Probabilistic PCA and ICA subspace mixture models for image segmentation

Dick de Ridder*†, Josef Kittler† and Robert P.W. Duin*
*Pattern Recognition Group
Dept. of Applied Physics, Delft University of Technology
Lorentzweg 1, 2628 CJ Delft, The Netherlands
dick@ph.tn.tudelft.nl
†Centre for Vision, Speech and Signal Processing
Dept. of Electronic & Electrical Engineering, University of Surrey
Guildford, GU2 5XH Surrey, United Kingdom

Abstract

High-dimensional data, such as images represented as points in the space spanned by their pixel values, can often be described in a significantly smaller number of dimensions than the original. One of the ways of finding low-dimensional representations is to train a mixture model of principal component analysers (PCA) on the data. However, some types of data do not fulfill the assumptions of PCA, calling for application of different subspace methods. One such a method is ICA, which has been shown in recent years to be able to find interesting basis vectors (features) in signal and image data. In this paper, a mixture model of ICA subspaces is developed similar to a mixture model of PCA subspaces proposed by others. The new algorithm is applied to a natural texture segmentation problem and is shown to give encouraging results.

1 Introduction

Image data can often be described in a much lower number of parameters than there are pixels in the original image, due to the large redundancy in normal images and the fact that neighbouring pixels are highly correlated. Representing images as points in a high-dimensional space does not allow easy exploitation of this redundancy. Structured images can more naturally be represented by subspaces, with points in the subspace corresponding to slightly translated, rotated, scaled etc. versions of the same image. The subspace then becomes an invariant description of a single image or image patch. Recent years have seen a renewed interest in such subspace models to model (image) data. Originally proposed as early as the 1970s and 1980s, a problem then was the large computational cost of these methods. With the advent of more computational power, fitting subspace models to large datasets has become feasible. Moreover, it is now also possible to train mixture models of subspaces to model data non-linearly by approximating its distribution by a number of linear subspaces.

In recent work, several methods to do this have been proposed and applied (see [12] for an overview). In [6], an EM algorithm was developed to find mixtures of local prin-

BMVC 2000 doi:10.5244/C.14.12

cipal component analysers (PCA), which was applied to handwritten digit recognition invariant to size, skew etc. This model was refined by Tipping and Bishop in [12], in which a probabilistic formulation of PCA was proposed and used in conjunction with an EM algorithm. A similar but simpler method, using k-means clustering, was proposed in [3] and [4] and applied to image segmentation, object recognition and image database retrieval. Although in principle any of a number of methods can be used to find mixture models, probabilistic formulations of subspace-finding methods allow for a natural extension using a method very much like that used for learning mixtures of Gaussians. Therefore, we will consider Tipping and Bishop's algorithm [12] for learning mixtures of PCA models, which will be discussed briefly in section 2.

An open problem is that this PCA subspace approach works well only for structured image content, e.g. regular textures or edges. However, if the image contains more natural, irregular textures, a PCA description quickly becomes less discriminative. This is where other subspace methods can come in, such as independent component analysis (ICA), which recently has received much attention. The goal of a method for ICA mixture models should be two-fold: firstly, to find subspaces; secondly, to find characteristic directions in each model giving a better description of the data than PCA. This is the contribution of this paper: in section 3, a mixture model of ICA subspaces is developed analogous to the PCA mixture model of Tipping and Bishop mentioned before. As an illustration of possible applications, in section 4 the technique is used for image segmentation and compared to mixtures of PCA models. Section 5 ends with some conclusions.

2 Probabilistic PCA

Tipping and Bishop [12] found a probabilistic formulation of PCA by viewing it as a latent variable problem, in which the d-dimensional observed data vector $\mathbf{t}_n$, $n = 1, \ldots, N$ can be described in terms of an m-dimensional unobserved vector $\mathbf{s}_n$ and a noise term,

$$\mathbf{t}_n = \mathbf{A}\mathbf{s}_n + \epsilon \tag{1}$$

where $\mathbf{A}$ is a $d \times m$ matrix ($m < d$) and ϵ is multivariate i.i.d. Gaussian with a diagonal covariance matrix $\sigma^2\mathbf{I}$. The probability of observing data vector $\mathbf{t}_n$ given the latent vector $\mathbf{s}_n$ is (writing $\beta = \frac{1}{\sigma^2}$):

$$p(\mathbf{t}_n|\mathbf{s}_n, \mathbf{A}) = \frac{1}{(2\pi)^{\frac{d}{2}}\beta^{-\frac{d}{2}}} \exp\left(-\frac{\beta}{2}(\mathbf{t}_n - \mathbf{A}\mathbf{s}_n)^T(\mathbf{t}_n - \mathbf{A}\mathbf{s}_n)\right) \tag{2}$$

Using a Gaussian prior (zero mean, unit standard deviation) over the latent variables $\mathbf{s}_n$, an EM algorithm can be developed to find the parameters β_{ML} and $\mathbf{A}_{ML}$. Furthermore, the algorithm can be quite naturally extended to find mixtures of such models, by introducing a mean $\boldsymbol{\mu}^i$ for each model i and re-estimating $p(\mathbf{t}_n|i)$ and the prior probabilities for each model, $p(i)$, in each step of the EM algorithm.

