Machine Learning for Computer Vision CW1

Tung-Duong Mai (20180745)
School of Computing
KAIST
Daejeon, South Korea
john_mai_2605@kaist.ac.kr

Tien Dat Nguyen (20190883)
School of Computing
KAIST
Daejeon, South Korea
kaistdat123@gmail.com

***Abstract*—Coursework 1 on the computationally efficient PCA, incremental PCA, PCA-LDA (fisherface) for face recognition, and Randomised Decision Forests**

I. Computationally Efficient Eigenfaces

A. Mean face

Fig. 1a–1c shows some example faces in the dataset. The mean face (average of all images in the train set) is shown in Fig. 1d. We can observe that this face looks blurred; however, it still has general characteristics and components of a normal face such as eye, nose, hair, oval-shape face …

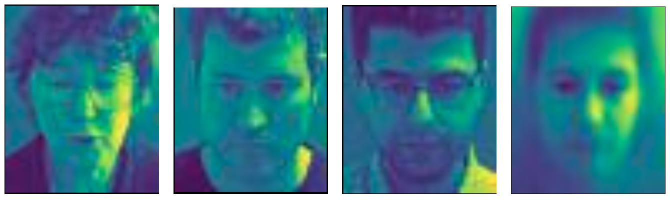

(a) Example 1 (b) Example 28 (c) Example 48 (d) Mean face

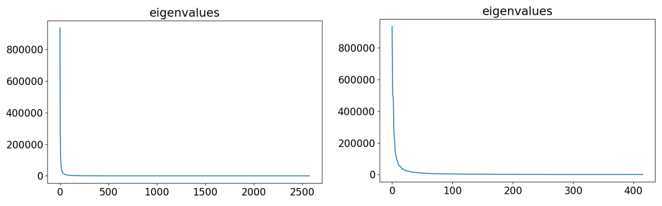
Fig. 1. (a), (b), (c) Some example faces in the dataset. (d) The mean face

B. Eigenvectors and eigenvalues of covariance matrix

In this problem, we have N = 416 data points of dimension D = 2576.

1) Direct computation: Covariance matrix $S = \frac{1}{N}AA^T$ has 2576 eigenvalues (magnitude shown in Fig. 2a), which is reasonable since S is a 2576×2576 matrix. Let $a_0, a_1, ..., a_{2575}$ be the eigenvalues in non-increasing order of their magnitude. Regard the eigenvalues with magnitude below 10^{-6} as 0. From a_{415} onward, the magnitude of the eigenvalues falls below this threshold ($|a_{414}| \approx 77.86$ and $|a_{415}| \approx 1.11 \times 10^{-10}$). S has 415 nonzero eigenvalues $a_0, a_1, .., a_{414}$.

2) Low-dimensional computation: In this approach, $S_1 = \frac{1}{N}A^T A$ is used for eigenvalues computation. S_1 has 416 eigenvalues (shown in Fig. 2b), which is reasonable since S is a 416×416 matrix. Considering the same threshold, S_t has 415 nonzero eigenvalues identical with 415 largest eigenvalues of S $a_0, a_1, .., a_{414}$.



(a) Direct computation (b) Low-dimensional computation

Fig. 2. Graph for magnitude of eigenvalues

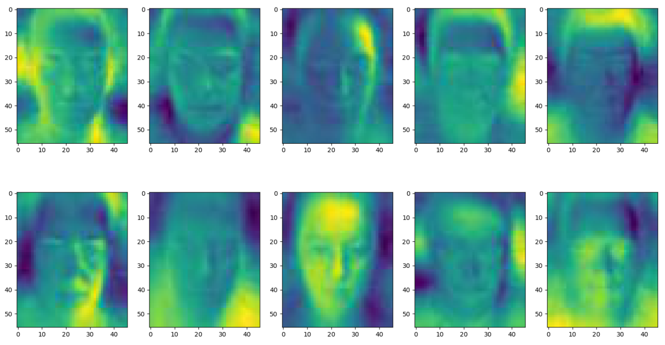
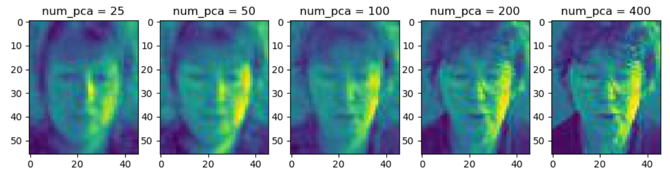


