Benchmarking Different Search-by-Image Systems with Mean Average Precision

Abstract

Google's Search-by-Image uses perceptual hash algorithm (Alibaba Clouder, 2017) which has the advantage of delivering results very quickly, but trade-off with accuracy. In search for more accurate algorithms, we tried different ways to create image descriptor ("ID" an image) that is more robust to varieties in that image (e.g., Bag-of-Visual-Words, EOH, RGBH), as well as different ways to compute similarity between any 2 different descriptors (e.g., Euclidean, PCA, Cosine). After experiments these methods and their combinations, we found that EOH and RGBH without gridding image and similarity computed by Mahalanobis distance to be the most accurate based on Mean Average Precision evaluation, thus defying many pre-experiment expectations as we later lay out in the Post-experiment analysis section.

Techniques implemented

To make an image descriptor

With colour histogram (RGBH)

The foremost downside with perceptual hash algorithm is that it focuses to much of the overall structure of the image, so if there are some small texts captioned into the image, its descriptor will be different despite most contents stay the same. To fix this problem, we try to include the colour distribution of the image, thus resulting colour histogram. The algorithm which computes colour histogram work as follows: Assuming a colour image is in RGB format (hence the abbreviation)

Step 1: Normalise every pixel of RGB image from [0,255] to [0, Q-1] where Q is the number of histogram bins to encode Q different possible values per colour channel.

Step 2: Compress 3-colour-layer RGB image down to 1-layer image but not by converting RGB to grayscale to preserve the colour distribution. We do this scaling all pixels in 1 colour channel by Q^2, scaling all pixels in another colour channel by Q^1, and scaling all pixel in remaining colour channel by Q^0. Finally, we sum every colour pixel on the same position on an image together, e.g.,
$S(i,j) = scaled\ R(i,j) + scaled\ G(i,j) + scaled\ B(i,j)$

Step 3: Flatten 1-layer image we have gotten from step 2 to 1-dimensional vector

Step 4: Make histogram from the 1-dimensional vector. As we use Q bins to encode Q possible different values per colour channel, so to encode 3 colour channels we need Q^3 bins (it makes senses to think Q in term of number of possible outcomes in statistics). Finally, we normalise this histogram by its sum of all values so that the colour histogram isn't too bias against other histograms that we will combine along with colour histogram

With edge orientation histogram (EOH)

Another downside with perceptual hash algorithm is that there are other algorithms able to capture more detailed structure of an object in an image thanks to existence of edge-detection filters, one of those IDs is EOH.

The edge filter we used is Sobel, which detects edges either via horizontal filter $\partial f/\partial x$ or vertical filter $\partial f/\partial y$, so it produces 2 edge pixels $\partial(f \star h)/\partial x$ or $\partial(f \star h)/\partial y$ for every grayscale pixel (Efros, 2005). Divide $\partial(f \star h)/\partial y$ by $\partial(f \star h)/\partial x$ will get edge gradient (same idea as gradient of y = f(x) is y/x). And taking arctan of edge gradient will get edge orientation, which not only tells an object's structure but also the direction toward which its surface is facing – a crucial piece of information which perceptual hash algorithm never able to capture. Finally, making EOH is like making colour histogram, just without step 2 and the size of histogram is only Q rather than Q^3 because Q in EOH is defined as the number of angle quadrants.

Implementing EOH is matter of integrating MATLAB's imgradient function to our work, which includes 3 edge-orientation processing steps and 3 standard histogram steps.

EO processing step 1: Normalise image and convert to grayscale.

EO processing step 2: Apply gaussian filter to image, the reason why we need to apply blur are shown in slides 55 and 56 of (Athitsos, 2021).

EO processing step 3: Use MATLAB imgradient() function to calculate edge orientation directly. But as their orientation outputs ranging from (-180,180) so we need to add elementwise by 180 to get orientation outputs ranging from (0,360).

3 standard histogram step is similar to step 1, step 3, and step 4 in computing colour histogram with the only difference in Q.