3 Probabilistic ICA

Independent component analysis (ICA) finds directions in the data which lead to independent components instead of just uncorrelated ones, as PCA does. There is a wide range

of algorithms for performing ICA, based on entropy minimisation, minimisation of mutual information, optimisation of a non-Gaussianity measure, and maximum likelihood (ML). The latter leads to a probabilistic formulation. Recently, a mixture model of ML-trained local independent component analysers has been proposed [9]. A drawback of this method is that it does not extend easily to finding true subspaces; that is, only spaces with as many dimensions as the original space can be found. In applications such as image coding this need not be a problem, as the input data fills the original space quite well (studies have shown that even overcomplete bases can be found [10]). However, in tasks such as segmentation or image description for image database retrieval, images can often be described in a number of dimensions far lower than the number of pixels in a local window. Therefore, in this section a probabilistic mixture model of true ICA subspaces is developed, based on work performed earlier by MacKay [11], Lee et al. [8, 9], Lewicki and Sejnowski [10] and Hyvärinen [7].

3.1 ICA subspaces

For the formulation of an ICA subspace model, the starting point is the same model as used for PCA (eqn. (1)), where in ICA terminology the vectors $\mathbf{s}_n$ are the *sources*. The difference is that these sources are not assumed to have a Gaussian distribution; instead, ICA looks for super-Gaussian or sub-Gaussian distributions. This necessitates a gradient-following algorithm in which the distribution of the estimated sources is made to deviate from a Gaussian. Note that for the following derivation the pseudo-inverse of the mixing matrix $\mathbf{A}$ is used to find the unmixing matrix $\mathbf{W}$, i.e. $\mathbf{W} = \mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ and, conversely, $\mathbf{A} = \mathbf{W}^+ = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$.

The likelihood for one vector $\mathbf{t}_n$ is:

$$p(\mathbf{t}_n|\mathbf{A}) = \int p(\mathbf{t}_n|\mathbf{s},\mathbf{A})p(\mathbf{s})d\mathbf{s} \tag{3}$$

The integral in the likelihood (3) can be approximated by $p(\mathbf{t}_n|\mathbf{s}_{MP},\mathbf{A})p(\mathbf{s}_{MP})$ [2, 11], where $\mathbf{s}_{MP}$ is the most probable $\mathbf{s}_n$*. However, the result will have to be normalised for the change in volume under the Gaussian due to $\mathbf{A}$. The log likelihood thus is:

$$\begin{aligned}&\ln p(\mathbf{t}_n|\mathbf{A}) \approx \\ &\frac{d}{2}\ln\frac{\beta}{2\pi} + \frac{m}{2}\ln(2\pi) - \frac{\beta}{2}(\mathbf{t}_n - \mathbf{A}\mathbf{u}_n)^T(\mathbf{t}_n - \mathbf{A}\mathbf{u}_n) - \frac{1}{2}\ln\det\left(\mathbf{A}^T\mathbf{A}\right) + \ln p(\mathbf{u}_n)\end{aligned} \tag{4}$$

where $\mathbf{u}_n = \mathbf{W}\mathbf{t}_n$ is an estimate of $\mathbf{s}_n$. We can now derive a learning rule in $\mathbf{A}$ by differentiating (4) w.r.t. $\mathbf{A}$, but we chose to find a learning rule in $\mathbf{W}$ instead (see appendix A for a derivation):

$$\frac{\partial}{\partial\mathbf{W}}\ln p(\mathbf{t}_n|\mathbf{A}) = \mathbf{A}^T\mathbf{t}_n\mathbf{t}_n^T\left(\mathbf{I} - \mathbf{A}\mathbf{W}\right) + \mathbf{A}^T + \phi(\mathbf{u}_n)\mathbf{t}_n^T \tag{5}$$

where $\phi(\cdot)$ is the *score function*, indicating what type of distribution the algorithm looks for. Following Lee and Sejnowski [8], two different functions can be used to find either super-Gaussian or sub-Gaussian sources, i.e. source distributions j which are more

