Pattern Recognition with Eigenfaces

Georgios Gryparis
Imperial College London
00598177
gg1409@ic.ac.uk

Paul Courty
Imperial College London
01247148
pc2816@ic.ac.uk

1. Computationally Efficient PCA

1.1. Dataset & Partitioning

The provided dataset is a $D \times N$ matrix, containing $N = 520$ face images, each of size $D = 46 \times 56 = 2576$ pixels. The N images describe 52 people with 10 images per person. Every image is pre-processed to be a 2D centered face portrait (Face Detection) to allow comparison for face recognition. The dataset was partitioned using a 4:1 ratio providing for a sufficient training set ($N_{train} = 416$) as well as a testing set ($N_{test} = 104$) with two images per identity.

1.2. Principal Component Analysis (PCA)

PCA consists of projecting a set of observations onto a feature space that spans the significant variations within the set. Variation is captured by the covariance matrix of our dataset. Its eigenvectors (principal components) form an uncorrelated orthogonal basis set which spans the whole image feature space, and its corresponding eigenvalues give the amount of variation in each of these dimensions. To perform PCA, we first normalize the training dataset by removing the mean face from each image. We define

$$A = [\,\varphi_1, \varphi_2, \dots, \varphi_N] \in \mathbb{R}^{N \times D}$$

where $\varphi_i = x_i - \bar{x}$ and $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ is the mean face

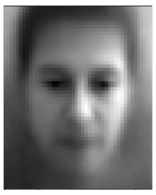


Figure 1. Mean Face of Training Dataset

In classic PCA, the covariance matrix is computed ($S = \frac{1}{N}AA^T \in \mathbb{R}^{D \times D}$), as well as its eigenvectors and eigenvalues. It should be noted that:

$$rank(S) = rank(AA^T) = rank(A) \le \min(D, N_{train})$$

As D > N_{train} and $\bar{x}$ a linear combination of vectors in x,

$$rank(A) = rank(x - \bar{x}) \le N_{train} - 1$$

Therefore, S has at most $N_{train} - 1$ non-zero eigenvalues.

Figure 2 shows the eigenvalues of S in descending order. Note that the plot covers only the 415 largest principle components, as the previous claim was empirically verified (a clear drop in order of magnitude, from 10^2 to 10^{-1} is observed after M = 415). Hence, the first 415 principle components hold all the information we need:

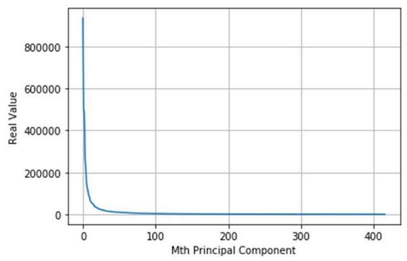


Figure 2. Eigenvalues of S in descending order

We exploit the above to reduce the time complexity and memory requirements without losing any information by using low-dimensional PCA. We compute the eigenvalues and eigenvectors of the matrix $S_{low} = \frac{1}{N}A^T A \in \mathbb{R}^{N \times N}$. Let λ and v be the eigenvalues and eigenvectors of S_{low}:

$$S_{low}v = \lambda v \quad \Leftrightarrow \quad A\frac{1}{N}A^T Av = A\lambda v \quad \Leftrightarrow \quad SAv = \lambda Av$$

Therefore, S_{low} has the same N_{train} eigenvalues as S, and the eigenvectors, u, of S can be computed by $u = Av$. Note that for reconstruction, we must normalize, $\|u\| = 1$.

This allows us to perform PCA with identical results and a significant reduction in computational time. Because the time complexity of eigendecomposition is $O(n^3)$, we expect much higher eigendecomposition time for $S \in \mathbb{R}^{D \times D}$ than for $S_{low} \in \mathbb{R}^{N \times N}$. This was empirically verified, 11.41 for S versus 0.12 seconds for S_{low}.

The low-dimensional PCA eigenvalues were identical to those obtained by classic PCA (Figure 2). The eigenvectors were also equivalent, with angular deviation of 0 or π.

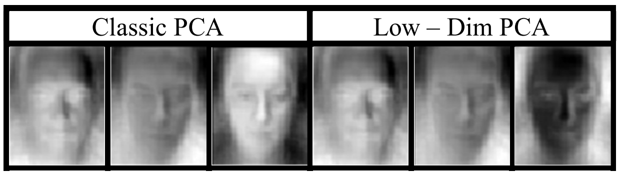


Figure 3. First three eigenfaces for both methods

1.3. Face Reconstruction

As seen in Figure 2, most information in our dataset is captured in the first few principle components. Depending on the accuracy required, we can attempt to reconstruct our data from first M principle components. The information captured in the first M principle components, described by$[100 \times \sum_{i=1}^{M} \lambda_i]/[\sum_{i=1}^{N_{train}} \lambda_i]$, is tabulated below.

