Implementation Note on Embedded Multiresolution Mixtures (EMM) in the MATLAB Environment

Roland Kwitt
Department of Computer Sciences
University of Salzburg
E-Mail: rkwitt@gmx.at
Source: http://www.wavelab.at/sources

July 23, 2009

In this technical note, we explain our implementation of the Embedded Multiresolution Mixtures (EMM), as they first appeared in a work by Vasconcelos and Lippman [5]. Further, we present the implementation of several similarity measures between the EMMs which are all more or less accurate approximations of the Kullback-Leibler divergence.

1 Notation

Before we go on, we introduce some notational conventions. All letters written in typewritten font, e.g. `A`,`b`, etc., denote MATLAB variables. Small boldface letters such as $\mathbf{x}$ denote vectors, big boldface letters denote matrices, e.g. $\mathbf{\Sigma}$.

2 Dataset Construction and GMM Learning

The basic building blocks of [5] are the Discrete Cosine Transform (DCT) and multivariate Gaussian Mixture Models (GMM). The idea is to move a sliding window of $N \times N$ pixel over the whole input image with a predefined stepsize of d pixel in both (i.e. vertical and horizontal) directions. Originally, a 8×8 window and $d = 2$ are proposed. At each new window position the DCT is applied and the first K coefficients are extracted. Since the original paper does not give a description of how to extract the **first** K coefficients, we adopt the commonly-known **zigzag scan** order. The whole principle is illustrated in Fig. 1. The corresponding MATLAB function is `[data,J] = dct2block(I,blksize,cnum,shift)`.

```
function [data,J] = dct2block(I,blksize,cnum,shift)
    szx = size(I,1);
    szy = size(I,2);
    cnt = 1;
    N = (szx-blksize)/shift + 1;
    J = zeros(blksize,blksize,N*N);
    for i=1:shift:szx-blksize+1
        for j=1:shift:szy-blksize+1
```

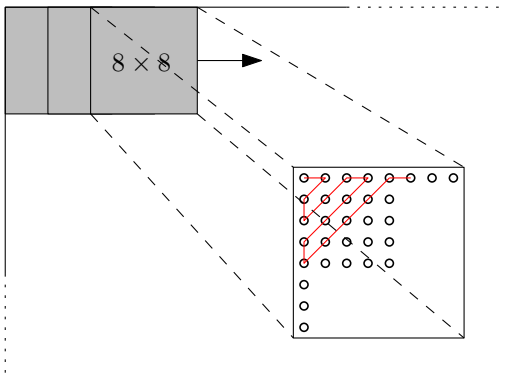

Figure 1: Illustration of the sliding 8×8 window and the zigzag-scan extraction of the first 16 DCT coefficients (including the DC coefficient).

```
            blk = I(i:i+blksize-1,j:j+blksize-1);
            J(:,:,cnt) = dct2(blk);
            cnt = cnt + 1;
        end
    end
    Z = zigzag(blksize);
    it = 1;
    for z = cnum
        tmp = [J(Z(z,1),Z(z,2),:)];
        data(:,it) = tmp(:);
        it = it+1;
    end
end
```

The `dct2block` routine arranges the K coefficients of all M sliding window positions in a $M \times K$ matrix $\mathbf{X}$ (return value `data` of the routine `dct2block`). Since the DCT possesses very good decorrelation properties (asymptotically close to the optimal decorrelating transform), the coefficients can be assumed uncorrelated. By assuming that each row-vector $\mathbf{x}_i \in \mathbb{R}^K$ of the matrix $\mathbf{X}$ is a realization of some underlying distribution, we can use $\mathbf{x}_1, \ldots, \mathbf{x}_M$ to fit a multivariate Gaussian Mixture Model (GMM) with C components. In the original paper the authors propose to use $C = 8$. In detail, the MATLAB Statistics Toolbox routine `gmdistribution.fit` is used to estimate the GMM parameters. Due to the fact that we assume no correlation between the coefficients, we can use a diagonal covariance matrix for each mixture component. The call to the `gmdistribution.fit` to fit the GMM is

```
...
obj = gmdistribution.fit(data, ...
8,'CovType','diagonal','Options',options, 'Regularize', eps);
...
```

which uses the default initialization of the `gmdistribution.fit` routine and regularizes the covariance matrices by adding a small constant value $\epsilon > 0$ to all diagonal elements. More advanced initialization will be discussed next.