*From here on, for notational simplicity we will simply write $\mathbf{s}_n$

peaked or less peaked than the Gaussian:

$$\begin{aligned} &\text{super-Gaussian} \quad (k_j = 1) \quad : \quad \phi(u_{n,j}) = -\tanh(u_{n,j}) - u_{n,j} \\ &\text{sub-Gaussian} \quad (k_j = -1) \quad : \quad \phi(u_{n,j}) = \tanh(u_{n,j}) - u_{n,j} \end{aligned} \tag{6}$$

where $u_{n,j}$ is the j^{th} dimension of the n^{th} source vector, and k_j is an indicator which allows for automatic switching between super-Gaussian and sub-Gaussian models [8]:

$$k_j = \text{sign}\left(E(\text{sech}^2(u_{n,j}))E(u_{n,j}^2) - E(u_{n,j}\tanh(u_{n,j}))\right) \tag{7}$$

These expectations are taken over the entire dataset, i.e. for $n = 1, \ldots, N$. If a matrix $\mathbf{K}$ is constructed with k_j, $1 \leq j \leq m$, on the diagonal, eqns. (6) simplify to $\phi(\mathbf{u}_n) = -\mathbf{K}\tanh(\mathbf{u}_n) - \mathbf{u}_n$ and the summed gradient for a matrix $\mathbf{t}$ containing all points $\mathbf{t}_n$, $1 \leq n \leq N$, becomes

$$\frac{\partial}{\partial \mathbf{W}} \ln p(\mathbf{t}|\mathbf{A}) = \beta\left(\mathbf{A}^T\mathbf{t}\mathbf{t}^T\left(\mathbf{I} - \mathbf{A}\mathbf{W}\right)\right) + N\mathbf{A}^T - \left(\mathbf{K}\tanh(\mathbf{u}) + \mathbf{u}\right)\mathbf{t}^T \tag{8}$$

The term $(\mathbf{I} - \mathbf{A}\mathbf{W})$ describes the difference between the actual inverse and the pseudo-inverse, since if $\mathbf{A} = \mathbf{W}^{-1}$ the first term adds up to zero, and Lee's algorithm [8] results. In a paper by Lewicki and Sejnowski [10], another approach is chosen. Instead of working out the true derivative of the target function w.r.t. $\mathbf{A}$, they approximate $\mathbf{A}$ by only that part of it which can be inverted; therefore, $(\mathbf{I} - \mathbf{A}\mathbf{W})$ adds up to zero. Although useful when finding overcomplete bases, this approach does not seem right here: it means there is no longer an incentive to look for subspaces at all. Since $\mathbf{A}\mathbf{W} \neq \mathbf{I}$, there will always be a misfit between $\mathbf{t}$ and $\mathbf{A}\mathbf{u}$, and this should somehow be expressed in the likelihood function.

3.2 Automatic sphering

As incomplete bases are learned, the algorithm will look for both subspaces and independent components. If these goals are contradictory (i.e. looking for independent components in a predominantly Gaussian subspace), depending on the variances in the independent components and the main subspace, the algorithm will favour finding subspaces over finding independent components. To make the algorithm find independent components invariant to variance, the data can be sphered first. That is, the covariance matrix of the data can be required to be unity, i.e. $\mathbf{C} = E(\mathbf{t}\mathbf{t}^T) = \mathbf{I}$. In a mixture model (discussed below) it is not possible to pre-sphere the data, as it is unknown which data points will be assigned to which model. It is possible however to build the sphering into the model by estimating the covariance matrix and sphering in the algorithm itself, using $\mathbf{t}'_n = \mathbf{C}^{-\frac{1}{2}}\mathbf{t}_n$ and $\mathbf{u}_n = \mathbf{W}\mathbf{t}'_n$. The sphered data and corresponding sources can then be used in log-likelihood (4) and also in the gradient (8), giving (as β can be fixed at 1, but $\mathbf{C}$ has to be incorporated in the log-likelihood):

$$\begin{aligned} \ln p(\mathbf{t}_n|\mathbf{A}) \quad &\approx \quad \frac{m-d}{2}\ln(2\pi) - \frac{1}{2}(\mathbf{t}'_n - \mathbf{A}\mathbf{u}_n)^T(\mathbf{t}'_n - \mathbf{A}\mathbf{u}_n) \\ &\quad -\frac{1}{2}\ln\det\left(\mathbf{A}^T\mathbf{A}\right) - \frac{1}{2}\ln\det\left(\mathbf{C}\right) + \ln p(\mathbf{u}_n) \end{aligned} \tag{9}$$

$$\frac{\partial}{\partial \mathbf{W}} \ln p(\mathbf{t}_n|\mathbf{A}) \quad = \quad \mathbf{A}^T\mathbf{t}'_n{\mathbf{t}'_n}^T\left(\mathbf{I} - \mathbf{A}\mathbf{W}\right) + N\mathbf{A}^T + \left(\mathbf{K}\tanh(\mathbf{u}) - \mathbf{u}\right){\mathbf{t}'_n}^T \tag{10}$$