Table 1. Information in M first principle components

Information Captured [%]	Principle Component M
50	4
75	17
90	63
95	118
99	263

Face reconstruction from the feature space to the input space (original images) is performed by adding the mean face to the weighted sum of the facial features (eigenface):

$$\tilde{x}_n = \bar{x} + \sum_{i=1}^{M} a_{ni} u_i \quad \text{where} \quad a_{ni} = \varphi_n^T u_i$$

According to Table 1, the reconstruction accuracy will increase the more principle components we add. The reconstruction error $J(M)$ can be subsequently computed. We defined the distortion measure as follows:

$$J(M) = \frac{1}{N} \sum_{n=1}^{N} \|x_n - \tilde{x}_n\|^2 = \sum_{i=M+1}^{N_{train}} \lambda_i$$

The above equality is verified empirically in Figure 4.

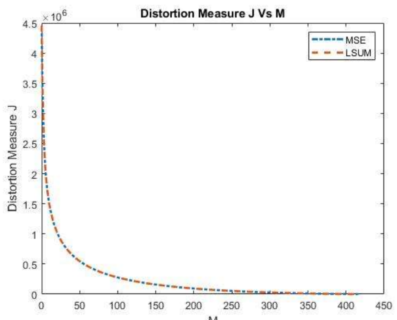


Figure 4. Reconstruction error J (Mean square error) and sum of N_train-M-1 last eigenvalues vs M

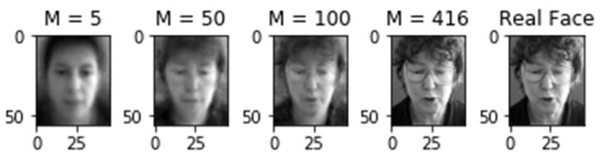


Figure 5. Reconstruction of image 1 for M = {5, 50, 100, 416}. Real image included for comparison. As expected, reconstruction improves with M (100% at M = 416)

1.4. Nearest Neighbor Classification

The Nearest Neighbor (NN) classification method was applied to our testing dataset. The testing images were projected in the feature space and individually compared to the projections of the training dataset in our model. The testing images were then labeled using the minimum Euclidian Distance, Figure 6. For completeness, kNN was also used for k=3, 5, 7 (see Appendix A). Best results were obtained using simple NN, k=1 (67% accuracy for M > 94).

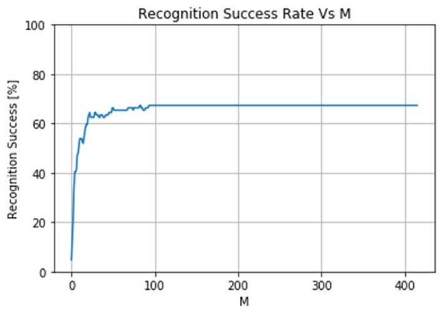


Figure 6. NN Recognition accuracy vs M

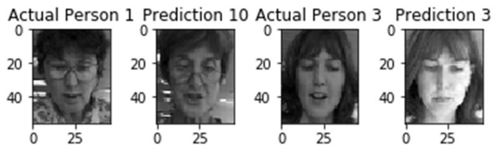


Figure 7. Example of failure and success in labelling

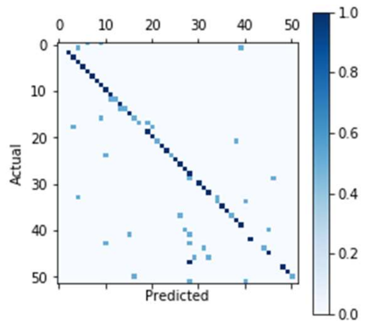


Figure 8. Normalized confusion matrix for M = 100

Finally, execution time and memory usage increase with M (Figure 9). As such there is a trade-off between recognition rate and computational cost. As accuracy plateaus for M > 94, larger values of M offer no benefit.

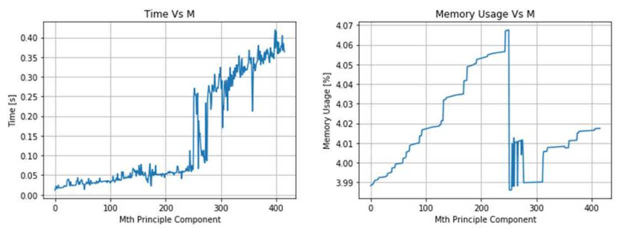


Figure 9. Execution time and Memory Usage of NN classification vs M

1.5. Alternative Classification Method

In this method, the principal (eigen) subspace is computed for the images in each class within our data. Test images are subsequently projected onto each eigenspace and their reconstruction error measured. Test images are assigned to the class whose eigenspace resulted in minimum reconstruction error.

