An open-source platform for underwater image and video analytics

Matthew Dawkins[1], Linus Sherrill[1], Keith Fieldhouse[1], Anthony Hoogs[1], Benjamin Richards[2], David Zhang[3], Lakshman Prasad[4], Kresimir Williams[5], Nathan Lauffenburger[5] and Gaoang Wang[6]

[1]Kitware, Inc., {matt.dawkins, linus.sherrill, keith.fieldhouse, anthony.hoogs}@kitware.com
[2]NOAA Pacific Islands Fisheries Science Center, benjamin.richards@noaa.gov
[3]SRI International, david.zhang@sri.com
[4]Los Alamos National Laboratory, prasad@lanl.gov
[5]NOAA Alaska Fisheries Science Center, {kresimir.williams,nathan.lauffenburger}@noaa.gov
[6]University of Washington, gaoang@uw.edu

Abstract

Global fisheries and the future of sustainable seafood are predicated on healthy populations of various species of fish and shellfish. Recent developments in the collection of large-volume optical data by autonomous underwater vehicles (AUVs), stationary camera arrays, and towed vehicles has made it possible for fishery scientists to generate species-specific, size-structured abundance estimates for different species of marine organisms via imagery. The immense volume of data collected by such devices quickly exceeds manual processing capacity and creates a strong need for automatic image analysis. This paper presents an open-source computer vision software platform designed to integrate common image and video analytics, such as stereo calibration, object detection and object classification, into a sequential data processing pipeline that is easy to program, multi-threaded, and generic. The system provides a cross-language common interface for each of these components, multiple implementations of each, as well as unified methods for evaluating and visualizing the results of different methods for accomplishing the same task.

1. Introduction

The Magnuson-Stevens Fishery Conservation and Management Act [1], the framework for fisheries management in the United States, requires that managed fish stocks undergo periodic assessment to determine if they are overfished or are experiencing overfishing. A basic stock assessment requires data on fishery abundance, biology (e.g. age, growth, fecundity), and catch. While demands to continually improve stock assessments are high, a lack of ade-
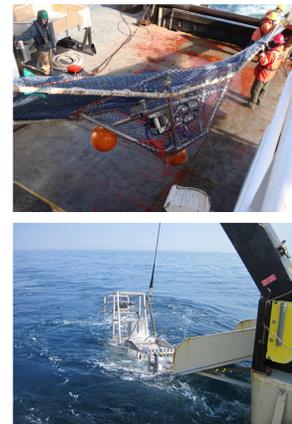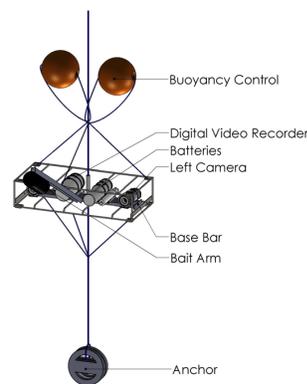

Figure 1. A few examples of devices used to collect imagery underwater, including towed vehicles (top right [25], bottom right [13]) and a stationary camera array (left).

quate input data remains an important impediment to their accuracy, precision, and credibility remains [12]. Recent developments in low-cost autonomous underwater vehicles (AUVs), stationary camera arrays, and towed vehicles has made it possible for fishery scientists to begin using imagery and video to generate species-specific, size-structured abundance estimates for different species of marine organisms (see Figure 1). To this end, government agencies and universities are increasingly employing camera-based surveys for abundance estimation [5]. However, the volume of optical data produced quickly exceeds the capabilities of human analysis. To move into operational use, automated video analysis solutions are needed to extract species-specific, size-structured abundance measures from optical data streams.
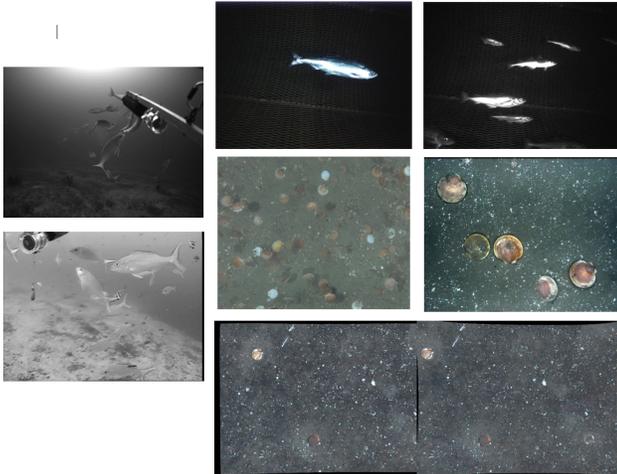
Figure 2. Example optical data from the devices shown in Figure 1.

