CS5680 Final Project- Fall 2021

Detection of Brain Tumors in MRI Images, Using Combination of Fuzzy C-Means and SVM

Joshua Brock

1 Steps to run code

First, check if there is a directory titled “tmp” in the directory contain main.m. If it doesn’t exist, create it now, and add two subdirectories: “yes” and “no”. To run the code with default settings, simply run the command “main()” in the Matlab console while in the appropriate directory. The descriptions for all input parameters are found in the header of main.m. It is recommended that the user run the program with the default values. If the user must see intermediate plots or change the kernel function, they may do so, but the numTraining and resizeVal arguments are included for developer testing purposes only.

In summary, the user should run “main()” in the Matlab console to run with default values or “main(kernelFn, bPlot)” to change the kernel function or plot results.

2 Approximate execution time

The execution time depends on the kernel function used to train the support vector machine. On average, expect approximately 5 minutes for loading, preprocessing, and saving the image to the temp directory, 20 minutes for feature extraction, between 1 and 4 hours for training the SVM, and 5 minutes for classification testing. In short, fully executing will take between 1.5 and 4.5 hours depending on the kernel function. The order from fastest to slowest is gaussian, linear, and rbf.

3 Summary of important code sections and changes/improvements from the paper

3.1 Preprocessing

3.1.1 Image Enhancement

This code is found in enhance.m. It follows the approach found in the paper, but I decided to use Matlab’s “imenhance” function due to the images in the data set not being consistently pre-enhanced. Matlab’s built-in function works well enough for the purposes of this project. For testing this section, I plotted the histograms of all the enhanced images and verified they all had similar distributions.

3.1.2 Skull Stripping

This code is in skullStrip.m. I started with the approach found in the paper which was to double threshold and then do an open-close operation with a disk structuring element of radius 3. I opted to change this because it did not work for the arbitrarily sized data I had in my data set. I followed roughly the same approach, but my improvement was to change the size of the structuring element to be proportional to the image size. More specifically, I make the structuring element radius equal to the image size divided by 30. This works in nearly all the images. I zero pad

the array too so I can do an aggressive close-open operation to get rid of any holes in the brain caused by thresholding. I then multiply the original image by the threshold image and output.

I verified this section was correct by looking at images as they came out of the function and verifying the skull was gone.

3.1.3 Segmentation

This code is in segmentation.m. In this, I follow the approach followed by the paper. I opted to use Matlab's fcm with 3 segments. This is identical to what they did in the paper and essentially gives 1 segment to the background and 2 segments to brain tissue. The distribution of the brain tissue segments is how the SVM determines if there is a tumor present in the brain. I tested this by visually comparing my results with those in the paper.

3.1.4 Feature Extraction

Feature extraction is done in main.m. Due to time, I decided to change the extraction method from gray level run length matrix to Matlab's "bagOfFeatures" built-in function. This seems to work well with the SVM.

3.1.5 Training the Support Vector Machine (SVM)

The classifier is trained in trainClassifier.m. I used Matlab's "trainImageCategoryClassifier" function to train the SVM. This function uses an SVM under the hood, so it does exactly what I want it to. The paper did not specify what kernel function they used, so I decided to make it up to the user. I have tested this with "linear," "rbf," and "gaussian" kernel functions. "rbf" seems to give the best results.

I test the image classifier using the Matlab "evaluate" function, which gives a table containing the percentage of true positive/negatives and false positive/negatives. I the calculate the accuracy, sensitivity, and specificity using the results from the paper:

Sensitivity = TP/(TP+FN)*100%	(measure of how accurate the positive test is)
Specificity = TN/(TN+FP)*100%	(measure of how accurate the negative test is)
Accuracy = (TP+TN)/(TP+TN+FP+FN)*100%	(measure of the overall accuracy)

Results were then compared to the ones in the paper to verify they were roughly the same.

4 Experimental results of the approach proposed in the paper

Refer to 3.1.5 for how each of these metrics are calculated.

Kernel Function	Accuracy	Sensitivity	Specificity
Linear	91.66%	83.33%	100%
Quadratic	83.33%	66.66%	100%
Polynomial	87.50%	75.00%	100%

5 My experimental results

Refer to 3.1.5 for how each of these metrics are calculated.

Kernel Function	Accuracy	Sensitivity	Specificity
Linear	69.00%	58.00%	75.00%
RBF	79.50%	75.00%	84.00%
Gaussian	58.00%	35.00%	81.00%

RBF seems to be the most accurate kernel function at 79.5%. Also of note is the sensitivity is almost on par with that from the paper, which is probably the most important part, because we'd rather have the positive test be accurate the negative test. This is because it's better for a doctor to think there may be a tumor and realize there is not than for the system to miss a tumor and doctor find out too late.

