Supplemental Material for 'Group Testing for Accurate and Efficient Range-Based Near Neighbor Search for Plagiarism Detection'

Harsh Shah[1], Kashish Mittal[1], and Ajit Rajwade[1]

Department of Computer Science and Engineering, Indian Institute of Technology Bombay

1 Contents

This supplemental material contains the following:

1. A figure showing the negative logarithm of the normalized histograms of dot products between feature vectors of query images and those of all gallery images from each of the four datasets we experimented with in the main paper. These are shown in Fig. 2. Compare with Fig. 2 of the main paper where normalized histograms are shown.
2. A comparison of the techniques Our-Sum and Our-Max proposed in this paper to recent group testing based NN search methods such as [10, 14, 20] is presented in Sec. 2.
3. A discussion regarding the limitation of using the recent advances in the compressed sensing literature to the NN search problem is presented in Sec. 3.
4. Statistics for the average number of points pruned away in each round of binary splitting in our method on each of the four databases, are presented in Fig. 3 for Our-Sum and Fig. 4 for Our-Max. Notice that for both the proposed algorithms (Our-Sum and Our-Max) there is nearly no pruning encountered during the initial rounds. But as the adaptive tests continue, negative tests are encountered in further rounds, and hence pools get pruned away. Surprisingly, most negative tests for Our-Max are encountered in the last round in contrast to that for Our-Sum.
5. Our numerical experiments, illustrated in Fig. 5 reveal that $P(\boldsymbol{q}^t\boldsymbol{y} \leq \rho)$ increases with λ and decreases as L increases. This tallies with intuition, and supports the hypothesis that a larger number of pools will be pruned away in a given round if there is more rapid decrease in the probability of dot product values from 0 to 1.
6. A theoretical analysis of the Our-Max algorithm is presented in Sec. 4.
7. The hyper-parameters for various competing algorithms are presented in Sec. 5.
8. Details of augmentation to images for generating queries in a streaming setting are presented in Sec. 6 and Fig. 7.
9. A comparison of softmax features (on top of VGGNet features) versus VGGNet features in terms of retrieval precision and retrieval recall is presented in Table 1 and Sec. 7.

10. Experiments with a low similarity value, i.e. ρ (=0.3), and experiments on a dataset with 12-million points are presented in Sec. 8.

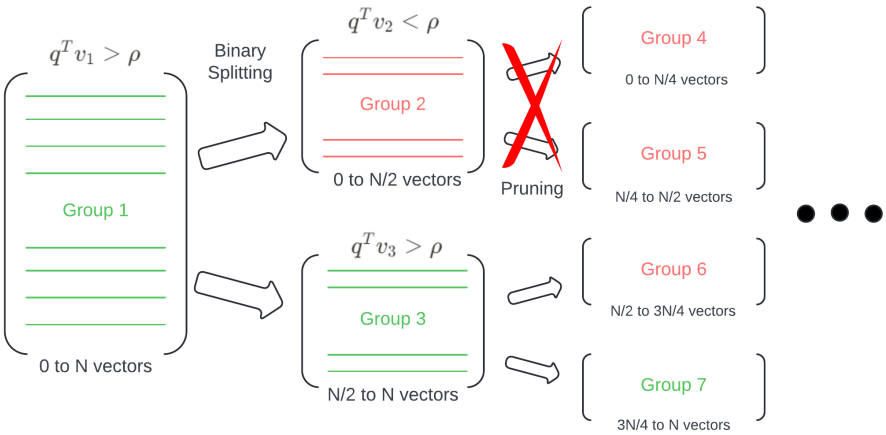

Fig. 1: Schematic of our GT algorithm for NN search

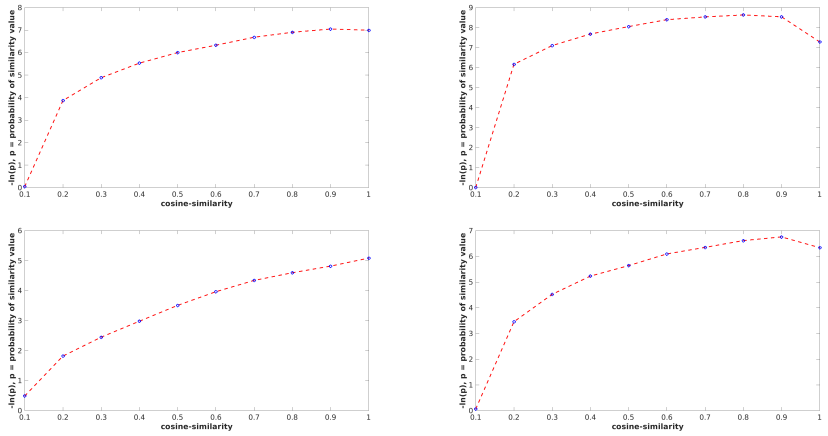

Fig. 2: Negative logarithm of normalized histograms of dot products between feature vectors of query images (10K in number) and those of all gallery images of MIRFLICKR, ImageNet, IMDB-Wiki and InstaCities (left to right, top to bottom). Compare to Fig. 2 of the main paper.

