Brain MRI Classification using the Expectation Maximization Algorithm

Kamil Bojanczyk
Adarsh Kowdle

Abstract:

We have made a brain magnetic resonance image (MRI) classification algorithm that uses a two-stage expectation maximization (EM) algorithm. The algorithm was tested on a phantom data set before being applied to a set of normal brain MR images for further testing. We accomplished a working expectation maximization algorithm for normal brain MRI data sets. Due to the complexity of the problem, we restricted our data sets to normal brains without any lesions, cancer growths, or other abnormalities. Without this restriction, we would require a whole new data set for training and a more complex algorithm. A Gaussian mixture model (GMM) was constructed for three brain labels, namely white matter (WM), gray matter (GM), and cerebral spinal fluid (CSF). This model and the bias field parameters are updated as we iterate through the EM loop until we reach desirable results. The results of the algorithm showed that we correctly segmented white matter, gray matter, and cerebral spinal fluid with an un-normalized overall accuracy of 74.32%. We have found complications with the existing image data set; mainly that not all the images were segmented beforehand, disallowing for a larger data set to compare against. The algorithm worked better than our baseline low pass filter approach by 16.61% for correctly segmenting each class of WM, GM, and CSF.

Introduction:

The brain is the center of the nervous system and is composed of three types of material: white matter (WM), grey matter (GM), and cerebral spinal fluid (CSF). Segmenting the MR images into these three categories has clinical significance: it is vitally important in the classification of brain structure. Quantitative analysis of these images requires a large amount of time, thus the necessity for an automated segmentation approach becomes clear when looking at this volume of data collected. Along with being time consuming, manual segmentation on a large data set has large inter and intra observer variability.

Once segmentation has been performed, these models can be used for quantitative studies of the brain, such as the volumes of each matter class in order to discern whether an anomaly is present. Segmented images of the brain also aid in three-dimensional visualization for pre and post operational surgery results.

Intensity-based segmentation of the brain MRI is a well know approach for automated classification of the brain into the required WM, GM and CSF classes. However, inhomogeneities in the magnetic field (bias field) result in intensity variations, thus directly relying on intensities results is an inaccurate classification. A better approach is to handle this classification as a machine learning problem. The aim is to have the system learn parameters that describe the various classes from the available data, resulting in better classification.

The main problem when reconstructing MR images is not noise; rather, it is the inhomogeneity in the image [1]. The inhomogeneity is caused by the limitations of the MRI machine and patient-induced electrodynamic interactions. This bias field is not always visible to the human eye, but it can cause serious misclassifications when run through an intensity-based segmentation algorithm. Another problem is finding a good initialization point for the algorithm. It has been shown by Styner that finding the correct initialization point is essential in returning reliable results. We plan to bypass this problem by manually selecting WM, GM, and CNF classes [1]. This idea is adapted from Dawant *et. al.* [3].

Methods and Materials:

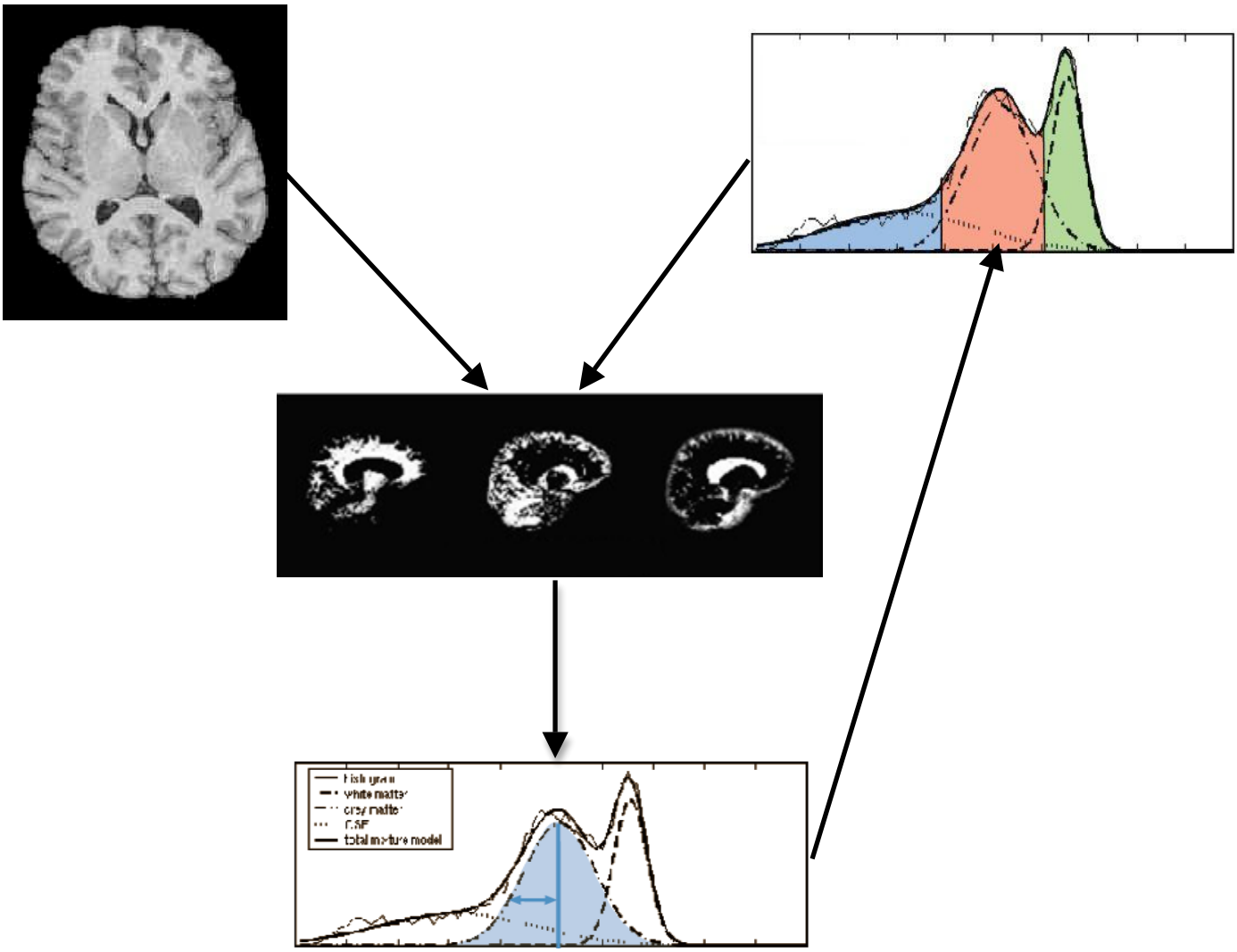
We used the expectation maximization algorithm with bias field correction for our experiment. We chose the EM algorithm because of it was shown to work well in the literature and it is an iterative approach, allowing for a hopefully converging solution. Our algorithm was adapted from Leemput *et. al.* [5]. We used polynomial fitting for the bias field because of literature reviews and because we wanted to advance the algorithm brought about by the 2008 class project on MRI.

The objective of our experiments was to correctly classify our image data sets by more than 85% for WM, GM, and CSF. We also wanted to compare the results to the baseline method and hope to achieve better results. We predicted the outcome to be within 10% better than the baseline. After modification of our initial algorithm we found that our new outcome should be above the results of the previous years' results, but below the three-step approach to the EM algorithm with bias field correction.

Description of algorithm

An interesting algorithm brought out by Leemput *et. al.* is an extended expectation maximization algorithm (described as a three stage EM algorithm) [5]. The extended EM algorithm aims at parameterizing the different classes based on intensities while at the same time determining parameters that describe the bias field. With an iteration of the EM algorithm, the mixture model parameters and the bias field parameters are updated and perform the classification. In order to understand the EM algorithm, it is important to understand the two main stages that make up the algorithm, shown in **Figure 1**. (Note that EM is an iterative process, so the stages described below are at an intermediate iteration of the algorithm).

Stage I: **Classification (Expectation step)**



Stage II: Update the mixture model (Maximization step)

Figure 1: Illustration of the two stages of the EM algorithm.

Stage I - **Classification**: At this stage of the loop, we assume that we have an estimate of the parameters that describe the mixture of K Gaussians as shown in **Figure 2**, where K is the number of classes (here GM, WM, and CSF). This mixture model provides the likelihood for every voxel in the MRI, which allows for statistical classification of every voxel into one of these K classes, achieving the classification required as part of the loop. Note that the intensity is the effective intensity after the current bias field *estimate* has been subtracted from the intensity

at each voxel. Gaussian mixture models are used because we can represent the various classes of the brain as Gaussians with the mean equal to mean intensity.

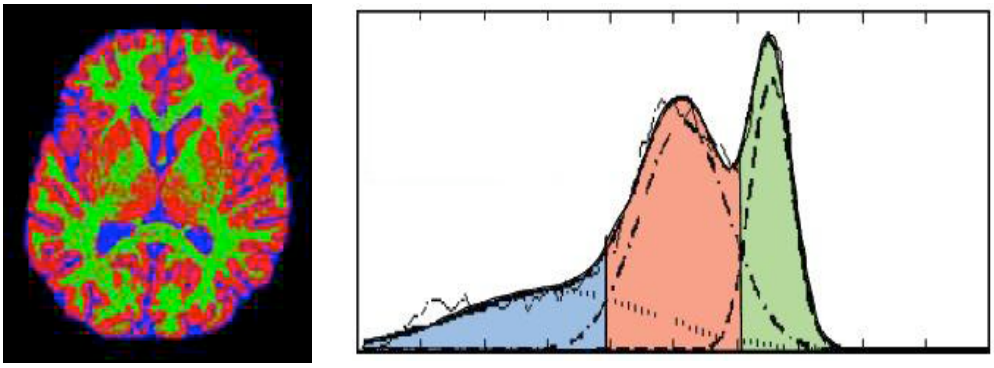


Figure 2: Histogram of intensities modeled as a mixture of three Gaussians. (Intensity on the X axis and pixel count on Y axis) Blue: CSF, Red: Grey matter, Green: White matter

Stage II - The mixture model and classification: Using the current classification of the MRI, we model the Gaussian distribution describing a class with mean intensity μ and variance σ^2. The voxel intensities corrected by the bias field are used to update the mixture model. In our approach, K=3, corresponding to WM, GM, and CSF. The intensity distributions of each class are modeled as a Gaussian distribution with the statistical mean being average intensity, and variance being variation around the average intensity. Once the mean and the variance of each tissue type are known, the voxels can be classified into each class based on voxel intensity.

