Analyzing sensory neurons by extracting features from color image using linear models

a Special Project

by Morten Arngren, s030192

on July 2, 2006

1 Introduction

One of many major research areas in mathematical modeling is the topic of visual neuroscience. It deals with how image-statistics of a natural environment is reflected in the processing of the early visual system in the human brain. Throughout the literature many attempts have been made to form mathematical models for the different stages in neurological processing of the input from the retina, to name a few refer to [3], [4] and [13]. The common understanding is that a great part of the processing can be approximated via simple linear models.

This report aims to describe the visual neurological processing from a mathematical approach with different linear decompositions: *Principal Component Analysis* (PCA), *Independent Component Analysis* (ICA) and *Non-Negative Matrix Factorization* (NNMF). A small set of simulations will be conducted on natural images and the results will be analyzed and compared to similar experiments [3] [4].

In section 2 a short description of the visual cortex is given from a signal processing point of view and how this is related to the three decompositions mentioned above. Afterwards the 3 decomposition models PCA, ICA and NNMF are presented in section 3, 4 and 5 respectively. A set of simulations on natural images is described in section 6 and the results related to the visual cortex. Finally the report in concluded in section 7.

2 Visual Cortex

Understanding the sensory neurons of the visual system in the brain has received much attention as a primary research area in the last many decades. The main approach is attempting to model the sensory processing with linear modeling based on the statistical properties of the environment. In this section we will try to give a very short overview of the main issues for the neurological visual processing system based on Simoncelli & Olshausen [13], Hoyer & Hyvärinen [4] and Hoyer [3].

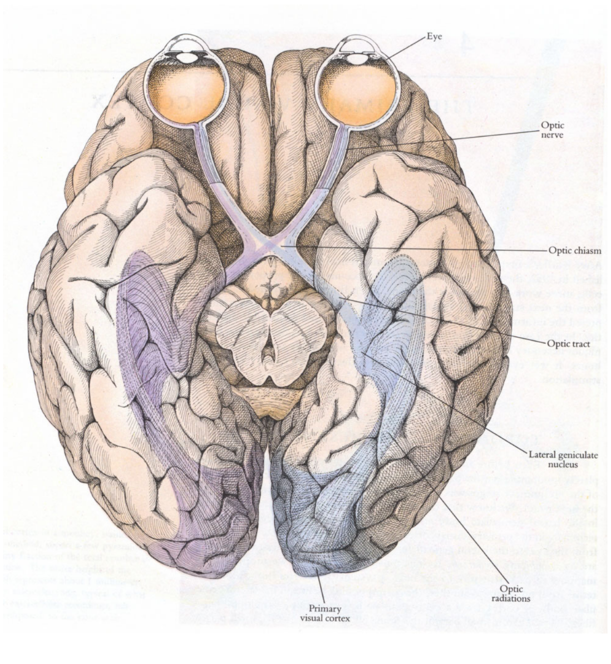

Figure 2.1: Physical outline of the visual cortex.

The visual cortex, also known as the primary visual cortex or V1, is part of the neural visual center of the brain and occupies about one third of the surface of the cerebral cortex in humans, illustrated in figure 2.1 [1].

The signal path in the brain can be described very shortly from a macro-perspective point of view. Visual stimuli from the natural world enters the eye and is projected on the retina (photo sensitive backside of the eye) and from here the Lateral Geniculate Nucleus (LGN) receives the visual information. The LGN is considered the primary processor of visual information and together with the retina, they are thought to eliminate the correlation and remove redundancy in the visual stimuli as stated by Hoyer & Hyvärinen [4] and Simoncelli & Olshausen [13]. The image data then still contains obvious structures like lines, edges contours etc. after decorrelating and thus in terms of efficient coding in the neurons much work is still to be done [13]. We will later review redundancy reduction in terms of *Principal Component Analysis* (PCA) and show corresponding simulation results.

From here the signal is passed to the visual cortex, which has different levels of complex processing, denoted V1-V5. In this context we shall only focus on V1 (primary visual cortex) as the first and most simple of the levels.

[1] Image taken from http://www.nmr.mgh.harvard.edu/∼rhoge/HST583/doc/HST583-Lab1.html

The neurons in V1 can be split up in simple- and complex cells, where the simple cells are thought to perform some linear filtering of the visual input. The spatial characteristics of these filters (receptive fields) can be described as having oriented and bandpass features (Hoyer [3] and Simoncelli & Olshausen [13]). Further neural processing of the sensory input based on the architecture of the visual cortex suggests a hierarchical organization, where the neurons becomes increasingly selective to more complex image structures [13].

If we focus on V1 of the visual cortex it has been argued by Simoncelli & Olshausen [13] that neural responses must be statistically independent in order to avoid duplication of information in the neurons and hence for the coding of information to be efficient. Thus applying the linear *Independent Component Analysis* (ICA) model with independence as the constraint to natural images seems justified and results from Hoyer [3] and Simoncelli & Olshausen [13] gave features similar to those found in receptive fields of simple cells in V1. Similar results has also been found using *Linear Sparse Coding* (LSC), where image data is decomposed into a linear model under the constraint that the components must to be as sparse as possible [3]. In this report we will apply ICA on natural image data and expect to see similar features comparable with results from Hoyer [3] and Simoncelli & Olshausen [13].

One fundamental discrepancy of many of the models used in this context is their ability to model negative components and since neurons can not have negative firing rates [3], these models are not optimal for this purpose. Instead a non-negative factorization model should be used to constrain the components not to be negative. Using an extended non-negative model Hoyer has shown very sparse features for B&W image data [3]. In this context we will focus on *Non-Negative Matrix Factorization* (NNMF) and show simulated results expected to be similar those found by Hoyer [3].

To start our analysis, we will give a presentation of *Principal Component Analysis*.

3 Pre-processing

One of the most simple decompositions is the *Principal Component Analysis* (PCA), where the dataset $\mathbf{X}$ is transformed onto a set of orthogonal axes (denoted the principal components) based on maximizing 2nd order information or variance. In this context we will describe PCA as a pre-processing step in terms of whitening data.

The process of *whitening* data can be an important step in data analysis, as it allows 1st and 2nd order information to be reduced or aligned for datasets from different sources. Whitening is the process of removing any mean-value from the data, ensuring unity variance and decorrelating the data (i.e. zero covariance). In this section we describe whitening and the theory behind.

Let us denote the dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, where $\mathbf{x}_j$ is an n dimensional column vector, i.e. $\mathbf{X}$ is an $n \times m$ matrix. In this context, we introduce the *expectation* operator $\mathcal{E}(\cdot)$ as statistical averaging given by

$$\mathcal{E}\{\mathbf{x}\} = \int_{-\infty}^{\infty} \mathbf{x} p_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} \tag{3.1}$$

where $p_{\mathbf{x}}(\mathbf{x})$ denoted the probability density function for $\mathbf{x}$.

The sample mean μ_i can easily be determined as the 1st order moment by $\mu_i = \mathcal{E}\{\mathbf{y}_i\}$, for $i = 1, 2, \ldots, n$, where $\mathbf{y}_i$ in this case is the m-dimensional i'th row vectors of $\mathbf{X}$. As the data $\mathbf{x}_j$ is usually based on measurements, the probability density $p_{\mathbf{x}}(\mathbf{x})$ may not always be known. The mean vector $\mu = \{\mu_1, \mu_2, \ldots, \mu_n\}$ can in these cases be approximated empirically by

$$\mu \approx \widehat{\mu} = \frac{1}{m} \sum_{j=1}^{m} \mathbf{x}_j \tag{3.2}$$

Similarly the covariance is determined by the 2nd order central moment (2nd order moment with zero mean) by $\mathbf{C}(x_k, x_l) = \mathcal{E}\{(x_k - \mu_k)(x_l - \mu_l)\}$, for $k, l = 1, 2, \ldots, n$. For $k = l$, the covariance becomes $\mathbf{C}(x_k, x_k) = \sigma_k^2$, i.e. the variance of x_k. The corresponding *covariance matrix* $\mathbf{\Sigma}$ for $n = 2$ can be expressed as

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \mathbf{C}(x_1, x_2) \\ \mathbf{C}(x_2, x_1) & \sigma_2^2 \end{bmatrix} \tag{3.3}$$

From the definition of $\mathbf{C}$ it is easy to see $\mathbf{C}(x_1, x_2) = \mathbf{C}(x_2, x_1)$ and that $\mathbf{\Sigma}$ is symmetric and always positive semi-definite. This can similarly be approximated empirically unbiased by

$$\widehat{\mathbf{\Sigma}} \approx \frac{1}{m-n} \sum_{j=1}^{m} (\mathbf{x}_j - \widehat{\mu})(\mathbf{x}_j - \widehat{\mu})^T \tag{3.4}$$

With the definition of the covariance matrix, it is easy to see that a white dataset has the identity matrix $\mathcal{I}$ as the covariance matrix, i.e. $\mathbf{\Sigma} = \mathcal{I}$, where the data components have unity variance and zero covariance.

To decorrelate the data $\mathbf{X}$, we decompose the covariance matrix into $\mathbf{\Sigma} = \mathbf{U \Lambda U^T}$, which can be re-written to :

$$\mathbf{\Sigma u}_i = \lambda_i \mathbf{u}_i \tag{3.5}$$

where $\mathbf{U}$ holds the orthonormal eigenvectors $\mathbf{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$, where $\mathbf{UU}^T = \mathcal{I}$ and diag$(\mathbf{\Lambda}) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ is the corresponding set of eigenvalues. Since $\mathbf{\Sigma}$ is always positive semi-definite, the eigenvalues are always non-negative. Geometrically this can be illustrated as in figure 3.1.

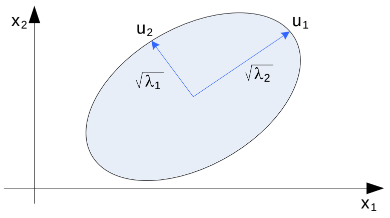

Figure 3.1: Geometric illustration of the eigenvalue decomposition of the data subset $\mathbf{X}$ for dimensions $n = 2$. Here the ellipse depicts the constant joint probability density $p(\mathbf{x_1}, \mathbf{x_2})$ and the eigenvectors u_1 and u_2 denotes the directions of maximum variance with lengths proportional to the eigenvalues, λ_1 and λ_2.

In the eigenvalue decomposition, the orthonormal eigenvectors $\mathbf{U}$ denotes the directions of maximum variance (called the *Principal Components*) and the corresponding eigenvalues are proportional to their lenghts, refer to [2].

Decorrelating the data corresponds to a coordinate transformation of the datapoints $x_{i,j}$ using the eigenvectors $\mathbf{U}$ with lengths $\mathbf{\Lambda}^{-\frac{1}{2}}$ as basis vectors, i.e. :

$$\mathbf{z}_i = \mathbf{\Lambda}^{-\frac{1}{2}} \mathbf{U}^T (\mathbf{x}_i - \mu) = \mathbf{V}(\mathbf{x}_i - \mu) \tag{3.6}$$

By sorting the eigenvectors wrt. eigenvalues in descending order, the dimension of the dataset $\mathbf{X}$ can easily be reduced to d dimensions by projecting the datapoints $x_{i,j}$ onto a subset of the eigenvectors with the d largest eigenvalues $\mathbf{U}_d = \{\mathbf{u}_1, \mathbf{u}_1, \ldots \mathbf{u}_d\}$, where $d < n$.

