Automating Image Segmentation Verification and Validation by Learning Test Oracles

Kambiz Frounchi, Lionel C. Briand*, Leo Grady**, Yvan Labiche, and Rajesh. Subramanyan**

Carleton University, Software Quality Engineering Laboratory, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada, {kambiz, labiche}@sce.carleton.ca
*Simula Research Laboratory and University of Oslo, P.O. Box 134, Lysaker, Norway, briand@simula.no
**Siemens Corporate Research, 755 College Road, Princeton, NJ 08540, U.S.A, {leo.grady, rajesh.subramanyan}@siemens.com

ABSTRACT

An Image Segmentation Algorithm is an algorithm that delineates (an) object(s) of interest in an image. The output of the image segmentation algorithm is referred to as a segmentation. Developing image segmentation algorithms is a manual, iterative process involving repetitive verification and validation tasks. This process is time-consuming and depends on the availability of medical experts, who are a scarce resource. We propose a procedure that uses machine learning to construct an oracle, which can then be used to automatically verify the correctness of image segmentations, thus saving substantial resources. During the initial learning phase, segmentations from the first few (optimally two) revisions of the segmentation algorithm are manually verified by experts. The similarity of successive segmentations of the same images is also measured. This information is then fed to a machine learning algorithm to construct a classifier that distinguishes between consistent and inconsistent segmentation pairs based on the values of the similarity measures associated with each segmentation pair. Once the accuracy of the classifier is deemed satisfactory for the purposes of the application, the classifier is then used to determine whether the segmentation, systems' output by subsequent versions of the algorithm under test, are (in)consistent with already verified segmentations from previous versions. This information is then used to automatically make conclusions about the (in)correctness of the segmentations. To demonstrate the performance of the

approach, the proposed solution was successfully applied to 3D segmentations of the cardiac left ventricle obtained from CT scans.

Keywords: Verification and Validation, Test Oracle, Software Quality, Segmentation, Machine Learning

1 INTRODUCTION

Image Segmentation is the act of extracting content of interest from an image [12]. The extracted content is represented by labeling each pixel in the image. Image segmentation algorithms have been devised to automatically segment an image without the need of an expert manually delineating the objects of interest in the image. The usual method of verifying and validating a medical image segmentation algorithm begins by applying a first version of the segmentation algorithm (activity B in Figure 1) to a set of images (the images constitute the test suite, each image representing a test case). The results obtained from the segmentation algorithm are then manually evaluated by medical experts (activity C). If an agreed[1] number of segmentations are correct the algorithm is deemed correct otherwise the algorithm is modified. When modification is required, the revised algorithm is re-applied to the image set and the same evaluation procedure is repeated until a correct version of the segmentation algorithm is reached (Figure 1). Practice shows that the number of iterations can be large, sometimes in the dozens, thus making the evaluation process very time consuming. Indeed, medical experts are required for the evaluation of each iteration and this often results in long waiting times. Manually evaluating such large numbers of subjects is also prone to human errors and inter-expert variability.

We treat this problem as an instance of the *oracle problem*, which is the problem of finding a procedure to assess the correctness of test results (in our case image segmentations) [5]. Our proposed solution leads to the partial automation of segmentation oracles, thus making verification and validation more time-efficient and less reliant on medical experts. We use machine learning to build a classifier that determines the consistency of segmentation pairs (segmentations obtained from different versions of the

[1] Agreed between the segmentation algorithm designer and medical experts.

segmentation algorithm but extracted from the same patient image) and then use this information to predict the correctness of segmentations.

To teach the machine learning algorithm to distinguish between consistent and inconsistent segmentation pairs, the (dis)similarity between different segmentation pairs are quantified using several measures and their consistency is determined from expert evaluations of the first few revisions of the segmentation algorithm. Ideally, as in our case study, two revisions are needed to find an accurate classifier. Though our case study focuses on a specific segmentation application, the approach is re-usable in other (medical) image segmentation verification and validation contexts.

The rest of this paper is organized as follows: Section 2 gives the reader some background about the adopted image segmentation comparison measures and a brief description of the machine learning concepts used in this research work. We detail our test oracle approach in Section 3, listing the results from the cardiac left ventricle segmentation verification and validation study in Section 4 where we describe several classifiers and compare their performance. A discussion on related work is given in Section 4.5 and conclusions are drawn in Section 6.

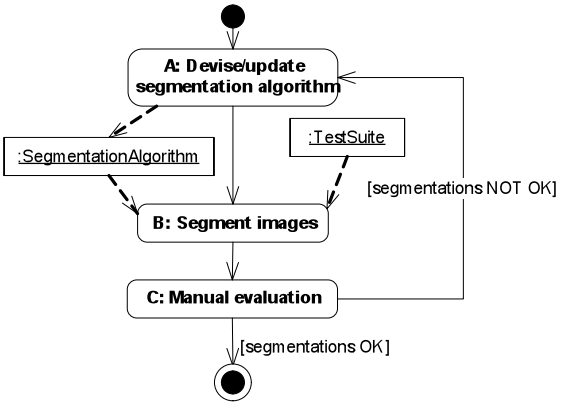


Figure 1 Manual Image Segmentation Algorithm Evaluation Process

2 BACKGROUND

In this section some background is given on two main subjects covered in this paper: 1) the image segmentation similarity measures that are required for training the machine learning algorithms and 2) the machine learning algorithms themselves and their evaluation techniques.

2.1 Similarity Measures

As explained in the previous section, we rely on several measures to quantify the similarity between segmentations with respect to different criteria. Apart from a few measures that we have defined ourselves, the rest of the measures are obtained from the image processing literature and adapted to our needs. We divide the measures into three types: volume difference, overlap, and geometrical measures. An overview of the different categories is given here. For a detailed description of the measures please refer to Appendix A. Table 1 summarizes the measure names, types (VD = Volume Difference, O = Overlap, G = Geometrical), and the acronym descriptions.

Volume difference measures calculate either in absolute or relative terms the difference in the number of voxels labeled in the two segmentations multiplied by the volume associated with each voxel in the image. These measures are relevant because a common purpose for medical image segmentation is the measurement of volume In the application of our case study, the most important output of the segmentation system was the left ventricle volume in the service of computing the ejection fraction.

Overlap measures calculate some kind of overlap between the two segmentations. The intersecting and non-intersecting regions of the two segmentations are identified and different fractions are defined, each measure placing more emphasis on the extent of agreement of some regions of interest. The original references for these measures are in [13, 14].

Table 1 Similarity Measures

Measure	Type	Description
AVD	VD	Absolute value of Volume Difference
ANVD	VD	Absolute value of Normalized Volume Difference
DSC	O	Dice Similarity Coefficient
TC	O	Tanimoto Coefficient
TPVF	O	True Positive Volume Fraction
FPVF	O	False Positive Volume Fraction
ADBD	G	Average Distance to Boundary Difference
HD	G	Hausdorff Distance
BD	G	Baddeley Distance
PMME	G	Peli Malah Mean Error
PMMSE	G	Peli Malah Squared Error
PFOM	G	Pratt's Figure Of Merit
SODI	G	Odet's ODI_n
SUDI	G	Odet's UDI_n
ODI	G	Odet's ODI
UDI	G	Odet's UDI
PAD	G	Principal Axis Difference
RMMSD	G	Root Mean Square Surface Distance

Geometrical measures compare the segmentations in terms of their shape differences capturing variations such as the *distance* between the boundary voxels of the two segmentations. These measures may help in finding such cases where for example a segmentation has a high percentage of overlap with the correct segmentation but incorrectly labels some voxels that make the segmentation incorrect in the view of a medical expert. The original references for these measures can be found in [15-17, 19-22].

2.2 Machine Learning

Machine learning algorithms can be categorized into different types. However, in this research work we are only interested in classification algorithms since we want to learn about the relationships between segmentation (dis)similarities and expert evaluations. The input of these algorithms is a set of instances that are each characterized by the values of a number of attributes and associated with a class (referred to as training instances). The algorithm constructs a classifier that shows some form of relationship between the attributes (for example in the form of rules) that leads to a class. The classifier can now predict the class of unknown instances for which we do not know their class [23]. Better classifiers are constructed when more training instances with proportional number of instances from different classes are available. In our case, an instance is a pair of two segmentations for the same patient but generated by different segmentation algorithm versions, attributes are (di)similarity measures, and classes describe whether expert evaluations for the two segmentations are consistent.

We have used the WEKA-implemented machine learning algorithms J48, JRIP and PART in our case study [23]. The use of decision branches and rules in these machine learning algorithms allows technical and medical experts to easily interpret the classifiers and gain more confidence in the decisions made by the classifier and the overall approach. Although more complicated machine learning algorithms exist, this is left to future work. Our main focus in this paper is to demonstrate our automated segmentation evaluation approach and, as seen in Section 4, the adopted classifiers perform very well in terms of classification accuracy

J48 implements the very well-established C4.5 algorithm that is a standard algorithm to create decision trees. C4.5 uses a divide and conquer approach, choosing different attributes in order of least entropy[2] to divide the instances into different branches, growing the tree recursively and stopping the growth of a branch when the class of the instances in that branch can be determined or in other words a leaf node has been reached. PART uses partial decision trees to construct rules from the branches that lead to a leaf node covering the most instances. It attempts to avoid building a full decision tree for each rule by growing the resultant attribute split subsets with the lower entropies first, which leads to small sub-trees and more generic rules [23]. JRIP implements the RIPPER (Repeated Incremental Pruning to Produce Error Reduction) algorithm [24] that is a rule-induction technique. For each class, JRIP starts by finding a rule that covers most of the training instances and has the best success rate (least number of misclassified instances). This procedure is repeated recursively until all instances are covered for that class and then repeated for the other classes. A procedure known as *incremental reduced-error pruning* refines each rule immediately after construction and a number of *global optimization* stages are applied after the construction of all the rules for further refinement [24]. In reduced-error pruning the training set is split into a growing set to construct the rules and a pruning set for pruning which means fewer instances are used for training. We select C4.5 pruning for J48 and PART. Please refer to [23] for further discussion on pruning techniques.