The EM algorithm ties together the classifications of stage I and class distribution parameter estimation of stage II. The algorithm fills in the missing data during stage I and then finds the parameters that increase the likelihood for the complete data during stage II.

Detailed explanation of the EM algorithm

We are using an intensity-based model when classifying and segmenting our images, as opposed to a spatial model. An explanation of the two steps of the expectation and maximization algorithm is described below.

Expectation step:

This step involves the classification of the input data into the 3 classes using the current estimate of the mixture model.
The likelihood for each of the three classes is given by,

$$p(y_i|\Gamma_i = j, \theta_j) = G_{\sigma_j}(y_i - \mu_j)$$

Hence, the posterior, which determines the class of the voxel, is given by,

$$p(\Gamma_i = j \mid y_i, \theta) = \frac{p(y_i \mid \Gamma_i = j, \theta_j) p(\Gamma_i = j)}{\sum_j p(y_i \mid \Gamma_i = j, \theta_j) p(\Gamma_i = j)}$$

Where:

G_{σ_j} = Gaussian modeling class j

Γ_i = class of i^{th} voxel

μ_j = mean intensity of the j^{th} class

σ_j = standard deviation of the j^{th} class

(Note: The prior for each class was determined as the normalized ratio of voxels of each class with respect to all the voxels in the image, in the training data)

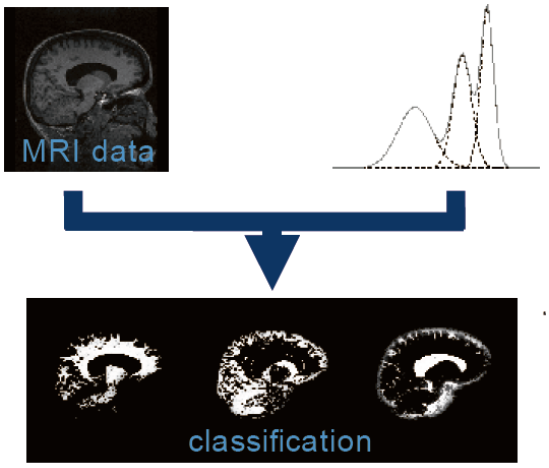


Figure 3: Expectation step. The statistical classification of each voxel based on the current image parameter estimation.

Maximization step:

This step involves estimating the new parameters for the mixture model given the new likelihoods for each class, based on the expectation step. The parameters in the Gaussian mixture model are re-estimated based on the current classification.

The updating step is given by,

$$\mu_j = \frac{\sum_i y_i p(\Gamma_i = j \; y_i, \theta)}{\sum_i p(\Gamma_i = j \; y_i, \theta)} \qquad \sigma_j^2 = \frac{\sum_i p(\Gamma_i = j | y_i, \theta)(y_i - \mu_j)^2}{\sum_i p(\Gamma_i = j \; y_i, \theta)}$$

which gives the new mean and variance, respectively.

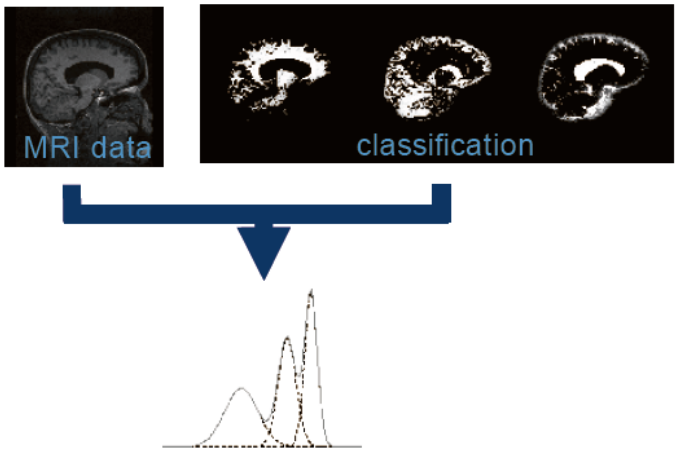


Figure 4: Maximization step. The parameters are updated in this step, based on the previous parameters.

The expectation step and maximization step described above are iterated until the desired result is achieved.

Experiments:

We used the phantom data available in the ECE dataset and simulated a bias field using known Gaussian noise. This experiment will test for a working EM algorithm. After we have tested that our EM algorithm works, we will use real data sets with MR brain images (ECE dataset). We will compare the results of the classification from the EM algorithm with the ground truth, which are manually segmented images of the brain available as part of the dataset. We will check the normalized accuracy to evaluate the class accuracy and overall un-normalized accuracy of the resulting classification to see how well the algorithm performs. In order to rate the algorithm, we plan to compare it with a low pass filter and illumination threshold-based brain MRI classification.

Dataset

In our project, we will use the data from the Harvard Medical School brain database. The 20 normal MR brain data sets and their manual segmentations were provided by the Center for Morphometric Analysis at Massachusetts General Hospital. The coronal three-dimensional T1-weighted spoiled gradient echo MRI scans were performed on two different imaging systems.

- Ten FLASH scans on four males and six females were performed on a 1.5 tesla Siemens Magnetom MR System (Iselin, NJ) with the following parameters: TR = 40 msec, TE = 8 msec, flip angle = 50 degrees, field of view = 30 cm, slice thickness = contiguous 3.1 mm, matrix = 256x256, and averages = 1.

- Ten 3D-CAPRY scans on six males and four females were performed on a 1.5 tesla General Electric Signa MR System (Milwaukee, WI), with the following parameters: TR = 50 msec, TE = 9 msec, flip angle = 50 degrees, field of view = 24 cm, slice thickness = contiguous 3.0mm, matrix = 256x256, and averages = 1.

Segmentation was performed on the positional-normalized scan by trained investigators using a semi-automated intensity contour mapping algorithm and also using signal intensity histograms.

The previous batch reduced the voxel intensity range from 16-bits to 8-bits. We planned to use the 16-bit data as reducing the range amounts to losing a lot of valuable information, however we discovered (as will be discussed below) that the 16-bit data had complications with a lack of segmentation and poor modeling of the GMM.

Results:

We began by estimating the Gaussian mixture model parameters by training on ten of the 8-bit images. After getting these values, the parameters were used to test our algorithm on the phantom brain slices and we found that our algorithm classified the slices correctly. Finally, we tested our algorithm on the remaining ten 8-bit voxel intensity images, and our results were as follows.

Our baseline algorithm was a lowpass fitler with illumination thresholding. The thresholds values were taken from our GMM training step. The tables below summarize our results.

A confusion matrix was used for the performance metric. After finding the absolute number of voxels classified as WM, GM, and CSF, the normalized confusion matrix was created in order to show the accuracy of our results on a percentage basis. Tables 1-4 show the confusion matrices and normalized accuracies for our algorithm and for the baseline algorithm, respectively.

Figures 4-6 show the three classes on an entire MR image for the ground truth, baseline algorithm output, and our EM algorithm output, respectively.