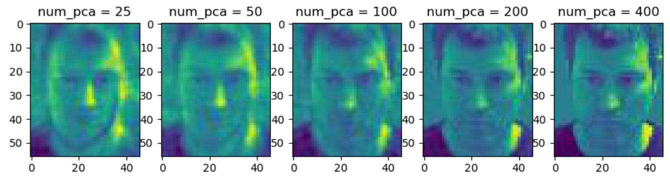
Fig. 3. Eigenfaces corresponding to 10 largest eigenvalues: 936389, 507218, 478365, 267222, 215490, 148960, 125026, 110179, 90207, 83676

3) Behavior of the eigenvalues and eigenvectors: The magnitude of the eigenvalues drops quickly, where a few first eigenvalues dominates all the remaining (for example, $|a_0|/|a_{10}| \approx 14161$ and $|a_0|/|a_{50}| \approx 103858$). This explains the very steep graph that we observed. For a more informative graph, refer to the graph of first 20 largest eigenvalues (Appendix. A). To visualize eigenvectors, the corresponding eigenfaces of top 10 eigenvalues is shown in Fig. 3

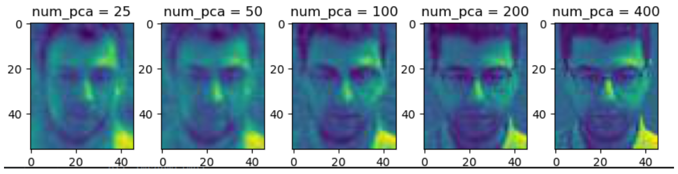
4) Result comparison between 2 methods: As discussed, all 415 nonzero eigenvalues from the low-dimensional computation method (using S_1) is identical to 415 largest eigenvalues of S. The corresponding eigenvectors u and v, although different, is closely related by $u = Av$ where u, v are from direct and low-dimensional computation, respectively. Since typically $N << D$, this meaningful result allows us to perform computation on a much smaller matrix of size $N \times N$ (instead of $D \times D$)



(a) Image 1



(b) Image 28



(c) Image 48

Fig. 4. Reconstructed face with different number of PCA bases

5) Pros and Cons of each method: Low-dimensional computation saves computational complexity. The method completes in 0.13s on our machine, 150 times faster than the direct computation (20.04s). However, if N is so small that it cannot cover all the non-zero eigenvalues, this method suffers from information loss. We also need an extra step to get the eigenvectors of S.

Direct computation immediately returns the eigenvectors of S and is immune to information loss, but is too slow and inefficient.

This is a trade-off between information preservation and computation complexity. Consider that the magnitude of the eigenvalues usually drops so quickly (Sec. I-B3), the information loss is generally not serious, and the benefits of low-dimensional method generally outweigh its drawback

C. Face reconstruction

We implement face reconstruction for some faces with id 1, 28, 48 (Fig. 1a–1c). The reconstructed faces are shown in Fig. 4, with num_pca denotes the number of PCA bases.

Across the above face identities, the reconstructed faces look clearer and closer to the original face when we increase the number of PCA bases. With 400 PCA bases, the reconstructed faces are nearly identical to the real face. We explain this with the concept of information loss: with a small number of PCA bases, we can only reconstruct most general information of the face, as many distinctive features of the face is lost. Consequently, the reconstructed faces tend to be blurred. When we add more PCA bases, we retain more information of the original face, hence, the reconstructed faces looks more similar to the original face.

II. PCA-LDA for Face Recognition

TABLE I
Recognition accuracy

M_{pca}	M_{lda}	**Recognition accuracy (%)**	M_{pca}	M_{lda}	**Recognition accuracy (%)**
364	2	12.5	100	35	83.65
364	10	40.38	100	45	81.73
364	20	67.31	100	51	81.73
364	35	75.96	70	2	29.81
364	45	77.88	70	10	78.85
364	51	78.85	**70**	**20**	**88.46**
200	2	13.46	70	35	84.62
200	10	76.92	70	45	78.85
200	20	81.73	70	51	76.92
200	35	83.65	55	2	26.92
200	45	83.65	55	10	78.85
200	51	84.61	55	20	86.54
100	2	19.23	55	35	79.81
100	10	82.69	55	45	75.0
100	20	83.65	55	51	74.04

A. Recognition accurracies by varying the M_{pca} and M_{lda}

We experiment with several values of M_{pca} and M_{lda}. The best recognition accuracy is 88.46%, when $M_{pca} = 70$ and $M_{lda} = 20$. Full results is shown in Table I.

$M_{pca} = 364$ generally gives the worst recognition accuracy, implying that keeping more information does not necessarily improve the performance. We suspected that the information from insignificant components are not necessary for recognition task. Instead, they are likely noise leading to overfitting, lowering recognition accuracy. Using reasonably lower M_{pca} to only keep the most important information help the model generalize better to unseen data.