Our training data has 52 classes each with 8 images, as such we calculate 52 eigenspaces spanned by at most 7 principle components each. We tabulate the accuracy of this method for varying hyperparameter M below.

Table 2. Alternative Method Accuracy vs M

M	1	2	3	4	5	6	7
Accuracy [%]	66.35	74.04	77.88	78.85	77.88	78.85	79.81

We note that this method produces much higher peak accuracy than kNN classification (80% vs 67%)

2. Incremental PCA

2.1. Data partitioning and Algorithm description

The training data is further partitioned into four subsets ($\mathbf{X_1}$, $\mathbf{X_2}$, $\mathbf{X_3}$, $\mathbf{X_4}$), $\mathbf{X_i} \in \mathrm{R}^{2576\times104}$, each comprised of 104 images (2 images per person).

Incremental PCA allows us to update an existing eigenspace to account for new observations. This is important because it results in reduced memory usage making it possible to deal with large problems. It can, also, under conditions, be faster to create an eigenmodel for a set of observations incrementally, than by batch computation. In solutions including multiple updates, the accuracy of the model suffers; however, when only few updates are performed the accuracy is on par with the batch method.

For a set of observations $\mathbf{X} \in \mathrm{R}^{D\times N}$, comprised of k subsets $\{\mathbf{X_1}, \mathbf{X_2}, \dots \mathbf{X_k}\}$, IPCA involves the following k steps:

<u>i=1:</u>	Use low dimensional PCA to generate an eigenmodel of $\mathbf{X_1}$: $\mathbf{E_1} = \{N_1, \boldsymbol{\mu}_1, \mathbf{P_1}, \boldsymbol{\Lambda}_1\}$.[1]
<u>i=2…k:</u>	Use low dimensional PCA to generate an eigenmodel of $\mathbf{X_i}$: $\mathbf{E_i} = \{N_i, \boldsymbol{\mu}_i, \mathbf{P_i}, \boldsymbol{\Lambda}_i\}$. Set $\mathbf{E_2}=\mathbf{E_i}$ and merge $\mathbf{E_1}$ and $\mathbf{E_2}$ to produce $\mathbf{E_3}$. Update $\mathbf{E_1}=\mathbf{E_3}$

To merge $\mathbf{E_1}$ and $\mathbf{E_2}$ we use the following procedure:

1. $N_3 = N_1+N_2$, $\boldsymbol{\mu}_3 = (N_1\boldsymbol{\mu}_1 + N_2\boldsymbol{\mu}_2)/N_3$ and $\boldsymbol{\Delta\mu} = \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$
2. Find the sufficient spanning set $\boldsymbol{\Phi}$ of $[\mathbf{P_1}\ \mathbf{P_2}\ \boldsymbol{\Delta\mu}]$ by QR decomposition.
3. Calculate $\mathbf{S_3}$ (formula in Appendix B)
4. Solve the eigenproblem $\mathbf{S_3} = \mathbf{R}\,\boldsymbol{\Lambda}_3\,\mathbf{R}^T$.
5. Keep only d_3 eigenvalues and eigenvectors.
6. Then $\mathbf{P_3} = \boldsymbol{\Phi}\mathbf{R}$, which is subsequently normalized.

2.2. Time Complexity

The total time complexity of low-dimensional PCA on a $D\times N$ dataset (including matrix multiplications, eigenvalue decomposition and other steps) is $O(N^{q1}D)$ [2]. The merging of two eigenmodels $D\times d_1$ and $D\times d_2$ has complexity $O(d_{tot}^{q2}\,D)$ with $d_{tot} = d_1+d_2+1$ [3] (details in Appendix B). Hence, for incremental PCA on $\mathbf{X}$, comprised of k subsets $\{\mathbf{X_1}, \mathbf{X_2}, \dots \mathbf{X_k}\}$ we expect:

1. Since $\sum_{i=1}^{k} N_i^{q1} < \left(\sum_{i=1}^{k} N_i\right)^{q1} = N^{q1}$ we expect that the total time to compute eigenmodels for all k subsets will be smaller than the batch time: $\sum_{i=1}^{k} t_i < t_B$.

If we set $N_i = n = \frac{N}{k}$ and $d_i = d \Rightarrow d_{tot} = 2d+1$:

2. The time to merge eigenmodels $\mathbf{E_1}$ and $\mathbf{E_2}$ in step i, t_{mi}, is equal in all steps ($t_{mi}=t_m\ \forall i$). Similarly, all eigenmodel computation times are equal ($t_i=t\ \forall i$).
3. When $d_{tot} \ll n$, then $t_m \ll t$ and thus total time for IPCA t_{tot}, is smaller than the batch time ($t_{tot} \approx k \times t < t_B$).
4. As d_{tot} increases, at some point t_{tot} will become equal to t_B. This is significant as it marks the maximum value of d_{tot} for which IPCA is faster than batch PCA.
5. When $d_{tot} \gg n$, the value of t_m will keep increasing.