3.3 A mixture model

Following Lee, it is now straightforward to construct a mixture model. Say that the goal is to find h m-dimensional ICA models. The overall model now becomes [2, 9]:

$$p(\mathbf{t}) = \sum_{i=1}^{h} p(\mathbf{t}|i)p(i) \tag{11}$$

The values to be estimated are $\mathbf{A}^i$, $\mathbf{s}^i$ and $\boldsymbol{\mu}^i$: the mixing matrix, the sources and the offset for model i, respectively. Furthermore, for the automatic switching between super-Gaussian and sub-Gaussian models a switching matrix $\mathbf{K}^i$ can be used and for the automatic sphering a covariance matrix $\mathbf{C}^i$ can be estimated for each model.

For estimation of $p(\mathbf{t}|i)$, the source component densities can be approximated by [9]:

$$\text{super-Gaussian} \quad (k_j = 1) \quad : \quad \ln p(u_{n,j}) \approx -|u_{n,j}| \tag{12}$$

$$\text{sub-Gaussian} \quad (k_j = -1) \quad : \quad \ln p(u_{n,j}) \approx -\ln\cosh(u_{n,j}) - \frac{(u_{n,j})^2}{2} \tag{13}$$

Estimation of $p(\mathbf{t}_n|i)$ can be done using (10), and $p(\mathbf{t}_n)$ can be calculated from (11) (note that the only difference is that the model mean $\boldsymbol{\mu}^i$ has to be subtracted from $\mathbf{t}_n$ beforehand). The probability of a model i given a data vector $\mathbf{t}_n$ can be found using Bayes' rule to be $p(i|\mathbf{t}_n) = (p(\mathbf{t}_n|i)p(i))/p(\mathbf{t}_n)$, giving the following formulae for the various model parameters, for the full algorithm including automatic switching and sphering (the tilde indicating the new estimates):

$$\mathbf{t}'_n = (\mathbf{C}^i)^{-\frac{1}{2}}(\mathbf{t}_n - \boldsymbol{\mu}^i) \tag{14}$$

$$\tilde{\mathbf{u}}^i_n = \mathbf{W}^i\mathbf{t}'_n \tag{15}$$

$$\tilde{p}(i) = \frac{1}{N}\sum_{n=1}^{N} p(i|\mathbf{t}_n) \tag{16}$$

$$\tilde{\boldsymbol{\mu}}^i = \frac{\sum_{n=1}^{N} p(i|\mathbf{t}_n)\mathbf{t}_n}{\sum_{n=1}^{N} p(i|\mathbf{t}_n)} \tag{17}$$

$$\tilde{\mathbf{C}}^i = \frac{\sum_{n=1}^{N} p(i|\mathbf{t}_n)(\mathbf{t}_n - \boldsymbol{\mu}^i)(\mathbf{t}_n - \boldsymbol{\mu}^i)^T}{\sum_{n=1}^{N} p(i|\mathbf{t}_n)} \tag{18}$$

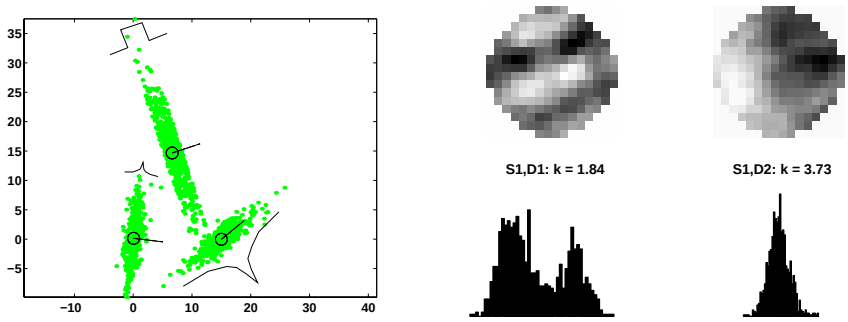
$$\tilde{k}^i_j = \text{sign}\left(\frac{\sum_{n=1}^{N} p(i|\mathbf{t}_n)\,\text{sech}^2(\tilde{u}^i_{n,j})}{\sum_{n=1}^{N} p(i|\mathbf{t}_n)} \frac{\sum_{n=1}^{N} p(i|\mathbf{t}_n)(\tilde{u}^i_{n,j})^2}{\sum_{n=1}^{N} p(i|\mathbf{t}_n)} - \frac{\sum_{n=1}^{N} p(i|\mathbf{t}_n)\,\tilde{u}^i_{n,j}\tanh(\tilde{u}^i_{n,j})}{\sum_{n=1}^{N} p(i|\mathbf{t}_n)} \right) \tag{19}$$