6 Comparing Results

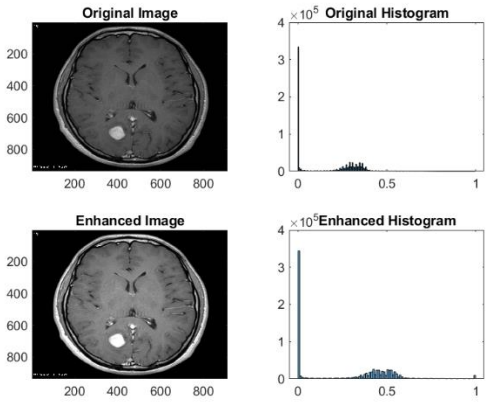
My results are not as accurate as the ones in the paper, though the RBF kernel function gets close. I think the main reason for my drop in accuracy is using a data set with inconsistencies in image histograms and size. The paper had the advantage of using a data set with no enhanced images and constant size, while I did not have that luxury. Recall that I use the Matlab imadjust function to enhance the images to achieve consistent results across all the images. Some of the accuracy issues may be attributed to image enhancement, but I think skull-stripping was the issue in most cases. My skull-stripping function works very well about 90% of the time, but the remaining 10% end up with undesirable artifacts in them, such as remaining "skull shards" or missing pieces of the brain. These likely messed up the SVM classifier as I didn't manually check and remove the bad images.

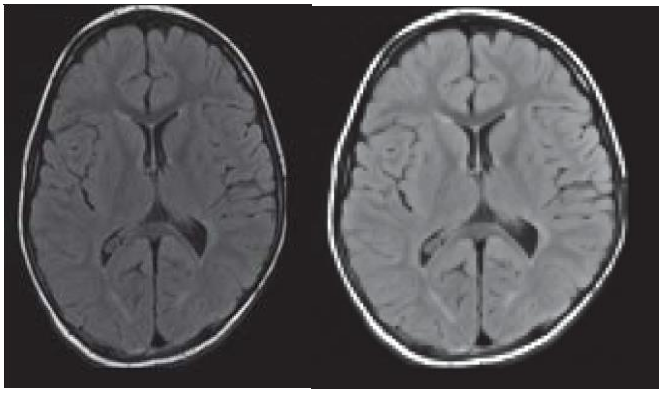
7 Analysis

As said above, my results are not as good as those from the paper. My method is less effective than the paper's for the data both of us used respectively. I reiterate that my results would likely be similar if I had test data that was consistently sized and enhanced. The performance is probably about as good as I can get it in the time frame. I use several Matlab build-in functions, which should, in theory, be optimized. I also try to use vectorized operations where possible. The things that take a lot of time is reading, writing, and training, which are all out of my control. I'm confident my algorithm produces correct results because the preprocessing results are the same as those in the paper.

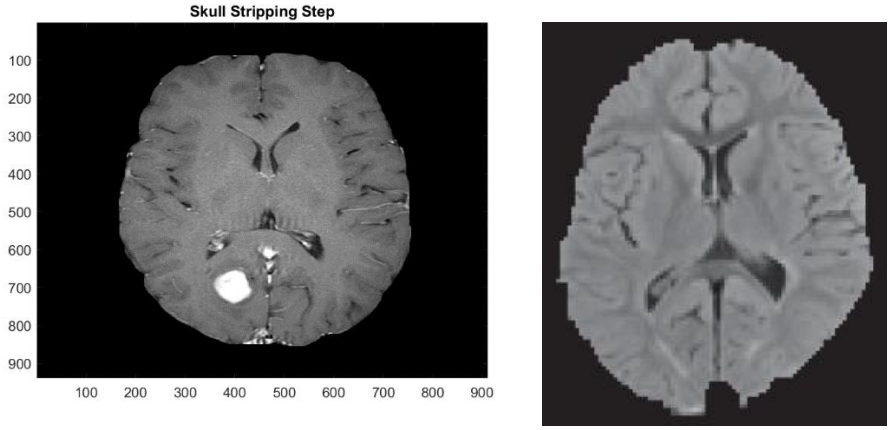
8 Images generated at each step

8.1 Enhancement (my implementation on the left, paper on the right)

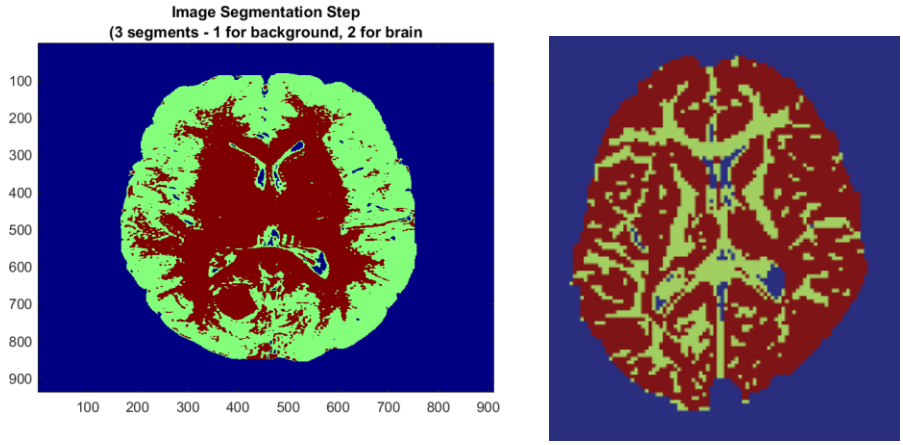




8.2 Skull Stripping (my implementation on the left, paper on the right)



8.3 Segmentation (my implementation on the left, paper on the right)



8.4 SVM

Those results are shown in section 4.

9 Website Links

The data is located here:

https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection

If you run this program do EXACTLY as follows:

1. Go to the link above
2. Click download (you may need to create an account)
3. Go to where the download file is and unzip
4. Go into the resulting unzipped folder “archive”
5. Move the folder “brain_tumor_dataset” to the folder containing main.m
6. Run the program as specified