Online Incremental Face Recognition System Using Eigenface Feature and Neural Classifier

Seiichi Ozawa[1], Shigeo Abe[1], Shaoning Pang[2] and Nikola Kasabov[2]
[1]*Graduate School of Engineering, Kobe University,*
[2]*Knowledge Engineering & Discover Research Institute, Auckland University of Technology*
[1]*Japan,*
[2]*New Zealand*

1. Introduction

Understanding how people process and recognize faces has been a challenging problem in the field of object recognition for a long time. Many approaches have been proposed to simulate the human process, in which various adaptive mechanisms are introduced such as neural networks, genetic algorithms, and support vector machines (Jain et al., 1999). However, an ultimate solution for this is still being pursued. One of the difficulties in the face recognition tasks is to enhance the robustness over the spatial and temporal variations of human faces. That is, even for the same person, captured images of human faces have full of variety due to lighting conditions, emotional expression, wearing glasses, make-up, and so forth. And the face features could be changed slowly and sometimes drastically over time due to some temporal factors such as growth, aging, and health conditions.

When building a face recognition system, taking all the above variations into consideration in advance is unrealistic and maybe impossible. A remedy for this is to make a recognition system evolve so as to make up its misclassification on its own. In order to construct such an adaptive face recognition system, so-called *incremental learning* should be embedded into the system because it enables the system to conduct learning and classification on an ongoing basis. One challenging problem for this type of learning is to resolve so-called "plasticity and stability dilemma" (Carpenter & Grossberg, 1988). Thus, a system is required to improve its performance without deteriorating classification accuracy for previously trained face images.

On the other hand, feature extraction plays an essential role in pattern recognition because the extraction of appropriate features results in high generalization performance and fast learning. In this sense, incremental learning should be considered not only for a classifier but also for the feature extraction part. As far as we know, however, many incremental learning algorithms are aiming for classifiers. As for the incremental learning for feature extraction, Incremental Principal Component Analysis (IPCA) (e.g., Oja & Karhunen, 1985; Sanger, 1989; Weng et al., 2003; Zhao et al., 2006) and Incremental Linear Discriminant Analysis (Pang et al., 2005; Weng & Hwang, 2007) have been proposed so far. Hall and Martin (1998) proposed a method to update eigen-features (e.g., eigen-faces) incrementally based on eigenvalue decomposition. Ozawa et al. (2004) extended this IPCA algorithm such that an eigen-axis was augmented based on the accumulation ratio to control the dimensionality of an eigenspace easily.

Source: State of the Art in Face Recognition, Book edited by: Dr. Mario I. Chacon M.,
ISBN -3-902613-42-4, pp. 250, January 2009, I-Tech, Vienna, Austria

Recently, a prototype face recognition system was developed by the authors (Ozawa et al, 2005) based on a new learning scheme in which a classifier and the feature extraction part were simultaneously learned incrementally. In this system, IPCA was adopted as an online feature extraction algorithm, and Resource Allocating Network with Long-Term Memory (RAN-LTM) (Kobayashi et al., 2001) was adopted as a classifier model. It was verified that the classification accuracy of the above classification system was improved constantly even if a small set of training samples were provided at a starting point. To accelerate learning of IPCA, we also proposed an extended algorithm called *Chunk IPCA* (Ozawa et al., 2008) in which an eigenspace is updated for a chunk of given training examples by solving a single intermediate eigenvalue problem.
The aim of this chapter is to demonstrate the followings:

1. how the feature extraction part is evolved by IPCA and Chunk IPCA,
2. how both feature extraction part and classifier are learned incrementally on an ongoing basis,
3. how an adaptive face recognition system is constructed and how it is effective.

This chapter is organized as follows. In Section 2, IPCA is first reviewed, and then Section 3 presents the detailed algorithm of Chunk IPCA. Section 4 explains two neural classifier models: Resource Allocating Network (RAN) and its variant model called RAN-LTM. In Section 5, an online incremental face recognition system and its information processing are described in detail, and we also explain how to reconstruct RAN-LTM when an eigenspace model is dynamically updated by Chunk IPCA. In Section 6, the effectiveness of incremental learning in face recognition systems is discussed for a self-compiled face image database. Section 7 gives conclusions of this chapter.

2. Incremental Principal Component Analysis (IPCA)

2.1 Learning assumptions and outline of IPCA algorithm

Assume that N training samples $\mathbf{x}^{(i)} \in R^n$ $(i = 1, \ldots, N)$ are initially provided to a system and an eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$ is obtained by applying Principal Component Analysis (PCA) to the training samples. In the eigenspace model Ω, $\bar{\mathbf{x}}$ is a mean vector of $\mathbf{x}^{(i)}$ $(i = 1, \ldots, N)$, $\mathbf{U}_k$ is an $n \times k$ matrix whose column vectors correspond to eigenvectors, and $\mathbf{\Lambda}_k = \mathrm{diag}\{\lambda_1, \ldots, \lambda_k\}$ is a $k \times k$ matrix whose diagonal elements are non-zero eigenvalues. Here, k is the number of eigen-axes spanning the eigenspace (i.e., eigenspace dimensionality) and the value of k is determined based on a certain criterion (e.g., accumulation ratio). After calculating Ω, the system keeps the information on Ω and all the training samples are thrown away.
Now assume that the $(N+1)$th training sample $\mathbf{x}^{(N+1)} = \mathbf{y} \in R^n$ is given. The addition of this new sample results in the changes in the mean vector and the covariance matrix; therefore, the eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$ should be updated. Let us define the new eigenspace model by $\Omega' = (\bar{\mathbf{x}}', \mathbf{U}'_{k'}, \mathbf{\Lambda}'_{k'}, N+1)$. Note that the eigenspace dimensions might be increased from k to $k+1$; thus, k' in Ω' is either k or $k+1$. Intuitively, if almost all energy of $\mathbf{y}$ is included in the current eigenspace spanned by the eigenvectors $\mathbf{U}'_k$, there is no need to increase an eigen-axis. However, if $\mathbf{y}$ includes certain energy in the complementary eigenspace, the dimensional augmentation is inevitable; otherwise crucial information on $\mathbf{y}$ might be lost. Regardless of the necessity in eigenspace augmentation, the

eigen-axes should be rotated to adapt to the variation in the data distribution. In summary, there are three main operations in IPCA: (1) mean vector update, (2) eigenspace augmentation, and (3) eigenspace rotation. The first operation is easily carried out without past training examples based on the following equation:

$$\bar{\mathbf{x}}' = \frac{1}{N+1}\left(N\bar{\mathbf{x}} + \mathbf{y}\right) \ \in R^n . \tag{1}$$

Hence, the following subsections give the explanation only for the last two operations.

2.2 Eigenspace augmentation

There have been proposed two criteria for judging eigenspace augmentation. One is the norm of a residue vector defined by

$$\mathbf{h} = (\mathbf{y} - \bar{\mathbf{x}}) - \mathbf{U}_k^T \mathbf{g} \quad \text{where} \quad \mathbf{g} = \mathbf{U}_k^T (\mathbf{y} - \bar{\mathbf{x}}) . \tag{2}$$

Here, T means the transposition of vectors and matrices. The other is the accumulation ratio whose definition and incremental calculation are shown as follows:

$$A(\mathbf{U}_k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j} = \frac{(N+1)\sum_{i=1}^{k} \lambda_i + \left\| \mathbf{U}_k^T (\mathbf{y} - \bar{\mathbf{x}}) \right\|^2}{(N+1)\sum_{j=1}^{n} \lambda_j + \left\| \mathbf{y} - \bar{\mathbf{x}} \right\|^2} \tag{3}$$

where λ_i is the ith largest eigenvalues, n and k mean the dimensionality of the input space and that of the current eigenspace, respectively. The former criterion in Eq. (2) was adopted in the original IPCA (Hall & Martin, 1998), and the latter in Eq. (3) was used in the modified IPCA proposed by the authors (Ozawa et al., 2004). Based on these criteria, the condition of increasing an eigen-axis $\hat{\mathbf{h}}$ is represented by:

$$\text{[Residue Vector Norm]} \quad \hat{\mathbf{h}} = \begin{cases} \mathbf{h}/\|\mathbf{h}\| & \text{if } \|\mathbf{h}\| > \eta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\text{[Accumulation Ratio]} \quad \hat{\mathbf{h}} = \begin{cases} \mathbf{h}/\|\mathbf{h}\| & \text{if } A(\mathbf{U}_k) < \theta \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where η (in the original IPCA, η is set to zero) and θ are positive constants. Note that setting a too large threshold η or too small θ would cause serious approximation errors for eigenspace models. Hence, it is important to set proper values to η in Eq. (4) and θ in Eq. (5). In general, finding a proper threshold η is not easy unless input data are appropriately normalized within a certain range. On the other hand, since the accumulation ratio is defined by the ratio of input energy in an eigenspace over the original input space, the value of θ is restricted between 0 and 1. Therefore, it would be easier for θ to get an optimal value by applying the cross-validation method. The detailed algorithm of finding θ in incremental learning settings is described in Ozawa et al. (2008).