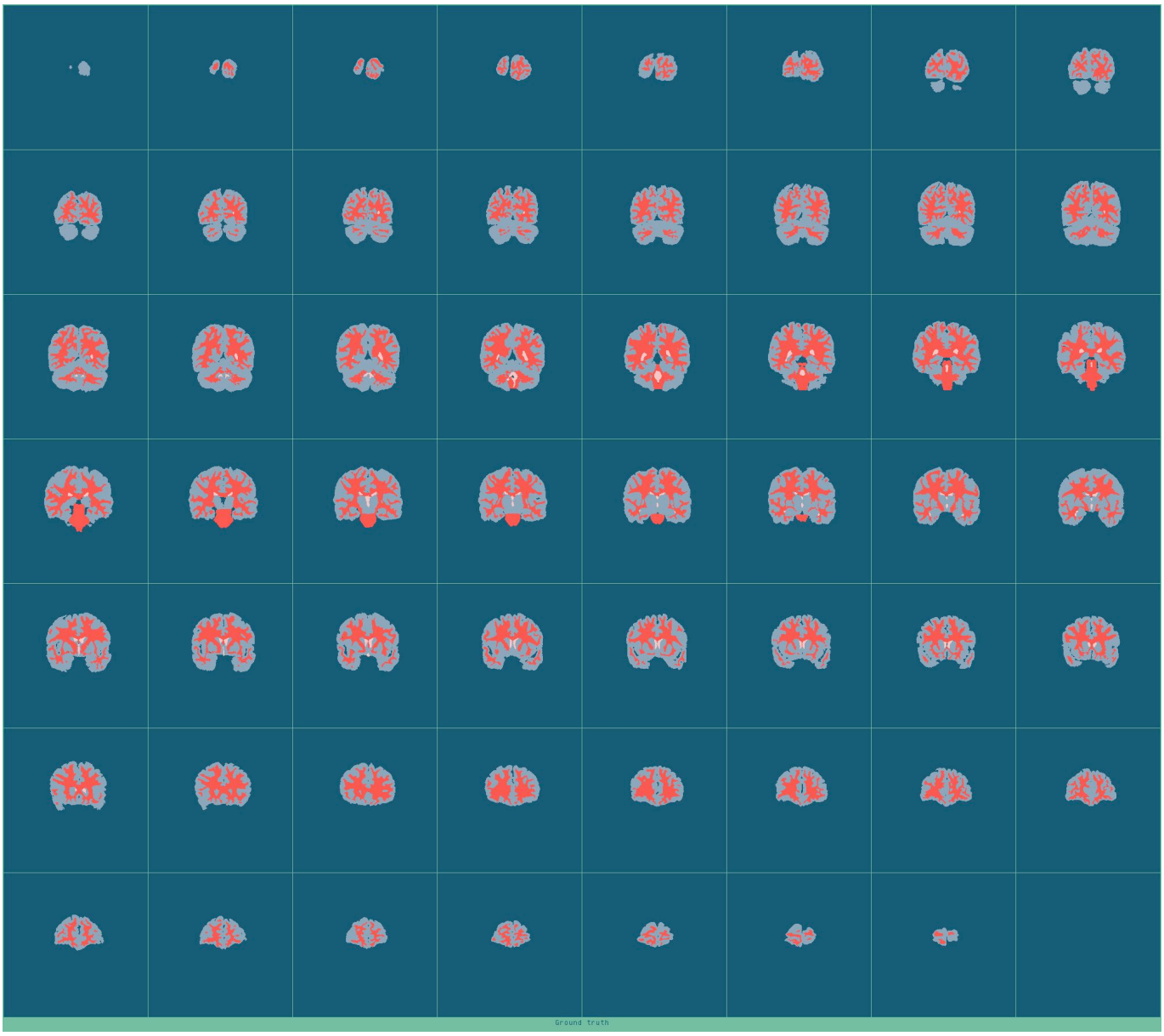


Figure 5: Tile showing ground truth classification for WM(red), GM(grey), and CSF(pink).

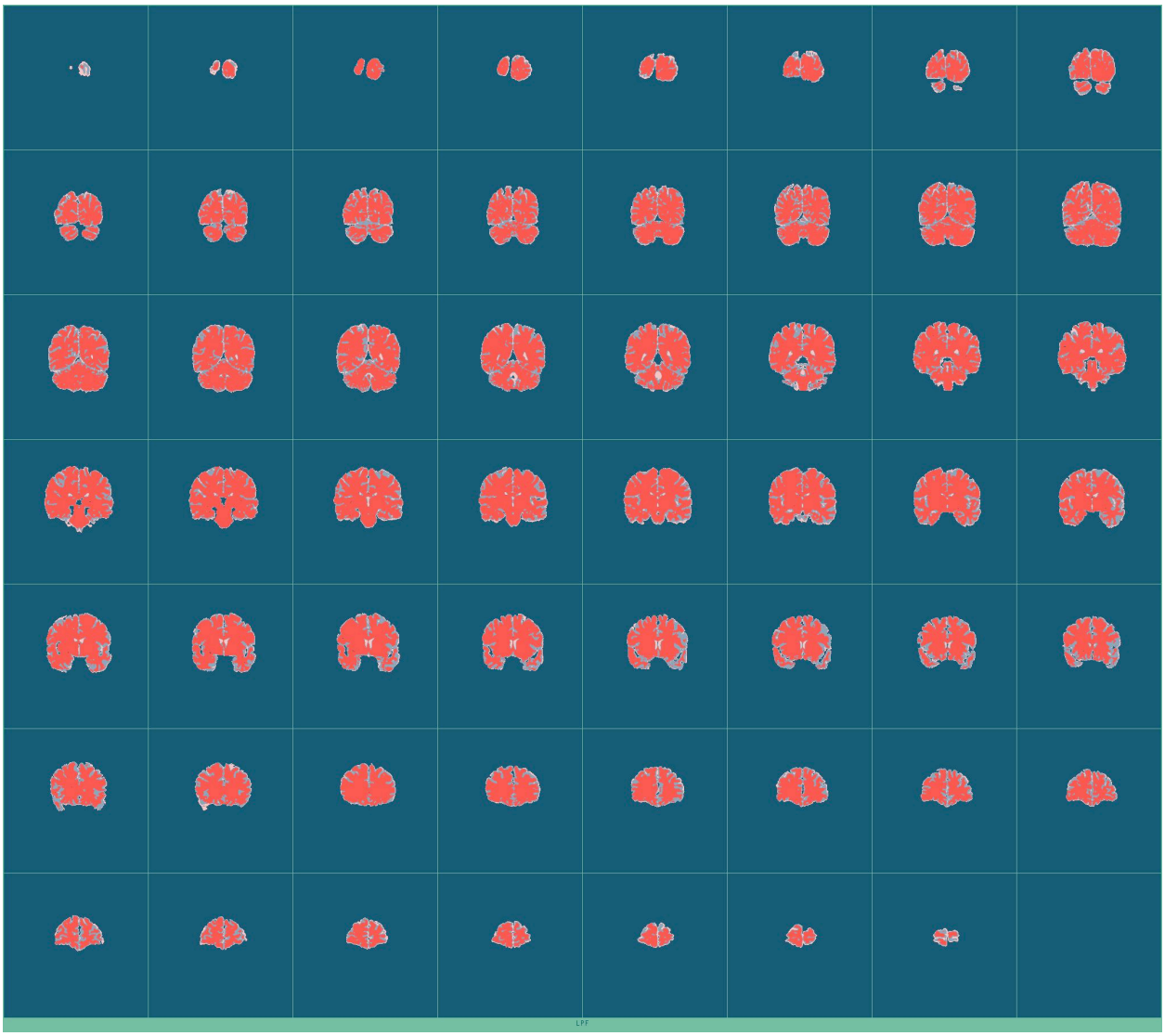


Figure 6: Tile showing the output of the baseline algorithm for WM(red), GM(grey), and CSF(pink).

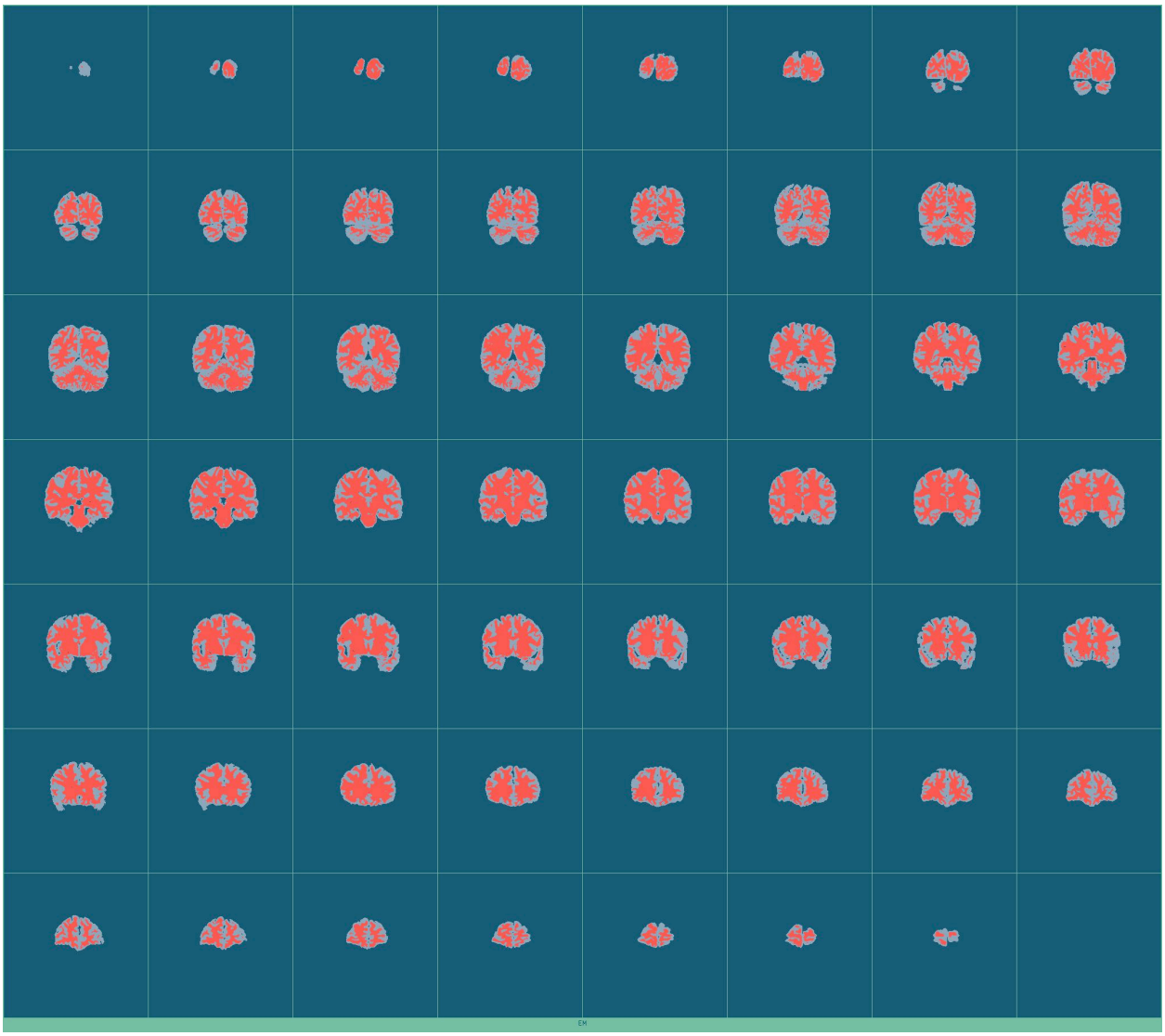


Figure 7: Tile showing the output of our EM algorithm for WM(red), GM(grey), and CSF(pink).