This eigenvalue decomposition can also be achieved by *Singular Value Decomposition* (SVD), where the data itself is decomposed into $\mathbf{X} = \mathbf{U \Gamma V}^T$. Here $\mathbf{U}$ (symmetric $n \times n$ matrix) and $\mathbf{V}$ (symmetric $m \times m$ matrix) holds the ordered orthonormal eigenvectors for the column and row vectors in $\mathbf{X}$ respectively and the diagonal elements of $\mathbf{\Gamma}$ ($n \times m$ matrix) holds the corresponding shared singular values, refer to [12] for details. From the SVD we can construct a covariance matrix $\mathcal{E}\{\mathbf{X}\mathbf{X}^T\} = \mathbf{U}\mathbf{\Gamma}\mathbf{V}^T\mathbf{V}\mathbf{\Gamma}\mathbf{U}^T = \mathbf{U}\mathbf{\Gamma}^2\mathbf{U}^T$, which is exactly the eigenvalue decomposition with $\mathbf{\Gamma}^2 = \mathbf{\Lambda}$. This means the decorrelation can be achieved by :

$$\mathbf{z}_i = \mathbf{\Gamma}^{-1}\mathbf{U}^T(\mathbf{x}_i - \mu) \tag{3.7}$$

where $\mathbf{\Gamma}$ and $\mathbf{U}$ can be reduced in size $d < n$ to obtain dimension reduction of the data.

To illustrate the effect of decorrelation, we have applied PCA to a small set of image data (1000 samples of size $12 \times 12 \times 3$), which will be described in details in section 6. Figure 3.2 show the scatterplots of the image data over dimension 1 and 2 and the corresponding 1st and 2nd principal axes after decorrelation.

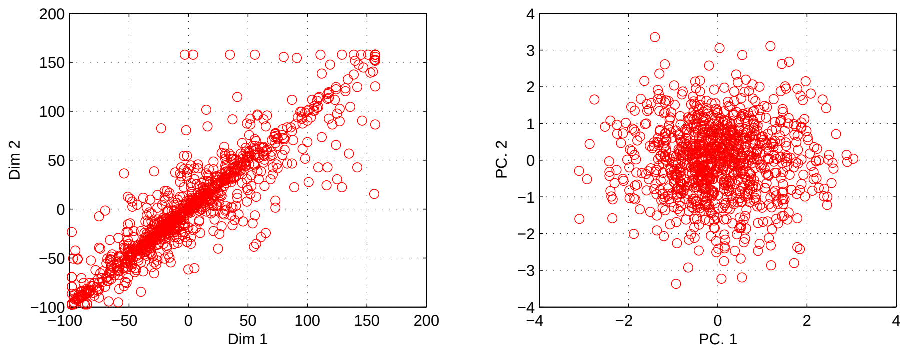

Figure 3.2: Left: Scatterplot of un-processed image data for 1st and 2nd dimension. *Right*: Scatterplot of 1st and 2nd principal axes after whitening.

The left illustration of figure 3.2 clearly shows how the image data is strongly correlated since information of one variable gives information of the other. The dynamic range for image data [0; 255] can also be seen as the scatterplot is bound from approximately $\sim$[-100; 150]. The right figure show how the data is uncorrelated after whitening with unity variance. Information of one variable now gives no information of the other.

The *Principal Component Analysis* (PCA) presented will also later be applied to image data in terms of sensory processing in section 6.2, as we shall see.

One of the limitations of PCA is that it is based on 2nd order information only. A different linear decomposition called *Independent Component Analysis* (ICA) exploit higher order statistics and is presented next.

4 Independent Component Analysis

In this section we give a short theoretical background for *Independent Component Analysis* (ICA). Initially we base the derivation of the algorithm on the *negentropy* and later relate this to the *max. likelihood* approach of ICA. Finally we focus on the practical issues of ICA, namely the usage of the `FastICA`[2] algorithm, refer to [5] for details.

In some applications a data vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, where x_i is a random variable, can best be described as a linear composition of statistically independent components, denoted $\mathbf{s} = \{s_1, s_2, \ldots, s_n\}$, i.e. their joint probability factorize into $p(s_1, s_2) = p(s_1)p(s_2)$. If we let the n dimensional vector $\mathbf{x}$ denote the data, ICA is described by the linear model :

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{4.1}$$

[2]Website: http://www.cs.helsinki.fi/u/ahyvarin/papers/fastica.shtml

where the latent variable $\mathbf{A}$ denotes the $n \times n$ mixing matrix and $\mathbf{s} = \{s_1, s_2, \ldots, s_n\}$ are the independent components. In the pursuit to decompose $\mathbf{x}$, both the components $\mathbf{s}$ and the mixing matrix $\mathbf{A}$ must be found under the constraint that the individual components s_i must be statistically independent.

Let us denote the whitened data by $\mathbf{z} = \{z_1, z_2, \ldots, z_n\}$ through linear transformation as given in (3.6), i.e. $\mathbf{z} = \mathbf{V}\mathbf{x}$, where $\mathbf{V}$ is an orthogonal whitening matrix. We can then rewrite the ICA linear transform to $\mathbf{z} = \mathbf{V}\mathbf{x} = \mathbf{V}\mathbf{A}\mathbf{s} = \tilde{\mathbf{A}}\mathbf{s}$, where the new mixing matrix $\tilde{\mathbf{A}}$ becomes orthogonal since $\mathcal{E}\{\tilde{\mathbf{A}}\tilde{\mathbf{A}}^{\mathbf{T}}\} = \mathcal{I}$. We can further isolate the independent components $\mathbf{s} = \tilde{\mathbf{A}}^{\mathbf{-1}}\mathbf{z} = \mathbf{W}\mathbf{z}$. This requires matrix inversion of $\tilde{\mathbf{A}}$, but this can be avoided by estimating $\mathbf{W}$ directly, as we shall see.

ICA is based on the theory from the *Central Limit Theorem*, which loosely states that the sum of two independent random variables are more gaussian than any of the two separately, i.e. $Gauss(x_1 + x_2) > Gauss(x_1)$. In other words, it is possible to locate an independent component s_i by maximizing non-gaussianity and for this purpose different measures for gaussianity exist: *Kurtosis* (4^{th} order cumulant), *Entropy*, *Mutual Information* to mention a few. In this context we shall use *negentropy*, a variation of entropy.

4.1 Negentropy

From information theory *entropy* can be used as a measure of '*gaussianity*' and is defined as

$$H(\mathbf{x}) = -\int p_x(\eta) log(p_x(\eta)) d\eta \tag{4.2}$$

where $\mathbf{x}$ is a random vector with probability density $p_x(\mathbf{x})$. Compared to kurtosis, the entropy is characterized by its robustness toward outliers, but can as shown be complicated to calculate.

An important property of the entropy is that for a gaussian distributed variable $\mathbf{x}$, the entropy $H(\mathbf{x})$ is largest compared to a non-gaussian variable $\mathbf{y}$ with same variance, i.e. $H(\mathbf{x}) > H(\mathbf{y})$. To ensure a non-negative measure for non-gaussianity, the *negentropy* can be used, given by $J(\mathbf{x}) = H(\mathbf{x}_{gauss}) - H(\mathbf{x})$.

However the negentropy $J(\mathbf{x})$ can be difficult to determine due to the probability density factor $p_x(\mathbf{x})$ in (4.2). The most classic approximation of the negentropy is expressed by:

$$J(\mathbf{x}) = \frac{1}{12}\mathcal{E}\{\mathbf{x}^3\}^2 + \frac{1}{48}\text{kurt}(\mathbf{x})^2 \tag{4.3}$$

where the *kurtosis* kurt($\mathbf{x}$) is the 4^{th} order cumulant given by $\text{kurt}(x) = \mathcal{E}\{x^4\} - 3(\mathcal{E}\{x^2\})^2$. Unfortunately kurtosis is sensitive to outliers and hence suffers from robustness. Instead the higher-order cumulants in (4.3) can be approximated by general non-quadratic functions, $G(\mathbf{x})$. The negentropy can then be written as :

$$J(\mathbf{x}) \propto [\ \mathcal{E}\{G(\mathbf{x})\} - \mathcal{E}\{G(\nu)\}\ ]^2 \tag{4.4}$$

where ν is gaussian variable with zero mean and unity variance. A typical choice for the non-quadratic function $G(\mathbf{x})$ could be $G(\mathbf{x}) = a_1^{-1}\log\cosh(a_1\mathbf{x})$ as an example, where $1 \leq a_1 \leq 2$. From (4.4) it is clear to see that the negentropy is maximized when $\mathcal{E}\{G(\mathbf{x})\}$ is maximized.

4.2 Maximizing Negentropy

Next we want to maximize the negentropy $J(\mathbf{w}^T\mathbf{z})$, where $\mathbf{z}$ is the whitened data, i.e. find a vector $\mathbf{w}$, which maximizes the negentropy and thus locates a independent component $\mathbf{s}_i$.

To maximize $\mathcal{E}\{G(\mathbf{w}^T\mathbf{z})\}$, we formulate a Lagrange-function $\mathcal{F}$ under the constraint of keeping unity variance $\mathcal{E}\{(\mathbf{w}^T\mathbf{z})^2\} = ||\mathbf{w}||^2 = 1$ to ensure whitened data, i.e. $\mathcal{F} = \mathcal{E}\{G(\mathbf{w}^T\mathbf{z})\} - \lambda\mathcal{E}\{(\mathbf{w}^T\mathbf{z})^2\} - 1$. Using the *Karush-Kuhn-Tucker* conditions (refer to [11] for details), we can express the optimal point in terms of the gradient and constraint as :

$$\frac{\partial\mathcal{F}}{\partial\mathbf{w}} = \mathcal{E}\{\mathbf{z}g(\mathbf{w}^T\mathbf{z}) + \beta\mathbf{w}\} = 0 \tag{4.5}$$

where $g(\cdot)$ is the derivative of the non-linear function $G(\cdot)$. This expression can be solved by applying Newton's method, i.e. as $\Delta\mathbf{w} = -[\frac{\partial^2\mathcal{F}}{\partial\mathbf{w}^2}]^{-1}\frac{\partial\mathcal{F}}{\partial\mathbf{w}}$ (refer to [6] for details) and thus the 2nd order derivate Hessian matrix $\mathcal{H}$ can be approximated as :

$$\mathcal{H} = \frac{\partial^2\mathcal{F}}{\partial\mathbf{w}^2} = \mathcal{E}\{\mathbf{z}\mathbf{z}^T g'(\mathbf{w}^T\mathbf{z})\} + \beta\mathcal{I} \tag{4.6}$$

This can be further reduced in order to simplify inversion of the Hessian matrix. As the data $\mathbf{z}$ is whitened, we can approximate $\mathcal{E}\{\mathbf{z}\mathbf{z}^T g'(\mathbf{w}^T\mathbf{z})\} \approx \mathcal{E}\{\mathbf{z}\mathbf{z}^T\}\mathcal{E}\{g'(\mathbf{w}^T\mathbf{z})\} = \mathcal{E}\{g'(\mathbf{w}^T\mathbf{z})\}\mathcal{I}$, i.e. $\mathcal{H}$ becomes diagonal and can easily be inverted. Finally the ICA update iteration can be expressed as :

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} = \mathbf{w} - \frac{\mathcal{E}\{\mathbf{z}g(\mathbf{w}^T\mathbf{z}) + \beta\mathbf{w}\}}{\mathcal{E}\{g'(\mathbf{w}^T\mathbf{z})\} + \beta} \tag{4.7}$$

where β can be computed from (4.5) as $\beta_i = -\mathcal{E}\{y_i g(y_i)\}$, where $\mathbf{y} = \mathbf{W}\mathbf{z}$.

