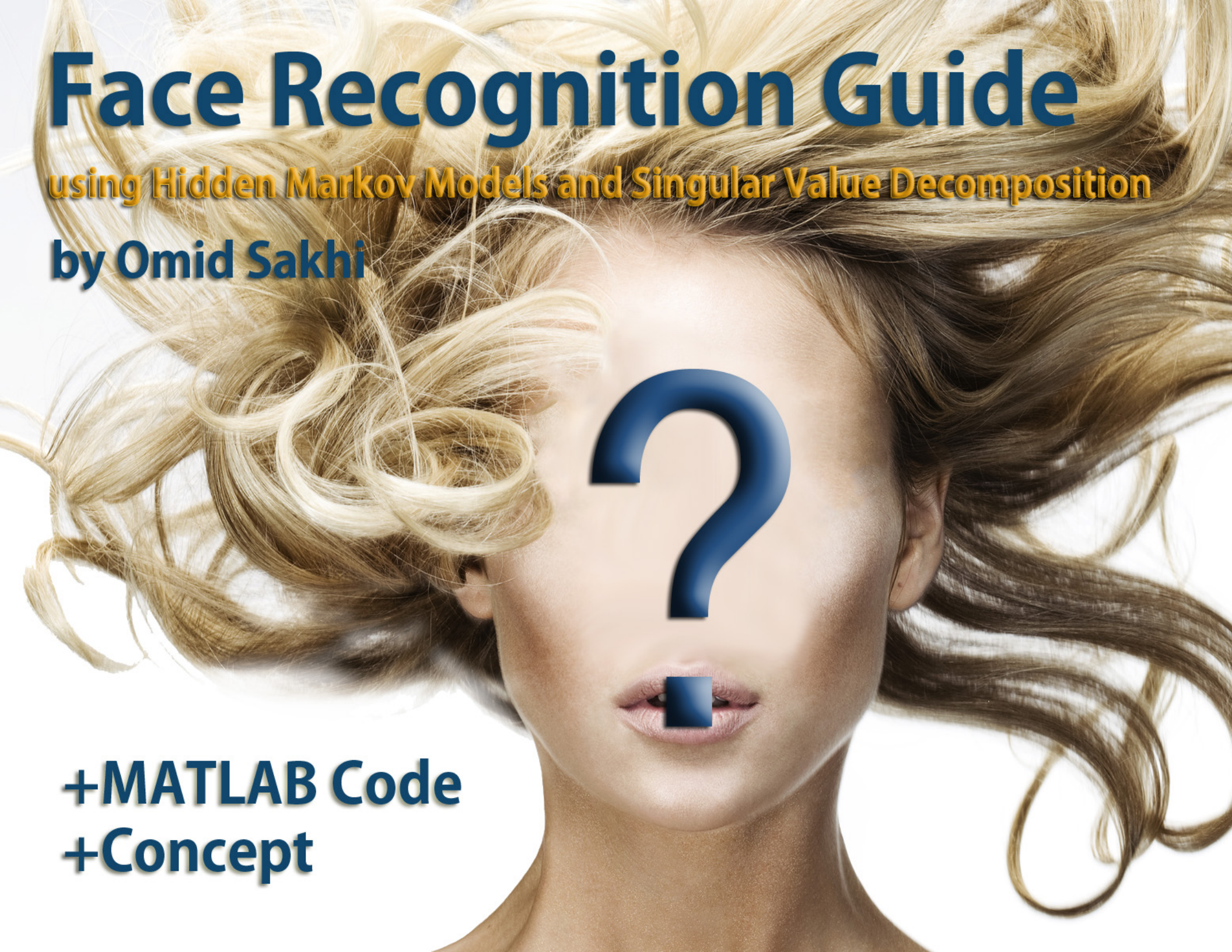
# Face Recognition Guide

## using Hidden Markov Models and Singular Value Decomposition

by Omid Sakhi



+MATLAB Code
+Concept

# Face Recognition Guide +Code +Concept

## for MATLAB

A Practical Guide to Face Recognition using
Hidden Markov Models
and
Singular Value Decomposition
+
line by line description of the code

First Edition - build 1.0 - 17 December 2012

**by Omid Sakhi**

# 1 Introduction

Well! I guess Hi!

I know this is not a common introduction for an ebook. But this is not a common ebook. Most papers that are published for face recognition are mostly technical, with many formulas, and require good mathematical background. Plus that, if you want to implement the program yourself, you need to have solid programming background. For a beginner, starting to understand and implement his/her own version of face recognition, Hidden Markov Models and many more related topics by reading these papers can be a daunting task.

It is for this reason that this guide is written, to provide readers with a deep understanding of the rationale and intuition behind face recognition in step-by-step using very basic problems. And get you ready for the advanced methods found in journals and conferences. It is proven that the fastest way to understand any algorithm and method is through images, drawings and diagrams. This is the approach that I take here but from time to time, you may encounter a mathematical formula that is not a big deal since I will broke it down in a practical way.

One more thing that is lacking in the published papers, is details on implementation. This guide will fill this gap not only to tell you what to do, but also to show you line by line how I do it. This is similar as if you are watching over my shoulders as I write my program and talk with you about what is going on. Sample source code written for MATLAB is also provided for study and reference.

All right! What we want to do here is to recognize faces using singular value decomposition (SVD) features and hidden Markov models (HMM). But before I start, let me say a story for you. If you haven't read my story in my previous ebook on *Face Detection with Neural Networks and Gabor Feature Extraction* it is fine. Back in 2006, when I first stated to work on that program to detect faces, I had no experience with MATLAB. I was excellent at C and OK in C++ but no MATLAB

at all. That project was my first serious programming challenge to learn MATLAB. And by serious programming challenge I mean doing something more than just summing and averaging numbers. So after a while I became excellent at MATLAB programming. Not only I learned *Artificial Neural Networks* but I wrote a program that totally worked. You can still find it here. Now the point of the story is that I believe you can do the same too. I mean even if you are a beginner in MATLAB programming and face recognition, you can learn it fast but step-by-step. This is the best technique I have found to learn programming fast and it is to start from other programs and understand it by coping the program line by line until it becomes clear for you. If you follow this ebook, you can learn many details and become a powerful programmer.

This ebook helps you step by step to understand the basics of face recognition (or in general object recognition) using singular value decomposition (SVD) and hidden Markov models (HMM). At the end of this ebook you will be totally familiar with HMM and you will see the results from the code that you write.

In this ebook, I will not only work on the concept but I explain line by line how to write the program for recognition. Unlike other books that teach you every theoretical method for face recognition, this ebook teach you one face recognition method the way I do it; from concept to implementation. So be excited because nothing here is going to waste your time. Everything should be linked to a practical level which can be implemented right away.

The best way to learn the face recognition program in this ebook is to read this ebook step by step. You can change the parameters, play with them and check the results. But later you should continue your research on face recognition and replace different parts of the system until you reach to a better method and publish or compare your results.

First things first. If you haven't got the program

with this ebook. Please download it from here. The second code is more advanced and come with this ebook in a package. In the second code, you have the possibility to change each parameter separately.

Next, the best way to get in touch with me is though my facebook page. I am waiting for you there to share your experience and ask your questions. You can find it at My Facebook Page. Also if you have any question you can contact me on My Email. So let's get started.