Filters and wrappers attempt at taking out attributes that do not add any significant improvement in building a better classifier. The Correlation-based Feature Selection (CFS) filter chooses a subset of attributes from the original attribute set that have a high correlation with the class and a low correlation with each other. Wrappers select attributes by first training a classifier from different subsets of the original attribute set and choosing the subset of attributes that trains the best performing classifier. Both filters and wrappers require searching the attribute space. We have chosen an exhaustive search method for CFS where all the possible attribute sets are considered while choosing a

[2] Entropy is a term acquired from information theory that in simple terms conveys the extent of non-uniformity in a group of instances: a group of instances that show equal proportions across classes have an entropy of 1 (entropy ranges from 0 to 1), depicting a uniform distribution.

greedy method for the J48 wrapper as an exhaustive search proves to be very time-extensive in this case because of the requirement to build a classifier for each attribute set. In a greedy search method such as forward hill climbing, we start from an empty subset adding attributes to the set until the addition of no attribute will result in a performance improvement at which point the current subset is the selected subset of attributes. The main motivation for choosing a wrapper and CFS filter were the benchmarking results of different attribute selection methods reported in [25].

A standard technique to test classifier performance is s*tratified 10-fold cross-validation*. This technique splits the training set into 10 folds, each time training the classifier with 9 folds and testing it with the remaining fold. A procedure known as *stratification* randomizes the instances in each fold such that each one contains a similar proportion of the different classes. In order to prevent bias the procedure is repeated 10 times in our experiments.

Metrics such as accuracy and the area under the Receiver Operating Characteristic (ROC) curve [23] are used as indicators of performance in cross-validation. Accuracy refers to the success rate of the classifier (percentage of correct predictions). The ROC curve is a plot of the true positive rate (instances that are correctly classified as positive, in our case consistent) versus the false positive rate. The larger the area under this curve the better the classifier performs, reaching perfect performance when the area is 1.

3 SEMI-AUTOMATED VERIFICATION AND VALIDATION APPROACH

Figure 2 shows an activity diagram depicting the flow of activities in our approach. Two series of activities take place concurrently in this process, as illustrated by two swimlanes: *Segmentation evaluations* and *Learning classifier*. The segmentations are evaluated manually in the segmentation evaluations swimlane (Section 3.1) until an accurate classifier is constructed in the learning classifier swimlane (Section 3.2) thus allowing the automated evaluation of segmentations. We propose a component diagram of the automated oracle in Appendix B.

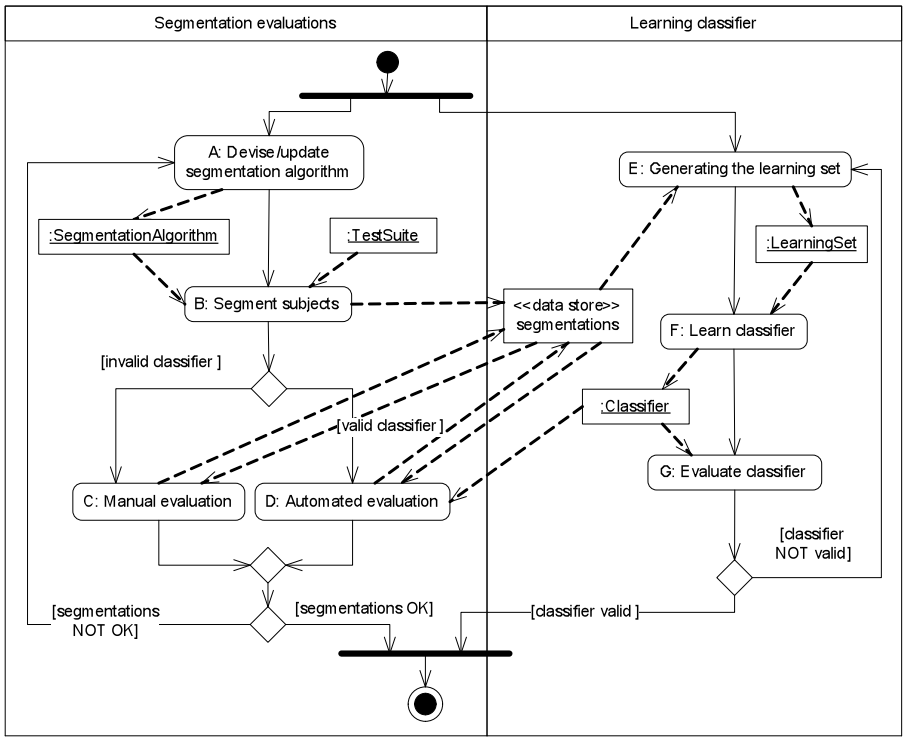


Figure 2 Segmentation evaluation process

3.1 Segmentation Evaluations Swimlane

This swimlane has four activities: activities A to D; and is similar to the activity diagram of Figure 1. The first time the *Devise/Update segmentation algorithm* activity (Activity A) is performed, an initial version of the image segmentation algorithm is devised. Each time this activity is repeated, i.e., when the segmentations produced by the image segmentation algorithm are deemed inadequate, the image segmentation algorithm is revised. Further revisions of the image segmentation algorithm are made in subsequent iterations until a satisfactory set of segmentations are produced.

During the *Segment images* activity (Activity B), the segmentation algorithm produced in activity A is used to segment a set of sample images (*Test suite*) used as benchmark. This results in a set of segmentations, each segmentation corresponding to one sample

image. We refer to the segmentations obtained from revision i of the segmentation algorithm (i.e., during the i^{th} iteration of this swimlane) as segmentation set set_i.

The segmented images are verified either manually or automatically in the next two activities (C and D), depending on whether an accurate classifier has been learnt. If this is not the case (Section 3.2), the segmentations have to be manually evaluated (Activity C—*Manual evaluation*). When an accurate classifier is available (Section 3.2), the evaluation is done automatically: activity *Automated evaluation* (Activity D). In activity D, the classifier predicts, based on the similarity measurements (Activity E) between the segmentations produced by the current revision (i) of the segmentation algorithm and the segmentations produced by previous revisions (j<i) of the segmentation algorithm, whether or not a segmentation pair is consistent. Table 2 shows how the correctness of segmentation n produced by revision i of the segmentation algorithm ($S_{n,i}$) can be obtained from the consistency classifications of the learnt classifier. If the classifier predicts segmentation $S_{n,i}$ to be consistent (with respect to a number of similarity measures) with a correct segmentation $S_{n,j}$ (j<i) then segmentation $S_{n,i}$ is predicted to be correct. If segmentation $S_{n,i}$ is predicted to be inconsistent with a correct segmentation or consistent with an incorrect segmentation then it is predicted to be incorrect. In the case where segmentation $S_{n,i}$ is inconsistent with an incorrect segmentation, no conclusion can be drawn and the correctness of segmentation $S_{n,i}$ has to be manually evaluated by an expert.

Table 2 Mapping between classifier results and the evaluation of the test image segmentation

Evaluation of $S_{n,j}$ (j < i)	**Predicted consistency of segmentation pair ($S_{n,j} – S_{n,i}$)**	**Evaluation of $S_{n,i}$**
Correct	Consistent	Correct
Correct	Inconsistent	Incorrect
Incorrect	Consistent	Incorrect
Incorrect	Inconsistent	Requires manual evaluation

After either activities C or D, if an agreed[1] percentage of image segmentations are evaluated to be incorrect, we go back to activity A where the image segmentation

algorithm is revised. Otherwise, the testing process ends and the current revision of the image segmentation algorithm is deemed to be correct.

3.2 Learning Classifier Swimlane

This swimlane has three activities: activities E to G. In activity E (*Generating the learning set*), pairs of segmentations obtained from multiple revisions of the image segmentation algorithm (current revision i, and revision j, j<i) are compared using a set of similarity measures (Section 2.1). At least the first two sets of segmentations generated by the first two revisions of the segmentation algorithm are required to get the first set of similarity measurements (i.e., the first set of evaluated segmentation pairs from versions 1 and 2 of the segmentation algorithm). In other words, at least two iterations of the *Segmentation evaluations* swimlane (with manual evaluation in Activity C) are necessary.

Pairing segmentations of the same images/patients across two segmentation sets set_i and set_j results in three distinct subsets of paired segmentations. The first set is composed of the pairs of segmented images that were both deemed correct by an expert, denoted by set_{yy} (i.e., 'y' for "yes" for the two versions). The second set is composed of the pairs of segmented images where either the first or second segmented image was deemed incorrect, denoted by set_{yn} (one is correct: 'y', and one is incorrect: 'n'). The third set is the set of all the pairs of segmented images that were both considered incorrect, denoted by set_{nn}.

The machine learning algorithm (Activity F) does not use set_{nn} as the information obtained from comparing two incorrectly segmented images would not help the learning algorithm construct a classifier to recognize diagnostically equivalent segmentations. Diagnostically equivalent segmentations refers to two segmentations that both lead to the same diagnostic by the medical expert. Two segmentations may be incorrect for two completely different reasons and thus we cannot categorize them as consistent with each other. Table 3 explains how we categorize a pair of segmentations to be (in)consistent where $S_{n,i}$ and $S_{n,j}$ are two segmentations obtained from image n using versions i and j of the segmentation algorithm, respectively.

The application of the comparison measures to image pairs in *setyy* and *setyn* generates a set of tuples in the form of (smi1, …, smik, Consistency), where *smij* denotes similarity measures j on image pair I and "Consistency" cam take two values: *yy* (consistent) or *yn* (inconsistent). Index *k* is the number of similarity measures. This set of tuples is the training set that is fed into the machine learning algorithm: activity F (Learn Classifier). The machine learning algorithm learns which combination of measures and which ranges if their values depict consistent or inconsistent segmentation pairs.

Table 3 Consistency of two segmentations

Manual evaluation of $S_{n,i}$	**Manual evaluation of $S_{n,j}$**	**Class**
Correct	Correct	Consistent
Correct	Incorrect	Inconsistent
Incorrect	Correct	Inconsistent
Incorrect	Incorrect	Unusable (not a class)

The learnt classifier (activity F) is validated using techniques such as 10-fold cross-validation (Section 2.2) in activity G (*Evaluate classifier*). A classifier is deemed *valid* if the average error estimate is considered to be sufficiently low to be used in practice. This means that the classifier correctly predicted a reasonable proportion of segmentations under test to be consistent or inconsistent with a reference segmentation of the same image, or in other words predicted if the segmentation under test is correct or not. Once the classifier is learnt, the evaluation in the *Segmentation evaluation* swimlane can be done automatically (Activity D). Even in the case where we do not succeed in learning a classifier with very high average accuracy, we can expect that there will always be some parts of the classifier predictions with very low error rate that can be trusted with high confidence. For example, particular branches in decision trees or rules in rule induction techniques may exhibit much higher accuracy than other branches or rules. If the classifier can provide accurate predictions for a practically significant number of cases, then the classifier is a good candidate for partially automating the segmentation evaluations.