Using this update iteration (4.7), we can find an independent component s_i when the direction of the vector $\mathbf{w}$ does not change significantly after an iteration. This approach only finds a single independent component s_i (a "one-unit" algorithm) so to locate all the independent components simultaneously as $\mathbf{W}$, we can rewrite (4.7) into :

$$\mathbf{W} \leftarrow \mathbf{W} + \mathrm{diag}(\alpha_i)[\mathrm{diag}(\beta_i) + \mathcal{E}\{\mathbf{g}(\mathbf{y})\mathbf{y}^\mathbf{T}\}]\mathbf{W} \tag{4.8}$$

where $\alpha = -1/(\mathcal{E}\{\mathbf{g}'(\mathbf{w}^\mathbf{T}\mathbf{z})\} + \beta_i)$. This does however not prevent us from locating the same component twice. As mentioned earlier in this section we saw that the mixing matrix $\tilde{\mathbf{A}}$ is orthogonal and thus $\mathbf{W}$ is also orthogonal since $\mathbf{W} = \tilde{\mathbf{A}}^{-1} = \tilde{\mathbf{A}}^T$. This means we can use (4.8) in an symmetric orthogonalization type approach, where $\mathbf{W}$ is orthogonalized after each iteration of (4.8) by :

$$\hat{\mathbf{W}} \leftarrow (\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W} \tag{4.9}$$

where the term $(\mathbf{W}\mathbf{W}^T)^{-1/2}$ is obtained from the eigenvalue decomposition $(\mathbf{W}\mathbf{W}^T)^{-1/2} = \mathbf{E}\Lambda^{-1/2}\mathbf{E}^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$. From this it is easy to see how $\hat{\mathbf{W}}$ is orthonormal as the covariance matrix becomes

$$\begin{aligned}\hat{\mathbf{W}}\hat{\mathbf{W}}^T &= ((\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W})((\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W})^T = (\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W}\mathbf{W}^T((\mathbf{W}\mathbf{W}^T)^{-1/2})^T \\ &= (\mathbf{W}\mathbf{W}^T)^{1/2}((\mathbf{W}\mathbf{W}^T)^{-1/2})^T = \mathbf{W}^{1/2}(\mathbf{W}^T)^{1/2}(\mathbf{W}^T)^{-1/2}\mathbf{W}^{-1/2} = \mathcal{I}\end{aligned}$$

Please refer to Hyvärinen [6] for a more detailed description of ICA and negentropy.

4.3 Max. Likelihood

In addition to negentropy ICA can also be performed by *maximum likelihood*, i.e. based on $p_x(\mathbf{x}|\mathbf{As})$, denoted $p_x(\mathbf{x})$. Since ICA is a linear transformation $\mathbf{x} = \mathbf{As}$, we exploit that the probability density can be written as $p_x(\mathbf{x}) = |\det(\mathbf{A^{-1}})|p_s(\mathbf{s})$, refer to appendix A.1 for proof.

Since the components s_i are independent, we can express the probability density $p_x(\mathbf{x})$ for the whitened data $\mathbf{z}$

$$p_z(\mathbf{z}) = |\det(\mathbf{W})|p_s(\mathbf{s}) = |\det(\mathbf{W})|\prod_i p_i(s_i) = |\det(\mathbf{W})|\prod_i p_i(\mathbf{Wz}) \tag{4.10}$$

where $\mathbf{W} = \tilde{\mathbf{A}}^{-1}$. If we evaluate the probability density p_i over T observations of $\mathbf{x}$ (or $\mathbf{z}$), we can express the log-likelihood ℓ as :

$$\ell(\mathbf{W}) = T\mathcal{E}\{\sum_{i=1}^{n}\log(p_i(\mathbf{w}_i^T\mathbf{z}_i))\} + T\log|\det\mathbf{W}| \tag{4.11}$$

where n is the length of a single observation vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. Under the assumption that the independent components s are whitened, it can be proved that the term $\log|\det(\mathbf{W})|$ is constant (refer to appendix A.2 for proof) and thus maximizing the likelihood $\ell(\mathbf{W})$ reduces to $\ell(\mathbf{W}) = T\mathcal{E}\{\sum_{i=1}^{n}\log(p_i(\mathbf{w}_i^T\mathbf{z}_i))\}$. If we further choose to approximate the probability density $p_i(s_i)$ with a non-linear function $\tilde{p}_i(s_i) \propto \cosh^{-1}(s_i)$, the gradient of the log becomes

$$g_i(s_i) = \frac{\partial}{\partial s_i}\log\tilde{p}_i(s_i) = \frac{\tilde{p}_i'(s_i)}{\tilde{p}_i(s_i)} = \frac{-\cosh^{-2}(s_i)\sinh(s_i)}{\cosh^{-1}(s_i)} = -\frac{\sinh(s_i)}{\cosh(s_i)} = -\tanh(s_i) \tag{4.12}$$

This means maximizing (4.11) can be reduced to the form $\ell'(\mathbf{W}) \approx \mathcal{E}\{g(\mathbf{w}^T\mathbf{z})\}$ and thus the update equation (4.8) can also be seen as an implementation of max. likelihood. It can further be compared to the Bell-Sejnowski approach given by $\Delta\mathbf{W} \propto (\mathcal{I} + \mathcal{E}\{\mathbf{g}(\mathbf{y})\mathbf{y}^T\})\mathbf{W}$ (refer to [1] and [6] for details).

This derivation of ICA given in (4.8) is included in the `FastICA` algorithm (refer to [5]) and will be used later for the simulations of ICA.

4.4 Discussion

The ICA model given in (4.1) can be interpreted in different ways. In solving a cocktail party problem, where x are the microphone-recorded signals, the independent sources s are in focus as they more or less remain constant if the microphone changes position (then the mixing matrix $\mathbf{A}$ changes as well). In terms of analyzing images, the mixing matrix $\mathbf{A}$ can be viewed as a constant codebook common for all images and the independent sources s are the corresponding encoding for each image. The mixing matrix $\mathbf{A}$ is therefore more interesting when analyzing image data.

Having given a short description of the theory behind ICA, a few observations can be seen. As maximizing non-gaussianity is used to locate independent components, the distribution of the individual data components can not be gaussian, since maximizing non-gaussianity then fails. This is also derived from the fact that ICA is based on higher order statistics and since a gaussian distribution is fully described by its mean μ and variance $\mathbf{\Sigma}$, ICA can not separate gaussian distributed data.

Note that the ICA decomposition may not guarantee completely independent components s unless the original data x was in fact a linear mixture of independent components. Still the ICA algorithm can give the linear axes (columns of mixing matrix $\mathbf{A}$) on which the data is the most independent. In terms of image data, the model seems to give successful results after all, as we shall see.

5 Non-Negative Matrix Factorization

In certain real world applications, an observed dataset $\mathbf{V}$ can best be described as generated exclusively from adding components and not necessarily being independent. A decomposition into non-negative parts may in these cases give a more meaningful interpretation conceptually. Classic examples would be an RGB image, where subcolors (R, G or B) are added constructively to generate a color or it could be wordcount from a document and so on.

Non-Negative Matrix Factorization (NNMF) seeks to factorize a matrix $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots \mathbf{v}_m\}$, where v_j is an n dimensional vector, into a low rank, sparse, non-negative approximation on the form :

$$\mathbf{V} \approx \mathbf{WH} \tag{5.1}$$

where the columns of $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \dots \mathbf{w}_r\}$ are the basis vectors and the $r \times m$ matrix $\mathbf{H}$ holds the encoding, where the desired rank r is chosen so that $(n+m)r < nm$. All matrices in (5.1) are under the constraint not to include any negative elements.

5.1 Approximating Factors

To find the non-negative factorizations of $\mathbf{W}$ and $\mathbf{H}$, the cost function based on Euclidean distance can be written as :

$$C_{LS} = \parallel \mathbf{V} - \mathbf{WH} \parallel^2 = \sum_{ij} (v_{ij} - (wh)_{ij})^2 \tag{5.2}$$

We can further formulate the update of $\mathbf{W}$ and $\mathbf{H}$ as to minimize (5.2) by using a gradient descent type approach expressed as:

$$H_{a\mu} \leftarrow H_{a\mu} - \Delta \frac{\partial C_{LS}}{\partial \mathbf{H}} = H_{a\mu} + \eta_{a\mu}[(W^T V)_{a\mu} - (W^T W H)_{a\mu}] \tag{5.3}$$

By setting $\eta_{a\mu} = \frac{H_{a\mu}}{(W^T W H)_{a\mu}}$ we can rewrite the update of $\mathbf{H}$ and similarly for $\mathbf{W}$ to :

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T W H)_{a\mu}} \qquad W_{ia} \leftarrow W_{ia} \frac{(V H^T)_{ia}}{(W H H^T)_{ia}} \tag{5.4}$$

where the numerically large term $(W^T W H)_{a\mu}$ and small term $(W^T V)_{a\mu}$ of the gradient in (5.3) end up in the nominator and denominator respectively of (5.4) and thus ensures a decrease of the cost function. Note that if $\mathbf{V}$ is non-negative (i.e. contains only non-negative elements), the update of both $\mathbf{W}$ and $\mathbf{H}$ also remain non-negative. Refer to [9] and [8] for a more detailed description on the update algorithm on NNMF.

Having presented the theory behind the factorizations PCA, ICA and NNMF, we proceed with simulation results.

6 Simulations

In this section, we present a set of simulations in order to analyze the decomposition of natural color images. A set of images (example is shown on the front page) by courtesy of *Patrik O. Hoyer* [3] are subject the following decompositions :

[3]http://www.cs.helsinki.fi/u/phoyer/

- *Principal Component Analysis*
- *Independent Component Analysis*
- *Non-Negative Matrix Factorization*

The different simulations are conducted in `MATLAB`, please refer to appendix B for a list of the code. Before we present the actual simulation results, a small description of the dataset is given.

6.1 Dataset

Considering the signal path of the visual information, the neurons in the visual cortex ultimately receives their input from the cones in the retina (color-sensitive part of the eye). The image data should resemble this format and thus choosing an image database consisting of 20 color images of size $384 \times 256 \times 3$ in RGB format seems reasonable. The dataset is constructed by sampling 2500 small image-patches of size 12×12 pixel from each image forming a $12 \times 12 \times 3 = 144 \times 3 = 432$ dimensional vector $\mathbf{x}$. This leads to a vector-space representation of a total of $2500 \times 20 = 50000$ column vectors $\mathbf{x}_j$ in a large data matrix $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m\}$, where $m = 50000$ and thus $\mathbf{X}$ becomes a 432×50000 data matrix.