This paper presents the Video and Image Analytics for Marine Environments (VIAME[12]) open-source computer vision software platform for automating the image analysis process, for applications in marine imagery or any other type of video analytics. The system provides a common interface for several algorithm stages (stereo matching, object detection, object recognition), multiple implementations of each, as well as unified methods for evaluating different algorithms for accomplishing the same task. The common open-source framework facilitates the development of additional image analysis modules and pipelines through continuing collaboration within the image analysis and fisheries science communities.

The platform can be divided into three core components: the pipeline processing framework and infrastructure; image processing elements that fit into the framework; and auxiliary tools outside the streaming framework that provide training, GUIs and evaluation. The pipeline subsystem allows image processing elements to be implemented in the most popular languages used for computer vision, including C, C++, Python, and Matlab. A graphical interface is provided in the framework for visualizing individual object detections, making new annotations, and filtering detections based on classification values. There are two separate evaluation tools included, one for generating basic statistics over detections compared with groundtruth (detection rates, specificity, false alarm rate, etc.) and a second for generating receiver operating characteristic (ROC) curves for detections which contain associated category probabilities. Both the evaluation and GUI tools work with either single frame object detections, or spatiotemporal object tracks.

The platform has been released as open-source software on github. Multiple object detectors tuned to different types

---

[1]http://www.viametoolkit.org
[2]https://github.com/Kitware/VIAME

of marine imagery have been integrated, with results shown in Section 4. The provided evaluation elements are also available, with example computed metrics and ROC curves for select detection problems shown in Section 6. It is anticipated that the marine research community will continue to adopt the framework, enabling collaboration and re-use of modules across different research groups working on related problems. Although the framework is primarily being used for marine imagery at present, it is based on a more generic pipelined vision processing system that applies to video analytics in any domain. It is hoped that the general vision community would find the framework useful and that a vibrant open-source community will develop around the platform.

2. Related Work

There are currently many open-source computer vision packages available on the internet. Many of the most popular repositories, such as OpenCV [4], Caffe [15], and Theano [2], typically contain lower-level functions for image processing operations but lack full end to end processing pipelines that combine multiple stages of processing in a streaming architecture. Our pipeline framework also allows input and output endcaps to be easily reconfigured; for example, core algorithmic pipelines can get input data from a UDP streaming source instead of a video or image list reader via a simple config file change. Pipeline architectures and system settings can be changed easily in the same config file with recompilation. Most current computer vision toolkits also lack standard input and output formats for storing output products, such as object detections, which are contained within our framework. These standard outputs are designed to work with the included GUI and evaluation tools.

At the other end of the spectrum, there are several open-source media streaming frameworks such as GStreamer [23] and FFmpeg [3]. These libraries were designed, however, primarily for multimedia applications and not explicitly for image and video processing. One of the key elements of our pipeline framework is an individual process
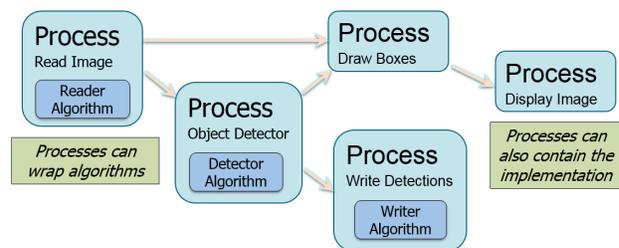

Figure 3. Simple pipeline abstraction containing an image reader, an arbitrary object detector, an output display, and an output writer process.
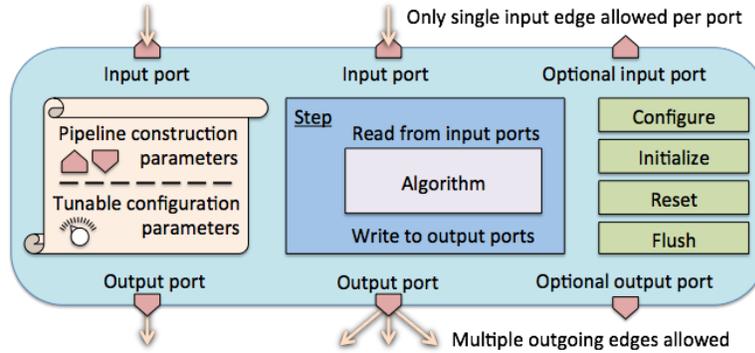
Figure 4. An overview of an individual process. All items, such as the number of ports, types of ports, main step function, and green auxiliary functions are optionally defined by process implementers. The algorithm itself can either live within the process, or be a wrapper around an external repository or binary.