Depending the choice of M_{pca}, dependency of recognition accuracy on M_{lda} would be different. With high M_{pca} (364 or 200), the recognition accuracy increase with higher M_{lda}. However, for lower M_{pca} (100, 70 or 55), the recognition accuracy seems to be maximized if $M_{lda} = 20$.

Increasing M_{lda} from 2 to 10 leads to a big improvement in the recognition accuracy, but further increase induces a much smaller change. Thus, the 10 largest eigenvectors obtained from LDA process are essential for the face recognition task.

B. Recognition accuracy of PCA and PCA-LDA

Table II-B shows the recognition accuracy of PCA and PCA-LDA (for PCA-LDA, the highest accuracy across different values of M_{lda} is shown). We observe that combining PCA and LDA can improve the recognition accuracy by about 11% - 22% compared to using PCA alone.

C. Rank of scatter matrix

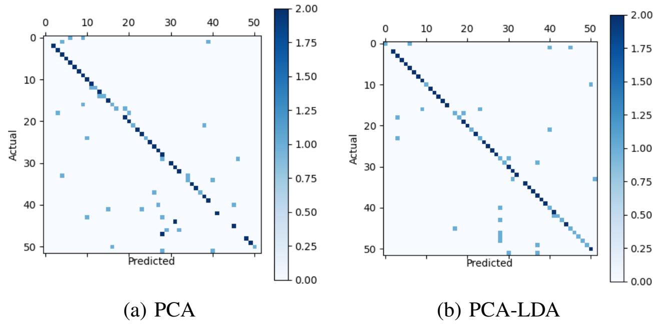
In PCA-LDA method, the rank of within-class scatter matrix S_W and between-class scatter matrix S_B only depend on M_{pca}. By implementation, we observe that :

If $M_{pca} \geq 51$, $rank(S_B) = 51, rank(S_W) = M_{pca}$

If $M_{pca} \leq 50$, $rank(S_B) = rank(S_W) = M_{pca}$

TABLE II
PCA-LDA AND PCA PERFORMANCE

M_{pca}	Recognition accuracy (%)	
	PCA	PCA-LDA
364	**67.31**	78.85
200	**67.31**	84.61
100	**67.31**	83.65
70	66.35	**88.46**
55	65.38	86.54



(a) PCA (b) PCA-LDA

Fig. 5. Confusion matrix of face recognition

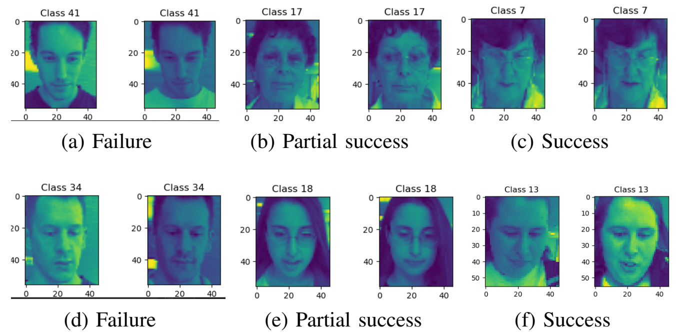


Fig. 6. Some example faces in 3 classes in the data with different recognition accuracy: (a-c) PCA (d-f) PCA-LDA

TABLE III
PERFORMANCE OF INCREMENTAL PCA,BATCH PCA AND PCA-1

-	**Incremental PCA**	**Batch PCA**	**PCA-1**
Training time (s)	0.032	0.060	0.003
Recognition accuracy (%)	66.35	66.35	63.46
Reconstruction error	682816	675444	1242676

D. Confusion matrix and examples

Confusion matrix for $M_{pca} = 70$ and $M_{lca} = 51$ is shown in Fig. 5. Result for some other values can be found in Appendix. B. The confusion matrix is highly diagonal with less off-diagonal points in PCA-LDA case. This demonstrating that LDA-PCA outperforms PCA but both perform pretty well. Some examples for successes/failures are in Fig. 6 (Partial success means 1 correct and 1 incorrect image for that class)

E. Time/memory

For PCA with $M_{pca} = 70$, training costs 0.163s and testing costs 0.403s (0.566s in total). With PCA-LDA, for $M_{pca} = 70$, and $M_{lda} = 51$, PCA training costs 0.165s, LDA training cost 0.018s and testing costs 0.395s (0.578s in total).