Refer to appendix B.2 for details on the `MATLAB` implementation of constructing the dataset.

6.2 Principal Component Analysis

The first and most simple Principal Component decomposition described in section 3 is applied to the dataset matrix $\mathbf{X}$. Initially the mean is removed and the corresponding eigenvectors $\mathbf{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d\}$ are shown in figure 6.1 for $d = 432$ dimensions. In case of a dimension reduction $d < 432$, as we shall see later as part of ICA and NNMF.

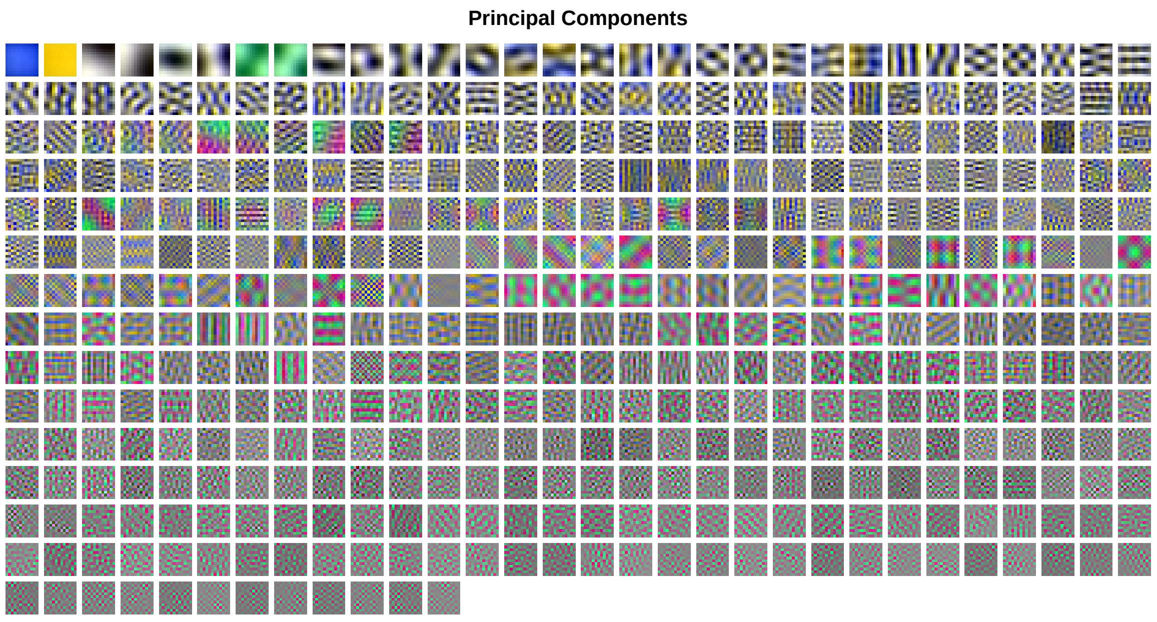

Figure 6.1: The principal components for all 432 dimensions sorted wrt. eigenvalues. Each patch is normalized to have max. dynamic range for all 3 channels (RGB) for improved visualization.

In figure 6.1 it is clear to see how some principal components are split up in yellow/blue and red/green patches. This is due to the maximization of variance in PCA, as yellow/blue and red/green are complementary colors.

In addition the spatial frequency is increasing as the corresponding eigenvalues (i.e. variance) decreases and hence the lowest eigenpatches has little contribution and can be considered noise. These features can be compared to 2D Fourier basis functions and also reveals how the eigenpatches does not show distinct sparseness, as features of high energy and high spatial frequency are mutually exclusive. Only very few patches with high eigenvalue affect only the luminance and has no color information. In addition a small gray background can be seen in most patches and is due to the mix of positive and negative contributions (or pixels) achieved with PCA leading to this zero mean-value. These results were also established by Hoyer & Hyvärinen [4].

To further analyze the color distribution, a few eigenpatches are shown in figure 6.2, where the colors are projected onto a plane in the colorspace, where $R + G + B = \text{constant}$, i.e. the luminance is ignored. This makes a hexagon shape, where the center is the all gray spot and the actual colors are spread toward the edges.

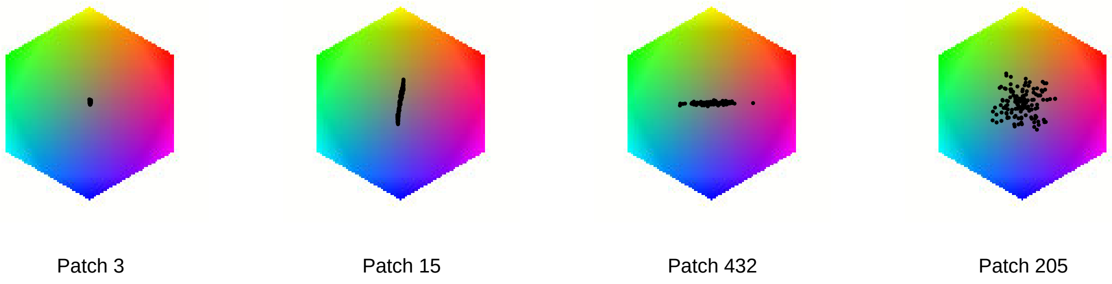

Figure 6.2: Color hexagons for different principal components

The first patch 3 has virtually no color information and holds only structural textures and luminance as can be seen in figure 6.1. Patch 15 and 432 reveals the maximization of variance in PCA, where some of the eigenpatches are split up in yellow/blue and red/green color information. Patch 432 also shows that even though the corresponding eigenvalue is the lowest, the patch still holds color and limited texture information and does not necessarily represent noise. The last patch 205 is very spread in its color information and mainly holds structural information, which can also be seen from figure 6.1.

In terms of reducing dimensions of a dataset $\mathbf{X}$, the eigenpatches with lowest eigenvalues are neglected in order to preserve max. amount of information. In this case, the lowest eigenpatches holds mostly yellow/blue and red/green features with high spatial frequency, but since the eigenvalues are low, these patches represent little information in dimension reduction. This phenomenon has also been observed in psychological experiments as stated by Hoyer & Hyvärinen [4] and will also be evident in our later simulations.

Next we present the results from our ICA simulations.

6.3 Independent Component Analysis

ICA based on higher order statistic is capable of extracting image patches, which are mutually independent as already discussed in section 4. Our derivation of the ICA algorithm assumes whitened data $\mathbf{Z}$ and in conjunction with whitening of the data matrix $\mathbf{X}$ into $\mathbf{Z}$, different sizes of dimension reduction is applied, $d = \{100, 160, 200, 250\}$. A similar processing of data is most likely also done in real neurons (Hoyer & Hyvärinen [4]). By reducing dimensions of $\mathbf{Z}$, we also lower the computational load and avoid amplifying directions with small variance, which would decrease the signal-to-noise ratio.

For the simulation we use the `FastICA` algorithm by Hyvärinen [5]. Part of this implementation includes a pre-processing step, which can both whiten and reduce dimensions of the data $\mathbf{X}$ by use of PCA as described in section 3. This means we can conveniently use `FastICA` for all our data-processing steps. We further choose the approximation of the log-gradient of the probability density function of the independent sources to be $g(s_i) = \tanh(s_i)$ and symmetric orthogonalization, as presented in section 4 in order to use the update equation given in (4.8). The function call and implementation of the ICA simulations are shown in appendix B.1.

As mentioned earlier, the columns of the mixing matrix $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_d\}$ can be seen as "universal" image components (more about this later) and for dimensions $d = 160$ and random initial mixing matrix $\mathbf{A}$, these ICA patches are shown in figure 6.3.

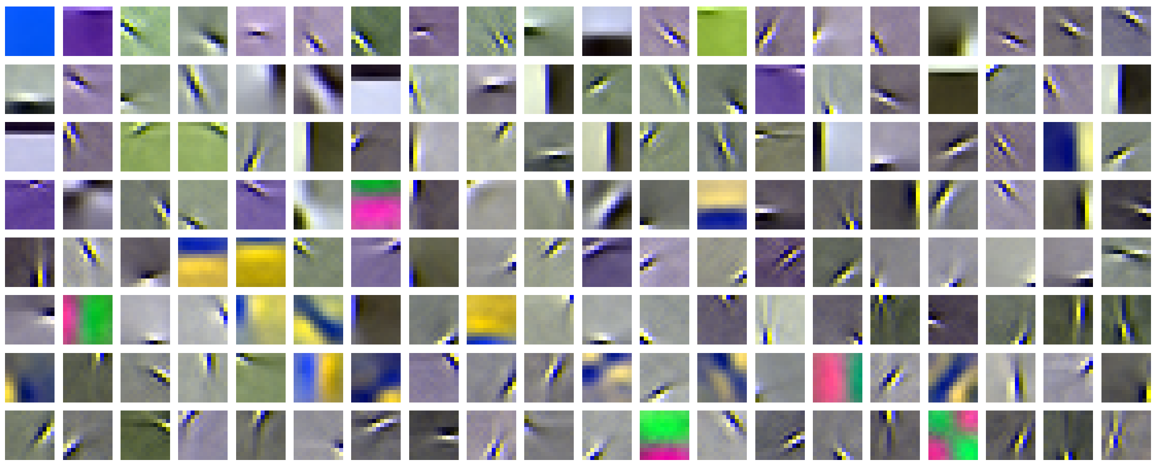

Figure 6.3: The image components for $d = 160$ dimensions sorted wrt. highest energy. Each patch is normalized to have max. dynamic range for all 3 channels (RGB) for improved visualization.

From the figure it is clear to see how the image patches has a large group resembling gabor filters. Such filters, modeled by a Guassian distribution modulated by a complex sine, being very sparse represent the very basic structural information of an image and can for instance be used in image clustering, refer to [7]. The gabor filters are all oriented with high bandpass spatial frequencies similar to the responds expected from the receptive fields in the simple cells of V1, as discussed earlier [13]. The dominance of these bright/dark patches is mostly due to the dimension reduction performed as pre-processing during whitening of the data [4] and also shows moderate sparseness compared to PCA with concentrated high energy and high spatial frequency. A similar result was found by Hoyer & Hyvärinen [4] using ICA and Hoyer [3] and Simoncelli & Olshausen [13] using a sparseness constrained model.

In addition a small set of color image patches both yellow/blue and red/green can be seen similar to the large group found under PCA. This only verifies that the max. variance directions are also found in ICA image decomposition. This group can be considered to form a color system incl. brightness, where the gabor-filter patches would form a structural group with no color information. In addition color patches are expected to hold low spatial frequency [4], which can also be seen from figure 6.3.

The fact that most patches are achromatic as they represent mainly brightness information, is also in alignment with most neurons in V1 respond equally to different colored stimulations [4]. Relatively new studies also suggests that color-sensitive neurons with orientation do exists, as opposed to they were believed not to be oriented and that such neurons are expected to have low spatial frequencies [4]. Such patches can also be found among the image components in figure 6.3.