# 2 Face Recognition

What exactly do we want to do in this ebook? To answer this question let me go right to the cover, in case you are also wondering why I chose a picture of a cute puppy for the cover of this ebook. Originally what I had in mind was a robotic puppy who can recognize people and probably say their name in dogish language. But I couldn't find a cute picture of a robotic puppy, so I said what the heck!! I am going with this one.

Now imagine that we have a cute robotic puppy who can actually see the environment. Means that it has a camera (or even two cameras) placed somewhere on her face. These cameras every several mili-seconds take a picture of the environment. Now here it comes! We want the puppy to recognize children by their faces and say their names.

The image that is captured using the camera may contain many objects, chairs, windows, etc. But occasionally someone may appear to play with the puppy. Each time that Jessie (Our puppy) is capturing an image, she looks for a face inside that image. This is called *Face Detection*. Face detection gives us the location of the face. For example, it says the face is centred at $(300, 400)$ with a height of 112 and a width of 92. It just says we have a face or several faces in the scene. However, it doesn't say who he/she is or who they are. *Face Recognition* actually recognizes the person and returns its name.

## 2.1 Components of The System

So what do we need? To make a face recognition system, we need three parts.

1. A dataset of faces for training
2. An unseen face to be recognized
3. A model to recognize the face

The first thing that should be done before writing a program is to gather some faces. As you can see in the folder of the program I have gathered many faces from the ORL face database. There are 40 people in this dataset and each person has 10 images. These images have a size of $112 \times 92$. 112 is the height of the image and 92 is the width of the image. Each image contains only one face.

To organize the faces you should put all the face images of one person in a separate folder. Because I didn't know the name of each person in the dataset, I named the folders like s1,s2,...,s40. I also added 10 images of my own face and added them in a folder called "omid". Figure 1 shows

Figure 1: Smaple of faces that we use for training and testing

examples of faces that you can find in these folders.

You can add your images too. Simply use a digital camera or a webcam to capture 10 images of yourself. Then, crop each image until you only see your face in it. You can see other images to find out how to correctly crop your images. But when you crop your images, you must make sure that the size is equal to $112 \times 92$. It takes some time because you should resize and crop and undo several times to get it correctly. The best software to do on Windows operating system is

Photoshop. If you have a lot of patient you can also use Microsoft Paint but I wish you good luck in that case. On Linux operating system you can use GIMP which is free. There is also a Windows installer for GIMP that is not officially supported, but works. As you know each image that you see on your computer is saved with a different format. You might be families with .JPG or .PNG formats but for some unknown reasons this ORL face database was saved in .PGM format! Probably to scare you off from using it! You can view these images using PhotoShop or IrfanView. If you don't have access to PhotoShop, IrfanView is nice, because you can also cut/crop files or convert them into .PGM format. Just be sure that the sizes of the final images are $112 \times 92$.

The other way to view images is to use MATLAB itself with the commands below. imread can read an image file and imshow can display it on the screen. Just remember that your working directory must be the same folder that contains "data" directory.

```
I = imread('data/s1/1.pmg');
imshow(I);
```

There are two ways that we can use this dataset. First we can divide it into equal parts and use some parts for training and some parts for testing. Then we name them "train dataset" and "test dataset". Or we can use the whole dataset for training and use other images (for example from webcam) to test the system.

Here I prefer to split the dataset into two datasets. I use 5 images from each person for training the system and the other 5 images for testing the system. Note that we cannot use the whole dataset for training and testing. That is called *Sanity Check* and if you report any results based on that it is called *Cheating*. The images that you use for testing the accuracy of the system must be different from those that you use for training the system.

So why? The reason is that sometimes models that should recognize a face overfit to the data. That means if the pixel values even change slightly, the

model is unable to give correct results. Models that overfit to the data give good results when they see the same exact face image again, however if the background or lighting condition is changed, the model gives wrong answers. This is also true for classification tasks. In case overfitting occurs in the training, if you use the same images for testing, you have no way to know that the model will actually performs well on unseen data. Only by improving the model or the extracted features we can reduce the overfitting that sometimes occurs. Then we can say that the model has generalization ability.

For testing the system as I mentioned before, I use part of the dataset. Also, if your computer has a webcam, it is pretty simple to capture a frame from the webcam device and use the frame to test the system.

The last part of the system is the model. The model that we use here is based on Hidden Markov Models and to extract features from images we use Singular Value Decomposition (SVD). You can think about the model as a black box that has two parts. The first part is for training. We give the model images of a number of people. Let's say we give it 5 images for each person. This part is called training and the model learns to recognize these people. Then, the second part of the model is for testing. In testing we give the model, one image that we don't know who he/she is. The model returns a probability for each person that it was trained on. For example if the model has learned 40 people then it gives us 40 probabilities. Each probability indicates how likely the input image could be that person. Then we simply can say that the person that returns the highest probability is in fact the person that is on the input image.

There have been several other methods for face recognition that you can read if you are interested.

- Picture processing by computer complex and recognition of human faces
- Eigenfaces for recognition

- Face recognition by elastic bunch graph matching
- Face recognition: A convolutional neural-network approach.
- Face recognition/detection by probabilistic decision-based neural network
- Face recognition by support vector machines
- Parametrization of a stochastic model for human face identification
- Face identification and feature extraction using hidden Markov models

Having said about everything that we need, we start by reading images and storing them in memory.

# 3 Loading Dataset

What we do here is part of **gendata.m** script. As I mentioned before, all the faces that we have gathered so far are organized in folders. I have called my folder "data". In the data folder, we have several other folders that each one belongs to one person. In each folder we have 10 PGM files that hold the pixel values of faces. Here for loading dataset, the goal is to read all the faces and store them in memory, so that we can later use them for training our model. The MATLAB command that we use here is `dir`. The `dir` command, retrieves the contents of a folder. We can simply write:

```
dataFolderContents = dir ('./data');
```

where *dataFolderContents* is a *cell matrix*. You probably know what a matrix is. A matrix in MATLAB is an array that holds one single type of data. For example an integer matrix can hold integer values. A cell matrix is a matrix that holds other matrices with different types and formats. For example cell $(1, 1)$ of a cell matrix can be an integer matrix and the cell $(1, 2)$ can be a string. This is a powerful data structure in MATLAB that can be effectively used to store and organize data for our program. So back to the command that we just wrote, it reads the hard drive and stores information about the contents

of the "data" folder in "dataFolderContents".

Now, the "dataFolderContents" is a cell matrix of size $N \times 1$. $N$ is equal to the number of folders and files plus 2. Each cell of this cell matrix is a *structure*. A structure is a data structure that has some attributes. In the structures that are returned from `dir`, these attributes are the name of the file or folder, data and time it was created and many more. However, we only need to know about two of these attributes. One is the *name* attribute and the the other is *isdir* attribute. Of course to get the number of files and folders in "dataFolderContents" we can use the `size` command as follows:

```
nRecords = size(dataFolderContents,1);
```

which only returns the first dimension (number of rows) and we already know that the second dimension (number of columns) is one. Because that is what `dir` always returns.