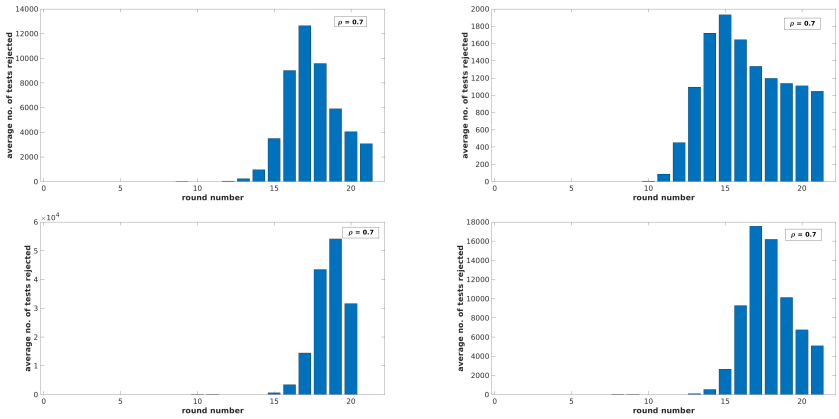

Fig. 3: Histograms of the number of pools pruned in every round of binary splitting for the Our-Sum method, for MIRFLICKR, ImageNet, IMDB-Wiki and InstaCities (left to right, top to bottom), all for $\rho \geq 0.7$.

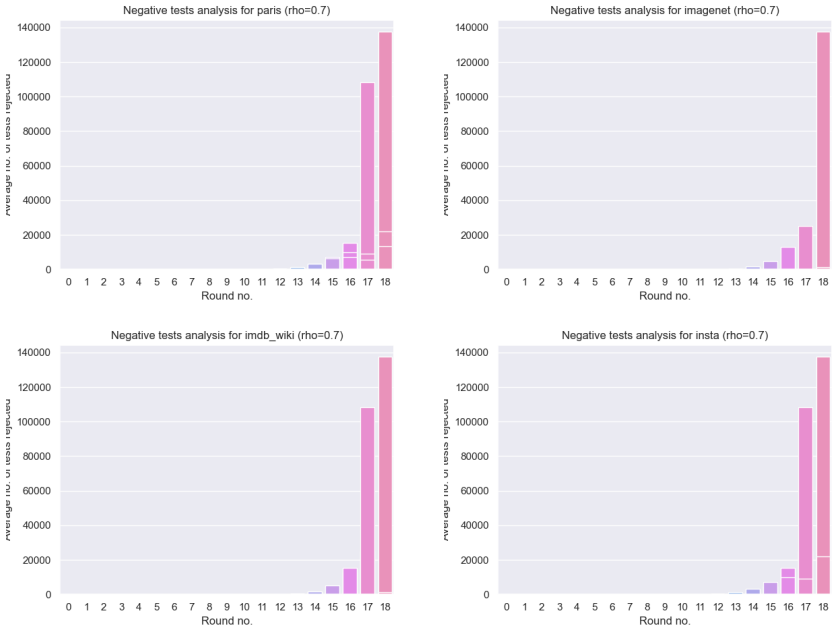

Fig. 4: Histograms of the number of pools pruned in every round of binary splitting for the Our-Max technique for MIRFLICKR, ImageNet, IMDB-Wiki and InstaCities (left to right, top to bottom), all for $\rho \geq 0.7$.

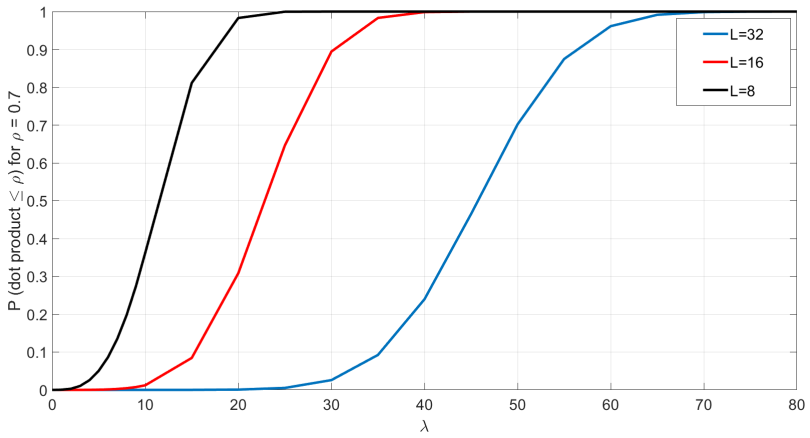

Fig. 5: The probability that the dot product of a query vector $\boldsymbol{q}$ with a pool created from L participating vectors, falls below $\rho = 0.7$, as a function of λ for $L \in \{8, 16, 32\}$.

