Deep Metric Learning to Rank

Fatih Cakir*[1], Kun He*[2], Xide Xia[2], Brian Kulis[2], and Stan Sclaroff[2]

[1]FirstFuel, fcakirs@gmail.com
[2]Boston University, {hekun,xidexia,bkulis,sclaroff}@bu.edu

Abstract

We propose a novel deep metric learning method by revisiting the learning to rank *approach. Our method, named FastAP, optimizes the rank-based Average Precision measure, using an approximation derived from distance quantization. FastAP has a low complexity compared to existing methods, and is tailored for stochastic gradient descent. To fully exploit the benefits of the ranking formulation, we also propose a new minibatch sampling scheme, as well as a simple heuristic to enable large-batch training. On three few-shot image retrieval datasets, FastAP consistently outperforms competing methods, which often involve complex optimization heuristics or costly model ensembles.*

1. Introduction

Metric learning [3, 18] is concerned with learning distance functions that conform to a certain definition of similarity. Often, in pattern recognition problems, the definition of similarity is task-specific, and the success of metric learning hinges on aligning its learning objectives with the task. In this paper, we focus on what is arguably the most important application area of metric learning: nearest neighbor retrieval. For nearest neighbor retrieval applications, the similarity definition typically involves neighborhood relationships, and nearly all metric learning methods follow the same guiding principle: the true “neighbors” of a reference object should be closer than its “non-neighbors” in the learned metric space.

Taking inspirations from the information retrieval literature, we treat metric learning as a *learning to rank* problem [22], where the goal is to optimize the total ordering of objects as induced by the learned metric. The learning to rank perspective has been adopted by classical metric learning methods [20,24], but has received less attention recently in deep metric learning. Working directly with ranked lists

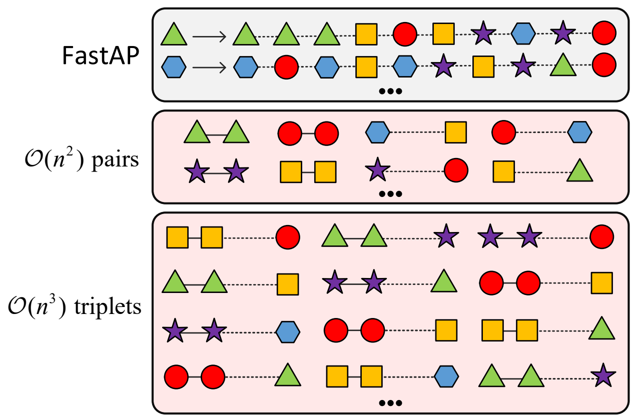


Figure 1: We propose FastAP, a novel deep metric learning method that optimizes Average Precision over ranked lists of examples. This solution avoids a high-order explosion of the training set, which is a common problem in existing losses defined on pairs or triplets. FastAP is optimized using SGD, and achieves state-of-the-art results.

has two primary advantages over many other approaches: we avoid the high-order explosion of the training set, and focus on orderings that are invariant to distance distortions. This second property is noteworthy in particular, as it allows circumventing the use of highly sensitive parameters such as distance thresholds and margins.

Our main contribution is a novel solution to optimizing Average Precision [4], a performance measure that has seen wide usage within and beyond information retrieval, such as in image retrieval [29], feature matching [14], and few-shot learning [38]. To tackle the highly challenging problem of optimizing this rank-based and non-decomposable objective, we employ an efficient quantization-based approximation, and tailor our algorithmic design for stochastic gradient descent. The result is a new deep metric learning method that we call FastAP.

We evaluate FastAP on three few-shot image datasets, and observe state-of-the-art retrieval results. Notably, with

*Equal contribution. Kun He is now with Facebook Reality Labs.

a single neural network, FastAP often significantly outperforms recent ensemble metric learning methods, which are costly to train and evaluate.

2. Related Work

Metric learning [3, 18] is a general umbrella term for learning distance functions in supervised settings. The distance functions can be parameterized in various ways, for example, a large body of the metric learning literature focuses on Mahalanobis distances [20, 24, 31, 41], which essentially learn a linear transformation of the input data. For nonlinear metric learning, solutions that employ deep neural networks have received much attention recently.

Aside from relatively few exceptions, *e.g.* [33, 39], deep metric learning approaches commonly optimize loss functions defined on pairs or triplets of training examples. Pair-based approaches, such as contrastive embedding [11], minimize the distance between a reference object and its neighbors, while ensuring a margin of separation between the reference and its non-neighbors. Alternatively, local ranking approaches based on triplet supervision [30, 31] optimize the relative ranking of two objects given a reference. These losses are also used to train individual learners in ensemble methods [16, 27, 42, 43].

Given a large training set, it is infeasible to enumerate all the possible pairs or triplets. This has motivated various mining or sampling heuristics aimed at identifying the "hard" pairs or triplets. A partial list includes lifted embedding [34], semi-hard negative mining [30], distance-weighted sampling [40], group-based sampling [2], and hierarchical sampling [8]. There are also other remedies such as adding auxiliary losses [15, 27] and generating novel training examples [26, 45]. Still, these approaches typically include nontrivial threshold or margin parameters that apply equally to all pairs or triplets, rendering them inflexible to distortions in the metric space.

Pair-based and triplet-based metric learning approaches can be viewed as instantiations of learning to rank [22], where the ranking function is induced by the learned distance metric. The learning to rank view has been adopted by classical metric learning methods with success [20, 24]. We revisit learning to rank for deep metric learning, and propose to learn a distance metric by optimizing Average Precision (AP) [4] over entire ranked lists. This is a listwise approach [5], and allows us to focus on the true quantity of interest: orderings in the metric space. AP naturally emphasizes performance at the top, predicts other metrics well [1], and has found wide usage in various metric learning applications [13, 14, 21, 29, 38].

Average Precision has also been studied considerably as an objective function for learning to rank. However, its optimization is highly challenging, as it is non-decomposable over ranked items, and differentiating through the discrete sorting operation is difficult. One notable optimization approach is based on smoothed rankings [7, 37], which considers the orderings to be random variables, allowing for probabilistic and differentiable formulations. However, the probabilistic orderings are expensive to estimate. Alternatively, the well-known SVM-MAP [44] optimizes the structured hinge loss surrogate using a cutting plane algorithm, and the direct loss minimization framework [25,35] approximates the gradients of AP in an asymptotic manner. These methods critically depend on the loss-augmented inference to generate cutting planes or approximate gradients, which can scale quadratically in the list size.

We propose a novel approximation of Average Precision that only scales linearly in the list size, using distance quantization and a differentiable relaxation of histogram binning [39]. For the special case of binary embeddings, [13] uses a histogram-based technique to optimize a closed-form expression of AP for Hamming distances. [14] subsequently extends it to the real-valued case for learning local image descriptors, by simply dropping the binary constraints. However, doing so implies a different underlying distance metric than Euclidean, thus creating a mismatch. In contrast, our solution directly targets the Euclidean metric in a general metric learning setup, derives from a different probabilistic interpretation of AP, and is capable of large-batch training beyond GPU memory limits.

3. Learning to Rank with Average Precision

We assume a standard information retrieval setup, where given a feature space $\mathcal{X}$, there is a query $q \in \mathcal{X}$ and a retrieval set $\mathcal{R} \subset \mathcal{X}$. Our goal is to learn a deep neural network $\Psi : \mathcal{X} \to \mathbb{R}^m$ that embeds inputs to an m-dimensional Euclidean space, and is optimized for Average Precision under the Euclidean metric.

To perform nearest neighbor retrieval, we first rank items in $\mathcal{R}$ according to their distances to q in the embedding space, producing a ranked list $\{x_1, x_2, \ldots, x_N\}$ with $N = |\mathcal{R}|$. Then, we derive the precision-recall curve as:

$$\text{PR}(q) = \{(\mathsf{Prec}(i), \mathsf{Rec}(i)), i = 0, \ldots, N\}, \tag{1}$$

where $\mathsf{Prec}(i)$ and $\mathsf{Rec}(i)$ are the precision and recall evaluated at the i-th position in the ranking, respectively. The Average Precision (AP) of the ranking, can then be evaluated as:

$$\begin{aligned} \text{AP} &= \sum_{i=1}^{N} \mathsf{Prec}(i) \triangle \mathsf{Rec}(i) & (2) \\ &= \sum_{i=1}^{N} \mathsf{Prec}(i)(\mathsf{Rec}(i) - \mathsf{Rec}(i-1)). & (3) \end{aligned}$$

Note that we assume $\mathsf{Prec}(0) = \mathsf{Rec}(0) = 0$ for convenience.