To verify the above, we measure the time to compute the eigenmodel of our training data by batch PCA, Incremental PCA and time to compute the eigenmodel of $\mathbf{X_1}$. For ease of comparison we set the number of principal components $d_i = d \Rightarrow d_{tot} = 2d+1$. Results are plotted below (Fig 10). Logscale has been utilized for better visualization. Results are the execution times averaged over 20 recordings.

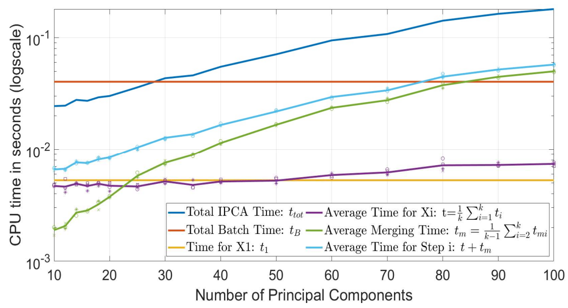


Figure 10. Computation time for Batch PCA, IPCA and PCA on 1st subset only vs d

Individual markers in Figure 10 represent t_i, t_{mi} for each value of d and are averaged to create plots for t and t_m. As expected, they all fall very close to their respective means (t_i, t_{mi} are approximately constant).

[1] N_1 is the number of observations in $\mathbf{X_1}$, $\boldsymbol{\mu}_1$ is the mean of $\mathbf{X_1}$, $\boldsymbol{\Lambda}_1$ contains d_1 largest eigenvalues, $\mathbf{P_1}$ the corresponding eigenvectors

[2] $q_1=\max\{p_1, 3-\log_{N_i}(D)\}$ and $1<p_1<2$ depending on the algorithms applied

[3] $q_2=\max\{p_2, 3-\log_{d_{tot}}(D)\}$ and $1<p_2<2$ depending on the algorithms applied

For d > 25, the merging time (t_m) exceeds the time to produce an eigenmodel of the next subset (t_m>t). For d > 30, the IPCA becomes slower than the Batch computation. The time necessary to eigenmodel the first subset is very close to the average time to calculate an eigenmodel of any of the subsets. Finally, the Batch calculation is much slower than calculating the eigenmodels of subspaces (which is expected as they differ in size by a factor 4).

We conclude that if we are prepared to take less than 30 principal components (thus setting an upper bound to our accuracy) we can use incremental PCA which will be faster and more memory efficient. Note that for this to be valid the batch and incremental PCA must perform similarly in terms of accuracy (see section 2.4).

2.3. Face Reconstruction

Face reconstruction error for PCA trained on $\mathbf{X_1}$ (PCA_{SET-1}), Incremental and Batch PCA trained on $\mathbf{X}$ (PCA_{INC} and PCA_B) were computed using the same methods as in subsection 1.3. See Figure 11. Note that in the incremental case M represents the number of principle components kept in each step (denoted by d in section 2.2).

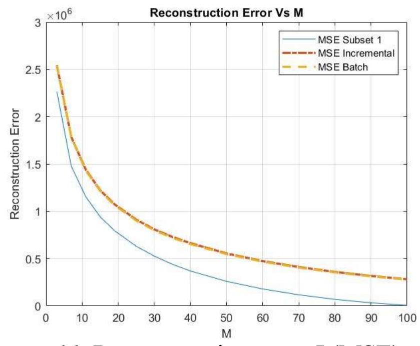


Figure 11. Reconstruction error J (MSE) vs M for PCA_{SET-1}, PCA_{INC} and PCA_B (M < $N_{subset1}$=104)

We note that PCA_{SET-1} has the lowest reconstruction error for a given value of M. This is expected, since $\mathbf{X_1}$ has all of its information in $N_1 - 1 = 103$ principle components; whereas $\mathbf{X}$ requires $N_{train} - 1 = 415$ components.

To explain why PCA_{INC} and PCA_B have almost identical reconstruction errors, we must compare their eigenvectors. We quantify this by the angular deviation $\cos^{-1}\left(\frac{<u_i,\ v_i>}{||u_1||\ ||v_i||}\right)$

Table 3.
Mean angular deviation between PCA_{INC} and PCA_B vs M

Principal Components M	Mean Angular Deviation [°]
7	0.13
15	0.20
40	0.32
100	0.35

The mean angular deviation is small, explaining why the reconstruction error is so similar.

As seen in Figure 12 most of the deviation comes from the last eigenvectors (those capturing little information, corresponding to small eigenvalues). Thus, the Mean Angular Deviation overestimates the information lost during incremental PCA.