2 Comparison to other Group Testing Methods for NN Search

The oldest work on GT for NN search was presented in [20]. In this method, the original data vectors are represented as a matrix $\boldsymbol{F} \in \mathbb{R}^{N \times d}$. The pools are represented as a matrix $\boldsymbol{Y} \in \mathbb{R}^{m \times d}$ obtained by pre-multiplying $\boldsymbol{F}$ by a randomly generated balanced binary matrix $\boldsymbol{A}$ of size $m \times N$ where $m < N$. Given a query vector $\boldsymbol{q}$, its similarity with the jth pool is computed, yielding a score $v_{yj} = \boldsymbol{q}^t \boldsymbol{Y^j}$, where $\boldsymbol{Y^j}$ is the jth group vector (and the jth row of $\boldsymbol{Y}$). This is repeated for every $j \in [m]$. Let $\mathcal{P}_i$ be the set of pools in which the ith vector, i.e. $\boldsymbol{f_i}$, belongs. Then, a likelihood score $L_i \triangleq \sum_{j \in \mathcal{P}_i} v_{yj}$ is computed for every vector. The vector from $\boldsymbol{F}$ with the largest likelihood score (denote this vector by $\boldsymbol{f_{k0}}$) is considered to be one of the nearest neighbors of $\boldsymbol{q}$. Next, the value $\boldsymbol{q}^t \boldsymbol{f_{k0}}$ is computed and the pool-level similarity scores for all pools containing $\boldsymbol{f_{k0}}$ are updated to yield new scores of the form $v_{yj} = v_{yj} - \boldsymbol{q}^t \boldsymbol{f_{k0}}$. Thereafter, all likelihood scores are updated. This process is called back-propagation in [20]. Again, the vector from $\boldsymbol{F}$ with the highest likelihood is identified and this procedure is repeated some R times, and finally a sort operation is performed in order to rank them. In [20], it is argued that the time complexity of this procedure is $O(d(m + R))$. However, this is an approximate search method, and there is no guarantee that the R neighbors thus derived will indeed be the nearest ones. As a result, a large number of false positives may be generated by this method.

The work in [14] clusters the collection $\mathcal{D}$ into disjoint groups such that the members of a group are as orthogonal to each other as possible. As a result, the similarity measure $\boldsymbol{q}^t \boldsymbol{y_j}$ between $\boldsymbol{q}$ and a group vector $\boldsymbol{y_j}$ is almost completely dominated by the similarity between $\boldsymbol{q}$ and exactly one member of the group. A careful sparse coding and decoder correction step is also used, and the sparsity

of the codes plays a key role in the speedup achieved by this method. However, this method is again an approximate method and heavily relies on the near-orthogonality of each group.

The work in [10] randomly distributes the collection $\mathcal{D}$ of N vectors over some B cells. This is independently repeated R times, creating a grid of $B \times R$ cells. Within each cell, we have a collection of participating members $M_{r,b}$ and a corresponding group test $C_{r,b}$. The group test is essentially a binary classifier which determines whether $M_{r,b}$ contains at least one member which is similar to the query vector $\boldsymbol{q}$. This is an approximate query, with a true positive rate p and false positive rate q. It is implemented via a distance-sensitive bloom filter [9,15], which allows for very fast querying. The bloom filter is constructed with m binary arrays, with some L concatenated hash functions used in each array. For all positive tests $C_{r,b}$, a union set of all their members is computed. This is repeated R times to create R candidate sets, and the intersection of these R sets is created. Each union can contain many false positives, but this intersection filters out non-members effectively. Precise bounds on p, q are derived in [10], and the time complexity of a single query is proved to be $O(N^{1/2+\gamma} \log^3 N)$ where $\gamma \triangleq \log s_{|K|}/(\log s_{|K|+1} - \log s_{|K|})$ where $s_{|K|}$ stands for the similarity between $\boldsymbol{q}$ and the Kth most similar vector in $\mathcal{D}$. The query time is provably sub-linear for queries for which $\gamma < 1/2$. This is obeyed in data which are distributed as per a Gaussian mixture model with components that have well spread out mean vectors and with smaller variances. But for many distributions, this condition could be violated, leading to arbitrarily high query times in the worst case. This method, too, produces a large number of false negatives and false positives.