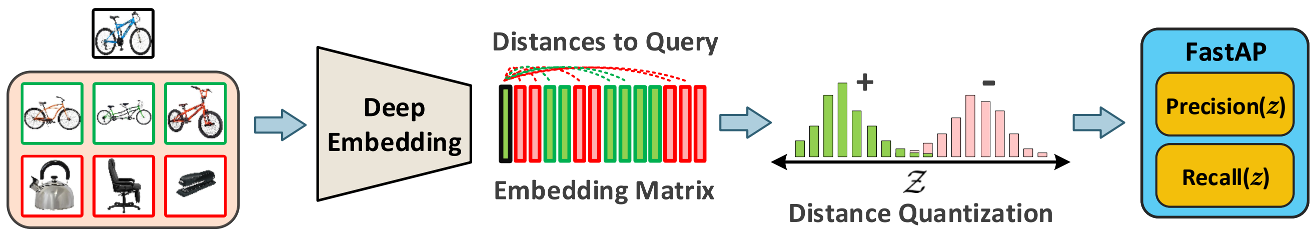


Figure 2: The discrete sorting operation prevents learning to rank methods from directly applying gradient optimization on standard rank-based metrics. FastAP approximates Average Precision by exploiting distance quantization, which reformulates precision and recall as parametric functions of distance, and permits differentiable relaxations. Our approximation derives from the interpretation of AP as the area under the precision-recall curve.

A problem with the above definition of AP is that to obtain the precision-recall curve, the ranked list first needs to be generated, which involves the discrete sorting operation. Sorting is a major hurdle for gradient-based optimization: although it is differentiable almost everywhere, the derivatives are either zero or undefined. Instead, our main insight is that there exists an alternative interpretation for AP, and it is based on representing precision and recall as functions of *distance*, rather than ranked items.

3.1. FastAP: Efficient AP Approximation

In the information retrieval literature, AP is often also interpreted as the area under the precision-recall curve (AUPR) [4]. Such a relation exists since (3) asymptotically approaches AUPR when the neighbor set cardinality goes to infinity:

$$\text{AUPR} = \int_0^1 \text{Prec}(\text{Rec})\, d\,\text{Rec} \tag{4}$$

$$= \lim_{|\mathcal{R}^+|\to\infty} \sum_{i=1}^{N} \text{Prec}(i) \triangle \text{Rec}(i). \tag{5}$$

where $\mathcal{R}^+, (\mathcal{R}^-) \subset \mathcal{R}$ denotes the (non) neighbor set of query q. The AUPR interpretation of AP is particularly interesting as it allows viewing precision and recall as parametric functions of distance, rather than ranked items. As we will show, this will help us circumvent the non-differentiable sorting operation, and develop an efficient approximation of AP.

Formally, a continuous precision-recall curve (as opposed to the finite set in (1)) can be defined as

$$\text{PR}_z(q) = \{(\text{Prec}(z), \text{Rec}(z)), z \in \Omega\}, \tag{6}$$

where z denotes distance values between the query and items in $\mathcal{R}$, with domain Ω. With this change of variables, AP becomes:

$$\text{AP} = \int_\Omega \text{Prec}(z)\, d\,\text{Rec}(z). \tag{7}$$

We next define some probabilistic quantities so as to evaluate (7). Let $\mathcal{Z}$ be the random variable corresponding to distances z. Then, the distance distributions for $\mathcal{R}^+$ and $\mathcal{R}^-$ are denoted as $p(z|\mathcal{R}^+)$ and $p(z|\mathcal{R}^-)$. Let $P(\mathcal{R}^+)$ and $P(\mathcal{R}^-) = 1 - P(\mathcal{R}^+)$ denote prior probabilities, which indicate the skewness of the retrieval set $\mathcal{R}$ with respect to the query. Finally, let $F(z) = P(\mathcal{Z} < z)$ denote the cumulative distribution function (CDF) for $\mathcal{Z}$.

Given these definitions, we redefine precision and recall as follows:

$$\text{Prec}(z) = P(\mathcal{R}^+|\mathcal{Z} < z) = \frac{P(\mathcal{Z} < z|\mathcal{R}^+)P(\mathcal{R}^+)}{P(\mathcal{Z} < z)} \tag{8}$$

$$= \frac{F(z|\mathcal{R}^+)P(\mathcal{R}^+)}{F(z)}, \tag{9}$$

$$\text{Rec}(z) = P(\mathcal{Z} < z\,|\mathcal{R}^+) = F(z|\mathcal{R}^+). \tag{10}$$

Substituting these terms in (7), we get:

$$\text{AP} = \int_\Omega P(\mathcal{R}^+|\mathcal{Z} < z)\, dP(\mathcal{Z} < z|\mathcal{R}^+) \tag{11}$$

$$= \int_\Omega \frac{F(z|\mathcal{R}^+)P(\mathcal{R}^+)}{F(z)} p(z|\mathcal{R}^+)\, dz. \tag{12}$$

Note that we have used the fact that $dP(\mathcal{Z} < z|\mathcal{R}^+) = p(z|\mathcal{R}^+)\, dz$, *i.e.*, the derivative of the CDF is its corresponding PDF.

It should be clear now that (12) can also be approximately evaluated using finite sums. We first assume that the output of the embedding function Ψ is L_2-normalized, so that Ω, or the support of the distributions in (12), is a bounded range $[0, 2]$. Then, we quantize the interval $[0, 2]$ using a finite set $Z = \{z_1, z_2, \ldots, z_L\}$, and denote the resulting discrete PDF as P. Finally, we name our new approximation **FastAP**:

$$\text{FastAP} = \sum_{z\in Z} \frac{F(z|\mathcal{R}^+)P(\mathcal{R}^+)}{F(z)} P(z|\mathcal{R}^+). \tag{13}$$

We next re-express FastAP using histogram notation. Specifically, we create a distance histogram with bins centered on each element of Z. Let h_j be the number of items that fall into the j-th bin, and let $H_j = \sum_{k \le j} h_k$ be the cumulative sum of the histogram. Also, let h_j^+ count the number of neighbors of q in the j-th bin, and H_j^+ be its cumulative sum. With these definitions, we can rewrite the probabilistic quantities in (13), and get a simple expression for FastAP:

$$\text{FastAP} = \sum_{j=1}^{L} \frac{\frac{H_j^+}{N_q^+} \cdot \frac{N_q^+}{N}}{\frac{H_j}{N}} \cdot \frac{h_j^+}{N_q^+} = \frac{1}{N_q^+} \sum_{j=1}^{L} \frac{H_j^+ h_j^+}{H_j}. \quad (14)$$

It takes $O(NL)$ time to perform histogram binning and compute the FastAP approximation. A small L suffices in practice, as we will show in the experiments section.

4. Stochastic Optimization

In this section, we describe how to optimize FastAP (14) using SGD. AP is defined with respect to the retrieval problem between a query and a retrieval set. With minibatches, the natural choice is to define *in-batch* retrieval problems where retrieval set $\mathcal{R}$ is restricted to examples in the minibatch. Specifically, we use each example as the query q to retrieve its neighbors from the rest of the batch. Each of these in-batch retrieval problems emits one AP value, and the overall objective of the minibatch is the average of them, or the mean AP (mAP).