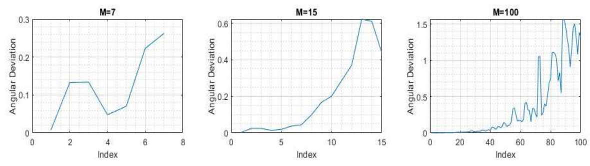


Figure 12. Angular deviation vs eigenvector index for M = {7, 15, 100}

Absolute percentage error in eigenvalues of PCA_{INC} vs PCA_B is tabulated in Appendix A.

2.4. Face Recognition

Face recognition was performed using k Nearest Neighbors classification. We observe that PCA_{SET-1} performs poorly and has a peak recognition performance of 28% at $k = 3$ and $M = 11$. The poor performance can be explained by the small sample size. PCA_{INC} and PCA_B perform very similarly. Both have peak recognition performance of 67% at $k = 1$ and $M = 94$. This is expected as the two models have very small differences.

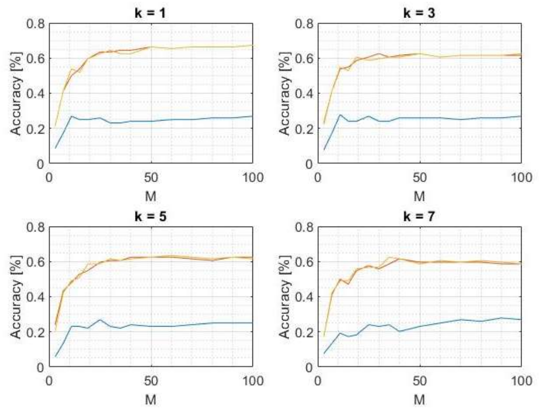


Figure 13. kNN recognition accuracy vs M for PCA_{SET-1}, PCA_{INC}, PCA_B $\left(\boldsymbol{k} = \{\mathbf{1}, \mathbf{3}, \mathbf{5}, \mathbf{7}\}\right)$.
Blue: PCA_{SET-1}, Red: PCA_{INC}, Yellow: PCA_B.

The most important parameter for incremental PCA is the number of principle components kept in each step. These can either be set to a constant value (M) or implicitly set by setting a threshold for the eigenvalues whose corresponding eigenvectors should be considered (e.g. keep eigenvectors containing more than some percentage of the total information)

3. PCA-LDA Ensemble for Face Recognition

3.1. PCA-LDA

3.1.1. Theoretical Background

Linear Discriminant Analysis (LDA) consists of transforming the provided data, through feature space rotation, to find the direction that best separates distinctive classes of data. Whereas PCA increases data variance both within and between classes, LDA only increases data variance between classes and minimizes within classes. This allows for clearer and more distinct clusters to form in feature space, ultimately increasing face recognition accuracy. The optimization problem can be summarized by:

$$\boldsymbol{W}_{pca}=\text{argmax}_{\text{W}}|W^T S_T W|, \quad \boldsymbol{W}_{lda}=\text{argmax}_{\text{W}}\left|\frac{W^T S_B W}{W^T S_W W}\right|$$

S_W is the within class scatter matrix, S_B is the between class scatter matrix, and the total scatter matrix $S_T=S_W+S_B$.

In the case of LDA, finding the direction that best separates data from different classes is equivalent to maximizing the numerator ($W^T S_B W$) and minimizing the denominator ($W^T S_W W$). The solution to this optimization problem, given that S_W is non-singular, are the eigenvectors of the matrix $S_W^{-1} S_B$:

$$S_W^{-1} S_B W = \lambda W$$

However, $S_W \in \mathbb{R}^{D\times D}$ has a maximum rank of $N-c$, where c is the number of classes in the dataset. In our case $N<D$, meaning that S_W is likely singular. Therefore, PCA is implemented first to reduce the overall dimension. We keep the M_{pca} eigenvectors with largest eigenvalues, and then perform LDA. Thus, the previous equation becomes:

$$\left(W_{pca}^T S_W W_{pca}\right)^{-1}\left(W_{pca}^T S_B W_{pca}\right)W = \lambda W$$

The M_{lda} eigenvectors of W with the largest eigenvalues are kept, forming the matrix W_{lda}. Both M_{pca} and M_{lda} are restricted by the rank of the scatter matrices S_B and S_W respectively, $c-1=51$ and $N-c=364$ respectively. Finally, combining the two methods, we obtain the following optimal transformation:

$$W_{opt}^T = W_{lda}^T W_{pca}^T$$

3.1.2. Empirical Results

We performed PCA-LDA on our data and found that in our empirical results $\text{rank}(S_B)=51$ $\text{rank}(S_w)=364$.

We then used Nearest Neighbour to classify our testing data. The results are displayed in Figure 14 and Figure 15 for different hyperparameter values M_{pca} and M_{lda}.