After performing our calculations, the results are shown below in the following tables and figures.

EM Algorithm

Confusion Matrix		Predicted class				
		WM	GM	CSF		Sum of rows
Actual class	WM	573619	123191	2		696812
	GM	329637	819683	468		1149788
	CSF	905	27615	1276		29796

Table 1: Confusion matrix for the EM algorithm, showing the voxels classified.

Normalized Accuracy	WM	GM	CSF
WM	82.3205	17.6792	0.0003
GM	28.6694	71.2899	0.0407
CSF	3.0373	92.6802	4.2825

Table 2: Normalized accuracy for the EM algorithm. The formula is: conf_mat/ (sum(each row of conf_mat)) *100.

Un-normalized accuracy: sum(diagonal elements of the confusion matrix) / sum(all elements of confusion matrix) * 100 = **74.32%**

Baseline Algorithm

Confusion Matrix		Predicted class				
		WM	GM	CSF		Sum of rows
Actual class	WM	590986	92859	12967		696812
	GM	473031	468785	208112		1149788
	CSF	1386	5238	23205		29796

Table 3: Confusion matrix for the baseline algorithm, showing the voxels classified.

Normalized Accuracy	WM	GM	CSF
WM	84.8128	13.3263	1.8609
GM	41.1357	40.7665	18.0978
CSF	4.6465	17.5601	77.7934

Table 4: Normalized accuracy for the baseline algorithm. The formula is: conf_mat/ (sum(each row of conf_mat)) *100.

Un-normalized accuracy: sum(diagonal elements of the confusion matrix) / sum(all elements of confusion matrix) * 100 = **57.71%**

The results of our experiment imply that our EM algorithm is more effective than our baseline approach when segmenting matter into three categories. With the average accuracy for WM, GM, and CSF 74.32%, we have performed better than the baseline method by 16.61%.

Discussion:

One of the experiments we carried out was to study the convergence of the Gaussian mixture model using the expectation maximization approach. To begin, the experiments were tested out on phantom brain slices to ensure that the expectation maximization loop was working. The brain slices were classified into the three classes accurately, as was expected. While testing the EM algorithm on actual brain MRI data it was observed that with every new slice of data the GMM parameters were refined. The initial mixture model learnt from the training images resulted in three Gaussians shown in figure 7. After ten iterations of the EM algorithm, the variance of the three Gaussians reduces, representing the three brain classes in a better and well separated as shown in figure 8, thus proving that our EM algorithm was working as expected and the results obtained were promising.

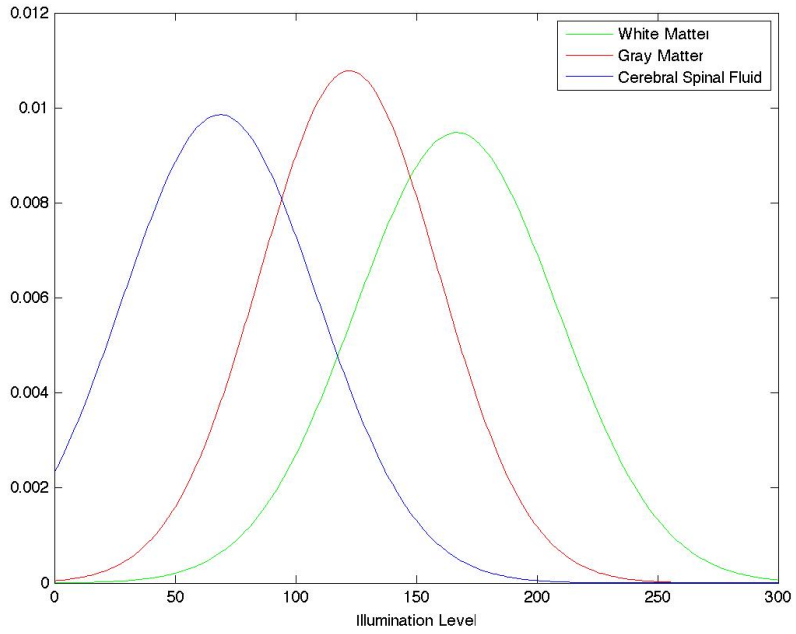


Figure 7: A histogram of the three classes prior to the EM algorithm.

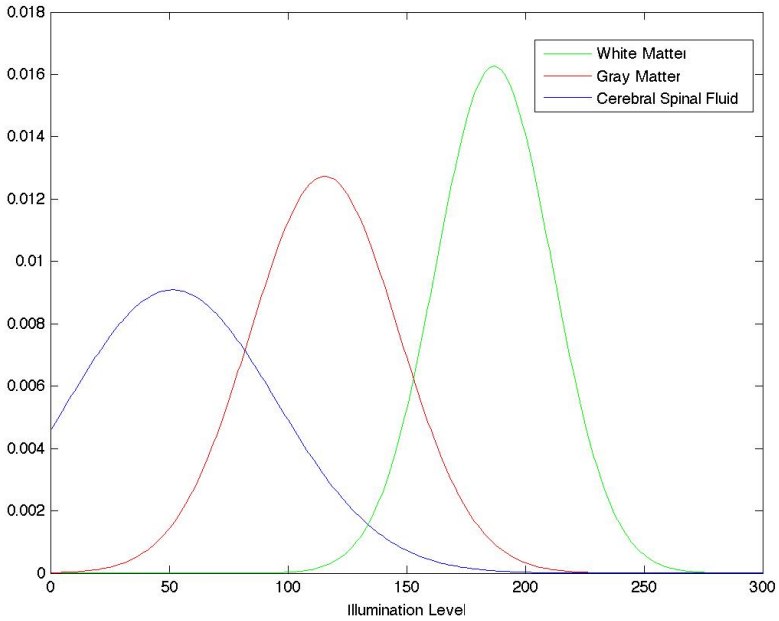


Figure 8: A histogram of the three classes after the EM algorithm was performed on it.

The dataset we planned to use was the 20 brain MRI images in 16-bit precision. Unlike the 8 bit data, the brain was not separated out from the rest of the image in the 16-bit images. We used the ground truth to extract the brain from the 16 bit images. After doing so, we realized that the ground truth for the brain images have a frame offset i.e. there are frames at the start of the MRI which do not have a corresponding ground truth. Since, correcting the ground truth was a tedious task, we instead created a lookup table (a text file), which tells you the start frames in the image and the ground truth, which is used in the processing.

The 16-bit brain data was extracted from the 16-bit MRI. This new data was then used to train the Gaussian mixture model (GMM). It was observed that the Gaussians obtained for gray matter and white matter were very close to each other thereby resulting in very poor classification, classifying the whole slice as either white or gray matter at times. On further exploration it was found that, the 16-bit data segmented did not span the whole 16-bit dynamic range i.e. from 0 to 65535. This means there would be a lot of gaps in the bins when creating the histogram of the 65536 illumination levels implying that the distribution of white matter and gray matter had a lot of variance, resulting in heavily overlapping distributions of white and gray matter. This was the reason why we could not work with this GMM. The 8-bit data, on the other hand, has been obtained by scaling the dynamic range of illumination levels to 0-255 levels. In other words, because every bin in the 8-bit data corresponds to 256 bins in the 16-bit data, hence the well-behaved distribution seen with the 8-bit data was not seen with the 16-bit data.

We used 10 test cases out of 20 were used for training. The Gaussian mixture model parameters were estimated using the 10 training images and their ground truths. A

considerable amount of variance was present in the data for all three classes, confirming the fact that we cannot use the parameters obtained directly, since defining thresholds between Gaussians with large variance is a hard task. The EM step as shown before helped as expected in clarifying and defining these thresholds.

Conclusion:

We took an EM algorithm approach to classification of MR Images of the brain. The decision for this algorithm over others was because of previous results in the literature and because of its iterative approach. We initially were going to include the third step of bias field correction, however we ran into complications with the image data, limiting us to completing the two step algorithm. The two step algorithm begins first with the expectation step. The expected value of each voxel is calculated, separating the image into WM, GM, CSF, or other. Next, the maximization step finds the most likely class that these voxels belong to. This is an iterative process, therefore the algorithm loops around until we go to a converging solution where the values for each class have little to no change.

After performing our two-step algorithm, a confusion matrix was used to evaluate our accuracy because it divides up the accuracy on a per-class basis. This sort of metric is important because it will determine if the algorithm works well for segmenting CSF, WM, or GM correctly. We correctly performed our EM algorithm with classification on ten images. We found out that we had an accuracy of 82.32%, 71.29%, 4.28% for WM, GM, and CSF, respectively, and an average accuracy of 74.32%. This is better than our baseline approach that had an average accuracy of 57.71%. To further improve results, a working bias field estimator is the next logical step to take. Also, running training on a larger data set with a complete set of ground truths associated with it will allow us to have more accurate results because our GMM estimation needed more images in order to correctly estimate the WM field.

References

[1] M. Styner, C. Brechbuhler, G. Szekely, G. Gerig, "Parametric estimate of intensity inhomogeneities applied to MRI" *IEEE Transactions on Medical Imaging*, vol. 19, no. 3, pp. 153-165, March 2000.