To perform gradient descent, we must ensure the histograms in (14) are constructed as to permit gradient backpropagation. To this end, we adopt the simple linear interpolation technique proposed by [39] which replaces the regular binning operation for histogram construction with a differentiable soft binning technique. Essentially, this interpolation relaxes the integer-valued histogram bin counts h to continuous values, which we denote using $\hat{h}$. The cumulative sums are also relaxed as $\hat{H}$. With a differentiable binning operation, we can now obtain partial derivatives for FastAP. Specifically, using subscript i to indicate that the current query is x_i, we have:[†]

$$\frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,k}^+} = \frac{1}{N_i^+} \sum_{j=1}^{L} \frac{\partial}{\partial \hat{h}_{i,k}^+} \left(\frac{\hat{H}_{i,j}^+ \hat{h}_{i,j}^+}{\hat{H}_{i,j}} \right) \quad (15)$$

$$= \frac{1}{N_i^+} \sum_{j \ge k} \frac{\hat{H}_{i,j} \hat{h}_{i,j}^+ + \hat{H}_{i,j}^- \hat{h}_{i,j}^+}{\hat{H}_{i,j}^2}. \quad (16)$$

The relaxation of histogram binning is also used by [6, 13] to tackle the "leaning to hash" problem, with similar in-batch retrieval formulations. FastAP differs in two main aspects: the objective function, and the underlying distance metric. In particular, [13] optimizes a complex closed form of AP for the Hamming distance. In this case, the number of histogram bins naturally corresponds to the number of discrete levels in the Hamming distance. However, such a convenience does not exist for real-valued distances. Instead, histogram binning is used for approximation purposes in FastAP. The number of histogram bins now becomes a variable parameter, which involves an interesting trade-off, as we will discuss later.

[†]The complete derivations are provided in the supplementary material.

4.1. Large-Batch Training

Large batch sizes can be beneficial for training deep neural networks with SGD [32]; this is also observed in our experiments (Section 5.3). However, batch sizes are limited by GPU memory. In the case of classification, the effective batch size can be increased through data parallelism [10]. However, data parallelism for FastAP is less trivial, as it is a *non-decomposable* objective: the objective value for each example in a minibatch depends on other examples in the same batch.

Inspired by a similar solution for the triplet loss [9], we propose a heuristic to enable large-batch training for FastAP. The main insight is that the loss layer takes the embedding matrix of the minibatch as input (see supplementary material). Thus, a large batch can be first broken into smaller chunks to incrementally compute the embedding matrix. Then, we compute the gradients with respect to the embedding matrix, which is a relatively lightweight operation involving only the loss layer. Finally, gradients are back-propagated through the network, again in chunks. This solution works even with a single GPU.

4.2. Minibatch Sampling

Typically, metric learning methods derive neighborhood relationships from class labels – instances sharing the same class label are considered neighbors. In this scheme, sampling strategies for pairs and triplets have been well-studied. However, the listwise formulation of FastAP leads to different considerations for minibatch sampling. The first consideration is that sampling should be done on the class level; instance-level sampling cannot guarantee that each example has at least one neighbor in the same minibatch, thus might lead to ill-defined in-batch retrieval problems.

The second consideration is that the sampled minibatches need to represent "sufficiently hard" in-batch retrieval problems to train the network. One option, as we mentioned above, is to use large batches. However, when training with large batches is not feasible, a sampling strategy for classes becomes crucial. Our reasoning, illustrated in Figure 3, is that sampling classes purely at random may not create hard retrieval problems: for example, it is easy to retrieve images of a bicycle from a pool of toasters, chairs, *etc*. However, if the pool also includes images of other bicycles, the retrieval problem becomes more challenging.

Rather than treating all classes as equally different from

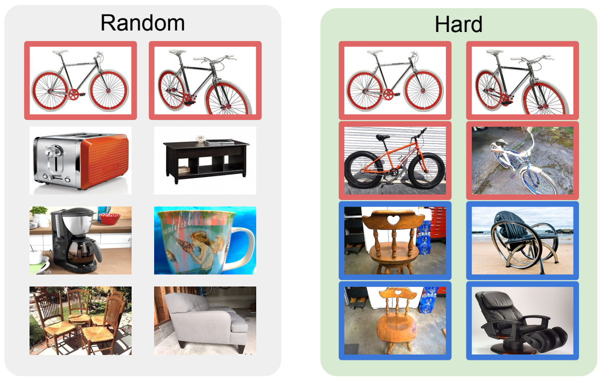


Figure 3: Minibatch sampling: examples of the random strategy and our hard strategy. Our strategy constructs more challenging in-batch retrieval problems by sampling classes from a small number of categories in each minibatch.

each other, we would like to utilize any available side information on the similarities between classes. One example of such information is WordNet similarity [28] for ImageNet class labels. Alternatively, for the datasets considered in our experiments, *category* labels are available: classes belonging to the same category are more similar. For example, a category can be "bicycle" while class labels correspond to individual bicycle instances. Following this intuition, we develop a category-based sampling strategy as follows: each minibatch first samples a small number of categories, *e.g*. bicycle and chair, and then samples individual classes from them. Experimentally, this "hard" sampling strategy consistently outperforms sampling classes at random, and therefore is used in all experiments reported in Section 5. Please refer to the supplementary material for more details.

5. Experiments

We evaluate our metric learning method, FastAP, on three standard image retrieval datasets that are commonly used in the deep metric learning literature. These datasets are: Stanford Online Products, In-Shop Clothes Retrieval, and PKU VehicleID.

- **Stanford Online Products** is proposed in [34] for evaluating deep metric learning algorithms. It contains 120,053 images of 22,634 online products from eBay.com, where each product is annotated with a distinct class label. Each class has 5.3 images on average. Following [34], we use 59,551 images from 11,318 classes for training, and the remaining 60,502 images from 11,316 classes for testing.

- **In-Shop Clothes Retrieval** [23] is another popular dataset in image retrieval. Following the setup in [23], 7,982 classes of clothing items with 52,712 images are used in experiments. Among them, 3,997 classes are for training (25,882 images) and 3,985 classes are for testing (28,760 images). The test set is partitioned into a query set and a gallery set, where the query set contains 14,218 images of 3,985 classes, and the gallery set contains 12,612 images of 3,985 classes. At test time, given an image in the query set, we retrieve its neighbors from the gallery set.

- **PKU VehicleID** [21] is a dataset of 221,763 images from 26,267 vehicles captured by surveillance cameras. The training set has 110,178 images of 13,134 vehicles and the test set has 111,585 images of 13,133 vehicles. This dataset is particularly challenging as different vehicle identities are considered as different classes even if they share the same model. We follow the standard experimental protocol [21] to test on the small, medium and large test sets, which contain 7,332 images of 800 vehicles, 12,995 images of 1,600 vehicles, and 20,038 images of 2,400 vehicles, respectively.

These datasets all have a limited number of images per class, which results in challenging few-shot retrieval problems. Also, as we mentioned, all three datasets provide high-level category labels: Stanford Online Products contains 12 product categories such as "bicycle", "chair", *etc*. For In-Shop Clothes Retrieval, each clothing item belongs to one of 23 categories such as "MEN/Denim" and "WOMEN/Dresses". For PKU VehicleID, each category corresponds to a unique vehicle model.

5.1. Experimental Setup

We consider the binary relevance setup where images with the same class label are neighbors, and non-neighbors otherwise. We report a standard retrieval metric, Recall at k (R@k), defined as the percentage of queries having at least one neighbor retrieved in the first k results.

We fine-tune ResNet [12] models pretrained on ImageNet, and replace the final softmax classification layer with a fully-connected embedding layer, with random initialization. We experiment with both ResNet-18 and ResNet-50, and set the embedding dimensionality to 512 by default. The embeddings are normalized to have unit L_2 norm. In all experiments, we use the Adam optimizer [17] with base learning rate 10^{-5} and no weight decay, and amplify the embedding layer's learning rate by 10 times. Following standard practice, images in all datasets are resized to 256×256, and the embedding network takes crops of size 224×224 as input. Random crops and random flipping are used during training for data augmentation, and a single center crop is used at test time.