4 CASE STUDY

This section describes the application of our approach to the verification and validation of a left ventricle segmentation algorithm (some examples of correct and incorrect cardiac left ventricle segmentations are given in Appendix C). The details of our experiments are described in Section 0 (Appendix D describes the tools we used for the purposes of this case study). We then explain the attribute sets that we have used to train the classifiers (Section 4.2), show the best performing classifiers and give some intuition on how to interpret them (Section 4.3) and compare the performance of the classifiers with respect to the type of attribute set, machine learning algorithm and the filtering method used in Section 4.4. Section 4.5 compares the run-time of the different measures with a small summary of the case study results in Section 4.6.

4.1 Experiment Setup

In this case study we intend to reach the following objectives:

1) Analyze the performance of our verification and validation approach to a cardiac left ventricle segmentation algorithm devised by Siemens Corporate Research. In other words, can this approach lead to practical benefits?
2) Investigate which similarity measures have the highest impact in determining the consistency of two segmentations and understand the tradeoff between using more expensive measures (in terms of run-time and complexity) and achieving better classifier performance.
3) Analyze and compare the performance of several classification algorithms combined with attribute filtering techniques to determine whether they are appropriate for our application and produce plausible results.

The segmentation algorithm we used in this case study identifies the left ventricle of the heart[3] from a Computed Tomography (CT) Scan. CT-Scan images of the heart of 50 patients have been taken at different times during the cardiac cycle, resulting in 181 CT-Scans. Each CT-Scan is a set of 2D images (that set creates a 3D image). Due to the

[3] The volume of the left ventricle can then be computed at different times during the cardiac cycle for diagnosis purposes.

variety in acquisition protocol and equipment across clinical centers, each CT-Scan has between 53 and 460 2D images, and each 2D image depicts either a 256 by 256 or 512 by 512 pixel slice of the heart. Each CT-Scan, representing a patient's heart at a given instant, constitutes a test case in the test suite. The segmentations were obtained using a form of the general segmentation algorithm presented in [26] that was customized for cardiac segmentation.

Two successive versions of the segmentation algorithm were considered. The 181 CT-Scans were segmented and segmentations were evaluated by a medical expert (activities A, B, and C in Figure 2 were executed twice).

Pairs of segmentations from the two segmentation versions for the same CT-Scan were compared using the 18 similarity measures of Table 1 (activity E in Figure 2). 181 tuples (recall Section 3.2) were thus generated combining similarity measures and the consistency class of each pair: 104 tuples were Consistent and 74 tuples were Inconsistent.

As explained in Section 2.2, we considered classification (machine learning) algorithms and selected C4.5, JRIP, and PART (activity F in Figure 2). (Though activity F in the process of Figure 2 does not require the use of several classification algorithms, in a research process like ours, we are interested in evaluating several of them.) The performance of those algorithms is evaluated using the 10 stratified 10-fold cross validations by measuring their classification accuracy and the area under the ROC curve (activity G in Figure 2).

4.2 Attribute Sets

Recall from Section 2.1 that we consider three types of measures: volume difference, overlap, and geometrical measures. The latter category tends to be much more expensive to compute than the former two. We are interested in studying the changes in performance of the learning process when using only the least expensive type of measure, the two least expensive types, or when using all measures. Applying (or not) two attribute

filtering mechanisms to these three, we end up considering nine different attribute/measure sets (Table 4) to train the machine learning algorithms.

Table 4 Attribute Sets

	Attribute Set	Selection Criteria	Label
1	TC-DSC-TPVF-FPVF-AVD-ANVD	Overlap and volume difference measures	OV
2	TC-DSC-AVD-ANVD	Overlap and volume difference measures chosen by the CFS filter using exhaustive search	OVC
3	TC-FPVF	Overlap and volume difference measures chosen by the J48 decision tree wrapper using forward selection greedy search	OVW
4	BD-HD-PFOM-RMSSD-ADBD-SODI-ODI-SUDI-UDI-PAD-PMME-PMMSE	Geometrical measures	G
5	HD-BD-PFOM-RMSSD-ODI-UDI-PAD	Geometrical measures chosen by the CFS filter using exhaustive search	GC
6	PFOM-SODI-SUDI	Geometrical measures chosen by the J48 decision tree wrapper using forward selection greedy search	GW
7	TC-DSC-TPVF-FPVF-AVD-ANVD-BD-HD-PFOM-RMSSD-ADBD-SODI-ODI-SUDI-UDI-PAD-PMME-PMMSE	All measures	A
8	TC-DSC-AVD-ANVD-HD-RMSSD-ODI-UDI	All measures chosen by the CFS filter using exhaustive search	AC
9	TC-FPVF-SODI	All measures chosen by the J48 decision tree wrapper using forward selection greedy search	AW

We want to see how well the different types of similarity measures would do standalone and what would be the effect of combining these measures. The overlap and volume difference measures have been combined together. These measures do not consider the shape differences between the two segmentations, thus being less complex in terms of computation, and also take noticeably less run-time (more than two orders of magnitude faster in most cases) compared to the geometrical measures. We will refer to these measures as *simple* measures.

In order to investigate how the classifier performance would change when built using a filtered set of attributes we have applied the Correlation-based Feature Selection (CFS)

filter and a J48 based wrapper to the three main categories of attribute sets creating six more attribute sets.

4.3 Classifiers

Before we delve into the classifiers, let us summarize the configuration parameters of the machine learning algorithms (Table 5). The parameters for J48 and PART are similar as PART uses partial decision trees in its construction process. The number of instances per leaf node parameter restricts the minimum number of instances that could reach a leaf node to four. Higher values for this parameter take away the freedom of C4.5 to form leaf nodes but very low values also may cause over-fitting to the data. Values are chosen after finding that it is either significantly better or equivalent (using a *t*-test using accuracy as the comparison criterion) when the parameter is swept from 2 to 10. Other parameters are also empirically optimized following a similar procedure. C4.5 pruning also proves to be either significantly better or equivalent to reduced error pruning, which is the motivation behind its selection for J48\PART.

Figure 3 shows the best performing decision tree trained with the wrapper selected attributes (attribute set AW in Table 4) and produced by J48. Based on performing 10 times a 10 fold cross validation, this classifier accuracy is 94.92% and its ROC area is 0.95. The numbers shown in the parentheses in Figure 3 respectively (from left to right) represent the number of training instances that reach each leaf node and the number of these instances that are incorrectly classified by that leaf node.

Table 5 Classifier configuration parameters

Parameter	Value	Algorithm
Number of instances per leaf node	4	J48\PART
Pruning method	C4.5	J48\PART
Number of folds for pruning	3	JRIP
Number of global optimizations	4	JRIP

Reading the decision tree shows that if the overlap (using measure TC) between the two segmentations is less than approximately 75% then the segmentations are categorized as inconsistent (as confirmed by 59 of the 74 inconsistent pairs in the training set with only 1 misclassified instance), otherwise a distance of less than 0.13 determined by the geometrical measure SODI will lead to a consistent pair (confirmed by 98 of the 104 consistent pairs, with only 3 misclassifications). The overlap measure FPVP helps classify the remaining pairs of segmentations, when the TC overlap is greater than 75% and the SODI distance is greater than 0.13. If the FPVP overlap between the two segmentations is less than 0.28% then we have a consistent pair again, otherwise segmentations are inconsistent. The branch predictions can be considered quite accurate with a minimum error rate of 3% (TC-SODI-consistent branch) and a maximum error rate of 22% (TC-SODI-FPVF-consistent branch), which only applies to a small number of the training instances (2 errors out of only 9 instances). In practice, the maximum error rate justifiable should be determined for each application, a threshold above which the manual verification of a segmentation if required.

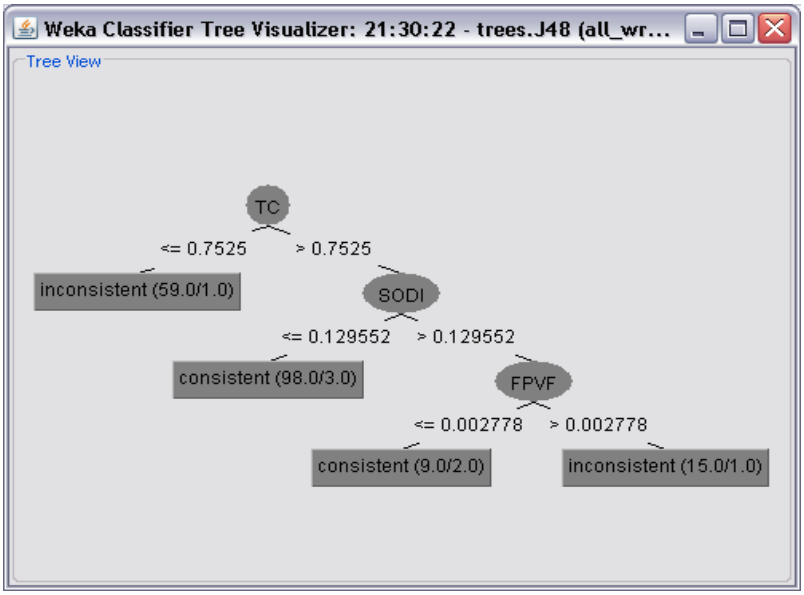


Figure 3 Best decision tree classifier

Trained with the same attribute set (set AW), PART constructs four rules:

Rule 1: If ((TC>0.7525) & (SODI<=0.129552) then class = consistent (98/3)

Rule 2: Else if (TC<=0.76311) then class = inconsistent (59/1)

Rule 3: Else if (FPVF>0.002778) then class = inconsistent (15/1)

Rule 4: Else class = consistent (9/2)

Interpreting each of the branches in the decision tree of Figure 3 as a rule, we see that the PART rules are essentially the same as the J48 classifier in Figure 3 with a small difference in the threshold used for TC for identifying inconsistent pairs (0.76311 in the second rule versus 0.7525 in the decision tree). This stems from the fact that PART constructs a different decision tree to generate the second rule, which means that it may not necessarily come up with the same threshold as the first rule when trying to cover the remaining instances that are not covered by rule 1.