[2] Guillemaud, R.; Brady, M., "Estimating the bias field of MR images," *Medical Imaging, IEEE Transactions on* , vol.16, no.3, pp.238-251, June 1997.

[3] B. M. Dawant, A. P. Zijdenbos, and R. A. Margolin, "Correction of intensity variations in MR images for computer-aided tissue classification", *IEEE Trans. Med. Imag.*, vol. 12, pp. 770–781, Dec. 1993 (taken from Van Leemput).

[4] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1993. (taken from Guillemaud).

[5] Van Leemput, K.; Maes, F.; Vandermeulen, D.; Suetens, P., "Automated model-based bias field correction of MR images of the brain," *Medical Imaging, IEEE Transactions on* , vol.18, no.10, pp.885-896, Oct. 1999.

[6] W.M. Wells, W.E.L. Grimson, R. Kikinis, F.A. Jolexz, "Adaptive Segmentation of MRI Data," *IEEE Transactions on Medical Imaging* , vol 15, no. 4, pp. 429-443, August 1996.

Code:

The GMM training file:

```
/*****************************************************************/
/* Example VisX4 program gmm_training                            */
/*         Compute function on a 3D image structure              */
/* Syntax:                                                       */
/*             gmm_training im_filepath = images path            */
/*                          gt_filepath = ground truth path      */
/*                          [-v]                                 */
/*****************************************************************/
/*****************************************************************/
/*  This Software is Copyright (c) 1996, 2000 Anthony P. Reeves  */
/*  All rights reserved.                                         */
/*****************************************************************/

#include "VisXV4.h"        /* VisX structure include file     */
#include "Vutil.h"         /* VisX utility header files       */
#include "stdio.h"
#include "math.h"

VisXfile_t *VXim,          /* image file structure       */
  *VXgt;          /* ground truth file structure     */
VisXelem_t *VXlist_im, *VXlist_gt;        /* VisX data structure          */
VisXparam_t par[] =        /* command line structure        */
 {
   "im_filepath=",    0,   /* images file path       */
   "gt_filepath=",    0,   /* groundtruth file path      */
   "-v",     0,   /* visible flag     */
   0,      0    /* list termination */
 };

/*****************************************************************/
/* #defines                                                      */
/*****************************************************************/
#define  IMVAL    par[0].val
#define  GTVAL    par[1].val
#define  VFLAG    par[2].val
#define  WM  254
#define  GM  192
#define  CSF 128
#define  OTHER 0
#define  WM_ID  0
#define  GM_ID  1
```

```
#define  CSF_ID 2

main(argc, argv)
    int argc;
    char *argv[];
{

  VisX3dim_t im;                  /* input image structure       */
  VisX3dim_t gt_im;                   /* input segmentedground truth image structure       */
  VisXelem_t *vptr_im, *vptr_gt;
  char *im_filename, *gt_filename;
  char im_name[25];
  FILE *f_training, *f_gmm;//, *f_out;
  int i,j,k;                /* index counters             */
  int im_id, gt_id, num_images;
  double sum_wm=0;
  double sum_wm2=0;
  double num_wm=0;
  double sum_gm=0;
  double sum_gm2=0;
  double num_gm=0;
  double sum_csf=0;
  double sum_csf2=0;
  double num_csf=0;
  double sum_other=0;
  double sum_other2=0;
  double num_other=0;
  double mean_wm, var_wm, mean_gm, var_gm, mean_csf, var_csf, mean_other, var_other;
  unsigned char t;
  Vparse(&argc, &argv, par);     /* parse the command line       */

  /* Loop over all the training image filenames */
  f_training  = fopen("training_filenames.txt", "r");      /* open the output file */
  fscanf(f_training, "%d\n", &num_images);
  for (im_id = 0; im_id < num_images; im_id++)
   {
     fscanf(f_training, "%s\n", &im_name);
     im_filename = (char *)calloc(strlen(IMVAL) + strlen(im_name) + 1,sizeof(char));
     gt_filename = (char *)calloc(strlen(GTVAL) + strlen(im_name) + 1,sizeof(char));
     strcat(im_filename, IMVAL);
     strcat(im_filename, im_name);
     strcat(gt_filename, GTVAL);
     strcat(gt_filename, im_name);
     printf("%s\n", im_filename);
```

```
VXim   = VXopen(im_filename, 0);      /* open image file          */
VXgt   = VXopen(gt_filename, 0);      /* open ground truth file       */

VXlist_gt = VXread(VXgt);
if(VXNIL == (vptr_gt = VXfind(VXlist_gt, VX_PBYTE))){
      fprintf(stderr, "no ground truth images found\n");
      exit(1);
}

VXlist_im = VXread(VXim);        /* read file                */
if(VFLAG!=NULL){
      if(VXNIL == (vptr_im = VXfind(VXlist_im, VX_PSHORT))){
        fprintf(stderr, "no images found\n");
        exit(1);
      }
}
else if (VFLAG==NULL){
      if(VXNIL == (vptr_im = VXfind(VXlist_im, VX_PBYTE))){
        fprintf(stderr, "no images found\n");
        exit(1);
      }
}

VXset3dim(&im, vptr_im, VXim); /* initialize input structure   */
VXset3dim(&gt_im, vptr_gt, VXgt); /* initialize input structure   */

for (k = gt_im.zlo; k <= gt_im.zhi; k++){
      for (i = gt_im.ylo; i <=  gt_im.yhi; i++){
        for (j = gt_im.xlo; j <= gt_im.xhi; j++){
          if(VFLAG!=NULL){
            if(gt_im.u[k][i][j]==WM){
                  sum_wm += im.s[k+1][i][j];
                  sum_wm2 += pow(im.s[k+1][i][j],2);
                  num_wm++;
            }
            if(gt_im.u[k][i][j]==GM){
                  sum_gm += im.s[k+1][i][j];
                  sum_gm2 += pow(im.s[k+1][i][j],2);
                  num_gm++;
            }
            if(gt_im.u[k][i][j]==CSF){
                  sum_csf += im.s[k+1][i][j];
                  sum_csf2 += pow(im.s[k+1][i][j],2);
                  num_csf++;
```

```
            }
            if(gt_im.u[k][i][j]==OTHER){
                sum_other += im.s[k+1][i][j];
                sum_other2 += pow(im.s[k+1][i][j],2);
                num_other++;
            }
           }
           else if(VFLAG==NULL){
            if(gt_im.u[k][i][j]==WM){
                sum_wm += im.u[k][i][j];
                sum_wm2 += pow(im.u[k][i][j],2);
                num_wm++;
            }
            if(gt_im.u[k][i][j]==GM){
                sum_gm += im.u[k][i][j];
                sum_gm2 += pow(im.u[k][i][j],2);
                num_gm++;
            }
            if(gt_im.u[k][i][j]==CSF){
                sum_csf += im.u[k][i][j];
                sum_csf2 += pow(im.u[k][i][j],2);
                num_csf++;
            }
            if(gt_im.u[k][i][j]==OTHER){
                sum_other += im.u[k][i][j];
                sum_other2 += pow(im.u[k][i][j],2);
                num_other++;
            }
           }
          }
         }
    }

    VXclose(VXim);              /* close files              */
    VXclose(VXgt);
  }
  fclose(f_training);

/* Compute mean and variance for the three classes */
mean_wm = sum_wm / num_wm;
var_wm  = (sum_wm2 / num_wm) - pow(mean_wm,2);
mean_gm = sum_gm / num_gm;
var_gm  = (sum_gm2 / num_gm) - pow(mean_gm,2) ;
mean_csf = sum_csf / num_csf;
```

```
 var_csf  = (sum_csf2 / num_csf) - pow(mean_csf,2);
 mean_other = sum_other / num_other;
 var_other  = (sum_other2 / num_other) - pow(mean_other,2);

 f_gmm = fopen("gmm_parameters.txt", "w");
 fprintf(f_gmm, "%f %f\n%f %f \n%f %f \n%f %f\n", mean_wm, var_wm, mean_gm, var_gm,
mean_csf, var_csf, mean_other, var_other);
 fprintf(f_gmm,"%f %f %f %f\n%f %f %f %f\n%f %f %f %f\n", sum_wm, sum_gm, sum_csf,
sum_other, sum_wm2, sum_gm2, sum_csf2, sum_other2 , num_wm, num_gm, num_csf,
num_other);
 fclose(f_gmm);
 exit(0);
}
```