Our experiments are run on an NVIDIA Titan X Pascal GPU with 12GB memory, which permits a batch size

Method	Dim.	Stanford Online Products			
		R@1	R@10	R@100	R@1000
LiftStruct [34]	512	62.1	79.8	91.3	97.4
Histogram [39]	512	63.9	81.7	92.2	97.7
Clustering [33]	64	67.0	83.7	93.2	–
Spectral [19]	512	67.6	83.7	93.3	–
Hard-aware Cascade [43]†	384	70.1	84.9	93.2	97.8
Margin [40]	128	72.7	86.2	93.8	98.0
BIER [27]†	512	72.7	86.5	94.0	98.0
Proxy NCA [26]	64	73.7	–	–	–
A-BIER [27] †	512	74.2	86.9	94.8	98.2
Hierarchical Triplets [8]	512	74.8	88.3	94.8	98.4
ABE-8 [16]†	512	76.3	88.4	94.8	98.2
FastAP, ResNet-18, $M = 256$	512	73.2	86.8	94.1	97.8
FastAP, ResNet-50, $M = 96$	128	73.8	88.0	94.9	98.3
FastAP, ResNet-50, $M = 96$	512	75.8	**89.1**	**95.4**	**98.5**
FastAP, ResNet-50, $M = 256$ ‡	512	**76.4**	89.0	95.1	98.2

† Ensemble method
‡ Large-batch training heuristic to overcome GPU memory limit

Table 1: Retrieval performance comparison on Stanford Online Products [34]. FastAP achieves state-of-the-art results, outperforming competing methods with either a simpler architecture or fewer embedding dimensions. M: batch size.

of $M = 256$ for ResNet-18, and $M = 96$ for ResNet-50. Our large-batch heuristic further enables training with arbitrary batch sizes. An ablation study on the batch size is also included in Section 5.3.

5.2. Comparison with State-of-the-art

We compare FastAP to a series of state-of-the-art deep metric learning methods. Most of these work either exclusively use pair-based or triplet-based local ranking losses [2, 8, 34, 40, 43, 45], or use them in ensembles [16, 27, 42]. The exceptions include [19, 33] where clustering objectives are optimized, as well as Histogram [39] that proposes a quadruplet-based loss. In addition, some methods also propose novelties other than the loss function: Proxy NCA [26] and HTG [45] generate novel training examples to improve triplet-based training, and ABE [16] employs an attention mechanism.

Stanford Online Products

We present R@k results for $k \in \{1, 10, 100, 1000\}$ on Stanford Online Products in Table 1. With a single ResNet-50 and 512 embedding dimensions, FastAP obtains state-of-the-art results, both with and without large-batch training. FastAP is also very competitive with 128 embedding dimensions, for example, it significantly outperforms Margin [40], a leading triplet-based method equipped with a principled sampling strategy. In fact, only HTL [8] and ABE-8 [16], both of which learn 512-d embeddings, are able to achieve better overall performance than the 128-d embeddings learned by FastAP.

We also compare FastAP to the ensemble methods, namely, HDC [43], BIER/A-BIER [27], and ABE-8 [16]. These methods combine embedding vectors obtained either from different layers in the same network, or from different networks. Ensemble models can be very demanding to train: for example, BIER ensembles 3 learners with a different loss in each, and A-BIER extends it with the addition of adversarial loss. Next, ABE-8 is an ensemble of 8 different learners, trained on a GPU with 24GB memory. In contrast, FastAP trains a single embedding network with a single loss function, and outperforms these methods when using 12GB of GPU memory. Therefore, the complexity-performance trade-off for FastAP is much more desirable.

It is also noteworthy to contrast FastAP with Histogram [39], which first proposed the differentiable relaxation of histogram binning for deep metric learning. The histogram loss is a quadruplet-based loss that encourages the distance distributions of neighbors and non-neighbors to be separated. However, this loss is only loosely correlated with retrieval performance metrics, and we suspect that designing appropriate sampling strategies for quadruplets is even more challenging than for triplets. FastAP strongly outperforms the histogram loss.

In-Shop Clothes Retrieval

On the In-Shop Clothes Retrieval dataset, we add two methods into our comparisons: the original FashionNet [23] that learns clothing features by predicting landmark locations and multiple attributes, and a recently proposed ensemble model named DREML [42].

Method	Dim.	In-Shop Clothes Retrieval					
		R@1	R@10	R@20	R@30	R@40	R@50
FashionNet [23]	4096	53.0	73.0	76.0	77.0	79.0	80.0
Hard-aware Cascade [43]†	384	62.1	84.9	89.0	91.2	92.3	93.1
BIER [27]†	512	76.9	92.8	95.2	96.2	96.7	97.1
DREML [42]†	9216	78.4	93.7	95.8	96.7	–	–
Hard Triplet Generation [45]	–	80.3	93.9	95.8	96.6	97.1	–
Hierarchical Triplets [8]	128	80.9	94.3	95.8	97.2	97.4	97.8
A-BIER [27]†	512	83.1	95.1	96.9	97.5	97.8	98.0
ABE-8 [16]†	512	87.3	96.7	97.9	98.2	98.5	98.7
FastAP, ResNet-18, $M = 256$	512	89.0	97.2	98.1	98.5	98.7	98.9
FastAP, ResNet-50, $M = 96$	512	90.0	97.5	98.3	98.7	**98.9**	**99.1**
FastAP, ResNet-50, $M = 256$‡	512	**90.9**	**97.7**	**98.5**	**98.8**	**98.9**	**99.1**

† Ensemble method
‡ Large-batch training heuristic to overcome GPU memory limit

Table 2: Retrieval performance comparison on the In-Shop Clothes Retrieval dataset [23]. Both the ResNet-18 and ResNet-50 versions of FastAP outperform all competing methods.

Method	Dim.	PKU VehicleID					
		Small		Medium		Large	
		R@1	R@5	R@1	R@5	R@1	R@5
Mixed Diff+CCL [21]	1024	49.0	73.5	42.8	66.8	38.2	61.6
GS-TRS [2]	1024	75.0	83.0	74.1	82.6	73.2	81.9
BIER [27]†	512	82.6	90.6	79.3	88.3	76.0	86.4
A-BIER [27]†	512	86.3	92.7	83.3	88.7	81.9	88.7
DREML [42]†	2304	88.5	94.8	87.2	94.2	83.1	92.4
FastAP, ResNet-18, $M = 256$	512	90.9	96.0	88.9	95.2	85.3	93.9
FastAP, ResNet-50, $M = 96$	512	90.4	96.5	88.0	95.4	84.5	93.9
FastAP, ResNet-50, $M = 256$‡	512	**91.9**	**96.8**	**90.6**	**95.9**	**87.5**	**95.1**

† Ensemble method
‡ Large-batch training heuristic to overcome GPU memory limit

Table 3: Retrieval performance comparison on PKU VehicleID [21]. FastAP performs significantly better than recent ensemble models which are costly to train and evaluate.

The R@k results for In-Shop Clothes Retrieval are presented in Table 2, with $k \in \{1, 10, 20, 30, 40, 50\}$. FastAP outperforms all competing methods with a clear margin. Notably, we point out that for this dataset, DREML uses a large ensemble of 48 ResNet-18 models, each producing a 198-d embedding vector, resulting in a 9216-d embedding when concatenated. With a single ResNet-18 and 512-d embeddings, FastAP obtains a 15% relative increase in R@1 compared to DREML, and also outperforms two strong ensemble models, A-BIER and ABE-8.

PKU VehicleID

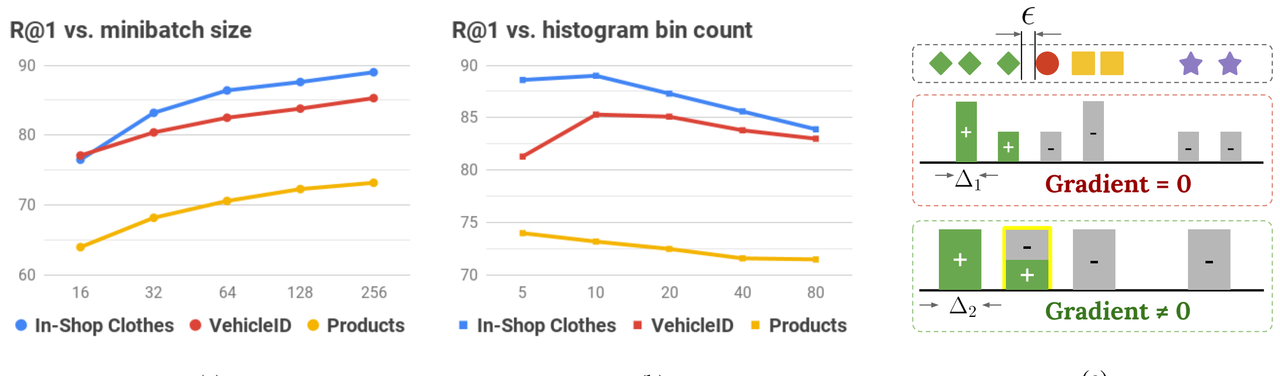
We report R@k for $k \in \{1, 5\}$ on the small, medium, and large test sets of PKU VehicleID. We include the original baseline, Mixed Diff+CCL from [21], as well as GS-TRS [2] which proposes a group-based triplet method to reduce intra-class variance. Other included methods that report results on this dataset are ensemble methods: BIER, A-BIER, and DREML. For this dataset, DREML employs an ensemble of 12 ResNet-18 models with a 192-d embedding from each model, resulting in 2304-d embeddings.