class definition, which contains definable subroutines for loading model files at the beginning of processing, in addition to defining what actions to perform when new data is received (typically a new image in a video sequence, or a new metadata packet). These useful elements do not exist in GStreamer and FFmpeg. The platform also contains common data types for passing between process nodes, and base classes to simplify defining specific types of algorithm processes, e.g. separate base classes for object detectors and stereo correspondence algorithms. Finally, our framework allows pipelines to be defined using a directed graph, which is more general than the simple linear pipelines of GStreamer and FFmpeg. The platform manages the complexity of inter-process synchronization, communications and dependencies, which can be tricky and problematic to program correctly in other frameworks.

In the marine science environment, computer vision has yet to reach its full potential. There is some work in techniques for automated detection, identification, measurement, tracking, and counting fish in underwater optical data streams [9, 6, 7, 8, 14, 20, 21]. However, few of these systems are fully automated, with all of the functions required to produce highly successful and accurate results [22]. Consequently there is little operational use of automated analysis. Furthermore, researchers and algorithms tend to remain specialized in a specific type of fish and/or collection system, whereas there may be significant gain in efficiently experimenting across domains. The proposed framework facilitates such collaborations and experiments by making it easy and cheap to share algorithms, implementations, and data.

3. Architecture

Two key features provided by this software, VIAME, are common data types and a stream processing toolkit called Sprokit. Sprokit allows the users to configure multiple pro-

cessing elements in a graphical pipeline, such as in Figure 3, while the common data types are used as "edges" in the graph. Individual nodes within this pipeline are implemented within plugin modules, both for compartmentalization and to allow modules to be easily shared.

3.1. Pipeline Framework

Sprokit implements a processes and pipes data flow application structure, and an easy way to describe and run these type of applications. There are several advantages to implementing this type of application. The main benefit for VIAME is easily interchangeable modules enabled by using common data types. For example, if one team produces a color correction algorithm, it can then be easily integrated into anothers applications or pipelines.

The main unit of construction of these pipelines is the process. A process is a class that derives from the Sprokit base process class and implements a specific operation. Both C++ and python implementations of the base class are supported, key portions of a process are shown in Figure 4. The main feature of a process are its input and output ports that are used to transfer data. The typical process lifecycle starts with creating a new object of the desired process type. This new process is then called on its configure method and passed the set of configuration items for it to use. The process uses this configuration data to establish operating parameters. In the case of algorithm wrapping processes, these configuration options will specify which implementation to use and how it should be configured. After all processes are configured, the pipeline is created by connecting process outputs to process inputs. The process is now ready to start processing and its step method is called where it reads from the input ports, processes the data and puts the results on the output ports.

Processes are designed to implement a single well defined operation so that a solution can be created by connecting these elements. Having a fine-grained approach to

breaking an overall application into smaller processes promotes reuse and sharing of the processing elements. In addition, it improves the ability to parallelize the solution.

3.2. Plugin Architecure

The VIAME platform is built on the concept of dynamically loadable plugins. A plugin is physically a shared library that contains an implementation of software components. The VIAME toolkit supports two types of plugins, algorithms and processes. These plugins are discovered at run time and added to their respective internal registries. The

```
# Process definitions and configs
#
process input
  :: frame_list_input
  :image_list_file     input_files.txt
  :frame_time          0.3333
  :image_reader:type   ocv

process detector
  :: image_object_detector
  :detector:type       ex_fish_detector
  :detector:model1     model_file.xml
  :detector:threshold  0.20

process draw
  :: draw_detected_object_boxes
  :default_line_thickness 3

process disp
  :: view_image
  :annotate_image      true
  :pause_time          2.0
  :title               VIAME images

# Global pipeline configs
#
config _pipeline:_edge
      :capacity        5

# Connections between processes
#
connect from input.image
        to    detector.image

connect from detector.detected_object_set
        to    draw.detected_object_set
connect from input.image
        to    draw.image

connect from input.timestamp
        to    disp.timestamp
connect from draw.image
        to    disp.image
```

Table 1. Example simple detector pipeline config file.

locations searched for plugins can be easily customized for any desired installation layout. The advantages from a software engineering perspective are reduced coupling between software components. For example, a plugin could be built that uses OpenCV without introducing that dependency into the core VIAME libraries. After a plugin is created (algorithm or process), it can be individually distributed and used by others without requiring any new source code or rebuilding steps.