Note that the first two rows of the `dir` answer is always "." and "..". "." refers to the same directory and ".." refers to the parent directory. And it has been like this for god knows, from 20 years ago. So we also want to skip them while reading the folder names. Here is the final code for accessing the folder of each person in "data" folder individually:

```
for p = 1: nRecords
    if (strcmp(dataFolderContents(p,1).name,'.')
        % we should skip this
        continue;
    end
    if (strcmp(dataFolderContents(p,1).name,'..')
        % we should also skip this one
        continue;
    end
    if (dataFolderContents(p,1).isdir==0)
        % the record is not a folder
        % and we should skip this
        continue;
    end
    % we have the folder name in
    % dataFolderContents(p,1).name
end
```

Inside the code you see somewhere a comment that says "we have the folder name" and you can write this piece of code and print the folder name. Sim-

ply by `fprintf(dataFolderContents(p,1).name)` command. What happens is that the $p$ variable is changing inside the for-loop. Each time it increments starting from 1. Then we access each row to check if that row points to a folder name or not. If that row does not point to a valid folder name then `continue;` is executed that simply goes back to the start of the loop with a new $p$.

Next, inside the loop we again need to execute another `dir` command to get the names of PGM files. Typically we expect 10 files for each person and they are numbered. To get the person name we can write `name = dataFolderContent(p,1).name` and to get the list of PGM files for each person we write `dir(['./data/',name,'/*.pgm'])` and store the answer in `personFolderContents`. Note that because we explicitly asked for *.pgm, the answer does not contain any folder name or non sense stuff such as "." and "..". The code becomes:

```
fprintf('Loading faces...\n');
ufft = [1 5 6 8 10];
for p = 1: nRecords
    if (strcmp(dataFolderContents(p,1).name,'.')
        continue;
    end
    if (strcmp(dataFolderContents(p,1).name,'..')
        continue;
    end
    if (dataFolderContents(p,1).isdir==0)
        continue;
    end
    personName = dataFolderContents(p,1).name;
    fprintf([personName,' ']);
    personFolderContents = ...
       dir(['./data/',personName,'/*.pgm']);
    for face=1:5
        folderName = ['./data',personName];
        fileName = personFolderContents(...
        ufft(face),1).name,1).name;
        I = imread([folderName,'/',fileName]);
        % do someting with this face
        % for example extract features
    end
end
```

In the above code we mentioned `fprintf` two times. The first time it just prints "Loading faces..." in the command windows and `\n` ensures that the cursor goes to the next line. It is a good practice to print something on the screen, so that when you are executing the code, you don't wonder which part of the code is in execution.

`ufft` is a vector that contains 5 integer values between 1 and 10. These 5 values are the index of faces that we use for training. In this case for each

person we use only the faces number 1,5,6,8 and 10.The index addressing is carried out

When we read the image and load it into $I$ matrix, if we don't do any thing and close the loop, it will not be saved. Usually we perform some pre-processing and operations on the image and extract features. Then we are able to save those features in a form that is accessible for training. Before we extract any feature, I would like to describe Hidden Markov Models and give you the overview of how it fits into face recognition.

# 4 Hidden Markov Models

Hidden Markov model is a simple tool for determining the past and future of an event in a sequence of events. More than that it can answer what is the occurrence probability of a sequence of events, given all the correlated observations. I know that description don't help at all to a beginner, but maybe an example than shed a light on the situation.

Imagine we live in an era where no one could register the temperature of the weather, however people have written their observation about the sky. Suppose we want to determine the temperature for a day based on the observation of rain, snow and sun. Let's say evidence indicates that the probability of a hot day (H) followed by another hot day is 0.7 and the probability that a cold day (C) is followed by another cold day is 0.6. The information can be summarized as:

$$B = \begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

This also implies that the probability of a cold day (C) followed by a hot day (H) is 0.4 or with 40% chance. We call this matrix a *state transition matrix* and each element of the matrix is a transition probability. Let's suppose that another evidence indicates a correlation between the observation of the weather (e.g being sunny, rainy or snowy) and the temperature. This evidence is given by:

$$A = \begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Sunny} & \textit{Rainy} & \textit{Snowy} \end{array} \\ \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.4 & 0.5 \end{bmatrix} \end{array}$$

We also call this *emission probabilities*. For this system, the *state* is the average daily temperature– either $H$ or $C$. The transition from one state to the other is a *Markov process*, since the next state depends on the current observations and the previous state of the model. However, the actual states are "hidden" since we can't observe the temperature. We call this system a *Hidden Markov Model*(HMM). There is also a way to show this model, displayed in figure 2.

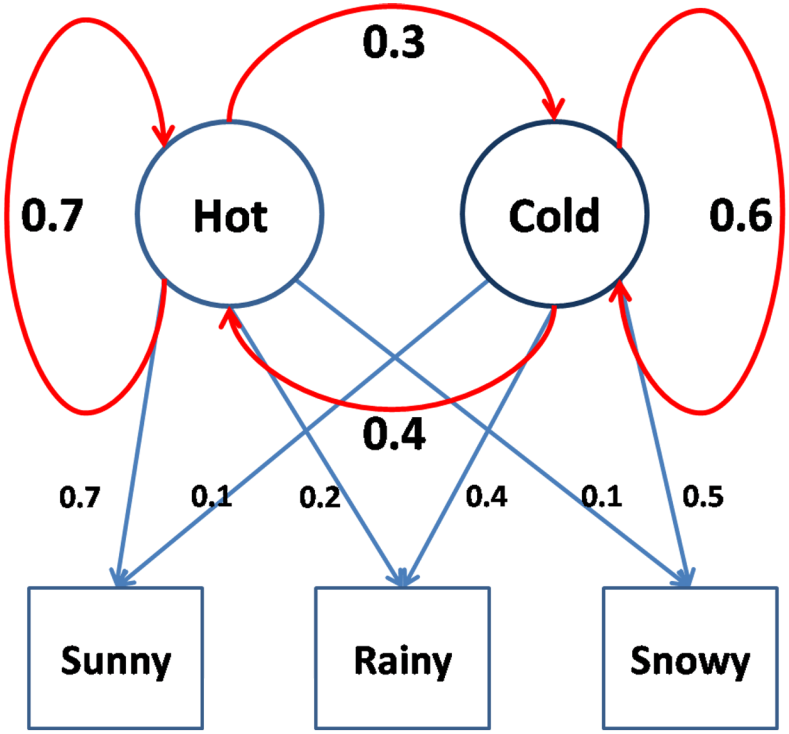



Figure 2: Our hidden Markov model

To complete our example, suppose that the initial states of the system is $\pi = \{0.5, 0.5\}$. It assumes that on day 0, there has been a 50% chance of having a hot or cold day.

The matrices $\pi$, $A$ and $B$ are probability matrices, meaning that the elements of each row sum to 1 and each row is a probability distribution. An HMM model $O = (\pi, A, B)$ can

be defined only by having these three matrices. Given this model, we can answer many questions. For example, let's consider a particular four-day period for which we had this observation of $\{Sunny, Sunny, Rainy, Rainy\}$.

- What is the probability of having a hot day in day 0 ($P(H)$)? The answer is that according to $\pi$ the probability of having a hot day in day 0 is 0.5.
- What is the probability of having two hot days in a row on day 0 and 1 ($P(HH)$)? The answer is $(\pi(Hot) = 0.5) \times (P(Hot|Sunny) = 0.7) \times (P(Hot \rightarrow Hot) = 0.7) \times (P(Hot|Sunny) = 0.7)$.
- What is the probability of having a sequence $\{C, H, C\}$? The answer is $(\pi(C) = 0.5) \times (PC|Sunny = 0.1) \times (P(C \rightarrow H) = 0.4) \times (P(H|Sunny) = 0.7) \times (P(H \rightarrow C) = 0.3) \times (P(C|Rainy) = 0.5)$
- What is the most likely sequence of hot and cold days that best describes our observations? The answer to this question can be determined using Viterbi algorithm. We don't want to get into details here, since it can be done easily with only one line of code in MATLAB.