2.3 Eigenspace rotation

If the condition of Eq. (4) or (5) satisfies, the dimensions of the current eigenspace would be increased from k to $k+1$, and a new eigen-axis $\hat{\mathbf{h}}$ is added to the eigenvector matrix $\mathbf{U}_k$. Otherwise, the dimensionality remains the same. After this operation, the eigen-axes are rotated to adapt to the new data distribution. Assume that the rotation is given by a rotation matrix $\mathbf{R}$, then the eigenspace update is represented by the following equation:

1) If there is a new eigen-axis to be added, $\mathbf{U}'_{k+1} = [\mathbf{U}_k, \hat{\mathbf{h}}]\mathbf{R}$, (6)

2) otherwise, $\mathbf{U}'_k = \mathbf{U}_k \mathbf{R}$. (7)

It has been shown that $\mathbf{R}$ is obtained by solving the following intermediate eigenproblem (Hall & Martin, 1998):

1. If there is a new eigen-axis to be added,

$$\left\{ \frac{N}{N+1} \begin{bmatrix} \mathbf{\Lambda}_k & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \frac{N}{(N+1)^2} \begin{bmatrix} \mathbf{g}\mathbf{g}^T & \gamma\mathbf{g} \\ \gamma\mathbf{g}^T & \gamma^2 \end{bmatrix} \right\} \mathbf{R} = \mathbf{R}\mathbf{\Lambda}'_{k+1}, \tag{8}$$

2. otherwise,

$$\left\{ \frac{N}{N+1} \mathbf{\Lambda}_k + \frac{N}{(N+1)^2} \mathbf{g}\mathbf{g}^T \right\} \mathbf{R} = \mathbf{R}\mathbf{\Lambda}'_k. \tag{9}$$

Here, $\gamma = \hat{\mathbf{h}}^T(\mathbf{y} - \bar{\mathbf{x}})$ and $\mathbf{0}$ is a k-dimensional zero vector; $\mathbf{\Lambda}'_{k+1}$ and $\mathbf{\Lambda}'_k$ are the new eigenvalue matrices whose diagonal elements correspond to k and $k+1$ eigenvalues, respectively. Using the solution $\mathbf{R}$, the new eigenvector matrix $\mathbf{U}'_k$ or $\mathbf{U}'_{k+1}$ is calculated from Eq. (6) or (7).

3. Chunk Incremental Principal Component Analysis (Chunk IPCA)

3.1 Learning assumptions and outline of chunk IPCA algorithm

IPCA can be applied to one training sample at a time, and the intermediate eigenproblem in Eq. (8) or (9) must be solved for each sample even though a chunk of samples are provided to learn at a time. Obviously this is inefficient from a computational point of view, and the learning may get stuck in a deadlock if a large chunk of training samples is given to learn in a short term; that is, the next chunk of training samples could come before the learning is completed if it takes long time for updating an eigenspace.

To overcome this problem, the original IPCA is extended so that the eigenspace model Ω can be updated with a chunk of training samples in a single operation (Ozawa et al., 2008). This extended algorithm is called *Chunk IPCA*.

Assume again that *N* training samples $\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\} \in R^{n \times N}$ have been given so far and an eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$ was obtained from these samples. Now, a chunk of L training samples $\mathbf{Y} = \{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(L)}\} \in R^{n \times L}$ are presented to the system. Let the updated eigenspace model be $\Omega' = (\bar{\mathbf{x}}', \mathbf{U}'_{k'}, \mathbf{\Lambda}'_{k'}, N+L)$. The mean vector $\bar{\mathbf{x}}'$ in Ω' can be updated without the past training samples $\mathbf{X}$ as follows:

$$\bar{\mathbf{x}}' = \frac{1}{N+L}\left(N\,\bar{\mathbf{x}} + L\,\bar{\mathbf{y}}\right)\cdot \tag{10}$$

The problem is how to update $\mathbf{U}'_k$ and $\mathbf{\Lambda}'_k$ in Ω'.

As shown in the derivation of IPCA, the update of $\mathbf{U}'_k$ and $\mathbf{\Lambda}'_k$ is reduced to solving an intermediate eigenproblem, which is derived from an eigenvalue problem using a covariance matrix. Basically, the intermediate eigenproblem for Chunk IPCA can also be derived in the same way as shown in the derivation of IPCA (Hall & Martin, 1998). However, we should note that the dimensionality k' of the updated eigenspace could range from k to $k+L$ depending on the given chunk data $\mathbf{Y}$. To avoid constructing a redundant eigenspace, the smallest k' should be selected under the condition that the accumulation ratio is over a designated threshold. Thus, an additional operation to select a smallest set of eigen-axes is newly introduced into Chunk IPCA. Once the eigen-axes to be augmented are determined, all the eigen-axes should be rotated to adapt to the variation in the data distribution. This operation is basically the same as in IPCA.

In summary, there are three main operations in Chunk IPCA: (1) mean vector update, (2) eigenspace augmentation with the selection of a smallest set of eigen-axes, and (3) eigenspace rotation. The first operation is carried out by Eq. (10). The latter two operations are explained below.

3.2 Eigenspace augmentation

In Chunk IPCA, the number of eigen-axes to be augmented is determined by finding the minimum k such that $A(\mathbf{U}_k) \geq \theta$ holds where θ is a threshold between 0 and 1. To introduce this criterion, we need to modify the update equation of Eq. (3) such that the accumulation ratio can be updated incrementally for a chunk of *L* samples. In Chunk IPCA, we need to consider two types of accumulation ratios. One is the accumulation ratio for a *k*-dimensional eigenspace spanned by $\mathbf{U}'_k = \mathbf{U}_k\mathbf{R}$ where $\mathbf{R}$ is a rotation matrix which is calculated from the intermediate eigenvalue problem described later. The other is that for a (*k*+*l*)-dimensional augmented eigenspace spanned by $\mathbf{U}'_{k+l} = [\mathbf{U}_k, \mathbf{H}_l]\mathbf{R}$ where $\mathbf{H}_l$ is a set of *l* augmented eigen-axes. The former is used for checking if the current *k*-dimensional eigenspace should be augmented or not. The latter one is used for checking if further eigen-axes are needed for the (*k*+*l*)-dimensional augmented eigenspace.

The new accumulation ratio $A'(\mathbf{U}'_k)$ and $A'(\mathbf{U}'_{k+l})$ are calculated as follows (Ozawa et al., 2008):

$$A'(\mathbf{U}'_k) = \frac{\sum_{i=1}^{k}\lambda_i + \frac{L}{N+L}\|\bar{\mathbf{g}}\|^2 + \frac{1}{N}\sum_{j=1}^{L}\|\mathbf{g}''_j\|^2}{\sum_{i=1}^{n}\lambda_i + \frac{L}{N+L}\|\bar{\boldsymbol{\mu}}\|^2 + \frac{1}{N}\sum_{j=1}^{L}\|\boldsymbol{\mu}''_j\|^2} \tag{11}$$

$$A'(\mathbf{U}'_{k+l}) \approx \frac{\sum_{i=1}^{k}\lambda_i + \frac{L}{N+L}\left\|\begin{matrix}\bar{\mathbf{g}}\\ \bar{\boldsymbol{\gamma}}\end{matrix}\right\|^2 + \frac{1}{N}\sum_{j=1}^{L}\left\|\begin{matrix}\mathbf{g}''_j\\ \boldsymbol{\gamma}''_j\end{matrix}\right\|^2}{\sum_{i=1}^{n}\lambda_i + \frac{L}{N+L}\|\bar{\boldsymbol{\mu}}\|^2 + \frac{1}{N}\sum_{j=1}^{L}\|\boldsymbol{\mu}''_j\|^2} \tag{12}$$

where $\bar{\mathbf{g}} = \mathbf{U}_k^T(\bar{\mathbf{y}} - \bar{\mathbf{x}})$, $\mathbf{g}_i'' = \mathbf{U}_k^T(\mathbf{y}^{(i)} - \bar{\mathbf{y}})$, $\bar{\boldsymbol{\mu}} = \bar{\mathbf{x}} - \bar{\mathbf{y}}$, $\boldsymbol{\mu}_j'' = \mathbf{y}^{(j)} - \bar{\mathbf{y}}$, $\bar{\boldsymbol{\gamma}} = \mathbf{H}_l^T(\bar{\mathbf{y}} - \bar{\mathbf{x}})$, and $\boldsymbol{\gamma}_i'' = \mathbf{H}_l^T(\mathbf{y}^{(i)} - \bar{\mathbf{y}})$. To update $A'(\mathbf{U}_k')$, the summation of eigenvalues λ_i $(i = 1,\ldots,n)$ is required, and this summation can be held by accumulating the power of training samples (Ozawa et al., 2004). Hence, the individual eigenvalues λ_i $(i = k+1,\ldots,n)$ are not necessary for this update.

As seen from Eqs. (11) and (12), we need no past sample $\mathbf{x}^{(j)}$ and no rotation matrix $\mathbf{R}$ to update the accumulation ratio. Therefore, this accumulation ratio is updated with the following information: a chunk of given training samples $\mathbf{Y} = \{\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(L)}\}$, the current eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$, the summation of eigenvalues $\sum_{j=1}^{n} \lambda_i$, and a set of augmented eigen-axes $\mathbf{H}_l$ which are obtained through the procedure described next.

In IPCA, a new eigen-axis is obtained to be orthogonalized to the existing eigenvectors (i.e., column vectors of $\mathbf{U}_k$). A straightforward way to obtain new eigen-axes is to apply Gram-Schmidt orthogonalization to a chunk of given training samples (Hall et al., 2000). If the training samples are represented by $\tilde{L}$ linearly independent vectors, the maximum number of augmented eigen-axes is $\tilde{L}$. However, the subspace spanned by all of the $\tilde{L}$ eigen-axes is redundant in general. Therefore, we should find a smallest set of eigen-axes without losing essential information on $\mathbf{Y}$.