A few selected patches are also shown in the color hexagon in figure 6.4. Initially the first two patches, 84 and 157, represent the same color patches as we saw under PCA, which holds very little textural information. Most of the Gabor-like patches has little color contribution evident by patch 11 shown in the figure. Still patch 3 with Gabor like structures also holds yellow/blue color information and are also ranked with high energy. In PCA a few patches were mix-patches, i.e. included color, brightness and structural information, whereas for ICA no mix-patches can be identified.

Next we present the simulation results from the non-negative decomposition.

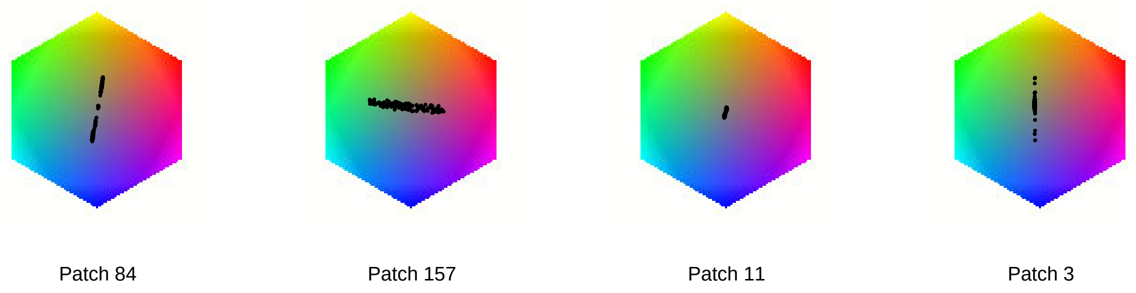

Figure 6.4: Color hexagons for different independent component patches.

6.4 Non-Negative Components

For Non-Negative Matrix Factorization, the dataset $\mathbf{X}$ is decomposed into non-negative parts as described in section 5. This means the dataset $\mathbf{X}$ can not contain negative elements and hence removing mean via whitening as a pre-processing step can not be performed. The NNMF algorithm implemented as `SNMF2D` (refer to IMM/Toolbox) is applied for different dimensions $d = \{100, 160, 200, 250\}$. Figure 6.5 shows the non-negative patches found for random initial basis matrix $\mathbf{W}$ and $d = 160$ dimensions.

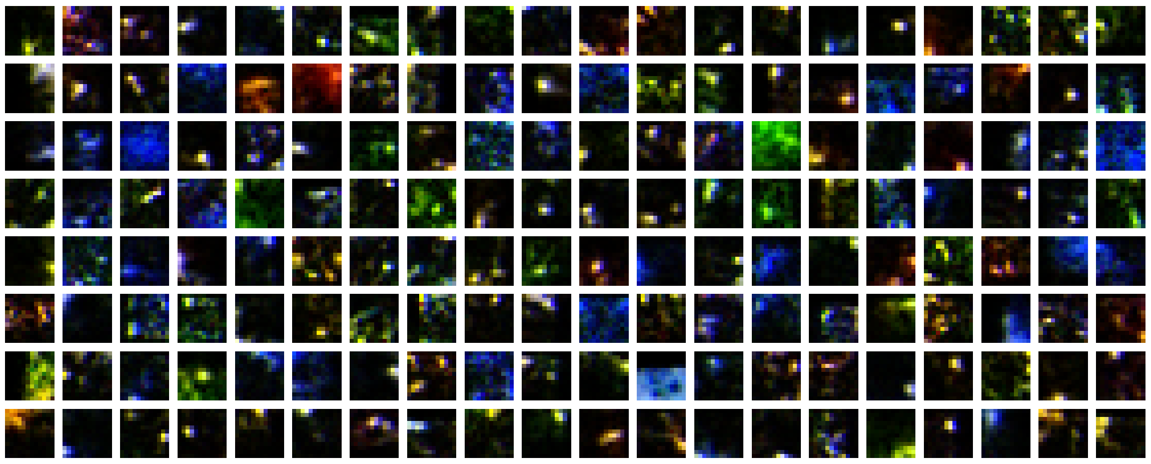

Figure 6.5: Non-negative components for $d = 160$ sorted wrt. highest energy. Each patch is normalized to have max. dynamic range for all 3 channels (RGB) for improved visualization.

The collection of non-negative patches are extremely sparse and can not directly be split up in a textural group and color group, as we could for PCA and ICA. No patches represent color or textural information only can be identified and all patches are unoriented. In general is hard to identify any perceptual interpretation of the non-negative patches. A few studies applying the non-negativity constrain on a linear model also shows similar results [3] [13].

The color hexagons for a few patches are shown in figure 6.6, which shows a typical picture of how the patches only holds color information for each channel, i.e. R, G or B, whereas only a few represent textural information alone as shown in the last hexagon. This R, G and B split up is due to the non-negative constraint in NNMF, since pixels from any patch can not be subtracted by any other and hence the sparsity. This can also be verified from figure 6.5.

Having presented the simulation results from the three different linear decompositions, we proceed with a small comparing analysis.

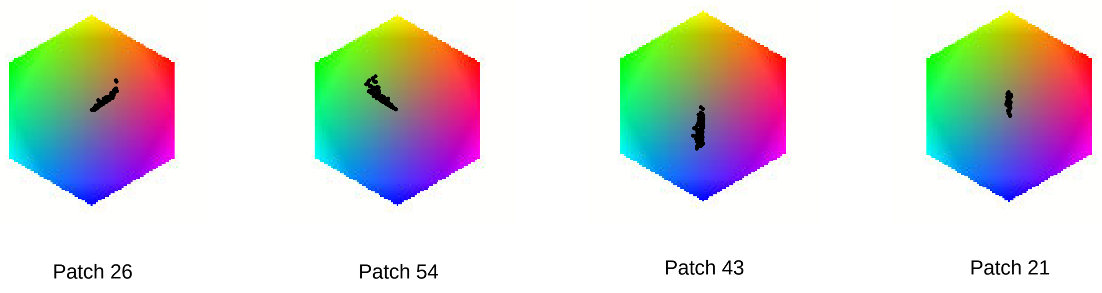

Figure 6.6: Color hexagons for different non-negative patches.

6.5 Comparing decompositions

Each of the decompositions are linear and can all be written on the form: $\mathbf{x} = \mathbf{A}\mathbf{s}$. The product $\mathbf{A}$ and $\mathbf{s}$ on the righthand side can then be interpreted as a codebook (matrix $\mathbf{A}$) and the corresponding encoding (vector $\mathbf{s}$) for each image patch $\mathbf{x}$.

6.5.1 Sparsity of codebook

One way to evaluate the decompositions is to analyze them with respect to sparsity of the codebook $\mathbf{A}$, i.e. how sparse are the fundamental patches for each decomposition. To evaluate the sparsity visually a patch from each decomposition is shown in figure 6.7 (left) as vectors, one for each color channel. In addition the amount of pixels below a certain threshold is counted for all normalized codebook patches in all three decompositions and shown in figure 6.7 (right).

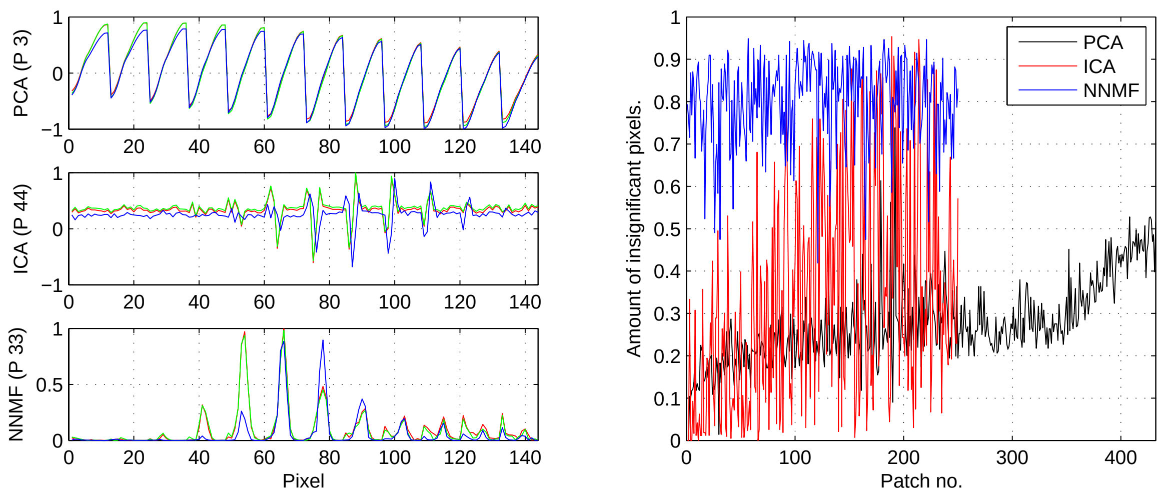

Figure 6.7: Left: Image for each decomposition shown as normalized vectors. *Right:* Share of insignificant pixels defined by a threshold of 10% of the max. subpixel strength for each patch.

Comparing the 3 vectors on the left image shows a typical patch for each decomposition. For ICA the gray area from a Gabor-like patch is clearly seen as an overall bias and the non-negative pixels from NNMF is also easily spotted. For all patches the right figure reveals how NNMF has the most sparse collection of codebook patches with an average of 80% even though the curves are extremely noisy. Considering that NNMF can not include negative values, this makes sense, since the non-negative approach leads to mostly black pixels to avoid 'adding too much' information. In contrast PCA and ICA can both subtract misplaced information added by other patches, which is not possible for NNMF.

6.5.2 Generalization

Viewing the decompositions as a codebook $\mathbf{A}$ with corresponding encoding vectors $\mathbf{x}$, as mentioned, means we can compare the decomposition with respect to generalization by evaluating the ability of the codebooks $\mathbf{A}$ to encode data.

For the analysis, we generate a permanent dataset or testset $n \times m$ $\mathbf{X}$ by sampling 13440 $12 \times 12 \times 3$ image patches randomly from the same image database as before, yielding an 432×13440 testset matrix. The `MATLAB` code for generating the testset is shown in appendix B.5.

For the purpose of encoding, we use the `SNMF2D` algorithm (refer to IMM/Toolbox) and prohibit the codebook matrix $\mathbf{A}$ from being updated. This means for the linear model $\mathbf{x} = \mathbf{A}\mathbf{s}$, only $\mathbf{s}$ is being updated during the iterations. As a pre-processing step, we ensure the data is non-negative by adding a bias equal to the minimum value. In addition the vectors in the codebooks $\mathbf{a}_i$ are all set to unity length, $|\mathbf{a}_i| = 1$ from the different decompositions, where $i = 1, 2, \ldots, d$.

In our simulations, we have used the range $d = \{100, 160, 200, 250\}$ as before and for ICA and NNMF we simply use the corresponding set from previous simulations. For PCA we extract the d largest eigenvectors (that is with the largest eigenvalue) in order to preserve information. To evaluate the encoding performance, we employ *Mean-Square-Error* (MSE) given by :

$$MSE = \frac{1}{nm} \sum_{j=1}^{m} (\mathbf{x}_j - \mathbf{A}\tilde{\mathbf{s}}_j)^2 \tag{6.1}$$

where $\tilde{\mathbf{S}} = \{\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_1, \ldots, \tilde{\mathbf{s}}_m\}$ is the estimated encoding. The `MATLAB` code for the simulations is shown in appendix B.6 and the results are shown below in figure 6.8.