On the same attribute set (set AW), JRIP generates three rules:

Rule 1: If (TC<=0.737932) then class=inconsistent (58/1)

Rule 2: Else if ((SODI>=0.131275) & (FPVF>=0.002838) then class =

inconsistent (16/1)

Rule 3: Else class = consistent (107/5)

JRIP covers all the instances that belong to one class before proceeding to the next class, whereas PART uses partial decision trees for rule induction and depending on the chosen leaf at each stage, instances belonging to a different class may be covered. Again, the rules produced by JRIP are very similar to the ones produced by J48 and PART, with small differences in thresholds used for TC, SODI and FPVP.

We see from all three classifiers that TC is the best discriminator for negative instances (inconsistent pairs) while the combination of TC and SODI is the best

discriminator of positive instances (consistent pairs). A discussion on the classifiers produced by the other attribute sets using the three machine learning algorithms (J48, PART and JRIP) is given in [1].

4.4 Classifier Performance Comparisons

In this section we investigate the performance of the classifiers with respect to the attribute set (Section 4.4.1), the machine learning algorithm (Section 4.4.2) and the application of filters\wrappers (Section 4.4.3).We refer to Figure 4 and Figure 5 throughout the discussion. Each point in these figures is the average across the cross validation results and thus the *t*-test results described in this section are ran over a sample size of 100 (using the accuracy metric).

4.4.1 Effect of attribute set type.

In this section, we compare the performance of the classifiers trained with the geometrical measures compared to simple measures (overlap and volume difference) and then investigate how using all the measures will affect the classifier performance using the accuracy and ROC area metrics.

As Figure 4 and Figure 5 show, the geometrical measures (set G) do not show any noticeable performance improvement compared to just using the simple measures (set OV). This is also confirmed with a *t*-test where no *significant* performance difference is found between the classifiers trained by geometrical measures as opposed to simple measures. This indicates that using solely geometrical measures may not be required for this case study considering that they result in no performance improvement and also are more complex to implement and test. Indeed we found that, unless an attempt to develop more efficient implementations is undertaken, geometrical measures will take much more time to execute than the simple measures (several orders of magnitude slower).

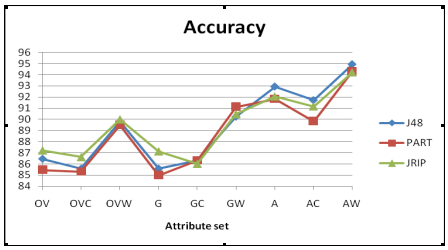


Figure 5 Accuracy of classifiers

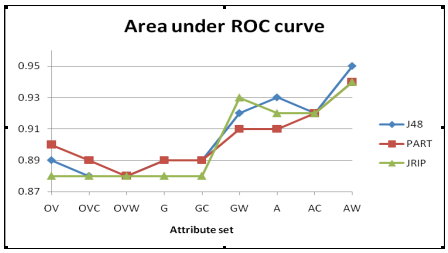


Figure 5 Area under ROC curve of classifiers

However, combining the geometrical and simple measures (set A) seems to be very promising. We see (Table 6) significant accuracy improvements for all three classifiers compared to just using either of the simple (from 4.82% to 6.47%) or geometrical measures (from 4.91% to 7.35%). A similar trend is visible for the ROC areas (Figure 5). This shows that the classifiers achieve very high discriminating power when taking advantage of both the simple and geometrical measures.

Table 6 Comparison of classifier accuracies with respect to the attribute set category

	Sets				
	OV	**G**	**A**	**Δ (A-OV)**	**Δ (A-G)**
J48	86.46	85.58	92.93	6.47	7.35
PART	85.46	84.98	91.83	6.37	6.85
JRIP	87.23	87.14	92.05	4.82	4.91

4.4.2 Effect of different machine learning algorithms.

In this section we investigate whether machine learning algorithms yield significant performance differences. Table 7 shows that there is not a noticeable difference in the accuracy of the classifiers trained with the three machine learning algorithms. JRIP trains

a better classifier when only having geometrical measures available performing 2.16% better than PART and 1.56% better than J48. When using all the measures, regardless of applying the wrapper, J48 generally performs slightly better, a result also confirmed with the ROC area. We have not considered the results for the classifiers that are trained with the CFS filtered attribute sets as these classifiers do not achieve any significant accuracy improvement over just using the original attribute sets (Section 4.4.3). From Table 7, one can conclude that using any of the three machine learning algorithms would be equivalent except for the fact that one may consider interpreting decision trees easier than rules.

Table 7 Comparison of classifier accuracies with respect to the machine learning algorithm

	J48	PART	JRIP	Δ(J48-PART)	Δ(J48-JRIP)	Δ(PART-JRIP)
OV	86.46	85.46	87.23	1	Not Sig.	-1.77
OVW	89.78	89.50	90.00	Not Sig.	Not Sig.	Not Sig.
G	85.58	84.98	87.14	Not Sig.	-1.56	-2.16
GW	90.24	91.13	90.46	-0.89	Not Sig.	Not Sig.
A	92.93	91.83	92.05	1.1	Not Sig.	Not Sig.
AW	94.92	94.32	94.21	0.6	0.71	Not Sig.

4.4.3 Effect of wrappers and filters.

The effect of filtering and using wrappers is investigated in this section. Table 8, Table 9, and Table 10 show respectively the classifier accuracies achieved when trained by the simple, geometrical and all measures sets before and after applying the CFS filter and J48 wrapper. We see that using the training instances with only the attributes selected by the wrapper always achieve a significant improvement in the accuracy of the trained classifier compared to using all the attributes: the maximum accuracy improvement is 7.66% for the J48 classifier when using the geometrical measures. Similar trends were observed with the ROC area. On the contrary the CFS filter in some cases significantly degrades the classifier's accuracy. Using a PART wrapper or a JRIP wrapper also improves classifier accuracy, but the improvement is not significantly better than the J48 wrapper using any combination of classifier\attribute set, thus we do not report the results here due to space constraints.

Table 8 Comparison of classifier accuracies with respect to the use of filters and wrappers (using only simple measures)

	OV	OVC	OVW	Δ(OVC-OV)	Δ(OVW-OV)
J48	86.46	85.59	89.78	-0.87	4.19
PART	85.46	85.30	89.50	Not Sig.	4.04
JRIP	87.23	86.63	90.00	Not Sig.	2.77

Table 9 Comparison of classifier accuracies with respect to the use of filters and wrappers (using only geometrical measures)

	G	GC	GW	Δ(GC-G)	Δ(GW-G)
J48	85.58	86.31	93.24	Not Sig.	7.66
PART	84.98	86.31	90.13	1.33	5.15
JRIP	87.14	86.03	90.46	Not Sig.	3.32

Table 10 Comparison of classifier accuracies with respect to the use of filters and wrappers (using all measures)

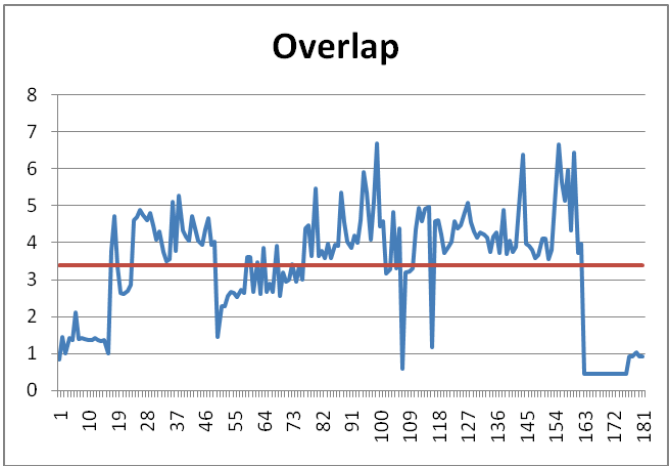
	A	AC	AW	Δ(AC-A)	Δ(AW-A)
J48	92.93	91.71	94.92	-1.22	1.99
PART	91.83	89.84	94.32	-1.99	2.49
JRIP	92.05	91.17	94.21	Not Sig.	2.16

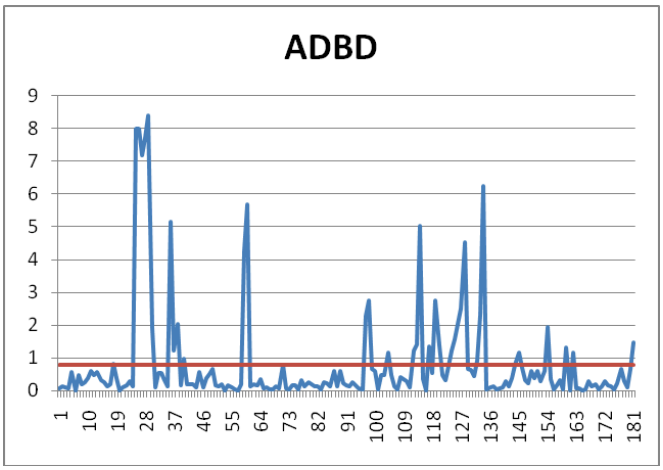
4.5 Timing Considerations

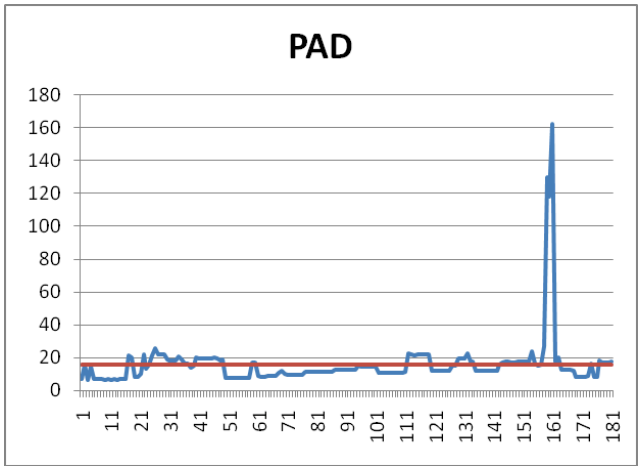
As mentioned in Appendix D, the comparison measures were implemented in the MATLAB environment. The matrix-oriented environment of MATLAB makes the implementations very concise but MATLAB suffers from being very slow in 3D image processing operations. The graphs in Figure 6 show the calculation time of the various

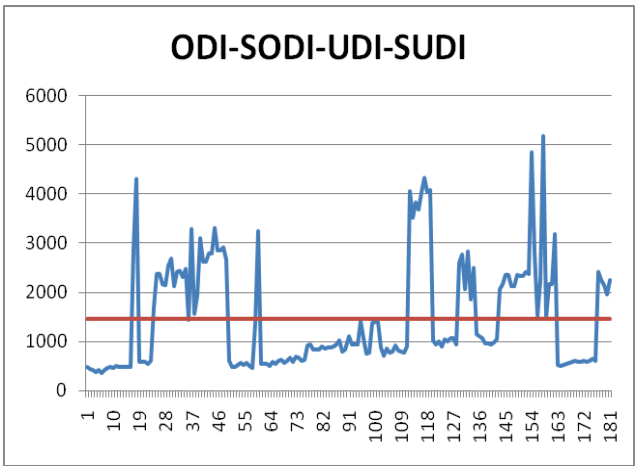
comparison measures for each of the 181 segmentation pair comparisons. The time varies based on the size of the segmentation files that have to be compared. Some graphs represent the aggregate time consumption of a few measures. Graph 1 shows the total time of calculating TPVF, FPVF, TC and DSC. Although graphs 1, 4, 5 and 6 show the aggregate calculation time of the measures but as these measures use mostly common operations in their implementations, the time involved in calculating any one of them is comparable with the aggregate time.