Table 3 presents retrieval performance comparisons. FastAP again is able to outperform the state-of-the-art with both ResNet-18 and ResNet-50. An interesting observation is that unlike on other datasets, when using the same amount of GPU memory, ResNet-18 consistently outperforms ResNet-50 in R@1. We hypothesize that since the ResNet-18 model is able to use larger batches, the increased hardness of the in-batch retrieval problems partially outweighs its lower model capability on this dataset.

5.3. Ablation Studies

Batch Size

During SGD training, the list size in the in-batch retrieval problems is determined by the minibatch size. Here, we



(a) (b) (c)

Figure 4: Ablation studies. We monitor R@1 as a function of hyper-parameters on all datasets. (a) FastAP benefits from using large batch sizes. (b) Distance quantization induces a trade-off between close approximation of AP *vs*. "hardness" of the resulting objective, and peak performance is observed around 10 bins. (c) Illustration of the trade-off.

present an ablation study where we vary the training batch size for FastAP, with the ResNet-18 backbone. We measure the R@1 on all three datasets (for VehicleID we use the large subset).

As provided in Figure 4a, R@1 monotonically improves with larger batch size on all three datasets. This observation resonates with the fact that large batches reduce the variance of the stochastic gradients, which has been shown to be beneficial [32]. On the other hand, from the learning to rank perspective, we argue that larger batches result in harder in-batch retrieval problems during training, since the network is required to rank the neighbors in front of a larger set of non-neighbors, and this in turn leads to better generalization.

Distance Quantization

A hyper-parameter of FastAP is the number of histogram bins used in distance quantization, which controls the quality of approximation. To study its effects, we also run an ablation study with varying numbers of histogram bins during training, keeping other parameters fixed (ResNet-18, batch size 256). Figure 4b shows the impact on test set performance in terms of R@1.

Intuitively, using more histogram bins during training would result in a closer approximation of Average Precision. However, we observe that retrieval performance does not necessarily improve with more bins, and in fact consistently peaks around 10 bins. To understand this trade-off, we give a simple example in Figure 4c, where a ranked list achieves perfect AP, with a small margin of separation ϵ between positive and negative examples. A fine-grained quantization of this ranked list produces all-positive histogram bins, followed by all-negative ones, which means that the FastAP approximation also evaluates to 1. In this case, the gradients are zero, *i.e*., there are no learning signals for the network. In contrast, we argue that a coarser quantization would produce histogram bins where positives and negatives are mixed, thus generating nonzero gradients to further push them apart, which helps generalization.

The width of histogram bins has a similar effect as the margin parameter in triplet losses, and it is desirable to keep it reasonably large, or equivalently, the number of bins relatively small. On the other hand, if there are too few bins in the histogram, approximation quality also deteriorates; in the extreme case of there being only one bin, no learning is possible. In our experiments, 10-bin histograms usually provide the best trade-off, and performance is not overly sensitive to this parameter.

6. Conclusion

We revisit the "learning to rank" principle to propose a deep metric learning method, FastAP. Our main contribution is a novel solution to optimizing Average Precision under the Euclidean metric, based on the probabilistic interpretation of AP as the area under precision-recall curve, as well as distance quantization. Compared to many existing solutions to this much-studied problem, FastAP is more efficient, and works in a stochastic setting by design. We further propose a category-based minibatch sampling strategy and a large-batch training heuristic. On three standard datasets, FastAP consistently outperforms the current state-of-the-art in few-shot image retrieval, and demonstrates an excellent performance-complexity trade-off.

Acknowledgements

This work is conducted at Boston University, supported in part by a BU IGNITION award, and equipment donated by NVIDIA.

References

[1] Javed A. Aslam, Emine Yilmaz, and Virgiliu Pavlu. The maximum entropy method for analyzing retrieval measures. In *Proc. ACM SIGIR Conference on Research & Development in Information Retrieval*, 2005.

[2] Yan Bai, Feng Gao, Yihang Lou, Shiqi Wang, Tiejun Huang, and Ling-Yu Duan. Incorporating intra-class variance to fine-grained visual recognition. In *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 2017.

[3] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[4] Kendrick Boyd, Kevin H. Eng, and C. David Page. Area under the precision-recall curve: Point estimates and confidence intervals. In *Machine Learning and Knowledge Discovery in Databases*, 2013.

[5] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proc. International Conference on Machine Learning (ICML)*, 2007.

[6] Fatih Çakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. Hashing with mutual information. *arXiv preprint arXiv:1803.00974*, 2018.

[7] Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, 13(3):216–235, 2010.

[8] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep metric learning with hierarchical triplet loss. In *Proc. European Conference on Computer Vision (ECCV)*, 2018.

[9] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus. End-to-end learning of deep visual representations for image retrieval. *International Journal of Computer Vision (IJCV)*, 124(2):237–254, 2017.

[10] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[11] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] Kun He, Fatih Çakir, Sarah Adel Bargal, and Stan Sclaroff. Hashing as tie-aware learning to rank. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] Kun He, Yan Lu, and Stan Sclaroff. Local descriptors optimized for average precision. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[15] Xinwei He, Yang Zhou, Zhichao Zhou, Song Bai, and Xiang Bai. Triplet-center loss for multi-view 3d object retrieval. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[16] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *Proc. European Conference on Computer Vision (ECCV)*, 2018.

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

[18] Brian Kulis. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[19] Marc T. Law, Raquel Urtasun, and Richard S. Zemel. Deep spectral clustering learning. In *Proc. International Conference on Machine Learning (ICML)*, 2017.

[20] Daryl Lim and Gert R. Lanckriet. Efficient learning of Mahalanobis metrics for ranking. In *Proc. International Conference on Machine Learning (ICML)*, 2014.

[21] Hongye Liu, Yonghong Tian, Yaowei Yang, Lu Pang, and Tiejun Huang. Deep relative distance learning: Tell the difference between similar vehicles. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[23] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] Brian McFee and Gert R. Lanckriet. Metric learning to rank. In *Proc. International Conference on Machine Learning (ICML)*, 2010.

[25] Pritish Mohapatra, Michal Rolínek, C.V. Jawahar, Vladimir Kolmogorov, and M. Pawan Kumar. Efficient optimization for rank-based loss functions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[26] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[27] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. Deep metric learning with BIER: Boosting independent embeddings robustly. *arXiv preprint arXiv:1801.04815*, 2018.

[28] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. WordNet::Similarity: measuring the relatedness of concepts. In *Proceedings of Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2004.

[29] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[30] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[31] Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[32] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *ICLR*, 2018.

[33] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[34] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[35] Yang Song, Alexander Schwing, Richard S. Zemel, and Raquel Urtasun. Training deep neural networks via direct loss minimization. In *Proc. International Conference on Machine Learning (ICML)*, 2016.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[37] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proc. ACM International Conference on Web Search and Data Mining*, 2008.

[38] Eleni Triantafillou, Richard S. Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[39] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[40] Chao-Yuan Wu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[41] Eric P. Xing, Michael I. Jordan, Stuart J. Russell, and Andrew Y. Ng. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[42] Hong Xuan, Richard Souvenir, and Robert Pless. Deep randomized ensembles for metric learning. In *Proc. European Conference on Computer Vision (ECCV)*, 2018.