Main File

```
/*****************************************************************/
/* Example VisX4 program main()                                  */
/*          Main function for the brain MRI EM algorithm         */
/* Syntax:                                                       */
/*                    im_filepath   = images path                */
/*                    out_filepath  = output path                */
/*                    gt_filepath   = for accuracy               */
/*                    [-v]                                       */
/*****************************************************************/
/*****************************************************************/
/*  This Software is Copyright (c) 1996, 2000 Anthony P. Reeves  */
/*  All rights reserved.                                         */
/*****************************************************************/

#include "VisXV4.h"        /* VisX structure include file     */
#include "Vutil.h"         /* VisX utility header files       */
#include "stdio.h"
#include "math.h"

VisXfile_t *VXim,          /* image file structure     */
 *VXout,         /* ouput file structure     */
 *VXgt;          /* ground truth             */
VisXelem_t *VXlist, *VXlist_gt; /* VisX data structure   */
VisXparam_t par[] =        /* command line structure   */
 {
  "im_filepath=",    0,   /* images file path      */
  "out_filepath=",   0,   /* output file path      */
  "gt_filepath=",    0,   /* output file path      */
```

```
  "-v",    0,  /* visible flag    */
  0,       0   /* list termination */
 };

/*****************************************************************/
/* #defines                                                      */
/*****************************************************************/
#define  IMVAL    par[0].val
#define  OUTVAL   par[1].val
#define  GTVAL    par[2].val
#define  VFLAG    par[3].val
#define  WM  254
#define  GM  192
#define  CSF 128
#define  OTHER 0
#define  WM_ID  0
#define  GM_ID  1
#define  CSF_ID 2
#define  PI 3.14159265358
#define  MAX_ITR 10
#define  CONFVAL "confusion_matrix.txt"

/*****************************************************************/
/* function declarations                                         */
/*****************************************************************/
void gmm_classify( VisX3dim_t *im, VisX3dim_t gt, int frame_offset);

int max_value(float *p_array,
              int values_in_array);

void update_gmm_parameters(VisX3dim_t seg_im, VisX3dim_t im);

void update_confusion_mat(VisX3dim_t im, VisX3dim_t gt_im, int frame_offset);


/*****************************************************************/
/* main function                                                 */
/*****************************************************************/
main(argc, argv)
   int argc;
   char *argv[];
{

 VisX3dim_t im;            /* input image structure              */
```

```
  VisX3dim_t gt_im;           /* input segmented ground truth image structure */
  VisX3dim_t tm;
  VisXelem_t *vptr, *vptr_gt;
  char *im_filename, *out_filename, *gt_filename;
  char im_name[25];
  FILE *f_testing, *f_gmm, *f_conf_mat;
  int i,j,k;                /* index counters             */
  int im_id, max_id, num_images, itr_id, frame_offset, frame_offset_gt;
  double conf_mat[3][3] = {{0, 0, 0},
                           {0, 0, 0},
                           {0, 0, 0}};

  unsigned char t;
  Vparse(&argc, &argv, par);    /* parse the command line      */


  /* Create the confusion matrix file */
  f_conf_mat  = fopen(CONFVAL, "w");     /* open the output file */
  for (i = 0; i < 3; i++){
    fprintf(f_conf_mat, "%lf %lf %lf\n", conf_mat[i][0], conf_mat[i][1], conf_mat[i][2],
conf_mat[i][3]);
  }
  fclose(f_conf_mat);

  /* Loop over all the training image filenames */
  f_testing  = fopen("testing_filenames.txt", "r");     /* open the output file */
  fscanf(f_testing, "%d\n", &num_images);
  for (im_id = 0; im_id < num_images; im_id++)
    {
     fscanf(f_testing, "%s %d\n", &im_name, &frame_offset);
     im_filename  = (char *)calloc(strlen(IMVAL) + strlen(im_name) + 1,sizeof(char));
     out_filename = (char *)calloc(strlen(OUTVAL) + strlen(im_name) + 1,sizeof(char));
     gt_filename  = (char *)calloc(strlen(GTVAL) + strlen(im_name) + 1,sizeof(char));
     strcat(im_filename, IMVAL);
     strcat(im_filename, im_name);
     strcat(out_filename, OUTVAL);
     strcat(out_filename, im_name);
     strcat(gt_filename, GTVAL);
     strcat(gt_filename, im_name);
     VXim   = VXopen(im_filename, 0);      /* open image file         */
     VXout  = VXopen(out_filename, 1);     /* open output file        */
     VXgt   = VXopen(gt_filename, 0);      /* open ground truth file   */

     VXlist = VXread(VXim);        /* read file                */
```

```
    if(VFLAG!=NULL){
        if(VXNIL == (vptr = VXfind(VXlist, VX_PSHORT))){
          fprintf(stderr, "no images found\n");
          exit(1);
        }
    }
    else if (VFLAG==NULL){
        if(VXNIL == (vptr = VXfind(VXlist, VX_PBYTE))){
          fprintf(stderr, "no images found\n");
          exit(1);
        }
    }

    VXlist_gt = VXread(VXgt);
    if(VXNIL == (vptr_gt = VXfind(VXlist_gt, VX_PBYTE))){
        fprintf(stderr, "no ground truth images found\n");
        exit(1);
    }

    VXset3dim(&im, vptr, VXim); /* initialize input structure   */
    VXset3dim(&gt_im, vptr_gt, VXgt); /* initialize input structure   */

    /* Make a copy of the input image */
    VXembed3dim(&tm, &im, 0, 0, 0, 0, 0, 0); /* embed input structure   */

    for(itr_id=0 ; itr_id<MAX_ITR ; itr_id++)
        {
          printf("Interation number: %d\n", (itr_id+1));

          /* Call the classify function */
          printf("Classifying...\n");
          gmm_classify(&im, gt_im, frame_offset);

          /* Update the GMM parameters */
          printf("Updating GMM parameters...\n");
          update_gmm_parameters(im, tm);

          if(itr_id != (MAX_ITR-1)){
            VXembed3dim(&im, &tm, 0, 0, 0, 0, 0, 0); /* embed input structure   */
          }
        }

    /* After classification update the confusion matrix and finally compute the accuracy metric
*/
```

```
    printf("Updating confusion matrix...\n");
    VXwrite(VXout, VXlist);      /* write data               */
    update_confusion_mat(im, gt_im, frame_offset);
    printf("Done!!\n");

    VXclose(VXim);             /* close files              */
    VXclose(VXout);
   }
 fclose(f_testing);
 exit(0);
}



/* Find maximum value in vector */
int max_value(float *p_array,
              int num_array)
{
 int position, max_position=0;
 float max_value;

 position = 0;
 max_value = p_array[position];
 for (position = 1; position < num_array; position++)
  {
   if (p_array[position] > max_value)
        {
          max_value = p_array[position];
      max_position = position;
        }
  }
 return max_position;
}



/* Classify using GMMs */
void gmm_classify( VisX3dim_t *im, VisX3dim_t gt, int frame_offset)
{
 VisX3dim_t tm;
 double sum_wm=0;
 double sum_wm2=0;
 double num_wm=0;
 double sum_gm=0;
```

```
 double sum_gm2=0;
 double num_gm=0;
 double sum_csf=0;
 double sum_csf2=0;
 double num_csf=0;
 float mean_wm, var_wm, mean_gm, var_gm, mean_csf, var_csf;
 FILE *f_gmm;
 float wm_likelihood, gm_likelihood, csf_likelihood;
 float wm_prior, gm_prior, csf_prior;
 float wm_posterior, gm_posterior, csf_posterior;
 float posterior[3];
 int i,j,k;              /* index counters          */
 int im_id, max_id, num_images, temp_flag=1;

 f_gmm = fopen("gmm_parameters_EM.txt", "r");
 fscanf(f_gmm, "%f %f\n%f %f \n%f %f \n", &mean_wm, &var_wm, &mean_gm, &var_gm,
&mean_csf, &var_csf);
 fscanf(f_gmm,"%lf %lf %lf\n%lf %lf %lf\n%lf %lf %lf\n", &sum_wm, &sum_gm, &sum_csf,
&sum_wm2, &sum_gm2, &sum_csf2 , &num_wm, &num_gm, &num_csf);
 fclose(f_gmm);
 printf("%f %f\n%f %f \n%f %f \n", mean_wm, var_wm, mean_gm, var_gm, mean_csf, var_csf);

 VXembed3dim(&tm, im, 0, 0, 0, 0, 0, 0); /* Copy input image */

 /* Priors */
 wm_prior = num_wm / (num_wm + num_gm + num_csf);
 gm_prior = num_gm / (num_wm + num_gm + num_csf);