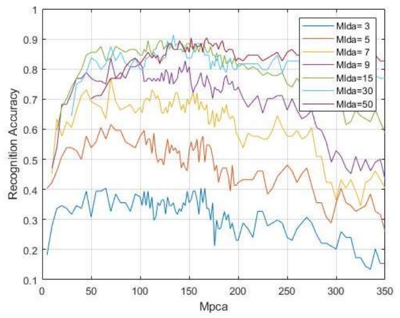
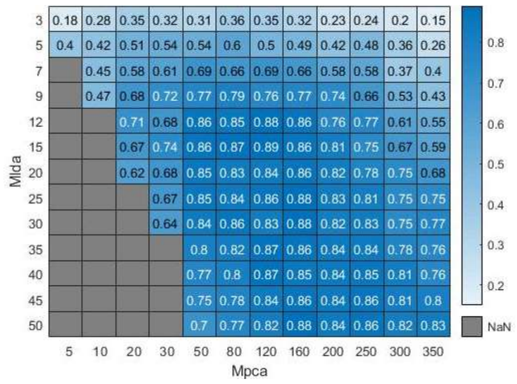


Figure 14. Recognition Accuracy vs M_{pca} for varying M_{lda}



Mlda \ Mpca	5	10	20	30	50	80	120	160	200	250	300	350
3	0.18	0.28	0.35	0.32	0.31	0.36	0.35	0.32	0.23	0.24	0.2	0.15
5	0.4	0.42	0.51	0.54	0.54	0.6	0.5	0.49	0.42	0.48	0.36	0.26
7		0.45	0.58	0.61	0.69	0.66	0.69	0.66	0.58	0.58	0.37	0.4
9		0.47	0.68	0.72	0.77	0.79	0.76	0.77	0.74	0.66	0.53	0.43
12			0.71	0.68	0.86	0.85	0.88	0.86	0.76	0.77	0.61	0.55
15			0.67	0.74	0.86	0.87	0.89	0.86	0.81	0.75	0.67	0.59
20			0.62	0.68	0.85	0.83	0.84	0.86	0.82	0.78	0.75	0.68
25				0.67	0.85	0.84	0.86	0.88	0.83	0.81	0.75	0.75
30				0.64	0.84	0.86	0.83	0.88	0.82	0.83	0.75	0.77
35					0.8	0.82	0.87	0.86	0.84	0.84	0.78	0.76
40					0.77	0.8	0.87	0.85	0.84	0.85	0.81	0.76
45					0.75	0.78	0.84	0.86	0.84	0.86	0.81	0.8
50					0.7	0.77	0.82	0.88	0.84	0.86	0.82	0.83

Figure 15. Accuracy for various M_{lda} and M_{pca}

A maximum accuracy of 91.35% was obtained for $M_{pca}=134$ and $M_{lda}=30$, which is better than the accuracy for any PCA method previously applied. To obtain peak accuracy >70% we need $M_{lda}>9$, for peak accuracy >85% we need $M_{lda}>12$. Furthermore, we must ensure $M_{PCA}<200$ to avoid overfitting and $M_{PCA}>50$ to avoid underfitting. Note that by definition $M_{PCA}>M_{lda}$.

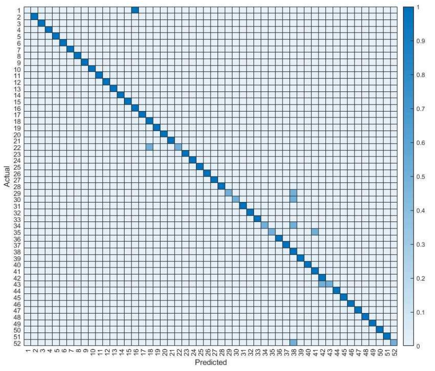


Figure 16. Normalized Confusion Matrix for M_{lda}=30, M_{PCA}=134

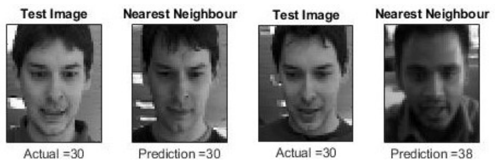


Figure 17. Example of success and failure in labelling

3.2. PCA-LDA Ensemble

3.2.1. Randomization in feature space

Introducing randomness to our model predictions results in improved generalization and robustness. One way of doing so, is sampling randomly our feature space. We perform PCA to our dataset and keep the M0 eigenvectors with largest eigenvalues. From the remaining N – 1 – M0 eigenvectors, we randomly select M1 eigenvectors. These steps are performed a T number of times, with replacement, to create T number of decorrelated submodels W_{pca}. LDA is then applied to all these T submodels. As we do not have the space to investigate M_{lda} as a variable, we set $M_{lda} = 51$ ensuring that all of the information after PCA is captured.