Adding-on grid

Both EOH and RGBH are normalised histograms, thus small but important features will get overwhelmed by image's background (i.e., pixels of small cow on grass field will get overwhelmed by grass' pixels). To prevent this, we divide an image into smaller grids where each grid results a distinct EOH or RGBH. In this way, small features have a chance to retain high magnitude despite normalisation. Therefore, when we sum all gridded histograms together to get 1 overall histogram, the overall histogram is marginally better in emphasising small but important features and hopefully increase accuracy

Gridding algorithm capitalises on MATLAB mat2cell() function which divides 2D matrix into cells, each cell is a sub 2D matrices that aren't necessary equal sizes. However, if we use mat2cell() in conjunction with gridding that resulting equal size submatrices, then mat2cell() pose 4 implementation challenges which became the four steps for this algorithm:

Step 1: mat2cell() doesn't work if the sum of row/column of all submatrices doesn't add up back to the row/column of the original 2D matrix. Considering we divide image into square grids, a problem immediately arises because there are remainder row/column as we divide the image's row or column by grid size. So, we solve this problem by taking only the quotient or rounding up the result to the nearest integer, then create a new image matrix with trimmed off remainder row and column.

Step 2: mat2cell() demands inputs that reflect the dimensional distributions for the submatrices, whether the inputs are presented as vector or scalar. Like say we want to split matrix A[5x5] into 2 cells: 5x2 and 5x3, then we need to input mat2cell(A,5,[2 3]). Given that we are trying to split image into submatrix cell of [grid size x grid size x N colour channels], then row distribution for mat2cell() is [grid size grid size … grid size] where its size is [1 x number of grids per row]; and column distribution is [grid size grid size … grid size] where its size is [1 x number of grids per column].

Combining EOH with RGBH with grids

Note that EOH and RGBH are effectively row vectors, we can concatenate 2 vectors together to increase accuracy, which is exactly what we did in the implementation.

With Bag-of-Visual-Words (BOVW)

Before we begin to describe our implementation of BOVW, we want to state that as we refer to image descriptor, we do not mean them as SIFT descriptor. Our BOVW has 3 subcomponents Scale-Invariant-Feature-Transform (SIFT) that creates multiple descriptors per image; K-means which clusters all descriptors of image database into hopefully meaningful clusters (clusters are visual words); sparse visual-word histogram. As Prof John Collomose had done hardest works in building SIFT() and KMeans() function, the challenges remained are integrating SIFT() to KMeans() and creating a sparse frequency vector from the output KMeans(). Solve these challenges resulting 3 implementation steps:

Step 1: Given all SIFT descriptors of the entire image, compute centroids (cluster's centre) for 23 clusters. The reason why we choose 23 clusters is because number of clusters = number of visual words = number of categories in image database. However, we also try other numbers of clusters = 72, 230, 128 to guess optimal point in elbow plot.

Step 2: Find the closest cluster for each descriptor in the image database by computing distances of each descriptor to all centroids, get the minimum distance. But getting min distance alone does not suffice for the computer to know which cluster does the minimum distance belongs to, so Prof. John Collomose used Boolean expression to normalise the matrix that contains distances of all descriptors to all centroids. In this Boolean matrix, per each descriptor(column), value 1 occurs at the row (cluster) where the descriptor has min distance and 0 everywhere else. So, Prof. John used a loop to index only that "row" while keeping everywhere else as 0 using multiplication. Finally, noting that 0 occurs everywhere else, Prof. John used max() to reduce the matrix that used to have row = number of clusters down to row vector.

Step 3: Creating sparse visual-word histogram. Now each SIFT descriptor has its own visual word, but each image has hundreds of SIFT descriptors, so we extract hundreds of visual words from the row-vector that were originally belonged to that image. Next, we create a histogram where the number of bins Q = number of clusters. Finally, there are 2 options to normalise this histogram: either we normalise it by summation like EOH and RGBH; or normalise by tf – idf. Same idea as image gridding, tf – idf is meant to give more bias weight to visual words that aren't common in the BOVW because we assume that important features tend not to be very common, tf – idf is defined as:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t = tf_{t,d} \times \log\left(\frac{N}{df_t}\right)$$

Where:

- $tf_{t,d}$ is the frequency of visual word t in image d
- N is the total number of images
- df_t is the number of images containing visual word t.
- $\log\left(\frac{N}{df_t}\right)$ is a number that evaluates the level of common of visual word t in the image database. As df_t in numerator, this number is high if visual word t is common and low if visual word t is not.