In both method, majority of the training time is dedicated to computing eigenvalues and eigenvectors of covariance matrix for PCA. Time for eigendecomposition of scatter matrices S_B, S_W for LDA is insignificant. Thus, executing PCA-LDA take almost as much time as PCA, but gives a much better recognition accuracy than using PCA alone.

III. INCREMENTAL PCA

We compare incremental PCA with other methods, including batch PCA and PCA trained only by the first subset (abbreviated as PCA-1 from now on), in terms of training time, construction error and face recognition accuracy. $M_{pca} = 70$ is used in experiment in this section. Full result is reported in Table. III

A. Training time

Since other computation can be accelerated by proper implementation, we only considers eigen-decomposition. Incremental-PCA (0.032s) cost significantly less training time compare to batch-PCA (0.060s). PCA-1 is by far the fastest method (0.003s).

Breaking down the performance of incremental PCA: the time to decompose the covariance matrix is roughly the same for all 4 subsets (around 0.003s) and the time to merge 4 eigenspace models is 0.020s.

Incremental PCA involves many eigen-decompositions; however, the time cost of each decomposition is much smaller compared to that of batch PCA decomposition, resulting in a total smaller training time.

Time complexity of decomposition is decided by the matrix size. In batch PCA, the matrix that we perform decomposition is of the size 416×416. The size becomes 104×104 and 141×141 for each subset in incremental-PCA and merging step, respectively (note that we use $d_1 = d_2 = M_{pca} = 70$). Let T be the time cost for batch PCA (T=0.06s). The time for each subset of incremental PCA is then expected to be 0.06T, and the merging time (3 merges) is expected to be 0.36T. In our result, these values are 0.05T and 0.33T, respectively.

This method's impact is of 2-fold. First, to perform eigen-decomposition with a dataset, we can save time by divide the dataset into smaller subsets. Second, in the online learning setting (data is received incrementally), using incremental PCA does not only reduce storage requirements, but also saves time.

B. Reconstruction error and recognition accuracy

The reconstruction error of incremental PCA and batch PCA is similar (682816 and 675444 respectively) and about half of PCA-1 (1242676). Average recognition accuracy for incremental PCA and batch PCA are identical (66.35%) and higher than that of PCA-1 (63.46%). Incremental PCA performs similarly with batch PCA and outperforms PCA-1.

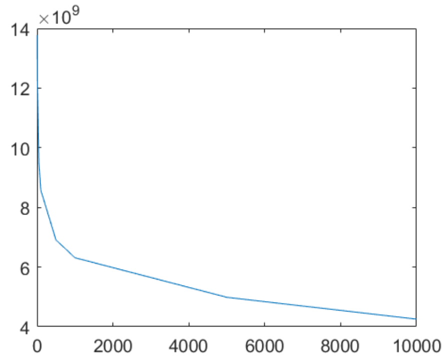


Fig. 7. Graph of inter-cluster distance versus k

PCA-1 uses only 25% of the training data, inducing a large amount of information loss. This results in a high reconstruction error and low recognition accuracy. In case of incremental PCA and batch PCA, all available information is used. Even though incremental PCA divides the data into 4 subsets, their results is merged afterward. Hence, its reconstruction error and recognition accuracy is similar to batch PCA.

C. *Parameters*

M_{pca} (the number of PCA bases) is the most important parameter for all 3 methods. Apart from $M_{pca} = 70$, we also perform the same experiment on $M_{pca} = 100$ and the results is reported in Appendix C.

In all 3 method, increase M_{pca} to 100 leading to higher recognition accuracy, lower reconstruction errors and similar training times increase in all 3 methods. This is because less information is lost. Training time is almost unchanged, since most of the time is spent on the eigendecomposition step.

However, even when we change M_{pca}, the individual trend stays the same: incremental PCA performs similarly to batch PCA and outperform PCA-1.

IV. K-MEANS CODEBOOK

A. *Kmeans implementation*

Using a vocabulary size of 50, we tried kmeans on 100k SIFT descriptors with 4 Matlab built-in initialization (uniform, cluster, kmean++, sample). All other parameters are set as default. The result shows similar results among all initialization (around 9.5×10^9 total within-class distance). Therefore, the kmean++ initialization (default) is chosen.

To avoid local minima, we use 3 replications (run 3 times).

B. *Vocabulary size*

Vocabulary size is a tradeoff between capturing relevant changes in image parts and avoid capturing noises. We try with various degrees of magnitude of vocabulary size: 10, 50, 100, 500, 1000, 5000, 10000 (Matlab crashes with k = 50000). We use **Elbow method** on the graph of sum within-cluster distance versus k (Fig. 7). As k increases, each cluster has fewer elements and the distance should generally decrease. Therefore, we choose the elbow points, where this distance decline the most (so that the distance decrease is attributed to a better clustering, not more centroids). We choose k = 500 as there exists a clear elbow at k = 500.