Face recognition is performed using NN classification on each submodel separately and then fusion rules are used to label the data. The hyperparameters T, M0 and M1 are varied and their effects measured. The fusion rules used are simple majority voting and sum rules. Fusion sum rule is described by:[4]

$$\text{label}_{\text{predicted}}(x) = \text{argmax}_{l_i} \sum_{t=1}^{T} P(l_i|x,t)$$

where $P(l_i|x,t)$ is the probability the true label of x is l_i when the t^th individual model is used. To estimate these probabilities, we use:

$$\widehat{P}(l_i|x,t) = \left(1 + \frac{(w_t^x)^T w_t^i}{\left||w_t^x|\right| \left||w_t^i|\right|}\right)/2$$

where $w_t^i = W_{opt,t}^T \mu_i$ and $w_t^x = W_{opt,t}^T x$. Note that the final estimates were scaled to ensure that $\sum_{i=1}^{c} \widehat{P}(l_i|x,t) = 1$, where c is the number of classes in the data.

3.2.2. Empirical Results

The recognition accuracy was plotted against M0 with varying M1 and T. Peak values of PCA-LDA ensemble were higher than peak PCA-LDA accuracy.

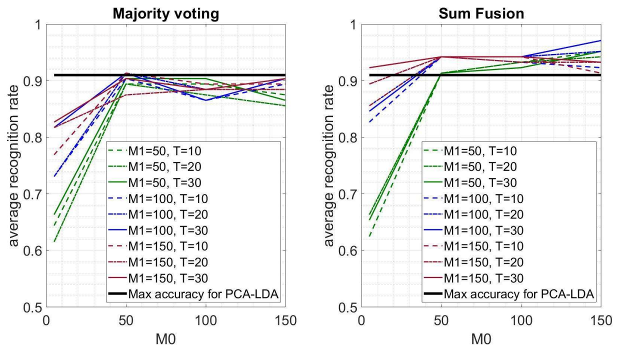


Figure 18. The recognition accuracy vs M0 for $M_{lda} = 51$ and a range of M1, T using majority voting and sum rule.

[4] Wang, Xiaogang and Tang, Xiao; Random Sampling for Subspace Face Recognition, The Chinese University of Hong Kong, February 24, 2005

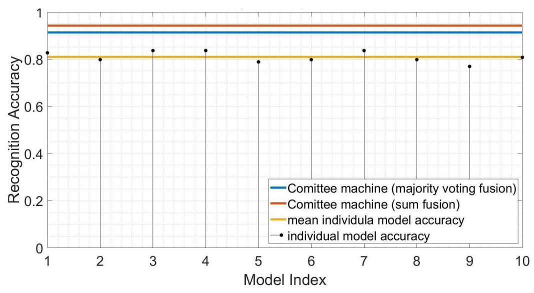


Figure 19. Accuracy for PCA-LDA ensemble and for individual models vs model index [t] for M_{LDA}=51, T=10, M0=50, M1=150

As can be seen from Figures 17 and 18 the sum rule outperforms majority voting significantly in term of peak accuracy (92% vs 97% respectively). For some combinations of hyperparameters, majority voting is more accurate. Both fusion rules are an improvement over PCA-LDA as well as the average of individual model accuracies illustrating the effectiveness of randomization.

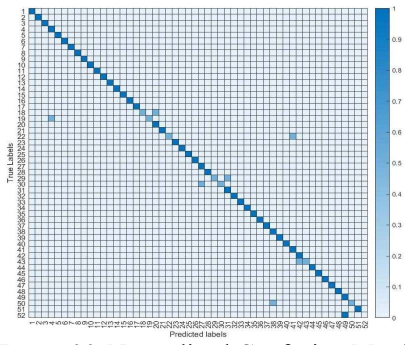


Figure 20. Normalized Confusion Matrix for M_{LDA}=51, T=10, M0=50, M1=150

4. Summary of Results

We explored several dimensionality-reduction methods and classification algorithms for face recognition. Parameters were varied to find optimal performance within each method. Peak accuracies found within the set of parameters investigated are summarized in Table 4.