Finally, using Bayes' rule the learning rule for the unmixing matrix $\mathbf{W}^i$ can be expressed as follows:

$$\frac{\partial}{\partial \mathbf{W}^i}\ln p(\mathbf{t}_n) = p(i|\mathbf{t}_n)\frac{\partial}{\partial \mathbf{W}^i}\ln p(\mathbf{t}_n|\mathbf{A}^i, \boldsymbol{\mu}^i) \tag{20}$$

i.e. simply the gradient (10) weighed by $p(i|\mathbf{t}_n)$.

Figure 1 gives examples of a 1D subspace mixture model trained on 2D data and a 2D subspace mixture model trained on image data. Note how, for the 2D data, variance in a certain direction gets ignored if the direction contains a Gaussian distribution, due to the sphering.



(a) (b)

Figure 1: (a) Three 1D ICA models, each cluster containing one sub- or super-Gaussian component (which was found) and one Gaussian component. (b) ICA basis vectors found in patches (round, 16 pixel radius) taken from a natural texture (top) and the corresponding sub-Gaussian (left) and super-Gaussian (right) projections of the image samples onto the basis vectors, with their kurtoses.

4 Application: image segmentation

As an illustration of the differences between the methods, both PCA and ICA subspace mixture models are applied to some image segmentation problems. The problems are deliberately kept simple to highlight the differences between the methods.

PCA subspace mixture models have already been shown to work quite well on *structured* textures [3]; these results are not repeated here. Instead, five irregular, natural texture images from the Brodatz album were artificially combined (using a cross-shaped mask) to create 10 2-texture images and scaled to a range of $[0, 1]$. The images are shown in figure 2. On these images, PCA and ICA mixture models were trained each containing two 2-dimensional subspaces. The training data consisted of 1000 round image patches extracted from the combination images, where the radius of the patch was 8 pixels. Preprocessing the entire data set using PCA to remove noise directions (leaving eigenvectors explaining 90% of the variance) typically left 10-20 of the original 44 dimensions. Note that, besides speeding up the algorithms, this is necessary to make the data meet the assumptions of i.i.d. Gaussian noise outside the subspaces better.

Both the PCA and ICA algorithm were initialised by setting the model origins to grey-values found by k-means clustering. All other parameters were initialised to small random values, and the prior probabilities were initially set equal. The PCA mixture model needed no further settings, but the proposed ICA method needed careful setting of a learning rate, which was set to 1.0×10^{-5}. Both methods were stopped when the change in the likelihood fell below a threshold (1.0×10^{-8}) or after 5000 iterations, whichever came first. Training the ICA mixture models is a computationally very intensive process. Speed-ups are possible, such as only re-estimating the covariance matrices $\mathbf{C}^i$ and the switching matrices $\mathbf{K}^i$ once in a number of iterations, but these were not employed here.

Figure 2 shows the resulting segmentations, found by assigning each central pixel of an image patch the label of that subspace i for which $p(\mathbf{t}_n|i)$ was highest. It is immediately clear that neither subspace method is ideal for this type of application. Some texture