The problem of finding an optimal set of eigen-axes is stated below.

Find the smallest set of eigen-axes $\mathbf{H}^* = \{\mathbf{h}_1,\ldots,\mathbf{h}_{l^*}\}$ *for the current eigenspace model* $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$ *without keeping the past training samples* $\mathbf{X}$ *such that the accumulation ratio* $A'(\mathbf{U}_{k+l^*}')$ *of all the given training samples* $\{\mathbf{X}, \mathbf{Y}\}$ *is larger than a threshold* θ.

Assume that we have a candidate set of augmented eigen-axes $\mathbf{H}_l = \{\mathbf{h}_1,\ldots,\mathbf{h}_l\}$. Since the denominator of Eq. (12) is constant once the mean vector $\bar{\mathbf{y}}$ is calculated, the increment of the accumulation ratio from $A'(\mathbf{U}_k')$ to $A'(\mathbf{U}_{k+l}')$ is determined by the numerator terms. Thus, let us define the following difference $\Delta\tilde{A}'(\mathbf{U}_{k+l}')$ of the numerator terms between $A'(\mathbf{U}_k')$ and $A'(\mathbf{U}_{k+l}')$:

$$\Delta\tilde{A}'(\mathbf{U}_{k+l}') = \frac{L}{N+L}\left\|\mathbf{H}_l^T(\bar{\mathbf{x}} - \bar{\mathbf{y}})\right\|^2 + \frac{1}{N}\sum_{j=1}^{L}\left\|\mathbf{H}_l^T(\mathbf{y}^{(j)} - \bar{\mathbf{y}})\right\|^2 \overset{\text{def}}{=} \sum_{i=1}^{l}\Delta\tilde{A}_i' \tag{13}$$

where

$$\Delta\tilde{A}_i' = \frac{L}{N+L}\left\{\mathbf{h}_i^T(\bar{\mathbf{x}} - \bar{\mathbf{y}})\right\}^2 + \frac{1}{N}\sum_{j=1}^{L}\left\{\mathbf{h}_i^T(\mathbf{y}^{(j)} - \bar{\mathbf{y}})\right\}^2. \tag{14}$$

Equation (13) means that the increments of the accumulation ratio is determined by the linear sum of $\Delta\tilde{A}_i'$. Therefore, to find the smallest set of eigen-axes, first we find $\mathbf{h}_i$ with the largest $\Delta\tilde{A}_i'$, and put it into the set of augmented eigen-axes $\mathbf{H}_l$ (i.e., $l = 1$ and $\mathbf{H}_l = \mathbf{h}_i$). Then, check if the accumulation ratio $A'(\mathbf{U}_{k+l}')$ in Eq. (12) becomes larger than the threshold θ. If not, select $\mathbf{h}_i$ with the second largest $\Delta\tilde{A}_i'$, and the same procedure is repeated until $A'(\mathbf{U}_{k+l}') \ge \theta$ satisfies. This type of greedy algorithm makes the selection of eigen-axes very simple. The algorithm of the eigen-axis selection is summarized in **Algorithm 1**.

```
Algorithm 1: Eigen-axis Selection
Input: Eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$, $L$ training samples $\mathbf{Y} = \{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(L)}\}$,
    the summation of eigenvalues $\sum_{j=1}^{n} \lambda_i$, and threshold $\theta$ of accumulation ratio.
  Calculate the mean vector $\bar{\mathbf{y}}$ in Eq. (10).
  Calculate the accumulation ratio $A'(\mathbf{U}'_k)$ in Eq. (11).
  If $A'(\mathbf{U}'_k) \geq \theta$ then
      Terminate this algorithm.
  End if
  For $i = 1$ to $L$ do
      Obtain the following residue vectors $\mathbf{h}_i$ using the ith sample $\mathbf{y}^{(i)}$:
          $\mathbf{h}_i = \frac{\mathbf{r}_i}{\|\mathbf{r}_i\|}$ where $\mathbf{r}_i = (\mathbf{y}^{(i)} - \bar{\mathbf{x}}) - \mathbf{U}_k \mathbf{U}_k^T (\mathbf{y}^{(i)} - \bar{\mathbf{x}})$.
  End for
  Define an index set $\Gamma$ of $\mathbf{h}_i$, and initialize $\mathbf{H}$ and $l$.
  Loop
      Find the residue vector $\mathbf{h}_{i'}$ that gives the largest $\Delta\tilde{A}'_i$ in Eq. (14): $i' = \arg\max_{i \in \Gamma} \Delta\tilde{A}'_i$.
      $\mathbf{H} \leftarrow [\mathbf{H}, \mathbf{h}_{i'}]$, $l \leftarrow l + 1$, and $\Gamma \leftarrow \Gamma - i'$.
      Calculate the updated accumulation ratio $A'(\mathbf{U}'_{k+l})$ in Eq. (12).
      If $\Gamma$ is empty or $A'(\mathbf{U}'_{k+l}) > \theta$ then
          Terminate this algorithm.
      End if
  End loop
Output: Augmented eigen-axes: $\mathbf{H}_l = \{\mathbf{h}_1, \ldots, \mathbf{h}_l\}$.
```

3.3 Eigenspace rotation

Next, let us derive the update equations for $\mathbf{U}_k$ and $\mathbf{\Lambda}_k$. Suppose that l eigen-axes are augmented when a chunk of L training samples $\mathbf{Y}$ is provided; that is, the eigenspace dimensions are increased by l. Let us denote the augmented eigen-axes as follows:

$$\mathbf{H}_l = \{\mathbf{h}_1, \ldots, \mathbf{h}_l\} \in R^{n \times l}, \quad 0 \leq l \leq L. \tag{15}$$

Then, the updated eigenvector matrix $\mathbf{U}'_{k+l}$ is represented by

$$\mathbf{U}'_{k+l} = [\mathbf{U}_k, \mathbf{H}_l]\mathbf{R} \tag{16}$$

where $\mathbf{R}$ is a rotation matrix. It has been shown that $\mathbf{R}$ is obtained by solving the following intermediate eigenproblem (Ozawa et al., 2008):

1. If there are new eigen-axes to be added (i.e., $l \neq 0$),

$$\left\{ \frac{N}{N+L} \begin{bmatrix} \mathbf{\Lambda}_k & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \frac{NL^2}{(N+L)^3} \begin{bmatrix} \bar{\mathbf{g}}\bar{\mathbf{g}}^T & \bar{\mathbf{g}}\bar{\boldsymbol{\gamma}}^T \\ \bar{\boldsymbol{\gamma}}\bar{\mathbf{g}}^T & \bar{\boldsymbol{\gamma}}\bar{\boldsymbol{\gamma}}^T \end{bmatrix} + \frac{N^2}{(N+L)^3} \sum_{i=1}^{L} \begin{bmatrix} \mathbf{g}'_i \mathbf{g}'^T_i & \mathbf{g}'_i \boldsymbol{\gamma}'^T_i \\ \boldsymbol{\gamma}'_i \mathbf{g}'^T_i & \boldsymbol{\gamma}'_i \boldsymbol{\gamma}'^T_i \end{bmatrix}\right.$$

$$\frac{L(L+2N)}{(N+L)^3}\sum_{i=1}^{L}\begin{bmatrix}\mathbf{g}_i''\mathbf{g}_i''^T & \mathbf{g}_i''\boldsymbol{\gamma}_i''^T \\ \boldsymbol{\gamma}_i''\mathbf{g}_i''^T & \boldsymbol{\gamma}_i''\boldsymbol{\gamma}_i''^T\end{bmatrix}\Bigg\}\mathbf{R}=\mathbf{R}\boldsymbol{\Lambda}'_{k+l}, \tag{17}$$

2. otherwise,

$$\left\{\frac{N}{N+L}\boldsymbol{\Lambda}_k+\frac{NL^2}{(N+L)^3}\overline{\mathbf{g}}\overline{\mathbf{g}}^T+\frac{N^2}{(N+L)^3}\sum_{i=1}^{L}\mathbf{g}_i'\mathbf{g}_i'^T+\frac{L(L+2N)}{(N+L)^3}\sum_{i=1}^{L}\mathbf{g}_i''\mathbf{g}_i''^T\right\}\mathbf{R}=\mathbf{R}\boldsymbol{\Lambda}'_k. \tag{18}$$

Here, $\mathbf{g}_i' = \mathbf{U}_k^T\left(\mathbf{y}^{(i)}-\overline{\mathbf{x}}\right)$ and $\boldsymbol{\gamma}_i' = \mathbf{H}_l^T\left(\mathbf{y}^{(i)}-\overline{\mathbf{x}}\right)$. As seen from Eqs. (17) and (18), the rotation matrix $\mathbf{R}$ and the eigenvalue matrix $\boldsymbol{\Lambda}'_{k+l}$ correspond to the eigenvectors and eigenvalues of the intermediate eigenproblem, respectively. Once $\mathbf{R}$ is obtained, the corresponding new eigenvector matrix $\mathbf{U}'_{k+l}$ is given by Eq. (16).

The overall algorithm of Chunk IPCA is summarized in **Algorithm 2**.