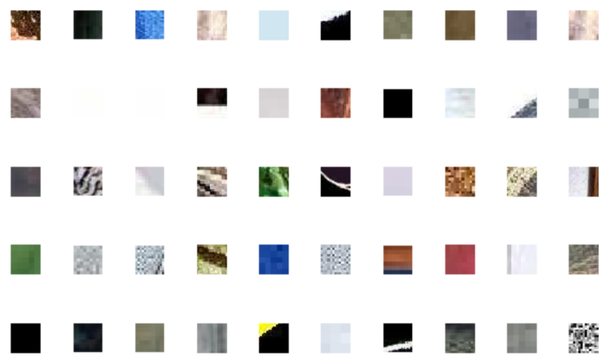


Fig. 8. First 50 codewords of the codebook

C. *Codebook*

Visualized first 50 code words in the codebook (vocabulary size 500) is shown in Fig. 8. The codewords are visualized by the axis-aligned boundary box of the patch whose descriptor lies closest to the k-mean centroids. Most of them carry some meaningful information (not just plain background)

D. *Bag-of-word histogram*

Fig. 9 shows Bag-of-word histogram and some patches on 2 examples of class 'yin_yang' in train set and test set. We observed that the patch lies on the semantically meaningful part of the image, including curvatures, dots, color contrast, ... (not the background). Histogram for 2 images are vastly similar. In the histograms, we can see that only a handful of bins have remarkable counts. Corresponding patches are semantically meaningful and the images from the same class receives similar histogram. These observations confirm that our clustering is performant. Other classes' result (2 examples for each class) is shown in Appendix. D. At a glance, we observe similar characteristic: the patch (yellow) concentrated at the semantically meaningful part (not the background), and histogram of images in the same class are similar.

E. *Quantization process*

Step 1: Get the SIFT descriptors for each image.

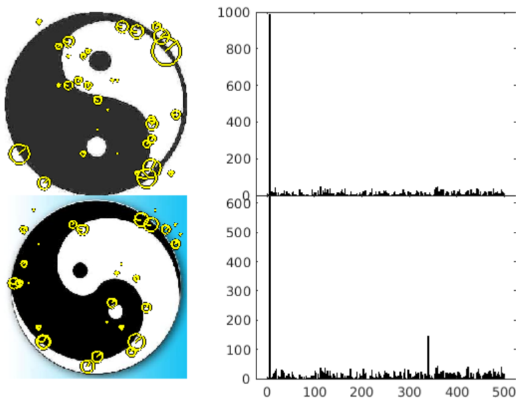
Step 2: For each descriptors, get the centroids that is closest and give a count to its corresponding codeword.

Step 3: Put all codeword's counts into a vector and use this as the image bag-of-word.

V. RANDOMIZED FOREST

A. *Data*

The Caltech101 dataset with codebook from IV is used. To have a grasp of data distribution, train and test data distribution is plotted in Fig. 10 (2-D projection by tSNE).


(a) Train set

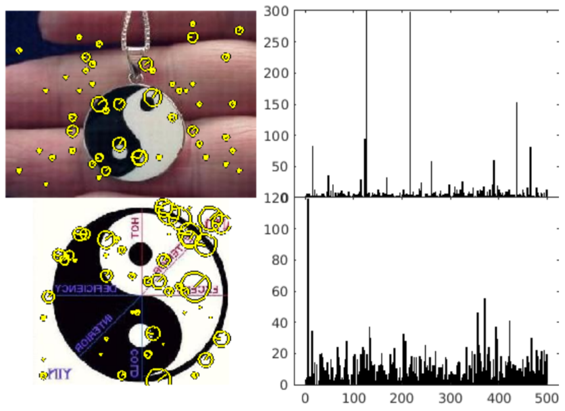
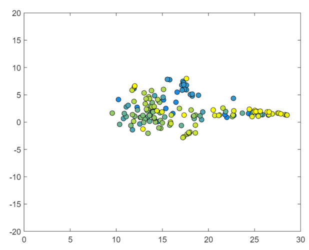

(b) Test set

Fig. 9. Example images in class yin_yang with some SIFT patches (left) and corresponding histogram (right)