Algorithms and processes are structured in a polymorphic class hierarchy. The algorithm base class is defined in VIAME for all basic operation types, such as image filters, object detectors, and detection refining. Instantiating the concrete implementation of an algorithm is handled by a class factory mechanism built into VIAME and controlled by user supplied configuration entries. These config entries specify the implementation type to use in addition to other implementation related parameters. Abstracting the algorithm details into a configuration file makes it easy to utilize different implementations without requiring programming expertise to modify the source code. Algorithm configurations can be directly created and modified by subject matter experts or, in the future, by using GUI tools.

To meet the need of the research community, algorithms can be implemented in C++, Python or Matlab. The C++ implementations are implemented in a class derived from the abstract base class. Matlab implementations are a set of Matlab script files that implement the interface mediated by a C++ to Matlab adapter.

Algorithms can be instantiated and used directly in a program, or within a Sprokit process. VIAME supplies a set of processes that support base abstract algorithm definitions. These can be considered algorithm wrapper processes which instantiate the configured algorithm then pass the process inputs to the algorithm. Algorithm outputs are passed downstream to the next process. This approach provides algorithm level agility along with functional or topological agility.

3.3. Basic Processes and Pipelines

In addition to processes containing image processing algorithms, there are a number of utility processes in the system for performing tasks such as reading/writing object detections and imagery, drawing detections on images, displaying detections in a simple GUI, and others. An example pipeline is elaborated in Table 1 which creates an object detector alongside a few helper processes. Contained in this file is the definition of all processes in the pipeline, the dataflow between processes, and any configuration values that each process requires.
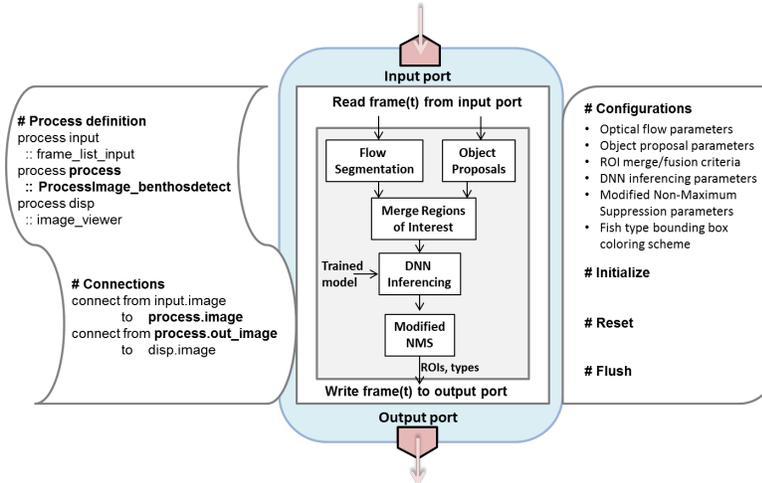
Figure 5. BenthosDetect as a plugin module in VIAME. Left: pipeline configuration. Middle: BenthosDetect flow diagram. Right: configuration parameters are initialized in a user configuration file and loaded up by VIAME at run time.

3.4. Third Party Library Support and Build System

Cross-platform build support for most operating systems (Windows, Mac, Linux) is provided via using CMake [17] instead of only a single-platform build system. Many leading open-source vision libraries involving C/C++ also currently use CMake [4, 15]. Within the platform are multiple enable flags to turn on support for different third party libraries, such as OpenCV, Python and Matlab. Every individual plugin in the system also contains a corresponding enable flag, in order to compartmentalize code and allow users to only build what they need. When a certain enable flag is set, all dependencies for the plugin are also automatically turned on and built internally, in order to use as little system packages as possible and simultaneously require few dependencies when building the platform on a fresh system. Alternatively this feature can be turned off, if developers wish to build or install all dependency packages themselves externally.

4. Example Algorithm Modules

A number of initial algorithm modules have either been implemented or wrapped within the platform. There are a number of ways to accomplish this based on whether or not the submodule is hosted externally on its own website, or if all of its core code rests within VIAME. Some modules are specific to individual applications, though others are more general and can be retrained to solve novel problems. A few select examples are detailed in the following sections.

4.1. BenthosDetect

BenthosDetect detects and identifies fish and benthic organisms that live in and on the bottom of the ocean floor. The BenthosDetect module is its own loadable plugin in the VIAME system. The core functions of the module were written in C++ and wrapped with python by boost-python.