2.1 Initialization of the EM Algorithm

A critical issue for running the EM algorithm to fit a GMM is the initialization of the GMM parameters. Each component of the GMM requires a weight, a mean vector and a diagonal covariance matrix. The starting parameters of the EM algorithm are initialized according to [4], where the author uses the Linde–Buzo-Gray (LBG, aka generalized Lloyd) algorithm [1] with the codeword splitting procedure proposed by Gray [3]. LBG terminates when the difference between the average distortions of two successive iterations is less than 10^{-3}. The initial mean vectors of the GMM are taken to be the codebook vectors of the LBG algorithm. All samples are then assigned to the nearest codebook vectors and the component-wise variance is used to initialize the diagonal elements of the GMM covariance matrices. The proportion of samples corresponding to each codebook vector are used as distribution weights. Next, we can run the EM algorithm, which terminates if either 200 iterations are reached or the log–likelihood difference between two successive iterations is less than 10^{-6}. the code snipplet for initializing the EM algorithm using LBG is

```
...
[mv,assignment] = vqlbg(block',components);
for a=1:components
    modelI.mu(a,:) = mv(:,a)';
    modelI.Sigma(1,:,a) = var(block(assignment == a,:));
    modelI.PComponents(a) = length(find(assignment == a))/length(assignment);
end
...
```

and the `vqlbg` algorithm (LBG) is implemented by the following routine. Note that `d` denotes the dataset and `k` denotes the desired number of codebook vectors. The output argument `r` gives the resulting codebook vectors and `ind` contains the assignment of each sample to one of the `k` codebook vectors.

```
function [r,ind] = vqlbg(d,k)
e    = .001;
r    = mean(d, 2);
dpr = 10000;
for i = 1:log2(k)
     r = [r*(1+e), r*(1-e)];
     while (true)
         z = disteu(d, r);
         [m,ind] = min(z, [], 2);
         t = 0;
         for j = 1:2^i
              r(:, j) = mean(d(:, find(ind == j)), 2);
              x = disteu(d(:, find(ind == j)), r(:, j));
              for q = 1:length(x)
                   t = t + x(q);
              end
         end
         if (((dpr - t)/t) < e)
              break;
         else
```

```
            dpr = t;
        end
    end
end
```

3 What are Embedded Multiresolution Mixtures ?

Regarding the notion of embedded multiresolution mixture model, we have to establish some theory first. The general GMM with C components is given by

$$P(\mathbf{x}) = \sum_{c=1}^{64} \pi_c \mathcal{G}(\mathbf{x}, \mu_c, \mathbf{\Sigma}_c), \tag{1}$$

where μ_c and $\mathbf{\Sigma}_c$ signify the mean vector and covariance matrix of the c-th mixture component. Vasconcelos recapitulates that by restricting a Gaussian distribution in $\mathbb{R}^n$ to $\mathbb{R}^k$ with $k < n$, it is still a Gaussian. This fact can be extended to Gaussian mixtures as well. Hence, if we extract the first k elements of each mean vector μ_c and the upper $k \times k$ matrix of each covariance matrix $\mathbf{\Sigma}_c$, we obtain an embedded mixture model. In the context of modeling the DCT coefficients this embedded mixture model is termed a embedded multiresolution mixture model (EMM), since the DCT is actually a multiresolution decomposition. In case $C = 1$, we simply approximate the DC coefficient histogram. As it is noted in the original paper, increasing the dimensionality captures more frequency information on the one hand, but we loose rotational invariance on the other hand. However, the most interesting point is, that we only have to estimate the EMM for all 64 coefficient of the 8×8 window **once** and can then simply reduce the dimensions by the aforementioned procedure. In our implementation the embedding process is accomplished by the function `em = emmembed(models,emNum)`. Note, that we try to capture cases where positive-definiteness is possible violated. This is done in this routine, so that the MATLAB function `gmdistribution.fit` does not complain and exits. The routine returns the EMM in the variable `em`.

```
function em = emmembed(models,emNum)
    for i=1:size(models,2)
        newSigma = models{i}.Sigma(:,1:emNum,:);
        [d1,d2,d3] = size(newSigma);
        for j=1:d3
            % check for entries which indicate non positive definiteness
            [maxi,indmaxi] = max(newSigma(:,:,j));
            [mini,indmini] = min(newSigma(:,:,j));
            if (mini < maxi*eps)
             newSigma(:,indmini,j) = maxi*eps;
            end
        end

        em{i} = gmdistribution(models{i}.mu(:,1:emNum), ...
            newSigma, ...
            models{i}.PComponents);
    end
end
```