To compute descriptor similarity

With Euclidean distance

Since image descriptor is a row vector, computing similarity any 2 descriptors is like computing distance between 2 coordinate-vectors in Euclidean space: take vector A – vector B, squared elementwise $(A-B)^2$, sum all the squared elements, then square-root the sum.

With Mahalanobis distance & PCA

To calculate Mahalanobis distance, first we apply PCA to all image descriptors in the feature space to find the eigenvectors-eigenvalues of all descriptors. Because there are insignificant eigenvalues-eigenvectors that are nearly null space, we trim off those pairs and reproject the feature space to fit in the trimmed-off eigenvectors-eigenvalues. As this process of reprojection thought as multiplying all-image-descriptors matrix by U^T where U is the trimmed-off eigenvectors when we finally calculate Mahalanobis distance, we only need divide elementwise $u_i^T(A-B)^2$ by the trimmed-off eigenvalues V to get $Mdist = \sqrt{u_i^T(A-B)^2\ (\sigma_i^2)^{-1}}$

With Cosine distance

The idea of cosine distance was intuitively inspired to us by one of Prof. Patrick Winston's lectures (Winston, 2014). One thing to note in our implementation of cosine distance is that cosine distance gets larger as image gets more similar, so we need to sort all distances in descending order instead of ascending order for other distancing methods.

Evaluating Accuracy

The statistical method we choose to evaluate search's accuracy of each system is and PR curve, which is computed by sweeping through entire image database that is already sorted in ascending order of distance (descending order for Cosine distance) and calculate Precision $P(i)$ and Recall $R(i)$ where:

- Precision $P(i)$ = number of relevant images found so far / i number of images we have swept so far. The numerator is a counter for the loop
- Recall $R(i)$ = number of relevant images found so far / total number of relevant images in the same category as query. For the numerator, we need sweep the image dataset with another loop just to count the number of images in same category before we can compute P and R.

Because each PR curve is unique to 1 Search-by-Image system and 1 query image, so for the same Visual-Search system there are as many PR-curve variations as the size of the database. To generalise all variations, we need to somehow summarise each PR-curve to a single value. Fortunately, such single value exists and is called Average Precision (AP), it is defined as:

$$AP = \frac{\sum_{i=1}^{database} P(i) \times rel(i)}{N\ images\ in\ same\ category}$$

Where $rel(i)$ is a Boolean checker that outputs 1(TRUE) if image i is relevant and 0(FALSE) if otherwise. The reason why $rel(i)$ is here is because we want to only sum $P(i)$ exclusively whenever there's a new relevant image first found as those "TRUE" $P(i)$ are normalised (sum up to 1) and thus not affected by the number of relevant images with respect to the size of the database. For our implementation, all $rel(i)$ are contained in 2nd row and all $P(i)$ are contained in 1st row of matrix P so that we can easily keep track our data in the same variable for debugging purpose.

With AP, we can take an extra step to summarise all APs for all query image per Visual-Search, the variable that summarises all APs will be called Mean Average Precision(MAP), and is defined as:

$$MAP = \frac{\sum_{que}^{database} \sum_{i} AP(i)}{database}$$

Experiments

Verifying PR, AP

Before showing experiment results, we should demonstrate that our evaluation methods (PR and AP) work as intended where:

- Precision = 1 at beginning but become very low toward the end
- Recall is very low at beginning but recall = 1 at the end
- Better Visual-Search system for a given query image the closer its PR curve is to the ideal point (1,1) – green dot figure 1.

Figure 1, 2, 3 show how evaluation methods "responded" to different Search-by-Image systems. To verify the evaluation methods, we double-check it by looking at the similarity of each candidate image in the above panel to the query image at the top left: If PR-curve and AP suggest that the system is good/bad, but top panel shows otherwise then our evaluation methods is busted.