BenthosDetect generates regions of interest from an optical flow segmentation [26] and object proposals (e.g., selective search [24]), as shown in Figure 5. Object proposals are based on local spatial features and flow segmentation group

Figure 6. Types of sea creatures used in training: background: 0, black eelpout:1, crab:2, longnose skate:3, north pacific hakefish:4, rexsole:5, rockfish:6, sea anemone:7, seasnail:8, seaurchin:9, starfish:10, sunflowerstar:11.

Figure 7. Example of benthic organisms detected by BenthosDetect. Each region of interest is classified by a pretrained network and color bounding boxes are used to show types. Green boxes indicate non-organism.
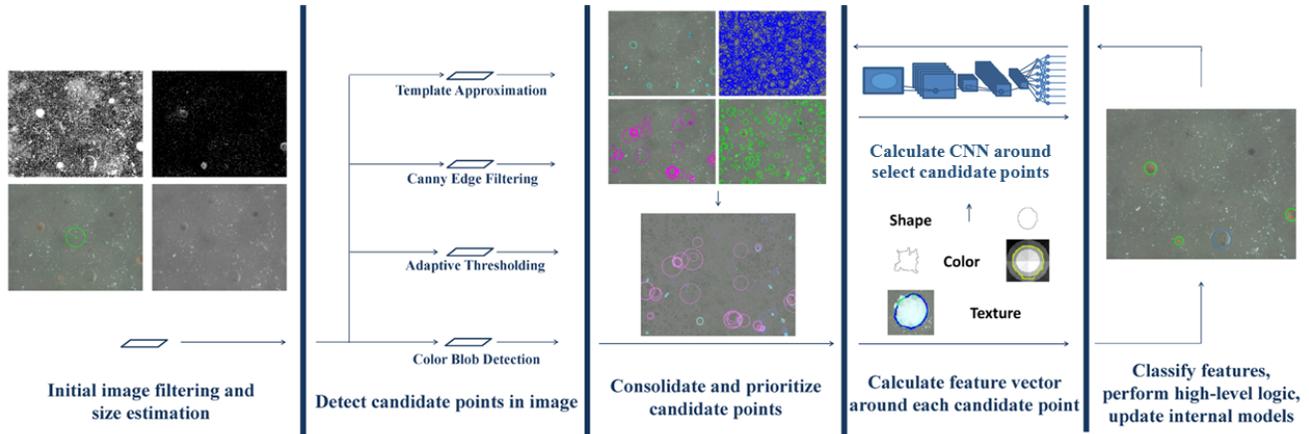
Figure 8. ScallopTK detector architecture. Candidate detection locations are extracted from the input image by four techniques, which are then classified according to a combination of handcrafted and automatically learned features.

features based on local optical flow magnitude and orientations. Each region is processed by a pretrained deep convolutional neural network (CNN). The best score among all types determines the type of fish detected for this ROI. The modified Non-Maximum Suppression removes overlapped regions that have less score [26]. The outputs are the detected bounding boxes and corresponding types of sea creatures per frame.

The benthic organisms used in the training were annotated based on data collected at the Monterey Bay Aquarium Research Institute. 12 types of sea creatures (˜12,000 samples) were trained from an annotated 1/2 hour video with no data augmentation. The classifier was tested on annotated ROI images from a separate section of video and tested as shown in Figure 6. The macro-averaging performance of detection from the ROC is 0.70 from the dataset. An example of detection on a frame is shown in Figure 7.

4.2. ScallopFinder

ScallopFinder is an algorithm being developed that is based on exploiting geometric and spectral properties of scallops. It is currently prototyped in Matlab and does not require training. Edge pixels from a canny edge detector are used to hypothesize circles by finding best fits for edge

Figure 9. Detections of adult scallops, baby scallops, and sand dollars by ScallopFinder.

pixel chains using a circle fitting routine. The pixel RGB values are converted to HSV values and treated as a three-dimensional distribution. Next, the substrate is used as a reference to detect objects embedded in it that are spectrally distinct from the benthic substrates ambience. This is carried out by computing the Mahalanobis distance of each pixel HSV value from the mean of this distribution. A multiplicative filter on the distances is applied to bias them towards higher red hue and saturation. Thus, pixels that are distinct from the dominant background, and that are redder have a higher value in a reconstituted image. The hypothesized circles from the edge detection are then sampled both on the inside and the outside with respect to the reconstituted image pixels to obtain the aggregate brightness (red-filtered Mahalanobis distance) and the disparity between its interior and exterior samples. Circles with brightness and disparity above the respective median values are chosen as candidate detections. This narrows down the objects on the benthic substrate to scallops and sand dollars.

