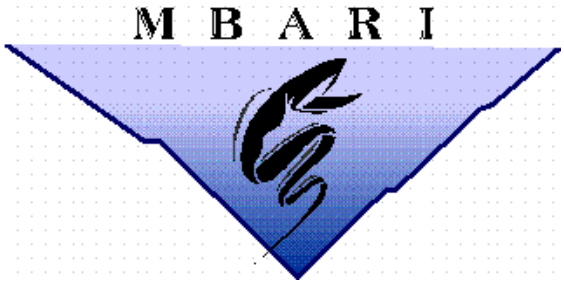
The MBARI Automated Video Event Detection Project



AVED Classifier User Interface

Software Requirements Specification

Version 0.4

Revision History

0.1 10-Aug-05 - DC initial draft
0.2 August 18, 2005 DC revision based on meeting with Brian and Andrew
0.3 September 1, 2005 DC minor revision to based on email between Brian and Andrew
0.4 August 28, 2006 DC modified overview, added use cases and generally modified requirements to convey meaning more clearly

Classifier Overview

The core classifier code is written in matlab. It is based on a test program given to us by student Marc'Aurelio Ranzato from Caltech that has been slightly modified by MBARI. A JNI interface has been written to access the classifier library from Java.

There are three main sequences in the AVED classifier: creating a new class or classes, then training and testing the class or classes.

Classification learning is the process of learning a discrete-valued function from discrete inputs (images in this case).

The foundation of the classifier is a collection of images files. Each image file collection constitutes a "class" which typically represents a single animal in its various sizes and orientations, .e.g. a Rathbunaster class, Poeobius class, etc. By itself, each class is not useful until the classifier is trained to recognize them. Essentially, the "class" image files are "training" data the classifier uses to learn to recognize objects, e.g. Rathbunaster, in unknown images.

Once a class is created, an intermediate step called collection is needed that must be done before training and testing. In the collection step, image features are extracted from each image file, then some intermediate calculations are done on those features, and the results are stored in a matlab file for use in the testing and training phases.

Once a class is created and collected, training can take place. The process of learning or "training" is done. This "training" step is the root of the classifier intelligence. It uses an algorithm Marc'Aurelio Ranzato from Caltech devised to learn how to recognize biological particles in images.

To test how well a class performs as a discriminate for a given animal, validation of the class is done. Validation takes a 10% random sample of images from a known class and attempts to recognize them against the class it came from. This step essentially tests the class against itself to give some indication of how good the class is as a discriminator when presented with nothing else.

Once validation is done, to use the classifier for actual work, a training set must be generated. Once a training set is generated, this training set can then be used to recognize unknown images.

Classifier Use Scenarios

Refining to create the best class

This classifier is only as good as its training data. From experience, we've found creating a good class is very time consuming and somewhat of an art form. It does require some level of expertise to create a "good" class of images. One scenario is to refine the class data through repeated addition or removal of class image files, followed by validation to see how the class image files changes the results. For example, one scenario is to run validation, followed by removal of the images that were not recognized correctly, then re-running the validation test. Sometimes there are images in the class that are simply poor representations, or, worse are "junk" that can be eliminated this way. Repeated iterations of the aforementioned steps can take days or weeks to create a well defined class.

Creating subclasses or renaming classes

As volumes of images data are compiled, another scenario is to create subclasses from classes, or renaming classes that were incorrectly categorized. This scenario requires some thought about file handling and maintaining the history of the class data.

Creating user-defined training sets

A training set is one or more classes. For example, a "Benthic" training set might include Rathbunaster-californicus, Leukothele, "Benthic Junk". These training sets are targeted to a particular dive, or to what a particular user is looking for in a sequence of images. A user may want to define these training sets for their preferences, and also make them available for everyone to use.