Table 4. Peak recognition accuracies of the methods used with their respective optimal hyperparameters

Method	Accuracy [%]	Hyperparameters
PCA kNN	67.31	k = 1, M > 94
PCA Alt-Method	79.81	M = 7
IPCA kNN	67.31	k = 1, M > 94
PCA-LDA	91.35	M_{PCA} = 134, M_{lda} = 30
PCA-LDA Feature Space Randomization	97	M_{lda} = 51, T = 10, M0 = 50, M1 = 150

Appendix A

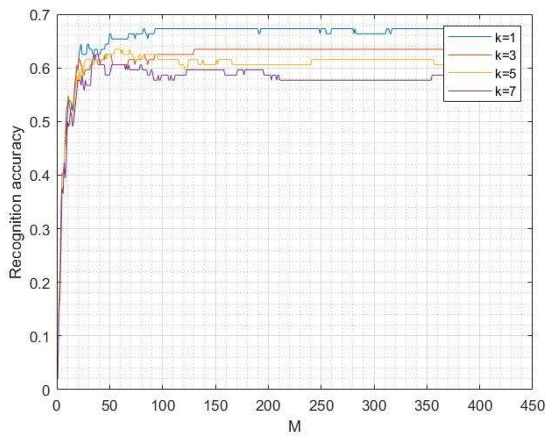


Figure 21. Recognition accuracy of kNN algorithm vs M (k=1, 3, 5, 7). This figure refers to section 1.4.

Table 5. Average absolute eigenvalue percentage error between PCA_{INC} and PCA_B vs M. This table refers to section 2.3

Principal Components M	Average absolute eigenvalue percentage error
7	0.0526
15	0.0572
40	0.0589
100	0.0217

Appendix B

To explain the reasoning behind the time complexities of deduced in section 2.2 for the Incremental PCA we describe the steps involved.

The eigenmodel for $\mathbf{X_i}$ is generated by low dimensional PCA which involves two types of computationally complex tasks: matrix multiplication $N_i \times D$ with $D \times N_i$ matrices and eigendecomposition of $N_i \times N_i$ matrix. The matrix multiplication is naively $O(N_i^2 D)$. Smart algorithms reduce this to $O(N_i^{p1} D)$, with $p_1 \approx 1.3$ when $1 < \log_{N_i}(D) < 5$ [5]. The eigendecomposition has complexity $O(N_i^3)$. Hence, the total complexity is $O(N_i^{q1} D)$, with $q_1 = \max\{p_1, 3 - \log_{N_i}(D)\}$.

To merge $\mathbf{E_1}$ and $\mathbf{E_2}$ we use the following procedure:

1. $N_3 = N_1 + N_2$, $\boldsymbol{\mu_3} = (N_1\boldsymbol{\mu_1} + N_2\boldsymbol{\mu_2})/N_3$ and $\boldsymbol{\Delta\mu} = \boldsymbol{\mu_1} - \boldsymbol{\mu_2}$
2. Find the sufficient spanning set $\boldsymbol{\Phi}$ of $[\mathbf{P_1}\ \mathbf{P_i}\ \boldsymbol{\Delta\mu}]$ by QR decomposition. Define $d_{tot} = d1+d2+1$, then step involves the QR decomposition of a $D \times d_{tot}$ matrix which is naively $O(d_{tot}^2 D)$ but improved algorithms reduce this to $O(d_{tot}^{p2} D)$ with $1 < p_2 < 2$ depending on size of d_{tot} vs D and the characteristics of $[\mathbf{P_1}\ \mathbf{P_i}\ \boldsymbol{\Delta\mu}]$.
3. Calculate $\mathbf{S_3} = \mathbf{A_1} + \mathbf{A_2}$ where:
$$\mathbf{A_1} = \frac{N_1}{N_3}\boldsymbol{\Phi}^T \mathbf{P_1}\boldsymbol{\Lambda_1}(\boldsymbol{\Phi}^T\mathbf{P_1})^T + \frac{N_2}{N_3}\boldsymbol{\Phi}^T \mathbf{P_2}\boldsymbol{\Lambda_2}(\boldsymbol{\Phi}^T\mathbf{P_2})^T$$
$$\mathbf{A_2} = \frac{N_1 N_2}{N_3^2}\ \boldsymbol{\Phi}^T \boldsymbol{\Delta\mu}\ (\boldsymbol{\Phi}^T \boldsymbol{\Delta\mu})^T$$
Note: $\mathbf{A_1}$, $\mathbf{A_2}$ are $d_{tot} \times d_{tot}$. Their calculation requires multiplication of $d_{tot} \times D$ with $D \times d1$ and $D \times d2$ matrices. The above formula differs from the lecture notes but is equivalent and reduces complexity.
4. Solve the eigenproblem $\mathbf{B} = \mathbf{R}\ \boldsymbol{\Lambda_3}\ \mathbf{R}^T$. Since B is a $d_{tot} \times d_{tot}$ matrix, the complexity is $O(d_{tot}^3)$.
5. Keep only d_3 eigenvalues and eigenvectors.
6. Then $\mathbf{P_3} = \boldsymbol{\Phi}\mathbf{R}$, which is subsequently normalized.

Hence, the process of merging two eigenmodels has total complexity $O(d_{tot}^{q2} D)$ with $q_2 = \max\{p_2, 3 - \log_{dtot}(D)\}$.

[5] Le Gall, François and Urrutia, Floren; Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor (2018)