4.3. ScallopTK

The Scalable Adaptive-Localization and Laplacian Object Proposal Toolkit (ScallopTK) Detector [11] is a module which is useful as a general object detector for detecting any objects which are either blob or ellipse-like. It was created primarily to detect shellfish and address the challenges with detecting them, such as differentiating between distractor categories (e.g. rocks, sand dollars), live organisms, and dead organisms. For each input image, the system generates many initial candidate regions of potential shellfish, and then classifies each region using a combination of AdaBoost pre-classifiers and a convolutional neural network (CNN) applied on top of sized-normalized image chips extracted around each candidate. The optional AdaBoost pre-classifiers are applied to manually-created fea-
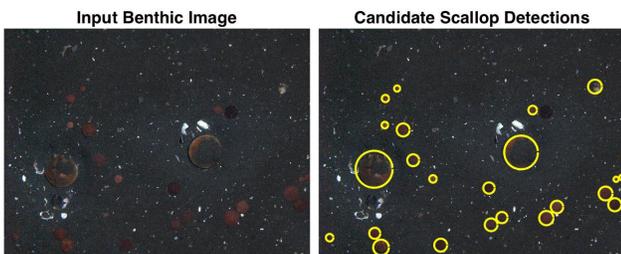
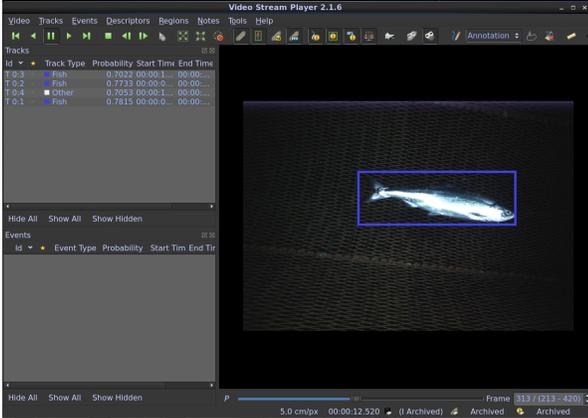Figure 10. Video viewer example. The left panel shows an entry for every spatiotemporal object track, and its corresponding category probability.
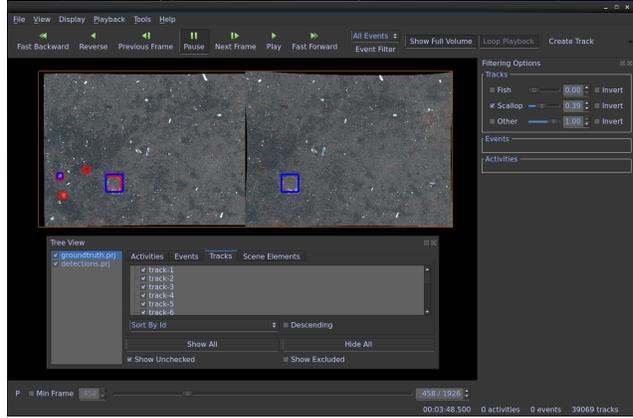

Figure 11. Image viewer example. The right hand bar has a filter for selecting which categories to show, in addition to an optional threshold on probabilities. Both computed (blue) and groundtruth (red) detections are shown in this example.

tures, and used as a speed optimization for reducing the number candidates evaluated by the CNN to a reasonable number. This full pipeline is shown in Figure 8.

4.4. FishRuler

The FishRuler plugin attempts to detect fish in video in order to estimate population abundance and size distributions for particular species of fish. It detects fish via a gaussian mixture model (GMM) segmentation, followed by assorted heuristics to filter detections based on whether or not they are likely to contain just one or multiple fish. Histogram of oriented gradient classifiers [10] are used on oriented extracted image chips around each detection to classify fish species. Any detection likely to contain multiple fish in the same connected-component blob is not used for the final size estimation step. The module itself is currently broken into two processes. A detector process which outputs detections from the GMM, and a classifier process which accepts detections alongside an input image, and produces species classification scores.

4.5. General Object Detectors

Faster R-CNN [19] was added to the platform as one of the initial object detector examples due to its generality. Unlike the hand-made object proposal generators of BenthosDetect, ScallopFinder, and ScallopTK, Faster-RCNN attempts to learn an object proposal detector using a custom region proposal network CNN architecture. The same features generated by this network are also re-used for proposal classification. In addition to Faster-RCNN we also plan on adding other high-scoring general object detectors, such as YOLO [18] and SSD [16] to serve as baseline object detectors when encountering novel problems in the domain.

