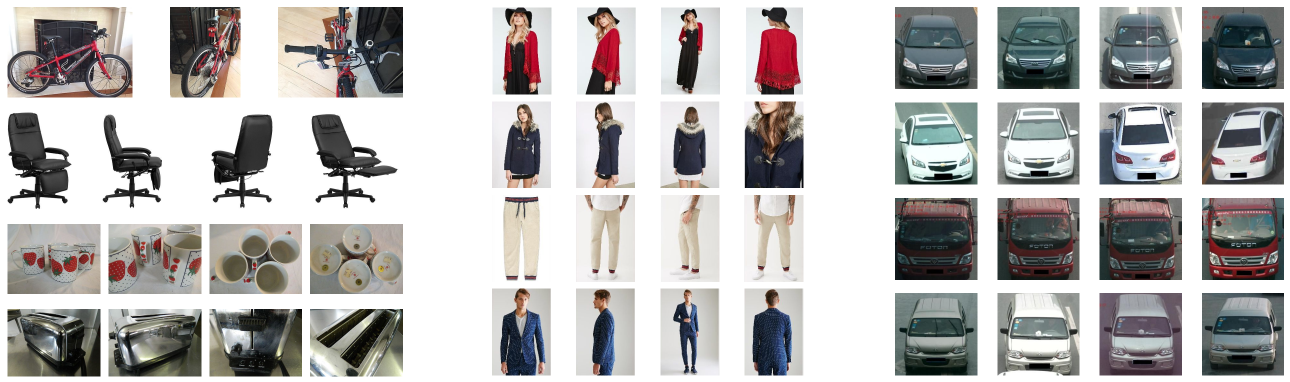
Deep Metric Learning to Rank
Supplementary Material

A. Minibatch Sampling



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

Figure 1: Examples from the three datasets used in the paper. Each row of images 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 1. 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 2, 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 1.

- **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.

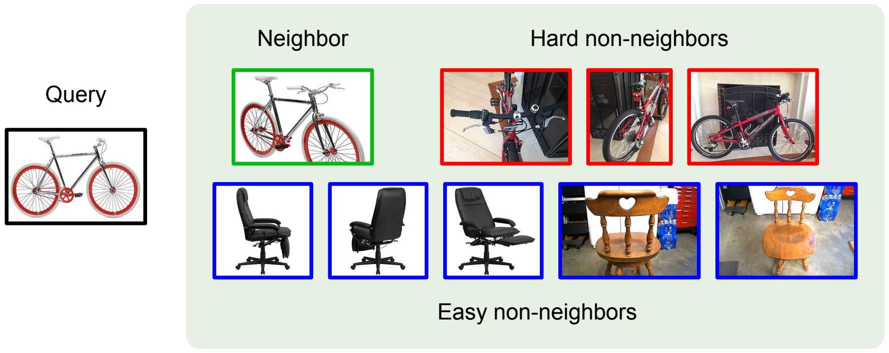


Figure 2: An example in-batch retrieval problem with batch size $M = 10$, under the hard sampling strategy. The query is from a bicycle class with $n = 2$ images. The database contains one neighbor of the query, 3 hard negatives (another bicycle), and 5 easy negatives (chairs).

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 1: Category labels in all three datasets.

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.

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 2 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 2: 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 3, 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.

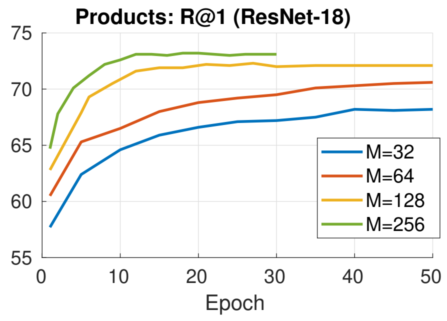


Figure 3: Learning curves of FastAP on the Products dataset, with different batch sizes. We observe that larger batches lead to faster convergence.