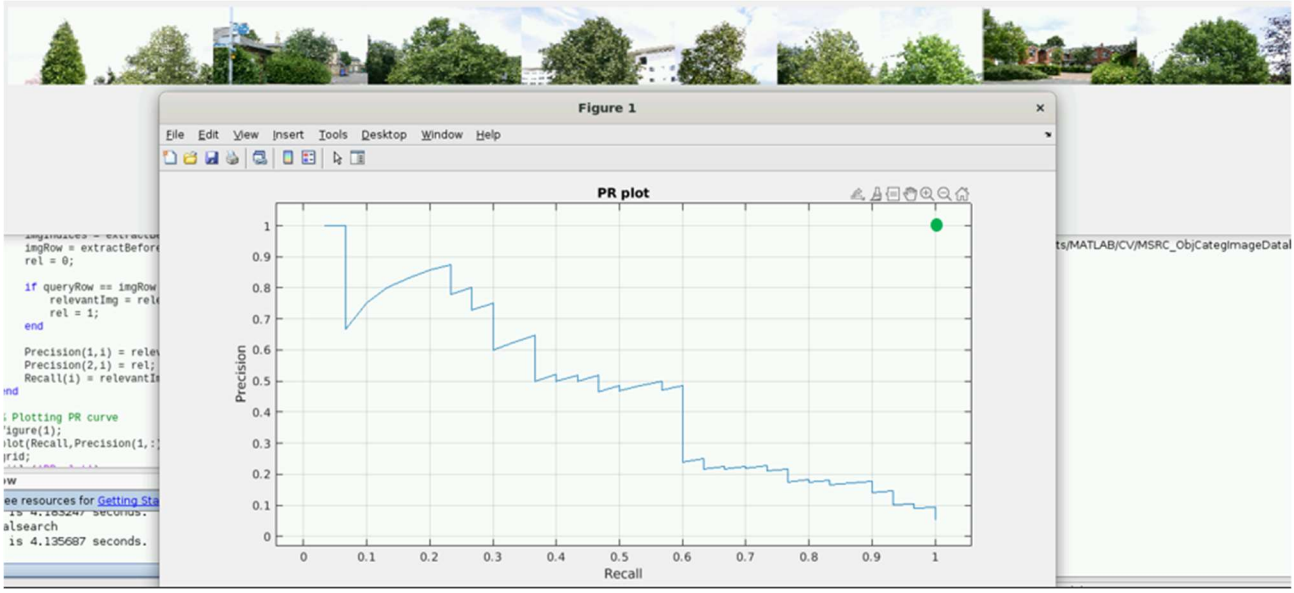


Figure 1: Query image index = 364, Search-by-Image system = EOH + RBH + Grid + Euclidean distance, AP = 0.4883.