(a) Train set

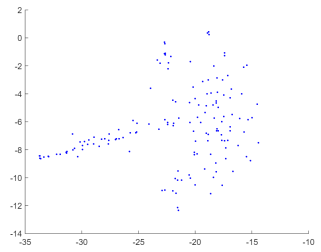
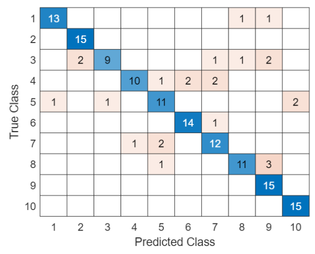
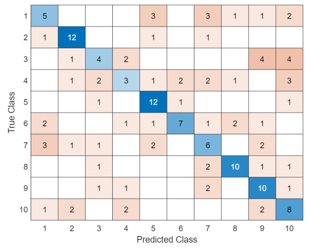

(b) Test set

Fig. 10. Data visualization for train set (colorized by class) and test set



(a) Train set (Accuracy 83%)



(b) Test set (Accuracy 51%)

Fig. 11. Accuracy and confusion matrix for train set and test set

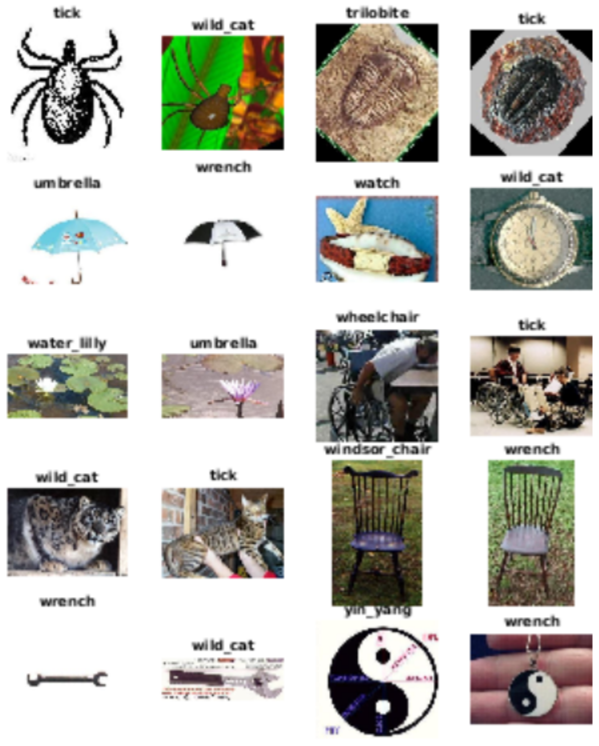


Fig. 12. Success (column 1, 3) and failure (column 2, 4) examples for each class. The predicted class by RF is annotated above each example.

B. Result on default parameters

Set of parameters in the sample code is used as default set (number of trees: 10, depth of trees: 10, degree of randomness 0.0632, Weakleaners: axis-aligned). Default vocabulary size is 500. The RF achieves 83% accuracy on train set set 51% on test set. Confusion matrix is shown in Fig. 11

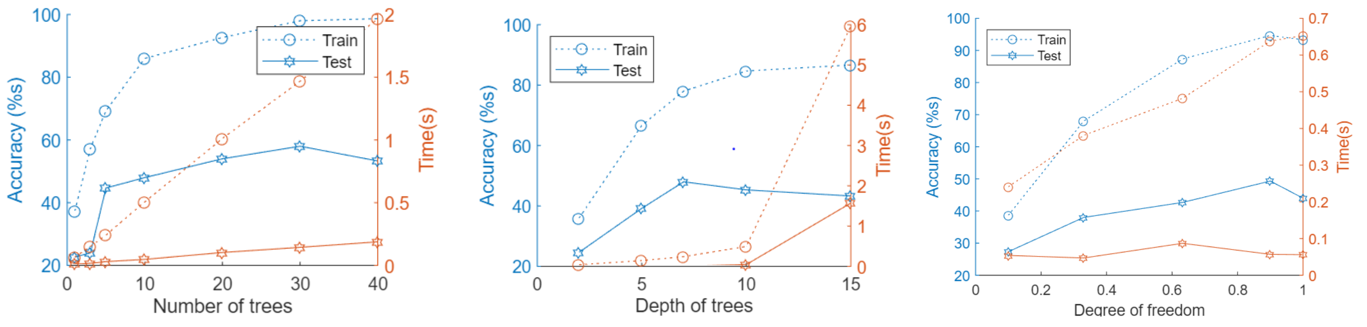
Fig. 12 is some examples of success/failure for each class.

C. Parameter tuning

We report in Fig. 13 the train, test accuracy and time while varying a specific parameter: number of trees, depth of trees, degree of randomness and type of week learners.