5. Command Line and GUI Interfaces

There are currently two graphical user interfaces (GUIs) in the platform. Both can display either object tracks or object detections and their associated probabilities, though one is geared more towards video and the other raw images. Examples of both are shown in Figures 10 and 11 respectively. In addition to displaying object detections, the GUIs can be used to manually generate groundtruth detections or tracks, for use with either model training or evaluation. Outside of the GUIs there are a number of core command line tools to assist with running pipelines and other functionality. "Processopedia" outputs all known processes in the system that can be configured, "PipelineRunner" assists with running pipelines and inputting any extra parameters, and "Plugin-Explorer" lists all known plugin modules.

6. Evaluation and Experimentation

Using manual ground truth annotations, we have started comparing different variants of integrated modules within the platform. There are two tools which accomplish this, one for ROC generation and one for generating fixed evaluation metrics (detection rate, false alarm rates, etc). Both of these tools can score either individual object detections, spatiotemporal object tracks, or spatiotemporal events against some groundtruth. Due to the multitude of different ways to score the same problem, the evaluation tools have evolved to contain a number of input parameters over time, such as the spatial and temporal overlap criteria of computed on groundtruthed tracks. Figure 12 shows an example ROC generated by these tools comparing several detector variants in the system for the same problem, and Table 2 shows an example fixed metric output.

```
Detection−Pd: 0.791209
Detection−FA: 213
Detection−PFA: 0.515738
Frame−NFAR: not computed
Track−Pd: 0.791209
Track−FA: 213
Computed−track−PFA: 0.515738
Track−NFAR: not computed
Avg track (cont., purity): 1.34, 1
Avg target (cont., purity): 1.47, 0.79
Track−frame−precision: 0.5

VIAME−Hash : "a2123cde"
```

Table 2. Example "score tracks" tool output. In this case, only single-frame object detections were scored on the same scenario as in Figure 12, therefore the track and detection Pd (Percent detection) statistics are equivalent.

7. Conclusion

In addition to including several state-of-the-art algorithms with the ability to run and compare them in operational pipelines, our platform contains multiple features which aid in the rapid integration of new algorithms into the framework. Future work will involve the addition of new algorithm types (such as habitat classification and additional object trackers), the integration of new algorithms, adding new types of GUIs to the system, and additional general system improvements. The ability to configure and change algorithm pipelines in a GUI will be a useful addition as well as a useful debugging tool. We also plan on adding a database, along with the ability to ingest a video and perform queries on arbitrary concepts, such as performing a search for all instances of a particular species that a predefined detection model doesn't already exist for (i.e. search by example). This could be accomplished, for example, by performing iterative query refinement on top of CNN descriptors generated around general object proposals.

The open-source, modular nature of this system facilitates the development of a versatile and dynamic platform capable of addressing current and future needs in automated image processing. Much of the existing code for automated image analysis and video analytics in the maritime domain is unique to a specific sensor, data type, or research question. We hope that the development of this architecture will lead to more complete and versatile pipelines, capable of taking the novice image analyst from imagery to data with minimal effort. We hope that the development of this system will catalyze a community of developers to build upon this work to develop and integrate new algorithms and capabilities to meet the ever-changing needs of the marine science, fisheries and wildlife communities.
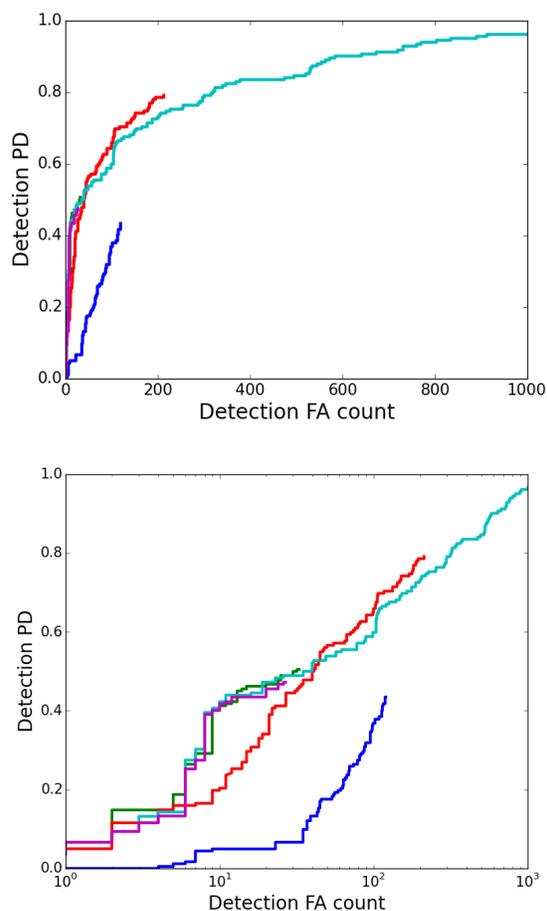

Figure 12. Example ROCs generated by the system comparing different detectors. In this case, a 500 image test sequence containing scallops (the targets of interest in this experiment) was used. Both ROCs show the same content, though the bottom image shows logarithmic scaling of the x-axis.