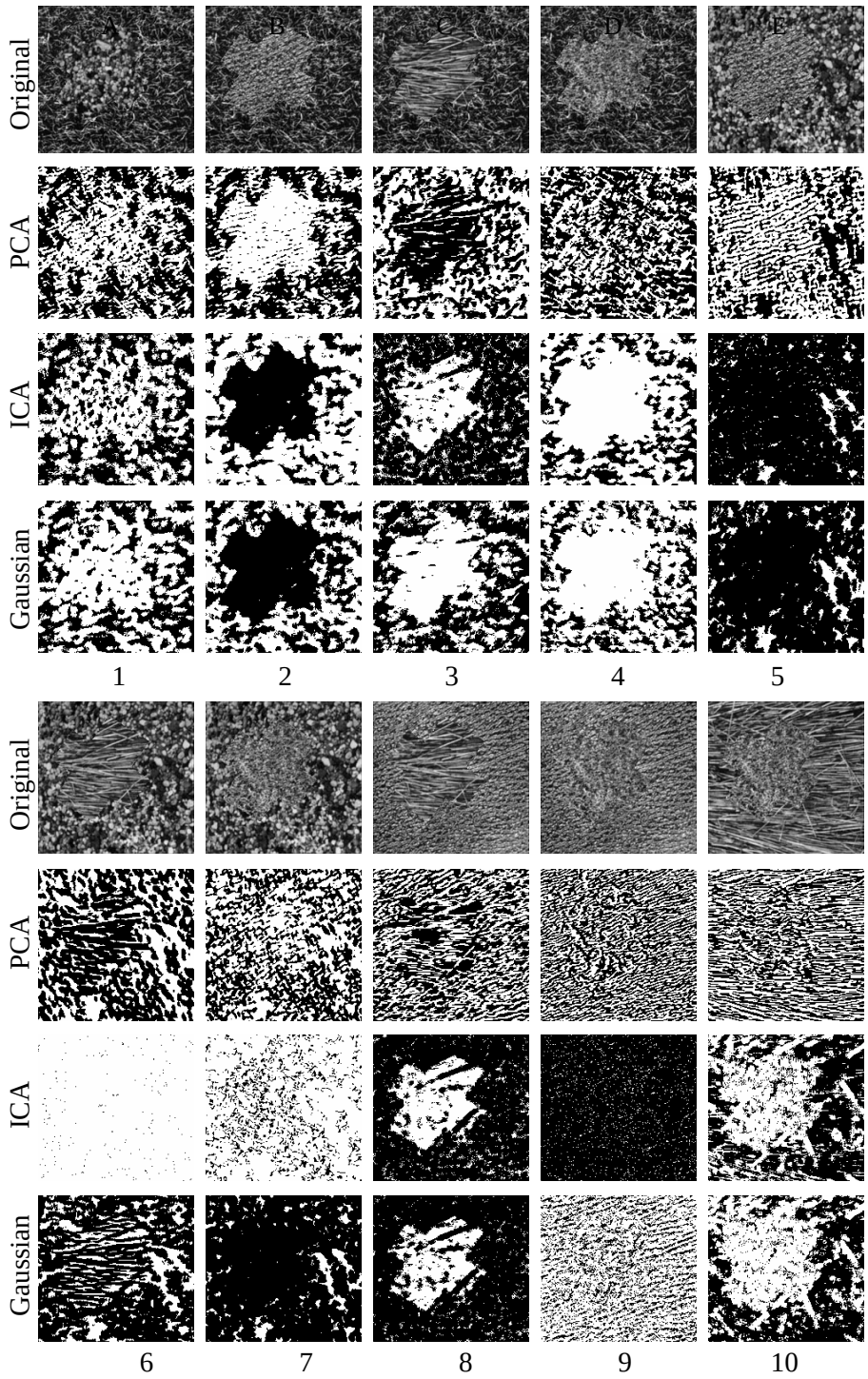


Figure 2: Segmentation results on a number of combinations of natural textures from the Brodatz album, by PCA, ICA and Gaussian mixture models.

combinations (2, 3, 8, 10) are segmented relatively well, given the difficulty of the task; others (5, 6, 7, 9) are too hard to segment and in the remaining segmentations (1, 4) the basic shape of the cross can be seen, but the overall segmentation is rather poor. The results do illustrate, however, that the ICA subspace mixture models have found more descriptive models than the PCA ones in most cases, giving better segmentation results.

A criticism of the ICA model might be that, unlike PCA, it uses the entire covariance matrix instead of just the covariance structure in the subspace. To verify whether this might be responsible for the better behaviour, the images were also segmented by training Gaussian mixture models with full covariance matrices (or, equivalently, a PCA model where $m = d$). The use of the covariance matrix indeed seems to be the main reason for the better behaviour. However, in some cases (2, 3, 4, 10) segmentations by the ICA model are still slightly better (i.e. showing less oversegmentation) than those by the Gaussian mixture model, indicating that some useful independent component directions have been found in the data.

5 Conclusion and discussion

Analogous to the probabilistic PCA mixture model of Tipping and Bishop, an ICA subspace mixture model has been developed. This algorithm generalises the ICA mixture model proposed earlier by Lee et al. Experimental results suggest that for certain kinds of problems, such as segmentation of natural textures, mixtures of ICA subspaces can perform better than mixtures of PCA subspaces.