Table 11 shows the average calculation times of the measures. As shown, the Average Distance to Boundary Difference is the fastest measure, followed by the overlap measures (note that we are showing the total time of all the four overlap measures) and the principal axis. The volume difference measures are also very fast to compute. They have not been involved here as those numbers were readily available in this case study and did not require any implementation. The most time-intensive measures are the Hausdorff and Baddeley distances followed by the other geometrical measures. We see a significant gap in calculation times of the majority of the geometrical measures (excluding ADBD and PAD) and the overlap\volume difference measures. This is due to the fact that the overlap measures do not require any 3D processing while the geometrical measures all require 3D processing which MATLAB proves to be very slow at. Also the fact that MATLAB uses an interpreter-based language makes it slower than a compiler based language. Its garbage collection mechanisms do not prove to be very effective too as in our experiments large portions of RAM (up to 8 GBs) were allocated at some points of time. It is important to note that these time considerations will be less notable if using an efficient C implementation of the measures, as the original image segmentation algorithm is implemented in the C language and is significantly faster than the geometrical measure calculations (the segmentation algorithm takes up to a few 10s of seconds to output the segmentation of the 3D volume). Considering the relatively small size of the training set, the classifier construction and classification times were insignificant.


Overlap
8
7
6
5
4
3
2
1
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181


ADBD
9
8
7
6
5
4
3
2
1
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181


PAD
180
160
140
120
100
80
60
40
20
0
1
11
21
31
41
51
61
71
81
91
101
111
121
131
141
151
161
171
181


ODI-SODI-UDI-SUDI
6000
5000
4000
3000
2000
1000
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
154
163
172
181

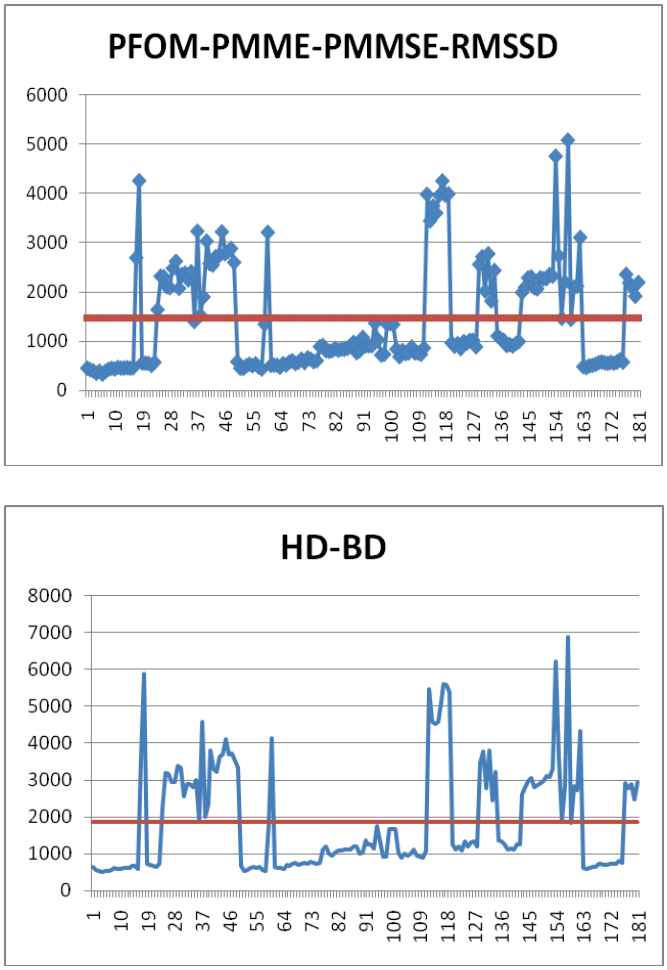


Figure 6 Time consumption of comparison measures (x-axis = segmentation pair number, y-axis = time consumption in seconds)

Table 11 Average comparison measure calculation times

Comparison Measures	**Average Calculation Time (secs)**
ADBD	0.80
overlap (TPVF, FPVF, TC, DSC)	3.39

PAD	15.62
PFOM, PMME, PMMSE, RMSSD	1435.57
ODI-SODI-UDI-SUDI	1465.50
HD-BD	1853.61

4.6 Conclusions

We see that taking advantage of both the simple and geometrical measures results in significantly better performing classifiers compared to using only simple or geometrical measures. The use of wrappers improves the results even further. Measures such as the Tanimoto coefficient (overlap) and SODI (geometrical) prove to be very strong measures for distinguishing consistent pairs of segmentations from inconsistent ones as they are used in the majority of the best performing classifiers. Although all the classifiers perform reasonably well but the J48 classifier is the best performing classifier achieving an approximately 95% accuracy, 0.9 kappa statistic and 0.95 area under the ROC curve.

5 RELATED WORK

There is a large body of work in the image processing field on studying and improving image segmentation algorithms but this is not the focus of our work. Rather we try to find a solution to the oracle problem in the (medical) image segmentation verification and validation domain. Defining an oracle is not a trivial task and may not be feasible at all times.

In [2, 3] different solutions to the oracle problem are proposed: design redundancy, data redundancy, consistency checks, and the use of simplified data. Design redundancy attempts to check if the output of the software under test corresponds to one (or many) extra implementation of the specification of the software under test. It may be that the output of all the software versions is incorrect which is less likely if it is assumed that the software versions are independent. The independence assumption has been empirically debated in [9], showing that it may not at all times and care should be taken on the

reliability of multi-version programming. Design redundancy is similar to multi-version programming in regression testing [4]. Our approach is similar to design redundancy since we use different implementations of the segmentation algorithm. However, there is no design redundancy. In data redundancy, similar to N-copy programming in the fault tolerance domain [7], the input is re-expressed in different forms and the output of the original input is checked for agreement with the re-expressed inputs. Consistency checking simplifies the oracle to just verifying whether certain plausible conditions are met by the software under test, and testing with simplified inputs ignores testing with the more complex inputs and only considers simpler inputs for which an oracle can be devised.

None of these approaches offer any quantification of the reliability of the oracle. Design redundancy has the highest overhead though this cost is argued to be very low in [11] as the implementation of the software is only a fraction of the overall cost of a software project.

In the case where formal specifications exist for the software under test, researchers have proposed methods to derive oracles from the formal specifications. This has been investigated for real-time systems in [8], where the test class consists of some test executions or sequences of stimuli. Test oracles in the form of assertions are obtained using symbolic interpretation of the specifications for each control point. A discussion on oracles that are composed of assertions based on properties specified either in first order logic or computation tree logic is given in [6]. Assertions under the form of pre and post-conditions can also be used as test oracles [10].

The use of machine learning in [18] for evaluating the best of two segmentations, produced by two different segmentation algorithms (or two different settings of one segmentation algorithm), presents similarities to our approach. However, they do not use any similarity measures to compare the two segmentations as we do for finding (in)consistent segmentation pairs. Instead, to measure the goodness (e.g., color uniformity) of each segmentation, they measure it separately. Each segmentation measurement leads to one decision tree, and the comparison of segmentations is obtained

by combining the two decision trees using meta-learning. The construction of the decision trees and the accuracy of the modeling is not clearly discussed in [18]. It is also noteworthy that their measures require the processing of image properties such as color and texture while our measures only rely on the data from the segmentation labeling.

6 CONCLUSIONS

In this paper we have proposed an approach to replace the oracle constituted by (medical) experts by an automated oracle. It is relying on machine learning to construct a classifier from similarity measures that can predict the consistency between two segmentations. This is then used to predict the correctness of new segmentations as algorithms evolve. This approach is generic in the sense that it can be applied to any image segmentation software that goes through such an iterative development\verification procedure. The machine learning algorithms are *plug and play* components that can be replaced with other algorithms if they result in better performance. The approach saves time and effort required by (medical) experts, thus leading to faster development time to delivery and hopefully increased testing.

We investigated the performance of our approach on the evaluation of cardiac left ventricle segmentation algorithms. Although there is room for improvement, the results are very promising and fairly simple classifiers show very good classification accuracies, thus suggesting that the correctness of most segmentations can be determined automatically. For example, using C4.5 and an attribute set of all the measures that have been filtered by a wrapper, we achieve an average accuracy of approximately 95% and an average area of 0.95 under the ROC curve when only using the training data obtained from the first two revisions of the segmentation algorithm. Using machine learning also helps to understand which similarity measures are relevant in determining what differences between segmentations are important from a medical standpoint. Results also show that the choice of a particular learning algorithm is not a significant decision, among the ones we considered here generating logical rules. Using a wrapper to pre-select similarity measures is however effective. For maximum accuracy, all types of similarity measures should be combined in the constructed classifiers.

Future work will be directed towards testing the performance of the approach on various other segmentation applications. Most of the measures defined in this report can be re-used in other segmentation contexts as the measures are not dependant on the left ventricle segmentation problem. Even though the results in this study are promising, more experiments need to be conducted in order to investigate how many segmentation algorithm iterations are required for the machine learning algorithm to be able to converge towards an accurate classifier in other applications. Also trying out this solution during the life cycle of a real-world software project will lead to a better quantitative understanding on the extent of improvement in the quality of the software and the time required for completing the project.