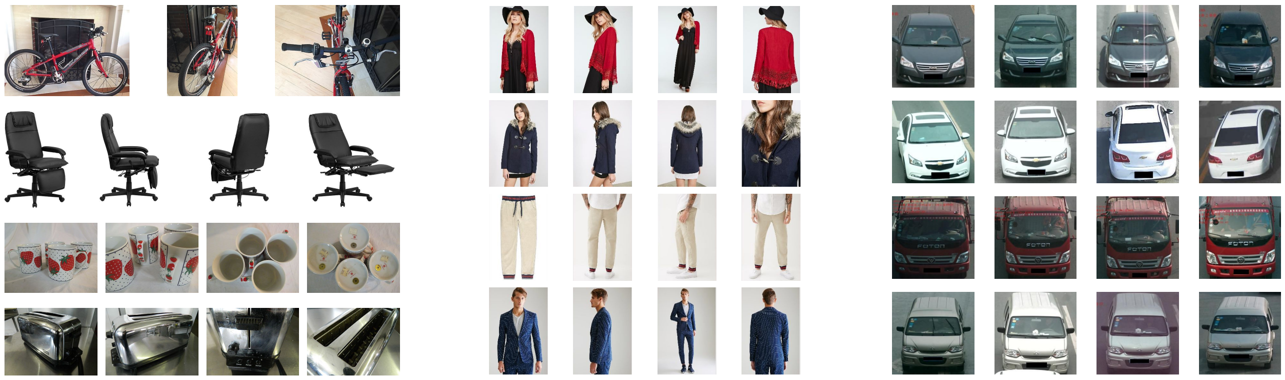
[43] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. Hard-aware deeply cascaded embedding. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[44] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proc. ACM SIGIR Conference on Research & Development in Information Retrieval*, 2007.

[45] Yiru Zhao, Zhongming Jin, Guo-Jun Qi, Hongtao Lu, and Xian-Sheng Hua. An adversarial approach to hard triplet generation. In *Proc. European Conference on Computer Vision (ECCV)*, 2018.

Supplementary Material

A. Minibatch Sampling



(a) Stanford Online Products (b) In-Shop Clothes Retrieval (c) PKU VehicleID

Figure 5: Examples from the three datasets used in the paper. Each row corresponds to a distinct class.

As mentioned in the paper, FastAP uses class-based sampling. Moreover, under the few-shot setup where the number of examples per class is small, a minibatch will include all images from a class once that class gets sampled. Example classes from the three datasets are shown in Figure 5. For minibatch sampling, we compare two strategies. Below, let the minibatch size be M.

- **Random**: Randomly sample classes from the entire training set, until M images are obtained.
- **Hard**: First sample two categories. From each category, randomly sample classes until $M/2$ images are obtained.

Suppose we have sampled a class with n images in some minibatch. Now consider the in-batch retrieval problem where the query is an image x from this class. As shown in Figure 6, under the hard sampling strategy, there exist $(n-1)$ neighbors of x in the database, and another $(M/2-n)$ images from different classes in the same category, which we call the *hard negatives*, in the sense that it is generally harder to distinguish between classes in the same category. The remaining $M/2$ images are from a different category, and are referred to as the *easy negatives*. Due to the balanced batch construction, every in-batch retrieval problem shares the same structure, and only n may vary depending on the class. In contrast, under the random strategy, the chances of seeing hard negatives are much lower, especially when the number of categories is large.

Below we describe implementation details for each dataset. Information regarding the category labels is given in Table 4.

- **Stanford Online Products**: Each class belongs to one of 12 categories. In each training epoch, we iterate over all the pairs of different categories, and sample 5 minibatches for each pair. The number of minibatches per epoch is $\binom{12}{2} \times 5 = 330$.
- **In-Shop Clothes Retrieval**: Each class belongs to one of 23 categories. We take the same approach as above, except with 2 minibatches per category pair. The number of minibatches per epoch is $\binom{23}{2} \times 2 = 506$.
- **PKU VehicleID**: This dataset is slightly different in that not every class has a category label. A total of 250 categories (vehicle models) are labeled, covering roughly half of the training set. We take a different approach for this dataset: in each minibatch, we first sample $M/2$ images from a labeled category. Then, to get the other $M/2$ images, we randomly sample classes that do not have category labels. This way, each epoch generates $\sum_{c\in\mathcal{C}} 2N_c/M$ minibatches, where $\mathcal{C}$ denotes the set of labeled categories, and N_c is the total number of images in category c.

Although the hard strategy can be generalized to sample from more than two categories at a time, doing so reduces the number of hard negatives in the minibatches, and does not lead to performance gains in our experiments. Also, we observe that performance degrades when we only sample from one category in each minibatch. An explanation for this is that, the network only observes hard negatives in the in-batch retrieval problems, and thus would overfit to the small differences between similar classes, *e.g.* different bicycle models.

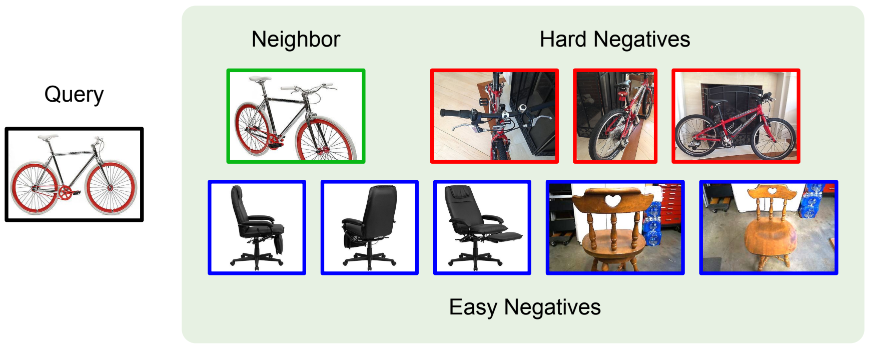


Figure 6: An example in-batch retrieval problem with batch size $M = 10$, under the hard sampling strategy. The query is from a bicycle class with $n = 2$ images. The database contains one neighbor of the query, 3 hard negatives (another bicycle), and 5 easy negatives (chairs).

Dataset	**#Category**	**Category Names**
Stanford Online Products	12	bicycle, cabinet, chair, coffee maker, fan, kettle, lamp, mug, sofa, stapler, table, toaster
In-Shop Clothes Retrieval	23	MEN/Denim, MEN/Jackets_Vests, MEN/Pants, MEN/Shirts_Polos, MEN/Shorts, MEN/Suiting, MEN/Sweaters, MEN/Sweatshirts_Hoodies, MEN/Tees_Tanks, WOMEN/Blouses_Shirts, WOMEN/Cardigans, WOMEN/Denim, WOMEN/Dresses, WOMEN/Graphic_Tees, WOMEN/Jackets_Coats, WOMEN/Leggings, WOMEN/Pants, WOMEN/Rompers_Jumpsuits, WOMEN/Shorts, WOMEN/Skirts, WOMEN/Sweaters, WOMEN/Sweatshirts_Hoodies, WOMEN/Tees_Tanks
PKU VehicleID	250 +unlabeled	vehicle models

Table 4: Category labels in all three datasets.

B. Additional Experiments

B.1. Ablation Study: Minibatch Sampling

We compare our category-based **hard** sampling strategy to the **random** strategy. Quantitatively, we can see from Table 5 that the hard strategy consistently outperforms random, across the three datasets and two architectures that we use. All FastAP results reported in earlier sections are from the hard strategy, while the random strategy is also competitive with the state-of-the-art.

FastAP	ResNet-18		ResNet-50	
R@1	Random	Hard	Random	Hard
Products	72.3	**73.2**	74.2	**75.8**
Clothes	81.7	**89.0**	84.1	**90.0**
VehicleID	79.5	**85.3**	82.8	**84.5**

Table 5: Quantitative comparison between minibatch sampling strategies for FastAP.