Algorithm 2: ***Chunk IPCA***

```
Input: Eigenspace model $\Omega = (\overline{\mathbf{x}}, \mathbf{U}_k, \boldsymbol{\Lambda}_k, N)$ and $L$ training samples $\mathbf{Y} = \{\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(L)}\}$.
    Perform Eigen-axis Selection (Algorithm 1) to obtain augmented eigen-axes $\mathbf{H}_l$.
    Solve the intermediate eigenproblem in Eq. (17) or (18) to obtain a rotation matrix $\mathbf{R}$ and
    an updated eigenvalue matrix $\boldsymbol{\Lambda}'_{k+l}$.
    Obtain the updated the eigenvector matrix $\mathbf{U}'_{k+l}$ in Eq. (16).
    Update the mean vector $\overline{\mathbf{x}}'$ in Eq. (10).
Output: Updated eigenspace model $\Omega' = (\overline{\mathbf{x}}', \mathbf{U}'_{k+l}, \boldsymbol{\Lambda}'_{k+l}, N+L)$.
```

3.4 Training of initial eigenspace

Assume that a set of initial training samples $D_0 = \{(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) \mid i = 1,\ldots,N\}$ is given before incremental learning gets started. To obtain an initial eigenspace model $\Omega = (\overline{\mathbf{x}}, \mathbf{U}_k, \boldsymbol{\Lambda}_k, N)$, the conventional PCA is applied to D_0 and the smallest dimensionality k of the eigenspace is determined such that the accumulation ratio is larger than θ. Since a proper θ is usually unknown and often depends on training data, the cross-validation technique can be applied to determining θ (Ozawa et al., 2008). However, for the sake of simplicity, let us assume here that a proper θ is given in advance. The algorithm of the initial training is shown in **Algorithm 3**.

Algorithm 3: ***Training of Initial Eigenspace***

```
Input: Initial training set $D_0 = \{(\mathbf{x}^{(i)}, z^{(i)}) \mid i = 1,\ldots,N\}$ and threshold $\theta$.
    Calculate the mean vector $\overline{\mathbf{x}}$ of $\mathbf{x}^{(i)} \in D_0$.
    Apply PCA to $D_0$ and obtain the eigenvectors $\mathbf{U}_n = \{\mathbf{u}_1,\ldots,\mathbf{u}_n\}$ whose eigenvalues
    $\Lambda_n = \mathrm{diag}\{\lambda_1,\ldots,\lambda_n\}$ are sorted in decreasing order.
    Find the smallest k such that the following condition holds: $A(\mathbf{U}_k) \geq \theta$.
Output: Eigenspace model $\Omega = (\overline{\mathbf{x}}, \mathbf{U}_k, \boldsymbol{\Lambda}_k, N)$.
```

4. Incremental learning of classifier

4.1 Resource Allocating Network (RAN)

Resource Allocating Network (RAN) (Platt, 1991) is an extended version of RBF networks. When the training gets started, the number of hidden units is set to one; hence, RAN has simple approximation ability at first. As the training proceeds, the approximation ability of RAN is developed with the increase of training samples by allocating additional hidden units.

Figure 1 illustrates the structure of RAN. The output of hidden units $\mathbf{y} = \{y_1, \ldots, y_J\}^T$ is calculated based on the distance between an input $\mathbf{x} = \{x_1, \ldots, x_I\}^T$ and center vector of the jth hidden unit $\mathbf{c}_j = \{c_{j1}, \ldots, c_{jI}\}^T$:

$$y_j = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_j\|^2}{\sigma_j^2}\right) \qquad \text{for } j = 1, \ldots, J \tag{19}$$

where I and J are the numbers of input units and hidden units, respectively, and σ_j^2 is a variance of the jth radial basis. The network output $\mathbf{z} = \{z_1, \ldots, z_K\}^T$ is calculated as follows:

$$z_k = \sum_{k=1}^{K} w_{kj} y_j + \gamma_k \qquad \text{for } k = 1, \ldots, K \tag{20}$$

where K is the number of output units, w_{kj} is a connection weight from the jth hidden unit to the kth output unit, and γ_k is a bias of the kth output unit.

When a training sample $(\mathbf{x}, \mathbf{d})$ is given, the network output is calculated based on Eqs. (19) and (20), and the root mean square error $E = \|\mathbf{d}-\mathbf{z}\|$ between the output $\mathbf{z}$ and target $\mathbf{d}$ for the input $\mathbf{x}$ is evaluated. Depending on E, either of the following operations is carried out:

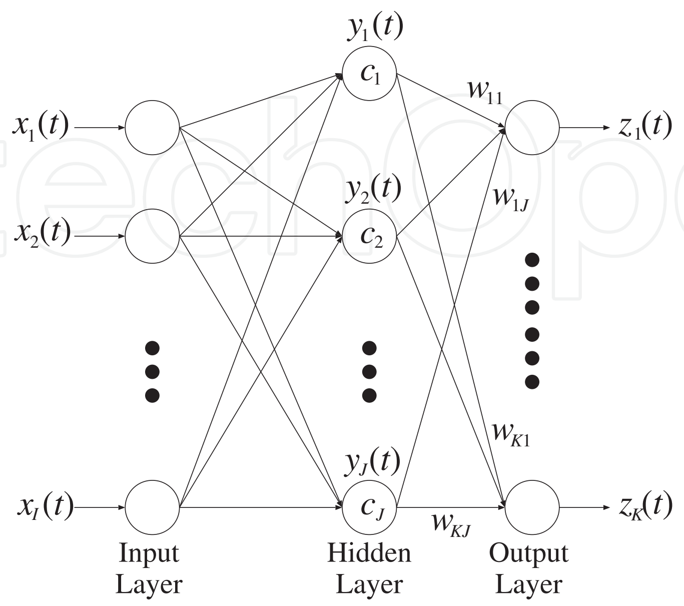


Fig. 1. Structure of Resource Allocating Network (RAN).

1. If E is larger than a positive constant ε and the distance between an input $\mathbf{x}$ and its nearest center vector $\mathbf{c}^*$ is larger than a positive value $\delta(t)$ (i.e., $E > \varepsilon$ and $\|\mathbf{x} - \mathbf{c}^*\| > \delta(t)$), a hidden unit is added (i.e., $J \leftarrow J + 1$). Then, the network parameters for the *J*th hidden unit (center vector $\mathbf{c}_J$, connection weights $\mathbf{w}_J = \{w_{1J}, \ldots, w_{KJ}\}$, and variance σ_j^2) are set to the following values: $\mathbf{c}_J = \mathbf{x}_p$, $\mathbf{w}_J = \mathbf{d} - \mathbf{z}$, and $\sigma_J = \kappa\|\mathbf{x} - \mathbf{c}^*\|$ where κ is a positive constant. $\delta(t)$ is decreased with time *t* as follows:

$$\delta(t) = \max\left[\delta_{\max} \exp\left(-\frac{t}{\tau}\right), \delta_{\min}\right] \tag{21}$$

where τ is a decay constant, $\delta_{\max}$ and $\delta_{\min}$ are maximum and minimum values of $\delta(t)$, respectively.
2. Otherwise, the network parameters are updated as follows:

$$w_{kj}^{NEW} = w_{kj}^{OLD} + \alpha\, e_k\, y_{pj} \tag{22}$$

$$c_{ji}^{NEW} = c_{ji}^{OLD} + \frac{\alpha}{\sigma_j^2}(x_{pi} - c_{ji})\, y_{pj} \sum_{k=1}^{K} e_k w_{kj} \tag{23}$$

$$\gamma_k^{NEW} = \gamma_k^{OLD} + \alpha\, e_k \tag{24}$$

where $e_k = d_k - z_k$ and α is a positive learning ratio.

Although the approximation ability is developed by allocating hidden units, the interference cannot be suppressed completely only by this mechanism. In the next section, we present an extended model of RAN in which a mechanism of suppressing the interference is explicitly introduced.

4.2 Resource allocating network with long-term memory

RAN is a neural network with spatially localised basis functions; hence it is expected that the catastrophic interference (Carpenter & Grossberg, 1988) is alleviated to some extent. However, since no explicit mechanism of suppressing the interference is introduced, the insufficient suppression might cause serious unlearning over the long run.

To suppress unexpected forgetting in RAN, Resource Allocating Network with Long-Term Memory (RAN-LTM) (Kobayashi et al., 2001) has been proposed. Figure 2 shows the architecture of RAN-LTM which consists of two modules: RAN and an external memory called *Long-Term Memory* (LTM). Representative input-output pairs are extracted from the mapping function acquired in RAN and they are stored in LTM. These pairs are called *memory items* and some of them are retrieved from LTM to learn with training samples. In the learning algorithm, a memory item is created when a hidden unit is allocated; that is, an RBF center and the corresponding output are stored as a memory item.