References

[1] K. Frounchi, "Learning a Test Oracle Towards Automating Image Segmentation Evaluation", M.A.S Thesis, Carleton University, Ottawa, Canada, 2008.

[2] M.D. Davis and E.J. Weyuker, "Pseudo-oracles for non-testable programs", *ACM*, 1981.

[3] E.J. Weyuker, "On Testing Non-testable Programs", *The Computer Journal*, vol 25, no. 4, 1982.

[4] P. Amman and J. Offut, *Introduction to Software Testing*, Cambridge University Press, 2008.

[5] A.P. Mathur, *Foundations of Software Testing*, Addison Wesley Professional, 2007.

[6] P.D.L. Machado and W.L. Andrade, "The Oracle Problem for Testing against Quantified Properties", *QSIC*, 2007.

[7] P.E. Ammann an J.C. Knight, "Data Diversity: An Approach to Software Fault Tolerance", *IEEE Transactions on Computers*, vol. 37, no. 4, 1988.

[8] D.J. Richardson, S.L. Aha and T.O.O'Malley,"Specification-based Test Oracles for Reactive Systems", *ACM*, 1992.

[9] J.C. Knight and Nancy G. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming", *IEEE Transactions on Software Engineering*, SE-12(1), pp. 96-109, 1986.

[10] L.C. Briand, Y. Labiche and H. Sun, “Investigating the Use of Analysis Contracts to Improve the Testability of Object Oriented Code”, *Software - Practice and Experience (Wiley)*, vol. 33 (7), pp. 637-672, 2003.

[11] T. Gilb, “Software Metrics”, New Jersey, 1977.

[12] D.L. Pham, C. Xu, and J.L. Prince, “A Survey of Current Methods in Medical Image Segmentation”, Department of Elec. And Comp. Eng., The john Hopkins University, and Laboratory of Personality and Cognition, National Institute on Aging, Technical Report JHU/ECE 99-01, Jan. 1998.

[13] J.K. Udupa, et. al, “A framework for evaluating image segmentation algorithms”, *Elsevier Computerized Medical Imaging and Graphics*, vol. 30, pp. 75-87, March 2006.

[14] W.R. Crum, P. Camara, and D.L.G. Hill, “Generalized overlap measures for evaluation and validation in medical image analysis”, *IEEE Transactions on Medical Imaging*, vol. 25, no. 11, pp. 1451-1461, November 2006.

[15] R. Klette, and A. Rosenfeld, *Digital Geometry: Geometric Methods for Digital Picture Analysis*, Elsevier, 2004.

[16] C. Rosenberger, S. Chabrier, H. Laurant, and B. Emile, “Unsupervised and Supervised Image Segmentation Evaluation,” in Advances in Image and Video Segmentation, Y. Zhang, Ed. IGI, 2006, pp. 365-393.

[17] X. Deng, et. al, “On Simulating Subjective Evaluation Using Combined Objective Metrics for Validation of 3D Tumor Segmentation”, *MICCAI*, 2007.

[18] H. Zhang, S. Cholleti, and S. A. Goldman, “Meta-Evaluation of Image Segmentation Using Machine Learning”, *IEEE CVPR*, 2006.

[19] A. J. Baddeley, “An Error Metric for Binary Images”, *Proceedings of Robust Computer Vision*, 1992.

[20] C. Odet, B. Belaroussi, and H. Benoit-cattin, “Scalable Discrepency measures for segmentation evaluation”, *IEEE ICIP*, 2002.

[21] E. Abdou, and W. K. Pratt, “Quantitave Design and Evaluation of Enhancement/Thresholding Edge Detectors”, Proceedings of the IEEE, vol. 67, no. 5, 1979.

[22] T. Peli, and D. Malah, “A Study of Edge Detection Algorithms”, *Computer graphics and image processing*, 20:1-21, 1982.

[23] I. H. Witten, and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Second Edition, Elsevier, 2005.

[24] W. W. Cohen, "Fast Effective Rule Induction", *Twelfth International Conference on Machine Learning*, pp. 115-123, 1995.

[25] M. A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining", IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 3, 2003.

[26] L. Grady, "Random Walks for Image Segmentation", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 28, No. 11, pp. 1768-1783, Nov., 2006.

[27] J. H. Friedman, J. L. Bentley and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209-226, 1997.

Appendix A Similarity Measure Descriptions

In this appendix we give a more detailed description on the similarity measures explaining each category separately: volume difference measures, overlap measures and geometrical measures. For further discussion on each measure please refer to the original references cited in the text. Throughout the discussion, SUT (Segmentation Under Test) is the segmentation that has an unknown evaluation (i.e. correct\incorrect) and RS (Reference Segmentation) is the segmentation with a known evaluation. Set theory is the adopted notation for defining the equations.

Volume Difference Measures

As their name implies, volume difference measures calculate either in absolute (Equation 1) or nominal (Equation 2) terms the difference in the number of pixels labeled in the two segmentations multiplied by the volume associated with each pixel in the image.

$$AVD = \left| Vol_{sut} - Vol_{rs} \right| \quad (1)$$

$$ANVD = \frac{|Vol_{sut} - Vol_{ts}|}{Vol_{ts}} \quad (2)$$

Overlap Measures

Overlap measures calculate some kind of overlap between the two segmentations. Dice's Similarity Coefficient (DSC) [14] defined in Equation 3 divides the number of common pixels between SUT and RS by the average number of pixels in the two segmentations and the Tanimoto Coefficient (TC) [14] defined in Equation 4 divides it by the number of pixels in the union of the two segmentations (Figure 7). In Figure 7, SUT and RS refer to the labeled region of the segmentations.

$$DSC = \frac{2|SUT \cap RS|}{|SUT| + |RS|} \quad (3)$$

$$TC = \frac{|SUT \cap RS|}{|SUT \cup RS|} \quad (4)$$

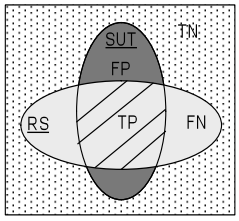


Figure 7 Mutually exclusive overlapping regions between the Segmentation Under Test (SUT) and the Reference Segmentation (RS)

We define two more overlap measures taken from statistical decision theory as follows [13]:

$$TPVF = \frac{|TP|}{|RS|} \quad (5)$$

$$FPVF = \frac{|FP|}{|U - RS|} \quad (6)$$

In these measures pixels that are (not) labeled in RS are referred to as *positive (negative)*, thus for example a truly positive pixel labeled by SUT is a pixel that is common with RS. TPVF stands for Truly Positive Volume Fraction and divides the number of pixels in SUT that are common with RS i.e. the Truly Positive ($TP = SUT \cap RS$) region by the number of pixels in RS. FPVF or the Falsely Positive Volume Fraction divides the number of pixels that are labeled in SUT but not a part of RS (the False Positive ($FP = SUT - RS$) region) by the number of pixels that are not part of RS. U in Equation 6 refers to all the labeled and non-labeled pixels in the segmentation i.e. $U = TP \cup TN \cup FP \cup FN$ where TN is the Truly Negative region or the region that is not labeled in neither SUT nor RS ($TN = U - SUT - RS$) and FN is the Falsely Negative region that contains pixels that according to RS should be labeled but have not been labeled by SUT i.e. $FN = RS - SUT$.

Geometrical Measures

Geometrical measures take into account the shape of the segmentations under comparison, capturing differences such as the size and position of the segmentation boundaries. To do so the distance between different sets of pixels in the two segmentations needs to be measured. This is done by constructing the distance map of each segmentation which assigns each pixel of the segmentation its distance to the segmentation boundary. We formally define the distance between two pixels using the distance function $d: S \times S \rightarrow R$ (S is a set of pixels) that satisfies the following condition [15]:

$$0 = d(p,p) \leq d(p,q) + d(q,p) = 2d(p,q) \; for all \; p,q \in S \tag{7}$$

For example the Euclidean distance (Equation 8) satisfies the above condition (7). We have used the Euclidean distance in the implementation of the similarity measures.

$$d_e(p,q) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} \tag{8}$$

Average Distance to Boundary Difference (ADBD)

In the Average Distance to Boundary Difference (ADBD), we calculate the average distance to the boundary for each of the reference and test segmentations (with the help of the distance map of the segmentations) and take their difference (Equation 9).

$$ADBD = \left| ADB_{RS} - ADB_{SUT} \right| \tag{9}$$

Hausdorff and Baddeley's Distance (HD & BD)

Hausdorff's Distance (HD) [15] finds the maximum distance between the reference and test segmentations while Baddeley's Distance (BD) [19] takes the average of the difference in the distances to the reference and test segmentations over all the pixels in the image (U) taking away the noise-sensitivity inherent in HD [16].

$$HD(A,B) = \max\{\max_{p\in RS}\min_{q\in SUT} d(p,q), \max_{p\in SUT}\min_{q\in RS} d(p,q)\} \tag{10}$$

$$BD = \left[\frac{1}{|U|}\sum_{p\in U}\left|\min_{q\in SUT} d(p,q) - \min_{q\in RS} d(p,q)\right|^{z}\right]^{\frac{1}{z}} \quad z \geq 1 \tag{11}$$

In HD, each pixel in each set (in our case study, each set of pixels here represents the labeled region in a segmentation) is scanned to find the minimum distance between that pixel and the boundary of the other set; for each set, the pixel that is farthest to the boundary of the other set is chosen, and the distance between the two sets is the larger distance of the two chosen pixels in each set. In Figure 8, $\max_{p\in A}\min_{q\in B} d(p,q)$ is the length of the arrow pointing from pixel p, $\max_{p\in B}\min_{q\in A} d(p,q)$ is the length of the arrow pointing from pixel q and the maximum of these two lengths is the Hausdorff distance between sets A and B (i.e., the length of the arrow pointing from pixel p).