B.2. Convergence of Training

We also study the effect of batch size on the convergence speed of FastAP. Since our minibatch formulation is a stochastic approximation to the retrieval problem defined over the entire dataset, we expect larger batches to give better approximation and faster convergence, as is common in stochastic optimization. In Figure 7, we present learning curves on the Stanford Online Products dataset with varying batch sizes and the ResNet-18 backbone. As can be seen, there is indeed a positive correlation between batch size and convergence speed. The same observation is made on the other datasets as well. For all datasets, convergence happens within 50 epochs.

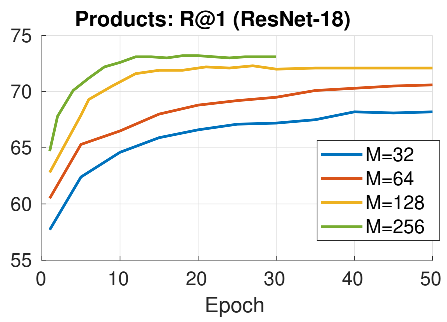


Figure 7: Learning curves of FastAP on the Products dataset, with different batch sizes. We observe that larger batches lead to faster convergence.

B.3. Results Using the GoogLeNet Architecture

In addition to the ResNet results reported in the paper, we also conduct another set of experiments with the GoogLeNet architecture [36], which is commonly used in the deep metric learning literature. Specifically, we use Inception-v1 without BatchNorm, which is known to perform similarly to ResNet-18 on ImageNet.

Results are reported in Table 6, where GoogLeNet uses the maximum allowable batch size on a 12GB GPU. As expected, GoogLeNet results are close to ResNet-18. More importantly, they are state-of-the-art on the Clothes and VehicleID benchmarks, and on-par with more complex models (*e.g.* BIER) on the Products dataset. Interestingly, GoogLeNet clearly outperforms both ResNet-18 and ResNet-50 on VehicleID, which supports our earlier hypothesis regarding the importance of batch size for this dataset. For the other datasets, it would be reasonable to expect the more complex Inception-v2 and Inception-v3 to obtain improved results that approach those of ResNet-50.

	Products				Clothes						VehicleID					
R@k	1	10	100	1000	1	10	20	30	40	50	S1	S5	M1	M5	L1	L5
GoogLeNet	72.7	86.3	93.5	97.5	88.9	97.1	97.9	98.3	98.6	98.7	91.0	96.7	89.1	95.5	85.7	94.2
ResNet-18	73.2	86.8	94.1	97.8	89.0	97.2	98.1	98.5	98.7	98.9	90.9	96.0	88.9	95.2	85.3	93.9
ResNet-50	75.8	89.1	95.4	98.5	90.0	97.5	98.3	98.5	98.7	98.9	90.4	96.5	88.0	95.4	84.5	93.9

Table 6: We additionally report GoogLeNet [36] results for experiments conducted in the paper. Batch sizes are 320, 256, and 96 for GoogLeNet, ResNet-18, and ResNet-50, respectively.

C. SGD for FastAP

We denote the neural network parameterization of our embedding as $\Psi : \mathcal{X} \to \mathbb{R}^m$. Let the induced Euclidean metric be d_Ψ. As mentioned in the paper, we L_2-normalize all embedding vectors. For $\forall x, y \in \mathcal{X}$, their squared Euclidean distance in the embedding space then becomes

$$d_\Psi(x,y)^2 = \|\Psi(x) - \Psi(y)\|^2 = \|\Psi(x)\|^2 + \|\Psi(y)\|^2 - 2\Psi(x)^\top \Psi(y) \tag{17}$$

$$= 2 - 2\Psi(x)^\top \Psi(y). \tag{18}$$

Since $\Psi(x)^\top \Psi(y) \in [-1, 1]$, the squared distance has closed range $[0, 4]$. Our derivations will work with the squared distance for convenience (to avoid a square root operation). The partial derivative of the squared distance is given by

$$\frac{\partial d_\Psi(x,y)^2}{\partial \Psi(x)} = -2\Psi(y). \tag{19}$$

We consider a minibatch setting, with a minibatch $B = \{x_1, \ldots, x_M\} \subset \mathcal{X}$. For $\forall x_i \in B$, the rest of the minibatch is partitioned into two sets according to neighborhood information: the set of x_i's neighbors, denoted B_i^+, and the set of non-neighbors B_i^-.

We define a set of M in-batch retrieval problems, where for each $i \in \{1, \dots, M\}$, x_i is used to query the database that is $B_i^+ \cup B_i^-$. Let FastAP$_i$ be the resulting FastAP value for the i-th in-batch retrieval problem. Without loss of generality, we assume B_i^+ and B_i^- are both non-empty, otherwise FastAP gets trivial values of 0 or 1. The overall objective for the minibatch, denoted FastAP$_B$, is the average of these M values.

C.1. Soft Histogram Binning

Given a query x_i, database items are sorted according to their squared distance to x_i in the embedding space. We then quantize the range $[0, 4]$ using L equally-spaced bin centers $\{c_1, c_2, \dots, c_L\}$, and produce a histogram $(h_{i,1}, \dots, h_{i,L})$ to count the number of items in each bin. This is done through defining a quantizer Q_i where

$$Q_i(x) = \arg\min_l |d_\Psi(x, x_i)^2 - c_l|, \forall x \in B \setminus \{x_i\}. \tag{20}$$

The regular histogram binning operation performs hard assignment:

$$h_{i,j} = \sum_{x \in B \setminus \{x_i\}} \mathbf{1}[Q_i(x) = j]. \tag{21}$$

Instead, we adopt the soft binning technique of [39] that relaxes the hard assignment with a piecewise-linear interpolation δ, with a width parameter Δ:

$$\hat{h}_{i,j} = \sum_{x \in B \setminus \{x_i\}} \delta(d_\Psi(x, x_i)^2, c_j), \tag{22}$$

$$\forall z \in \mathbb{R},\ \delta(z, c_j) = \begin{cases} 1 - \dfrac{|z - c_j|}{\Delta} & , \quad |z - c_j| \le \Delta \\ 0 & , \quad \text{otherwise} \end{cases} \tag{23}$$

The derivative of δ is piecewise-constant and easy to compute. To obtain the positive and negative histograms $(\hat{h}_i^+, \hat{h}_i^-)$, we restrict the sum to be over B_i^+ or B_i^-. We also choose Δ to be exactly the width of a histogram bin, *i.e.*, $\Delta = c_i - c_{i-1}$. This means that the relaxation has bounded support: for $\forall x \in B \setminus \{x_i\}$, x generally has nonzero contributions to two adjacent bins, and the contributions sum to 1. The only exception is when $d_\Psi(x, x_i)^2$ exactly coincides with the center of some bin, in which case the contribution of x to that bin is 1. Given this choice, the only variable parameter in our soft histogram binning formulation is L, or the number of histogram bins.

C.2. Minibatch Gradient Computation

Given minibatch $B = \{x_1, \dots, x_M\}$, the output of the embedding layer is an $m \times M$ matrix,

$$\Psi_B = [\Psi(x_1)\ \Psi(x_2)\ \cdots\ \Psi(x_M)], \tag{24}$$

where $\Psi(x_i) \in \mathbb{R}^m, 1 \le i \le M$. Our loss layer takes Ψ_B as input, and computes the minibatch objective FastAP$_B$. The derivative of the minibatch objective with respect to Ψ_B can be written as

$$\frac{\partial \text{FastAP}_B}{\partial \Psi_B} = \frac{1}{M} \sum_{i=1}^{M} \frac{\partial \text{FastAP}_i}{\partial \Psi_B} \tag{25}$$

$$= \frac{1}{M} \sum_{i=1}^{M} \sum_{l=1}^{L} \left(\frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,l}^+} \frac{\partial \hat{h}_{i,l}^+}{\partial \Psi_B} + \frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,l}^-} \frac{\partial \hat{h}_{i,l}^-}{\partial \Psi_B} \right). \tag{26}$$