To get more insight into Hidden Markov Models, I recommend two sources.

1. A Revealing Introduction to Hidden Markov Models.
2. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.

Just in case you might be interested in using HMM in C++, I have an implementation here that works more or less like its MATLAB version.

Now let's formulate a HMM model for face recognition. Suppose when we look at a face image from top to bottom, we see seven distinct regions; hair, forehead, eyebrows, eyes, nose, mouth and chin. Remember that we had a face image of size $112 \times 96$. If we assume that we have a fixed block of size $7 \times 96$ which covers the entire width of the image. When we put this block on top of the face image and we move this block 1 pixel at a time towards the bottom of the image, then at any time,

the block might show one of the seven regions that I just mentioned. So we can easily model this process by a 7-state hidden Markov model. But we also know four more information.

- If we happen to have a block with state "eyes", the next block can never be a "head" state because we are moving the block toward the bottom of the image. We roughly assume that the probability of going from one state to itself is 50% and the probability of going from one state to the next state is also 50%.
- Unless the person that we are recognizing has serious defects in his/her face, it is not possible to jump from a "head" state to "Nose" or any other states except "Forehead".
- The initial state of the system is always "head" with a probability of 1.
- The final state of the system is always "Chin". In other words, if a block has a "Chin" state, the next block and the states of the rest of the blocks are always "Chin". We call this an *attractor* state.

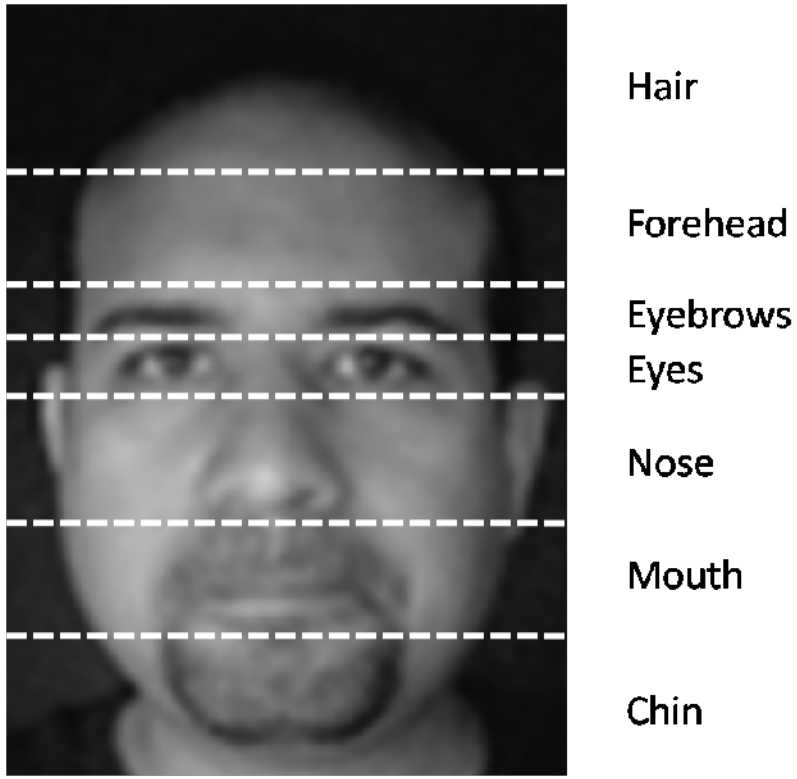



Figure 3: Severn regions of face

Figure 3 shows a view of these regions.

We can summarize the information in two matrices.

$$B = \begin{array}{c} \\ Head \\ Forehead \\ Eyebrows \\ Eyes \\ Nose \\ Mouth \\ Chin \end{array} \begin{array}{c} \begin{array}{ccccccc} Head & Forehead & Eyebrows & Eyes & Nose & Mouth & Chin \end{array} \\ \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \end{array}$$

$$\pi = \begin{array}{c} \begin{array}{ccccccc} Head & Forehead & Eyebrows & Eyes & Nose & Mouth & Chin \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The last piece of the puzzle is the observations. Imagine that for each block of the image we may observe one discrete number between 1 and 1260. I will certainly describe later how we come about 1260 discrete values but for know just accept it like this. These number are the equivalent of the observations of "sunny", "rainy" and "snowy" that we had on the example before. In the most generic form we can create the $A$ matrix like:

$$A = \begin{array}{c} \\ Head \\ Forehead \\ \vdots \\ Chin \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & 1260 \end{array} \\ \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix} \end{array}$$

Having $\pi$,$A$ and $B$ matrices, we have a HMM model for a generic face. The fact is that if all the faces in the would had this model, then we couldn't recognize anyone since everybody had the same face. What we should do is to make an HMM model specifically for each person in our dataset. For example if we want to recognize 40 people, we need 40 HMM models. But how we are going to do that? With only one line of code in MATLAB. There is a function that can train an HMM model. We give this function the generic face model that I just mentioned. What it does, is that it tweaks the probabilities until the joint probability of the whole observations and states sequence is maximized for that person. Meaning that the model has learned the sequence of observations that are extracted from the face of that person and the probability of the whole sequence is maximized for that.

Before I get into the codes for training the HMM model, I would like to talk about how we generate the observations. This leads to extracting features from the blocks of the face image and converting each block into one single discrete number between 1 and 1260.

# 5 Feature Extraction

## 5.1 Block Extraction

First of all, we convert each face image into gray-level. This can be done using:

```
try
    I = rgb2gray(I);
end
```

where everything between `try` and `end` is executed only if it was successful. This ensures that if the image in I is already a gray-level image and not a color one, it just ignore the `I=rgb2gray(I);` and doesn't let the script to crash.

Then we resize each face image to 50% of its size. The command for resize is:

```
I = imresize(I,[56 46]);
```

The reason for resizing the image is to gain more speed in training and testing. But it actually improves the accuracy of the recognition but preventing the model to overfit the data and the problems with curse of dimensionality.

The observation sequence is generated by dividing each face image of width $W = 46$ and height $H = 56$ into overlapping blocks of height $L = 5$ and width $W = 46$. A patch with size $L \times W$ slides from the top to bottom of the image and generates a sequence of overlapping blocks. If we get the overlapping size to be $P = L - 1$ the we ensure that each time we move the patch only one pixel. The number of blocks ($T$), extracted from each face image is computed as follows:

$$T = \frac{H - L}{L - P} + 1 = \frac{56 - 5}{5 - 4} + 1 = 52$$

The program to extract the blocks looks like this:

```
blk_height = 5;
blk_overlap = 4;
blk_begin = 1;
blk_index = 0;
for ...
   blk_end=blk_height:blk_height-blk_overlap:56
```

```
        blk = I(blk_begin,blk_end);
        %do something with the blk
        %for example extract features
        blk_index=blk_index+1;
        blk_begin=blk_begin+blk_height-blk_overlap;
    end
```

This means that each face image that we have can be converted into a sequence with 52 elements. But we have a problem. We just extracted blocks in a loop. Inside the loop we can do anything with the block. However, as I mentioned in our HMM model we need single discrete values. We have to find a way to convert each block into a single values in a way that it represents all the gray-level values inside a block. Each block has $5 \times 46 = 230$ gray-level values but we need just one value. This problem is inherently a problem of dimension reduction. To solve this problem we turn into Singular Value Decomposition (SVD). I don't want to justify here why SVD, but if you are interested, you can read the original work that includes various experiments to prove the superiority of this approach. You can search for it in Google by"A New Fast and Efficient HMM-Based Face Recognition System Using a 7-State HMM Along With SVD Coefficients".