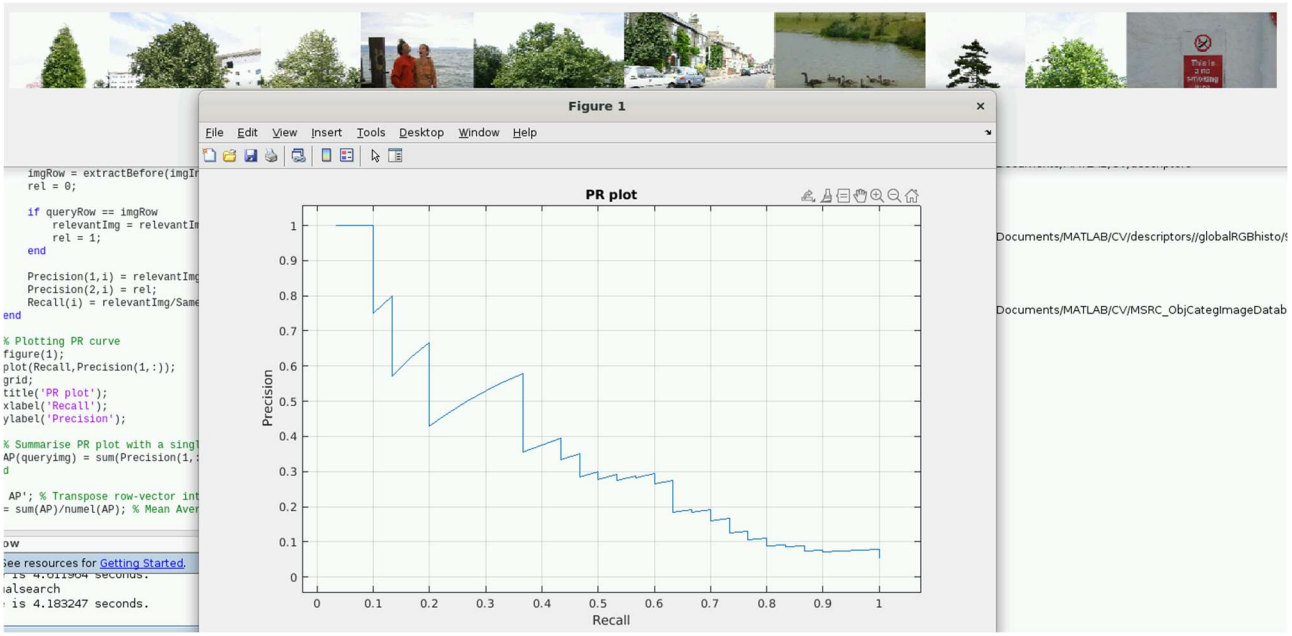


Figure 2: Query image index = 364, Search-by-Image system = EOH + RBH + Grid + Cosine distance, AP = 0.3856. Comparing to the search results in figure 1, figure 2 has more irrelevant candidate images so AP should be less, and PR curve should be further away from ideal point (1,1) than the PR curve in figure 1.

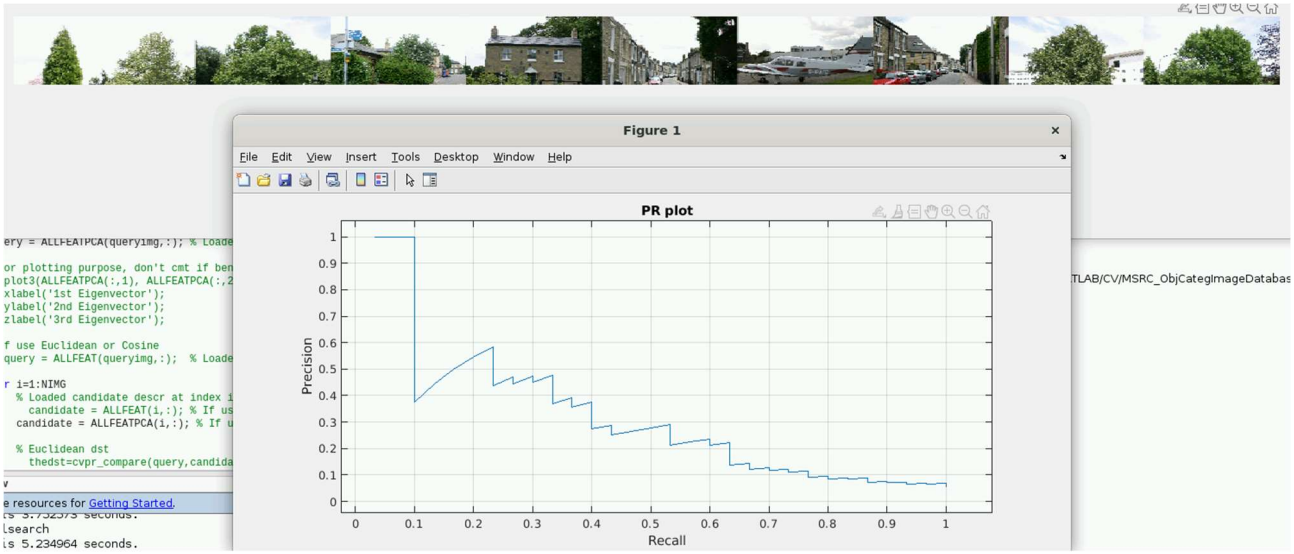


Figure 3: Query image index = 364, Search-by-Image system = EOH + RBH + Grid + Mahalanobis distance, AP = 0.3374. For similar reasons as figure 2, AP should be even less, and PR-curve should be even further away from ideal point (1,1) than the PR-curves in figure 1 and figure 2.

From figure 1, 2, 3 and many more testing on our sides, we confidently conclude that our PR and AP are working as they should. This means we can also trust in the creditability of MAP as MAP is essentially just the average value of many $AP(i)$.

MAP results for basic systems

For this report, we consider Search-by-Image system that exclusively used EOH, RGBH, or Grid are basic systems due to relative ease in implementation in comparison to BOVW. Table 1, 3 below illustrate MAP results for different basic systems when all distances were calculated under L2 norm (squared of sum of squared elements). As Cosine distances were not affected by L1, L2 norm, we put their results on a separate column. Since MAP results of Mahalanobis distance varied depend

how many eigenvectors were kept when we deflated eigenspace, only optimal MAP Mahalanobis results were shown while table 2, 4 are created to show how we derived such optimal results.