Classifying large volumes of unknown images

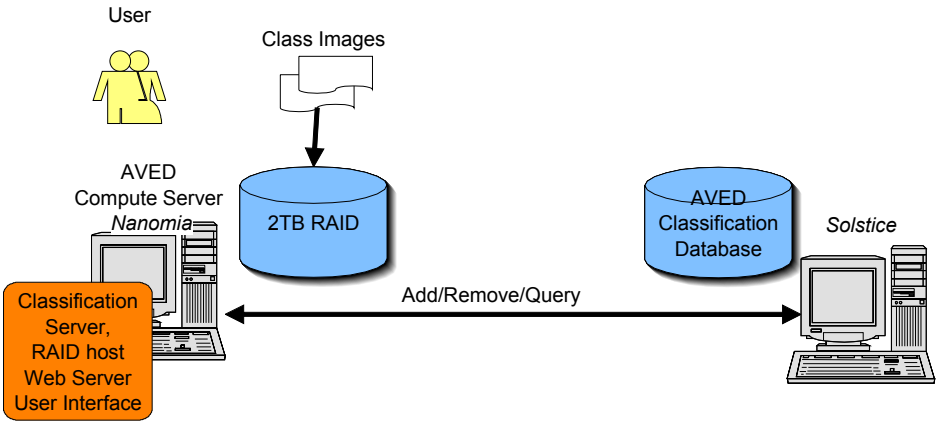
Once the training set is generated, it can be used for actual work. This could potentially be many thousands of images. This use scenario is not as well defined as we'd like simply because this is simply uncharted territory. However, it's safe to assume the goal is to easily 1) input unknown images 2) setup the system to recognize these images, and 3) easily view the resulting classification.

Classifier System Components

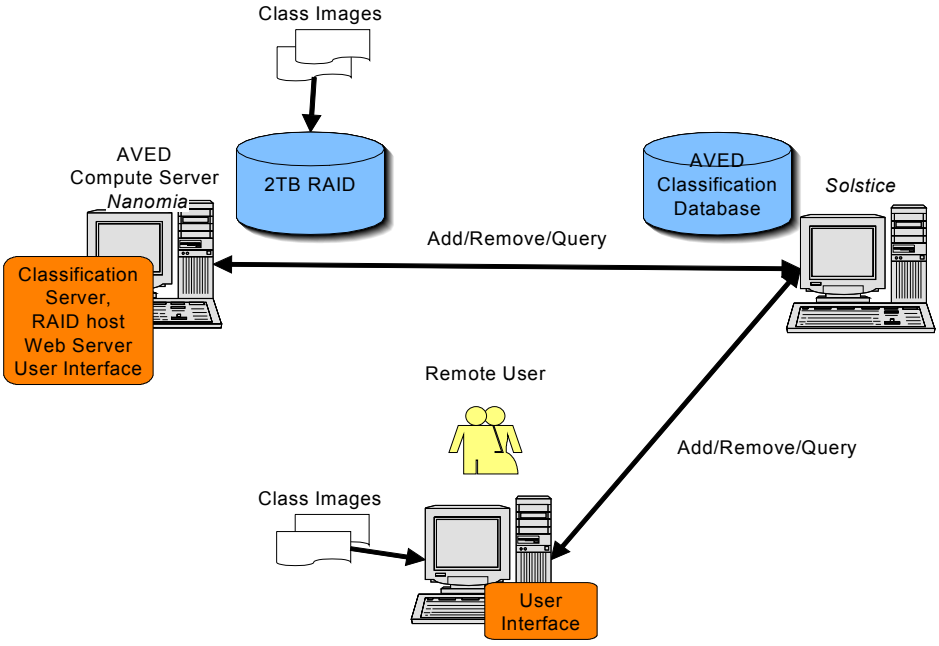
The AVED classification metadata is stored on a database hosted by *Solstice*. Physical class image files and matlab-specific data are hosted on the 2TB RAID connected to the *Nanomia* server.

Propose work in two phases:

2005 propose stand-alone application only works on *Nanomia*.



2006 propose remote interface that can work on remote machines.



2005 - Classifier standalone user interface requirements

Scope

This document covers only requirements for 2005 work.

General design

The software is to be designed and implemented in modern, maintainable technology, such as an object oriented programming language (OOP), which is supported over the useful life of the system. Software systems are to be well documented and to include written software test and maintenance procedures. There should also be maximum use of industry proven standards and technology and commercial off the shelf software (COTS).

Definitions

A class shall be defined as a collection of related images that represent a particular animal, collection of animals, or objects.

A training set shall be defined as more than one class.

Previewing a class

The interface shall include an alphabetical list that displays all available training classes. This list would include classes that may correspond to concepts in the VARS Knowledge Base (e.g. Poeobius-merses), or those that are more generic (e.g. “Benthic undefined” or “Benthic other”, which might represent a benthic animals that are not already defined by a class.)