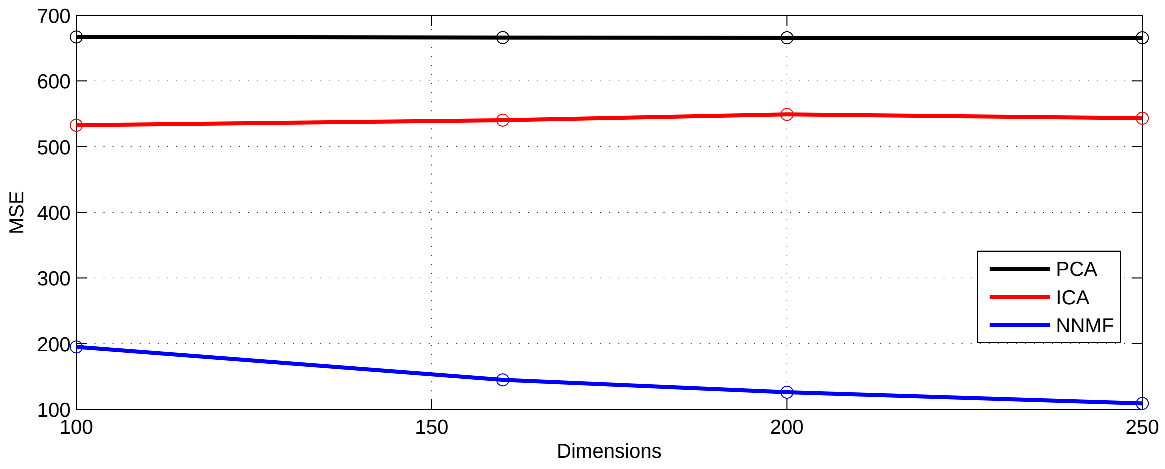

Figure 6.8: Error curves for the 3 decompositions.

The figure clearly shows a difference in the three decompositions and how NNMF has the best generalization with an decreasing error slope as the dimensions d increase. For PCA the MSE remain virtually the same, which proves that dimensions $>$100 for the eigenvectors contains insignificant information wrt. generalization. For ICA, the MSE curve is almost constant with small fluctuations considered insignificant. This suggests that the increase of dimensions for ICA does not contribute to better generalization of the data, as it clearly did for NNMF.

6.5.3 Sparsity of encoding vectors

In conjunction with the linear coding model it would be interesting to evaluate the sparseness of the encoding vectors $\mathbf{s}$ instead of just the codebook matrix $\mathbf{A}$. By calculating the mean of the share of pixels below 10% of all normalized coding vectors PCA, ICA and NMMF can be compared :

	$d = 100$	$d = 160$	$d = 200$	$d = 250$
PCA	92%	95%	96%	97%
ICA	89%	92%	95%	95%
NNMF	51%	56%	57%	59%

These figures clearly show how the ICA with the independence constraint and the PCA codebook incorporates a more sparse encoding than NNMF with the non-negativity constraint even though the corresponding mean-square-error for NNMF is considerably lower.

In general the analysis shows that if the codebook matrix $\mathbf{A}$ is sparse then the corresponding encoding vectors $\mathbf{s}$ are non-sparse and vice versa.

7 Conclusion

In our analysis we have given a short introduction to the neural processing of visual information from the retina to the basic part of the visual cortex from a information processing point of view. In the attempt to make a qualitative description of the different stages of processing, it is generally assumed that the early stages perform a type of decorrelation and redundancy reduction of the sensory input. Further studies suggest the subsequent stages of receptive fields of simple cells in V1 can best be modeled linearly with either independence or sparsity as the constraints [4] [3] [13].

We have presented the theoretical framework behind 3 major linear decompositions: *Principal Components Analysis* (PCA), *Independent Component Analysis* (ICA) and *Non-Negative Matrix Factorization* (NNMF) with the aim of analyzing these models with the neural processing described in the literature.

A set of simulations based on 20 natural images applied to PCA, ICA and NNMF was conducted. The whitening and hence decorrelation by PCA gave a set of oriented band-pass 2D fourier basis patches, where most patches could be grouped in either bright/dark, yellow/blue or red/green features. Similar results were also found by Hoyer & Hyvärinen [4].

Applying the ICA model showed sparse features resembling oriented gabor filters with high spatial frequency similar to the responds expected from the receptive fields in the simple cells in V1 [4]. From the NNMF model we achieved even more sparse features with no immediate perceptual interpretation, where the patches were split up in either R, G or B colors due to the non-negativity constraint [3].

In general a linear model $\mathbf{x} = \mathbf{A}\mathbf{s}$ can also be used for coding data using the matrix $\mathbf{A}$ as a codebook. Comparing the codebooks from the decompositions with respect to sparsity showed how NNMF clearly has the most sparse features used for encoding image data. This approach also lead to the evaluation of the coding efficiency using the different codebooks in a linear coding model. In this comparison the NNMF features gave the lowest mean-square-error and is thus the most efficient wrt. coding. Still a short analysis showed that the NNMF gave the least sparse encoding vectors compared to PCA and ICA.

Of all the theoretical framework and simulated data presented, this report only covers a very small area in this highly active research field. With the increasing computational power available we would expect future studies to cover even more complex neurological processing areas.

References

[1] Anthony J. Bell and Terrence J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation, 7:1129-1159*, 1995.

[2] Christopher M. Bishop. *Neural Network for Pattern Recongition*. Oxford Univerity Press, 1995.

[3] Patrik O. Hoyer. Modelling receptive fields with non-negative sparse coding. *Computational Neuroscience: Trends in Research*, 2003.

[4] Patrik O. Hoyer and Aapo Hyvärinen. Independent component analysis applied to feature extraction from colour and stereo images. *Network: Computation in Neural Systems*, 11, August, 2000.

[5] Aapo Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Trans. on Neural Networks*, 10(3), 1999.

[6] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. John Wiley & Sons Inc., 2001.

[7] T. Kolenda, L. K. Hansen, J. Larsen, and O. Winther. Independent component analysis for understanding multimedia content. In H. Bourlard, T. Adali, S. Bengio, J. Larsen, and S. Douglas, editors, *Proceedings of IEEE Workshop on Neural Networks for Signal Processing XII*, pages 757–766, Piscataway, New Jersey, 2002. IEEE Press. Martigny, Valais, Switzerland, Sept. 4-6, 2002.

[8] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 40:788–791, 1999.

[9] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. *Neural Information Processing Systems (NIPS)*, pages 556–562, 2000.

[10] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[11] K. Madsen, H.B. Nielsen, and O. Tingleff. *Optimization with Constraints, 2nd Edition*. DTU/IMM, March 2004.

[12] R. E. Madsen, L. K. Hansen, and O. Winther. Singular value decomposition and principal component analysis. Technical report, 2004.

[13] Eero P. Simoncelli and Bruno A. Olshausen. Natural image statistics and neural representation. *Annu. Rev. Neuroscience, 24:1193-216*, 2001.

A Appendix: Proofs

A.1 Probability density of a linear transformation

If we denote the ICA model by $\mathbf{x} = \mathbf{A}\mathbf{s}$ and assume a noise-free environment for the data $\mathbf{x}$, we can express the likelihood of the data $\mathbf{x}$ as

$$p_x(\mathbf{x}|\mathbf{A}, \mathbf{s}) = \delta(\mathbf{x} - \mathbf{A}\mathbf{s}) \tag{A.1}$$

where δ is the Dirac delta-function. We then marginalize the likelihood over the independent sources $\mathbf{s}$ and get

$$p_x(\mathbf{x}|\mathbf{A}) = \int p_x(\mathbf{x}|\mathbf{A}, \mathbf{s}) p_s(\mathbf{s})\ d\mathbf{s} = \int \delta(\mathbf{x} - \mathbf{A}\mathbf{s}) p_s(\mathbf{s})\ d\mathbf{s} \tag{A.2}$$

If we apply the rule for marginalizing over delta function $\int \delta(x - vs) f(s)\ ds = \frac{1}{v} f(x/v)$, we can rewrite (A.2)

$$p_x(\mathbf{x}|\mathbf{A}) = \frac{1}{|\det(\mathbf{A})|} p_s(\mathbf{A}^{-1}\mathbf{x}) = |\det(\mathbf{W})| p_s(\mathbf{s}) \tag{A.3}$$

where $\mathbf{W} = \mathbf{A}^{-1}$. This finally means we can express the probability density function of the data $p_x(\mathbf{x})$ based on the inverted mixing matrix $\mathbf{W}$ and the sources $\mathbf{s}$. Refer to MacKay [10] for further details.

A.2 ICA Log-likelihood function

If we denote the linear ICA model $\mathbf{x} = \mathbf{A}\mathbf{s}$, we can express the whitened data as $\mathbf{z} = \mathbf{V}\mathbf{x} = \mathbf{V}\mathbf{A}\mathbf{s} = \tilde{\mathbf{A}}\mathbf{s} \Leftrightarrow \mathbf{s} = \tilde{\mathbf{A}}^{-\mathbf{1}}\mathbf{z} = \mathbf{W}\mathbf{z}$.

As the components $\mathbf{s}$ are independent, they are also uncorrelated, since for two independent variables s_1 and s_2, we have $\mathcal{E}\{s_1 s_2\} = \mathcal{E}\{s_1\}\mathcal{E}\{s_2\}$ (refer to (3.1)), which implies uncorrelatedness as the covariance $\operatorname{cov}(s_1, s_2) = \mathcal{E}\{s_1 s_2\} - \mathcal{E}\{s_1\}\mathcal{E}\{s_2\} = 0$.

This means we can write the covariance matrix for the components as $\mathcal{E}\{\mathbf{s}\mathbf{s}^T\} = \mathcal{I} = \mathbf{W}\mathcal{E}\{\mathbf{z}\mathbf{z}^T\}\mathbf{W}^T$. If we take the determinant we get

$$\det \mathcal{I} = 1 = \det(\mathbf{W}\mathcal{E}\{\mathbf{z}\mathbf{z}^T\}\mathbf{W}^T) = (\det \mathbf{W})(\det \mathcal{E}\{\mathbf{z}\mathbf{z}^T\})(\det \mathbf{W}^T) \tag{A.4}$$

For a given dataset the covariance of the whitened data $\mathcal{E}\{\mathbf{z}\mathbf{z}^T\}$ is constant and hence from the righthand side of (A.4), we see that the term $\det(\mathbf{W})$ must also be constant (provided we whiten our data). Refer to Hyvärinen [6] for further details.