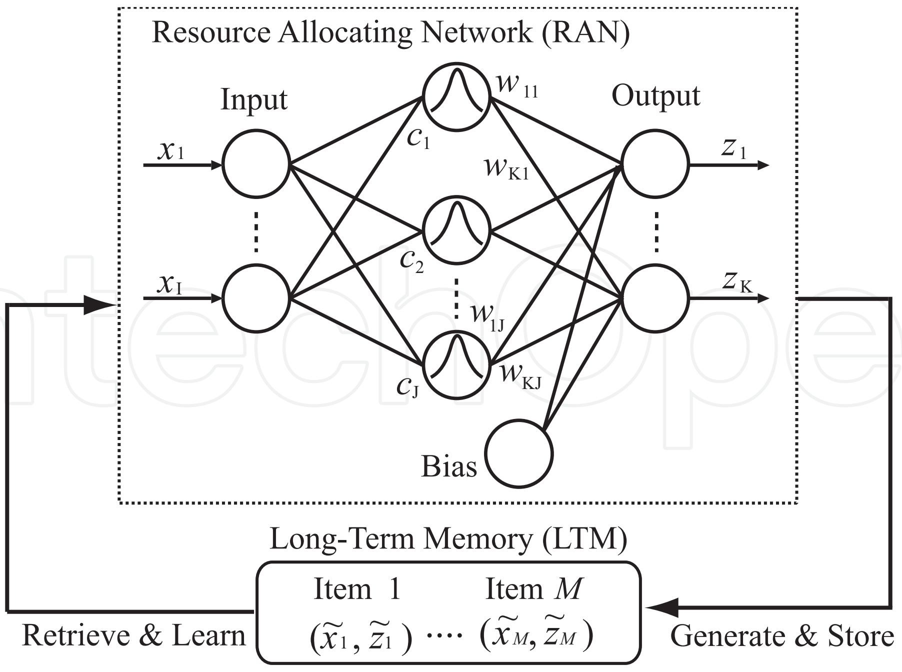


Fig. 2. Architecture of RAN-LTM.

The learning algorithm of RAN-LTM is divided into two phases: the allocation of hidden units (i.e., incremental selection of RBF centers) and the calculation of connection weights between hidden and output units. The procedure in the former phase is the same as that in the original RAN, except that memory items are created at the same time. Once hidden units are allocated, the centers are fixed afterwards. Therefore, the connection weights $\mathbf{W} = \{w_{jk}\}$ are only parameters that are updated based on the output errors. To minimize the errors based on the least squares method, it is well known that the following linear equations should be solved (Haykin, 1999):

$$\mathbf{\Phi W} = \mathbf{D} \tag{25}$$

where $\mathbf{D}$ is a matrix whose column vectors correspond to the target outputs and $\mathbf{\Phi}$ is a matrix of hidden outputs. Suppose that a new training sample $(\mathbf{x}, \mathbf{d})$ is given and M memory items $(\tilde{\mathbf{x}}^{(m)}, \tilde{\mathbf{z}}^{(m)})$ $(m = 1, \ldots, M)$ have already been created, then in the simplest version of RAN-LTM (Ozawa et al., 2005) the target matrix $\mathbf{D}$ are formed as follows: $\mathbf{D} = \{\mathbf{d}, \tilde{\mathbf{z}}^{(1)}, \ldots, \tilde{\mathbf{z}}^{(M)}\}^T$. Furthermore, $\mathbf{\Phi} = \{\phi_{ij}\}$ $(i = 1, \ldots, M+1)$ is calculated from the training sample and memory items as follows:

$$\phi_{1j} = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{\sigma_j^2}\right), \quad \phi_{i+1,j} = \exp\left(-\frac{\|\tilde{\mathbf{x}}_i - \mathbf{c}_j\|^2}{\sigma_j^2}\right) \quad (j = 1, \ldots, J; i = 1, \ldots, M). \tag{26}$$

Singular Value Decomposition (SVD) can be used for solving $\mathbf{W}$ in Eq. (25). The learning algorithm of RAN-LTM is summarized in **Algorithm 4**.

Algorithm 4: ***Learning of RAN-LTM***

```
Input: RAN-LTM and L training samples $(\mathbf{x}^{(p)},\mathbf{d}^{(p)})$ $(p = 1,\ldots,L)$.
  For p=1 to L do
    Find the nearest center $\mathbf{c}^*$ to the pth input $\mathbf{x}^{(p)}$.
    Calculate the output error E.
    If $E > \varepsilon$ and $\|\mathbf{x}^{(p)} - \mathbf{c}^*\| > \delta$ then
      Increment the numbers of hidden units and memory items: $J \leftarrow J+1$ and
      $M \leftarrow M+1$.
      Add a hidden unit and create a memory item $(\tilde{\mathbf{x}}^{(M)},\tilde{\mathbf{z}}^{(M)})$, and set the values as
      follows: $\mathbf{c}_J = \mathbf{x}^{(p)}$, $\mathbf{w}_J = \mathbf{d}^{(p)} - \mathbf{z}$, $\tilde{\mathbf{x}}^{(M)} = \mathbf{x}^{(p)}$, $\tilde{\mathbf{z}}^{(M)} = \mathbf{d}^{(p)}$.
    Else
      Calculate hidden outputs for the training sample $(\mathbf{x}^{(p)},\mathbf{d}^{(p)})$ and M memory
      items $(\tilde{\mathbf{x}}^{(M)},\tilde{\mathbf{z}}^{(M)})$ $(m = 1,\ldots,M)$ using Eq. (26).
      Define the activity matrix $\mathbf{\Phi}$.
      Decompose $\mathbf{\Phi}$ with SVD as follows: $\mathbf{\Phi} = \mathbf{UHV}^T$ where U and V are orthogonal
      matrices, and H is a diagonal matrix.
      Calculate the weight matrix as follows: $\mathbf{W} = \mathbf{VH}^{-1}\mathbf{U}^T\mathbf{D}$.
      Give the input $\mathbf{x}^{(p)}$ to RAN-LTM again, and calculate the output error E.
      If $E > \varepsilon$ then
        $J \leftarrow J+1$ and $M \leftarrow M+1$.
        Add a hidden unit and create a memory item $(\tilde{\mathbf{x}}^{(M)},\tilde{\mathbf{z}}^{(M)})$ and set the values
        as follows: $\mathbf{c}_J = \mathbf{x}^{(p)}$, $\mathbf{w}_J = \mathbf{d}^{(p)} - \mathbf{z}$, $\tilde{\mathbf{x}}^{(M)} = \mathbf{x}^{(p)}$, $\tilde{\mathbf{z}}^{(M)} = \mathbf{d}^{(p)}$.
      End if
    End if
  End For
Output: RAN-LTM
```

5. Face recognition system

Figure 3 shows the overall process in the proposed face recognition system. As seen from Fig. 3, the proposed system mainly consists of the following four sections: face detection, face recognition, face image verification, and incremental learning. The information processing in each section is explained below.

5.1 Face detection

In the face detection part, we adopt a conventional algorithm that consists of two operations: face localization and face feature detection. Figure 4 shows an example of the face detection process.

Facial regions are first localized in an input image by using the skin color information and horizontal edges. The skin color information is obtained by projecting every pixel in the input image to a skin-color axis. This axis was obtained from Japanese skin images in advance. In our preliminary experiment, the face localization works very well with 99% accuracy for a Japanese face database.

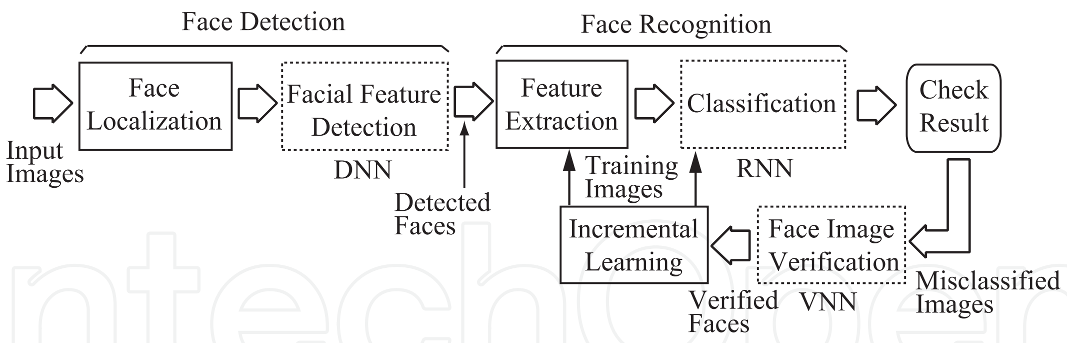


Fig. 3. The processing flow in the face recognition system. DNN and VNN are implemented by RBF networks, while RNN is implemented by RAN-LTM that could learn misclassified face images incrementally. In the feature extraction part, an eigen-space model is incrementally updated for misclassified face images by using Chunk IPCA.

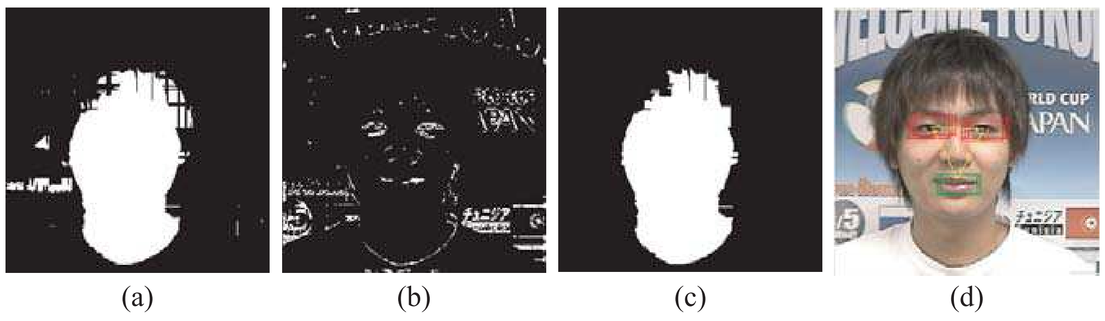


Fig. 4. The process of face detection: (a) an output of skin colour filter and (b) an output of edge filter, (c) a face region extracted from the two filter outputs in (a) and (b), and (d) the final result of the face detection part. Only one face was detected in this case.