Evaluating (26) is challenging as the M in-batch retrieval problems are inter-dependent on each other. However, there exists a solution to a similar problem. Specifically, [13] shows that when the metric is the Hamming distance and Ψ is the relaxed output of a hash mapping, (26) can be evaluated as

$$-\frac{1}{2} \frac{\Psi_B}{M} \sum_{l=1}^{L} \left(A_l^+ B_l^+ + B_l^+ A_l^+ + A_l^- B_l^- + B_l^- A_l^- \right), \tag{27}$$

where $A_l^+, A_l^-, B_l^+, B_l^-$ are $M \times M$ matrices:

$$A_l^+ = \text{diag}\left(\frac{\partial \mathcal{O}_1}{\partial \hat{h}_{1,l}^+}, \dots, \frac{\partial \mathcal{O}_M}{\partial \hat{h}_{M,l}^+}\right), \tag{28}$$

$$A_l^- = \text{diag}\left(\frac{\partial \mathcal{O}_1}{\partial \hat{h}_{1,l}^-}, \dots, \frac{\partial \mathcal{O}_M}{\partial \hat{h}_{M,l}^-}\right), \tag{29}$$

$$B_l^+ = \left[\mathbf{1}[x_j \in B_i^+] \left.\frac{\partial \delta(z, c_l)}{\partial z}\right|_{z=d_\Psi(x_j,x_i)^2}\right]_{ij}, \tag{30}$$

$$B_l^- = \left[\mathbf{1}[x_j \in B_i^-] \left.\frac{\partial \delta(z, c_l)}{\partial z}\right|_{z=d_\Psi(x_j,x_i)^2}\right]_{ij}. \tag{31}$$

Here, $\mathcal{O}$ denotes the objective, which could be any differentiable function computed on the relaxed histograms. Note that the scaling $-\frac{1}{2}$ in (27) is due to how the b-bit Hamming distance d_H is continuously relaxed [13]:

$$d_H(x, y) = \frac{b - \Psi(x)^\top \Psi(y)}{2} \quad \Rightarrow \quad \frac{\partial d_H(x, y)}{\partial \Psi(x)} = -\frac{1}{2}\Psi(y). \tag{32}$$

For our purposes, this framework can be reused with two modifications: replace $\mathcal{O}$ with FastAP, and change the underlying metric to Euclidean. We omit detailed derivations as they share much in common with [13], and state the result directly: for FastAP, (26) is evaluated as

$$-2\frac{\Psi_B}{M}\sum_{l=1}^{L}\left(F_l^+ B_l^+ + B_l^+ F_l^+ + F_l^- B_l^- + B_l^- F_l^-\right). \tag{33}$$

Note that the scaling is changed to -2 due to the use of Euclidean distance (19). F_l^+ and F_l^- are defined analogously as in (28) and (29), but with FastAP as the objective $\mathcal{O}$. We next detail how to compute them.

C.3. Differentiating FastAP

We focus on computing F_l^+ due to symmetry:

$$F_l^+ = \text{diag}\left(\frac{\partial \text{FastAP}_1}{\partial \hat{h}_{1,l}^+}, \dots, \frac{\partial \text{FastAP}_M}{\partial \hat{h}_{M,l}^+}\right). \tag{34}$$

Note that each entry in F_l^+ is derived from a different in-batch retrieval problem. Instead of directly computing it, we shall first compute a full $L \times M$ matrix F^+:

$$F^+ = \begin{bmatrix} \frac{\partial \text{FastAP}_1}{\partial \hat{h}_{1,1}^+} & \cdots & \frac{\partial \text{FastAP}_M}{\partial \hat{h}_{M,1}^+} \\ \vdots & \ddots & \vdots \\ \frac{\partial \text{FastAP}_1}{\partial \hat{h}_{1,L}^+} & \cdots & \frac{\partial \text{FastAP}_M}{\partial \hat{h}_{M,L}^+} \end{bmatrix} \in \mathbb{R}^{L \times M}, \tag{35}$$

and then, F_l^+ can be constructed from its l-th row. We will compute F^+ column-wise, which is the natural order, as each column is generated by the same in-batch retrieval problem.

The i-th column of F^+ is an L-vector:

$$\left(\frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,1}^+}, \frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,2}^+}, \dots, \frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,L}^+}\right). \tag{36}$$

To compute it, the first important observation is that FastAP decomposes over the histogram bins. This is seen from the definition of FastAP:

$$\text{FastAP}_i = \frac{1}{N_i^+}\sum_{j=1}^{L}\frac{\hat{H}_{i,j}^+\hat{h}_{i,j}^+}{\hat{H}_{i,j}} \triangleq \frac{1}{N_i^+}\sum_{j=1}^{L}\text{FastAP}_{i,j}. \tag{37}$$

Here, $N_i = |B_i^+|$. It is also easy to see that $\text{FastAP}_{i,j}$ only depends on the j-th histogram bin and earlier bins. Therefore we have the following property:

$$\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}_{i,l}^+} = 0, \quad \forall j < l. \tag{38}$$

Combining these observations, we can decompose the computation of (36) as

$$\left(\frac{\partial \sum_{j\geq 1}\text{FastAP}_{i,j}}{\partial \hat{h}_{i,1}^+}, \frac{\partial \sum_{j\geq 2}\text{FastAP}_{i,j}}{\partial \hat{h}_{i,2}^+}, \dots, \frac{\partial \sum_{j\geq L}\text{FastAP}_{i,j}}{\partial \hat{h}_{i,L}^+}\right) \tag{39}$$

$$= \left(\frac{\partial \text{FastAP}_{i,1}}{\partial \hat{h}_{i,1}^+}, \frac{\partial \text{FastAP}_{i,2}}{\partial \hat{h}_{i,2}^+}, \dots, \frac{\partial \text{FastAP}_{i,L}}{\partial \hat{h}_{i,L}^+}\right) + \left(\sum_{j>1}\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}_{i,1}^+}, \sum_{j>2}\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}_{i,2}^+}, \dots, 0\right). \tag{40}$$

The next important observation is that when $j > l$, the partial derivative $\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}_{i,l}^+}$ is *independent* of l. This allows us to efficiently evaluate the partial sums in the second part of (40). We verify this property below:

$$\left.\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}_{i,l}^+}\right|_{j>l} = \frac{1}{N_i^+}\frac{\partial}{\partial \hat{h}_{i,l}^+}\left(\frac{\hat{H}_{i,j}^+\hat{h}_{i,j}^+}{\hat{H}_{i,j}}\right) = \frac{1}{N_i^+}\frac{\hat{h}_{i,j}^+\hat{H}_{i,j} - \hat{H}_{i,j}^+\hat{h}_{i,j}^+}{\hat{H}_{i,j}^2} \tag{41}$$

$$= \frac{1}{N_i^+}\frac{\hat{H}_{i,j}^-\hat{h}_{i,j}^+}{\hat{H}_{i,j}^2}, \tag{42}$$

which is a function of j but not l. We denote this partial derivative as $\hat{f}_{i,j}^+$.

If we define two more shorthands:

$$\hat{g}_i^+ = \left(\frac{\partial \text{FastAP}_{i,1}}{\partial \hat{h}_{i,1}^+}, \frac{\partial \text{FastAP}_{i,2}}{\partial \hat{h}_{i,2}^+}, \dots, \frac{\partial \text{FastAP}_{i,L}}{\partial \hat{h}_{i,L}^+}\right) \in \mathbb{R}^L, \tag{43}$$

$$\hat{f}_i^+ = \left(\hat{f}_{i,1}^+, \hat{f}_{i,2}^+, \dots, \hat{f}_{i,L}^+\right) \in \mathbb{R}^L, \tag{44}$$

then (40) is simply computed as

$$\hat{g}_i^+ + U\hat{f}_i^+ \tag{45}$$

where U is an $L \times L$ upper-triangular matrix of 1's with zero diagonal, *i.e.*, $U_{ij} = \mathbf{1}[i < j]$.

By now we have computed the i-th column of F^+. To compute the whole matrix, we extend (45) to matrix form:

$$F^+ = \left[\hat{g}_1^+ \cdots \hat{g}_M^+\right] + U\left[\hat{f}_1^+ \cdots \hat{f}_M^+\right]. \tag{46}$$

And finally, F_l^+ (34) is formed by extracting the l-th row from F^+. We finally note that the time complexity for computing (26) is $O(LM^2)$.