Compared to these three afore-mentioned techniques, our method is exact, with the same accuracy as exhaustive search. The methods [10, 14, 20] require hyper-parameters (R for [20], sparse coding parameters in [14], B, R, m, L in [10]) for which there is no clear data-driven selection procedure. Our method, however, requires no parameter tuning. In experiments, we have observed a speed up of more than ten-fold in querying time with our method as compared to exhaustive search on some datasets. Like [20], and unlike [10, 14], our method does have a large memory requirement, as we require all pools to be in memory. Some techniques such as [10] require the nearest neighbors to be significantly more similar than all other members of $\mathcal{D}$ (let us call this condition $\mathcal{C}1$). Our method does not have such a requirement. However, our method will perform more efficiently for queries for which a large number of similarity values turn out to be small, and only a minority are above the threshold ρ (let us call this condition $\mathcal{C}2$). In our experimental study on diverse datasets, we have observed that $\mathcal{C}2$ is true always. On the other hand, $\mathcal{C}1$ does not hold true in a large number of cases, as also reported in [10, Sec. 5.3].

3 A Discussion on using Compressed Sensing Techniques for NN Search

Sec. 3 of the main paper describes three recent papers [10, 14, 20] which apply the principles of group testing to near neighbor search, and compares them to our technique. In this section, we describe our attempts to apply the latest developments from the compressed sensing (CS) literature to this problem. Note that CS and GT are allied problems, and hence it makes sense to comment on the application of CS to near neighbor search.

Consider that the original data vectors are represented as a matrix $\boldsymbol{F} \in \mathbb{R}^{N \times d}$. The pools are represented as a matrix $\boldsymbol{Y} \in \mathbb{R}^{m \times d}$ obtained by pre-multiplying $\boldsymbol{F}$ by a randomly generated balanced binary matrix $\boldsymbol{A}$ of size $m \times N$ where $m < N$. We considered the relation $\boldsymbol{v_y} = \boldsymbol{A v_x}$, where $\boldsymbol{v_y} \in \mathbb{R}^m$ contains the dot products of the query vector $\boldsymbol{q}$ with each of the pool vectors in $\boldsymbol{Y} \in \mathbb{R}^{m \times d}$, i.e. $\boldsymbol{v_y} = \boldsymbol{Y q}$. Likewise $\boldsymbol{v_x} \in \mathbb{R}^N$ contains the dot products of the query vector $\boldsymbol{q}$ with each of the vectors in the collection $\mathcal{D}$, i.e. $\boldsymbol{v_x} = \boldsymbol{F q}$. The aim is to efficiently recover the largest elements of $\boldsymbol{v_x}$ from $\boldsymbol{A}, \boldsymbol{v_y}$. Since algorithms such as Lasso [13] are of an iterative nature, we considered efficient non-iterative algorithms from the recent literature for recovery of nearly sparse vectors [21, Alg. 8.3]. These are called expander recovery algorithms. Theorem 8.4 of [21] guarantees recovery of the k largest elements of $\boldsymbol{v_x}$ using this algorithm, but the bounds on the recovery error are too high to be useful, i.e. $\|\boldsymbol{v_x^{(k)}} - \boldsymbol{v_{x,est}}\|_\infty \leq \delta \triangleq \|\boldsymbol{v_x} - \boldsymbol{v_x^{(k)}}\|_1$. Here, the vector $\boldsymbol{v_x^{(k)}}$ is constructed as follows: the k largest elements of $\boldsymbol{v_x}$ are copied into $\boldsymbol{v_x^{(k)}}$ at the corresponding indices, and all other elements of $\boldsymbol{v_x^{(k)}}$ are set to 0. Clearly, for most situations, δ will be too large to be useful. Indeed, in some of our numerical simulations, we observed δ to be several times larger than the sum of the k largest elements of $\boldsymbol{v_x}$, due to which Alg. 8.3 from [21] is rendered ineffective for our application. Moreover, for Theorem 8.4 of [21] to hold true, the matrix $\boldsymbol{A}$ requires each item in $\mathcal{D}$ to participate in a very large number of pools, rendering the procedure inefficient for our application. Given these restrictive conditions, we did not continue with this approach.