Although the model has been shown to work well on 2D artificial data, the advantage of using ICA mixture models in image segmentation has not yet been proven conclusively. An open issue is to what extent its better performance is caused by the fact that the ICA model estimates a full covariance matrix for each subspace, whereas the PCA model only estimates covariance within the subspace. As sphering is a necessity to obtain truly independent components, it would be beneficial to find a formulation of the algorithm in which there is no need to estimate the entire covariance matrix. This will also lower the number of samples needed to train the model.

Another point for further research is the speed of the algorithm, which should be improved to facilitate application. Finally, with respect to the segmentation application, it would be beneficial to find ways of automatically choosing optimal window sizes, subspace dimensionalities etc.

Acknowledgements

This work was partly supported by the Foundation for Computer Science Research in the Netherlands (SION), the Dutch Organisation for Scientific Research (NWO) and the Engineering and Physical Sciences Research Council (EPSRC) in the UK (grant numbers GR/M90665 and GR/L61095). The first author would like to thank the Centre for Vision, Speech and Signal Processing and the EPSRC for allowing him to visit the centre as a visiting fellow for six months.

References

[1] S. Amari. Natural gradient learning for over- and under-complete bases in ICA. *Neural Computation*, 11:1875–1883, 1990.

[2] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, Oxford, 1995.

[3] D. de Ridder, J. Kittler, O. Lemmers, and R.P.W. Duin. The adaptive subspace map for texture segmentation. In *Proceedings of the* 15^{th} *IAPR International Conference on Pattern Recognition*, 2000.

[4] D. de Ridder, O. Lemmers, R.P.W. Duin, and J. Kittler. The adaptive subspace map for image description and image database retrieval. In *Proceedings of the joint IAPR International Workshops on Syntactical and Structural Pattern Recognition (S+SSPR 2000)*, 2000.

[5] P.A. Devijver and J. Kittler. *Pattern recognition, a statistical approach*. Prentice-Hall, London, 1982.

[6] G.E. Hinton, P. Dayan, and M. Revow. Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997.

[7] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[8] T.-W. Lee, M. Girolami, and T.J. Sejnowski. Independent component analysis using an extended infomax algorithm for mixed sub-gaussian and super-gaussian sources. *Neural Computation*, 11(2):417–441, 1999.

[9] T.-W. Lee, M.S. Lewicki, and T.J. Sejnowski. Unsupervised classification with non-Gaussian mixture models using ICA. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. NIPS, MIT Press.

[10] M.S. Lewicki and T.J. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12(2):337–365, 2000.

[11] D. MacKay. Maximum likelihood and covariant algorithms for independent component analysis. Draft 3.7, 1996.

[12] M.E. Tipping and C.M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999.

A Derivation of the learning rule

Differentiating eqn. 4 with respect to $\mathbf{W}$ and ignoring constant terms gives

$$\frac{\partial}{\partial \mathbf{W}} \ln p(\mathbf{t}|\mathbf{A}) \approx \frac{\partial}{\partial \mathbf{W}} \left(\underbrace{-\frac{\beta}{2}(\mathbf{t}-\mathbf{A}\mathbf{W}\mathbf{t})^T(\mathbf{t}-\mathbf{A}\mathbf{W}\mathbf{t})}_{(I)} \underbrace{-\frac{1}{2}\ln\det\left(\mathbf{A}^T\mathbf{A}\right)}_{(II)} \underbrace{+\ln p(\mathbf{u})}_{(III)} \right) \tag{21}$$

where $\mathbf{u}$ has been partially replaced by $\mathbf{W}\mathbf{t}^{\dagger}$.

Differentiating (I) w.r.t. a single matrix element $\mathbf{W}_{ij}$ [5] and rewriting gives, using $\mathbf{A} = \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$ and therefore $\mathbf{W}^T\mathbf{A}^T = \mathbf{A}\mathbf{W}$ and $\mathbf{A}\mathbf{W}\mathbf{A}\mathbf{W} = \mathbf{A}\mathbf{W}$,

$$
\begin{aligned}
&-\frac{\beta}{2}\frac{\partial}{\partial \mathbf{W}_{ij}}\left(\mathbf{t}^T\mathbf{t} - \mathbf{t}^T\mathbf{A}\mathbf{W}\mathbf{t}\right) = \frac{\beta}{2}\frac{\partial}{\partial \mathbf{W}_{ij}}\left(\mathbf{t}^T\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}\mathbf{t}\right) \\
&= \frac{\beta}{2}\left(\mathbf{t}^T\mathbf{W}_{(j,i)}(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}\mathbf{t} + \mathbf{t}^T\mathbf{W}^T\left(\left[-(\mathbf{W}\mathbf{W}^T)^{-1}(\mathbf{W}_{(i,j)}\mathbf{W}^T + \right.\right.\right. \\
&\qquad \left.\left.\left. \mathbf{W}\mathbf{W}_{(j,i)})(\mathbf{W}\mathbf{W}^T)^{-1}\right]\mathbf{W}\mathbf{t} + (\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{W}_{(i,j)}\mathbf{t}\right)\right) \\
&= \frac{\beta}{2}(\underbrace{\mathbf{t}^T\mathbf{W}_{(j,i)}\mathbf{A}^T\mathbf{t}}_{(A)}\underbrace{-\mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{A}\mathbf{W}\mathbf{t}}_{(B)}\underbrace{-\mathbf{t}^T\mathbf{W}^T\mathbf{A}^T\mathbf{W}_{(j,i)}\mathbf{A}^T\mathbf{t}}_{(C)}\underbrace{+\mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{t}}_{(D)}) \qquad (22)
\end{aligned}
$$