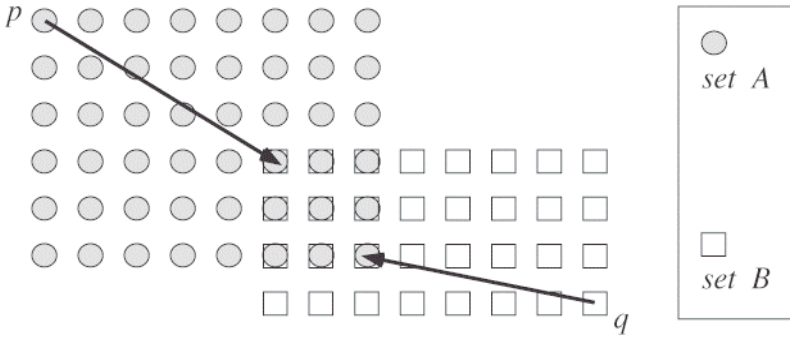


Figure 8 The Hausdorff distance is the distance from pixel p to the nearest common pixel between sets A and B

The parameter *z* in BD determines the relative importance of large localization errors i.e. a higher z penalizes large localization errors more than small localization errors. A localization error is the error initiated from distance between the boundary pixels of SUT and the RS. If z reaches infinity, BD tends towards HD [19].

Odet's Measures (ODI, UDI, SODI, SUDI)

Odet defines a few measures that aim more specifically at finding over-segmentation (the test segmentation has labeled pixels that are not labeled in the reference segmentation i.e. the over-segmented pixels (*OS* in Equation 12)) and under-segmentation (the test segmentation has not labeled pixels that are labeled in the reference segmentation i.e. the under-segmented pixels (*US* in Equation 13)). ODI and UDI respectively determine the extent of over-segmentation and under-segmentation by calculating the average distance between the over-segmentation (under-segmentation) boundary to the reference (test) segmentation boundary. *Bd* stands for boundary [20].

$$ODI = \frac{1}{|bd(OS)|}\sum_{p\in bd(OS)}\min_{q\in bd(RS)} d(p,q) \tag{12}$$

$$UDI = \frac{1}{|bd(US)|}\sum_{p\in bd(US)}\min_{q\in bd(SUT)} d(p,q) \tag{13}$$

SODI and SUDI are scalable versions of ODI and UDI introducing *dTH* and *n*. dTH is a normalization threshold that is set to the maximum perceived distance of the over-segmented and under-segmented pixels to the corresponding boundaries and *n* is a scaling factor similar to z in BD de-emphasizing smaller localization errors (when n>1) and putting a lot of emphasis on smaller localization errors (when 0<n<1) [20].

$$SODI = \frac{1}{|bd(OS)|} \sum_{p \in bd(OS)} \left(\frac{\min_{q \in bd(RS)} d(p,q)}{d_{TH}} \right)^n \tag{14}$$

$$SUDI = \frac{1}{|bd(US)|} \sum_{p \in bd(US)} \left(\frac{\min_{q \in bd(SUT)} d(p,q)}{d_{TH}} \right)^n \tag{15}$$

Pratt's Figure Of Merit (PFOM)

Pratt's Figure Of Merit (PFOM) [21], defined in Equation 16, is another widely used measure that has been empirically proven. PFOM only attains values greater than zero and less than or equal to one leading to higher similarity between the segmentations when its values reaches 1 where the two segmentations are considered identical with respect to PFOM.

$$PFOM = \frac{1}{\max\{|bd(SUT))|, |(bd(RS))|\}} \sum_{p \in bd(SUT)} \frac{1}{1 + \frac{1}{9} \min_{q \in bd(RS)} d^2(p,q)} \tag{16}$$

This measure is sensitive to over-segmentation and localization problems, but it does not take into account under-segmentation or shape errors. The insensitivity of PFOM to the pattern of pixel errors is seen in Figure 9 (taken from [23]]). If A is the true segmentation, B and C will result in the same value for PFOM as they both have two corners of over-segmented pixels (the top and right corners). PFOM has not been theoretically proven, but is widely used as an empirical measure [16].

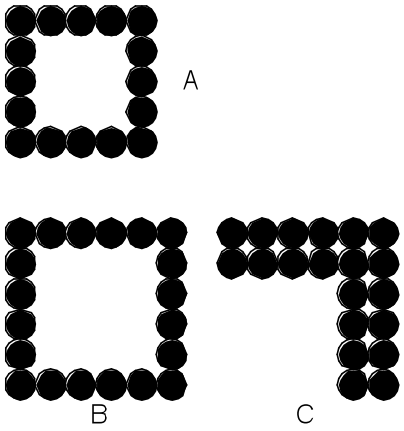


Figure 9 PFOM determines B and C as equally similar to A [23]

Peli and Malah's Measures (PMME and PMMSE)

Peli and Malah define PMME (Peli and Malah Mean Error) and PMMSE (Peli and Malah Squared Error) [22] that simply calculate the mean error (Equation 17) and mean squared error (Equation 18) of the distance between the boundary pixels of the segmentation under test and the boundary pixels of the reference segmentation. Unlike PFOM, these two measures are not normalized ranging between zero and infinity, larger values indicating more deviation between the segmentation under test and the true segmentation. If two segmentations are identical then the value of PMME and PMMSE will be zero.

$$PMME = \frac{1}{|bd(SUT)|} \sum_{p \in bd(SUT)} \left(\min_{q \in bd(RS)} d(p,q) \right) \tag{17}$$

$$PMMSE = \frac{1}{|bd(SUT)|} \sum_{p \in bd(SUT)} \left(\min_{q \in bd(TS)} d(p,q) \right)^2 \tag{18}$$

Root Mean Square Surface Distance (RMMSD)

RMMSD (Root Mean Square Surface Distance) [17] defined in Equation 19 takes the root mean square type average of the distances between the boundary (surface) pixels of

the segmentation under test and the reference segmentation over all the boundary pixels of both segmentations.

$$RMSSD = \sqrt{\frac{\sum_{p \in bd(SUT)} \left[\min_{q \in bd(RS)} d(p,q) \right]^2 + \sum_{q \in bd(RS)} \left[\min_{p \in bd(SUT)} d(q,p) \right]^2}{|bd(SUT)| + |bd(RS)|}} \quad (19)$$

Principal Axis Difference

The principal axes of an object (in our case the labeled region of the segmentation) are in the direction of the eigenvectors of the covariance matrix representing that object. The covariance matrix (Equation 20) represents the covariance of the coordinates (Equation 21), in this case 3D Cartesian coordinates. The covariance matrix is a symmetrical matrix because $c_{ij} = c_{ji}\ \{i, j = 1,2,3\}$.

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \quad (20)$$

$$c_{ij} = \sum_{i,j=\{1,2,3\}} x_i x_j \quad (21)$$

If we consider C as a transformation (a matrix that in the general case rotates and shifts the vectors that are applied to it), the eigenvectors are the vectors that do not rotate when the transformation is applied to them (22).

$$CV_i = \lambda_i V_i \quad (22)$$

In Equation 22, V_i (i>=1) are the eigenvectors of the covariance matrix and λ_i (i>=1) are the associated eigenvalues of the eigenvectors. In the coordinate system represented by the eigenvectors (principal axes), the covariance matrix, shown by C' in this coordinate system will have the form of a diagonal matrix (Equation 23) (a matrix that only has nonzero elements on the diagonal) where the eigenvalues of the covariance matrix are on the diagonal of the matrix.

$$C' = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{23}$$

As you see in the coordinate system represented by the principal axes, the covariances of the coordinates are zero with respect to each other, which means that the object is geometrically symmetrical around the principal axes (Figure 10).

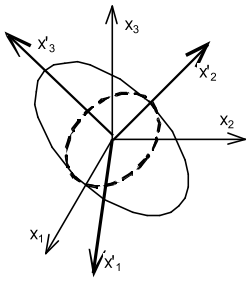


Figure 10 Object is geometrically symmetrical around the principal axes (x'_1, x'_2, x'_3)

A closer look at C' shows that the eigenvalues are in fact the variances of the coordinates of the object points (segmentation pixels) in the principal coordinate system. We define the Principal Axis Difference comparison measure as follows:

$$PAD = abs(abs(max(\text{`}_{i,TS})) - abs(max(\text{`}_{i,SUT}))) \quad i = \{1,2,3\}$$

PAD is calculating the difference of the largest variance of the pixels along the principal axes of the true segmentation and the segmentation under test. PAD solely concentrates on the shape of the segmentations and thus if the shape of the segmentation under test is the same as the true segmentation PAD would depict no difference between the segmentations, this is while the labeling in the segmentation under test may be located at a different location in the image from the expected location, making it an incorrect segmentation.

Appendix B Tool Support Architecture

The structural components in the design of the automated oracle are shown in Figure 11. The outputs (in our case the segmentations) from version V_i ($1 \le i \le n$ where n is the total number of versions of the Software Under Test (SUT) until its depicted as *valid*) of the software under test (in our case the software under test is the segmentation algorithm) are fed into the oracle and the oracle determines the correctness of the outputs.

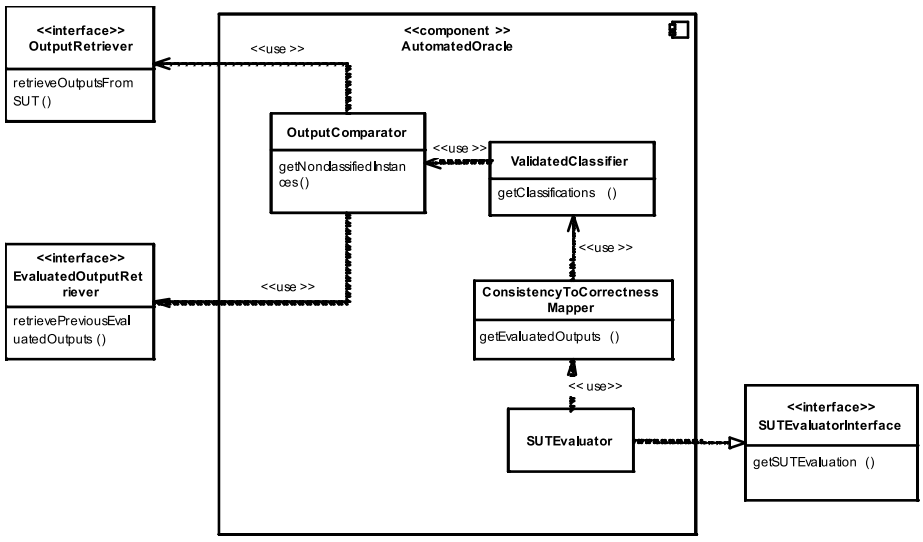


Figure 11 Structural components of the automated oracle

To do so, the *OutputComparator* class is used to retrieve the outputs from the current version of the SUT via the OutputRetriever interface. Each output O_{ji} ($1 \le j \le total\_num\_subjects, 1 \le i \le total\_num\_software\_versions$) produced from input I_j (the input in our case is the 3D volume CT-scan of the subject) is first compared to the outputs that have been produced from the same input I_j using previous versions of the software under test (version < V_i). The previously evaluated outputs have been retrieved via the *EvaluatedOutputRetriever* interface which may be provided by an external database used for storing the previously evaluated outputs. The output that is

being tested will be referred to as the Output Under Test (OUT). Table B.1 shows the contents stored in the external database more vividly.