4 Theoretical Analysis for Our-Max

In Sec. 4 of the main paper, we have presented a theoretical analysis of the expected number of dot product computations required for Our-Sum, which is the binary splitting procedure using summations for pool creation. A similar analysis for estimating the number of dot products required for Our-Max using solely the distribution assumption on dot products (truncated normalized exponential or TNE) is challenging. Consider a pool vector $\boldsymbol{y} = \sum_{i=1}^{K} \boldsymbol{f_i}$ created by the summation of K participating vectors $\{\boldsymbol{f_i}\}_{i=1}^{K}$. In Our-Sum, the dot product between the query vector $\boldsymbol{q}$ and the pool vector $\boldsymbol{y}$ is exactly equal to the sum of the dot products between $\boldsymbol{q}$ and individual members of the pool. Hence an analysis via the central limit theorem or the truncated Erlang distribution is possible, as explained in Sec. 5 of the main paper. For Our-Max, the pool vector $\boldsymbol{y}$ is

constructed in the following manner: $\forall j \in \{1, 2, ..., d\}, y_j = \max_{i \in \{1,2,...,K\}} f_{i,j}$ where $f_{i,j}$ stands for the jth element of vector $\boldsymbol{f_i}$. The dot product $\boldsymbol{q}^t\boldsymbol{y}$ is an *upper bound* on $\max_{i \in \{1,2,...,K\}} \boldsymbol{q}^t\boldsymbol{f_i}$ (assuming that all values in $\boldsymbol{q}$ and in each $\boldsymbol{f_i}$ are non-negative). One can of course use the TNE assumption of the individual dot products $\boldsymbol{q}^t\boldsymbol{f_i}$ and use analytical expressions for the maximum of TNE random variables. However the fact that $\boldsymbol{q}^t\boldsymbol{y}$ is an *upper bound* on $\max_{i \in \{1,2,...,K\}} \boldsymbol{q}^t\boldsymbol{f_i}$ complicates the analysis as compared to the case with Our-Sum. Hence, it is difficult to determine an expected number of dot-products per query for Our-Max, but it is possible to determine an upper bound on this quantity. This can be done as described below.

Let $X_i \triangleq \boldsymbol{q}^t\boldsymbol{f_i}$. Let D_P denote the similarity of the query vector $\boldsymbol{q}$ with the pool vector $\boldsymbol{y}$, and let P denote the set of vectors that participated in that pool. We can bound D_P with a constant c such that $D_P \leq c \cdot \max_{i \in P}(X_i)$ (see later in this section regarding the value of c). Let $n := |P|$. Then using this bound, we have the following relation:

$$p_n(\rho) = P[D_P < \rho] \geq P[c \cdot \max_{i \in P}(X_i) < \rho] \geq [F_X(\rho/c)]^n, \tag{1}$$

where the last inequality is based on order statistics, and where $F_X(.)$ denotes the CDF of the TNE. From the main paper, recall that Q_k stands for the expected number of pools in round k. Using the recursive relation for Q_k in terms of Q_{k-1}, we have the following:

$$Q_k \leq 2(1 - [F_X(\rho/c)]^{N/2^{k-2}})Q_{k-1}. \tag{2}$$

Using this, an upper bound on $E(\rho)$ can be obtained in the following manner:

$$E(\rho) \triangleq 1 + \sum_{i=2}^{\lceil \log_2 N + 1 \rceil} Q_i. \tag{3}$$

Notice there is no $\frac{1}{2}$ factor (in contrast to $E(\rho)$ for Our-Sum) here. This is because even after splitting, the similarity needs to be calculated for both the divided pools unlike in the Our-sum algorithm. The constant c can be theoretically as large as the dimension of the dataset (that is, $d = 1000$ for the datasets used here), giving a very loose upper bound. However, in order to obtain a sharper upper bound, we find the distribution of c for each pool in a given dataset and use the 90 percentile value (denoted by c_{90}) to evaluate the bound. Details for this are provided next.

Distribution of ratios $\left(c = \frac{D_P}{\max_{i \in P}(X_i)}\right)$: Any upper bound on the expected number of dot products required by Our-Max is dependent on the constant c. Although, theoretically c can be as large as the dimension of the database vectors, the distribution of c for each dataset studied and 90 percentile value of c (i.e., c_{90}) can be used to obtain the upper bounds on $E[\rho]$. Fig. 6 shows the distribution of c for the datasets used in our experiments. Here below, we mention the c_{90} values for various datasets used in evaluating the theoretical upper bound reported in Table 2 of the main paper.

- MIRFLICKR: $c_{90} \approx 10$.
- ImageNet: $c_{90} \approx 7$.
- IMDB-Wiki: $c_{90} \approx 10$.
- InstaCities: $c_{90} \approx 11$.

Clearly, these values are significantly lower than d, which facilitates better upper bounds, albeit with some probability of error.