4 Example

This example loads the 128×128 image `Bark.0000.01.tif` of the MIT VisTex database, extracts the first 16 DCT coefficients of each 8×8 sliding window (step size $d = 2$) and then fits a GMM using $C = 8$ components with the default initialization routine (i.e. random, type `help gmdistribution.fit'` for further explanations). We subsequently construct a EMM of the first 8 DCT coefficients using `emmembed`.

```
>> I = double(imread('/data/pictures/Bark.0000.01.tif'));
>> data = dct2block(I,8,1:16,2);
>> model = gmdistribution.fit(data,8, ...
'CovType', ...
'diagonal', ...
'Options', ...
options);
53 iterations, log-likelihood = -328622
>> model = emmembed({obj},8)
```

5 Measuring Image Similarity

As a last point, we explain how to measure image similarity in the framework of EMMs. We implement the Feature-Likelihood (FL) originally proposed in [5] and two approximations of the Kullback-Leibler divergence between two GMMs: an approximation proposed by Goldberger et al. [2] and the Asymptotic Likelihood Approximation (ALA) of Vasconcelos [4]. Further, we provide code for performing a Monte-Carlo simulation to approximate the KL divergence.

5.1 MC Approximation of the KL Divergence

Let p and q denote the pdfs of two C component EMMs of a certain dimensionality. The MC simulation approach is quite simple, since it exploits the fact that the term

$$\frac{1}{N} \sum_{s=1}^{S} [\log p(\mathbf{x}_s) - \log q(\mathbf{x})] \tag{2}$$

converges to the KL-divergence between p and q, given a random sample from p. In our implementation the overloaded MATLAB function `random` is used to draw a such a random sample from a given model. The pdf of each EMM using the data samples $\mathbf{x}_1, \ldots, \mathbf{x}_S$ is evaluated by using the overloaded MATLAB function `pdf`. In MATLAB syntax, assuming that `A` and `B` denote two fitted EMMs, a full example using MC simulation reads as

```
>> S = 10000;
>> r = random(A,S)
>> 1/S * sum(log(pdf(A,r))-log(pdf(B,r)));
```

The MC simulation appears in the function `rankemm` which is not listed here, since it is a wrapper to evaluate similarities between all models.

5.2 ALA and Goldbergers Approximation

In order to compute the ALA and Goldberger's approach, we need to know how to access the mean vectors and covariance matrices of the fitted models. MATLAB returns a structure when `gmdistribution.fit` is called which holds all the necessary information. To access the covariance matrix of the i-th component, we write `model.Sigma(:,:,i)` to obtain the diagonal elements as a row vector. To get a matrix, we write `diag(model.Sigma(:,:,i))`. The mean vectors are extracted in a similar fashion using `model.mu(i,:)` to obtain the mean vector μ_i. The weights π_i are stored in the structure field `PComponents`. Hence, `A.PComponents(i)` gives π_i. This is all we need to implement both approximations. Again, let p and q denote the pdfs of two EMMs and let p_i and q_i denote the pdfs of the components. Further, α_i, β_i denote the weights of each component. Both [4] and [2] have the general form

$$D(p||q) = \sum_{c=1}^{C} \alpha_c \left(D(p_i||q_{r(c)}) + \log \frac{\alpha_c}{\beta_{r(c)}} \right) \tag{3}$$

and differ only in the so called **alignment** function $r(c)$. Vasconcelos [4] defines $r(i)$ as

$$r(i) = \arg\min_j \left((\mu_{p,i} - \mu_{q,j}) \mathbf{\Sigma}_{q,j}^{-1} (\mu_{p,i} - \mu_{q,j})^T \right) \tag{4}$$

whereas Goldberger defines $r(i)$ as

$$r(i) = \arg\min_j \frac{1}{2} \left(\log \frac{|\mathbf{\Sigma}_{q,j}|}{|\mathbf{\Sigma}_{p,j}|} + \operatorname{trace}\left(\mathbf{\Sigma}_{q,j}^{-1} \mathbf{\Sigma}_{p,j} \right) + (\mu_{p,i} - \mu_{q,j}) \mathbf{\Sigma}_{q,j}^{-1} (\mu_{p,i} - \mu_{q,j})^T - N \right) - \log \beta_j, \tag{5}$$