RGBH + Euclidean	RGBH + (optimal) Mahalanobis			RGBH + Cosine
0.1947	0.1871	MAP		0.1528
16.6093	16.5918	Runtime(s)		17.3926
EOH + Euclidean	EOH + (optimal) Mahalanobis			EOH + Cosine
0.1746	0.1706	MAP		0.1459
15.8557	16.0734	Runtime(s)		16.3566
RGBH +EOH + Euclidean	RGBH + EOH + (optimal) Mahalanobis			RGBH + EOH + Cosine
0.2064	0.2218	MAP		0.1730
17.1804	17.5344	Runtime(s)		16.7429
			L2 Norm	
RGBH + Grid + Euclidean	RGBH + Grid + (optimal) Mahalanobis			RGBH + Grid + Cosine
0.1779	0.1805	MAP		0.1190
18.5083	19.0966	Runtime(s)		18.6320
EOH + Grid + Euclidean	EOH + Grid + (optimal) Mahalanobis			EOH + Grid + Cosine
0.1971	0.1809	MAP		0.1233
45.9795	45.2371	Runtime(s)		45.8345
RGBH + EOH + Grid + Euclidean	RGBH + EOH + Grid + (optimal) Mahalanobis			RGBH + EOH + Grid + Cosine
0.2021	0.2091	MAP		0.1352
47.5096	48.0113	Runtime(s)		47.7049

Table 1: MAP results for different Search-by-Image systems when all distances were calculated under L2 norm (squared of sum of squared elements) with Cosine put on a separate column as it was not affected by L1 or L2 norm.

RGBH + Mahalanobis			RGBH + EOH + Grid + Mahalanobis			RGBH + Grid + Mahalanobis			
Runtime	Accuracy	%Eigenvectors kept	Runtime	Accuracy	%Eigenvectors kept	Runtime	Accuracy	%Eigenvectors kept	
7.1702	0.1674	All	6.2254	0.1605	90%	7.0115	0.1663	63%	
7.3278	0.1697	75%	6.4098	0.1620	85%	6.0632	0.1764	48%	
7.0514	0.1774	50%	7.1752	0.1666	77%	6.1143	0.1805	37%	
6.7785	0.1849	40%	6.3317	0.1755	60%	6.0407	0.1774	30%	
6.4185	0.1859	39%	6.3595	0.1839	50%	6.0514	0.1775	25%	
6.2330	0.1871	37%	6.3105	0.1927	40%				
6.1934	0.1871	36%	6.4595	0.1971	30%	EOH + Grid + Mahalanobis			
7.3742	0.1827	35%	7.2466	0.2008	20%	Runtime	Accuracy	%Eigenvectors kept	
7.0493	0.1836	30%	7.5463	0.2023	12%	5.7042	0.1764	ALL	
			6.3834	0.2091	11%	6.9920	0.1784	90%	
EOH + Mahalanobis			6.1305	0.2091	10%	5.6446	0.1809	85%	
Runtime	Accuracy	%Eigenvectors kept	6.4377	0.2003	9%	5.5824	0.1809	77%	L2 Norm
5.6039	0.1504	ALL	6.2706	0.1729	7%	6.1768	0.1611	48%	
5.5197	0.1519	53%	6.2016	0.1278	5%	5.6563	0.1662	37%	
5.7471	0.1602	43%				5.6877	0.1662	30%	
6.2330	0.1706	37%	RGH + EOH + Malahanobis						
5.7059	0.1706	36%	Runtime	Accuracy	%Eigenvectors kept				
5.6336	0.1706	30%	5.9909	0.1879	40%				
5.6924	0.1706	25%	6.1761	0.1972	30%				
			6.5255	0.2101	20%				
			6.0756	0.2165	17%				
			5.9925	0.2218	15%				
			5.9253	0.2194	13%				
			6.4090	0.2109	10%				
			6.3827	0.1595	5%				

Table 2: MAP results for different Search-by-Image systems under different number(as %) of kept Eigenvectors. Optimal % of kept Eigenvectors are highlighted in the same colour codes as table 1.