## 5.2 Singular Value Decomposition (SVD)

SVD has been an effective tool in statistical data analysis and signal processing. Singular values of given data matrix contain information about the energy and level of noise of the matrix. Singular vectors of a matrix are orthonormal and exhibit some characteristics of the pattern, embedded in the signal. That makes it a good candidate for feature extraction in pattern recognition. The decomposition of a $m \times n$ matrix $X$ is a function of the form:

$$X = U \sum V^*$$

where $U_{m \times m}$ and $V^*_{n \times n}$ are orthogonal matrices and $\sum_{m \times n}$ is a diagonal matrix of singular values with components $\sigma_{ij} = 0, i \neq j$ and $\sigma_{ii} > 0$. Furthermore, it can be shown that there exists non-unique matrices $U$ and $V$ such that $\sigma_1 \geq \sigma_2 \geq ... \geq 0$. The columns of the orthogonal matrices are called the left and right singular vectors, respectively.

The main property of SVD that is relevant to face recognition is its stability on face images. So it

can be considered a robust extraction technique for face images. To get more insight into geometric interpretation of SVD you can watch this video on YouTube.

So, back to our problem, after extracting each block we have a matrix of size $5 \times 46$. SVD is applied on this block to give us three matrices ($U$,$\Sigma$,$V$). We only keep three single values from these matrices and skip the rest. The three values are $U(1,1)$,$\Sigma(1,1)$ and $\Sigma(2,2)$. These values have shown the best classification rate, confirmed by experiments in the original paper. The code below is used to compute SVD from the block matrix. This code goes write into the loop that calculates blocks.

```
%convert block matrix into double
blk_double = double(blk);
[U,S,V] = svd(blk_double);
coeff1 = U(1,1);
coeff2 = S(1,1);
coeff3 = S(2,2);
```

Now for each block, we have got three real values and we are still far from one single discrete value for each block to use as part of the observation sequence for HMM model. Here comes the quantization step.

## 5.3 Quantization

Since SVD coefficients have continues values, they can't be modelled by discrete HMM. So we should find a way to quantize these values.

The quantization is carried out by a process of rounding or truncation of some sort. This step is irreversible and we lose part of the information in the process. However, it is necessary for our application. Suppose we have vector of $X = (x_1, x_2, ..., x_n)$ with continues elements. $X_i$ should be quantized into $D_i$ distinct levels. The difference between two successive quantized values is:

$$\Delta_i = \frac{x_i^{max} - x_i^{min}}{D_i}$$

where $x_i^{max}$ and $x_i^{min}$ are the maximum and minimum values of $x_i$ in all possible observation vectors. By knowing $\Delta_i$, every value $x_i$ can be re-

placed with its quantized value computed as:

$$x_i^{\text{quantized}} = [\frac{x_i - x_i^{min}}{\Delta_i}]$$

The values of $U(1,1)$,$\Sigma(1,1)$ and $\Sigma(2,2)$ are quantized into 18,10 and 7 levels, respectively. However, this doesn't solve one problem. We have extracted each block and we have found the three coefficients that should be quantized, but we don't know the minimum and maximum value of coefficients in advance. Not knowing the minimum and maximum values which are necessary for quantization, we can not do the computation on-the-fly any more. What we do, is that we store all the coefficients in a matrix, so that in a second pass we can find the maximum and minimum values from all possible observed coefficients. So to recap the program, here I give the full code until the values are stored in a matrix.

```
dataFolderContents = dir ('./data');
nRecords = size(dataFolderContents,1);
fprintf('Loading faces...\n');
%this line is new
myDatabase = cell(0,0);
%this line is new
personIndex = 0;
for p = 1: nRecords
if (strcmp(dataFolderContents(p,1).name,'.')
    continue;
end
if (strcmp(dataFolderContents(p,1).name,'..')
    continue;
end
if (dataFolderContents(p,1).isdir==0)
    continue;
end
%this line is new
personIndex = person_index+1;
personName = dataFolderContents(p,1).name;
%this line is new
myDatabase{1,personIndex} = personName;
%this line is new
blkCell = cell(0,0);
fprintf([personName,' ']);
personFolderContents = ...
   dir(['./data/',personName,'/*.pgm']);
for face=1:size(personFolderContents,1)
    folderName = ['./data',personName];
    fileName = ...
       personFolderContents(face,1).name;
    I = imread([folderName,'/',fileName]);
    try
        I = rgb2gray(I);
    end
    I = imresize(I,[56 46]);
    %this line is new
    I = ordfilt2(I,1,true(3));
    blk_height = 5;
    blk_overlap = 4;
    blk_begin = 1;
    blk_index = 0;
    for blk_end=...
        blk_height:...
        blk_height-blk_overlap:...
        56
        blk = I(blk_begin,blk_end);
```

```
            blk_double = double(blk);
            [U,S,V] = svd(blk_double);
            coeff1 = U(1,1);
            coeff2 = S(1,1);
            coeff3 = S(2,2);
            %this line is new
            blkCell{blk_index,face} = [coeff1 ...
               coeff2 coeff3];
            blk_index=blk_index+1;
            blk_begin=...
            blk_begin+blk_height-blk_overlap;
        end
    end
    %this line is new
    myDatabase{2,personIndex} = blkCell;
end
```

In the above code, several new lines are added and I have marked them with "this line is new". Here I go over them line by line. The first line that is added is `myDatabase = cell(0,0);`. This line defined a cell matrix called "myDatabase". It will store all the information that we have talked about so far, as well as anything that we will compute in the future. Each column of this cell matrix belongs to only one person. The first row of `myDatabase` contains the name of the person. The second row is another cell matrix itself called "blkCell". blkCell is a cell matrix that has 52 rows and 5 columns. Each person has exactly one "blkCell". The 5 columns correspond to the number of images that we have for each person. But the 52 rows, as you guessed, corresponds to the number of blocks that we extract for each image. So what goes into the cells of "blkCell"? Only one matrix of size $1 \times 3$ which stores the three coefficient values, extracted from each block.

The next line that is added to the code is `personIndex=0`. Remember that we ignored some folder names like "." and ".."? Well! because we ignored them, $p$ is no longer a valid index for each person. To have a valid index for each person, we define `personIndex=0` and we have another new line `personIndex = person_index+1;` which increments personIndex after we have found a new valid person in the for-loop. It is not a big deal. It's just an index.

The next new line is `myDatabase{1,personIndex} = personName;`. It is clear that the name of the person goes to the first row of myDatabase as I mentioned. `I = ordfilt2(I,1,true(3));` is a new added line and applies a minimum order-static filter to the gray-level image. A minimum order-

static filter is a non-linear spatial filter. A sliding windows moves from left to right and then from top to bottom with a step size of one pixel. At each step, a patch of size $3 \times 3$ is replaced by one the minimum of the pixels. It is clear that this filter has a smoothing role and reduces the information of the image. You can ignore this line, but experimental results in the original paper of this work shows that this filter improves the recognition accuracy.