where N denotes the dimensionality of the EMMs. Both approaches are implemented in the MATLAB function `klapprox` which takes as arguments the two fitted EMMs `Q` and `I`, as well as the number of EMM components `C`, the dimensionality `N` and the method to use. `method` can either be `'vasc'` or `'gold'`.

```
function D = klapprox(Q,I,C,N,method)
    D = 0;
    for j=1:C
        switch method
            case 'vasc'
                d(j) = vascalign(Q,I,C,j);
            case 'gold'
                d(j) = goldalign(Q,I,C,j,N);
        end
        D = D + Q.PComponents(j) * ...
            (kldivgauss(Q.mu(j,:),I.mu(d(j),:), ...
            diag(Q.Sigma(:,:,j)),diag(I.Sigma(:,:,d(j))),N) +  ...
            log(Q.PComponents(j)/I.PComponents(d(j))) ...
            );
    end
end
```

The alignment function for Goldbergers approximation [2] is implemented as

```
function ind = goldalign(Q,I,C,i,N)
    dist = [];
    for j=1:C
        dist(j) = kldivgauss(Q.mu(i,:),I.mu(j,:), ...
            diag(Q.Sigma(:,:,i)),diag(I.Sigma(:,:,j)),N) - log(I.PComponents(j));
    end
    [mini,ind] = min(dist);
end
```

and Vasconcelos' [4] alignment function is implemented as

```
function ind = vascalign(Q,I,C,i)
   x = Q.mu(i,:);
   for j=1:C
       y = I.mu(j,:);
       S = diag(I.Sigma(:,:,j));
       dist(j) = (x-y)*inv(S)*(x-y)';
   end
   [mini,ind] = min(dist);
end
```

5.2.1 Running the C Code

Since both approximations are computationally expensive although they just use the GMM parameters (due to the alignment process), we provide C source code for both approximation in order to run similarity measurement between all models and compute a distance matrix. However, this requires to store the fitted EMMs on the harddisk. Note that our C source code only deals with diagonal covariance matrices. The routine to dump the EMMs to harddisk is `dump_emmsimple` which is called as follows: the EMMs have to be stored in a cell array which contains the fitted `gmdistribution` objects. Lets call this cell array `fe` and assume that it contains EMMs with $C = 8$ components and the dimensionality is 60. The directory where we dump the models is `/tmp/emm` (has to exists) and the `offset` (last parameter) is 0. This means that we dump all models. In case the `offset` is > 0, we start dumping models at model index `offset`.

```
 >> dump_emmsimple(fe,8,60,'/tmp/emm',0);
```

The directory then contains $C{\cdot}2{+}1$ files for each model. For `model0` and $C = 8$ for example, we obtain: `model0.0.mu`, ..., `model0.7.mu`, `model0.0.cov`,...,`model0.7.cov`, `model0.pi`. These files are used by the binary `emmsimple` to compute Goldbergers approximation. The source code can be found in the `sm` subdirectory and is compiled by running

```
$ make -f Makefile.emm
```

which gives the binary `emmsimple`. Typing `emmsimple --help` lists all available input parameters. In case of a $C = 8$ component EMM with dimensionality 60 we first copy this binary to the directory `/tmp/emm` (from the example above) and then run (Goldberger approximation)

```
$./emmsimple -C 8 -P 60 -v
```

to obtain a distance matrix `dist.bin`. To use the alignment proposed by Vasconcelos use the binary `ala` with the same parameters. The switch `-v` turns on verbose output messages. You

can further obtain a symmetric version of the approximations by using the switch `-S 1`. The implementation of the symmetric version simply swaps the models and averages the distance (this is the standard way to artificially symmetrize the KL divergence). The resulting distance matrix `dist.bin` can then be further processed in C, or loaded in MATLAB. For C processing, we note that the matrix is written using the GSL routine `gsl_matrix_fwrite` and can easily be read using `gsl_matrix_fread`. In MATLAB, the distance matrix can be loaded using

```
>> fid = fopen('/tmp/emm/dist.bin','rb');
>> A = fread(fid,'double');
>> fclose(fid);
>> A = reshape(A,[sqrt(length(A)) sqrt(length(A))]);
```