RGBH + Euclidean	RGBH + (optimal) Mahalanobis			RGBH + Cosine
0.0618	0.0747	MAP		0.1528
16.2840	16.5869	Runtime(s)		17.3926
EOH + Euclidean	EOH + (optimal) Mahalanobis			EOH + Cosine
0.0684	0.0823	MAP		0.1459
16.3134	15.5385	Runtime(s)		16.3566
RGBH + EOH + Euclidean	RGBH + EOH + (optimal) Mahalanobis			RGBH + EOH + Cosine
0.0816	0.0805	MAP		0.1730
17.2927	17.5599	Runtime(s)	L1 Norm	16.7429
RGBH + Grid + Euclidean	RGBH + Grid + (optimal) Mahalanobis			RGBH + Grid + Cosine
0.1047	0.1057	MAP		0.1190
18.3819	19.6301	Runtime(s)		18.6320
EOH + Grid + Euclidean	EOH + Grid + (optimal) Mahalanobis			EOH + Grid + Cosine
0.1019	0.1170	MAP		0.1233
45.4060	45.9058	Runtime(s)		45.8345
RGBH + EOH + Grid + Euclidean	RGBH + EOH + Grid + (optimal) Mahalanobis			RGBH + EOH + Grid + Cosine
0.1031	0.1244	MAP		0.1352
47.1471	48.6705	Runtime(s)		47.7049

Table 3: MAP results for different Search-by-Image systems when all distances were calculated under L1 norm (sum of elements) with Cosine put on a separate column. Under L1 Norm, accuracy was significantly hampered with insufficient trade-off in speed.

RGBH + Mahalanobis			RGBH + EOH + Grid + Mahalanobis			RGBH + Grid + Mahalanobis			
Runtime	Accuracy	%Eigenvectors kept	Runtime	Accuracy	%Eigenvectors kept	Runtime	Accuracy	%Eigenvectors kept	
5.9122	0.0703	75%	6.2679	0.1054	ALL	6.7611	0.1033	77%	
5.9305	0.0721	55%	6.9038	0.1050	80%	6.6645	0.1044	63%	
6.1885	**0.0747**	**50%**	6.6719	0.1059	70%	6.1539	0.1037	48%	
6.3090	0.0714	40%	6.6132	0.1038	50%	6.1024	0.1024	37%	
5.9560	0.0700	36%	6.4539	0.1053	30%	6.2115	0.1033	35%	
5.8696	0.0714	35%	6.8223	0.1238	10%	**6.6478**	**0.1057**	**30%**	
5.9941	0.0690	30%	**6.7897**	**0.1244**	**7%**	6.0669	0.1047	25%	
			6.0960	0.1107	5%				**L1 Norm**
EOH + Mahalanobis									
Runtime	Accuracy	%Eigenvectors kept	RGBH + EOH + Mahalanobis			EOH + Grid + Mahalanobis			
5.5890	0.0655	ALL	Runtime	Accuracy	%Eigenvectors kept	Runtime	Accuracy	%Eigenvectors kept	
5.5306	0.0675	53%	5.8547	0.0722	50%	5.8319	0.1089	90%	
5.5306	0.0704	43%	6.0338	0.0717	30%	5.7594	0.1169	85%	
5.5096	0.0733	30%	5.9152	0.0721	15%	5.8033	0.1169	77%	
6.1763	0.0733	25%	5.9041	0.0781	13%	**6.2510**	**0.1170**	**70%**	
5.0987	**0.0823**	**15%**	**6.0179**	**0.0805**	**10%**	5.7937	0.1092	50%	
5.9792	0.0675	5%	5.9097	0.0803	5%	5.8043	0.1164	37%	

Table 4: MAP results for different Search-by-Image systems under different number(as %) of kept Eigenvectors. Optimal % of kept Eigenvectors are highlighted in the same colour codes as table 2.