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 [1], 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 3, 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 3: We additionally report GoogLeNet [1] 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

$$d_\Psi(x,y)^2 = \|\Psi(x) - \Psi(y)\|^2 = \|\Psi(x)\|^2 + \|\Psi(y)\|^2 - 2\Psi(x)^\top \Psi(y) \tag{1}$$

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

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

$$\frac{\partial d_\Psi(x, y)^2}{\partial \Psi(x)} = -2\Psi(y). \tag{3}$$

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 $i \in \{1, \ldots, M\}$, 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 $\{c_1, c_2, \ldots, c_L\}$, and produce a histogram $(h_{i,1}, \ldots, h_{i,L})$ to count the number of items in each bin. This is done through defining a quantizer Q_i where

$$Q_i(x) = \arg\min_l |d_\Psi(x, x_i)^2 - c_l|, \forall x \in B \setminus \{x_i\}. \tag{4}$$

The regular histogram binning operation performs hard assignment:

$$h_{i,j} = \sum_{x \in B \setminus \{x_i\}} \mathbf{1}[Q_i(x) = j]. \tag{5}$$

Instead, we adopt the soft binning technique of [2] that relaxes the hard assignment with a piecewise-linear interpolation δ, with a width parameter Δ:

$$\hat{h}_{i,j} = \sum_{x \in B \setminus \{x_i\}} \delta(d_\Psi(x, x_i)^2, c_j), \tag{6}$$

$$\forall z \in \mathbb{R},\ \delta(z, c_j) = \begin{cases} 1 - \dfrac{|z - c_j|}{\Delta} & , \quad |z - c_j| \le \Delta \\ 0 & , \quad \text{otherwise} \end{cases} \tag{7}$$

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

$$\Psi_B = [\Psi(x_1)\ \Psi(x_2)\ \cdots\ \Psi(x_M)], \tag{8}$$

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

$$\begin{aligned} \frac{\partial \text{FastAP}_B}{\partial \Psi_B} &= \frac{1}{M} \sum_{i=1}^M \frac{\partial \text{FastAP}_i}{\partial \Psi_B} & (9) \\ &= \frac{1}{M} \sum_{i=1}^M \sum_{l=1}^L \left(\frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,l}^+} \frac{\partial \hat{h}_{i,l}^+}{\partial \Psi_B} + \frac{\partial \text{FastAP}_i}{\partial \hat{h}_{i,l}^-} \frac{\partial \hat{h}_{i,l}^-}{\partial \Psi_B} \right). & (10) \end{aligned}$$

Evaluating (10) 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, [3] shows that when the metric is the Hamming distance and Ψ is the relaxed output of a hash mapping, (10) can be evaluated as

$$-\frac{1}{2}\frac{\Psi_B}{M}\sum_{l=1}^{L}\left(A_l^+B_l^+ + B_l^+A_l^+ + A_l^-B_l^- + B_l^-A_l^-\right), \tag{11}$$

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

$$A_l^+ = \operatorname{diag}\left(\frac{\partial\mathcal{O}_1}{\partial\hat{h}_{1,l}^+}, \dots, \frac{\partial\mathcal{O}_M}{\partial\hat{h}_{M,l}^+}\right), \tag{12}$$

$$A_l^- = \operatorname{diag}\left(\frac{\partial\mathcal{O}_1}{\partial\hat{h}_{1,l}^-}, \dots, \frac{\partial\mathcal{O}_M}{\partial\hat{h}_{M,l}^-}\right), \tag{13}$$

$$B_l^+ = \left[\mathbf{1}[x_j \in \mathcal{B}_i^+]\left.\frac{\partial\delta(z,c_l)}{\partial z}\right|_{z=d_\Psi(x_j,x_i)^2}\right]_{ij}, \tag{14}$$

$$B_l^- = \left[\mathbf{1}[x_j \in \mathcal{B}_i^-]\left.\frac{\partial\delta(z,c_l)}{\partial z}\right|_{z=d_\Psi(x_j,x_i)^2}\right]_{ij}. \tag{15}$$

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 (11) is due to how the b-bit Hamming distance d_H is continuously relaxed [3]:

$$d_H(x,y) = \frac{b - \Psi(x)^\top\Psi(y)}{2} \quad\Rightarrow\quad \frac{\partial d_H(x,y)}{\partial\Psi(x)} = -\frac{1}{2}\Psi(y). \tag{16}$$

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 [3], and state the result directly: for FastAP, (10) is evaluated as

$$-2\frac{\Psi_B}{M}\sum_{l=1}^{L}\left(F_l^+B_l^+ + B_l^+F_l^+ + F_l^-B_l^- + B_l^-F_l^-\right). \tag{17}$$

Note that the scaling is changed to -2 due to the use of Euclidean distance (3). F_l^+ and F_l^- are defined analogously as in (12) and (13), 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:

$$F_l^+ = \operatorname{diag}\left(\frac{\partial\text{FastAP}_1}{\partial\hat{h}_{1,l}^+}, \dots, \frac{\partial\text{FastAP}_M}{\partial\hat{h}_{M,l}^+}\right). \tag{18}$$