1) Number of trees: This number tradeoffs between increase generalization and time complexity. The performance is shown in Fig. 13a. As expected, when there are more trees, at first training and test accuracy grows sharply. Then, the accuracy grows with slower speed and saturates afterwards. 30 trees is the optimal setting.

Train and test time is expected to grow linearly. This is also confirmed in Fig. 13a.



(a) Number of trees) (b) Depth of trees (c) Degree of randomness

Fig. 13. Accuracy and time cost while verying a parameter

2) Depth of trees: This number tradeoffs between learning more discriminative features and avoiding overfitting to noise (bias-variance tradeoff). The performance is shown in Fig. 13b. As expected, when the depth increases, training accuracy grows continually (as more features is learned to fit the trainset). On the other hand, test accuracy initially grows to its maximum (at depth of 7) and drops afterwards. This confirms that under depth 7, RF underfits and over depth 7, RF overfits. Depth of 7 is the optimal value.

Test time is expected to grow exponentially (since number of nodes grows exponentially with depth). This is also confirmed in Fig. 13b.

3) Degree of randomness: This number tradeoffs between optimizing the 'weak learner'; and diversity (we want a diverse group of strong enough 'weak leaners'). The performance is shown in Fig. 13c. As expected, when T increases, training accuracy grows continually (as more data is fed to each tree). On the other hand, test accuracy initially grows to its maximum (at T = 0.9) and drops afterwards. This confirms that for smaller T, RF weak learners are too weak and for bigger T, RF weak learners are too similar.

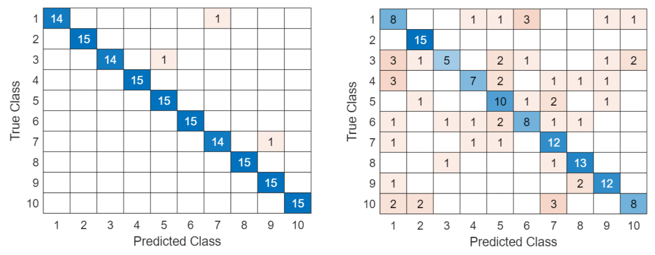
Test time is expected to grow linearly (since number of data grows linearly with T). This is also confirmed in Fig. 13c.

4) Type of weak learners: The result for 'two pixel test' weak leaners is similar to the default 'axis-align test'. Train and test accuracy is 84% and 49%. respectively.

5) Combine: Combine the set of optimized parameters (number of trees: 30, depth of trees: 7, degree of randomness 0.9, Weakleaners: axis-aligned), the train and test accuracy with confusion matrices are reported in Fig. 14.

D. *Effect of vocabulary size*

The result is shown in Fig. 15. Surprisingly, $k = 5$ outperforms other settings. This is attributed to relatively small size of the data (150 images). But this also means that better clustering does not directly translate to a better performance. The optimal k is problem-specific and also depends on the size of the data. Perhaps, 100k descriptors is too much for this small data.



(a) Train set (Accuracy 98%) (b) Test set (Accuracy 65%)

Fig. 14. Accuracy and confusion matrix for train set and test set

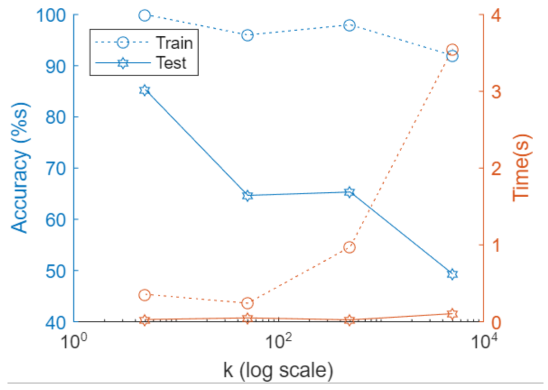


Fig. 15. Effect of vocabulary size (k)

E. *TF-IDF*

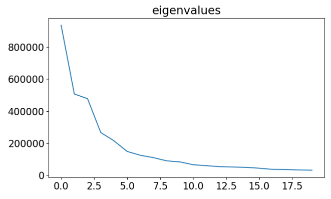
In NLP, TF-IDF is used to give more weights to important words, hence eliminate undesired effect of semantically insignificant words (such as stopwords). We adapt this method to handle the unimportant codewords (background, ...) and test with k = 500 (since for small k, TF-IDF does not add much meaning). The train and test accuracy reaches 96% and 66% for optimized parameters.