MAP results for BOVW

Euclidean distance (L2 Norm)			
Nclusters	**normalised by tf-idf**	**normalised by sum**	
23	0.1316	0.1165	
60	0.1316	0.1569	
65	0.1316	0.1553	**MAP**
69	0.1316	0.1811	
100	0.1316	0.1313	
230	0.1316	0.1786	

Mahalanobis distance (L2 Norm)			
Nclusters	**normalised by tf-idf**	**normalised by sum**	
23	0.1316	0.1316	
60	0.1316	0.1615	
65	0.1316	0.1316	**MAP**
69	0.1316	0.1711	
100	0.1316	0.1316	
230	0.1316	0.1316	

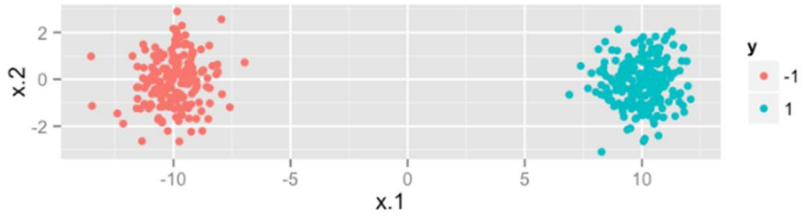
Table 5: MAP results for different BOVW systems where the results the choice for number of clusters and %Eigenvectors kept were already optimised

Post-experiment analysis

There are 5 observations we find from the experiments

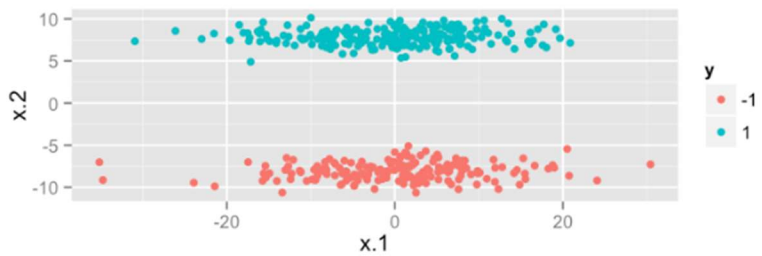
(1) Most optimal Mahalanobis MAP results occurred when the %Eigenvectors kept < 100. Before the experiment, we had expected that trimming off eigenvectors-eigenvalues would inevitably worsen results as less information = less accuracy. Yet the experiment proved otherwise because fundamentally because PCA alone can sometimes hurt accuracy like below:

PCA Helps



The direction of maximal variance is horizontal, and the classes are separated horizontally.

PCA Hurts



The direction of maximal variance is horizontal, but the classes are separated vertically

(vqv, 2013)

So, when a certain %Kept Eigenvectors boosted accuracy, it might have been because they accidentally trimming off bad eigenvectors-eigenvalues pair that hamper accuracy.

(2) L1 Norm did not trade-off significant accuracy loss with significant speed gain. We suspect this might be because the way MATLAB compute exponential polynomial have been optimised the same way how Fast Fourier Transform algorithm was implemented, therefore computation complexity only decreases by $O(NlogN)$ where N is the exponential order or LN - so L1 would be $O(1log1)$ and L2 would be $O(2log2)$

(3) Sometimes without using grid yielded high MAP than using grid. Notably in table 1, we saw RGBH + EOH + Euclidean (or Mahalanobis) yielded higher MAP than RGBH + EOH + Grid + Euclidean (or Mahalanobis). We think the reason for this phenomenon maybe because as gridding does not normalise small but important features within an image, it also does not normalise background features either. So, in this case, small but important features will get even more overwhelmed by background features than not using grid.

(4) SIFT did not increase accuracy as we had previously expected from lessons learnt in lectures despite our best attempt to use optimal number of clusters and %Eigenvectors kept. Again, we think might be because the BOVW feature dataset given to Euclidean or Mahalanobis was incompatible with those 2 distancing methods.

(5) Cosine made surprisingly bad results as we had expected it should had been second in term of MAP to Mahalanobis. Again, we think the reason for this phenomenon is closely resemblance to the reason why PCA sometimes hurt accuracy. If Cosine distancing method expects dataset to be ellipsoid shape in 45 degrees angle, whereas in reality dataset is a horizontal ellipsoid, then the method would misclassify potential candidate images

Conclusions

In conclusion, this report has proven to us every Visual-Search methods should be treated with careful consideration and we should never blindly judge that PCA nor BOVW will automatically bring better results just because their methods look more sophisticated than the classical Euclidean, EOH, RGBH, Cosine.