5.3 Feature Likelihood

Last, we discuss how to measure image similarity using the Feature-Likelihood proposed in [5]. The idea is very simple: instead of computing a DCT on each overlapping 8×8 subwindow, we compute a DCT on all non–overlapping 8×8 subwindows of a query image and extract the same coefficients as we uses during EMM fitting. Hence, we obtain a considerably smaller amount of feature vectors. These vectors are plugged into the pdf of the each fitted EMM and the likelihood (or more specifically the log-likelihood) is computed. Summing up this value over all extracted feature vectors gives the final feature likelihood. This measure is then used as a similarity criterion between the query image and a candidate image (or more specifically its EMM model). To obtain the feature vectors, use the MATLAB function `exzz` together with the MATLAB-intern function `blkproc`. Next, we show an example of how to compute the feature vectors for a color image using 8×8 blocks for the DCT (assume the image was loaded into `I` using `imread`).

```
>> for c=1:size(im,3)
    J(:,:,c) = blkproc(I(:,:,c),[8 8],fun);
end
>> X = exzz(J,'blk',8,'coefficients',1:16,'debug',debug);
```

The variable `X` now contains the feature vectors (each row is a feature vector of dimensionality 48). The number of vectors depends on the original image size and the blocksize of course. Given `X`, you can simply calculate the feature likelihood under model `modelA` using

```
>> fl = sum(log(pdf(modelA,X)));
```

In order to use the feature likelihood in the same way you can use `emmsimple` and `ala` for example, we provide a MATLAB routine `emmfl` (EMM Feature Likelihood). This routine accepts a cell array of `gmdistribution` objects as a first argument. Optional parameters, such as preprocessing, etc. can be listed by typing `help emmfl`. An exemplary call of `emmfl` using the EMM models stored in `models` for example is as follows:

```
>> D = emmfl(models,'debug',true);
```

References

[1] A. Gersho and R. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

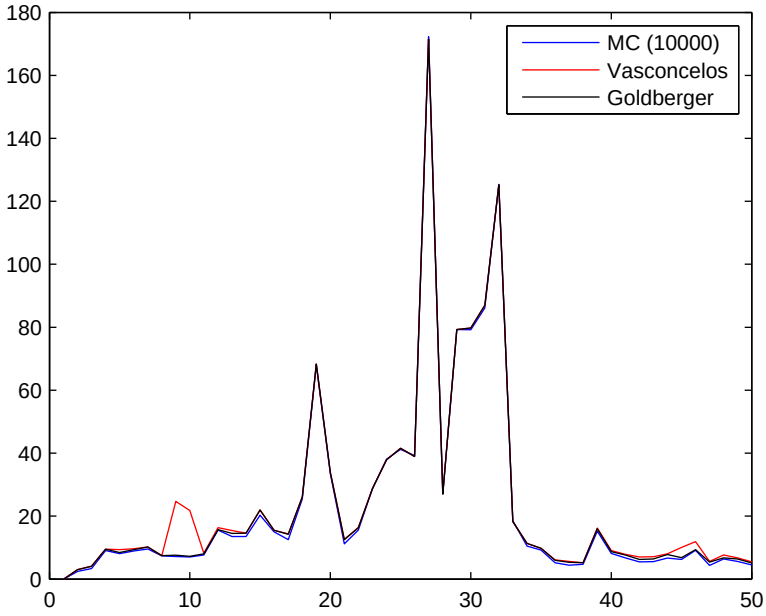

Figure 2: Visualization of the approximations of Vasconcelos and Goldberger compared to the Monte-Carlo simulation with 10000 samples for similarity measurement between one fitted EMM model and 50 candidate EMM models

[2] J. Goldberger, S. Gordon, and H. Greenspan. An efficient image similarity measure based on approximations of the KL-divergence between two Gaussian mixtures. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'03)*, pages 487–493, Nice, France, 2003.

[3] R. Gray. Vector quantization. *IEEE Accoustics, Speech, and Signal Processing (ASSP) Magazine*, 1(2), April 1984.

[4] N. Vasconcelos. On the efficient evaluation of probabilistic similarity functions for image retrieval. *IEEE Transactions on Information Theory*, 50(7):1482–1496, July 2004.

[5] N. Vasconcelos and A. Lippman. A probabilistic architecture for content-based image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'00)*, pages 216–221, Hilton Head, South Carolina, USA, 2000.