After the face localization, three types of facial features (eye, nose, mouth) are searched for within the localized regions through raster operations. In each raster operation, a small sub-image is separated from a localized region. Then, the eigen-features of the sub-image are given to *Detection Neural Network* (DNN) to verify if it corresponds to one of the facial features. The eigenspace model for the face detection part was obtained by applying PCA to a large image dataset of human eye, nose, and mouth in advance. This dataset is also used for the training of DNN.
After all raster operations are done, face candidates are generated by combining the identified facial features. All combinations of three facial features are checked if they satisfy a predefined facial geometric constraint. A combination of three features found on the geometric template qualifies as a face candidate. The output of the face detection part is the center position of a face candidate.
The overall process in the face detection part is summarized in **Algorithm 5**.

5.2 Face recognition and face image verification

In the face recognition part, all the detected face candidates are classified into registered or non-registered faces. This part consists of the following two operations: feature extraction and classification (see Fig. 3). In the feature extraction part, the eigenface approach is adopted here to find informative face features. A face candidate is first

Algorithm 5: ***Face Detection***

```
Input: Eigenspace model for the face detection part, DNN, and input image
  For every pixel in the input image do
    Calculate the projection (called skin-color feature) of the RGB values of a pixel to a
    skin-color axis.
    If the skin-color feature exists within a designated domain then
      Define the pixel as a skin region.
    Else
      Define it as a non-skin region.
    End if
  End for
  Remove grainy skin regions using a MinMax filter.
  Identify the smallest rectangular region surrounding all the skin regions.
  Redefine this rectangle as a skin region.
  Perform the edge extraction by applying a MaxMin filter and a Sobel horizontal filter to
  the skin region.
  Search for facial sub-regions including both the skin-color and edge information.
  Extract the sub-regions from the input image.
  For every facial sub-region do
    Set the starting point to the upper left corner of the sub-region.
    Repeat
      Extract a 40x40-pixel sub-image.
      Obtain eigen-features of the sub-image from the eigenspace model.
      Provide the eigen-features to DNN as inputs to identify one of the following facial
      features: eye, nose, or mouth.
      Move the starting point in a raster way.
    Until the starting point reaches to the lower right corner
    Find face candidates satisfying the predefined facial geometrical constraints.
    Obtain the center position of the face candidates.
  End for
Output: Center positions of face candidates.
```

projected to the eigen-axes to calculate eigen-features, and they are given to RAN-LTM called *Recognition Neural Network* (RNN). Then, the classification is carried out based on the outputs of RNN.

If the recognition is correct, RNN should be unchanged. Otherwise, RNN must be trained with the misclassified images to be classified correctly afterward. The misclassified images are collected to carry out incremental learning for both feature extraction part and classifier (RNN). Since the perfect face detection cannot be always ensured, there is a possibility that non-face images happen to be mixed with the misclassified face images. Apparently the training of these non-face images will deteriorate the recognition performance of RNN. Thus, another RBF network called *Verification Neural Network* (VNN) is introduced into this part in order to filter non-face images out.

The procedures of face recognition and verification are summarized in **Algorithm 6**.

Algorithm 6: ***Recognition and Verification of Face***

```
Input: Eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$, RNN, VNN, input image, and center
       positions of face candidates.
  For every face candidate do
    Extract a 90x90 sub-image from the input image around the center potition of the face
    candidate.
    Obtain eigen-features by projecting the sub-image to the eigen-axes $\mathbf{U}_k$.
    Provide the eigen-features to RNN to classify the input face image.
    If misclassification occurs then
      Provide the eigen-features to VNN to verify if it is face or non-face.
      If it is verified as a face then
        Keep the sub-image with the correct class label.
      End if
    End if
  End for
Output: Misclassified face sub-images and the class labels
```

5.3 Incremental learning

Since face images of a person can vary depending on various temporal and special factors, the classifier should have high adaptability to those variations. In addition, the useful features might also change over time for the same reason. Therefore, in the face recognition part, the incremental learning should be conducted for both feature extraction part and classifier.

The incremental learning of the feature extraction part is easily carried out by applying Chunk IPCA to misclassified face images. However, the incremental learning of classification part (RNN) cannot be done in a straightforward manner due to the learning of the eigenspace model. That is to say, the inputs of RNN would dynamically change not only in their values but also in the number of input variables due to the eigen-axis rotation and the dimensional augmentation in Chunk IPCA. Therefore, to make RNN adapt to the change in the feature extraction part, not only the network parameters (i.e., weights, RBF centers, etc.) but also its network structure have to be modified.

Under one-pass incremental learning circumstances, this reconstruction of RNN is not easily done without unexpected forgetting of the mapping function that has been acquired so far. If the original RAN is adopted as a classifier, there is no way of retraining the neural classifier to ensure that all previously trained samples can be correctly classified again after the update of the feature space because the past training samples are already thrown away. This can be solved if a minimum number of representative samples are properly selected and used for retraining the classifier. RAN-LTM is suitable for this purpose.

To implement this idea, we need to devise an efficient way to adapt the memory items in RAN-LTM to the updated eigenspace. Let an input vector of the mth memory item be $\tilde{\mathbf{x}}^{(m)} \in R^I$ and let its target vector be $\tilde{\mathbf{z}}^{(m)} \in R^K$: $\left\{\tilde{\mathbf{x}}^{(m)}, \tilde{\mathbf{z}}^{(m)}\right\}$ $(m = 1, \dots, M)$. Furthermore, let the original vector associated with $\tilde{\mathbf{x}}^{(m)}$ in the input space be $\mathbf{x}^{(m)} \in R^n$. The two input vectors have the following relation: $\tilde{\mathbf{x}}^{(m)} = \mathbf{U}^T(\mathbf{x}^{(m)} - \bar{\mathbf{x}})$. Now, assume that a new eigenspace model $\Omega' = (\bar{\mathbf{x}}', \mathbf{U}'_{k+l}, \mathbf{\Lambda}'_{k+l}, N + L)$ is obtained by applying Chunk IPCA to a chunk of L training samples $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_L)$. Then, the updated memory item $\tilde{\mathbf{x}}^{(m)\prime}$ should satisfy the following equation:

$$\tilde{\mathbf{x}}^{(m)\prime} = \mathbf{U}_{k+l}^{\prime T}(\mathbf{x}^{(m)} - \bar{\mathbf{x}}') = \mathbf{U}_{k+l}^{\prime T}(\mathbf{x}^{(m)} - \bar{\mathbf{x}}) + \frac{1}{N+1}\mathbf{U}_{k+l}^{\prime T}(\bar{\mathbf{x}} - \bar{\mathbf{y}}) \tag{27}$$

where $\bar{\mathbf{x}}'$ and $\mathbf{U}'_{k+l}$ are given by Eqs. (10) and (16), respectively. The second term in the right-hand side of Eq. (27) is easily calculated. To calculate the first term exactly, however, the information on $\mathbf{x}^{(m)}$, which is not usually kept in the system for reasons of memory efficiency, is needed. Here, let us consider the approximation to the first term without keeping $\mathbf{x}^{(m)}$.

Assume that l of eigen-axes $\mathbf{H}_l$ is augmented. Substituting Eq. (16) into the first term on the right-hand side of Eq. (27), then the first term is reduced to

$$\mathbf{U}_{k+l}^{\prime T}(\mathbf{x}^{(m)} - \bar{\mathbf{x}}) = \mathbf{R}^T \begin{bmatrix} \tilde{\mathbf{x}}^{(m)} \\ \mathbf{H}_l^T(\mathbf{x}^{(m)} - \bar{\mathbf{x}}) \end{bmatrix}. \tag{28}$$

As seen from Eq. (28), we still need the information on $\mathbf{x}^{(m)}$ in the subspace spanned by the eigen-axes in $\mathbf{H}_l$. This information was lost during the dimensional reduction process. The information loss caused by this approximation depends on how a feature space evolves throughout the learning. In general, the approximation error depends on the presentation order of training data, the data distribution, and the threshold θ for the accumulation ratio in Eqs. (11) and (12). In addition, the recognition performance depends on the generalization performance of RNN; thus, the effect of the approximation error for memory items is not easily estimated in general. However, recalling a fact that the eigen-axes in $\mathbf{H}_l$ are orthogonal to every vector in the subspace spanned by $\mathbf{U}_k$, the error could be small if an appropriate threshold θ is selected. Then, we can approximate the term $\mathbf{H}_l^T(\mathbf{x}^{(m)} - \bar{\mathbf{x}})$ to zero, and Eq. (27) is reduced to

$$\tilde{\mathbf{x}}^{(m)\prime} \approx \mathbf{R}^T \begin{bmatrix} \tilde{\mathbf{x}}^{(m)} \\ 0 \end{bmatrix} + \frac{1}{N+1}\mathbf{U}'^T(\bar{\mathbf{x}} - \bar{\mathbf{y}}) \tag{29}$$

Using Eq. (29), memory items in RNN can be recalculated without keeping the memory items $\mathbf{x}^{(m)} \in R^n$ in the input domain even after the eigenspace model is updated by Chunk IPCA. Then, RNN, which is implemented by RAN-LTM, is retrained with L training samples and M updated memory items $\left\{\tilde{\mathbf{x}}^{(m)\prime}, \tilde{\mathbf{z}}^{(m)\prime}\right\}$ $(m = 1, \ldots, M)$ based on **Algorithm 4**. The procedure of incremental learning is summarized in **Algorithm 7**.