 csf_prior = num_csf / (num_wm + num_gm + num_csf);

 for (k = gt.zlo; k <= gt.zhi; k++){
  for (i = gt.ylo; i <=  gt.yhi; i++){
   for (j = gt.xlo; j <= gt.xhi; j++){
        temp_flag=1;
        if(VFLAG!=NULL){
         if(gt.u[k][i][j]==0){
          temp_flag=0;
          im->s[k+frame_offset][i][j]=OTHER;
          continue;
         }
         else
          {
           temp_flag=1;
           /* Evaluate the likelihoods */
```

```
            wm_likelihood    = (1 / (sqrt(2 * PI * var_wm))) * exp(-pow((im-
>s[k+frame_offset][i][j] - mean_wm),2) / (2 * var_wm));
            gm_likelihood    = (1 / (sqrt(2 * PI * var_gm))) * exp(-pow((im->s[k+frame_offset][i][j]
- mean_gm),2) / (2 * var_gm));
            csf_likelihood   = (1 / (sqrt(2 * PI * var_csf))) * exp(-pow((im->s[k+frame_offset][i][j] -
mean_csf),2) / (2 * var_csf));
             }
          }
          else if(VFLAG==NULL){
           if(gt.u[k][i][j]==0){
            temp_flag=0;
            im->u[k+frame_offset][i][j]=OTHER;
            continue;
           }
           else
            {
             temp_flag=1;
             /* Evaluate the likelihoods */
             wm_likelihood    = (1 / (sqrt(2 * PI * var_wm))) * exp(-pow((im-
>u[k+frame_offset][i][j] - mean_wm),2) / (2 * var_wm));
             gm_likelihood    = (1 / (sqrt(2 * PI * var_gm))) * exp(-pow((im-
>u[k+frame_offset][i][j] - mean_gm),2) / (2 * var_gm));
             csf_likelihood   = (1 / (sqrt(2 * PI * var_csf))) * exp(-pow((im->u[k+frame_offset][i][j]
- mean_csf),2) / (2 * var_csf));
             }
          }
          if(temp_flag==1){
           /* Evaluate the posteriors */
           wm_posterior    = (wm_prior*wm_likelihood) /
((wm_prior*wm_likelihood)+(gm_prior*gm_likelihood)+(csf_prior*csf_likelihood));
           gm_posterior    = (gm_prior*gm_likelihood) /
((wm_prior*wm_likelihood)+(gm_prior*gm_likelihood)+(csf_prior*csf_likelihood));
           csf_posterior   = (csf_prior*csf_likelihood) /
((wm_prior*wm_likelihood)+(gm_prior*gm_likelihood)+(csf_prior*csf_likelihood));

           /* Put posteriors in array and find max */
           posterior[0] = wm_posterior;
           posterior[1] = gm_posterior;
           posterior[2] = csf_posterior;

           max_id = max_value(posterior, 3);
           switch(max_id){
           case WM_ID:if(VFLAG!=NULL){
              im->s[k+frame_offset][i][j] = WM;
```

```
              break;
             }
             else if(VFLAG==NULL){
              im->u[k+frame_offset][i][j] = WM;
              break;
             }
            case GM_ID:if(VFLAG!=NULL){
              im->s[k+frame_offset][i][j] = GM;
              break;
             }
             else if(VFLAG==NULL){
              im->u[k+frame_offset][i][j] = GM;
              break;
             }
            case CSF_ID:if(VFLAG!=NULL){
              im->s[k+frame_offset][i][j] = CSF;
              break;
             }
             else if(VFLAG==NULL){
              im->u[k+frame_offset][i][j] = CSF;
              break;
             }
            default:printf("default\n");
             break;
            }
           }
     }
    }
  }
  VXembed3dim(&tm, im, 0, 0, 0, 0, 0, 0); /* Copy input image */
}


/* Update the GMM parameters */
void update_gmm_parameters( VisX3dim_t seg_im, VisX3dim_t im)
{
  double sum_wm=0;
  double sum_wm2=0;
  double num_wm=0;
  double sum_gm=0;
  double sum_gm2=0;
  double num_gm=0;
  double sum_csf=0;
  double sum_csf2=0;
```

```
 double num_csf=0;
 double sum_other=0;
 double sum_other2=0;
 double num_other=0;
 float  mean_wm, var_wm, mean_gm, var_gm, mean_csf, var_csf;
 FILE *f_gmm;
 int i,j,k;              /* index counters          */
 int im_id, max_id, num_images;

 f_gmm = fopen("gmm_parameters_EM.txt", "r");
 fscanf(f_gmm, "%f %f\n%f %f \n%f %f \n", &mean_wm, &var_wm, &mean_gm, &var_gm,
&mean_csf, &var_csf);
 fscanf(f_gmm,"%lf %lf %lf\n%lf %lf %lf\n%lf %lf %lf\n", &sum_wm, &sum_gm, &sum_csf,
&sum_wm2, &sum_gm2, &sum_csf2, &num_wm, &num_gm, &num_csf);
 fclose(f_gmm);

 /* Update GMM parameters */
 for (k = im.zlo; k <= im.zhi; k++){
  for (i = im.ylo; i <=  im.yhi; i++){
   for (j = im.xlo; j <= im.xhi; j++){
        if(VFLAG!=NULL){
         if(seg_im.s[k][i][j]==WM){
          sum_wm += im.s[k][i][j];
          sum_wm2 += pow(im.s[k][i][j],2);
          num_wm++;
         }
         if(seg_im.s[k][i][j]==GM){
          sum_gm += im.s[k][i][j];
          sum_gm2 += pow(im.s[k][i][j],2);
          num_gm++;
         }
         if(seg_im.s[k][i][j]==CSF){
          sum_csf += im.s[k][i][j];
          sum_csf2 += pow(im.s[k][i][j],2);
          num_csf++;
         }
        }
        else if(VFLAG==NULL){
         if(seg_im.u[k][i][j]==WM){
          sum_wm += im.u[k][i][j];
          sum_wm2 += pow(im.u[k][i][j],2);
          num_wm++;
         }
         if(seg_im.u[k][i][j]==GM){
```

```
            sum_gm += im.u[k][i][j];
            sum_gm2 += pow(im.u[k][i][j],2);
            num_gm++;
           }
           if(seg_im.u[k][i][j]==CSF){
            sum_csf += im.u[k][i][j];
            sum_csf2 += pow(im.u[k][i][j],2);
            num_csf++;
           }
          }
    }
   }
  }

  /* Compute mean and variance for the three classes */
  mean_wm = sum_wm / num_wm;
  var_wm  = (sum_wm2 / num_wm) - pow(mean_wm,2);
  mean_gm = sum_gm / num_gm;
  var_gm  = (sum_gm2 / num_gm) - pow(mean_gm,2) ;
  mean_csf = sum_csf / num_csf;
  var_csf  = (sum_csf2 / num_csf) - pow(mean_csf,2);

  f_gmm = fopen("gmm_parameters_EM.txt", "w");
  fprintf(f_gmm, "%f %f\n%f %f \n%f %f \n", mean_wm, var_wm, mean_gm, var_gm, mean_csf,
var_csf);
  fprintf(f_gmm,"%lf %lf %lf\n%lf %lf %lf\n%lf %lf %lf\n", sum_wm, sum_gm, sum_csf,
sum_wm2, sum_gm2, sum_csf2 , num_wm, num_gm, num_csf);
  fclose(f_gmm);
}


/* Update the confusion matrix */
void update_confusion_mat( VisX3dim_t seg_im,  VisX3dim_t gt_im, int frame_offset)
{
  int seg_id, gt_id, i, j, k;
  double conf_mat[3][3];
  double wm_pc, gm_pc, csf_pc;
  FILE* f_conf_mat;

  /* Obtain the previously saved confusion matrix */
  f_conf_mat  = fopen(CONFVAL, "r");      /* open the output file */
  for (i = 0; i < 3; i++){
   fscanf(f_conf_mat, "%lf %lf %lf\n", &conf_mat[i][0], &conf_mat[i][1], &conf_mat[i][2]);
  }
```

```
fclose(f_conf_mat);

/* Evalute the confusion matrix for this image */
for (k = gt_im.zlo; k <= gt_im.zhi; k++){
  for (i = gt_im.ylo; i <= gt_im.yhi; i++){
    for (j = gt_im.xlo; j <= gt_im.xhi; j++){
          /* Determine the segmented pixel id */
          if(VFLAG!=NULL){
            switch(seg_im.s[k+frame_offset][i][j]){
            case WM : seg_id = WM_ID;
              break;
            case GM : seg_id = GM_ID;
              break;
            case CSF: seg_id = CSF_ID;
              break;
            default: seg_id = 4;
              break;
            }
          }
          else if(VFLAG==NULL){
            switch(seg_im.u[k+frame_offset][i][j]){
            case WM : seg_id = WM_ID;
              break;
            case GM : seg_id = GM_ID;
              break;
            case CSF: seg_id = CSF_ID;
              break;
            default: seg_id = 4;
              break;
            }
          }

          /* Determine the ground truth pixel id */
          switch(gt_im.u[k][i][j]){
          case WM : gt_id = WM_ID;
            break;
          case GM : gt_id = GM_ID;
            break;
          case CSF: gt_id = CSF_ID;
            break;
          default: gt_id = 4;
            break;
          }
          if((seg_id!=4) && (gt_id!=4))
```

```
            conf_mat[gt_id][seg_id]++;
    }
   }
  }

  /* Save the new updated confusion matrix */
  f_conf_mat  = fopen(CONFVAL, "w");      /* open the output file */
  for (i = 0; i < 3; i++){
   fprintf(f_conf_mat, "%lf %lf %lf\n", conf_mat[i][0], conf_mat[i][1], conf_mat[i][2]);
  }
  wm_pc = ((100*conf_mat[0][0])/( conf_mat[0][0]+ conf_mat[0][1]+ conf_mat[0][2]));
  gm_pc = ((100*conf_mat[1][1])/( conf_mat[1][0]+ conf_mat[1][1]+ conf_mat[1][2]));
  csf_pc = ((100*conf_mat[2][2])/( conf_mat[2][0]+ conf_mat[2][1]+ conf_mat[2][2]));
  fprintf(f_conf_mat, "%lf %lf %lf\n", wm_pc, gm_pc, csf_pc);
  fclose(f_conf_mat);

}
```