The other new line that I want to talk about is `blkCell = cell(0,0);`, which creates an empty cell matrix. This line is located inside the for-loop for each person. As a result, before reading the faces fro each person, this cell matrix is empty. Then we fill this with information with line `blkCell{blk_index,face} = [coeff1 coeff2 ... coeff3];`. Finally when all the three coefficients for all blocks of the 5 images are written into `blkCell`, we set the second row of `myDatabase` with it in `myDatabase{2,personIndex}=blkCell;`.

Back to the quantization, now we have all the data that we need to find the maximum and minimum of all the three coefficients. We know that the minimum of $\sum(1,1)$ and $\sum(2,2)$ is zero. But we still need to find the minimum of $U(1,1)$ and the maximum of $\sum(1,1)$,$\sum(2,2)$ and $U(1,1)$ for all coefficients in the entire observation vectors. This can be done by going one more pass through all the data and gathering them in separate matrices. Here is the code for that:

```
coeff1 = [];
coeff2 = [];
coeff3 = [];
n_persons = size(myDatabase,2);
for person_index=1:n_persons
    [n_blocks,n_images] = ...
       size(myDatabase{2,person_index});
    for image_index=1:n_images
        for block_index=1:n_blocks
            coeff1(:,end+1) = ...
               myDatabase{2,person_index}...
            {block_index,image_index}(1,1);
            coeff2(:,end+1) = ...
               myDatabase{2,person_index}...
            {block_index,image_index}(1,2);
            coeff3(:,end+1) = ...
               myDatabase{2,person_index}...
            {block_index,image_index}(1,3);
        end
    end
end
max_coeff1 = max(coeff1(:));
max_coeff2 = max(coeff2(:));
```

```
max_coeff3 = max(coeff3(:));
min_coeff1 = min(coeff1(:));
min_coeff2 = 0;
min_coeff3 = 0;
```

Then finding the $\Delta$ is easy.

```
eps = .000001;
Δ_coeff1 = (max_coeff1-min_coeff1)/(18-eps);
Δ_coeff2 = (max_coeff2-min_coeff2)/(10-eps);
Δ_coeff3 = (max_coeff3-min_coeff3)/(7-eps);
```

where *eps* is a very small value. In the third pass through all the data, the quantized values can be calculated and stored in the third row of `myDatabase`. The code is easy as follows:

```
min_coeffs = [min_coeff1 min_coeff2 min_coeff3];
Δ_coeffs = [Δ_coeff1 Δ_coeff2 Δ_coeff3];
for person_index=1:n_persons
    for image_index=1:n_images
        for block_index=1:n_blocks
            blk_coeffs = ...
               myDatabase{2,person_index}...
            {block_index,image_index};
            qt = ...
            floor((blk_coeffs-min_coeffs)./Δ_coeffs);
            myDatabase{3,person_index}...
            {block_index,image_index} = qt;
        end
    end
end
```

The third row of `myDatabase` contains 3 discrete values. The first value is between 0 and 17. The second value is between 0 and 9. The third value is between 0 and 6. But there is one piece of puzzle still missing. For each block of each face image of each person we have 3 discrete values. However, we need only one discrete value for each block. We call the value a "label". Now this is easy. Since there are three discrete values, we consider every possible combination that may happen with these values. So the total possible combination is $18 \times 10 \times 7 = 1260$. So now the magical 1260 number appears for the first time. If all the values are zero then the label is 1. If the discrete coefficient values are $(17, 9, 6)$ then we have the maximum value for the label which is 1260. Looks simple! right? Here is the single line code to generate the label:

```
label = qt(1)*10*7+qt(2)*7+ qt(3)+1;
```

Let's put it to the test. For coefficients of $(17, 9, 6)$ we have $17 * 10 * 7 + 9 * 7 + 6 + 1 = 1260$. Let's rewrite the code above and add this line of code after we computed the quantized values that are stored in $qt$.

```
min_coeffs = [min_coeff1 min_coeff2 min_coeff3];
Δ_coeffs = [Δ_coeff1 Δ_coeff2 Δ_coeff3];
for person_index=1:n_persons
    for image_index=1:n_images
        for block_index=1:n_blocks
            blk_coeffs = ...
               myDatabase{2,person_index}...
            {block_index,image_index};
            qt = ...
               floor((blk_coeffs-min_coeffs)./Δ_coeffs);
            myDatabase{3,person_index}...
            {block_index,image_index} = qt;
            label = qt(1)*10*7+qt(2)*7+ qt(3)+1;
            myDatabase{4,person_index}...
            {block_index,image_index} = label;
        end
        myDatabase{5,person_index}{1,image_index}=...
        cell2mat(myDatabase{4,person_index}...
        (:,image_index));
    end
    myDatabase{5,person_index} = ...
       cell2mat(myDatabase{5,person_index})';
end
```

As you can see, I have also added two more information for each person. In the forth row of `myDatabase` I have gathered the labels in a cell matrix. In the fifth row, after all the labels are computed for each image, I have converted the cell matrix into a regular matrix with integer values that holds the observation sequence for each face image. The command that converts cell matrices into regular matrices is cell2mat. Finally, with a matrix transpose in the fifth row for each person we have a matrix of size $5x52$. 52 is the number of observations per sequence which also is the total number of blocks for each face image. 5 is the number of faces that we use for training a model for each person. Figure 4 shows all the steps that I mentioned so far for converting a single face image to its observation sequence.

## 5.4 About Implementation

So far I tried to simplify every part of the program for you. This way you could understand it. What I mean by "simplify" is that I deliberately used numeric values for everything. However, if you look the code, you might find that instead of using values size as 56 as the height of the image and for example 5 as the height of the block, I have gathered all of them in a structure format, defined at the top of the **gendata.m**. This can make the program changeable. Here is the parameters that I defined at the beginning of the code:

```
params.blk_height = 5;
params.blk_overlap = 4;
```