Algorithm 7: ***Incremental Learning of Face Recognition Part***

```
Input: Eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$, RNN (RAN-LTM + memory items
       $\{\tilde{\mathbf{x}}^{(m)}, \tilde{\mathbf{z}}^{(m)}\}$ $(m = 1,\ldots,M)$), and $L$ training samples $(\mathbf{x}^{(p)}, \mathbf{d}^{(p)})$ $(p = 1,\ldots,L)$.
  Perform Chunk IPCA (Algorithm 2) for the training samples $(\mathbf{x}^{(p)}, \mathbf{d}^{(p)})$ $(p = 1,\ldots,L)$ to
  update the current eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$.
  Obtain the eigen-features of $(\mathbf{x}^{(p)}, \mathbf{d}^{(p)})$ $(p = 1,\ldots,L)$.
  Update all the memory items $\{\tilde{\mathbf{x}}^{(m)}, \tilde{\mathbf{z}}^{(m)}\}$ $(m = 1,\ldots,M)$ using Eq. (29).
  Perform Learning of RAN-LTM (Algorithm 4) to train RNN with the eigen-features of
  the $L$ training samples and the $M$ memory items.
Output: Eigenspace model $\Omega' = (\bar{\mathbf{x}}', \mathbf{U}'_{k+l}, \mathbf{\Lambda}'_{k+l}, N + L)$ and RNN
```

5.4 Overall algorithm of online incremental face recognition system

As mentioned in Section 5.1, the eigenspace model for the face detection part is obtained by applying PCA to a large dataset of human eye, nose, and mouth images. This dataset is used for training DNN as well. The training of VNN is carried out using a different dataset which includes a large amount of face and non-face images. Note that DNN and VNN are trained based on the learning algorithm of RAN (see Section 2). All the trainings for the face detection part, DNN, and VNN are conducted in advance.

Finally, we summarize the overall algorithm of the proposed online incremental face recognition system in **Algorithm 8**.

Algorithm 8: ***Online Incremental Face Recognition System***

```
Input: Initial training data $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ ($i$=1, ..., $N$).
  Perform Training of Initial Eigenspace (Algorithm 3) for $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ ($i$=1, ..., $N$) to obtain
  the eigenspace model $\Omega = (\bar{\mathbf{x}}, \mathbf{U}_k, \mathbf{\Lambda}_k, N)$.
  Obtain the eigen-features of $(\mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ ($i$=1, ..., $N$).
  Perform Learning of RAN-LTM (Algorithm 4) to train RNN with the eigen-features.
  Loop
    Input: video images of a person
      Repeat
        Perform Face Detection (Algorithm 5).
        Perform Recognition and Verification of Face (Algorithm 6).
        Keep a set of misclassified face sub-images and the class labels in the system.
      Until the person is out of sight.
    Perform Incremental Learning of Face Recognition Part (Algorithm 7).
  End loop
```

6. Performance evaluation

6.1 Experimental Setup

To simulate real-life environments, 224 video clips are collected for 22 persons (19 males / 3 females) during about 11 months such that temporal changes in facial appearances are included. Seven people (5 males / 2 females) are chosen as registrants and the other people (14 males /a female) are non-registrants. The duration of each video clip is 5-15 (sec.). A video clip is given to the face detection part, and the detected face images are automatically forwarded to the face recognition part. The numbers of detected face images are summarized in Table 1. The three letters in Table 1 indicate the code of the 22 subjects in which M/F and R/U mean Male/Female and Registered/Unregistered, respectively; for example, the third registered male is coded as MR3.

The recognition performance is evaluated through two-fold cross-validation; thus, the whole dataset is subdivided into two subsets: Set A and Set B. When Set A is used for learning RNN, Set B is used for testing the generalization performance, and vice versa. Note that since the incremental learning is applied only for misclassified face images, the recognition accuracy before the incremental learning is an important performance measure. Hence, there are at least two performance measures for the training dataset: one is the performance of RNN using a set of training samples given at each learning stage, and the other is the performance using all training datasets given so far after the incremental learning is carried out. In the following, let us call the former and latter datasets as *incremental dataset* and *training dataset*, respectively. Besides, let us call the performances over the incremental dataset and training dataset as *incremental performance* and *training performance*, respectively. We divide the whole dataset into 16 subsets, each of which corresponds to an incremental dataset. Table 2 shows the number of images included in the incremental datasets.

Set	MR1	FR1	MR2	MR3	FR2	MR4	MR5	FU1	MU1	MU2	MU3	MU4	MU5	MU6	MU7	MU8	MU9	MU10	Mu1	Mu12	Mu13	Mu14	Total
A	351	254	364	381	241	400	136	133	131	294	110	103	170	136	174	33	79	15	75	17	10	9	3766
B	170	220	297	671	297	241	359	126	228	292	80	233	117	202	182	14	9	14	28	18	9	9	3816

Table 1. Two face datasets (Set A and Set B) for training and test. The three letters in the upper row mean the registrant code and the values in the second and third rows are the numbers of face images.

Set	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	220	232	304	205	228	272	239	258	212	233	290	212	257	188	199	217
B	288	204	269	246	273	270	240	281	205	249	194	241	214	226	210	206

Table 2. Number of images included in the 16 incremental datasets.

Stage	Init.	1	2	…	12	13	14	15
Case 1	1	2	3	…	13	14	15	16
Case 2	1,2	3	4	…	14	15	16	---
Case 3	1,2,3	4	5	…	15	16	---	---

Table 3. Three series of incremental datasets. The number in Table 2 corresponds to the tag number of the corresponding incremental dataset.

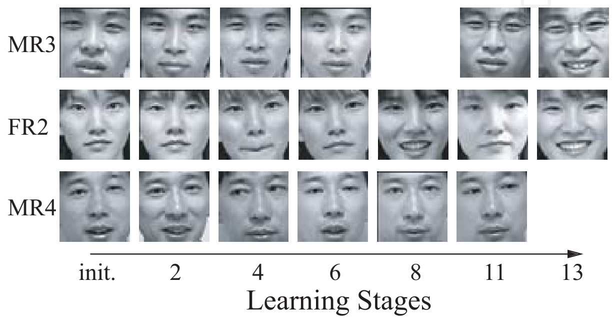


Fig. 6. Examples of face images trained at different learning stages.

The size of an initial dataset can influence the test performance because different initial eigen-spaces are constructed. However, if the incremental learning is successfully carried out, the final performance should not depend on the size of the initial dataset. Hence, the three different series of incremental datasets shown in Table 3 are defined to see the influence. Note that the number in Table 3 corresponds to the tag number (1-16) of the incremental dataset in Table 2. Hence, we can see that Case 1 has 15 learning stages and the number of images in the initial dataset is 220 for Set A and 288 for Set B, which correspond to 6.7% and 7.5% over the whole data. On the other hand, the sizes of the initial datasets in Case 2 and Case 3 are set to a larger value as compared with that in Case 1; while the numbers of learning stages are smaller than that in Case 1. Figure 6 shows the examples of detected face images for three registered persons at several learning stages.
When an initial dataset is trained by RNN, the number of hidden units is fixed with 50 in this experiment. The other parameters are set as follows: $\sigma^2 = 7$, $\varepsilon = 0.01$, and $\delta = 5$. The threshold θ of the accumulation ratio in IPCA is set to 0.9; thus, when the accumulation ratio is below 0.9, new eigen-axes are augmented.

6.2 Experimental results

Figure 7 shows the evolution of learning time over 15 learning stages when the chunk size L is 10 in Chunk IPCA (CIPCA). The curves of *CIPCA* and *IPCA* show the learning time for feature extraction, while those of *CIPCA+RAN-LTM* and *IPCA+RAN-LTM* mean the learning time for both feature extraction part and classifier. As you can see from the results, the learning time of feature extraction by Chunk IPCA is greatly reduced as compared with IPCA. This is also confirmed in Table 4.
The learning time of Chunk IPCA decreases as the chunk size increases, and Chunk IPCA is much faster than IPCA even though the feature dimensions at the final stage do not have large differences between IPCA and Chunk IPCA. When the chunk size is 10, Chunk IPCA is about 8 times faster than IPCA. The reason why the decreasing rate of the learning time becomes small for larger chunk size is that the time for finding eigen-axes dominates the total learning time (Ozawa et al., 2008).
To evaluate the effectiveness of learning an eigenspace, the classification accuracy of RAN-LTM is examined when the following three eigenspace models are adopted:
1. Static eigenspace model using PCA

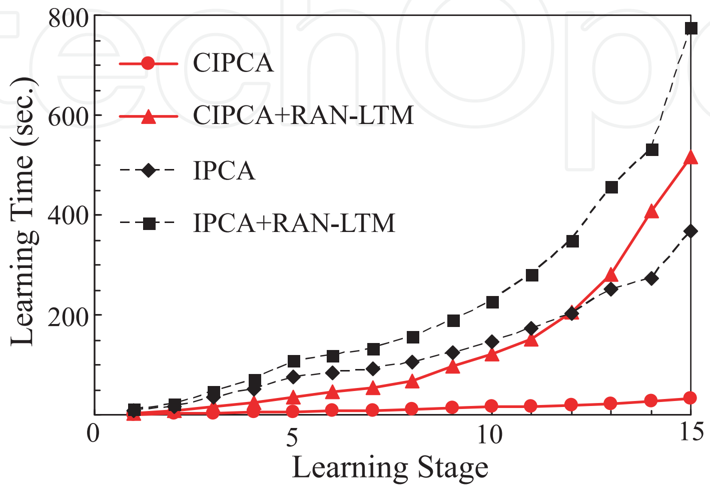


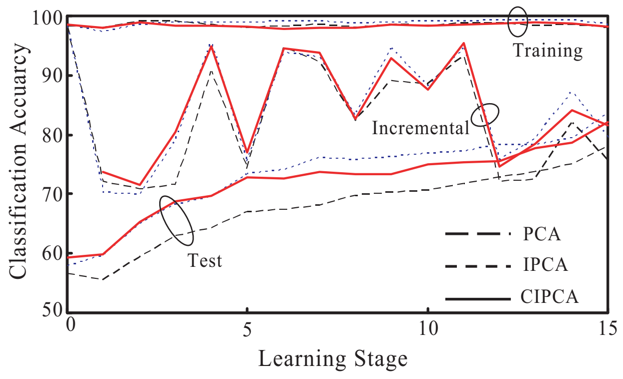
Fig. 7. Evolution of learning time for four different models (sec.).