B Appendix: Code Section

B.1 Main function

```
% Main, for simulating PCA, ICA and NNMF

clear;

path(path, 'D:\Data\Studie\toolbox\ICA');
path(path, 'D:\Data\Studie\toolbox\immoptibox');
path(path, 'D:\Data\Studie\toolbox\FastICA');
path(path, 'D:\Data\Studie\toolbox\SNMF2D');

% Initialize
K  = 20;                     % No. of images
N  = 25;                   % No. of patches pr. image
dX = 12; dY = 12; dC = 3;    % Image patch size

% Generate image vectors
b = form_image_vectors(K, N, dX, dY, dC);

% PCA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Performing␣PCA...');

% Subtract mean
b = b -  repmat(mean(b,2), 1, size(b,2));

% De-correlate
cov_b  = cov(b',1);                   % Calc. cor.matrix
[E, D] = eig(cov_b);                  % Calc. eigenvalues of cor.matrix
X      = inv(sqrt(D)) * E' * b;       % Whiten

U = E(:,end:-1:1);                    % Reverse order of eigenvectors
save PCA_20_2500_c E D;
fprintf('Ok\n');

% View patches
figure(10);
show_patch(U, dC, 30, 2);

% Color hexagon
figure(11)
  subplot(141), show_colorhex(U(:,3));
  xlabel('Patch␣3');
  subplot(142), show_colorhex(U(:,15));
  xlabel('Patch␣15');
  subplot(143), show_colorhex(U(:,432));
  xlabel('Patch␣432');
  subplot(144), show_colorhex(U(:,205));
  xlabel('Patch␣205');

% Scatter plots
figure(12)
  subplot(121), plot(b(1,:), b(2,:), 'ro'); grid;
  xlabel('Dim␣1'); ylabel('Dim␣2');
  subplot(122), plot(X(1,:), X(2,:), 'ro'); grid;
  xlabel('PC.␣1'); ylabel('PC.␣2');
```

```
% ICA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Performing␣ICA...');
% Perform ICA
[b_ICA, A_ICA, W] = fastica(b, 'approach', 'symm', 'g', 'tanh', 'mu', ...
                            1-1e-9, 'lastEig', 160);

A_ICA = sort_patch(A_ICA);    % Sort patches wrt. highest energy

save ICA_ML_160 b_ICA A_ICA;
fprintf('Ok\n');

% View patches
figure(20);
show_patch(A_ICA, dC, 20, 2);

% Color hexagon
figure(21)
  subplot(141), show_colorhex(A_ICA(:,84));
  xlabel('Patch␣84');
  subplot(142), show_colorhex(A_ICA(:,157));
  xlabel('Patch␣157');
  subplot(143), show_colorhex(A_ICA(:,11));
  xlabel('Patch␣11');
  subplot(144), show_colorhex(A_ICA(:,3));
  xlabel('Patch␣3');


% NNMF %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('Performing␣NNMF...');

% Non-Negative Matrix Factorization
[W,H] = SNMF2D(b, 160);
W     = sort_patch(W);                     % Sort patches wrt. highest energy

save NMF_160 W H;
fprintf('Ok\n');

% View image patches
figure(30);
show_patch(W, dC, 20, 2);

% Color hexagon
figure(31)
  subplot(141), show_colorhex(W(:,26));
  xlabel('Patch␣26');
  subplot(142), show_colorhex(W(:,54));
  xlabel('Patch␣54');
  subplot(143), show_colorhex(W(:,43));
  xlabel('Patch␣43');
  subplot(144), show_colorhex(W(:,21));
  xlabel('Patch␣21');


%%%%%% Comparison analysis

% Load patches
```

```
load PCA_20_2500_c;
load ICA_ML_250;
load NMF_250;

% Threshold set to 10%
thresh = 0.1;

% Analyze PCA
for i = 1:size(U,2)
  v_PCA    = U(:,i)/max(abs(U(:,i)));                    % Unity scaling
  s_PCA(i) = length(find(abs(v_PCA) < thresh))/432;      % Share under thresh.
end

% Analyze ICA and NNMF
for i = 1:250
  v_ICA     = A_ICA(:,i)/max(abs(A_ICA(:,i)));           % Unity Scale
  s_ICA(i)  = length(find(abs(v_ICA) < thresh))/432;     % Share under thresh.
  v_NNMF    = W(:,i)/max(abs(W(:,i)));                   % Unity Scale
  s_NNMF(i) = length(find(abs(v_NNMF) < thresh))/432;    % Share under thresh.
end

% Pick out a typical sparse vector for each
load ICA_ML_160;
load NMF_160;
v_PCA  = U(:,3)/max(abs(U(:,3)));
v_ICA  = A_ICA(:,90)/max(abs(A_ICA(:,90)));
v_NNMF = W(:,33)/max(abs(W(:,33)));

% Show patch as line
figure(40)
  subplot(321), plot(v_PCA(1:144),   'r'); hold on;
  subplot(321), plot(v_PCA(145:288), 'g');
  subplot(321), plot(v_PCA(289:432), 'b'); hold off;
  grid; ylabel('PCA␣(P␣3)'); axis([0 144 -1 1]);

  subplot(323), plot(v_ICA(1:144), 'r'); hold on;
  subplot(323), plot(v_ICA(145:288), 'g');
  subplot(323), plot(v_ICA(289:432), 'b'); hold off;
  grid; ylabel('ICA␣(P␣44)'); axis([0 144 -1 1]);

  subplot(325), plot(v_NNMF(1:144), 'r'); hold on;
  subplot(325), plot(v_NNMF(145:288), 'g');
  subplot(325), plot(v_NNMF(289:432), 'b'); hold off;
  grid; ylabel('NNMF␣(P␣33)'); axis([0 144 0 1]);
  xlabel('Pixel');

  % Sparsity
  subplot(122), plot(s_PCA, 'k'); hold on;
  subplot(122), plot(s_ICA, 'r');
  subplot(122), plot(s_NNMF, 'b');
  hold off; grid; axis([0 432 0 1]);
  xlabel('Patch␣no.'); ylabel('Amount␣of␣insignificant␣pixels.');
  legend('PCA', 'ICA', 'NNMF');
```

B.2 Generate image patches

```
function [b] = form_image_vectors(K, N, dX, dY, dC)

% Initialize
b = zeros(dX*dY*dC, N*K);

% Load images one-by-one
for i = 1:K
  % Load image
  filename = ['Images\' num2str(i) '.tiff' ];
  Image = imread(filename);
  [yImage xImage cImage] = size(Image);

  for p = 1:N
    % Generate random x and y position for patch
    xPos = ceil((xImage-dX)*rand);
    yPos = ceil((yImage-dY)*rand);

    % Extract patch
    patch = Image(yPos:yPos+dY-1, xPos:xPos+dX-1, 1:dC);

    % Convert to vector
    b(:,N*(i-1)+p) = reshape(patch, dX*dY*dC,1);
  end
end
```

B.3 Sorting patches wrt. energy

```
function A = sort_patch(A)

% Sort ICA components by largest energy

% Initialize
A_ORI = A;
E     = sum(A_ORI.^2);

% Loop over all patches
for i = 1:size(A, 2)
  [Y I] = max(E);            % Identify the largest
  A(:,i) = A_ORI(:,I);       % Copy it to new var.
  E(I) = 0;                  % Set origal to zero to avoid being found again
end
```

B.4 Viewing image patches

```
function show_patch(v, dC, X, d)

% Dimensions
dY = sqrt(size(v,1)/dC);
dX = sqrt(size(v,1)/dC);

k = 1; dx = d; dy = d;
N = size(v,2);
Y = ceil(N/X);
```

```
% Initialize
View   = 255*ones(dY*Y+d*(Y+1), dX*X+d*(X+1), dC, 'uint8');

% Pre-process
vv = v   -  repmat(min(v),  size(v,1), 1);   % Add mean to have all non-neg. values
vv = vv ./ repmat(max(vv), size(vv,1), 1);  % Scale to have unity values
vv = uint8(255*vv);                          % Convert to uint8 for plotting

% Convert vectors to image patches
for k = 1:N
  xPos = (dX+d)*mod(k-1, X)+1  + d;
  yPos = (dY+d)*floor((k-1)/X) + d;

  patch = reshape(vv(:,k), dX, dY, dC);
  View(yPos:yPos+dY-1, xPos:xPos+dX-1, 1:dC) = patch;
end

% Finally show patches as image
imshow(View);
```

B.5 Generate testset

```
% Generates image patches from all images without overlap

clear;

% Intialize
dX = 12; dY = 12; dC = 3;           % Size of patches
K = 20;                             % No. of images
p = 1;

% Load 1st image before loop
Image = imread('Images\1.tiff');
[yImage xImage cImage] = size(Image);

% Initialize
N = floor(xImage/dX) * floor(yImage/dY);
b = zeros(dX*dY*dC, K*N);

% Load images
for i = 1:K
  % Load i'th image
  filename = ['Images\' num2str(i) '.tiff' ];
  Image = imread(filename);
  [yImage xImage cImage] = size(Image);

  % Loop over x and y and extract patches
  for y = 0:floor(yImage/dY)-1
    for x = 0:floor(xImage/dX)-1
      yPos = dY*y+1;
      xPos = dX*x+1;

      % Extract patch
      patch = Image(yPos:yPos+dY-1, xPos:xPos+dX-1, 1:dC);
      % Convert to vector
      b(:,p) = reshape(patch, dX*dY*dC,1);
      p = p + 1;
    end
  end
end

% Finally save to file
save All_image_patch b;
```

B.6 Encoding testset with codebook

```
%

clear;
path(path, '/gbar/bohr/home1/s03/s030192/toolbox/ICA');
path(path, '/gbar/bohr/home1/s03/s030192/toolbox/immoptibox');
path(path, '/gbar/bohr/home1/s03/s030192/toolbox/FastICA');
path(path, '/gbar/bohr/home1/s03/s030192/toolbox/SNMF2D');

% Initialize
K = 20;                        % No. of images
N = 2500;                      % No. of patches pr. image
dX = 12; dY = 12; dC = 3;      % Image patch size

% Load image data
load All_image_patch;
load ICA_ML_250;

% Non-Negative Matrix Factorization
fprintf('Starting␣Encoding...\n');

% Pre-process
W = A_ICA;
W = W - repmat(min(W), 432, 1);          % Ensure non-negative data
% W = W ./ repmat(max(W), 432, 1);       % Unity length already done from
                                         % FastICA algorithm.
% Arguments to NNMF algorithm
opt.W          = W;
opt.const_WorH = 1;              % Do not update W, constant codebook
opt. maxiter   = 10000;
[W,H] = SNMF2D(b, 250, opt);

save _encode_ICA_250 H W;

fprintf('Encoding␣finished...\n');
```