8. Acknowledgement

References

[1] Magnuson-Stevens Fishery Conservation and Management Act. *Public Law*, 94:265, 1996.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Good-

fellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[3] F. Bellard, M. Niedermayer, et al. FFmpeg. *Available from: http://ffmpeg.org*, 2012.

[4] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[5] M. Cappo, E. Harvey, and M. Shortis. Counting and measuring fish with baited video techniques-an overview. In *Australian Society for Fish Biology Workshop Proceedings*, volume 1, pages 101–114, 2006.

[6] M.-C. Chuang, J.-N. Hwang, F.-F. Kuo, M.-K. Shan, and K. Williams. Recognizing live fish species by hierarchical partial classification based on the exponential benefit. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 5232–5236. IEEE, 2014.

[7] M.-C. Chuang, J.-N. Hwang, and K. Williams. Supervised and unsupervised feature extraction methods for underwater fish species recognition. In *Computer Vision for Analysis of Underwater Imagery (CVAUI), 2014 ICPR Workshop on*, pages 33–40. IEEE, 2014.

[8] M.-C. Chuang, J.-N. Hwang, and K. Williams. A feature learning and object recognition framework for underwater fish images. *IEEE Transactions on Image Processing*, 25(4):1862–1872, 2016.

[9] M.-C. Chuang, J.-N. Hwang, K. Williams, and R. Towler. Multiple fish tracking via viterbi data association for low-frame-rate underwater camera systems. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 2400–2403. IEEE, 2013.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[11] M. Dawkins, C. Stewart, S. Gallager, and A. York. Automatic scallop detection in benthic environments. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 160–167. IEEE, 2013.

[12] R. Deriso, T. Quinn, J. Collie, R. Hilborn, C. Jones, B. Lindsay, A. Parma, S. Saila, L. Shapiro, S. Smith, et al. Improving fish stock assessments, 1998.

[13] S. M. Gallager, H. Singh, S. Tiwari, J. Howland, P. Rago, W. Overholtz, R. Taylor, and N. Vine. High resolution underwater imaging and image processing for identifying essential fish habitat. In *Report of the National Marine Fisheries Service Workshop on Underwater Video Analysis*, page 50, 2004.

[14] Z. Gu, R. Wang, J. Dai, H. Zheng, and B. Zheng. Automatic searching of fish from underwater images via shape matching. In *OCEANS 2016-Shanghai*, pages 1–4. IEEE, 2016.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. SSD: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015.

[17] K. Martin and B. Hoffman. *Mastering CMake*. Kitware, 2010.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[20] A. Salman, A. Jalal, F. Shafait, A. Mian, M. Shortis, J. Seager, and E. Harvey. Fish species classification in unconstrained underwater environments based on deep learning. *Limnology and Oceanography: Methods*, 14(9):570–585, 2016.

[21] F. Shafait, A. Mian, M. Shortis, B. Ghanem, P. F. Culverhouse, D. Edgington, D. Cline, M. Ravanbakhsh, J. Seager, and E. S. Harvey. Fish identification from videos captured in uncontrolled underwater environments. *ICES Journal of Marine Science: Journal du Conseil*, page fsw106, 2016.

[22] M. R. Shortis, M. Ravanbakskh, F. Shaifat, E. S. Harvey, A. Mian, J. W. Seager, P. F. Culverhouse, D. E. Cline, and D. R. Edgington. A review of techniques for the identification and measurement of fish in underwater stereo-video image sequences. In *SPIE Optical Metrology 2013*, pages 87910G–87910G. International Society for Optics and Photonics, 2013.

[23] W. Taymans, S. Baker, A. Wingo, R. S. Bultje, and S. Kost. GStreamer Application Development Manual. *Publicado en la Web*, 2013.

[24] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[25] K. Williams, R. Towler, and C. Wilson. Cam-trawl: a combination trawl and stereo-camera system. *Sea Technology*, 51(12):45–50, 2010.

[26] D. Zhang, G. Kopanas, C. Desai, S. Chai, and M. Piacentino. Unsupervised underwater fish detection fusing flow and objectiveness. In *2016 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 1–7. IEEE, 2016.