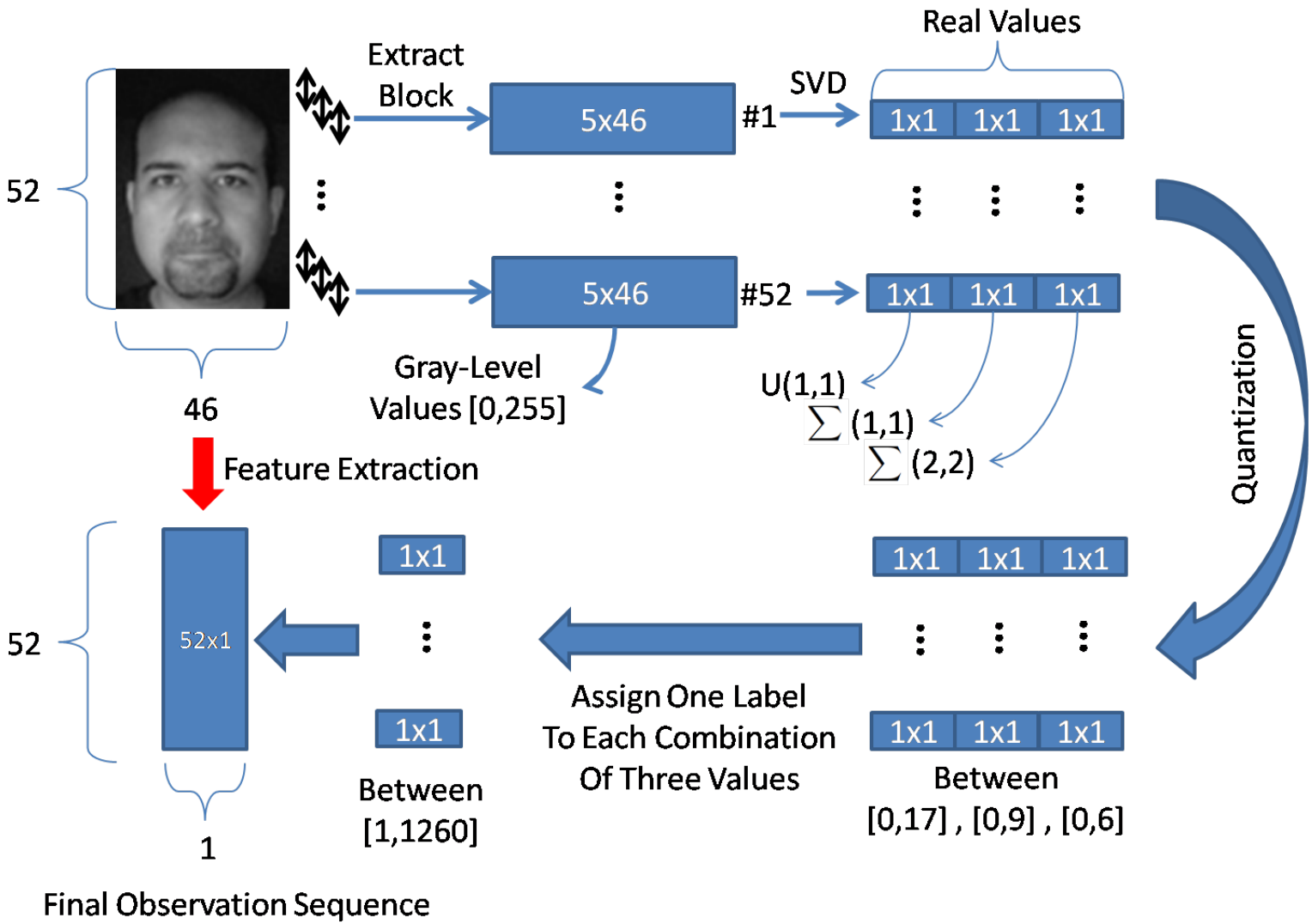



Figure 4: An overview of the steps for converting a single face image to its observation sequence

```
params.coeff1_quant = 18;
params.coeff2_quant = 10;
params.coeff3_quant = 7;
params.number_of_states = 7;
params.face_height = 56;
params.face_width = 46;
params.used_faces_for_training = [1 5 6 8 10];
params.used_faces_for_testing = [2 3 4 7 9];
```

Since every parameter of the program is here, if you change any parameter, the program still works :-).

# 6 Training

Well done! The hard part is over. Actually the training stage of a HMM is the hardest part if you want to implement it yourself. But thanks to MATLAB, it is only one line of code. Here it is:

```
[ESTTR,ESTEMIT] = hmmtrain(seq,TRGUESS,EMITGUESS)
```

where *TRGUESS* should be a good guess for the transition probabilities (matrix $B$) and *EMITGUESS* should be a good guess for emission probabilities (matrix $A$). *seq* is a matrix that has all the observation sequences that you want to use for training. For example in our code, we have a matrix of $5 \times 52$. Each row is one observation sequence and in each observation sequence we have 52 observations that corresponds to the number of blocks that we have for each face image. What we do is that for each person, we give all the observations that we would like to use for training as an input in *seq*. We also give it the generic face model that we worked out in the previous sections.

`hmmtrain` estimates the transition and emission probabilities for a hidden Markov model using the Baum-Welch algorithm. TRGUESS and EMITGUESS are initial probabilities and both *ESTTR* and *ESTEMIT* are final estimated probabilities. done! Let's get into the code.

```
TRGUESS = ones(7,7) * eps;
TRGUESS(7,7) = 1;
for r=1:6
        TRGUESS(r,r) = 0.6;
        TRGUESS(r,r+1) = 0.4;
end
EMITGUESS = (1/1260)*ones(1260,1260);
```

After we created the initial probabilities, we use those to train one HMM model for each person in our dataset and we store the final estimated probabilities in the sixth row of `myDatabse`. The only consideration is that it is of our interest to not have a probability of zero in the probability matrices, so we use a `max` operator to replace every zeros in the probability matrices with *epsilon*, a very small value.

```
fprintf('\nTraining ...\n');
for person_index=1:40
    fprintf([myDatabase{1,person_index},' ']);
    seqmat = cell2mat(myDatabase{5,person_index})';
    [ESTTR,ESTEMIT]=...
        hmmtrain(seqmat,...
        TRGUESS,...
        EMITGUESS,...
        'Tolerance',.01,...
        'Maxiterations',10,...
        'Algorithm', 'BaumWelch');
    ESTTR = max(ESTTR,eps);
    ESTEMIT = max(ESTEMIT,eps);
    myDatabase{6,person_index}{1,1} = ESTTR;
    myDatabase{6,person_index}{1,2} = ESTEMIT;
end
```

# 7 Recognition Process

The goal here is to examine the recognition accuracy of the system. After the training process, each class (face) is associated to a hidden Markov model. So, for 40-class classification problem, we have 40 distinct models. In testing, each test image (a face of size $112 \times 92$ goes through the same block extraction, feature extraction and quantization process. Then, each test image like the training images is represented by its own observation sequence (A vector of 52 integers which each has a value between 1 and 1260). For an incoming test image we simply calculate the probability of the obtained observation sequence given each HMM face model. Suppose $O_n$ is the HMM associated with person $n$ in our database and $X$ is the observation sequence for the test image. Then, a face image is recognized as person $d$ if:

$$P(X|O_d) = arg \max_n P(X|O_n)$$

The code can be found in **facerec.m**. Here is the function that gets `myDatabse` and the minimum and maximum of the coefficients as the input and returns the index of the person in our database.

```
function [person_index,maxlogpseq] = ...
    facerec(filename,myDatabase,minmax)
I = imread(filename);
try
    I = rgb2gray(I);
end
I = imresize(I,[56 46]);
I = ordfilt2(I,1,true(3));
min_coeffs = minmax(1,:);
max_coeffs = minmax(2,:);
Δ_coeffs = minmax(3,:);
seq = zeros(1,52);
for blk_begin=1:52
    blk = I(blk_begin:blk_begin+4,:);
    [U,S,V] = svd(double(blk));
    blk_coeffs = [U(1,1) S(1,1) S(2,2)];
    blk_coeffs = max([blk_coeffs;min_coeffs]);
    blk_coeffs = min([blk_coeffs;max_coeffs]);
    qt = ...
    floor((blk_coeffs-min_coeffs)./Δ_coeffs);
    label = qt(1)*7*10+qt(2)*7+qt(3)+1;
    seq(1,blk_begin) = label;
end
number_of_persons_in_database = size(myDatabase,2);
results = zeros(1,number_of_persons_in_database);
for i=1:number_of_persons_in_database
    TRANS = myDatabase{6,i}{1,1};
    EMIS = myDatabase{6,i}{1,2};
    [ignore,logpseq] = hmmdecode(seq,TRANS,EMIS);
    P=exp(logpseq);
    results(1,i) = P;
end
[maxlogpseq,person_index] = max(results);
fprintf(['This person is ...
    ',myDatabase{1,person_index},'.\n']);
```

If you have followed the evolution of the program in **gendata.m**, you are already familiar with everything that is going on inside the function above. There are only two parts that demand more explanation.