The preview of a class shall include one representative image and a list of filenames or URL references for each image. *(Would be nice to have thumbnail of all class images but not necessary).*

Creating a new class

The interface shall allow the user to create a class that corresponds to a concept name in the VARS Knowledge Base, or information outside the knowledge base.
Rationale: Classes will typically come from the VARS KB, but are not limited to the Knowledge Base. For example, in our collaborations we use video from areas outside Monterey, with concepts that may not exist in our KB and our testing may require grouping animals in some way that does not correspond to a concept.

The user shall provide a general description of the class and an example image at the creation of a new class.

The name of the class shall be formatted with the concept name appended by “-” and a unique database ID and version number for the class.

The interface shall allow the user to select a local directory or files within a local directory to import images that make up the class.

Once a class is locked, the user cannot update the class collection and must create a new class.

Deleting entire classes or individual images in a class

The way the classifier is designed requires all of the class images be organized by directory on *Nanomia*.

The interface shall allow a user to delete an entire class. This shall not be password protected, so anyone can delete a class.

The interface shall allow the user to tag individual images for deletion in a given class.

Note, even though images associated with a given class are deleted, they will still physically reside on Nanomia and have a database reference if the image is cross-referenced by other classes.

Collecting a class

Once a class is created, an intermediate step called collection is needed that must be done before training and testing. In the collection step, image features are extracted from each image file, then some intermediate calculations are done on those features, and the results are stored in a matlab file for use in the testing and training phases. This step can be hidden from the user, but this step collection takes the longest time to execute and should be run after adding or removing images from a given class, so some visual feedback is needed for the user and until this is run, training cannot start.

The interface shall allow one or more classes to be collected.

An indicator to display class collection is commencing shall be displayed. *Rationale: this take a long time to execute in Matlab, so the user needs some feedback on this.*

While collection is commencing the interface shall tag the classes as not usable and not allow other operations that require the classes to begin.

Validating a class

The intention of validating is to use this periodically to give feedback on performance, to make sure bogus images are not added to classes. This is to be used as a performance tool to indicate how too a class is as a discriminator when tested against itself and nothing else.

The classification software shall include the ability to validate an individual class or collection of classes.

A minimum of TBD images shall be required to validate a class. For class with less than TBD images, the validation function shall be disabled. *Rationale: the current classifier crashes when testing a class with a small number of images. The minimum number of images seems to depend on the size of the animals and/or amount of feature detail in the animal. Classes that represent larger animals like the Rathbunaster require fewer images than the small Poeobius. This may be fixed in later versions of the classifier library, but for now this is an outstanding problem.*

The results of the class validation shall display both percent correct and exact numbers of classification results, for example. 95% correct classification or 3812 out of 4013 correctly classified as Rathbunaster.

Note, in the classifier library, the validation function automatically generates a training set for the class. This operation is handled by the classifier matlab library and is hidden from the user. If changes are made to one or more classes in the training set (e.g. by adding/deleting), regeneration of the training set is required,

Generating a training set

To generate the training set, the interface shall allow the user to select from available class collections in an alphabetical drop-down or scrolling type list.

Recognizing unknown images by testing against a training set

The point in classification is to reliably recognize images not seen before, given a discrete function that can calculates what they are based on the training data. From the user's perspective, the resulting output of the classifier should minimally report 1) what class was assigned, and 2) with what confidence was it assigned.

The interface shall allow a user to first select an available training set to use.

Once a training set is selected, the interface shall allow the user to select a local directory with images to recognize. The directory listing shall show images in PNG or JPG format.

Once testing is started, a progress bar shall display the status of the testing.

A summary of results of the testing shall display both overall percent and exact numbers of classification results, for example. 15% or 150/1000 classified Rathbunaster, 10% or 100/100 as Unknown, and 75% 750/1000 as Other.

Details results shall be displayed in a grid format with the headers: file name, probability and class assignment. *(Would be nice to have a thumbnail of test input images too, but not required).*

User profiles

The user interface shall store all current settings into a user profile to be reloaded on startup.

Directory Defaults

The interface shall allow the user to select default locations for testing and training results.

Saving results

The interface shall allow the user to save the training and testing results to XML or comma delimited format.

If a file is not specified to save the results to, the UI shall prompt the user to select a file.