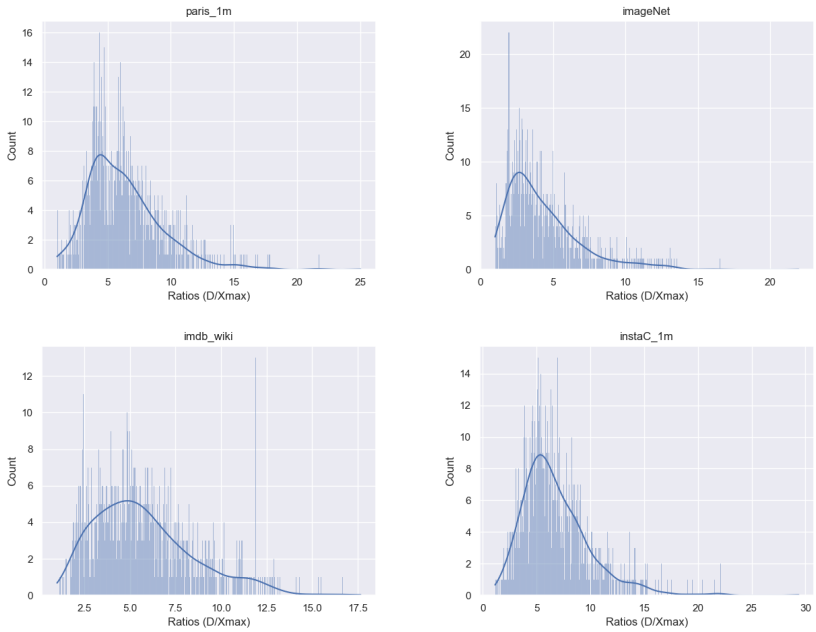

Fig. 6: Histograms showing distribution of the ratio $c = \frac{D_P}{\max_{i \in P}(X_i)}$ for pools in datasets MIRFLICKR, ImageNet, IMDB-Wiki and InstaCities (left to right, top to bottom). The histograms are computed over ~ 1000 queries, with 100 bins per histogram.

5 Hyper-parameters for the algorithms

In order to arrive at comparable values of precision and recall of other algorithms as compared to our proposed algorithms, we performed experiments with many hyper-parameters for each algorithm. Below are the hyper-parameters we used for various experiments with the competing methods. Note that in order to select the set of hyperparameters for each algorithm, we chose the one which gives high recall without significantly increasing time. So there can be other hyperparameter sets for an algorithm with higher recall but at the cost of disproportionate increase in query time. Both streaming and non-streaming setting uses the same

set of hyperparameters for each algorithm, except FLANN-R (details mentioned below).

1. FLINNG [10]: For each dataset, we ran this code with different hyper-parameters $B \in \{2^{12}, 2^{13}, \cdots, 2^{18}, 2^{19}\}$, $m \in \{2^2, 2^3, \cdots, 2^{10}\}$, $R \in \{2, 3, 4\}, L = 14$ as recommended in the suppl. mat. of [10] (see Sec. 3 of the main paper for their precise meaning) and chose the result that yielded the best recall.
2. FALCONN [7]: Number of tables = $\{100, 150, 200, 250, 300\}$, Number of probes = $\{40, 50, 70, 100\}$. Parameters used for reporting : (250, 70) respectively
3. IVF from the FAISS package [1, 8]: Number of lists = $\{32, 64, 128\}$, Number of probes = $\{2, 16\}$. Parameters used for reporting : (32, 16) respectively
4. HSNW from the FAISS package [1]: $M = 32$, Efconstruction = $\{32, 64\}$, Ef-search = $\{2\} \times$ Number of neighbors to retrieve, i.e. K. Parameters used for reporting : (32, 2) respectively
5. SCANN [2,12]: Number of leaves = $\{1, 2\} \times \sqrt{\text{Number of elements in the dataset}}$, Number of leaves to search = $\{1/2, 1/4, 1/8\} \times$ Number of leaves, Reorder number of neighbors = $\{4, 8, 16\} \times K$. Parameters used for reporting : (2, 1/4, 16) respectively
6. FLANN [6, 17]: We used `AutotunedIndexParams` with high target precision (0.85) and other parameters set to their default values. For IMDB-Wiki dataset, we used `KDTreeIndexParams(32)`, for building the index with 32 parallel kd-trees and `SearchParams(128)` during search phase. This was done because `AutotunedIndexParams` took long time to build index (more than 20 hours) on IMDB-Wiki dataset. For streaming setting, due to excess time taken by `AutotunedIndexParams` (leading to high *Index Build Time*), we used `KDTreeIndexParams(16)`, for building the index with 16 parallel kd-trees and `SearchParams(256)`.
7. FALCONN++ [3, 19]: We used the parameters suggested in examples of FALCONN++, with number of random vectors = 256, number of tables = 350, $\alpha = 0.01$, Index Probes (iProbes) = 4, Query Probes (qProbes) = $\min(40k, 2 \cdot \max(k \text{ among all queries})$. But we have different values of k for each query which can be as large as $N/10$. Hence, in order to prevent qProbes from exceeding the size of dataset, we cap it to $2 \cdot \max(k \text{ among all queries}))$.