When we extract $U(1,1)$, $\Sigma(1,1)$ and $\Sigma(2,2)$, we must ensure that the extracted SVD coefficients are not smaller or larger than the coefficients that we have already seen in the training process. To ensure this, we enforce these values to be within the previous seen minimum and maximum values by:

```
blk_coeffs = max([blk_coeffs;min_coeffs]);
blk_coeffs = min([blk_coeffs;max_coeffs]);
```

The final mystery is to calculate the probability of a given observed sequence for a specific HMM model. This can be done by:

```
[ignore,logpseq] = hmmdecode(seq,TRANS,EMIS);
```

For each person in our database, we set TRANS and EMIS to the probability matrices that are obtained for that person during training. *seq* is the observation sequence of size $1 \times 52$ for the test image. Then `hmmdecode` function returns a probability and we store all the values in `results`

vector. The returned probability is actually in logarithm of probability for the sequence and that is why it is named as `logpseq`.

Finally, the index of the person can be found by finding the index of the maximum probability in `results`. Hence, knowing the index of the person, we can return the name of the person, stored in the first row of `myDatabase` as the recognized person.

# 8 Testing

The final chapter in our face recognition, is to evaluate the system. We saw that by using `uttf` we chose 5 images for the training phase. Since we have 10 images for each person, again we use `ufft` vector, but this time the vector contains integer values that point to the indices of faces for each person that are not used for training. Since we already have a good function `facerec.m` that can return the index of the recognized face, it is simple to evaluate the system. The code is as follows:

```
function rr = testsys(myDatabase,minmax)
ufft = [2 3 4 7 9];
total = 0;%total number of faces
rr = 0; %recognition rate
fprintf('Please Wait...\n');
dfc = dir ('./data'); %data folder contents
%number of folders in data folder
nf = size(dfc,1);
pi = 0; %index of person
for p=1:nf
    if (strcmp(dfc(p,1).name,'.') || ...
        strcmp(dfc(p,1).name,'..') || ...
        (dfc(person,1).isdir == 0))
        continue;
    end
    pi = pi+1;
    pname = dfc(p,1).name; %person's name
    fprintf([pname,'\n']);
    %person folder contents
    pfc = dir(['./data/',pname,'/*.pgm']);
    for f=1:size(ufft,2) %face index
        total = total + 1;
        %file name
        fname = ...
            ['./data/',pname,'/',ppfc(ufft(f),1).name];
        %recognized person
        rp = facerec(fname,myDatabase,minmax);
        if (rp == pi)
            rr = rr + 1;
        end
    end
end
rr = rr/total*100;
fprintf(['\nRecognition Rate is ',num2str(rr),']);
```

The above function takes `myDatabase` and the minimum and maximum of coefficients as the input.

Then it doesn't do anything with them except to pass them to the function `facerec`. The rest of the function is about reading the contents of the "data" folder and recognizing each person. `rr` holes the number of correctly recognized faces until we divide it by the total number of faces in the end. Then it will be returned as the recognition accuracy of the system. Note that in the first line of the function we are indexing faces number 2,3,4,7 and 9 for each person as the faces to be recognized. They are different from the faces that are used in training.

# 9 Webcam Access

The function `getcam` is a very simple function that allow live view of the Webcam. Then a menu appears on the screen that allow the user to capture a frame from the video when he/she is ready. The function process the frame to turn it into gray-level and resize the middle part of the frame to a standard size face image with size $112 \times 92$. Here is the code:

```
function I = getcam()
vid = videoinput('winvideo', 1, 'RGB24_320x240');
preview(vid);
choice=menu('Capture Frame',...
            '   Capture   ',...
            '     Exit    ');
I = [];
if (choice == 1)
    I = getsnapshot(vid);
    try
        I = rgb2gray(I);
    end
    I = I(8:231,68:251);
    I = imresize(I,[112 92]);
end
closepreview(vid);
```

The two important functions here are `videoinput` and `getsnapshot`. However, you need the Image Acquisition Toolbox installed for these functions to be available on your system.

# 10 Main Menu

Finally, we bound everything together to get a nice menu for the user to use the program. You can find the code for this section in `mainmenu.m` which is also the entry point to the program. A simple way to do this, is by using `menu` command. Before that we check to see if `DATABASE.mat` exists. If it does exist,

we load it into the many. This file contains our myDatabase database and the minmax matrix which contains the minimum and maximum of SVD coefficients.

```
if (exist('DATABASE.mat','file'))
    load DATABASE.mat;
end
```

We would like to create a menu. But we don't want to exist the menu, each time that the user clicks on one item. The only way to exit the program should be when the user clicks on the "Exit" item of the menu. To do this we put the menu and everything associated with the menu inside an infinite loop.

```
while (1==1)
end
```

First we define the menu and it waits for the user to select one item from the menu.

```
choice=menu('Face Recognition',...
    'Generate Database',...
    'Calculate Recognition Rate',...
    'Recognize from Image',...
    'Recognize from Webcam',...
    'Exit');
```

Then we should check the selected item which is stored in choice. If the choice is equal to 1, means that the user wants to create the database. We can simply create the database by:

```
[myDatabase minmax] = gendata();
```

We must also make sure than gendata() saves all the nessesary variables in DATABASE.mat and define them in the memory. Then either by calling gendata() or loading the database from file, choice 2 is ready.

```
if (choice == 2)
    if (¬exist('myDatabase','var'))
        fprintf('Please generate database ...
            first!\n');
    else
        recognition_rate = ...
            testsys(myDatabase,minmax);
    end
end
```

Buy exist('myDatabase','var') we can check to see if the variable myDatabase exists in the memory

or not. If the variable does not exist in the memory, the program shows an appropriate message on the screen to inform the user that he/she should generate the database before testing the system. The third choice shows a file dialogue box on the screen for the user to choose one image file. It then reads the file and try to recognize the face of the person. The face inside the image should comply with the rules that we had for all the face images used for training.

```
if (choice == 3)
    if (¬exist('myDatabase','var'))
        fprintf('Please generate database ...
            first!\n');
    else
       pause(0.1);
       [fn fp] = uigetfile ...
           ({'*.pgm';'*.jpg';'*.png'});
       if fp ≠ 0 %no file path
           fname = [fp,fname];
           facerec (fname,myDatabase,minmax);
       end
    end
end
```

the forth choice is for getting a frame from the Webcam. We already developed a small function to capture the frame in previous section, here we only use it. Then we write the image file with a temporary and random file name and we pass the name of the file to `facerec.m` function for recognition.

```
if (choice == 4)
    I = getcam();
    if (¬isempty(I))
        fname = ...
            ['./',num2str(floor(rand()*10)+1),'.pgm'];
        imwrite(I,fname);
        if (exist('myDatabase','var'))
            facerec (fname,myDatabase,minmax);
        end
    end
end
```

The hardest part is the fifth choice for exiting the application.

```
if (choice == 5)
    return;
end
```

# 11 Last Words

At last, this program is finished. But I think you still need to practice many times until you get it right. I want to thank you personally for obtaining this ebook and reading up to here.

I have tried to gather the answers to many questions but if you have got any problem, please don't hesitate to contact me on my email. Here is my email again:

omid.sakhi@gmail.com

Looking forward to hearing from you and your progress.

Best Luck,
Omid