Table 12 Table of all outputs produced by the previous versions of the SUT (version < Vi) and their corresponding evaluations

Input	**SUT V_1**	**SUT V_2**	**…**	**SUT $V_{(i-1)}$**
I_1	O_{11}\Correct	O_{12}\Correct		$O_{1(i-1)}$\Correct
I_2	O_{21}\Incorrect	O_{22}\Incorrect		$O_{2(i-1)}$\Incorrect
I_3	O_{31}\Correct	O_{32}\Incorrect		$O_{3(i-1)}$\Correct
I_4	O_{41}\Incorrect	O_{42}\Incorrect		$O_{4(i-1)}$\Correct
…	…	…	…	…

The *OutputComparator* uses the similarity measures defined in Appendix A to compare O_{ji} (OUT) with the outputs: $O_{j(i-1)}$, $O_{j(i-2)}$, …, O_{j2}, O_{j1} and for each comparison (for example if the current version of the SUT is V_{10}, one comparison could be between $O_{2,10}$ (OUT) and the output produced by the 8th version of the SUT i.e. $O_{2,8}$; the input associated with both outputs is I_2) will produce a tuple of the comparison measure calculations (m_1, m_2, …, m_z) where z is the number of defined comparison measures. The *ValidatedClassifier* (a classifier that has been validated by Activity G) retrieves these tuples which serve as the non-classified instances and classifies them producing tuples in the form (m_1, m_2, …, m_z, consistent\in-consistent) which now contains the classification. The *Consistency to Correctness Mapper* class uses the information provided in Table B.1 and the logic explained in Table 2 to map the consistency classifications to correctness classifications. Table B.2 shows an example on how this is done.

Table 13 An example showing how the *Consistency to Correctness Mapper* module obtains the correctness of the OUT

Output that OUT has been compared to	Correctness Classification of Previous Outputs	Consistency Classification	Correctness Classification of OUT
$O_{9.6}$	Correct	Consistent	Correct
$O_{9.5}$	Incorrect	Inconsistent	Unknown
$O_{9.4}$	Correct	Inconsistent	Incorrect
$O_{9.3}$	Correct	Consistent	Correct
$O_{9.2}$	Incorrect	Consistent	Incorrect
$O_{9,1}$	Correct	Consistent	Correct

In this example, OUT is the output produced from the 7^{th} version (V_7) of the segmentation algorithm from input 9 (I_9) i.e. output $O_{9,7}$. The *Consistency to Correctness Mapper* uses a policy to make the final decision as to whether OUT is correct or not. This policy could be taking a majority vote (in this example, OUT has been determined to be correct in 3 instances, while it has been deemed incorrect in 2 instances, so OUT is evaluated to be correct) or just taking the evaluation made using the comparison with the last version of the SUT. The assumption here is that the last version of the SUT is the superior version of all the previous versions. We cannot automatically evaluate the correctness if all the outputs from the previous versions of the SUT are evaluated to be incorrect and the classifier has classified OUT to be inconsistent with all of them (the unknown case in Table 4.1) or in the case where there is an equal number of corrects and in-corrects (this is only an issue when using the majority vote policy). The evaluated outputs (outputs classified as correct\in-correct by the *ValidatedClassifier*) are retrieved

by the *SUTEvaluator* class that uses some defined criteria to determine whether we have enough correct outputs produced by the current version of the software under test to evaluate this version as a *valid* version and thus the final version of the software under test. The *SUTEvaluator* class has to provide an interface for the test monitor (the module that runs the test cases) to retrieve the oracle's evaluation on the current SUT. This is provided through the *SUTEvaluatorInterface*.

Appendix C Cardiac Left Ventricle Segmentation Examples

We show here some figures showing the left heart ventricle segmentations. Each figure is a slice of the CT-Scan of a different patient's heart where the left ventricle is segmented by labeling the representative pixels with green using some version of the segmentation algorithm. Figure 12 shows a correct segmentation while Figures 13, 14 and 15 show incorrect segmentations. Figure 13 shows a typical over-segmentation scenario (the most common problem in this case study) where the left atrium is also labeled. In Figure 14 the segmentation algorithm has missed the left ventricle and labeled the right ventricle instead. Under-segmentation is seen in Figure 15 where the left heart ventricle is not completely labeled. These are some examples of incorrect segmentations in this case study. More detailed discussion on the manual evaluation of each segmentation is out of the context of this research work and requires background in the heart anatomy. Also note that these figures only outline the segmented region in one slice of the heart. In order to evaluate the correctness of a segmentation, all the CT-Scan slices of the heart have to be looked at by an expert where an overall assessment is then given as to whether the segmentation is correct or not. Minor discrepancies in a few slices may be ignored by the expert. As explained earlier the tolerance level for accepting a segmentation as correct may be different for each expert.

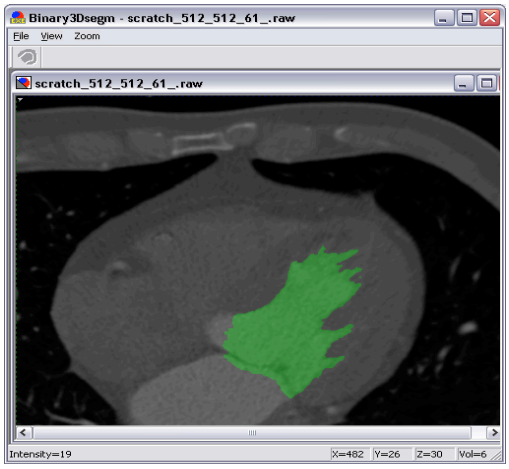


Figure 12 Correct Left Heart Ventricle Segmentation

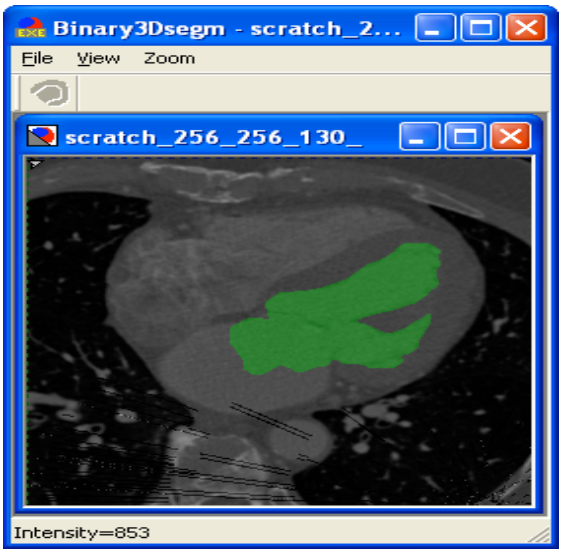


Figure 13 Incorrect Segmentation (over-segmentation to the left atrium)

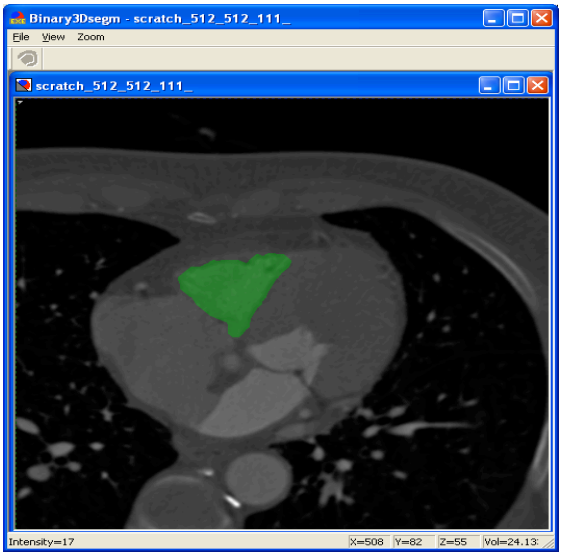


Figure 14 Incorrect Segmentation (Segmenting an incorrect region i.e. the right ventricle)

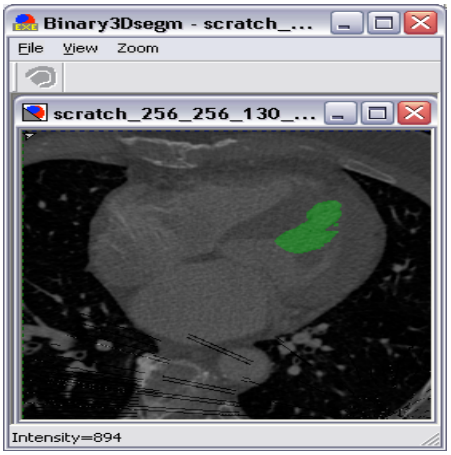


Figure 15 Incorrect Segmentation (under-segmentation)

Appendix D Experimentation Tools

Three main tools have been used in this case study:

1) Left Heart Ventricle Segmentation tool (s): This tool was created by Siemens Corporate Research experts. Each version of this tool implements a version of the image segmentation algorithm. To obtain the segmentations for each version of the image segmentation algorithm the tool had to be run for all of the test cases.

2) MATLAB: We used MATLAB to implement all the comparison measures feeding the segmentations obtained from the left heart ventricle segmentation tool to the comparison measures. In order to calculate the distance maps of the segmentations that are required for the implementation of the geometrical measures, we have used the MATLAB library function *bwdist* with the Euclidean metric option. This function uses the implementation in [27] to implement the distance transform. To find the boundary of the segmentations, we simply use the *gradient* function as the gradient at any pixel of a non-boundary pixel is zero (all neighbor pixels are labeled as 1) while it is nonzero in some direction for boundary pixels (a zero-labeled neighbor pixel exists in some direction).

5/13/09 9:36 PM
Comment: Could we reference here?

3) WEKA: The Waikato Environment for Knowledge Analysis tool was used for obtaining the entire machine learning results. The raw training data was obtained from the output of the MATLAB implemented comparison measures. This is a Java-based tool and is widely used in the research community. It contains the implementation of most of the various data mining techniques and provides an easy to use environment for experimentation.