6 Image Augmentations

In order to create queries suitable for the streaming environment for plagiarism detection, images from the database underwent specially tailored augmentations. The augmentations included artifacts that are somewhat subtle and which do not alter the image content very signficantly, but are designed to trick the detection system: Gaussian blur (with kernel size 3×3), color jitter (in hue, saturation and value), horizontal flip, downscaling of images and image rotation (by 3 degrees). Figure 7 shows an example of these augmentations.

ImageNet								
	Precision ↑				Recall ↑			
Feature↓, $\rho \rightarrow$	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
Softmax	0.7	0.73	0.76	0.81	0.74	0.70	0.66	0.60
Penultimate	0.66	0.85	0.96	0.99	0.2	0.07	0.015	0.001

ImageDBCorel								
	Precision ↑				Recall ↑			
Feature↓, $\rho \rightarrow$	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
Softmax	0.95	0.96	0.97	0.98	0.32	0.27	0.23	0.18
Penultimate	0.99	1	1	1	0.29	0.11	0.028	0.01

ImageDBCaltech								
	Precision ↑				Recall ↑			
Feature↓, $\rho \rightarrow$	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
Softmax	0.58	0.66	0.73	0.82	0.32	0.29	0.25	0.21
Penultimate	0.89	0.97	0.99	0.99	0.19	0.08	0.03	0.02

Table 1: Comparison of the retrieval accuracy (precision and recall) using cosine distance with (i) softmax features of VGGNet (1000 dimensional) and (ii) penultimate VGGNet features (4096 dimensional) for different values of cosine distance threshold $\rho \in \{0.6, 0.7, 0.8, 0.9\}$. Note the higher recall of softmax features.

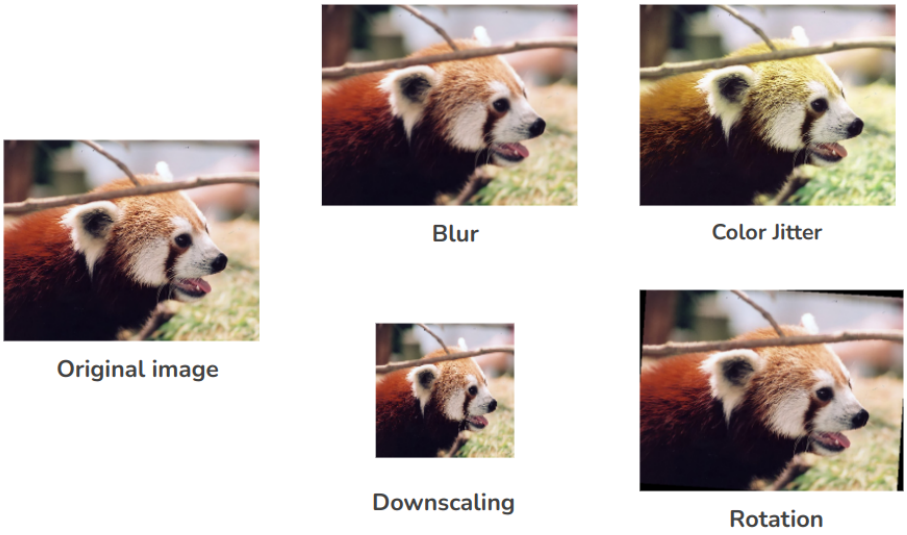

Fig. 7: Example of augmentations performed on an image from the ImageNet dataset

7 Comparison of VGG16 features

In this section, we provide a comparison between features extracted from penultimate layer of VGG16 (4096 dimensional) and the features extracted after the last layer (i.e. after applying the softmax functions leading to a 1000 dimensional feature vector as there are 1000 classes involved). For comparison, we use a setting akin to [16], in which database images which are similar to a given query image are retrieved. However, instead of retrieving a fixed number of images (k), we have performed range-based retrieval with different similarity thresholds ($\rho = \{0.6, 0.7, 0.8, 0.9\}$) on ImageNet [4], ImageDBCaltech (Caltech101) [11] and ImageDBCorel [18]. Table 1 shows the retrieval recall and retrieval precision values for the aforementioned datasets. Note that we define **retrieval precision** = # (images retrieved belonging to the same class as the query image) / # (images retrieved), and **retrieval recall** = # (images retrieved belonging to the same class as the query image) / # (images in the database having the same class as the query image). Note that the retrieval precision and retrieval recall as defined here pertain to identification of the *class labels*. These are different from the precision and recall reported in the main paper, which are based on near neighbors retrieved by a possibly approximate near neighbor search algorithm, in comparison to an exhaustive nearest neighbor search. Note that though the main algorithm proposed by our paper is fully accurate, the other algorithms we have compared to (i.e., [5, 10, 17]) perform only approximate near neighbor search. Observe from Table 1 that the recall rates for penultimate features noticeably lag behind those of softmax features, indicating that softmax features are better suited for plagiarism detection and other applications requiring high recall.