B.7 Sparseness and Mean-Square-Error

```
clear;

% Initialize
d = [100, 160, 200, 250];        % Dimensions
```

```
load All_image_patch;            % Load image data

% Mean-Square-Error for PCA
for i = 1:length(d)
  load(['encode/_encode_PCA_' num2str(d(i)) ]);
  diff = b - W*H;
  e_PCA(i) = sum(sum(diff.^2))/(size(diff,1)*size(diff,2));
  clear W H;
end

% Mean-Square-Error for ICA
for i = 1:length(d)
  load(['encode/_encode_ICA_' num2str(d(i)) ]);
  diff = b - W*H;
  e_ICA(i) = sum(sum(diff.^2))/(size(diff,1)*size(diff,2));
  clear W H;
end

% Mean-Square-Error for NNMF
for i = 1:length(d)
  load(['NMF_' num2str(d(i)) ]);
  load(['encode/_encode_NNMF_' num2str(d(i)) ]);
  diff = b - W*H;
  e_NNMF(i) = sum(sum(diff.^2))/(size(diff,1)*size(diff,2));
  clear W H;
end

% Plot MSE curves
figure(1)
  plot(d, e_PCA, 'k',  'linewidth', 2); hold on;
  plot(d, e_ICA, 'r',  'linewidth', 2);
  plot(d, e_NNMF, 'b', 'linewidth', 2);
  plot(d, e_PCA, 'ko');
  plot(d, e_ICA, 'ro');
  plot(d, e_NNMF, 'bo'); hold off;
  legend('PCA', 'ICA', 'NNMF'); grid;
  xlabel('Dimensions,␣d'); ylabel('MSE');


% Estimate sparsity
thresh = 0.1;           % Threshold of 10%

for i = 1:length(d)
  % For ICA
  load(['ICA_ML_' num2str(d(i))]);
  for j = 1:length(b_ICA)
    s_ICA(j) = length(find(abs(b_ICA(:,j)) < thresh))/size(b_ICA,1);
  end

  % For NNMF
  load(['NMF_' num2str(d(i))]);
  for j = 1:length(b_ICA)
    s_NNMF(j) = length(find(abs(H(:,j)) < thresh))/size(H,1);
  end

  % Collect mean sparsity
  [mean(s_ICA) mean(s_NNMF)]
end
```

```matlab
% Sparsity
thresh = 0.1;

for i = 1:length(d)
  % Sparsity for PCA
  load(['encode/_encode_PCA_' num2str(d(i))]); clear w;
  H = H -  repmat(min(H), size(H,1), 1);         % Normalize
  H = H ./ repmat(max(H), size(H,1), 1);
  for j = 1:size(H,2)
    s_PCA(j) = length(find(abs(H(:,j)) < thresh))/size(H,1);
  end

  % Sparsity for PCA
  load(['encode/_encode_ICA_' num2str(d(i))]); clear w;
  H = H -  repmat(min(H), size(H,1), 1);         % Normalize
  H = H ./ repmat(max(H), size(H,1), 1);
  for j = 1:size(H,2)
    s_ICA(j) = length(find(abs(H(:,j)) < thresh))/size(H,1);
  end

  % Sparsity for NNMF
  load(['encode/_encode_NNMF_' num2str(d(i))]); clear w;
  H = H -  repmat(min(H), size(H,1), 1);         % Normalize
  H = H ./ repmat(max(H), size(H,1), 1);
  for j = 1:size(H,2)
    s_NNMF(j) = length(find(abs(H(:,j)) < thresh))/size(H,1);
  end

  % Collect mean sparsity
  [mean(s_PCA) mean(s_ICA) mean(s_NNMF)]
end
```

B.8 Viewing color hexagon

This MATLAB function was kindly provided by *Patrik O. Hoyer* [4].

```matlab
function colorhex( a, dispmode, fname )
% colorhex - generates a color-hexagon plot
%
% SYNTAX:
% colorhex( a, dispmode, fname );
%
% If a is a column vector, then it is assumed that this is a basis vector,
% with first the red components, then the green, and finally the blue.
% If a is a matrix with three columns then the columns are assumed to
% contain the red/green/blue values.

global dogray;
global hexsize;

hexsize = 128;

if strcmp(dispmode,'gray')==1
  dogray = 1;
```

[4]http://www.cs.helsinki.fi/u/phoyer/

```
else
  dogray = 0;
end

if nargin==2,
  fname = [];
end


%-----------------------------------
% First, we generate the background
%-----------------------------------

if strcmp(dispmode,'none')==0

  % Some type of plot coming up
%  clf; cla;
%  iptsetpref('ImshowBorder','tight');
%  subplot('position',[0,0,1,1]);

end

if strcmp(dispmode,'color') | strcmp(dispmode,'gray')

  if 0
    x = rand(3,10000)*2-1;
    x = [-1 -1 -1; -1 -1 1; -1 1 -1; -1 1 1; 1 -1 -1; 1 -1 1; 1 1 -1; 1 1 1]';
    x = x-ones(3,1)*mean(x);
    y = xyz2hex(rgb2xyz(x));
    z = xyz2rgb(hex2xyz(y));
    keyboard;
  end

  % Color plot
  [X,Y] = meshgrid( 1:hexsize , 1:hexsize );
  XY(:,:,1) = X;
  XY(:,:,2) = Y;

  XYZ = hex2xyz(XY);
  RGB = xyz2rgb(XYZ);

  % Some may be outside range, to those we add k*[1 1 1] to get inside
  % the colour cube.
  dims = size(RGB);
  rgb(1,:) = reshape(RGB(:,:,1),[1 dims(1)*dims(2)]);
  rgb(2,:) = reshape(RGB(:,:,2),[1 dims(1)*dims(2)]);
  rgb(3,:) = reshape(RGB(:,:,3),[1 dims(1)*dims(2)]);

  inds1 = find(max(rgb)>1);
  vals1 = max(rgb(:,inds1));
  rgb(:,inds1) = rgb(:,inds1)-ones(3,1)*(vals1-1);

  inds2 = find(min(rgb)<-1);
  vals2 = min(rgb(:,inds2));
  rgb(:,inds2) = rgb(:,inds2)-ones(3,1)*(vals2+1);

  RGB(:,:,1) = reshape(rgb(1,:),[dims(1) dims(2)]);
  RGB(:,:,2) = reshape(rgb(2,:),[dims(1) dims(2)]);
  RGB(:,:,3) = reshape(rgb(3,:),[dims(1) dims(2)]);

  RGB = (RGB+1)/2;
%  keyboard;

  imshow(RGB);
%  truesize;
%  axis equal;
  axis off;

end

%-----------------------------------
% Next, we plot the pixels
%-----------------------------------

if isempty(a)==0

  % Normalize so maximum absolute value is 1 (same as in visual.m)
  a = a/max(max(abs(a)));

  if size(a,2)==1
    a = reshape(a,[size(a,1)/3 3]);
  end
  a = a';

  xyz = rgb2xyz(a);
  hexspots = xyz2hex(xyz);

  if strcmp(dispmode,'none')==0
    hold on; plot(hexspots(1,:),hexspots(2,:),'k.'); hold off;
  end
end

if isempty(fname)==0

  set(gcf,'PaperPositionMode','auto');
  if strcmp(dispmode,'color')
%    print( gcf, '-depsc', fname );
    print( gcf, '-dtiff', '-r300', fname );
  elseif strcmp(dispmode,'gray')
    print( gcf, '-deps', fname );
  end

end

return;

%---------------------------------------------------------------------------
%                                rgb / xyz
%---------------------------------------------------------------------------

function xyz = rgb2xyz( rgb )
xyz = rgbANDxyz( rgb, 'rgb2xyz');
return

function rgb = xyz2rgb( xyz )
rgb = rgbANDxyz( xyz, 'xyz2rgb');
```

```
return

function vout = rgbANDxyz( vin, whichway )

% Re-arrange data if necessary
if size(vin,3)>1
  mode = 1;
  dims = size(vin);
  vintemp = vin;
  clear vin;
  vin(1,:) = reshape(vintemp(:,:,1),[1 dims(1)*dims(2)]);
  vin(2,:) = reshape(vintemp(:,:,2),[1 dims(1)*dims(2)]);
  vin(3,:) = reshape(vintemp(:,:,3),[1 dims(1)*dims(2)]);
else
  mode = 0;
end

% This is the transformation matrix
rg = [ 1 -1 0]/sqrt(2);
by = [-1 -1 2]/sqrt(6);
bw = [ 1  1 1]/sqrt(3);
C = [rg; by; bw];

% Transform (simple linear transform)
if strcmp(whichway,'rgb2xyz')
  vout = C*vin;
elseif strcmp(whichway,'xyz2rgb')
  vout = inv(C)*vin;
else
  error('Select␣one␣way␣or␣the␣other!');
end

% Make data into old format again
if mode==1
  vouttemp = vout;
  clear vout;
  vout(:,:,1) = reshape(vouttemp(1,:),[dims(1) dims(2)]);
  vout(:,:,2) = reshape(vouttemp(2,:),[dims(1) dims(2)]);
  vout(:,:,3) = reshape(vouttemp(3,:),[dims(1) dims(2)]);
end

return

%--------------------------------------------------------------------------
%                               xyz / hex
%--------------------------------------------------------------------------

function vout = xyz2hex( vin )

% Universal constant!
global hexsize;

% Re-arrange data if necessary
if size(vin,3)>1
  mode = 1;
  dims = size(vin);
  vintemp = vin;
  clear vin;
  vin(1,:) = reshape(vintemp(:,:,1),[1 dims(1)*dims(2)]);
  vin(2,:) = reshape(vintemp(:,:,2),[1 dims(1)*dims(2)]);
  vin(3,:) = reshape(vintemp(:,:,3),[1 dims(1)*dims(2)]);
else
  mode = 0;
end

% Transform (direct projection)
vout = (vin(1:2,:)/4+0.5)*hexsize;

% Make data into old format again
if mode==1
  vouttemp = vout;
  clear vout;
  vout(:,:,1) = reshape(vouttemp(1,:),[dims(1) dims(2)]);
  vout(:,:,2) = reshape(vouttemp(2,:),[dims(1) dims(2)]);
end

return

function vout = hex2xyz( vin )

global dogray;

% Universal constant!
global hexsize;

% Re-arrange data if necessary
if size(vin,3)>1
  mode = 1;
  dims = size(vin);
  vintemp = vin;
  clear vin;
  vin(1,:) = reshape(vintemp(:,:,1),[1 dims(1)*dims(2)]);
  vin(2,:) = reshape(vintemp(:,:,2),[1 dims(1)*dims(2)]);
else
  mode = 0;
end

p1 = [0 1.6330];
p2 = [1.4142 0.8165];

p4 = p1-p2;
p4 = [p4(2) -p4(1)];
p4 = p4/norm(p4);
p4 = (p2*p4')*p4;
p4 = [p4 0];
p5 = [-p4(1) p4(2) 0];
p6 = [1.4142 0 0];

p4norm = norm(p4);
p5norm = norm(p5);
p6norm = norm(p6);

p4 = p4/p4norm;
p5 = p5/p5norm;
p6 = p6/p6norm;
```

```
%p4norm = p4norm*0.90;
%p5norm = p5norm*0.90;
%p6norm = p6norm*0.90;

% Inverse transform
vout = [(vin/hexsize-0.5)*4; zeros(1,size(vin,2))];

% Set values outside hexagon to white
outindices = find(abs(p4*vout)>=p4norm | abs(p5*vout)>=p5norm | ...
                  abs(p6*vout)>=p6norm);

inindices = find(abs(p4*vout)<p4norm & abs(p5*vout)<p5norm & ...
                 abs(p6*vout)<p6norm);

vout(1,outindices)=0;
vout(2,outindices)=0;
vout(3,outindices)=sqrt(3);

% If doing gray, set everything to gray!
if dogray
  vout(:,inindices)=0;
end

% Make data into old format again
if mode==1
  vouttemp = vout;
  clear vout;
  vout(:,:,1) = reshape(vouttemp(1,:),[dims(1) dims(2)]);
  vout(:,:,2) = reshape(vouttemp(2,:),[dims(1) dims(2)]);
  vout(:,:,3) = reshape(vouttemp(3,:),[dims(1) dims(2)]);
end

return
```