Acknowledgment

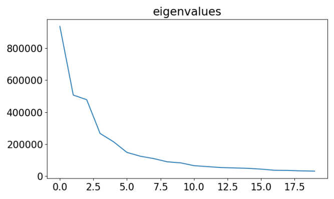
This coursework was done based on knowledge from module *'Special Topic in Computer Science: Machine Learning for Computer Vision'* at KAIST. The code is based on sample code provided in the mentioned course.

Appendix

A. *Top 20 eigenvalues (Q1)*



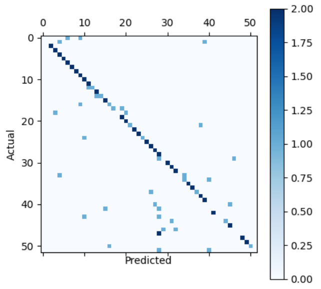
(a) Direct computation



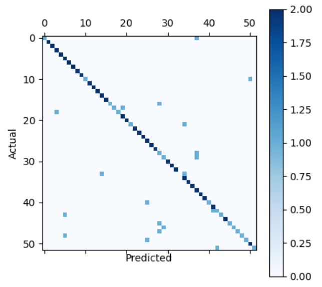
(b) Low-dimensional computation

Fig. 16. Graph for magnitude of eigenvalues

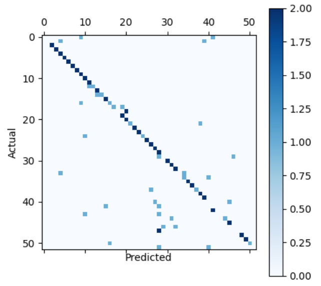
B. *Face recognition accuracy for various M_{pca} (Q2)*



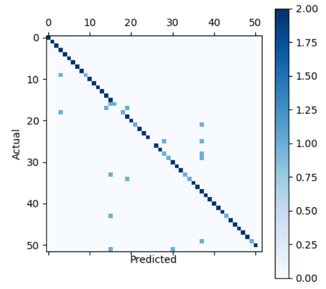
(a) PCA ($M_{pca} = 100$)



(b) PCA-LDA ($M_{pca} = 100$)



(c) PCA ($M_{pca} = 200$)



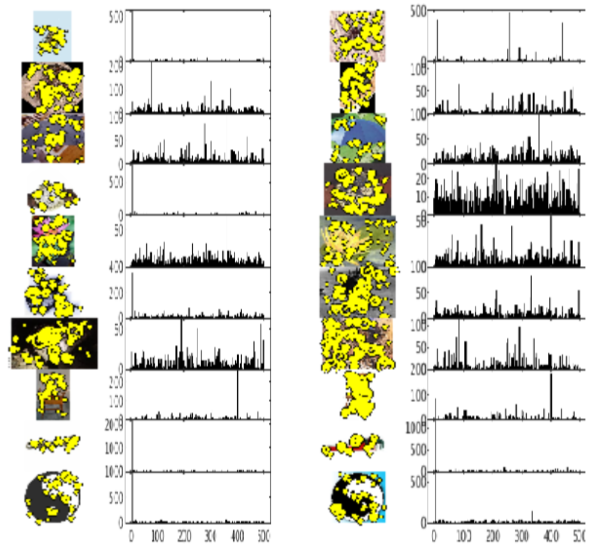
(d) PCA-LDA ($M_{pca} = 200$)

Fig. 17. Confusion matrix

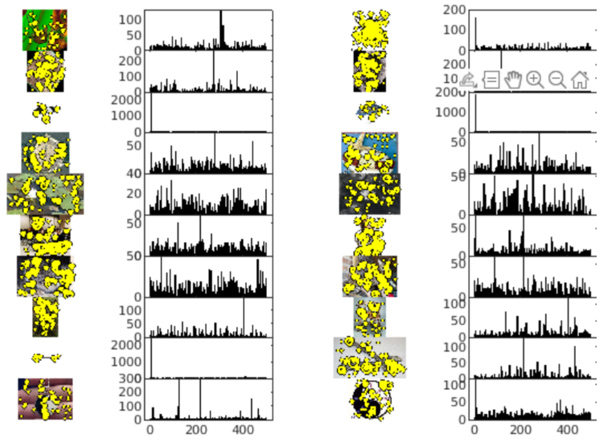
C. *Performance (Q3) for $M_{pca} = 100$*

-	**Incremental PCA**	**Batch PCA**	**PCA-1**
Training time (s)	0.080	0.082	0.006
Recognition accuracy (%)	67.31	67.31	66.34
Reconstruction error	581108	577257	1138571

D. *Bag-of-word histogram (Q4)*



(a) Train set



(b) Test set

Fig. 18. Example images with some SIFT patches (left) and corresponding histogram (right)