8 Other Experiments

Here we present results on two experiments with a static database: (i) One on retrieval with low similarity values, i.e. $\rho = 0.3$, and (ii) One on the complete ImageNet Database containing 12 million data-points. These results are reported in Table 2.

References

1. https://github.com/facebookresearch/faiss 9
2. https://github.com/google-research/google-research/tree/master/scann 9
3. https://github.com/NinhPham/FalconnLSF 9
4. https://www.kaggle.com/competitions/imagenet-object-localization-challenge/data 11
5. https://github.com/FALCONN-LIB/FALCONN 11
6. FLANN - fast library for approximate nearest neighbors: wbia-pyflann 4.0.4. https://pypi.org/project/wbia-pyflann/ 9

	$\rho = 0.3$				10M scale, $\rho = 0.8$
Alg./DB.	ImgN.	IMDBW.	InstaC.	MIRFL.	ImgN. 12M
IVF (FAISS)	157,0.99,1	**51**,0.91,1	97,0.97,1	98,0.98,1	1342,1,1
FALCONN	128,0.93,0.99	184,0.95,1	163,0.92,1	141, 0.89,1	1521,0.96,1
HNSW	15,0.96,0.96	3126,0.97,0.97	432,0.97,0.97	223,0.98,0.98	561,0.98,0.98
OUR-SUM	**7,1,1**	133,**1,1**	**90,1,1**	**67,1,1**	**269,1,1**

Table 2: Comparison of mean query times (ms), mean recall, mean precision (respectively) for different methods over 10K queries: (i) For $\rho = 0.3$ for different databases; and (ii) for the full ImageNet database with **12 million images** for $\rho = 0.8$. Hyperparameters for all algorithms are the same as mentioned earlier in in this supplemental material document. OUR-SUM gives better precision, recall than all methods, and generally better query times than others.

7. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: Proceedings of the forty-seventh annual ACM symposium on Theory of computing. pp. 793–801 (2015) 9
8. Babenko, A., Lempitsky, V.: The inverted multi-index. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(6), 1247–1260 (2014) 9
9. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970) 5
10. Engels, J., Coleman, B., Shrivastava, A.: Practical near neighbor search via group testing. In: Advances in Neural Information Processing Systems. vol. 34, pp. 9950–9962 (2021) 1, 5, 6, 9, 11
11. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In: 2004 Conference on Computer Vision and Pattern Recognition Workshop. pp. 178–178 (2004). https://doi.org/10.1109/CVPR.2004.383 11
12. Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., Kumar, S.: Accelerating large-scale inference with anisotropic vector quantization. In: International Conference on Machine Learning. pp. 3887–3896 (2020) 9
13. Hastie, T., Tibshirani, R., Wainwright, M.: Statistical learning with sparsity: the lasso and generalizations. CRC press (2015) 6
14. Iscen, A., Chum, O.: Local orthogonal-group testing. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 449–465 (2018) 1, 4, 5, 6
15. Kirsch, A., Mitzenmacher, M.: Distance-sensitive bloom filters. In: Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 41–50 (2006) 5
16. Maji, S., Bose, S.: Cbir using features derived by deep learning. ACM/IMS Trans. Data Sci. **2**(3) (sep 2021). https://doi.org/10.1145/3470568, https://doi.org/10.1145/3470568 11
17. Muja, M., Lowe, D.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(11), 2227–2240 (2014) 9, 11
18. Ortega-Binderberger, M.: Corel Image Features. UCI Machine Learning Repository (1999), DOI: https://doi.org/10.24432/C5K599 11

19. Pham, N., Liu, T.: Falconn++: A locality-sensitive filtering approach for approximate nearest neighbor search. In: Advances in Neural Information Processing Systems. vol. 35, pp. 31186–31198 (2022) 9
20. Shi, M., Furon, T., Jégou, H.: A group testing framework for similarity search in high-dimensional spaces. In: ACMMM. pp. 407–416 (2014) 1, 4, 5, 6
21. Vidyasagar, M.: An introduction to compressed sensing. SIAM (2019) 6