where $\mathbf{W}_{(i,j)}$ is an all-zero matrix with only element $\mathbf{W}_{ij}$ set to 1. Note that since (A) and (D) are each others transposed and give scalars, they are the same. This also holds for (B) and (C). Therefore, the total expression for (I) becomes

$$
\frac{\beta}{2}\left(2\mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{t} - 2\mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{A}\mathbf{W}\mathbf{t}\right) = \beta\left(\mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{t} - \mathbf{t}^T\mathbf{A}\mathbf{W}_{(i,j)}\mathbf{A}\mathbf{W}\mathbf{t}\right) \qquad (23)
$$

which for full matrices is

$$
\beta\left(\mathbf{A}^T\mathbf{t}\mathbf{t}^T - \mathbf{A}^T\mathbf{t}\mathbf{t}^T\mathbf{W}^T\mathbf{A}^T\right) = \beta\left(\mathbf{A}^T\mathbf{t}\mathbf{t}^T\left(\mathbf{I} - \mathbf{A}\mathbf{W}\right)\right) \qquad (24)
$$

For (II), following [5] the derivative can be found to be simply

$$
\frac{\partial}{\partial \mathbf{W}}\left(-\frac{1}{2}\ln\det(\mathbf{A}^T\mathbf{A})\right) = \frac{\partial}{\partial \mathbf{W}}\left(-\frac{1}{2}\ln\det\left((\mathbf{W}\mathbf{W}^T)^{-1}\right)\right) = \mathbf{A}^T \qquad (25)
$$

Finally for (III), using the chain rule,

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{W}_{ij}}\ln p(\mathbf{u}) &= \frac{\partial}{\partial \mathbf{W}_{ij}}\left(\ln\prod_{l=1}^{m}p(u_l)\right) = \frac{\partial}{\partial \mathbf{W}_{ij}}\left(\sum_{l=1}^{m}\ln p\left(\sum_{k=1}^{d}\mathbf{W}_{lk}t_k\right)\right) \\
&= \sum_{l=1}^{m}\left(\frac{\partial \ln p\left(\sum_{k=1}^{d}\mathbf{W}_{lk}\mathbf{t}_k\right)}{\partial\sum_{k=1}^{d}\mathbf{W}_{lk}t_k}\right)\left(\frac{\partial\sum_{k=1}^{d}\mathbf{W}_{lk}t_k}{\partial \mathbf{W}_{ij}}\right) = \phi_i(u_i)t_j \qquad (26)
\end{aligned}
$$

or, for full vectors,

$$
\frac{\partial}{\partial \mathbf{W}}\ln p(\mathbf{u}) = \phi(\mathbf{u})\mathbf{t}^T \qquad (27)
$$

Taking (I)-(III) together, the gradient becomes

$$
\frac{\partial}{\partial \mathbf{W}}\ln p(\mathbf{t}|\mathbf{A}) = \beta\left(\mathbf{A}^T\mathbf{t}\mathbf{t}^T\left(\mathbf{I} - \mathbf{A}\mathbf{W}\right)\right) + \mathbf{A}^T + \phi(\mathbf{u})\mathbf{t}^T \qquad (28)
$$

Note that it is possible to apply the often used *natural gradient* technique [1] by using:

$$
\left(\frac{\partial}{\partial \mathbf{W}}\right)' = \frac{\partial}{\partial \mathbf{W}} - \mathbf{W}\left(\frac{\partial}{\partial \mathbf{W}}\right)^T\mathbf{W} \qquad (29)
$$

to speed up convergence.

[†] In this appendix, for brevity $\mathbf{t}$ will be used to denote one data vector $\mathbf{t}_n$ and $\mathbf{u}$ to denote one source estimate $\mathbf{u}_n$. All formulae hold identically for matrices in which the columns are these vectors.