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^+:

$$F^+ = \begin{bmatrix} \frac{\partial\text{FastAP}_1}{\partial\hat{h}_{1,1}^+} & \cdots & \frac{\partial\text{FastAP}_M}{\partial\hat{h}_{M,1}^+} \\ \vdots & \ddots & \vdots \\ \frac{\partial\text{FastAP}_1}{\partial\hat{h}_{1,L}^+} & \cdots & \frac{\partial\text{FastAP}_M}{\partial\hat{h}_{M,L}^+} \end{bmatrix} \in \mathbb{R}^{L\times M}, \tag{19}$$

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:

$$\left(\frac{\partial \text{FastAP}_i}{\partial \hat{h}^+_{i,1}}, \frac{\partial \text{FastAP}_i}{\partial \hat{h}^+_{i,2}}, \dots, \frac{\partial \text{FastAP}_i}{\partial \hat{h}^+_{i,L}} \right). \tag{20}$$

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

$$\text{FastAP}_i = \frac{1}{N_i^+} \sum_{j=1}^{L} \frac{\hat{H}^+_{i,j} \hat{h}^+_{i,j}}{\hat{H}_{i,j}} \triangleq \frac{1}{N_i^+} \sum_{j=1}^{L} \text{FastAP}_{i,j}. \tag{21}$$

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:

$$\frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,l}} = 0, \quad \forall j < l. \tag{22}$$

Combining these observations, we can decompose the computation of (20) as

$$\left(\frac{\partial \sum_{j \geq 1} \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,1}}, \frac{\partial \sum_{j \geq 2} \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,2}}, \dots, \frac{\partial \sum_{j \geq L} \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,L}} \right) \tag{23}$$

$$= \left(\frac{\partial \text{FastAP}_{i,1}}{\partial \hat{h}^+_{i,1}}, \frac{\partial \text{FastAP}_{i,2}}{\partial \hat{h}^+_{i,2}}, \dots, \frac{\partial \text{FastAP}_{i,L}}{\partial \hat{h}^+_{i,L}} \right) + \left(\sum_{j>1} \frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,1}}, \sum_{j>2} \frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,2}}, \dots, 0 \right). \tag{24}$$

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 (24). We verify this property below:

$$\left. \frac{\partial \text{FastAP}_{i,j}}{\partial \hat{h}^+_{i,l}} \right|_{j>l} = \frac{1}{N_i^+} \frac{\partial}{\partial \hat{h}^+_{i,l}} \left(\frac{\hat{H}^+_{i,j} \hat{h}^+_{i,j}}{\hat{H}_{i,j}} \right) = \frac{1}{N_i^+} \frac{\hat{h}^+_{i,j} \hat{H}_{i,j} - \hat{H}^+_{i,j} \hat{h}^+_{i,j}}{\hat{H}^2_{i,j}} \tag{25}$$

$$= \frac{1}{N_i^+} \frac{\hat{H}^-_{i,j} \hat{h}^+_{i,j}}{\hat{H}^2_{i,j}}, \tag{26}$$

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:

$$\hat{g}_i^+ = \left(\frac{\partial \text{FastAP}_{i,1}}{\partial \hat{h}^+_{i,1}}, \frac{\partial \text{FastAP}_{i,2}}{\partial \hat{h}^+_{i,2}}, \dots, \frac{\partial \text{FastAP}_{i,L}}{\partial \hat{h}^+_{i,L}} \right) \in \mathbb{R}^L, \tag{27}$$

$$\hat{f}_i^+ = \left(\hat{f}^+_{i,1}, \hat{f}^+_{i,2}, \dots, \hat{f}^+_{i,L} \right) \in \mathbb{R}^L, \tag{28}$$

then (24) is simply computed as

$$\hat{g}_i^+ + U \hat{f}_i^+ \tag{29}$$

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 (29) to matrix form:

$$F^+ = \left[\hat{g}_1^+ \cdots \hat{g}_M^+ \right] + U \left[\hat{f}_1^+ \cdots \hat{f}_M^+ \right]. \tag{30}$$

And finally, F_l^+ (18) is formed by extracting the l-th row from F^+. We finally note that the time complexity for computing (10) is $O(LM^2)$.

References

[1] 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.

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

[3] 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.