The LPF File. This is used as the baseline.

```
/*****************************************************************/
/* Example VisX4 program lpf.c                                   */
/*          Baseline for brain MRI classification                */
/* Syntax:                                                       */
/*                    im_filepath   = images path                */
/*                    out_filepath  = output path                */
/*                    gt_filepath   = for accuracy               */
/*                    [-v]                                       */
/*****************************************************************/
/*****************************************************************/
/*  This Software is Copyright (c) 1996, 2000 Anthony P. Reeves  */
/*  All rights reserved.                                         */
/*****************************************************************/

#include "VisXV4.h"        /* VisX structure include file     */
#include "Vutil.h"         /* VisX utility header files       */
#include "stdio.h"
#include "math.h"

VisXfile_t *VXim,          /* image file structure     */
        *VXout,         /* ouput file structure     */
        *VXgt;          /* ground truth             */
VisXelem_t *VXlist, *VXlist_gt; /* VisX data structure   */
VisXparam_t par[] =        /* command line structure   */
 {
```

```
  "im_filepath=",    0,   /* images file path      */
  "out_filepath=",   0,   /* output file path      */
  "gt_filepath=",    0,   /* output file path      */
  "-v",     0,   /* visible flag     */
  0,        0    /* list termination */
 };

/*****************************************************************/
/* #defines                                                      */
/*****************************************************************/
#define  IMVAL    par[0].val
#define  OUTVAL   par[1].val
#define  GTVAL    par[2].val
#define  VFLAG    par[3].val
#define  WM_ID  0
#define  GM_ID  1
#define  CSF_ID 2
#define WM 254
#define GM 192
#define CSF 128
#define OTHER 0
#define CONFVAL "confusion_matrix.txt"
#define AVG 0.037037037

void update_confusion_mat(VisX3dim_t im, VisX3dim_t gt_im, int frame_offset);

main(argc, argv)
    int argc;
    char *argv[];
{
 VisX3dim_t im;              /* input image structure                    */
 VisX3dim_t gt_im;           /* input segmented ground truth image structure */
 VisX3dim_t tm;
 VisXelem_t *vptr, *vptr_gt;
 char *im_filename, *out_filename, *gt_filename;
 char im_name[25];
 FILE *f_testing, *f_gmm, *f_conf_mat;
 int        i,j,k,x,y,z,num_images;              /* index counters            */
 int im_id, frame_offset;
 unsigned char t;
 int tempsum = 0;
 double sum_wm=0;
 double sum_wm2=0;
 double num_wm=0;
```

```
 double sum_gm=0;
 double sum_gm2=0;
 double num_gm=0;
 double sum_csf=0;
 double sum_csf2=0;
 double num_csf=0;
 double sum_other=0;
 double sum_other2=0;
 double num_other=0;
 float mean_wm, var_wm, mean_gm, var_gm, mean_csf, var_csf, mean_other, var_other;
 float WMthresh, GMthresh, CSFthresh, Othresh;
 double conf_mat[3][4] = {{0, 0, 0},
                          {0, 0, 0},
                          {0, 0, 0}};

  Vparse(&argc, &argv, par);     /* parse the command line       */

/* Create the confusion matrix file */
 f_conf_mat  = fopen(CONFVAL, "w");      /* open the output file */
 for (i = 0; i < 3; i++){
   fprintf(f_conf_mat, "%lf %lf %lf\n", conf_mat[i][0], conf_mat[i][1], conf_mat[i][2]);
 }
 fclose(f_conf_mat);

 f_gmm = fopen("gmm_parameters.txt", "r");
 fscanf(f_gmm, "%f %f\n%f %f \n%f %f \n", &mean_wm, &var_wm, &mean_gm, &var_gm,
&mean_csf, &var_csf);
 fclose(f_gmm);


 WMthresh = (mean_wm+mean_gm)/2;
 GMthresh = (mean_gm+mean_csf)/2;
 CSFthresh = (mean_csf+0)/2;


 f_testing  = fopen("testing_filenames.txt", "r");      /* open the output file */
 fscanf(f_testing, "%d\n", &num_images);
 for (im_id = 0; im_id < num_images; im_id++)
  {
    fscanf(f_testing, "%s %d\n", &im_name, &frame_offset);
    im_filename  = (char *)calloc(strlen(IMVAL) + strlen(im_name) + 1,sizeof(char));
    out_filename = (char *)calloc(strlen(OUTVAL) + strlen(im_name) + 1,sizeof(char));
    gt_filename  = (char *)calloc(strlen(GTVAL) + strlen(im_name) + 1,sizeof(char));
    strcat(im_filename, IMVAL);
```

```
strcat(im_filename, im_name);
strcat(out_filename, OUTVAL);
strcat(out_filename, im_name);
strcat(gt_filename, GTVAL);
strcat(gt_filename, im_name);

VXim   = VXopen(im_filename, 0);      /* open image file         */
VXout  = VXopen(out_filename, 1);     /* open output file        */
VXgt   = VXopen(gt_filename, 0);      /* open ground truth file   */


VXlist = VXread(VXim);        /* read file                */
if(VFLAG!=NULL){
      if(VXNIL == (vptr = VXfind(VXlist, VX_PSHORT))){
        fprintf(stderr, "no images found\n");
        exit(1);
      }
}
else if (VFLAG==NULL){
      if(VXNIL == (vptr = VXfind(VXlist, VX_PBYTE))){
        fprintf(stderr, "no images found\n");
        exit(1);
      }
}

VXlist_gt = VXread(VXgt);
if(VXNIL == (vptr_gt = VXfind(VXlist_gt, VX_PBYTE))){
      fprintf(stderr, "no ground truth images found\n");
      exit(1);
}

VXset3dim(&im, vptr, VXim); /* initialize input structure   */
VXset3dim(&gt_im, vptr_gt, VXgt); /* initialize input structure   */

VXembed3dim(&tm, &im, 1, 1, 1, 1, 1, 1); /* embed input structure   */
if(VFLAG){
      fprintf(stderr,"bbx is %f %f %f %f %f %f\n", im.bbx[0],
              im.bbx[1],im.bbx[2],im.bbx[3],im.bbx[4],im.bbx[5]);
}

/* simple pixel computation -- notice the order of the loops */
for (k = im.zlo; k <= im.zhi; k++){
      for (i = im.ylo; i <= im.yhi; i++){
        for (j = im.xlo; j <= im.xhi; j++){
```

```
            for (z = (k-1); z <= (k+1); z++){
              for (y = (i-1); y <= (i+1); y++){
                    for (x = (j-1); x <= (j+1); x++){
                      tempsum += (AVG*tm.u[z][y][x]);
                    }
              }
            }
            im.u[k][i][j] = tempsum;
            tempsum = 0;

          }
        }
    }
    for (k = im.zlo; k <= im.zlo+frame_offset; k++){
        for (i = im.ylo; i <= im.yhi; i++){
          for (j = im.xlo; j <= im.xhi; j++){
            im.u[k][i][j] = 0;
          }
        }
    }

    for (k = gt_im.zlo; k <= gt_im.zhi; k++){
        for (i = gt_im.ylo; i <= gt_im.yhi; i++){
          for (j = gt_im.xlo; j <= gt_im.xhi; j++){
            if(gt_im.u[k][j][i] == 0){
              im.u[k+frame_offset][j][i] = OTHER;
            }
            else
             {
                  if (tm.u[k+frame_offset][j][i] >= WMthresh)
                    im.u[k+frame_offset][j][i] = WM;
                  else if ((tm.u[k+frame_offset][j][i] < WMthresh) && (tm.u[k+frame_offset][j][i]
>= GMthresh))
                    im.u[k+frame_offset][j][i] = GM;
                  else if ((tm.u[k+frame_offset][j][i] < GMthresh))// &&
(tm.u[k+frame_offset][j][i]>= CSFthresh))
                    im.u[k+frame_offset][j][i] = CSF;
             }
          }
        }
    }
```

```
    /* After classification update the confusion matrix and finally compute the accuracy metric
*/
    printf("Updating confusion matrix...\n");
    update_confusion_mat(im, gt_im, frame_offset);

    VXwrite(VXout, VXlist);      /* write data                */
    VXclose(VXim);              /* close files               */
    VXclose(VXout);
   }
 fclose(f_testing);
 exit(0);
}



/* Update the confusion matrix */
void update_confusion_mat( VisX3dim_t seg_im,  VisX3dim_t gt_im, int frame_offset)
{
 int seg_id, gt_id, i, j, k;
 double conf_mat[3][3];
 double wm_pc, gm_pc, csf_pc;
 FILE* f_conf_mat;

 /* Obtain the previously saved confusion matrix */
 f_conf_mat  = fopen(CONFVAL, "r");     /* open the output file */
 for (i = 0; i < 3; i++){
  fscanf(f_conf_mat, "%lf %lf %lf\n", &conf_mat[i][0], &conf_mat[i][1], &conf_mat[i][2]);
 }
 fclose(f_conf_mat);

 /* Evalute the confusion matrix for this image */
 for (k = gt_im.zlo; k <= gt_im.zhi; k++){
  for (i = gt_im.ylo; i <= gt_im.yhi; i++){
   for (j = gt_im.xlo; j <= gt_im.xhi; j++){

        /* Determine the segmented pixel id */
        switch(seg_im.u[k+frame_offset][i][j]){
        case WM : seg_id = WM_ID;
                break;
        case GM : seg_id = GM_ID;
                break;
        case CSF: seg_id = CSF_ID;
                break;
        default: seg_id = 4;
```

```
          break;
         }

         /* Determine the ground truth pixel id */
         switch(gt_im.u[k][i][j]){
         case WM : gt_id = WM_ID;
                 break;
         case GM : gt_id = GM_ID;
                 break;
         case CSF: gt_id = CSF_ID;
             break;
         default: gt_id = 4;
          break;
         }
         if((seg_id!=4) && (gt_id!=4))
          conf_mat[gt_id][seg_id]++;
    }
   }
  }

  /* Save the new updated confusion matrix */
  f_conf_mat  = fopen(CONFVAL, "w");      /* open the output file */
  for (i = 0; i < 3; i++){
   fprintf(f_conf_mat, "%lf %lf %lf\n", conf_mat[i][0], conf_mat[i][1], conf_mat[i][2]);
  }
  wm_pc = ((100*conf_mat[0][0])/( conf_mat[0][0]+ conf_mat[0][1]+ conf_mat[0][2]));
  gm_pc = ((100*conf_mat[1][1])/( conf_mat[1][0]+ conf_mat[1][1]+ conf_mat[1][2]));
  csf_pc = ((100*conf_mat[2][2])/( conf_mat[2][0]+ conf_mat[2][1]+ conf_mat[2][2]));
  fprintf(f_conf_mat, "%lf %lf %lf\n", wm_pc, gm_pc, csf_pc);
  fclose(f_conf_mat);

}
```