	IPCA	CIPCA(10)	CIPCA(50)	CIPCA(100)
Time (sec.)	376.2	45.6	22.5	18.1
Dimensions	178	167	186	192

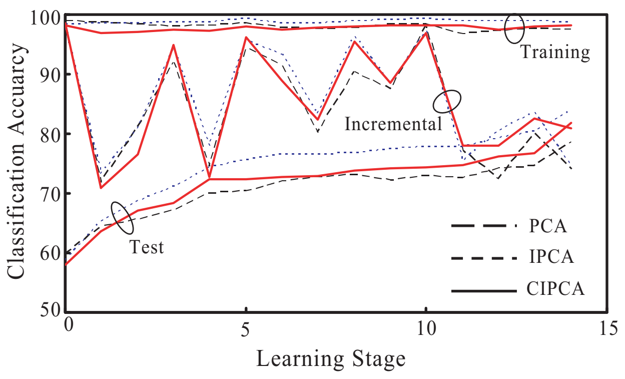
Table 4. Comparisons of Learning time and dimensions of feature vectors at the final learning stage. CIPCA(10), CIPCA(50), and CIPCA(100) stand for Chunk IPCA in which the chunk sizes are set to 10, 50, and 100, respectively.

2. Adaptive eigenspace model using the extended IPCA
3. Adaptive eigenspace model using Chunk IPCA.

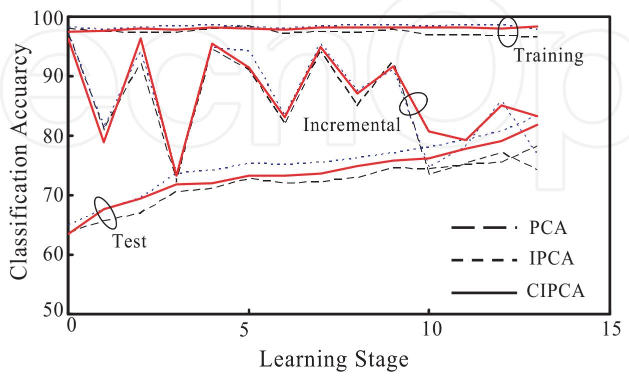
Figures 8 (a)-(c) show the evolution of recognition accuracy over 15 learning stages when the percentage of initial training data is (a) 6.7%, (b) 12.5%, and (c) 20%, respectively. As stated before, the size of an initial dataset can influence the recognition accuracy because different eigenspaces are constructed at the starting point. As seen from Figs. 8 (a)-(c), the initial test performance at stage 0 is higher when the number of initial training data is larger; however, the test performance of IPCA and Chunk IPCA is monotonously enhanced over the learning stages and it reaches almost the same accuracy regardless of the initial datasets. Considering that the total number of training data is the same among the three cases, we can say that the information on training samples is stably accumulated in RNN without serious forgetting even though RNN is reconstructed all the time the eigenspace model is updated.
In addition, the test performance of RNN with IPCA and Chunk IPCA has significant improvement against RNN with PCA. This result shows that the incremental learning of a



(a) Case 1 (Initial Data: 6.7%)



(b) Case 2 (Initial Data: 12.5%)



(c) Case 3 (Initial Data: 20%)

Fig. 8. Evolution of recognition accuracy for three different datasets (incremental, training, test) over the learning stages when the percentages of initial training datasets are set to (a) 6.7%, (b) 12.5%, and (c) 20.0%.

feature space is very effective to enhance the generalization performance of RNN. However, Chunk IPCA has slightly lower performance than IPCA. It is considered that this degradation originates from the approximation error of the eigenspace model using Chunk IPCA.
In Figs. 8 (a)-(c), we can see that although the incremental performance is fluctuated, the training performance of RNN with IPCA and Chunk IPCA changes very stably over the learning stages. On the other hand, the training performance of RNN with PCA rather drops down as the learning stage proceeds. Since the incremental performance is defined as a kind of test performance for the incoming training dataset, it is natural to be fluctuated. The important result is that the misclassified images in the incremental dataset are trained stably without degrading the classification accuracy for the past training data.
From the above results, it is concluded that the proposed incremental learning scheme, in which Chunk IPCA and RAN-LTM are simultaneously trained in an online fashion, works quite well and the learning time is significantly reduced by introducing Chunk IPCA into the learning of the feature extraction part.

7. Conclusions

This chapter described a new approach to constructing adaptive face recognition systems in which a low-dimensional feature space and a classifier are simultaneously learned in an online way. To learn a useful feature space incrementally, we adopted Chunk Incremental Principal Component Analysis in which a chunk of given training samples are learned at a time to update an eigenspace model. On the other hand, Resource Allocating Network with Long-Term Memory (RAN-LTM) is adopted as a classifier model not only because incremental learning of incoming samples is stably carried out, but also because the network can be easily reconstructed to adapt to dynamically changed eigenspace models.
To evaluate the incremental learning performance of the face recognition system, a self-compiled face image database was used. In the experiments, we verify that the incremental learning of the feature extraction part and classifier works well without serious forgetting, and that the test performance is improved as the incremental learning stages proceed. Furthermore, we also show that Chunk IPCA is very efficient compared with IPCA in term of learning time; in fact, the learning speed of Chunk IPCA was at least 8 times faster than IPCA.

8. References

Carpenter, G. A. and Grossberg, S. (1988). The ART of Adaptive Pattern Recognition by a Self-organizing Neural Network, *IEEE Computer*, Vol. 21, No. 3, pp. 77-88.

Hall, P. & Martin, R. (1998). Incremental Eigenanalysis for Classification, *Proceedings of British Machine Vision Conference*, Vol. 1, pp. 286-295.

Hall, P. Marshall, D. & Martin, R. (2000). Merging and Splitting Eigenspace Models, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 9, pp. 1042-1049.

Jain, L. C., Halici, U., Hayashi, I., & Lee, S. B. (1999). *Intelligent Biometric Techniques in Fingerprint and Face Recognition*, CRC Press.

Kasabov, N. (2007). *Evolving Connectionist Systems: The Knowledge Engineering Approach*, Springer, London.

Kobayashi, M., Zamani, A., Ozawa, S. & Abe, S. (2001). Reducing Computations in Incremental Learning for Feedforward Neural Network with Long-term Memory, *Proceedings of Int. Joint Conf. on Neural Networks*, Vol. 3, pp. 1989-1994.

Oja, E. & Karhunen, J. (1985). On Stochastic Approximation of the Eigenvectors and Eigenvalues of the Expectation of a Random Matrix, *J. Math. Analysis and Application*, Vol. 106, pp. 69-84.

Ozawa, S., Pang, S., & Kasabov, N. (2004) A Modified Incremental Principal Component Analysis for On-line Learning of Feature Space and Classifier. In: *PRICAI 2004: Trends in Artificial Intelligence*, Zhang, C. et al. (Eds.), LNAI, Springer-Verlag, pp. 231-240.

Ozawa, S., Toh, S. L., Abe, S., Pang, S., & Kasabov, N. (2005). Incremental Learning of Feature Space and Classifier for Face Recognition, *Neural Networks*, Vol. 18, Nos. 5-6, pp. 575-584.

Ozawa, S., Pang, S., & Kasabov, N. (2008). Incremental Learning of Chunk Data for On-line Pattern Classification Systems, *IEEE Trans. on Neural Networks*, Vol. 19, No. 6, pp. 1061-1074.

Pang, S., Ozawa, S. & Kasabov, N. (2005). Incremental Linear Discriminant Analysis for Classification of Data Streams, *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, Vol. 35, No. 5, pp. 905-914.

Sanger, T. D. (1989). Optimal Unsupervised Learning in a Single-layer Linear Feedforward Neural Network, *Neural Networks*, Vol. 2, No. 6, pp. 459-473.

Weng, J., Zhang Y. & Hwang, W.-S. (2003). Candid Covariance-Free Incremental Principal Component Analysis, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 25, No. 8, pp. 1034-1040.

Weng, J. & Hwang, W.-S. (2007). Incremental Hierarchical Discriminant Regression, *IEEE Trans. on Neural Networks*, Vol. 18, No. 2, pp. 397-415.

Zhao, H., Yuen, P. C. & Kwok, J. T. (2006). A Novel Incremental Principal Component Analysis and Its Application for Face Recognition, *IEEE Trans. on Systems, Man and Cybernetics, Part B*, Vol. 36, No. 4, pp. 873-886.