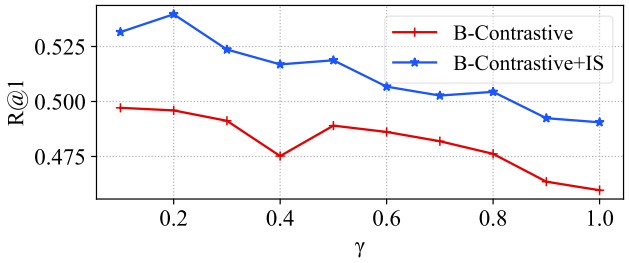

Figure 3. The recall@1 metrics are plotted against γ for the balanced contrastive loss with and without importance sampling. Importance sampling consistently boosts the metric.

The results are shown in figure 3. The recall@1 metric is plotted for balanced contrastive loss with and without importance sampling. As γ increases from 0 to 1, the metrics for both cases decrease. This is due to the decrease in the number of training data as γ increases. For $\gamma = 1$, the number of training data reduces approximately in half. What is notable is that applying the importance sampling boosts the performance consistently for all γ. This suggests that the importance weights are precise even when the number

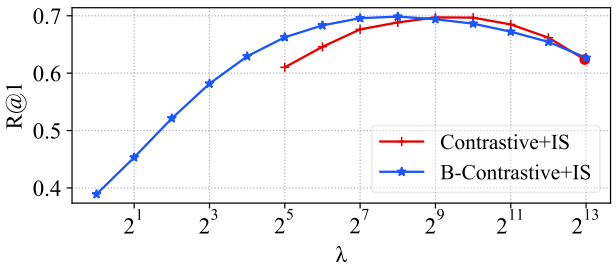

Figure 4. Recall@1 metric is plotted against λ for the balanced contrastive loss with importance sampling and against the induced λ for the contrastive loss with importance sampling.

of images are heterogeneous among classes in the training data.

6.5. Sensitivity analysis on λ

Recall that the hyperparameter λ in equation (16) controls the the number of negative examples compared for each positive example pair. We show experimentally that the model performance is not sensitive to λ. We train the balanced contrastive loss on the Stanford online product dataset with importance sampling varying λ from 2^0 to 2^{13}. To see that the semantic of λ of the number of negative exmaples compared is as claimed, we train the traditional contrastive loss with importance sampling with the induced λ varying from 2^5 to 2^{13}. Recall that the induced λ in (17) depends on the number of labels in the training data. To control the value, we artificially sample $\lambda + 1$ labels at every 100 iterations and use only examples of sampled labels for training.

The results are shown in figure 4. The recall@1 metric is plotted against λ for the balanced contrastive loss with importance sampling and the traditional contrastive loss with importance sampling. The horizontal axis is in log-scale. For λ in the range from 128 to 512, the metric is near the maximum. Experimenting with CUB200 and CARS196 datasets also gave the similar range. The performance degrades fast as λ approaches 0, which suggests that each positive pair should be compared with reasonably large number of negative examples. Although at a slower rate, the performance degrades as λ increases from the optimal point, which suggests that comparing with too many negative examples hurts. Interestingly, similar result holds for the traditional contrastive loss with induced λ.

6.6. Performance of the proposed method

Finally, we perform an extensive experiment on all three datasets and various methods. The results are shown in tables 2,3,4. The balanced contrastive loss with importance sampling (B-Contrastive+IS) outperforms by a large margin on CUB200 and CARS196 datasets. It matches the performance of other models on the Stanford online product dataset.

Method	R@1	R@2	R@4	R@8	NMI
Triplet [11]	50.98	62.61	73.09	82.68	59.82
Lifted [6]	50.91	63.28	74.29	83.31	60.20
Cluster [15]	51.40	63.50	74.66	84.37	60.78
N-pair [14]	51.45	63.79	75.07	84.23	60.93
Contrastive [3]	50.61	62.66	73.40	83.10	58.96
Contrastive+IS	53.34	65.41	76.01	85.20	61.48
B-Contrastive+IS	**54.22**	**66.42**	**76.37**	**85.23**	**62.69**

Table 2. Experiment on CUB200 dataset. Balanced contrastive loss with importance sampling outperforms other methods.

Method	R@1	R@2	R@4	R@8	NMI
Triplet [11]	56.82	68.90	78.85	86.98	54.26
Lifted [6]	63.81	74.73	83.18	89.63	60.89
Cluster [15]	60.95	72.70	82.20	88.80	57.17
N-pair [14]	66.82	78.50	86.07	91.65	62.32
Contrastive [3]	59.00	70.35	79.81	87.21	57.40
Contrastive+IS	66.77	76.44	84.61	90.62	61.97
B-Contrastive+IS	**69.18**	**79.40**	**86.79**	**91.80**	**62.89**

Table 3. Experiment on CARS196 dataset. Balanced contrastive loss with importance sampling outperforms other methods.

Method	R@1	R@2	R@4	R@8	NMI
Triplet [11]	**70.36**	**75.48**	79.60	83.09	89.95
Lifted [6]	68.78	74.47	79.14	83.21	89.68
Cluster [15]	65.31	71.10	75.91	80.14	88.66
N-pair [14]	69.48	75.05	**79.65**	**83.51**	89.12
Contrastive [3]	65.20	70.70	75.67	79.73	89.10
Contrastive+IS	62.38	67.89	72.70	76.72	88.03
B-Contrastive+IS	70.36	75.35	79.61	83.21	**90.12**

Table 4. Experiment on Stanford online product dataset. Balanced contrastive loss with importance sampling closely match the performance of other models.

7. Conclusion

We proposed a novel reweighting scheme for improving the contrastive loss based on a probabilistic approach and importance sampling. With importance sampling, we could study the effect of the loss function in isolation without being affected by sampling strategies. The balanced contrastive loss was the simplest loss function that we could use to illustrate this point. We expect more sophisticated loss functions will improve the performance of the model. Also, simple batch designs were used for the ease of calculation of the importance weights. More careful batch designs with corresponding importance weights may improve the performance further.

References

[1] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(Mar):1109–1135, 2010.

[2] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005.

[3] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[4] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] S. Hyun Oh, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[8] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[9] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. In *ICCV*, 2017.

[10] 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. *IJVC*, 2015.

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

[12] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2004.

[13] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *ICCV*, 2015.

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

[15] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy. Deep metric learning via facility location. In *CVPR*, 2017.

[16] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *NIPS*, 2014.

[17] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Closing the gap to human-level performance in face verification. deepface. In *CVPR*, 2014.

[18] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[19] 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.

[20] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2006.

[21] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl. Sampling matters in deep embedding learning. In *ICCV*, 2017.

[22] 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.